HANDOFF ENHANCEMENT IN MOBILE-IP ENVIRONMENT

by

WILLIAM WOO

B.Eng in Electronic and Communication Engineering, University of Birmingham, UK, 1991

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF ELECTRICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

November 1996

© William Woo, 1996

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical Engineering

The University of British Columbia
Vancouver, Canada

Date DEC 2, 1996

Abstract

Numerous efforts have been made in extending network connectivity to mobile computers using the Internet as a backbone. The Internet Engineering Task Force (IETF) has compiled a series of technical discussions into a basic IETF mobile-IP draft proposal. At the time of writing, the draft is in the process of becoming an Internet standard. In addition to this extension, there is a compatible route optimization scheme, Internet Mobile Host Protocol (IMHP), which can be employed to improve routing efficiency of the basic proposal.

In a Wireless and Mobile Data Network (WMDN), handoffs are required whenever a mobile host crosses cell boundaries. These handoffs introduce momentary disruptions and could significantly reduce the throughput at the transport layer. In this research, a new Handoff Enhanced scheme is introduced to further extend IMHP to improve both the routing efficiency and the transport layer performance during handoffs. This new scheme uses finite size buffers to store inbound datagrams for mobile hosts during handoffs. As a result, datagram losses are eliminated. This is found to improve the transport layer performance by a significant extent.

The transport layer performance of the three different schemes were evaluated using the OPNET simulation package. It is found that the IMHP yields very poor performance at high handoff rates. With the new enhanced scheme, the transport layer performance improves significantly. Besides, the new enhanced scheme employs route optimization and gives better performance than the basic IETF scheme.

Table of Contents

List of Tables

List of Figures

Acknowledgment

I would like to take this opportunity to thank Dr. Victor Leung for his continuous support and suggestions throughout this research. This work was supported by Motorola Wireless Data Group, Richmond, B.C., Canada and by the Natural Sciences and Engineering Research Council under Grant OGP0044286. Besides, I would express my appreciation towards my family and friends for their emotional support.

Chapter 1 Introduction

Computer networks provide a solution to share data and resources between different terminals within the same network. This is an efficient way to share information and resources. As more and more of these networks are present, there is a further need of connecting these independent networks together into an even bigger network infrastructure. This internetworking of networks is commonly referred to as the Internet. The Internet started out as a project in the military service in the United States of America and has been in place for over two decades. New networks are attached to the Internet at a fast pace. Together, the network provides a vast resource of information.

With the improvement in digital electronics technology, computers have been reduced in size. This makes a huge driving force for the market of mobile computing. There is a growing need for good computing power, mobility and network connectivity. The Internet is a prime candidate in providing network connectivity, since it is available globally. Current Internet implementation does not support host mobility and proposals have been made to address this issue.

1.1 Overview of Mobility Support over Internet Protocol

As there is no central organization governing the Internet, it is almost self-regulatory with a few regulatory bodies which strive to improve and implement additional capabilities of Internet. Internet Engineering Task Force (IETF) is an example of one of the boards. The responsibility of IETF is to investigate new additional features. The addition and modification of protocols will undergo discussion and be analyzed before they are actually implemented.

Data units within the Internet layer is a datagram and has a standard format. By having a standard protocol, the *Internet Protocol* (IP), computer applications on different platforms can share information and access computers on other parts of the Internet. Each computer connected to the Internet has a unique Internet address, which is a four byte integer. Internet addresses on the same network have the same network prefix which is part of the Internet address. Routing is based on this prefix. Under current Internet routing scheme, datagrams are routed to the same network if the network prefixes of the Internet addresses are identical. Therefore, when a host has physically changed its attachment point in the Internet to a different network, it cannot retain its current Internet address. This presents a hindrance to mobile network computing where mobile computers may be attached to different networks at different times. As a result, there are numerous solutions suggested which have attempted to extend this functionality. A common scenario for mobile computing requirement is shown in Figure 1.1. The presence of a virtual connection, between A.1

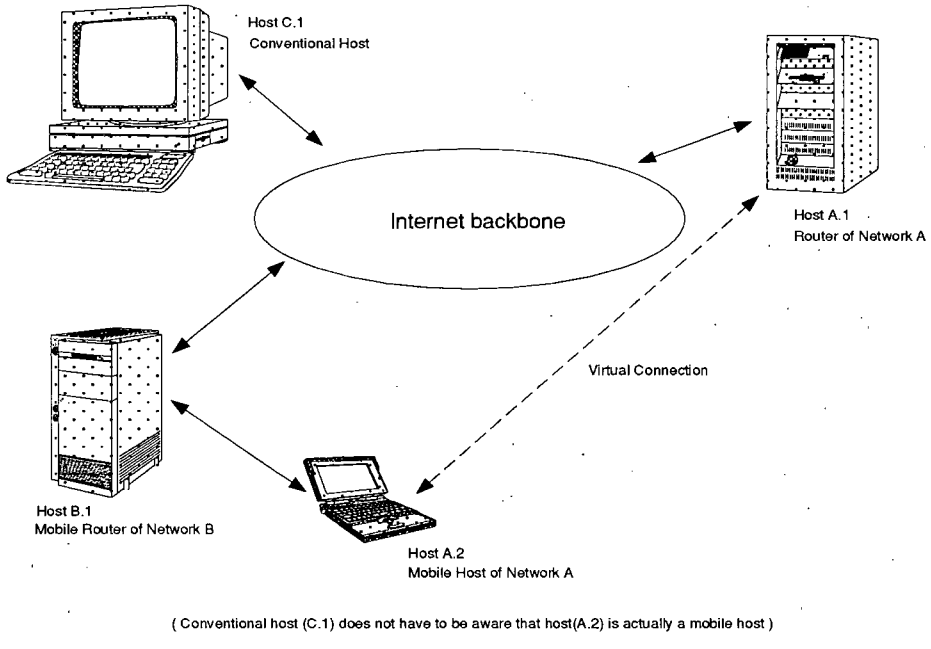


Figure 1.1 Mobile-IP allowing hosts to be mobile and reachable simultaneously

and A.2, allows the conventional host (C.1) to communicate with mobile host (A.2) via the home network router (A.1). This transparency enables any conventional hosts to be able to reach the mobile host irrespective of its current physical location.

IETF has compiled a series of technical discussions regarding mobility support over the existing Internet, and a draft [1] has been proposed. In this draft, a basic extension of additional protocols over IP is listed which provides basic mobility support. No assumptions upon the types of underlying mobile network has been made. This provides a general extension to be adopted for future mobility support.

Under this mobility extension, datagrams are delivered to the mobile host via its home network to its current location. The conventional host which is communicating with the mobile host could be physically very close to the mobile host. Under the basic IETF scheme, the datagrams are still routed to the home network first and then dispatched for the mobile host. There is a proposal [2] for optimizing the routing within the mobility extension. This proposal is compatible with existing IETF scheme. The basic idea of this scheme is to enable the conventional host to send datagrams directly to the mobile host's current residing network. It is not mandatory for hosts to support this scheme, therefore an extensive operating system upgrade is not required and is also backward compatible with the basic IETF scheme. If supported, this scheme is expected to deliver better performance over the basic scheme.

Both of the schemes have not placed any limitations on the degree of mobility supported. For a wireless mobile host which moves quickly from one network to the other, a handoff procedure has to be performed before the new foreign network provides network services to this host. It is shown in [3] that the nomadic behaviour has great impact on the transport layer perfor-

mance. This is a heavy performance penalty as most of the Internet applications are built around the transport layer protocols. Applications such as electronic mail (e-mail), file transfer protocol (FTP) and world wide web (WWW) all rely on the Transport Control Protocol (TCP) [4] for end-to-end data transportation. If a high performance penalty has to be paid, all these applications would be rendered useless over a highly mobile environment.

1.2 Motivation of Handoff Optimization and Scope of this Thesis

The basic IETF scheme has provided a basic requirement for supporting mobility. The proposal in [2] addresses its shortcomings regarding routing efficiency. However, under both schemes, there is no provision regarding the impact on transport layer efficiency due to a highly nomadic host behaviour. During the handoff phase of mobile connection, datagrams are bound to be routed incorrectly under these current proposals. For the mobility extension to be worthwhile implementing, it is necessary to improve on these proposals. It is of great interest to improve performance in a handoff intensive environment. It is the main emphasis of this research to design a new scheme to address both the routing and the handoff issues, thus giving better end-to-end performance at the transport layer.

This thesis gives performance analysis of the listed schemes and specifically monitors the performance during handoff periods. In the research, a new scheme is devised in order to enhance the performance during handoff. The end-to-end performance of each of the schemes is compared. Chapter two gives an overview of the basic IETF mobile-IP scheme and the route optimization extension. Chapter three lists the performance issues related in particular to handoff situation. Chapter four describes the new proposed handoff enhanced scheme. Chapter five outlines the simulation tool (OPNET), and the models for evaluating the protocols in [1], [2] and

the new handoff enhanced scheme. Chapter six shows the results of various simulations and discussions of the results. Chapter seven summarizes the analysis and concludes the findings in the thesis.

Chapter 2 Mobility Extension for Internet Protocol

The Internet provides a global network that is accessible by any computers within any network connected to it. Unlike the OSI [5] seven layer model for computer network architecture, the Internet architecture is based on a relatively simple protocol stack, with a four-layer protocol suite, commonly referred to as the TCP/IP, as shown in Figure 2.1.

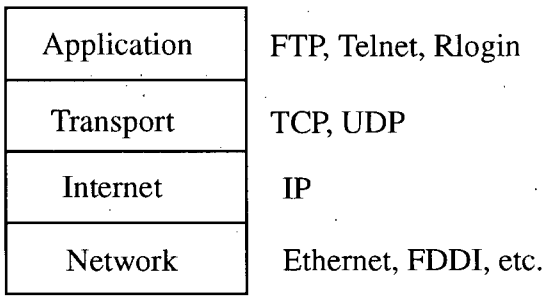


Figure 2.1 Protocol architecture of the Internet

The network layer handles all network oriented and physical layer issues particular to individual networks. The Internet layer is responsible for the delivery and routing of standard data units (Internet datagrams) between networks. Datagrams destined for a different network are relayed by one or more routers until they have reached their destination. Internet datagrams have a common header structure and are understood by routers throughout the Internet. The protocol for this layer is known as the Internet Protocol (IP). Conventional IP routers analyze the header portion of incoming datagrams to make routing decisions, as shown in Figure 2.2. Suppose an application within *host A* has established a connection with another application at *host B*. The datagrams are relayed by two routers, *router X* and *Y*. Whenever a datagram arrives at a router and is pending for delivery, the router will read the destination address within the header portion. From the destination address, the router can determine to which network to send the datagram. If,

however, *host B* has physically moved to another location within the Internet, existing routers cannot route datagrams to host B at its new location, since its IP address is still the same. Under the current Internet routing scheme, an IP address specifies a unique location within the Internet.

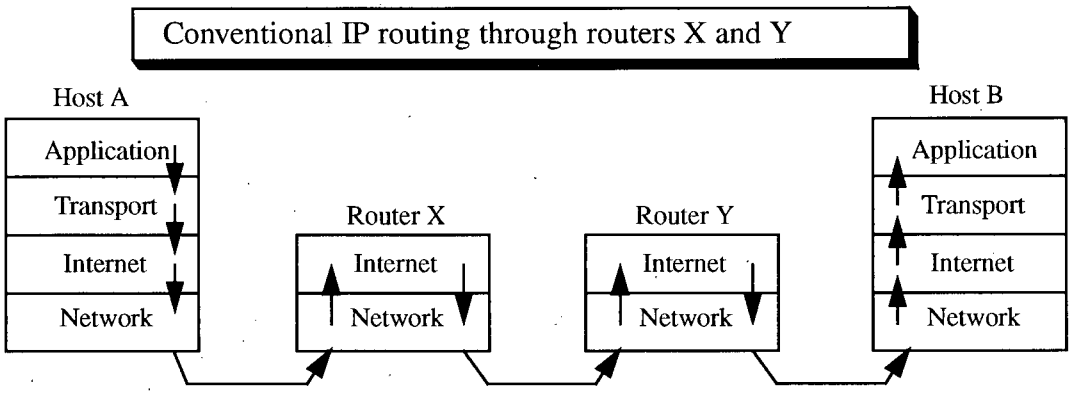


Figure 2.2 Conventional routing in Internet

Proposals have been made to support host mobility. One of the requirements is to implement this host mobility using existing Internet routing mechanism.

Under current Internet implementation, there are two ways to deliver datagrams to a mobile host: loose source routing (LSR) [6] and [7] in IP and encapsulation [1], [8]-[11]. The idea of LSR is to record the path of the datagram and tag the path information at the end of the datagram itself. With this option, both parties, which are involved in the communication, know the exact location of each other. On the other hand, the idea of encapsulation is to carry a datagram within another datagram. The original datagram is carried piggyback by an outer datagram.

It has been found [12] that some IP implementations, such as 4.3 BSD and SUN OS 4.1, do not process the LSR option correctly. This leads to encapsulation being a more feasible

solution for implementing host mobility within today's Internet without large scale operating system software changes.

The IETF has come up with a proposal [1] for mobility extension to the current Internet Protocol. Within this new proposal, there are three new entities defined which do not exist within current Internet Protocol. They are mobile host, home agent, foreign agent and various types of mobility bindings.

As opposed to a conventional host, a mobile host (MH) can have different points of attachment within the Internet at different times. Any host, regardless of the degree of mobility, which has that capability is classified as a mobile host. Therefore, a mobile host may be using an Ethernet interface at a foreign network or using a radio transceiver in a cellular radio network.

A home agent (HA) is a router at the home network of a mobile host. It is responsible for routing datagrams, destined for the particular mobile host, to its current location.

A foreign agent (FA) is an entity which is responsible for decapsulating tunnelled datagrams and providing routing services for mobile hosts. It also provides a care-of addresses for MHs so that the HAs know where to forward the datagrams for these MHs. However a MH can choose to obtain its care-of address by other means, thus acts as its own FA.

Mobility bindings is a cache entry for routing lookup. HAs, FAs and MHs are required to maintain its own list of bindings.

All of the above entities are new to the existing Internet Protocol. However, all the datagram traffic for these new entities make use of existing Internet routing mechanism. As a

result, all existing Internet routers can be used without modifications.

2.1 Overview of Mobile-IP Extension over Internet Protocol

MHs have fixed IP addresses and the same network masks as their home networks. Datagrams for these hosts are routed to a router within their home network [13]. The next step is to encapsulate those datagrams at their home networks and forward them to the foreign networks at which the mobile hosts are currently attached. This re-routing is accomplished by establishing mobility bindings at both the home and foreign networks. Incoming datagrams for a MH is first routed to its home network. A HA at the home network checks its cache list mobility bindings to see if the host in question has moved to another network or not. If so, it encapsulates the datagrams and sends them to their corresponding care-of addresses. This is illustrated in Figure 2.3. Besides, HA is required to handle all registration requests and keep mobility bindings for nodes that have moved to another networks.

Before a HA can forward datagrams to a mobile host at its current location, it has to know the care-of address for that mobile host. Each mobile host must obtain an appropriate care-of address reflecting its current location. Typically, foreign agents (FA) broadcast agent advertisements indicating the care-of addresses available for use. MH can then send registration request using one of those advertised addresses. Besides, a mobile host may obtain a care-of address by some other means such as Dynamic Host Configuration Protocol (DHCP) [14]. If the MH has its own care-of address, it may then function as its own foreign agent. The role of the host at the care-of address will be to decapsulate the tunnelled datagrams from the home agent and deliver them to their intended destination. Typically, a FA will have a cache list of bindings indicating the visiting mobile hosts' IP addresses and link-layer addresses.

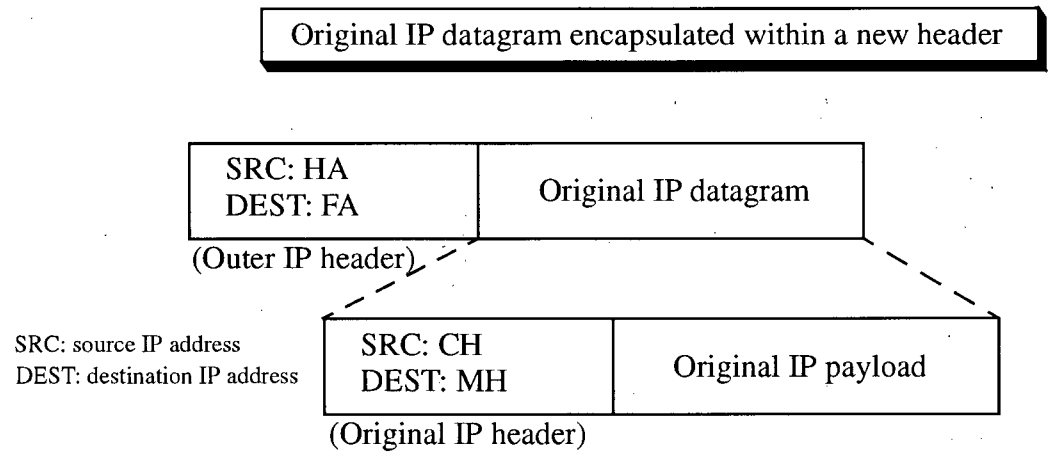


Figure 2.3 Datagram encapsulation at Home Agent

2.2 Registration Procedure for Mobile Hosts

Whenever a MH moves to a foreign network, or it is hopping from one foreign network to another, it has to send a registration request to the HA at its home network. This is mandatory as this is a request for a change in mobility binding and datagram routing. The HA has to authenticate the request to make sure that it is not a request from a malicious host. For each of those foreign networks, there is at least one FA beaconing the type of service available from that particular network. A beacon message is sent as an Internet Control Message Protocol (ICMP) [15] message with appropriate extensions. Upon reception of these beacon messages, the MH is able to decide whether it is possible to acquire network service from this FA. In the basic mobile-IP protocol proposal [1], the mobile registration requests are sent to well-known port 434 of User Datagram Protocol (UDP) [16]. Therefore, HAs and FAs have to listen to the UDP port 434 to retrieve any mobile registration requests. These registration messages are transported within standard UDP datagrams in the data portion. A standard mobile-IP registration message is shown in Figure 2.4. The mobile registration messages specification is listed in Table 2.1. In addition, there are extensions which can be tagged to registration messages in order to provide additional

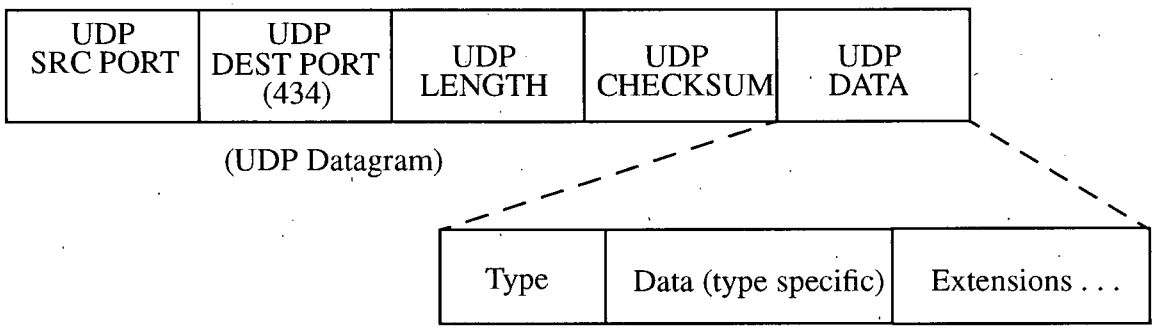


Figure 2.4 Mobile registration message format

information (e.g. mobile - home authentication). In each mobile registration message, there is a *type* field which specifies the type of message it represents. This enables the mobile-IP software to know how to interpret the message received and be able to take appropriate actions.

Table 2.1 Mobile-IP protocol specification

Mobile Registration Messages	Source	Destination	Type
Registration Request	Mobile Host	Home Agent	1
Registration Reply	Home Agent	Mobile Host	3

For security reasons, it is very important to verify the authenticity of the request. This will prevent some malicious hosts from re-routing the datagram path for a particular host by sending counterfeit registration request. Therefore, each registration request must be accompanied by a mobile-home authentication extension. A list of extensions is shown in Table 2.2. An authentica-

Table 2.2 List of mobile registration extensions

Mobile Registration Extensions	Type
Mobile-Home Authentication	32
Mobile-Foreign Authentication	33
Foreign-Home Authentication	34

tion extension is a secret known only between the two parties involved. From the authentication extension, the two parties can deduce the encryption algorithm and the key. The default authentication algorithm makes use of MD5 [17] with key sizes of 128 bits or more. To avoid replay attack, timestamps are mandatory with an option of nonce-based[1] replay protection. This requires a good random number generator. Eastlake *et al* [18] provides more information on pseudo-random number generators which is a core requirement in implementing security keys.

To acquire mobile network service from a foreign network, a MH will need to send a mobile registration request via a FA. After the FA has received this registration message, it determines the type of service requested. If the request is permitted, the FA will relay this registration message to the HA specified in one of the fields within the request. After the request has reached the specified HA, the HA either grants or denies the request. In either case, the HA will send a registration reply to the FA which has relayed the request. If the request is approved, the HA sets up or updates the mobility binding for this MH. Note that each MH can register with more than one FA, depending whether the HA permits this. Hence, each MH can have more than one mobility binding at its HA. Each of the mobility bindings has a specific lifetime. It is the responsibility of the MH to send another mobile registration before the existing binding expires.

Once the registration reply has reached the FA, it sets up or updates its cache binding list of visitors accordingly. The reply is finally delivered to the MH which has originated the request.

---

[1] Nonce is a patent assigned to IBM. Patent #5,148,479

2.3 Datagrams Delivery for Mobile Hosts.

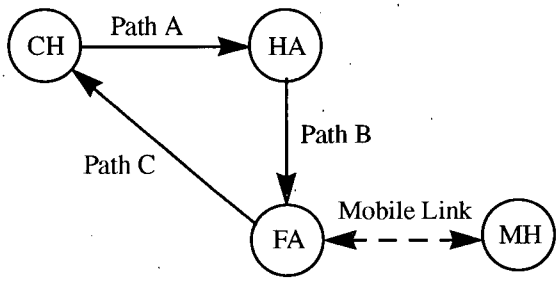


Figure 2.5 Datagram tunneling in mobile-IP

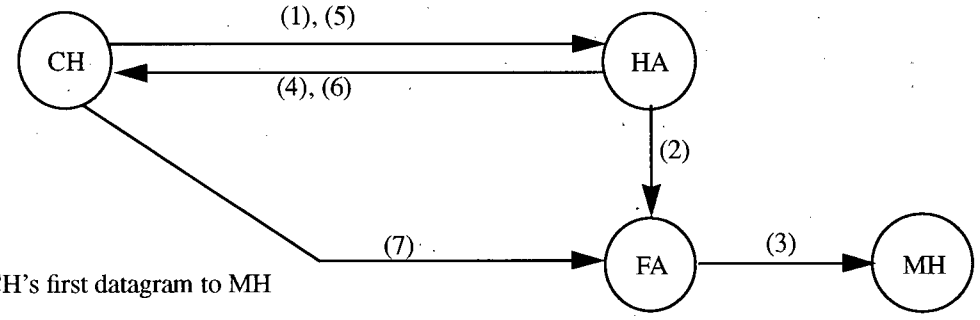
As soon as a registration for a MH is approved by its HA, datagrams from a corresponding host (CH) for this MH will be captured by the HA, encapsulated and forwarded to the FA by means of tunnelling. Datagrams for the MH arrives at the HA from the CH via Path A in Figure 2.5. An outer header is tagged to each of the original datagrams. This newly formed datagram will have the IP address of HA in the source field, while the destination will take the IP address of the FA or the care-of address of the MH. This datagram is then delivered to the FA through Path B. As the datagram arrives at the FA, it is decapsulated and the visitor binding list is consulted. If there is a valid visitor binding, the FA then sends this datagram via the corresponding mobile channel to the MH. On the other hand, datagrams originated from the MH are forwarded via the FA directly to the destination CH using Path C.

Datagrams from CH to MH clearly follows a suboptimal route. The routing paths A, B and C form a routing triangle and is commonly referred to as "triangular routing". The inefficiency of triangular routing is much more apparent whenever a MH is located at a network which is closer to its CH than to its own home network. There are different proposals which attempt to address this issue.

2.4 Route Optimization Extension in Mobile-IP

A route optimization method, incorporated in a superset of mobile-IP called the Internet Mobile Host Protocol (IMHP) [2], has been proposed. Route optimization is a backward compatible extension to the mobile-IP and requires a new type of entity, the cache agent (CA). The role of a CA is to keep a cache list of mobility bindings of hosts to which it has sent or routed datagrams to. Subsequently, the CA may tunnel datagrams directly to the care-of addresses of the MHs. As the name implies, the cache bindings that are stored at the CAs have limited lifetime. This is to avoid sending datagrams to an incorrect location, e.g. after a MH has moved to another foreign network. Generally, better performance can be achieved if a CH and its CA are co-located, to enable the delivery of datagrams to follow an optimal route right at the beginning.

The idea of route optimization is illustrated in Figure 2.6 where a CA is assumed to be co-located with CH. Originally the first datagram from the CH for MH reaches HA as indicated by the path (1). By this time, the HA realizes that the CH does not have up-to-date location information of MH. The HA tunnels datagrams to the MH as indicated in (2) and (3). In addition, the HA will also send a binding warning message to the CH, indicating that the CH does not have up-to-date location information. This is shown in message (4). If the CH supports this route optimization protocol, it can choose to be a cache agent (CA) for the MH. If the CH has the resources to function as a CA, it may send a binding update request to the HA, as in message (5). In response to that request, the HA sends location information of the MH, message (6), to the CA. The CA can update its cache location binding for the MH and subsequent datagram traffic travels directly to its care-of address, as in message (7). This will eliminate the routing inefficiency of sending via the HA.



1) CH's first datagram to MH
2) datagram tunnelled to FA
3) datagram decapsulated and delivered to MH
4) HA sends binding warning message to CH
5) CH sends binding request message to HA
6) HA sends binding update message to CH
7) Subsequent datagrams for MH sent to FA directly

Figure 2.6 Route optimization mechanism

If there is no CA along the path between CH and HA, the binding warning message would simply be discarded when it has reached the CH. If more datagrams are arriving at the HA from the CH for the MH, the HA would be able to deduce that the CH does not or chooses not to support route optimization. The HA should then choose to send binding warning messages to the CH at a lower rate or simply not to send them during subsequent datagram arrivals. This would avoid unnecessary traffic within the network.

After the MH has moved to another FA and has received registration approval from its own HA, the MH sends deregistration notification to its previous FA together with the current mobility binding. With this information, the previous FA can update the binding for this MH. The lifetime of this new binding is set at a value with the lifetime remaining in the original binding.

Chapter 3 Performance Issues Related to Handoffs

The basic IETF mobile-IP proposal places no restriction on the type of physical media that enables host mobility. Due to the much higher level of mobility supported by wireless and mobile data networks (WMDNs), these networks are prime candidates for application of mobile-IP and it is crucial that mobile-IP offers good performance in these network environments.

To allow a high user density in a WMDN, some frequency reuse scheme has to be incorporated, in a similar manner as the cellular telephone network. Each radio cell has a base station (BS) which provides connectivity between the WMDN and some wireline backbone. When a MH crosses over a cell boundary, a handoff operation is needed to transfer the MH's radio link from the old BS to the new BS.

While handoffs are transparent to the IP and mobile-IP layers in centralized WMDNs which employ dedicated backbone facilities and internetwork with the Internet via a single router, in this research we are concerned with a distributed WMDN environment where each cell or a cluster of cells constitutes a mobile subnet interconnecting directly to the Internet via a mobile-IP supporting router providing FA function to MHs roaming in its coverage area.

This scenario is applicable, e.g. in a campus-wide wireless local area network with attachment points at different departmental LANs, and may well provide a low cost solution to future wide area WMDNs as it eliminates the needs for dedicated backbones. In this scenario, mobile-IP is called upon to support handoffs, by creating new mobility bindings with the new FAs and terminating existing mobility bindings with the old FAs. A typical handoff scenario is shown in Figure 3.1.

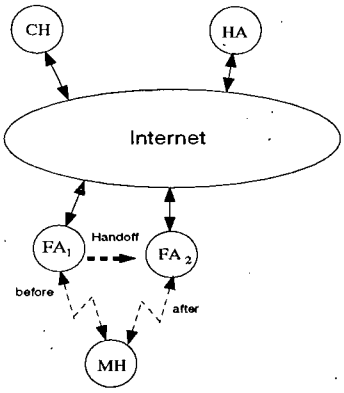


Figure 3.1 Typical mobile handoff scenario

3.1 Transport Control in TCP/IP

The design philosophy of TCP/IP has taken the approach that the Internet layer consists of an unreliable network. This allows a variety of different types of networks to be connected together. The responsibility of reliable connection is taken up by the transport layer. Transmission reliability is dependent on the transport layer protocol. TCP [4], one of the most important protocol in the transport layer of TCP/IP, is a reliable, connection-oriented protocol, which means the end-to-end connection is monitored for error recovery and to preserve data sequencing.

Data stream in TCP is composed of a sequence of bytes. Each of these sequence of data is called a segment. TCP expects an acknowledgment for each segment sent. There is a retransmission timer for monitoring the acknowledgment time. If the retransmission timer expires before an acknowledgment is received, TCP will assume the segment is lost and will retransmit the lost segment. *Round Trip Time* is the time taken between a segment sent and the corresponding acknowledgment received. To account for a variation in delay within the Internet, TCP uses an adaptive retransmission algorithm. In essence, TCP monitors the round trip time required for the acknowledgment to come through. Based on that information, TCP will set the value of the timer

for initiating the retransmission. Whenever it has received a new round trip time, TCP adjusts the retransmission timer value. TCP keeps an estimate of round trip time, *RTT*, and average RTT deviation, δ, as weighted averages and uses new round trip samples RTT' to adjust RTT and δ accordingly. To compute the new weighted averages, TCP uses a RTT gain factor, α, and a RTT deviation gain factor, λ, where $0 \le \alpha, \lambda < 1$, to weigh the old averages against the new ones based on the following algorithm:

$$Err = RTT' - RTT$$

$$(1-\alpha) \cdot RTT + \alpha \cdot RTT' \rightarrow RTT$$

$$\delta + \lambda(|Err| - \delta) \rightarrow \delta$$

New retransmission timeout = $RTT + \beta \cdot \delta$

Choosing a value of α close to 0 makes the weighted average adapt very slowly to the recent changes. If, however, α is chosen to be close to 1, the weighted average will be very sensitive to recent changes and disregards the long-term average RTT. Likewise, choosing a value of λ close to 0 tends to disregard recent RTT deviation. RTT deviation constant, β, is a scalar which specifies how long TCP layer will wait for the acknowledgment. If β is set to 1, TCP is overly eager to send retransmissions, which wastes network bandwidth. On the other hand, if β is set to too large a value, TCP will be waiting for too long before it realizes the packet is in fact lost. Again, the TCP throughput will suffer. It is therefore important to choose a value of β which is adequate but not excessive. The original specification recommends setting β to 2.

3.1.1 Karn's Algorithm and Timer Backoff

Consider the case where TCP has just sent a packet, and the acknowledgment has not been received before the retransmission timer expires. TCP will retransmit the packet, thinking the original packet has been lost. If the acknowledgment comes in soon after TCP has retransmitted the last packet, the new round trip time sample will be far too short and it is also ambiguous. To avoid this, TCP simply ignores the new RTT. This is known as Karn's algorithm [19]. The idea is to avoid this ambiguity altogether by only taking those round trip time samples from segments which have been transmitted only once. This would avoid the possibility of shortening the estimated round trip time erroneously. This is important as the estimate round trip time is used for determining the value for retransmission timer for the next transmission.

Besides not updating the round trip time estimate, Karn's algorithm requires TCP layer to implement a timer backoff strategy. An initial timeout value will be computed as before. If the transmission has not been successful, the timeout value will be increased by a multiplicative factor until an upper bound has been reached. Having an upper bound will keep the timeout value from being infinitely long. This timer backoff will increase the timeout value exponentially if the delay has been exceedingly long. This stabilizes the TCP and adapts the retransmission timeout value to a suitable value.

3.1.2 Response of TCP in congestion

TCP layer does not have *a priori* knowledge regarding the amount of delay that each data segment will undergo. Only after receiving the acknowledgment can it determine the exact round trip delay. During congestion in Internet, the segments will suffer from severe delays causing the retransmission timer to timeout. To the TCP layer, this timeout will trigger another retransmission

which will certainly choke the already congested network. This will cause a congestion collapse. To avoid that, TCP reduces the amount of traffic through the network. Different versions of TCP, together with several proposals, have methods to accomplish this: *slow start* and *multiplicative decrease.*

Multiplicative Decrease Congestion Avoidance is the reduction of the congestion window by half. This process is repeatedly done until the window size reaches one. For those segments in the reduced window size, the retransmission timer will be increased exponentially.

Slow-Start Recovery specifies that at the start of transmission on a new connection or increasing traffic volume after congestion, the congestion window should be set to one segment. If acknowledgment has arrived before timeout, the window will be increased one at a time. It also limits a linear growth of the congestion window, while *Multiplicative Decrease Congestion Avoidance* sets exponential decrease in traffic sent. These will help to alleviate the congestion problem.

There are efforts made [20]-[27] in establishing congestion control over TCP. Each of these methods introduce adaptive scheme for adjusting the retransmission timeout at the TCP layer. These methods stabilize the congested network by throttling the TCP throughput.

3.2 Impact of Handoff in TCP/IP over Mobile-IP

The Internet Protocol layer is known as a best-effort delivery system. No guarantee is made upon the success of datagram delivery. Datagram losses is handled by the TCP layer which requests for retransmissions of lost datagrams. If retransmission is needed, there is a backoff algorithm in TCP to avoid choking the congested network with too many datagrams. Before the

introduction of wireless networking and host mobility, TCP only deals with datagram losses primarily due to congestion.

When a MH crosses cell boundaries within a WMDN, it has to perform a handoff procedure and send a new registration request to its HA before it can continue to receive network services from its HA. Momentary disruptions of communications are therefore inevitable during handoffs, under the basic IETF mobile-IP scheme. This has a detrimental effect to the performance of the transport layer employing TCP, since the datagram losses are interpreted as network congestion. During handoffs, datagrams cannot be tunneled to the new FA until the new binding is authenticated at the HA. The authentication can take a long time if the MH's home and visited networks are separated by a large transmission delay. In the case of IMHP, the delay may be worsened, since it is necessary to update the CAs with the new care-of address.

As the retransmission is handled by the TCP layer, substantial datagram losses during handoffs would trigger the slow start procedure previously described. This could result in significant reduction in TCP throughput which takes a long time to recover. For a roaming MH, the rate of handoff could significantly affect the system performance experienced by the user.

Cáceres *et al* [28] has proposed a method for improving TCP performance during handoff. However, it requires modifications within the TCP implementation, which means that a vast majority of the Internet hosts today will not be able to make use of this improvement.

3.3 Handoff in Basic Mobile-IP Scheme

Each handoff gives rise to a vulnerable period in which datagrams in transit to the MH are potentially lost. This is illustrated by the timing diagram in Figure 3.2, representing the typical

handoff scenario in Figure 3.1.

Message A is a datagram sent by the CH to the MH. The routing of the Internet will deliver the datagram to the home network of the MH. At the home network, the datagram will be routed to the MH's home agent, HA. As shown in Figure 3.2, the datagram will be tunnelled to foreign agent, FA_1, which is currently registered with HA. FA_1 will decapsulate the tunnelled datagram and forward it to the MH. Message B is the datagram sent by the MH back to the CH. Since FA_1 is the MH's foreign agent, outbound datagrams from the MH will be first sent to FA_1 and then routed to the CH with existing Internet routing. In this case, no tunnelling or encapsulation is required. It is shown that message A has been routed in a suboptimal path, whereas message B has followed an optimal path. This is commonly referred to as "triangular routing" and is shown in Figure 2.5.

The MH has then moved to FA_2 and registration request has been sent. Before the registration has been approved, message C arrives at FA_1. FA_1 still thinks the MH is within its mobile region. FA_1 attempts to deliver the message to the MH via its mobile channel. Message C cannot reach the MH. During time period, D_{HF1}, in Figure 3.2, the messages that arrive at the HA are routed incorrectly to FA_1 and are lost. Besides, during time period, D_{F2H}, messages arriving at HA cannot be delivered to the FA_2 yet, since HA has not received a registration request yet. Therefore, during every handoff, there is a period in which datagram delivery is susceptible to losses. After handoff registration has been completed, datagram delivery for MH can resume. Subsequent messages for the MH will be routed via FA_2 to the MH.

It is shown that there is a period in which datagrams for the MH are prone to losses. The

length of this vulnerable period is directly related to the packet delays between both the old and new FAs and the HA of the MH. The vulnerable period, T_{VUL}, consists of the delay for datagrams sent prior to the handoff to reach FA_1 via HA, D_{HF1}, and for the new registration request to reach HA via FA_2, D_{F2H}, During T_{VUL}, datagrams will be incorrectly routed. Once the new registration is authenticated at HA, a new tunnel could be established to redirect datagrams to MH via FA_2. Delays between MH and FA_1 or FA_2 are assumed negligible.

$$T_{VUL} = D_{HF1} + D_{F2H} \tag{3.1}$$

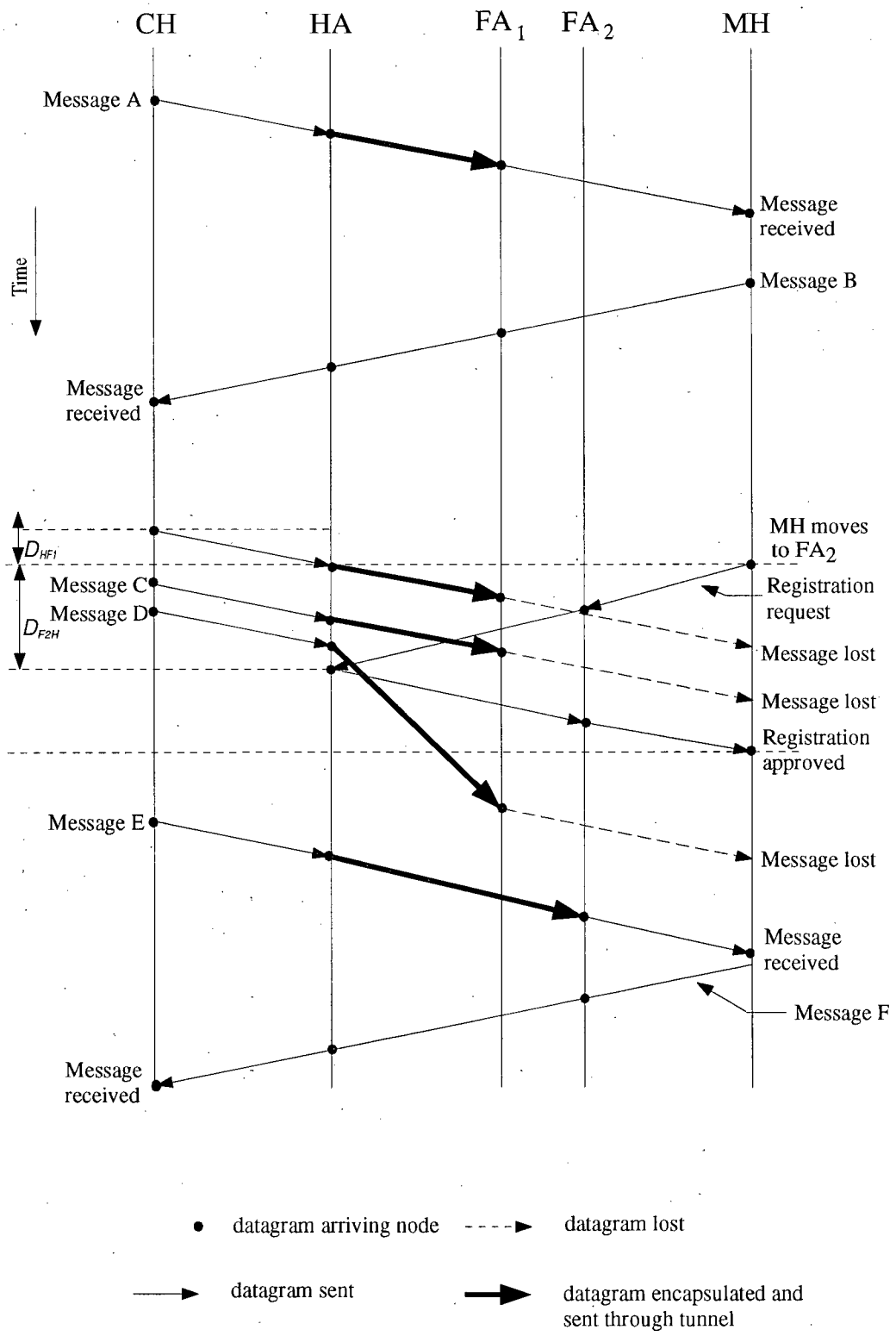


Figure 3.2 Timing diagram during handoff in basic mobile-IP scheme

3.4 Handoff in Route Optimization Scheme

In chapter two, the route optimization scheme, IMHP, is expected to deliver better performance over the basic IETF scheme, since it has eliminated triangular routing. However, during a handoff, the CA sends datagrams to the previous FA. Hence, these datagrams are routed incorrectly. It is shown in previous sections that datagram losses in the IP layer can impose a serious performance penalty over the TCP layer. The performance of the IMHP scheme degrades as the rate of handoff increases.

Whenever there is a handoff, a MH will perform registration with its HA as in the case of the basic IETF scheme. The CH therefore has outdated information regarding the MH's care-of address. All subsequent datagrams are sent to the previous FA. As a result, the IMHP scheme suffers from performance degradation during or immediately after handoffs.

Timing diagram for the IMHP scheme is shown in Figure 3.3. It is assumed that CH and CA are collocated. The worst case vulnerable period include the delay between the corresponding host, CH and FA_1, before the handoff, D_{CF1}, and the delay to update the new binding at either CH or FA_1 after the handoff, i.e.,

$$T_{VUL} = D_{CF1} + D_{F2H} + min(D_{HF2} + D_{F21}, 3D_{HC}) \tag{3.2}$$

D_{F2H} is the delay to register the new binding with HA after handoff. D_{HF2} is the delay for the HA to reply with a registration confirmation. D_{F21} is the delay for FA_2 to give FA_1 the new binding. $3D_{HC}$ is the delay for the HA to send a binding update invitation to the CH, a binding request from the CH to the HA, and the binding reply from the HA to the CH. Depending on the

current congestion over the Internet, the vulnerable period with route optimization could easily exceed the vulnerable period for the case without route optimization. As before, TCP performance suffers under a high rate of handoffs or if the delays are long. Methods to minimize loss of datagrams during mobile handoffs are therefore necessary to optimize the performance of IMHP.

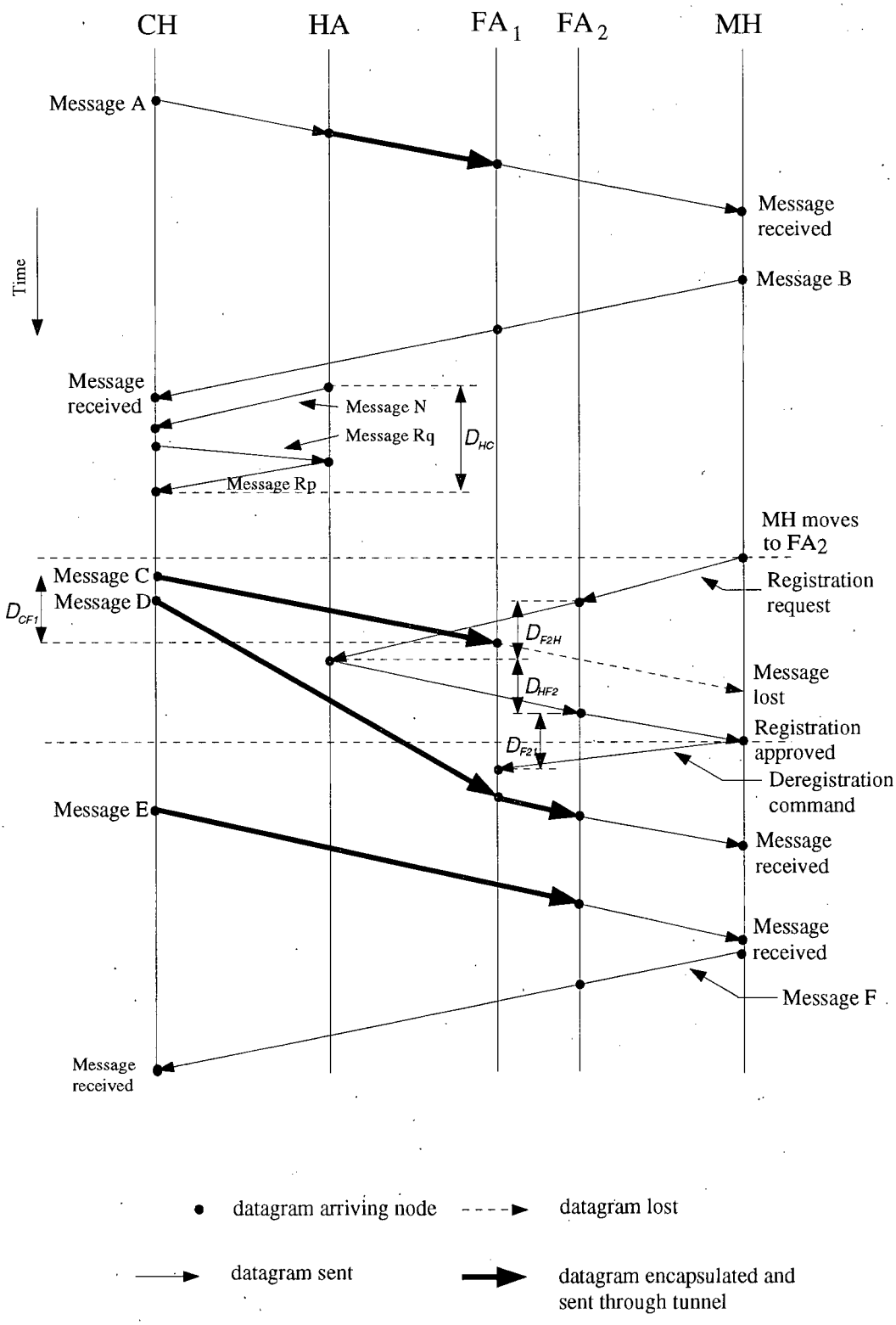


Figure 3.3 Timing diagram during handoff in IMHP scheme

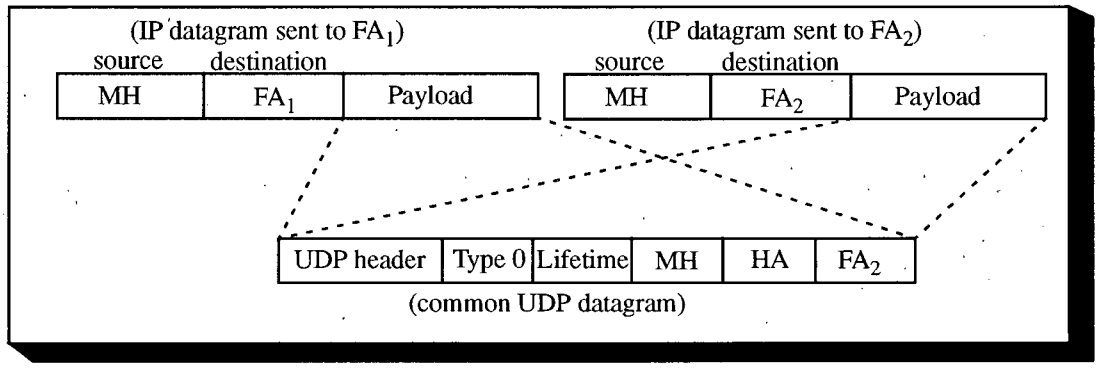
Chapter 4 Handoff Enhanced Mobile-IP Scheme

In sections 2.3 and 3.2, it is shown that performance is suboptimal due to either triangular routing or datagram loss during handoff. In this research, one of the objectives is to devise a method which improves the routing efficiency during handoff. The idea of this proposal is to eliminate or minimize the vulnerable period, while keeping the route optimization extension as in IMHP. By doing so, datagram loss will be minimized and hence the impact on Transport Layer is kept to a minimum. While improving routing efficiency, the security robustness of the new scheme should not be compromised.

4.1 Enhancement of Mobile-IP during Handoffs

The main emphasis of the scheme is to minimize datagram losses during handoffs and, at the same time, to yield better performance than the basic IETF scheme. This scheme uses a buffering technique at the previous FA which stores the inbound datagrams for the MH during the handoff period. Upon the reception of the registration confirmation at the previous FA, the stored datagrams are released to the MH. Each store buffer has its own lifetime. If an authenticated reply is not received within the time allowed, the contents of the buffer is flushed and the handoff is considered not successful.

Route optimization is performed as in section 2.4. However, during handoffs, there are additional procedures to be done. For example, the MH has switched from one foreign agent to another, say FA_1 to FA_2. During the handoff, the MH needs to send a new registration request via FA_2 to its own home agent HA. In this scheme, the MH sends two copies of the registration request. One is sent to the HA via FA_2, while the other one is sent via FA_1. This is illustrated in



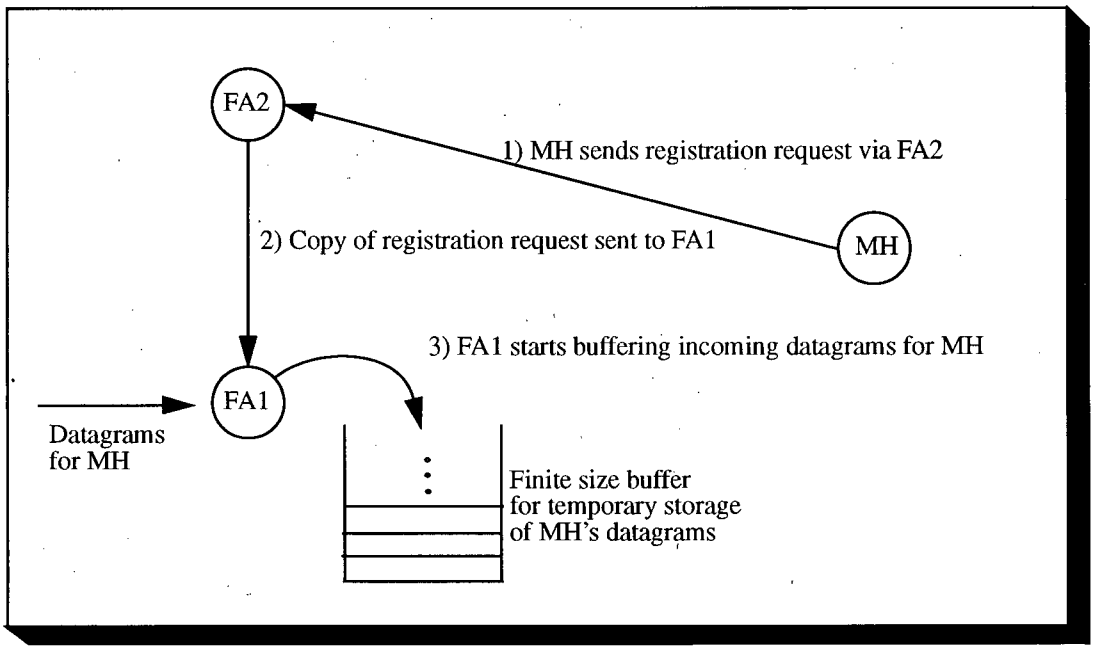
Registration messages sent to FA_1 and FA_2 during handoff
Note that the registration message is common in both IP datagrams

Figure 4.1 Registration messages sent during handoff

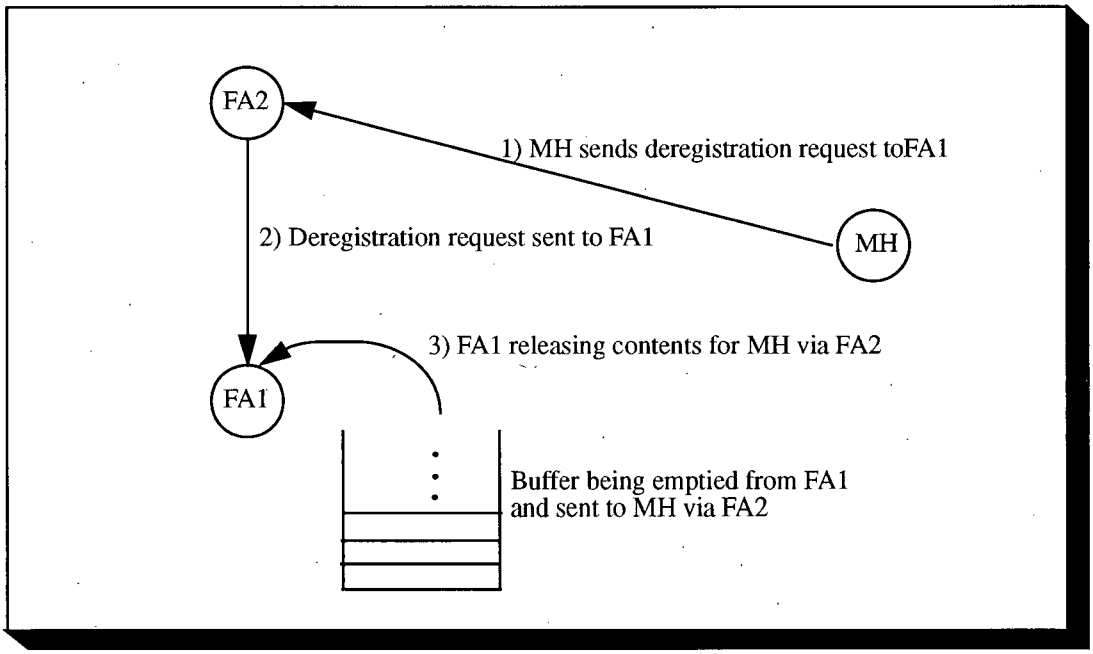
Figure 4.1 and the format of this type of message conforms to the basic IETF proposal [1]. Note that the registration request contains both the MH's address and the new FA_2's address. Upon receipt of this registration message, FA_1 can deduce that the MH is currently at the network of FA_2 and is pending for new registration approval. A finite size buffer is set up at FA_1 to store the incoming datagrams for the MH. At this point, the FA_1 still sends those datagram via its own mobile channel at which the MH has been registered with, since the new registration is still pending for approval. The size of this buffer, at FA_1, is set to a specific size which is determined by FA_1according to the resources available on that site. Once the registration has been completed, the MH will cancel the binding with FA_1 by sending a deregistration to FA_1 (registration request with lifetime set to 0). As soon as FA_1 has received this de-registration, it releases the contents of the buffer to the MH via FA_2. This is illustrated in Figure 4.2.

4.2 Security Considerations

One of the considerations of this new scheme is the security robustness compared with the



MH performing handoff from FA1 to FA2



MH performing deregistration with FA1 after successful registration with HA

Figure 4.2 Illustration of the Handoff Enhanced Scheme

existing Internet. Special care has been taken in order to avoid security loopholes. In this scheme, the incoming datagrams are stored in a buffer during handoff. This avoids the possibility of replay

attack in a WMDN environment. This is the reason that the datagrams are not forwarded to the new FA as soon as the previous FA knows about the handoff. In this case, even if a malicious host sends a forged registration to the FA (FA_1), the inbound datagrams for the MH will not be re-directed to the malicious host. FA_1 waits for an authenticated registration reply before sending the stored datagrams.

This scheme does not open up any additional opportunities for security attacks. Although wiretapping of datagrams (in WMDN) is still possible as in the case of the basic IETF scheme, it is considered, as in [1], that the scheme does not allow more security attacks than the existing Internet. As in the case of a wired Ethernet within a network, all local traffic are sent to the Ethernet interface, therefore wiretapping attack is also possible in the existing Internet.

4.3 Performance Considerations

Provided that the previous and current FAs are close together, the mobile registration request arrives at the previous FA in a short time. Hence, the previous FA can start buffering the incoming datagrams for the MH. Therefore, the enhanced scheme does not suffer from a vulnerable period. However, there are two constraints that limits the effectiveness of this scheme. Firstly, the buffer temporarily storing datagrams for the MH has a finite size. For a IP throughput of 1 k byte/s and the registration delay is 1 sec., the buffer size required for each TCP connection is 1 k byte. If the IP throughput is more higher than that, buffer overflow can occur if the registration takes too long to complete. In this case, storing some datagrams might not be useful to the TCP layer, since it has to request retransmissions of the lost datagrams eventually. Besides memory constraints, if the delay between the registration and reply is long, the retransmission timer in TCP may time out before the buffered datagrams could be delivered and acknowledged by MH.

Nominally, the retransmission timer of TCP is set at some multiples k of the round trip end-to-end delay between the CH and the MH, i.e. $2D_{CM}$. Therefore the proposed handoff optimization scheme offers transport performance enhancements only if the condition in equation (4.1) is met. The timing diagram for the enhanced scheme is shown in Figure 4.3.

$$T_{DEL} < 2(k-1)D_{CM} \tag{4.1}$$

However, the worst case scenario will only bring the performance down to the same as IMHP, but not worse. Note that it is possible to further enhance this scheme. Provided that the old and the current FAs are closely related and entrust each other, it is possible to start forwarding datagrams to the new care-of address location as soon as the old FA is aware of the fact that the MH has handed off to the new FA. This will require some form of security association to exist between the two FAs. The handoff registration message has to include this secret authentication as an extension. The proposed extension takes the format as described in section 2.2. The extended authentication is listed in the Table 4.1. However, as listed in [1], FAs are not expected to initiate any messages regarding mobile registration and reply. Having the FAs to negotiate the security association and redirect the routing is clearly violating the recommendations. Therefore, the proposed Handoff Enhanced Scheme adopts the method of storing the datagrams first as to comply with the security requirement laid down by [1]. This new scheme is expected to deliver better performance during handoff in a WMDN.

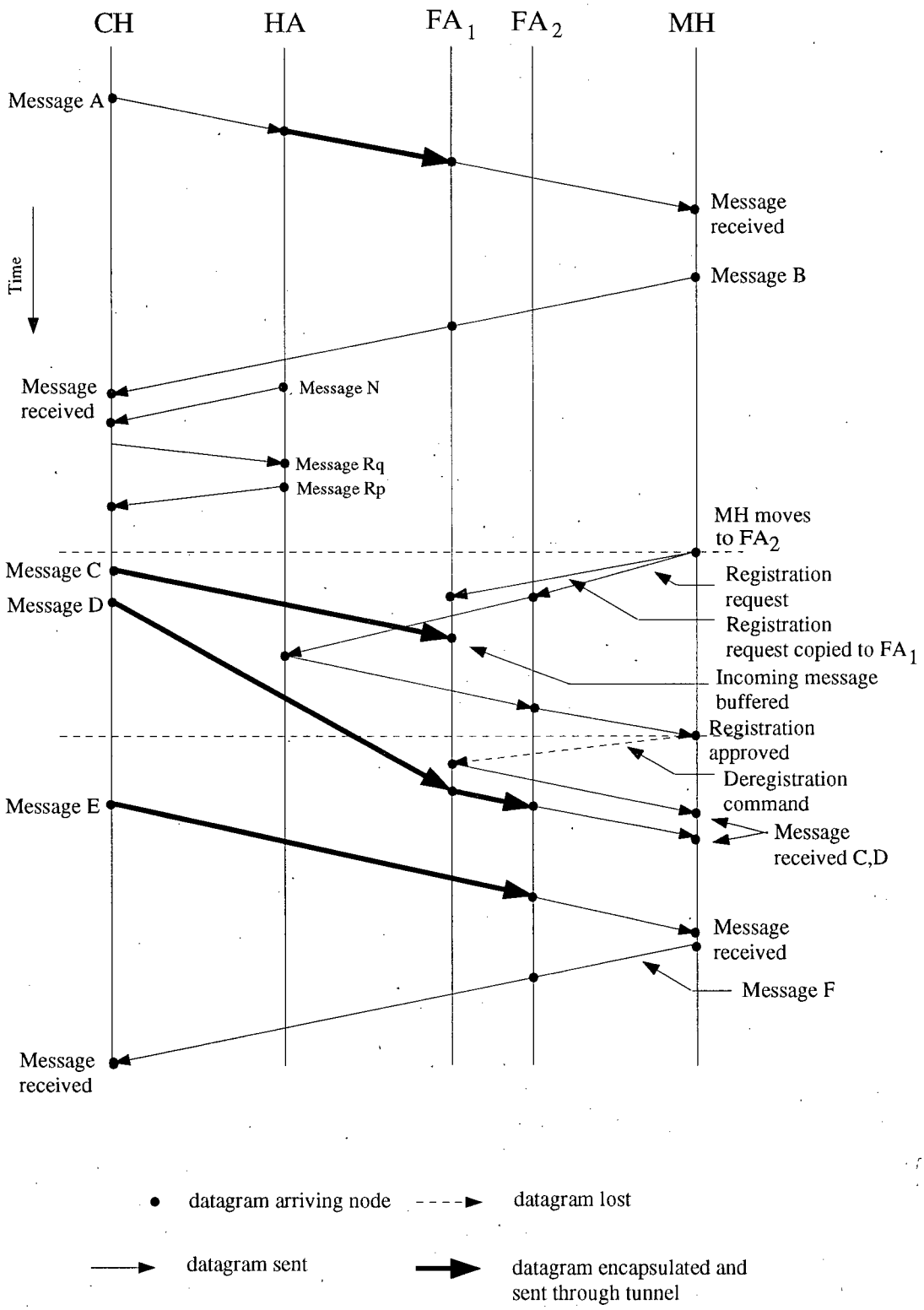


Figure 4.3 Timing diagram during handoff in the Handoff Enhanced mobile-IP scheme

Table 4.1 Registration authentication extension between Foreign Agents

Mobile Registration Extensions	Type
Foreign-Foreign Authentication	35

Chapter 5 Design of Simulation Model

Another objective of this research is to evaluate the transport layer performance with the mobile-IP proposals and extensions. The network models of the various listed schemes are developed and the evaluation is performed in simulation. The network model should adhere to the protocol as closely as possible. Once a network model has been established, different statistics can be obtained and performance evaluation can be based on those statistics.

Mobile-IP, like conventional IP, is an unreliable, connection-less layer. In order to assess the performance of the scheme, it is necessary to include the Transport and Application layer protocols. With these layers built into a mobile host, the host behaves as a real Internet host. It can negotiate a connection, assess connection status, request for retransmissions, and terminate connections, etc. Although all these modules are not within the Internet layer, it is necessary to include them in order to quantify the network performance on a connection-oriented standpoint.

The Internet is a dynamic and complicated network. It contains different kinds of networks which are linked together using the common TCP/IP protocol. There are few restrictions on network topology and hence the Internet is very flexible. As a result, it has gained widespread acceptance. Defining an analytical model for its throughput statistics and variability becomes very difficult, as there is no standard connection pattern between the networks. Consequently, there is no analytical model for the current Internet based on TCP/IP. The evaluation in this study is based on simulation. The model is developed using OPNET[1]. Mobile-IP is based on the IP model in OPNET, but heavily modified to incorporate the extended capabilities.

---

[1] OPNET is a network simulator software developed by MIL 3, Inc.

5.1 Overview of OPNET Simulator

OPtimized Network Engineering Tools (OPNET) is a comprehensive engineering system capable of simulating large communications networks with detailed protocol modelling and performance analysis. OPNET features include: graphical specification of models; a dynamic, event-scheduled Simulation Kernel; integrated data analysis tools; and hierarchical, object-based modelling. OPNET's hierarchical modelling structure accommodates special problems such as distributed algorithm development. OPNET delivers open systems methodology and an advanced graphical user interface known as the MIL 3 User Interface.

Due to its object-oriented design, complicated models can be set up and prototyping is made easier and more manageable. Each OPNET simulation is presented in a hierarchical fashion, and therefore, simulations written will have strong parallels with actual communication networks. Debug options are built into every OPNET simulation so that accurate state tracking is possible. A fully graphical interface makes model prototyping much clearer and less error-prone.

OPNET has a lot of standard modules written and all of those modules can be modified to cater to different needs. It is also possible to set up an entirely new network model and fine tune its behavior using the C programming language.

5.1.1 Hierarchy within OPNET Simulator

Each OPNET model consists of elements which are located in three different hierarchical domains: *Network*, *Node* and *Process*. Figure 5.1 depicts an application in OPNET. Network domain is where the topology of the network infrastructure is defined. A network can be made up of any number of subnetworks and nodes. Different types of links can be used to connect

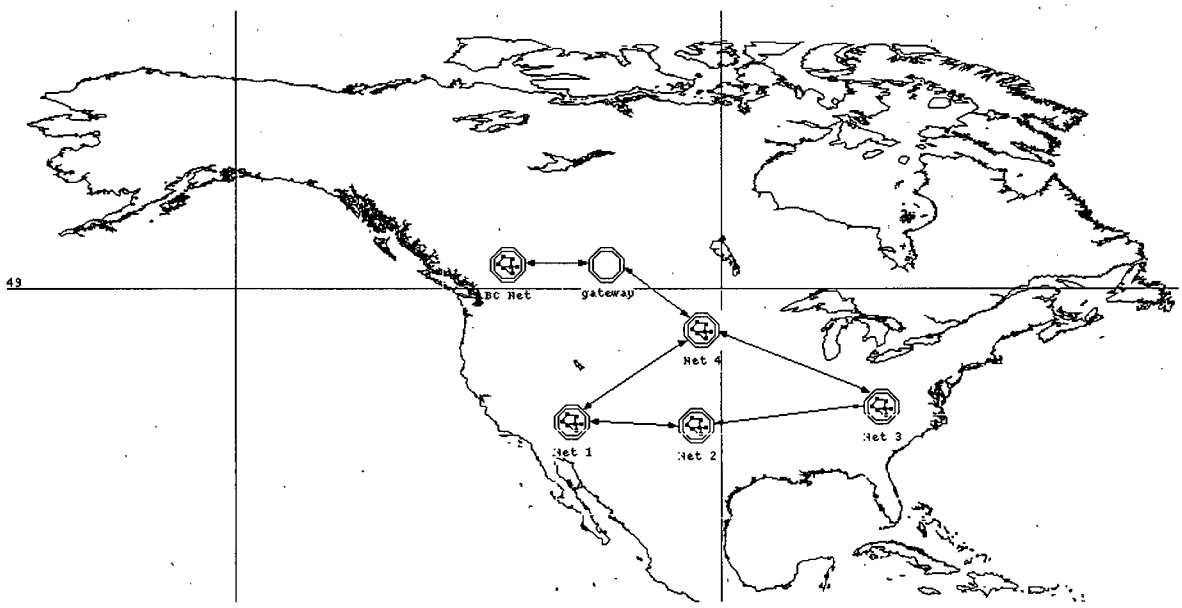


Figure 5.1 Hierarchy in OPNET with Networks, Nodes and Processes

networks and *nodes*.

Within a *network*, the smallest subdivision is a *node*. A *node* can perform different functions within the network; e.g. it can be a router or a workstation (e.g. *gateway* in Figure 5.1). The characteristics of each *node* model is defined by the *modules* within. There are different types of *modules*: processors, queues, generators, transmitters and receivers. Figure 5.2 showed an example of node layout of a conventional IP host over Ethernet. In the example, the host consists of different layers of protocols; each of these layers is responsible for different functions. The model is a close representation of real implementation of the protocol stack. Within the node, *app1* and *app2* belong to the Application layer of TCP/IP. TCP layer is realized by a single module. Within this module, it forks a new connection process for each connections. There are a few parameters which can be set in the generators, transmitters and receivers. However, the main core of functionality in each node is determined by the *processors* and *queues*, which are fully

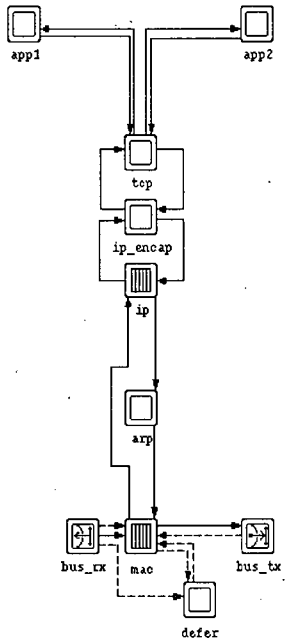


Figure 5.2 An example of node layout of an Ethernet workstation

programmable. Processors and queues are represented as finite state machines. It is a set of instructions which the node will perform under certain predefined conditions. Hence, a (parent) process can call upon other (children) processes. In Figure 5.2, the TCP layer is realized by a single TCP manager process as shown in Figure 5.3. The functions of the TCP manager is a relatively simple process involving only a few states. However, each TCP connection is handled by an independent TCP connection management sub-process. Thus, connection management is made simpler. Each connection management process is responsible for maintaining the state of the connection it is catered for. The connection management process being used in the simulation conforms to RFC 793 [4] and is shown in Figure 5.4 which is identical to the realization on pp. 199 of [13]. The main intelligence of connection control falls upon the ESTAB state within the TCP_CONN module. This type of process hierarchy abstraction can break down a complicated task into a modular system. This is very well suited for TCP/IP and other connection oriented

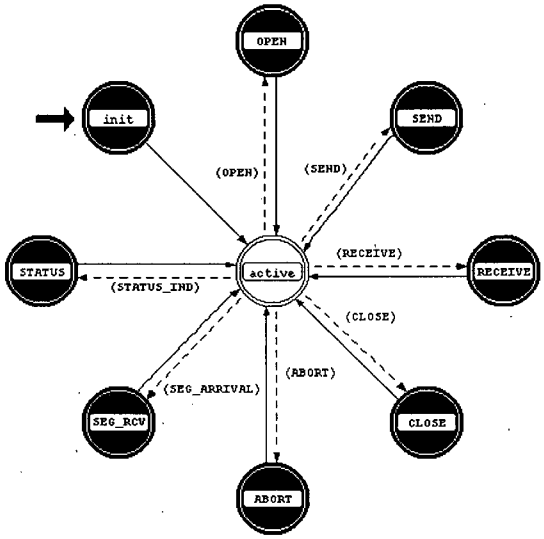


Figure 5.3 TCP manager employed within TCP/IP standard model

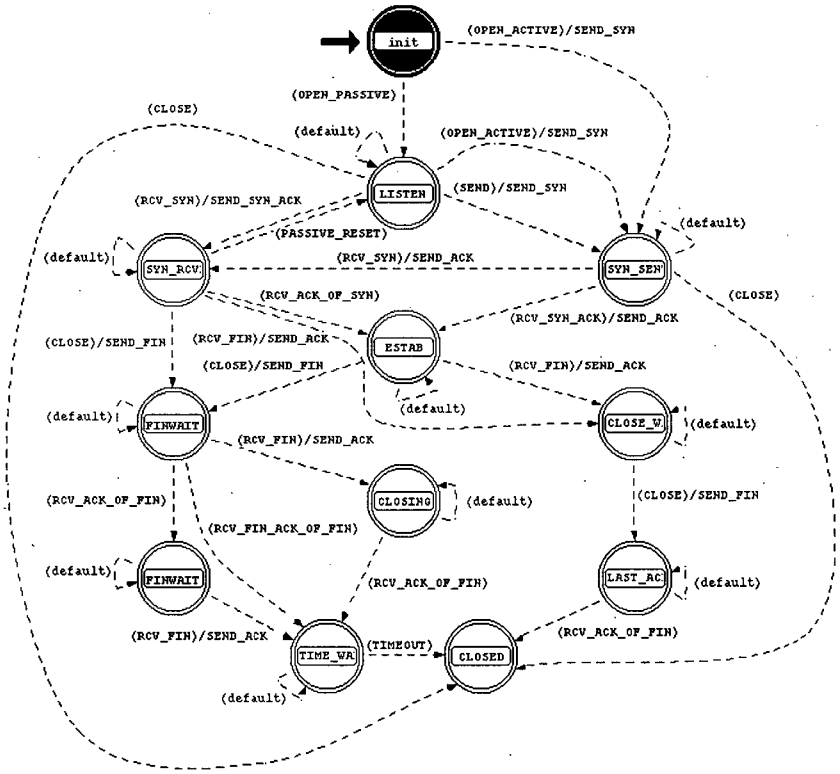


Figure 5.4 TCP connection management sub-process

protocols alike. The only limitation is the memory constraint of the workstation. This makes it

very suitable for simulations of "Layered Protocols". Information transfer can be classified into two main categories: packet driven and internal control interface (ICI). The normal way to communicate is to send information in data packets. There are occasions that a layer has to make certain control over other layers. For example, in the operating system implementing TCP/IP, the IP layer receives a UDP datagram and routes it to the appropriate destination. A UDP datagram does not contain information regarding the destination IP address due to the layering of protocols. It is obvious that absolute layer transparency is impossible [13]. Hence, that information is sent via internal control interface. The packet format and the ICI format can be selected using Parameter Editor in OPNET.

5.2 Design of Simulation Models

In previous chapters, a total of three different variations of the mobile-IP protocols were introduced. In order to assess the transport layer performance of each of these models, it is required to create three different models in OPNET™. Since UDP is used for delivering mobile registration messages, unlike a conventional TCP/IP model, there is a UDP layer in parallel with the TCP layer. Any host employing mobile-IP extension needs the UDP module in its node architecture in the simulation.

UDP delivery system is not supported in standard OPNET modules. In standard OPNET TCP/IP libraries, protocol demultiplexing is not implemented. Therefore, modifications has to be made to accommodate this need. If the mobile-IP models developed are to be used with other standard modules (e.g. IP over ATM), a protocol demultiplexing module has to be added between the Transport and the Internet layer. The module has to demultiplex datagrams from network and send them to appropriate transport layer protocols. It also has to mark the protocol field in the

header so that the receiving side will know what type of datagram it is. If any of the mobile-IP modules are to be used with other standard modules, this demultiplexing module has to be included to ensure proper interfacing. This demultiplexing unit is shown as *ip_encap* in Figure 5.10.

5.3 Network Configuration

Analysis is based on the network model in Figure 5.5. There are two mobile hosts, MH_1 and MH_2. MH_1 belongs to net_1, whereas MH_2 belongs to net_2. The networks net_1, net_2, and net_3 are part of the Internet. They are drawn explicitly for clarity and to show the mobile connections with MH_1 and MH_2. The two mobile hosts are designed to hop between foreign networks at predefined rates. Traffic characteristics were studied at different rates of handoffs.

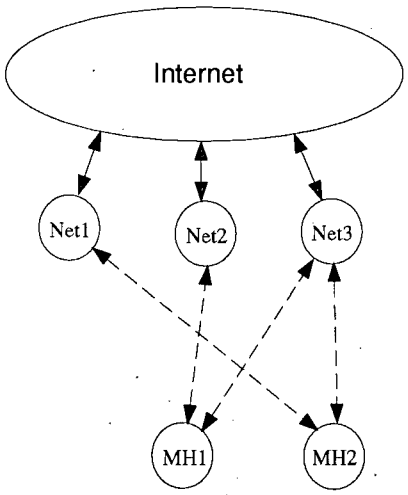


Figure 5.5 Network configuration of the simulation

As both of the mobile hosts are away from home, each of them has to register with their respective home networks in order to acquire network service from them. Besides, they also have

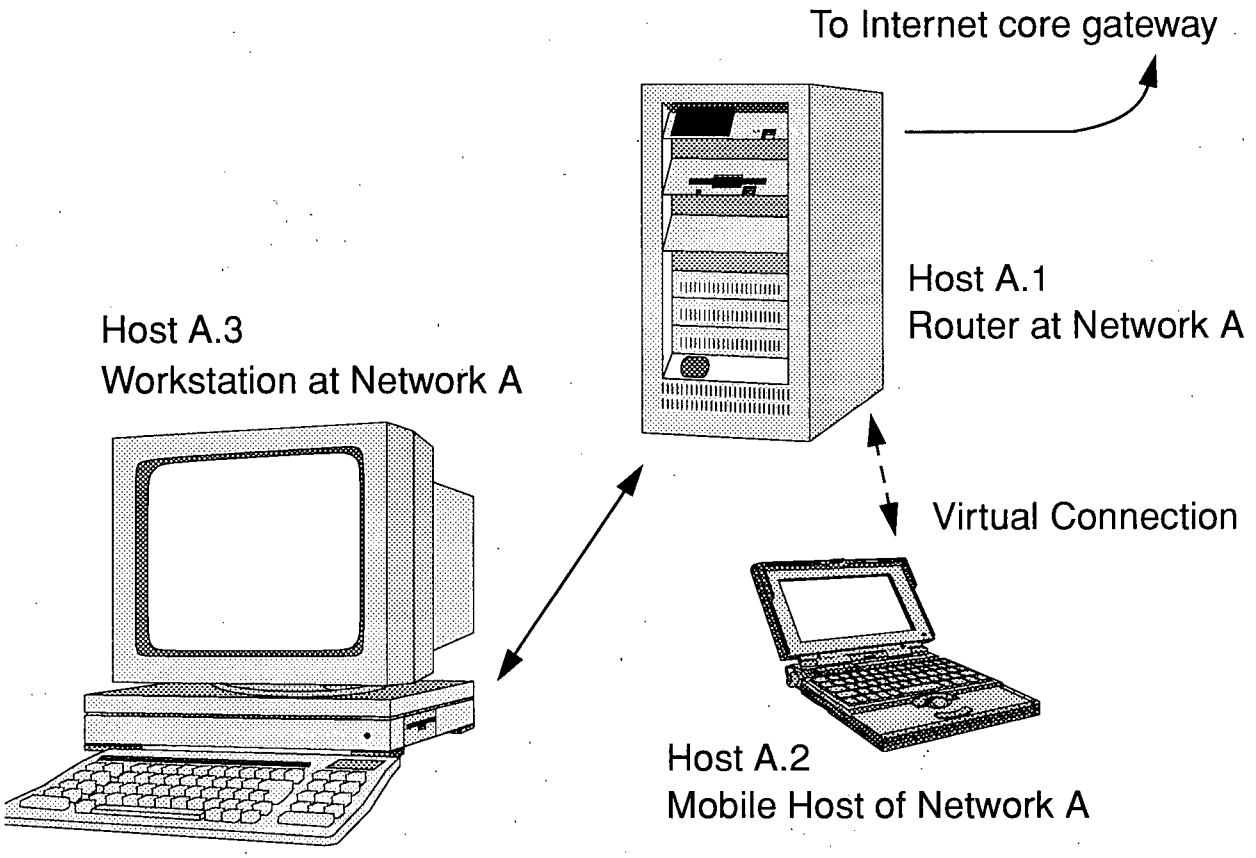


Figure 5.6 Host configuration within subnetworks

to rely on foreign agents to route their datagrams. Each of the networks (net1, net2 and net3) has a local router and a conventional host. Figure 5.6 depicts the connection topology within each of the networks in Figure 5.5. Host (A.1) is a local router. It serves as both FA and HA (i.e. it provides mobile-IP services for both visiting MHs and MHs belonging to the same network). Host (A.2) is a MH. It is connected via a virtual connection to its HA (host A.1). Host A.3 is a conventional host connected directly to its home router (host A.1). Since we are concentrating on Transport layer analysis, there is no assumption made upon the type network interface. In the simulation, each of the mobile-IP supporting router is named as a mobile agent (MA). MA is therefore a router which delivers datagrams for both conventional hosts and MHs. The former is just conven-

tional IP routing, whereas the latter is by the use of mobile-IP. Standard OPNET Internet model uses routing table lookup method to implement route selection. Internet traffic within the local subnet is handled by the local router (MA). For traffic that goes beyond the network boundary, they are routed to a default core gateway. This core gateway is capable of routing any IP datagrams to their respective networks, where they will be distributed locally. This scheme is outlined in chapter 13 of [13]. Figure 5.7 shows the core gateway employed in this simulation.

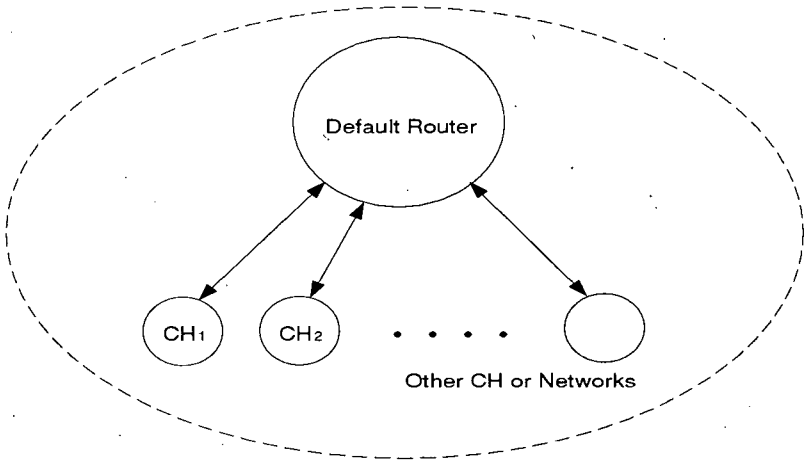


Figure 5.7 Core gateway model of the Internet model

The use of this core gateway serves two functions in this simulation. First of all, it makes the conventional routing task simpler by having a relatively small routing table at each of the local routers. As the Internet today consists of a large number of hosts, it is therefore resource inefficient to keep track of all the hosts' routing information on every host within Internet. In reality, existing local hosts keep minimal information regarding routing and rely on major gateways within Internet to do the rest of the routing. This is analogous to what is being implemented in OPNET. However, autonomous routing system concept is not implemented. An autonomous system advertises reachability information between routers. This is useful for indicating a certain

network, link or routing path is not functional. The system will then be able to route datagrams via a different path. In our simulation, since the number of networks involved is few and we have one core gateway only, implementing autonomous system would not bring any improvement.

All the default traffic has to go through the core gateway, so datagrams follows a single queue waiting to be routed as in Figure 5.8. This introduces a varying delay which the existing

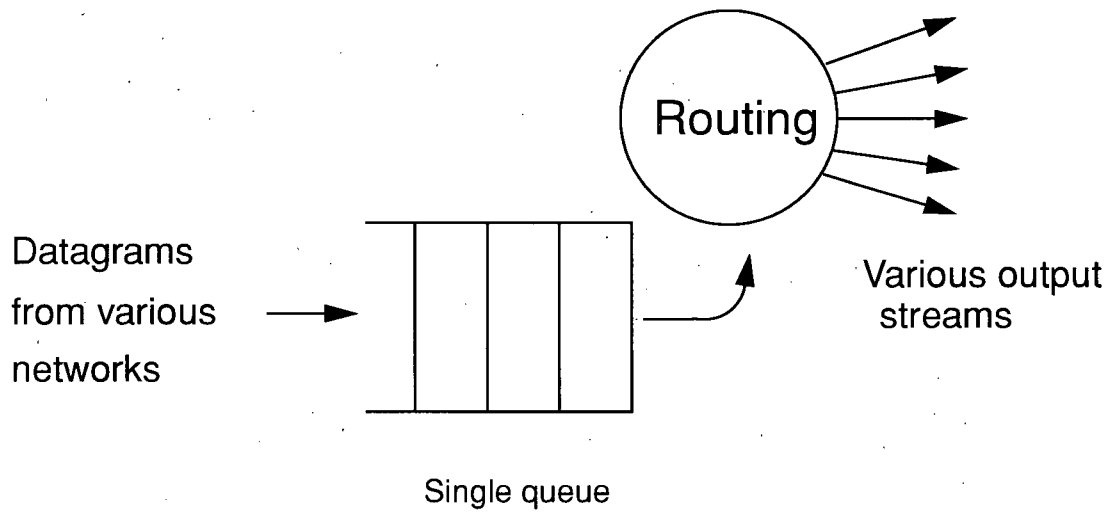


Figure 5.8 Queuing at the Internet core gateway

Internet imposes in datagrams delivery. The delay for every non-local datagram is varied, giving a more realistic representation of the current Internet. This approach of simulating TCP/IP traffic is commonly used and there are many variations of it. The simulation model in [3] uses a similar approach by having a single core gateway.

5.4 Host Configuration

There are basically three types of hosts in this simulation. They are conventional hosts,

mobile hosts (MH) and mobile agents (MA). Conventional hosts are those which will connect only to wireline hosts. It does not support any of the mobile-IP extensions. There is no UDP section within this type of hosts. It is shown in Figure 5.9. There is no assumption made upon the

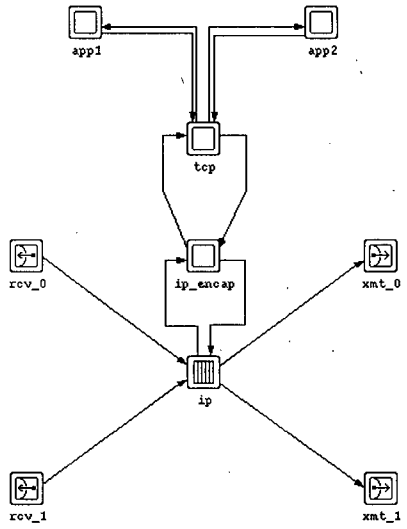


Figure 5.9 Components of conventional host

physical connection method. It is implemented as a direct link between hosts. Transmitters and receivers are point-to-point type. The IP module is the standard OPNET IP module. It is capable of handling fragmentation and assembly of IP datagrams. In addition, routing table is kept for making routing decisions. The *ip_encap* module will provide interfacing between TCP and IP layers, exchanging information regarding source and destination addresses of the datagrams being handled. TCP is responsible for maintaining the end-to-end connections. App1 and app2 are sources and sinks of data sent over the network.

MHs and MAs share an identical *node* layout, as shown in Figure 5.10. The difference between a MH and a MA is the registration handling unit - *reg* module. For a MH, the *reg* module calls a process which sends registration request to its HA. It also performs handoff at specific

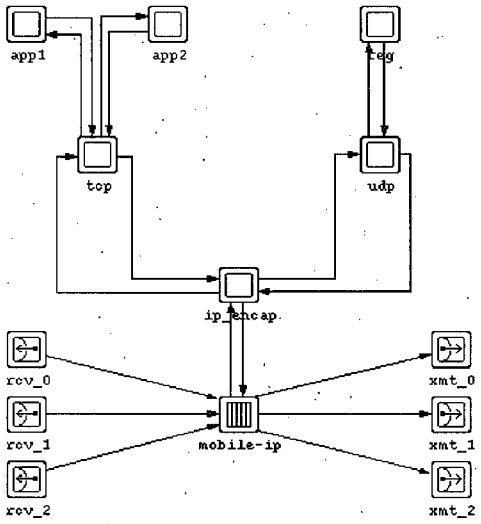


Figure 5.10 Components of mobile host or mobile agent

intervals. For a MA, the *reg* module calls a different process which handles registration requests and updates mobility binding for MHs.

Either a MH or MA has to support conventional IP routing as well as mobile-IP routing. Mobile-IP routing is based on the contents of the mobility bindings stored in the mobile-IP layer, and is able to change its delivery path during the course of a connection. Several lists of mobility bindings have to be maintained in order to accomplish this. For a MH, it has a list of bindings storing the information of its home agent, foreign agent and link-layer address. For a HA, there is a list of HA bindings. Each of those will store the information for those MHs which have moved away from this network. For a FA, it has to keep a mobility binding for each visiting MH.

Each of the MH sends a mobile registration when the simulation starts. Shortly before the lifetime of the registration expires or the MH has moved to another location, it sends a new registration request to the new FA. These mobile-IP messages are sent via UDP. Therefore, the *ip_encap* module has to demultiplex datagrams according to its type. When the mobile registra-

tion request reaches the *reg* module of a FA or a HA, it updates the appropriate list of mobility bindings. To accomplish this, the module interrupts the simulation flow and passes the necessary information via an ICI structure in OPNET.

Routing implementation in mobile-IP is significantly different than the standard OPNET IP module. When mobile-IP layer is required to make a routing decision, it first consults its mobility binding lists. For a MH, it always looks for a FA to deliver its datagrams. Whereas for a mobile agent (either a HA or a FA), it has to consult its bindings list to determine where to route the datagram to. If there is no mobility binding related to the destination of the datagram and it is not a MH, conventional Internet routing is used for datagram delivery.

5.4.1 Basic IETF Mobile-IP Scheme

This model has the basic features outlined in the basic IETF mobile-IP proposal [6]. In this model, mobile-IP routing, conventional Internet routing and fragmentation handling are supported. The mobility functions are separated in two different modules. The mobile routing and tunnelling are implemented within the mobile-IP layer, whereas the mobile registrations are kept in the application layer and transported via UDP.

5.4.1.1 Mobile-IP layer in basic Mobile-IP

The mobile-IP module is developed to replace the standard IP module in OPNET. It handles all mobile routing and tunnelling. It also maintains the cache lists of bindings and removes expired bindings.

This module is used by any hosts which supports mobile-IP extensions. The fragmentation handling within this module is based on the standard OPNET IP module, while the routing is

Table 5.1 Mobility binding lists in basic mobile-IP

List	Usage
Home Agent List	Storing care-of addresses for each mobile host
Foreign Agent List	Storing home agent address for each mobile host
Mobile Host List	Storing home agent address and foreign agent address

rewritten to accommodate the additional features. Before a datagram is routed, the source and destination addresses in the datagram header are read. In order to find a route for the destination, the lists of mobility bindings are consulted. Figure 5.1 shows the types of mobility bindings that are kept within the mobile-IP layer. If there is no match in these lists for the current datagram to be sent, the datagram is sent via conventional routing. Binding entries are represented as a linked list in C programming language. Home agent binding entries are listed in Table 5.2. Note that in a HA binding list, it is possible to have multiple entries for a single MH. This is to enable a MH to acquire services from multiple FAs. In this case, datagrams for this MH are duplicated and sent to each valid care-of address. The identifier is for validating mobile registration request. This is a known secret between the HA and the MH for authentication purposes. Timeout value is the duration in seconds that this binding should be allowed to exist. Refresh handler is an OPNET

Table 5.2 Home agent binding entries in mobile-IP

Search identifier	Entries	Unit
Mobile Host IP address		4-byte integer
	Care-of address (usually Foreign Agent address)	4-byte integer
	Identification	4-byte integer
	Timeout value	4-byte integer
	Refresh Event handler	Event pointer in OPNET

Table 5.3 Foreign agent binding entries in basic mobile-IP

Search identifier	Entries	Unit
Mobile Host IP address		4-byte integer
	Home agent IP address	4-byte integer
	Link-layer address	4-byte integer
	Timeout value	4-byte integer
	Refresh Event handler	Event pointer in OPNET

internal pointer which allows the module to cancel the binding refresh event. For example, if there is a deregistration before it has expired. It would then be necessary to cancel the refresh event as well to avoid accessing a stale binding after it has been deleted.

FA binding entries are shown in Table 5.3. Similar to a HA binding, timeout value and refresh handler are for maintaining the binding within the allowed lifetime. Note that each MH can only have one HA address in the binding. When a FA first receives a mobile registration from a MH, it creates a temporary binding storing the link-layer address to which the MH is attached.

When an encapsulated datagram is received, the FA determines whether it is the end of the tunnel for this datagram. If so, it decapsulates the datagram and checks the inner payload to find a match within its visitors' bindings. If a match is found, the datagram will be delivered locally according to the link-layer address in the binding.

For a MH, the binding is shown in Table 5.4. Whenever the MH is required to send a mobile registration, it determines which HA to register with and sends the registration to that IP address accordingly. (Note that there may be more than one HA available within a network.) The care-of address is obtained from the local FA which MH is acquiring network service from. The link-layer address is the physical layer address for routing. Identification serves as a common

Table 5.4 Mobile host binding in basic mobile-IP

Home agent information	Foreign agent information	Unit
Home agent IP address		4-byte integer
	Care-of address	4-byte integer
	link-layer address	4-byte integer
	identification	4-byte integer
	lifetime	4-byte integer
	Refresh handler	Event pointer in OPNET

secret between the FA and the MH. The time of existence of this binding is specified by the lifetime field. The event handler provides a way to cancel the binding refresh event, should it become unnecessary.

With all these bindings defined in the mobile-IP module, all MHs and MAs share a common mobile-IP module. The state diagram is shown in Figure 5.11. The state transition is

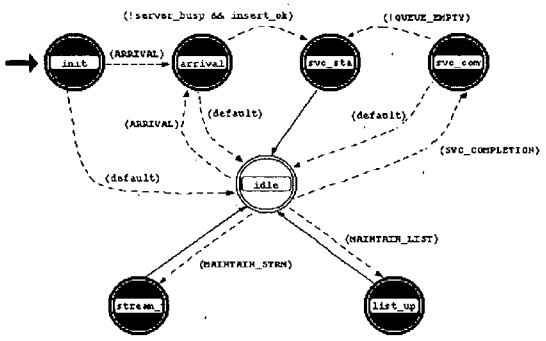


Figure 5.11 State diagram for mobile-IP module for basic mobile-IP scheme

outlined in Table 5.15. Note that there are two states which is called by the *reg* module. The two states are *stream_update* and *list_update*. Stream_update is for simulating handoff by connecting to a different physical stream, while the list_update is for maintaining the mobility bindings. The supplementary source code of the IETF model is listed in Appendix A.

Table 5.5 State transition of *mobile-ip* unit in basic mobile-IP

State	Usage	Next state
init	initialize variables, read simulation parameters and datagram arrives	arrival
	initialize variables, read simulation parameters and no arrivals yet	idle
idle	datagram arrives	arrival
	finished service for datagram	svc_completion
	stream update requested by *reg* module	stream_update
	binding update requested by *reg* module	list_update
arrival	schedule datagram for service	svc_start
	arrived datagram placed in queue	idle
svc_start	start service, schedule for finish	idle
svc_completion	finish routing (if more datagrams in queue)	svc_start
	finish routing (if no more datagrams in queue)	idle
stream_update	perform stream update (return control to *reg* module)	idle
list_update	perform binding update (activate by *reg* or current module)	idle

5.4.1.2 Mobile Registration modules in Basic Mobile-IP

There are also modules in the application layer developed for handling mobile registration procedures. There are two different types of registration modules. One is for MHs, while the other one is for both FAs and HAs. It is expected that a mobile router can function both as a FA and a HA.

5.4.1.3 Home Agent in basic Mobile-IP

The responsibilities for home agents in basic mobile-IP scheme is listed in Table 5.6. Each

Table 5.6 Responsibilities of home agent in basic mobile-IP

1	handle mobile registration request
2	send mobile registration reply
3	set up cache binding for mobile host that has been successfully registered
4	tunnel datagrams for registered mobile hosts
5	perform deregistration when mobile host has returned to home network

HA has two special units which handles the mobile datagram routing and registration procedures. They are shown in Figure 5.10 as the *reg* and the *mobile-ip* units. The *reg* unit analyzes all mobile registration requests and performs mobility binding updates. The state diagram for this registration handling unit for a home agent is shown in Figure 5.12 and the corresponding state transition is shown in Table 5.7.

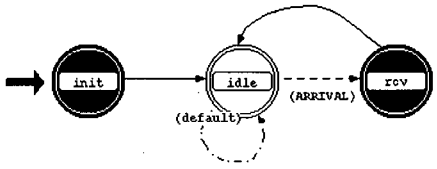


Figure 5.12 State diagram for mobile registration handling of mobile agent in basic mobile-IP

Table 5.7 State transition for *reg* unit of home agent in basic mobile-IP

State	Functions	Next State
init	initialize variables, read simulation parameters, send initial mobile registration	idle
idle	pending for mobile registration request	rcv
rcv	analyze registration request, update mobility binding, send registration reply	idle

Under this basic scheme, the *reg* unit takes a passive role in mobile registration management. HAs do not initiate any messages to FAs, MHs or other CHs. It only responds to mobile registration requests.

5.4.1.4 Foreign Agent in basic Mobile-IP:

Foreign agent in this basic mobile-IP scheme plays a silent role as well. It merely relays the mobile registration message from the mobile host to its home agent. It analyzes the registration requests and creates necessary mobility bindings according to the reply from the HAs. The functions of foreign agent is summarized in Table 5.8. These functions are realized and

Table 5.8 Responsibilities of foreign agent in basic mobile-IP

1	forward mobile registration request
2	relay mobile registration reply
3	maintain cache binding for mobile host that has been successfully registered

represented in a state diagram as shown in Figure 5.12.

When a FA first receives a mobile registration request from an unregistered MH, it first checks its own available resources. If permitted, a temporary binding is created. As soon as the registration reply from the HA has been received, the FA updates the visitor binding for this particular MH. The registration reply is then delivered back to the MH. State transition is shown in Table 5.9.

Table 5.9 State transition of *reg* unit for foreign agent in basic mobile-IP

State	Functions	Next state
init	initialize variables, read simulation parameters	idle
idle	pending for mobile registration requests or replies	rcv
rcv	update visitor binding list according to the message received	idle

5.4.1.5 Mobile Hosts in basic Mobile-IP:

Under the basic mobile-IP scheme, MHs initiate all mobile registration requests. In addition, handoff management is also performed by the *reg* module. The responsibilities are outlined in Table 5.10.

Table 5.10 Responsibilities of mobile host in basic mobile-IP

1	initiate new mobile registration request
2	process mobile registration request reply
3	process mobile registration reply
4	retransmit mobile registration before expiry

All of the above functions are implemented in the *reg* module and is implemented in a finite state machine in OPNET which is shown in Figure 5.13.

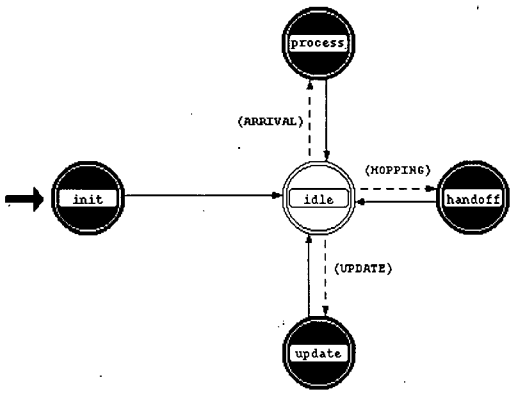


Figure 5.13 State diagram of *reg* module of mobile host in basic mobile-IP

Note that there is an extra handoff state within this module. The handoffs are simulated using a timer module within the state. Whenever the predefined handoff period expires, it registers with another FA. After the registration has been approved, the MH does not register with its previous FA. The previous FA deletes the binding when it has expired. The state transition is shown in Table 5.11.

Table 5.11 State transition of *reg* unit for mobile host in basic mobile-IP

State	Functions	Next state
init	initialize variables, read simulation parameters	idle
idle	pending for registration reply	process
	waiting for retransmission timer to expire	update
	waiting for handoff timer to expire	handoff
process	analyze registration reply, update mobility binding	idle
update	send retransmission if registration reply not received within timeout period	idle
handoff	send new mobile registration to another foreign agent whenever handoff time-out expires	idle

5.4.2 IMHP Route Optimized Mobile-IP model

For the IMHP scheme, all the basic mobile-IP features are supported. There is one additional feature as well. First of all, there is one more entity needed to be defined - cache agent (CA). The role of CA can be taken by any host along the path between the corresponding host and the home agent. Since datagrams can take on different path even though the endpoints of travel is the same, there is no guarantee that subsequent datagrams will take the exact same route during next travel. The responsibility of being a CA often falls upon the CH. The function of CA is to keep track of the care-of address of a certain mobile host. It would then be able to route datagrams for that particular address directly to its care-of address. Therefore, a cache list is kept at CA to indicate the MHs' care-of addresses.

Obviously there are additional duties to be carried out by the CA and the HA for this route optimization scheme. They are outlined in Table 5.12 and Table 5.13 respectively. In supporting route optimization, the CH has to include an additional module in its application layer, the state

Table 5.12 Responsibilities of cache agent

1	acknowledge binding update request
2	send binding update request
3	maintain cache binding listing for different mobile hosts
4	delete outdated mobility cache bindings

Table 5.13 Additional responsibilities for home agent to implement route optimization

1	send binding update warning to indicate a lack of up-to-date care-of address of mobile host
2	exponentially backoff in sending binding update warning after repeated no-response from Cache Agents
3	process and authenticate binding update request

diagram is shown in Figure 5.14. The function of this module is normally pending for binding

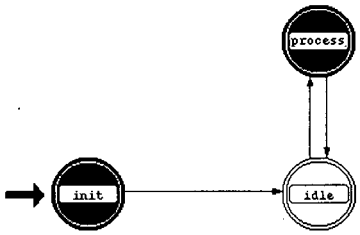


Figure 5.14 State diagram of cache agent using route optimization scheme

update warning. If that type of message is received, the CH sends a corresponding binding update request to the HA which originates the warning. After the binding update message has arrived, the CA updates the cache binding list in mobile-IP layer. Subsequent datagrams for the same MH are sent directly to its care-of address. Figure 5.15 depicted the mobile-IP layer routing. The state transition is listed in Table 5.14. The supplementary source code for the IMHP scheme is shown in appendix B.

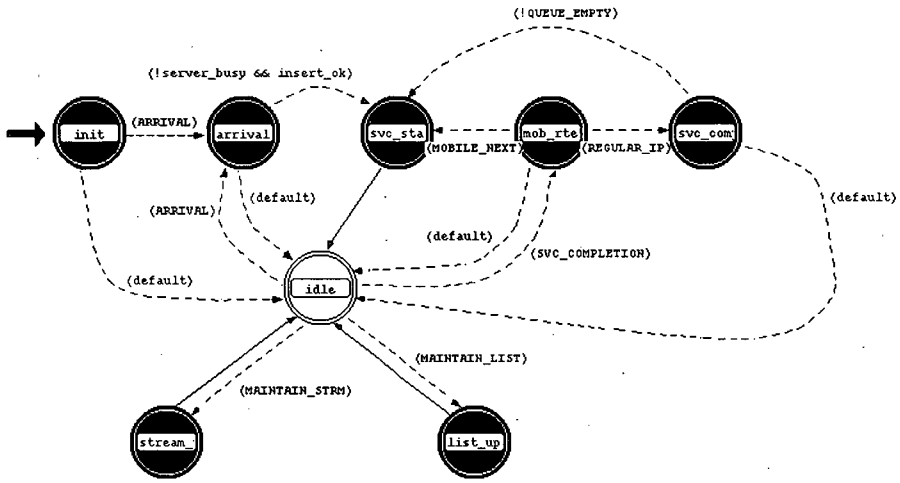


Figure 5.15 State diagram of mobile-IP module in route optimized scheme

Table 5.14 State transition of mobile-IP module in route optimized scheme

State	Description	Next state
init	initialize variables, read simulation parameters, and no arrivals yet	idle
	initialize variables, read simulation parameters, and datagram arrives	arrival
idle	datagram arrives	arrival
	datagram routing started	mob_rte
	stream update interrupt received	stream_update
	binding update request received	list_update
arrival	datagram arrived and enqueued	idle
	datagram arrived and start service	svc_start
svc_start	service finish time scheduled	idle
mob_rte	mobile datagram sent and schedule next datagram in queue for service	svc_start
	routing conventional IP datagram	svc_completion
	mobile datagram sent and no datagrams in queue	idle
svc_completion	finished sending datagram and no datagram in queue	idle
	finished sending datagram and schedule next datagram for service	svc_start
stream_update	switch to next physical stream	idle
list_update	perform binding update request	idle

5.4.3 Handoff Enhanced Scheme

For the enhanced mobile-IP scheme, all the route optimization features are supported together with one additional function for the FA. Each FA has to set aside a specific amount of memory buffer for storing datagrams destined for MHs when they are undergoing new mobile registration procedures at another FAs. The state diagram for this scheme is shown in Figure 5.16. The buffering of datagrams make this scheme structurally different from any of the previous schemes described. It requires some memory to be set aside. Moreover, the state transition is more sophisticated. It has to store up data packets for any visiting hosts which have just left for other

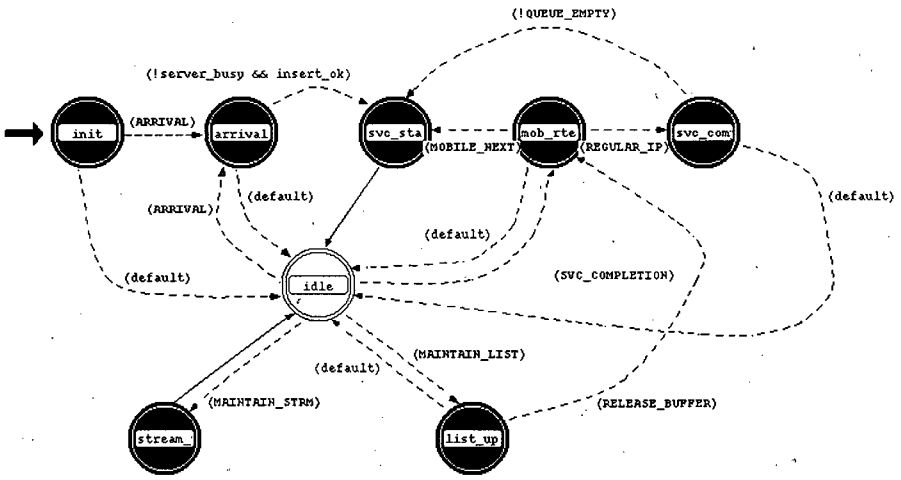


Figure 5.16 State diagram of mobile-IP module using Enhanced mobile-IP scheme

destinations. Upon pending for authenticated replies, the host has to queue up the packets. The state transition is given in Table 5.15. The supplementary source code of the Handoff Enhanced Scheme is listed in appendix C.

Table 5.15 State transition of *mobile-ip* unit for mobile host in handoff enhanced mobile-IP

State	Description	Next state
init	initialize variables, read simulation parameters and if datagrams arrive	arrival
	initialize variables, read simulation parameters and if no arrivals yet	idle
idle	datagram arrives	arrival
	finished service for datagram	mob_rte
	stream update requested by *reg* module	stream_update
	binding update requested by *reg* module	list_update
arrival	schedule datagram for service	svc_start
	arrived datagram placed in queue	idle
svc_start	start service, schedule for finish	idle
mob_rte	finish sending datagram for mobile host and if there are more datagrams in queue	svc_start
	no mobility binding exists, use conventional IP routing	svc_completion
	finish sending datagram and no more pending datagrams in queue	idle
svc_completion	finish routing (if more datagrams in queue)	svc_start
	finish routing (if no more datagrams in queue)	idle
stream_update	perform stream update (return control to *reg* module)	idle
list_update	perform binding update (return control to *reg* module)	idle
	release buffer command received	mob_rte

Chapter 6 Discussion of Simulation Results

Simulations were performed using models developed in previous chapters. Performance analysis was based on connection-oriented traffic. The effect of those parameters will be discussed in relevant sections later in this chapter.

6.1 Review of Simulation parameters

The simulation starts with the MHs initiating mobile registrations. The MH sends a mobile registration request to its HA. Upon receiving the registration request, the HA responds with a registration reply. The setup is listed in Figure 5.5.

Normal TCP connection was set up between MH_1 and Internet CH_1. An identical connection was set up between MH_2 and CH_2. The TCP connections remained open for a period of 2000 seconds (over 30 minutes). Handoff was implemented by using an internal timer. When the time had expired, the MH would switch to another mobile link which was connected to another FA. That would enable the MH to receive mobile service from a new FA. During the period of the connection, MH_1 was set to hop between net_2 and net_3, while MH_2 was set to hop between net_1 and net_3.

It is specified in [1] that consecutive mobile registrations should not be less than 1 sec apart. Preliminary results also show that TCP end-to-end delay does not change significantly when time between handoffs is increased from 40 secs. Hence we chose the handoff period to be as follows. MH_2 was set to perform a handoff every 40 seconds, while the time between handoffs for MH_1 was varied in a range from 40 seconds down to 2 seconds.

The TCP end-to-end delay of the connection was monitored as the prime interest of the study was the investigation of TCP traffic conditions over mobility links. The TCP end-to-end delay is defined as the time duration between the TCP segment entering the TCP layer of sending host and leaving for the application layer at the receiving host.

Table 6.1 Parameter set 1

Parameter type:	
Mobile link data rate:	57.6 kbps
Data rate within local network:	10 Mbps
Data rate from local network to Internet sites:	256 kbps
Maximum acknowledgment delay:	2 sec
RTT gain (α):	0.125
RTT deviation gain (λ):	0.25
RTT deviation constant (β):	4

The data segment size was set at 1024 bytes and the generated traffic was exponentially distributed with a 0.1 second interarrival time. The simulation was repeated using ten different simulation seeds using the parameters shown in Figure 6.1. An ensemble average is taken to ensure that the simulation would give a more realistic picture as opposed to a single case study. The results were averaged and tabulated. Since MH_2 had a fixed rate of handoff, the performance over this link was used as a basis for comparison with other handoff rates.

The *mobile link data rate* specifies the data rate from the MHs to the corresponding FAs. The *data rate within local network* is the data rate between a local router and conventional hosts within the same network. The *data rate from local network to Internet sites* shows the data rate from the local router to the Internet core gateway. The *maximum acknowledgment delay* is the maximum time allowed that the TCP layer will wait for an acknowledgment before sending

retransmissions. The *RTT gain* (α) and *RTT deviation gain* (λ) are the weighing factors that the TCP layer uses in updating the *RTT* average and deviation respectively, while the *RTT deviation constant* (β) is the multiplicative constant that TCP uses in generating a retransmission timeout value from the average *RTT*.

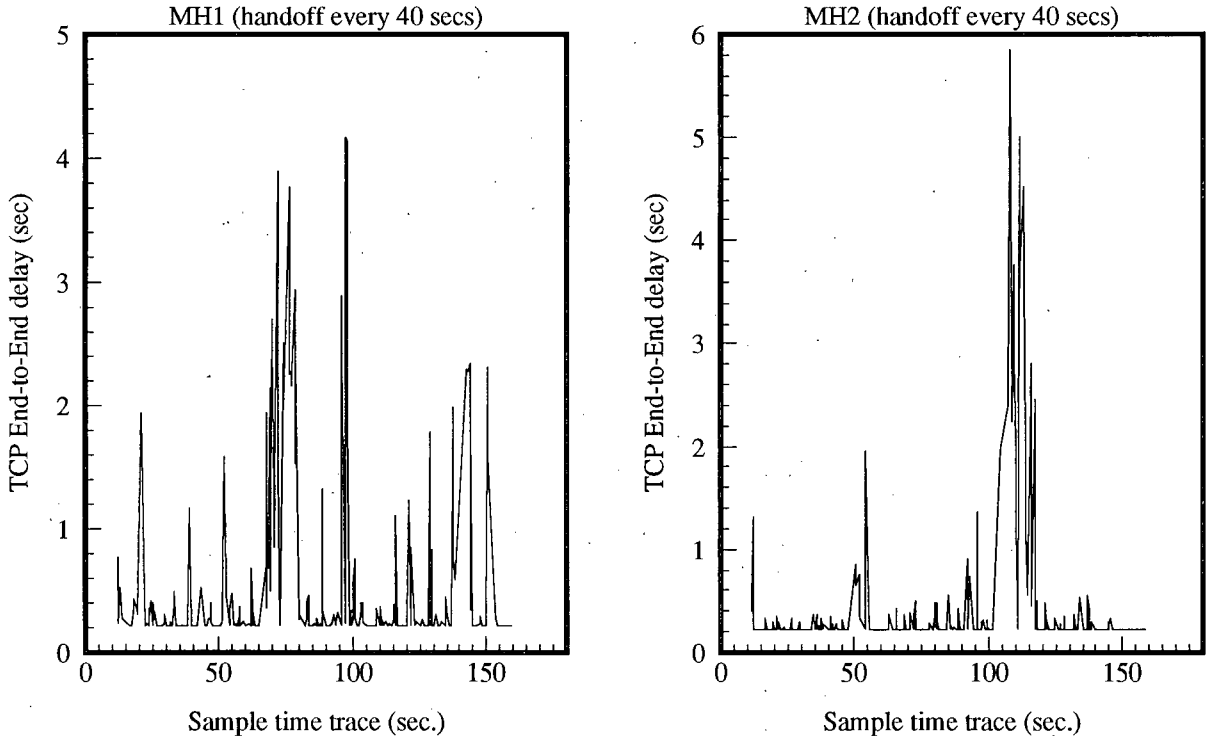


Figure 6.1 Delay connection characteristics of basic mobile-IP scheme (handoffs every 40 sec)

Figure 6.1 shows end-to-end connections measurements of MH_1 and MH_2 with handoff every 40 seconds. It is shown that at this rate, the delay is kept within a reasonable range. The performance of both MHs match up pretty closely as the link parameters are identical.

The same setup and identical parameters are used for analyzing the route optimized mobile-IP scheme (IMHP). In this scheme, triangular routing is eliminated. It is expected that improved performance can be achieved. It can be seen that the minimum floor end-to-end delay is

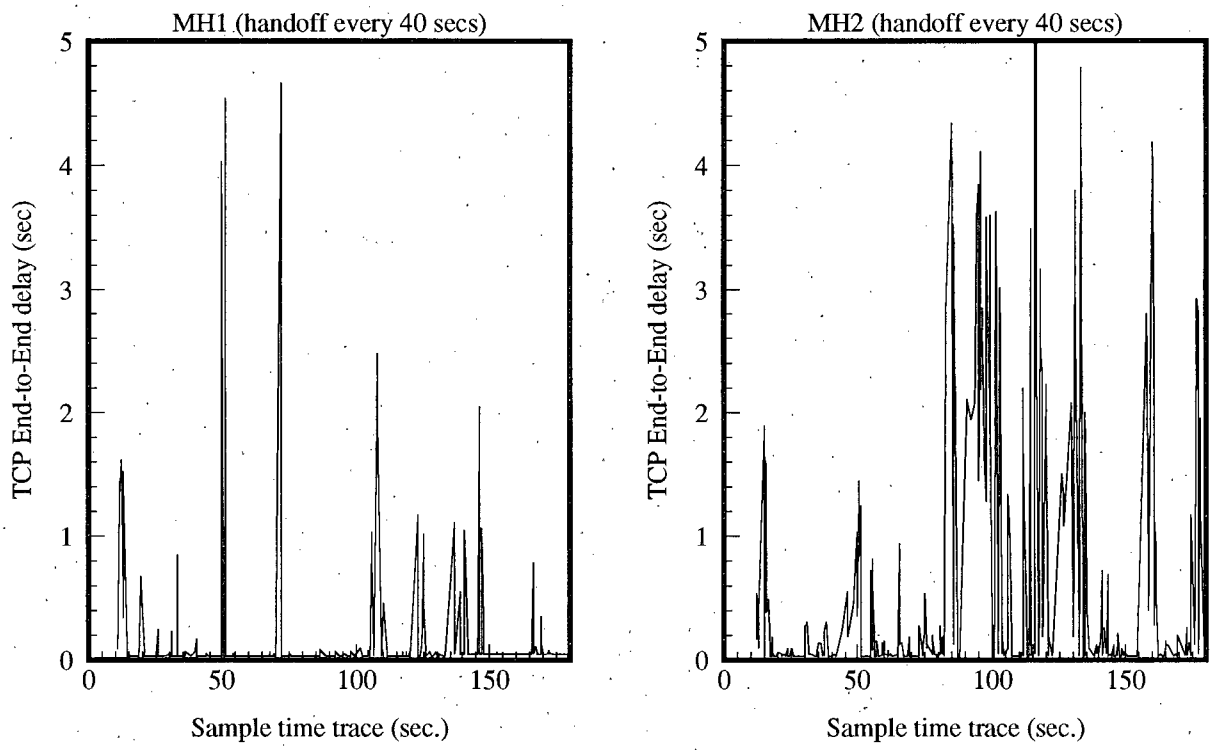


Figure 6.2 Delay characteristics of IMHP mobile-IP scheme (handoff every 40 sec)

considerably lower than that of the basic mobile-IP version. This matches up with our analysis in chapter two. At relatively low handoff rate, the end-to-end connection delay has been improved.

As a variant of route optimization scheme, the handoff enhanced scheme is expected to deliver similar performance as the route optimized scheme. The sample results are shown in Figure 6.3. The result verifies the analysis that at low handoff rate the handoff enhanced scheme delivers similar performance as the route optimized scheme. With this set of parameters, each of the schemes are capable of handling the traffic at an acceptable level. Therefore it is necessary to tighten the parameters in order to strain each of the schemes.

From the experimental results in the first part, it is apparent that TCP end-to-end delay will occasionally exceed a 2 second range. In order to strain the handoffs, the time between handoffs

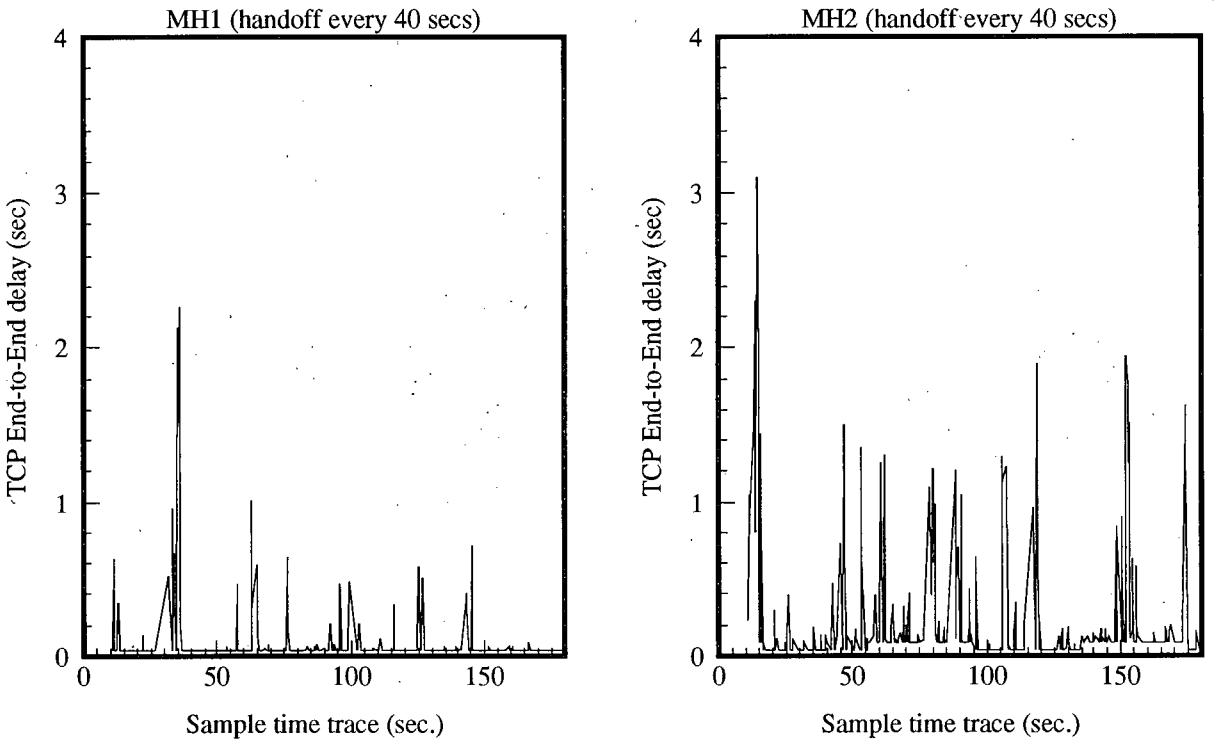


Figure 6.3 Delay characteristics of handoff enhanced mobile-IP scheme (handoff every 40 sec)

were reduced to 2 seconds. The same set of parameters, in Table 6.1, were used. Only the handoff rate of MH_1 was changed. This is to ensure that the change in performance is due to the difference in handoff rates but not from other factors.

The result for basic mobile-IP is shown in Figure 6.4. It is apparent that the performance of MH_1 has degraded tremendously compared with previous results (Figure 6.1). Although the floor level of the end-to-end delay is still quite low, there are delays which exceeds 10 seconds. On the other hand, MH_2 has similar performance compared to previous results. The change in performance is due to the handoff rate difference.

With the same handoff rate change applied to MH_1 as in the case of the basic IETF mobile-IP scheme, the simulation was performed using the IMHP scheme. The result is shown in

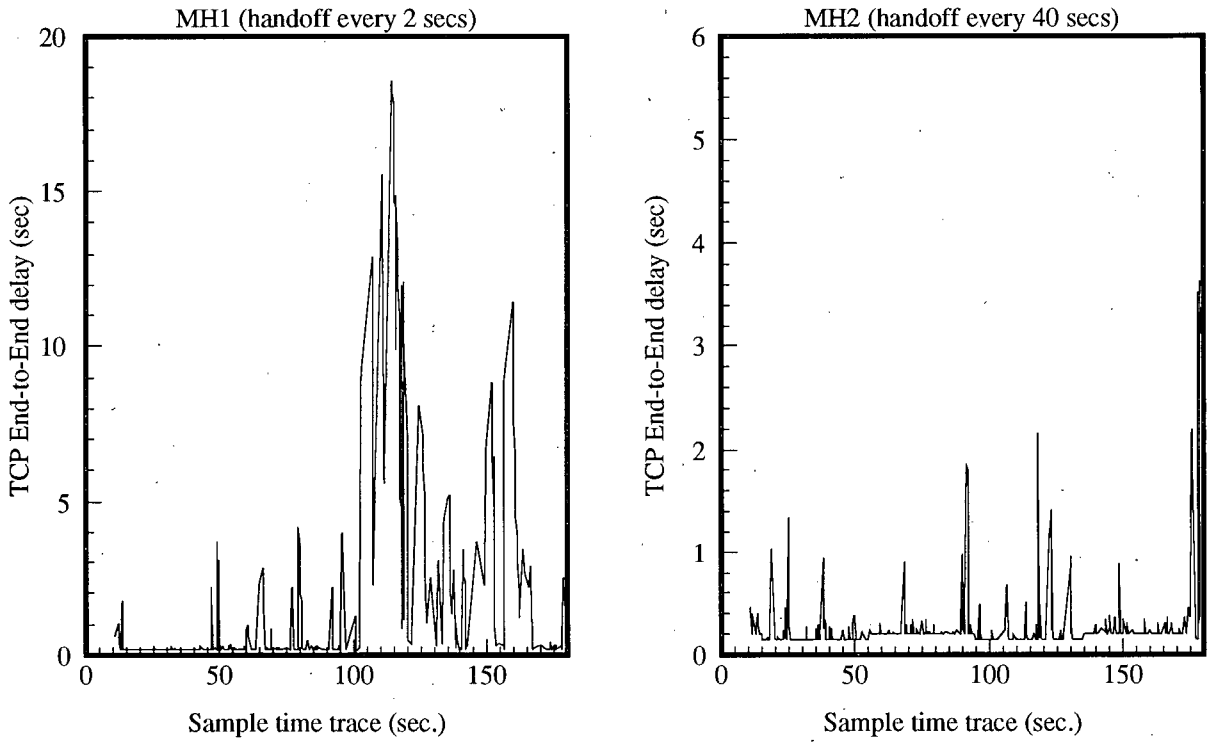


Figure 6.4 Delay characteristics of basic mobile-IP scheme

Figure 6.5. It can be seen that the performance degrades even more than the basic mobile-IP scheme. This matches with the analysis in chapter two. The results collected were verified through the use of a built-in debugger in OPNET to make sure the datagrams are routed according to the protocol specification.

The reason for the poor performance is that the datagrams are often routed to the incorrect location since MH_1 changes its point of attachment rapidly. This is because the CAs are delivering datagrams directly to the care-of addresses of MHs. This causes the TCP acknowledgment timer to expire. As described in section 3.2, TCP has mistakenly interpreted retransmissions as congestion within the network. TCP throttles the traffic by using exponential backoff strategy. Before TCP commences to send traffic down the link again, it sends SYN packets to establish the connec-

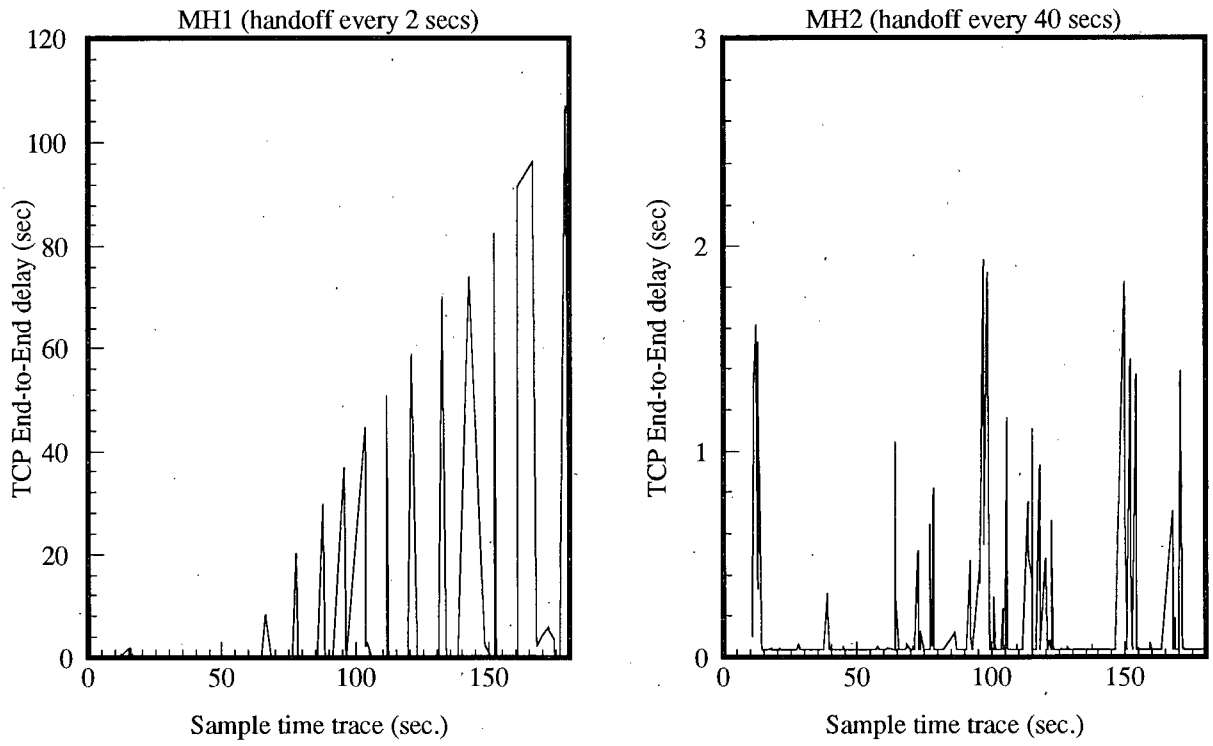


Figure 6.5 Delay characteristics of route optimized mobile-IP scheme

tion. As these packets are susceptible to losses as indicated, TCP ends up spending a great deal of time attempting to re-establish the connection. This scheme is deemed to have broken down under these network conditions.

The same conditions were applied to the handoff enhanced scheme. The results are shown in Figure 6.6. Although the handoff enhancement scheme is based on the route optimized scheme, it does not suffer from the huge delays as its predecessor. Under this scheme, the datagrams for mobile hosts are buffered at FA if the MH is registering with another FA. As soon as the registration completes, those buffered datagrams are sent to the MH. Nevertheless, these datagrams have to undergo a certain amount of delay before reaching their final destination. If however the travel time together with these delay still falls within the TCP retransmission count value, TCP does not

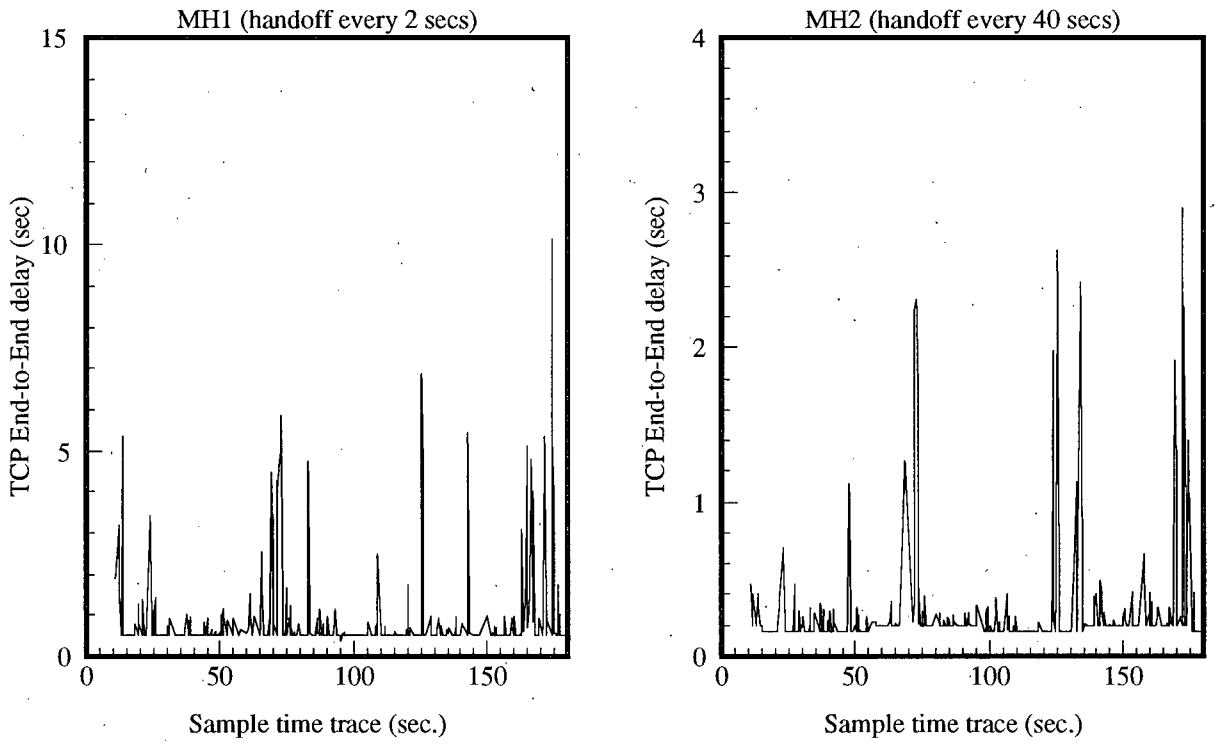


Figure 6.6 Delay characteristics of handoff enhancement scheme

try to synchronize and retransmit again. This is a major improvement over the route optimized scheme.

It still remains a question whether the handoff enhancement scheme is better than the basic mobile-IP scheme under these handoff intensive situations. The basic mobile-IP scheme does not utilize any route optimizations. It relies on the HA to tunnel the datagrams. As soon as the HA receives the mobile registration request from the MH, the HA updates its bindings accordingly. Datagrams can then be routed using the most update care-of address of the MH. Correct routing is the key to good performance in basic mobile-IP scheme, since the performance penalty in losing datagrams during handoff is severe in the transport layer. The key of good transport layer performance is to keep from losing datagrams.

Although the enhanced scheme, as a variant of the route optimized scheme, cannot guarantee the correctness of the routing during handoff. However, it has overcome the problem with another approach. By buffering the datagrams and forwarding it as soon as the registration is completed, the enhanced scheme can ensure that those datagrams received during the handoff will reach the correct host. In order to pick a best scheme among the three, there is a need to acquire long term averages of measured performance.

6.2 Simulation Results

Parameter set 1, in Table 6.1, is used and the result is shown in Figure 6.7. The time between handoffs ranges from 2 to 40 seconds, and the end-to-end delay of the basic mobile-IP scheme becomes the worst as it provides no route optimizations. For this situation with relatively few handoffs, the IMHP scheme performs the best with the end-to-end TCP delay about 0.2 seconds. As the handoff rate increases slightly, the end-to-end delay of the IMHP scheme increases sharply. This is because the datagram loss rate in the IMHP scheme is directly proportional to the number of incorrectly routed datagrams. Since route optimization scheme relies on CAs (often the corresponding host) to send the datagrams directly to FAs, those CAs do not have up-to-date information regarding the location of the MHs during handoffs.

The FAs do not play any active role in rectifying the problem. As the IMHP scheme cannot avoid losing datagrams, TCP interprets the situation as network congestion and performs exponential backoff and slow start which magnifies the problem. As a result, the IMHP scheme is very sensitive to change in handoff rate.

At low handoff rates, the new enhanced scheme shows a lower end-to-end delay compared with the basic mobile-IP scheme. However, as the handoff rate increases (i.e. the time between

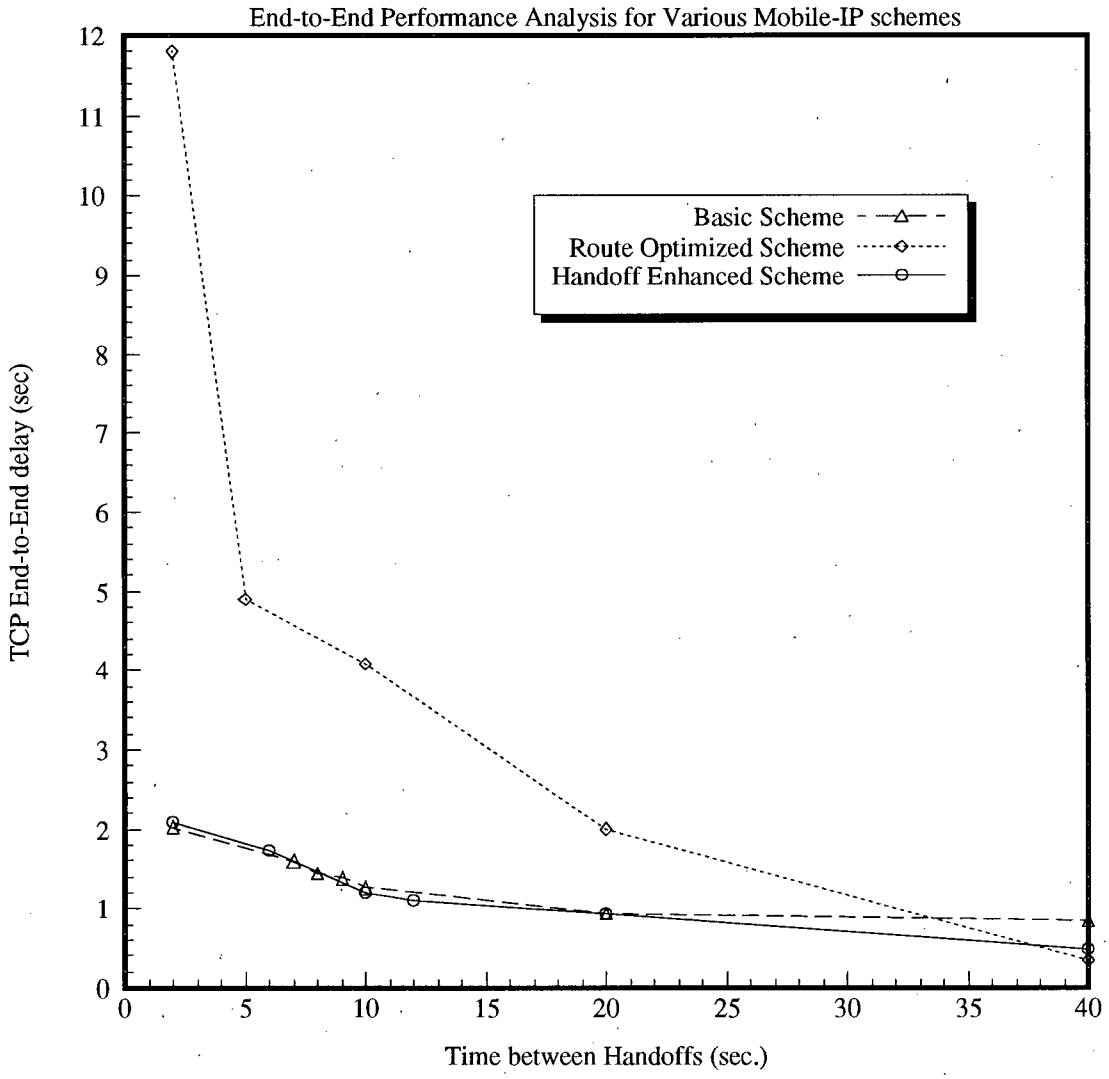


Figure 6.7 TCP performance for parameter set 1

handoff decreases), the two schemes give very similar results. This is due to the fact that the vulnerable period of the basic mobile-IP scheme is much smaller than the IMHP scheme, as described in chapter three.

A more detailed analysis shows that, for datagrams to be incorrectly routed, the datagrams must reach the HA during the handoff but before the HA is aware that the MH has moved to another FA. For this to happen, the datagrams must arrive during the time period marked as D_{HF1}

in Figure 3.2. A straightforward inference is that the vulnerability of the basic mobile-IP scheme depends heavily on the amount of time required for registration to reach the HA during the handoff. If the HA is far away from the MH's current care-of address, the transport layer end-to-end performance of the basic mobile-IP scheme is worsen as its vulnerable period has been increased.

6.3 Effect of the change in vulnerable period

From chapter four, it has been concluded that the handoff enhanced scheme does not suffer from vulnerable periods. The next parameter to be considered is the effect of the duration of vulnerable period imposed on the other two schemes. The time taken for each mobile registration from MH_1 to the HA at net_1 is increased, since the data rate to net_1 has been reduced. This is analogous in placing the HA at a location which is further away from the MH.

The parameters were slightly changed from those in set 1. This is to ensure that change in performance is due primarily to the changed factor alone. Simulation was performed using parameters in Table 6.2. Simulations were performed for a 2000 second connection period. This was repeated using 10 different random seeds. The result is shown in Figure 6.8.

With the increase in vulnerable period by reducing the data rate between net_1 and Internet

Table 6.2 Parameter set 2

Parameter type:	
Mobile link data rate:	57.6 kbps
Data rate within local network:	10 Mbps
Data rate from net1 to Internet gateway:	196 kbps
Data rate from net2 and net3 to Internet gateway:	256 kbps
Maximum acknowledgment delay:	2 sec

Table 6.2 Parameter set 2

Parameter type:	
RTT gain:	0.125
RTT deviation gain:	0.25
RTT deviation constant:	4

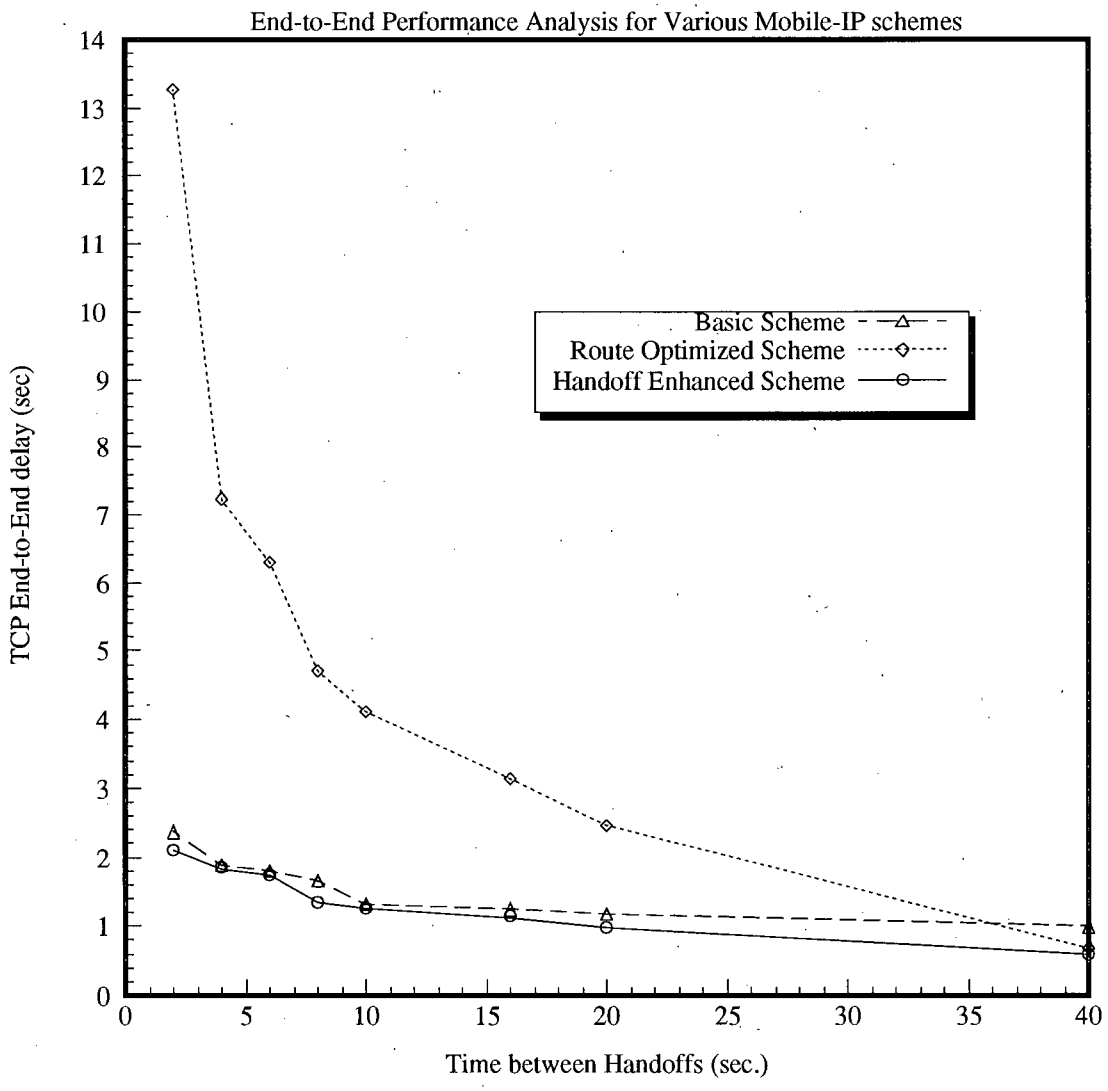


Figure 6.8 TCP performance for parameter set 2

core gateway, the IMHP scheme suffers even worse degradation than in parameter set 1. It fails sharply as soon as the time between handoff decreases. The IMHP scheme is therefore not a very

desirable scheme under handoff intensive situations.

The basic mobile-IP scheme demonstrates a rather flat response regarding to changes in handoff rate. The immunity to handoff rate is due to the lack of optimization. Datagrams have to be tunnelled by the HA through to the MH regardless of its current location. As handoffs become more frequent, the basic mobile-IP scheme begins to show weakness as the system is inevitably losing more datagrams due to incorrect routing. The end-to-end delay pattern stays relatively flat for handoff period from 10 to 40 seconds. Throughout this simulation, the handoff enhanced scheme performs the best among all three schemes. Although the data rate from net_1 and the Internet gateway has been decreased, this does not degrade the TCP end-to-end delay in the handoff enhanced scheme with the same extent as this scheme does not suffer from this increase in vulnerable period. Once again, the handoff enhancement scheme has shown immunity to vulnerable period which verifies the analysis in chapter four.

A third set of parameter was chosen for running the simulation. The parameters are shown in Table 6.3.

Table 6.3 Parameter set 3

Parameter type:	
Mobile link data rate:	57.6 kbps
Data rate within local network:	10 Mbps
Data rate from net_1 to Internet gateway:	128 kbps
Data rate from net2 and net3 to Internet gateway:	256 kbps
Maximum acknowledgment delay:	2 sec
RTT gain:	0.125
RTT deviation gain:	0.25
RTT deviation constant:	4

The simulation were repeated with 10 different random seeds. The result is shown in Figure 6.9. With the increased vulnerability of the basic scheme, the handoff enhanced scheme has shown improvement by a wider margin with a handoff period at the 2 second range. This is

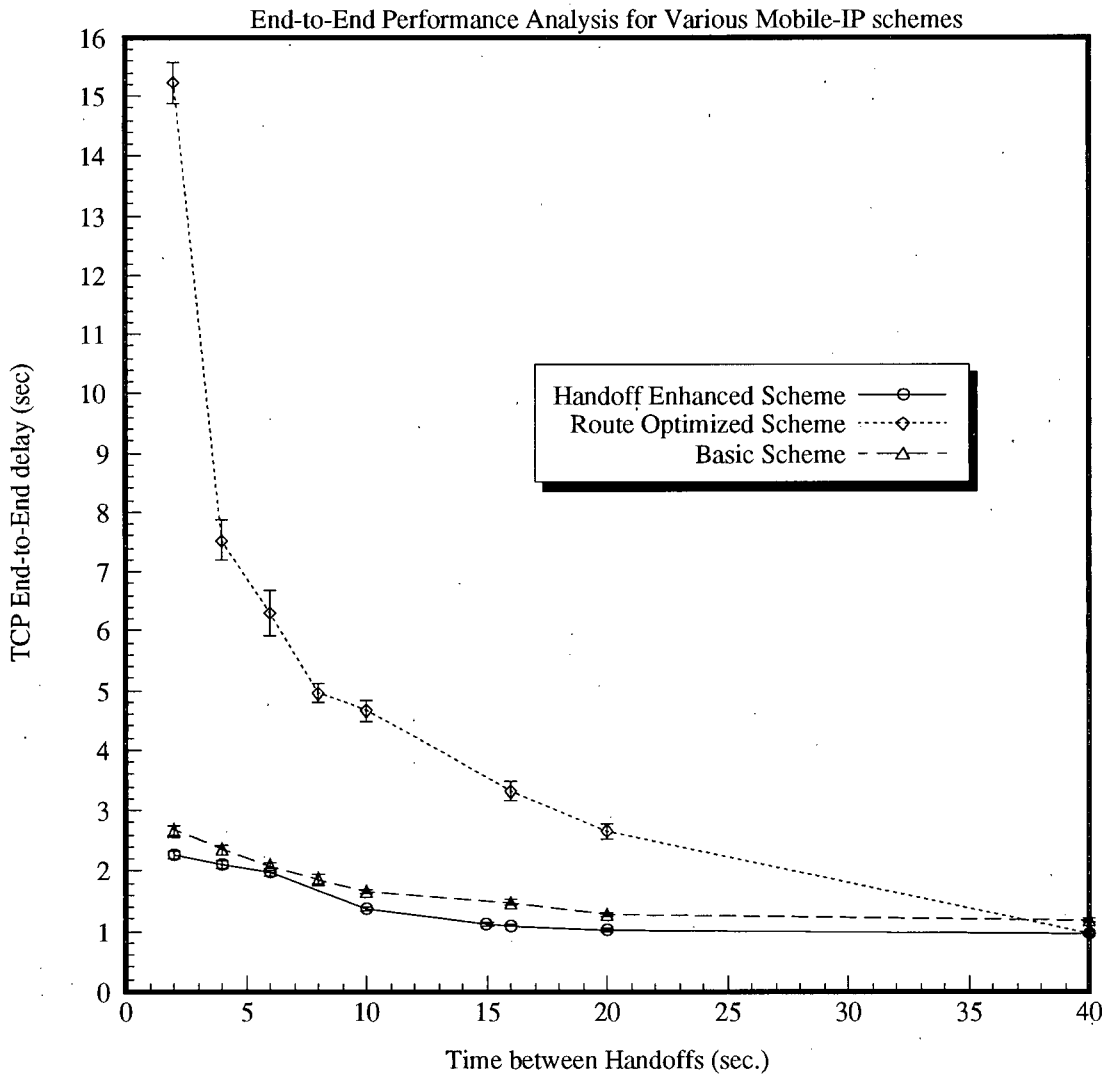


Figure 6.9 TCP performance for parameter set 3

due to that fact that the basic scheme is more vulnerable with the increased delay in handling mobile registrations. In addition, there is a clear margin of improvement throughout the entire handoff period range.

It has been shown that the registration delay has significant effect over the TCP end-to-end delay, the simulation model was modified to investigate the effect of further increasing propaga-

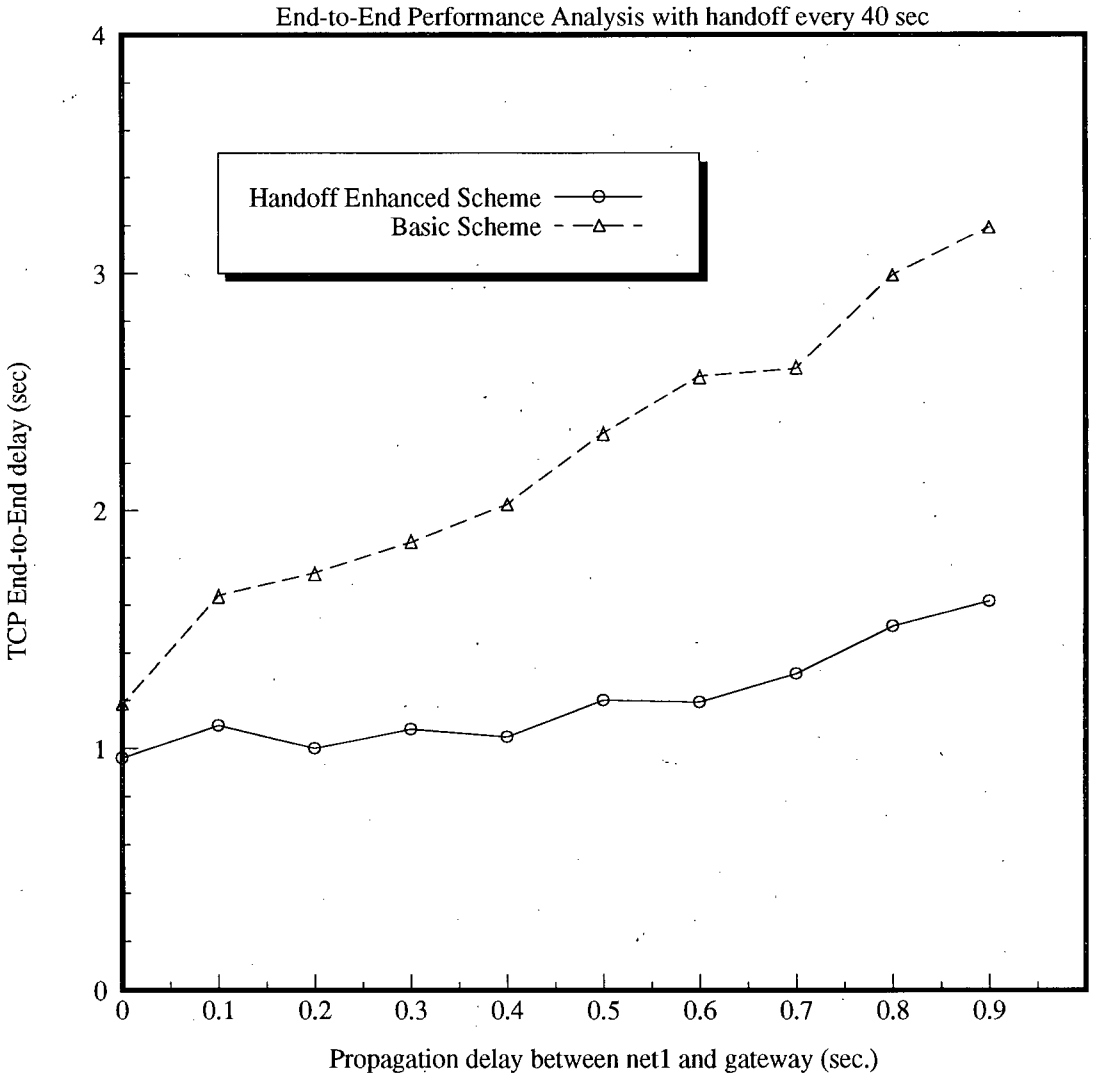


Figure 6.10 TCP performance with inserted propagation delay (handoff every 40 sec)

tion delay within the link from the HA at net_1 to the Internet core gateway. This additional propagation delay was varied between 0 to 0.9 sec. and the measurements were made over the TCP end-to-end delay for the connection over MH_1. Simulations were performed using the basic IETF scheme and the new handoff enhanced scheme. The parameters set 3 was chosen. The same

TCP connection was maintained at the MH_1 for a period of 2000 seconds. The results are shown

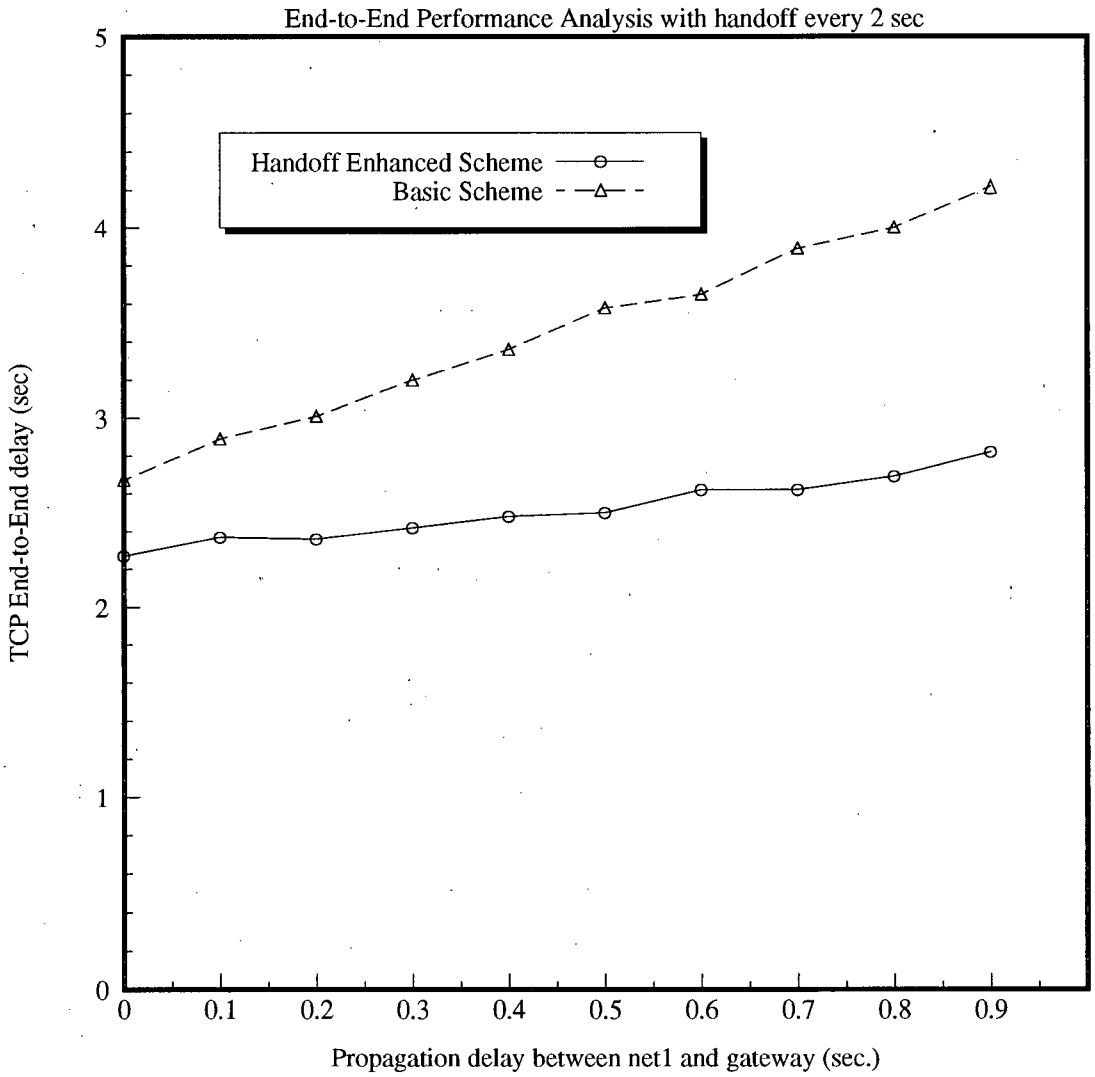


Figure 6.11 TCP performance with inserted propagation delay (handoff every 2 sec)

in Figure 6.10.

As the propagation delay between the HA at net_1 and the Internet core gateway increases, the probability of datagrams arriving during the vulnerable period increases. The increase in the measured TCP end-to-end delay is proportional to the increase in propagation delay. Again, this

agrees with the analysis in chapter three regarding the vulnerable period of the basic IETF scheme.

In the case of the new enhanced scheme, the same amount of propagation delay has been inserted into the model. It is shown that the new scheme has an increasing margin of improvement of TCP end-to-end delay over the basic scheme as propagation delay increases. This is due to the fact the new scheme does not suffer from a vulnerable period in which datagrams are lost.

The simulation was repeated with a higher handoff rate. In this case, the handoff period is changed to 2 seconds. The other parameters were kept the same as in the previous case. The result is shown in Figure 6.11. In this case of the basic IETF scheme, the TCP end-to-end delay is shown to have increased with the increase in the added propagation delay. This confirms that the TCP end-to-end performance of the basic IETF scheme is determined by the duration of vulnerable period. Once again, the new enhanced scheme has shown improvement in TCP end-to-end performance over the basic IETF scheme as the propagation delay increases.

6.4 Summary of Performance Comparison

The performance comparison for parameter set 2 is tabulated in Table 6.4. The TCP end-to-end delays are normalized to the basic mobile-IP scheme end-to-end delay.

The normalized TCP end-to-end delay for the IMHP scheme in parameter set 2 has only

Table 6.4 Performance comparison for parameter set 2

Time between handoffs (sec)	Normalized TCP end-to-end delay (%)	
	Route optimized scheme	Enhanced scheme
2	559	88
4	385	97

Table 6.4 Performance comparison for parameter set 2

Time between handoffs (sec)	Normalized TCP end-to-end delay (%)	
	Route optimized scheme	Enhanced scheme
6	352	96
10	283	80
16	312	96
20	208	84
40	82	80

Table 6.5 Performance comparison for parameter set 3

Time between handoffs (sec)	Normalized TCP end-to-end delay (%)	
	Route optimized scheme	Enhanced scheme
2	570	84
4	319	89
6	301	94
8	265	80
10	280	83
16	222	73
20	207	80
40	81	80

shown improvement over basic scheme when the handoff rate is very low. Whereas for the enhanced scheme, it shows consistent improvement over the basic scheme. With parameter set 3, the result is tabulated in Table 6.5. It is now evident that the handoff enhanced scheme is well suited for situations with high handoff rates. The IMHP scheme can only give better performance when the handoff rate is low. The improvement factor is about 12% for very low handoff situation. For the handoff enhanced scheme, it yields improvement of over 10% in most cases. The maximum gain from this scheme is 27%. Even at high handoff rate, the handoff enhanced scheme delivers performance improvement of over 12% for the system model and parameters considered.

With additional propagation delay at the link between the HA of net_1 and the Internet gateway, the results, for the system with handoffs every 40 seconds, are tabulated in Table 6.6. The TCP end-to-end delay is normalized against the basic IETF scheme. For the system setup and the parameters considered, improvements of up to 54% has been found in the new enhanced scheme.

Table 6.6 Normalized TCP end-to-end delay with handoff every 40 sec

Propagation delay (sec)	Normalized TCP end-to-end delay (%)
0.0	80
0.1	66
0.2	57
0.3	58
0.4	52
0.5	52
0.6	46
0.7	50
0.8	51
0.9	51

For the system with handoffs every 2 seconds and additional propagation delay for the link between net_1 and the Internet, the results are tabulated and shown in Table 6.7. The TCP end-to-end delay are normalized against the values for the basic IETF scheme.

The end-to-end performance of the new scheme is found to be better than the IETF

Table 6.7 Normalized TCP end-to-end delay with handoff every 2 sec

Propagation delay (sec)	Normalized TCP end-to-end delay (%)
0.0	85
0.1	82
0.2	78

Table 6.7 Normalized TCP end-to-end delay with handoff every 2 sec

Propagation delay (sec)	Normalized TCP end-to-end delay (%)
0.3	75
0.4	74
0.5	70
0.6	72
0.7	67
0.8	67
0.9	67

scheme. Improvements of up to 33% was found with the system setup considered.

From the results, it is shown that the IMHP scheme can deliver improved performance over the basic IETF scheme under light handoff situation. The reduction of triangular routing does bring along improvement in overall performance. However, as the handoff rate increases, the scheme suffers greatly and is deemed unstable.

It is evident that the improvements in TCP end-to-end delay performance has been achieved in most cases with the proposed handoff enhanced scheme. This scheme is proved to deliver better performance even when handoff intensity is high. Moreover, the scheme is able to withstand the change in mobile registration delay without compromising its performance, since it does not suffer from datagram losses during handoffs.

Chapter 7 Conclusion

In this study, a new handoff enhanced scheme was proposed. Together with two other different proposals of mobile-IP, these proposals were simulated in OPNET and the TCP end-to-end delays were measured by simulation.

The basic IETF mobile-IP scheme proposed was the basis for comparison, since it had been approved as an Internet standard. It was found that this basic scheme provided a stable system. However the performance could not be improved, since it does not employ any optimization. Consequently, the routing in this scheme is suboptimal, and hence, wasting network bandwidth which is already scarce in mobile links.

A route optimization (IMHP) scheme was analyzed to seek for improvement over the basic scheme. The scheme was found to be useful at low handoff rate. According to the simulation results, this new scheme degraded tremendously with a slight increase in handoff rate. Since datagrams were sent directly to the FAs, datagram loss became excessive during handoffs. As a result, the transport layer protocol had to devote a lot of bandwidth in synchronizing the connection. This scheme is only suitable for those MHs with infrequent handoffs.

The handoff enhanced scheme, being a variant of route optimization scheme, was shown by simulation that it could decrease the TCP end-to-end delay over the other two schemes. With the particular system model and parameters, it was found that the reduction in TCP end-to-end delay of up to 27% over the basic scheme could be achieved. When additional mobile registration delay had been introduced, it was found that the reduction in TCP end-to-end delay could be up to 54%.

Besides, the handoff enhanced scheme did not suffer from instability problems as found in the IMHP scheme. In most cases, the buffering method prevented datagram loss. This had reduced the TCP end-to-end delay by a significant amount, since datagram losses had detrimental effects over TCP connection. By preventing possible datagram loss, the transport layer protocol would not mistakenly throttle the traffic. By eliminating that source of datagram losses, the new handoff enhanced scheme is found to be well suited for situations with frequent handoffs. In addition, this new scheme waits for registration reply before relaying datagrams immediately. This would prevent the security loophole of replay message attack on the foreign agent by a malicious host. Thus the security of the system is not compromised.

Moreover, this scheme does not require any TCP layer modification to be made. As a result, the proposed handoff enhanced scheme for FAs can be easily applied. Hence, any TCP connections with MHs connected to these FAs can be improved.

Possible Furtherwork:

- As there are a lot of variations within Internet, another approach to test the scheme is to implement it over a relative simple network. Possible choices are Linux and FreeBSD, as the source code for these operating systems are publicly available and has been quite stable in performance. (Note that a PC with 80486[1] (66MHz) processor could out-perform a SPARCstation[2] IPX with the benchmark tests for Unix machines [29].) Starting from version 2.0 onwards, Linux kernel supports the use of *IP tunnelling*. Adapting the proposed scheme into these operating systems are made quite straightforward. Besides, for mobile computing to be feasible, the operating system has to be able to run on PC or laptop computers.
- Another interesting alternative would be to look at performance improvement with specialized code with the TCP layer to handle the handoff. This approach should give even more dramatic improvement. Indirect TCP scheme, for example, could be a good candidate for investigation..

---

1 80486 is a product developed by Intel Corporation, Inc.

2 SPARCstation is a registered trademark of SPARC International, Inc., licensed exclusively to Sun Microsystems, Inc.

Glossary

This section provides a list of acronyms used in this thesis.

BS	- Base Station
CA	- Cache Agent
CH	- Corresponding Host
FA	- Foreign Agent
HA	- Home Agent
ICI	- Internal Control Interface
IETF	- Internet Engineering Task Force
IMHP	- Internet Mobile Host Protocol
LAN	- Local Area Network
LSR	- Loose Source Routing
MA	- Mobile Agent (Home Agent and Foreign Agent)
MOBILE-IP	- Internet Protocol with mobility extension
RTT	- Round Trip Time
TCP	- Transport Control Protocol
UDP	- User Datagram Protocol
WMDN	- Wireless and Mobile Data Network

REFERENCES

[1] C. Perkins, IP Mobility Support, RFC 2002, Oct. 1996.

[2] A. Myles, D. B. Johnson, and C. E. Perkins, "A Mobile Host Protocol Supporting Route Optimization and Authentication," *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 839-849, June 1995.

[3] P. Manzoni, D. Ghosal, and G. Serazzi, "Impact of Mobility on TCP/IP: An Integrated Performance Study," *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 858-867, Jun. 1995.

[4] J. Postel, Transmission Control Protocol, RFC793, Sep. 1981.

[5] International Organization for Standardization, *Basic Reference Model for Open Systems Interconnection*, ISO 7498, 1984.

[6] C. Perkins, "Providing continuous network access to mobile hosts using TCP/IP," *Comput. Networks, ISDN Syst.*, vol. 26, pp. 357-369, 1993.

[7] D. B. Johnson, "Mobile host internetworking using IP loose source routing," School of Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, tech. rep. CMU-CS-93-128, Feb. 1993.

[8] F. Teraoka, Y. Yokote, and M. Tokoro, "A network architecture providing hosts migration transparency," in *Proc. ACM SIGCOMM '91*, Zurich, Switzerland, pp. 209-220, Sep. 1991.

[9] J. Ioannidis, D. Duchamp, and G. Q. Maquire, "IP based protocols for mobile internetworking," in *Proc. ACM SIGCOMM '91*, Zurich, Switzerland, pp. 235-245, Sep. 1991.

[10] H. Wada, T. Yozawa, T. Ohhishi, and Y. Tanaka, "Mobile computing environment based on Internet packet forwarding," in *Proc. Winter 1993 Usenix Conf.*, San Diego, CA, Jan. 1993.

[11] A. Myles and D. Skellern, "Comparison of mobile host protocols for IP," *Internetworking: Res., Experience*, vol. 4, no. 4, Dec. 1993.

[12] C. E. Perkins and P. Bhagwat, "A Mobile Networking System based on Internet Protocol," *IEEE Personal Communications*, pp. 32-41, First Quarter 1994.

[13] D. E. Comer, Internetworking with TCP/IP, vol. 1, Prentice-Hall, 1991.

[14] R. Droms, Dynamic Host Configuration Protocol, RFC 1541, Oct. 1993.

[15] J. Postel, Internet Control Message Protocol, RFC 792, Sep. 1981.

[16] J. Postel, User Datagram Protocol, RFC 768, Aug. 1980.

[17] Ronald L. Rivest, The MD5 Message-Digest Algorithm, RFC 1321, Apr. 1992.

[18] D. E. Eastlake, S. D. Crocker, and J. I. Schiller, Randomness Requirements for Security, RFC 1750, Dec. 1994.

[19] P. Karn, and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols," *Proceedings of ACM SIGCOMM '87*, pp. 2-7, 1987.

[20] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM '88*, Stanford, CA, pp. 314-329, Aug. 1988.

[21] M. Ulema, K. Khalil, "Access Traffic Characteristics of a Wide Area Network," *Proceedings of Global Data Networking*, pp. 146-153, Dec. 1993.

[22] J. Nagle, "Congestion Control in TCP/IP Internetworks," *ACM Communication Review*, pp. 11-17, Oct. 1984.

[23] S. Shenker, S. Zhang, and D. Clarck, "A Theoretical Analysis of Feedback Flow Control," *ACM Computer Communication Review*, pp. 156-165, Sep. 1990.

[24] D. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN System*, vol. 17, pp. 1-14, 1989.

[25] H. Zygmunt, "Adaptive Admission Congestion Control," ACM Computer Communications Review, vol. 21, no. 5, pp. 58-76, Oct. 1991.

[26] L. Huynh, R. Chang, and W. Chou, "Performance Comparison between TCP Slow-Start and a new Adaptive Rate-Based Congestion Avoidance Scheme," *MASCOTS '94: Modeling, Analysis, and Simulation Int'l Workshop*, pp. 300-307, 1994.

[27] K. Khalil, and Y. S. Sun, "Performance Considerations for TCP/IP in Wide Area Networks," *Local Computer Networks*, 19th Conference, pp. 166-175, 1994.

[28] R. Cáceres, and L. Iftode, "Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments," *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 850-857, Jun. 1995.

[29] R. Grehan, "Our new algorithm-based Native Mode suite tests processor performance and FPU capabilities for a variety of CPUs," *BYTE Magazine*, pp. 73-82, Mar. 1995.

Appendix A. Supplementary Source Code of the IETF Model

```
/* mobile_rte_sup.ex.c */
/* Routing support procedures for the Mobile IP example model */

#include <opnet.h>
#include "mobile-ip.h"
#include "mobile_rte_sup.h"
#include "ip_rte_sup.h"
#include "protocol.h"



/* Functions called by Process Module */

List*
mobile_rte_sup_table_setup (file_name)
    char                    *file_name;
    {
    List*                   mobile_strm_list_ptr;
    List*                   line_list_ptr;
    mobile_rte_table*table_ptr;


    /* Provides comprehensive routing table loading and parsing */
    /* services for Mobile IP module. */
    FIN (mobile_rte_sup_table_setup (file_name, net0, node0, net1, node1, net2, node2))

    /* Load the list of text lines from the specified file. */
    /* Note: this procedure will quit the simnulation if */
    /* file cannot be loaded, so it is assumed that there */
    /* are no problems upon returning. */
    line_list_ptr = mobile_rte_sup_table_load (file_name);

    /* Parse the contents of the obtained list into */
    /* a routing-instruction table. */
    mobile_strm_list_ptr = mobile_rte_sup_list_parse (line_list_ptr);

    /* In debug mode, if tracing is active, print the table */
    if (op_prg_odb_trace_active ())
            {
            mobile_rte_sup_table_print (mobile_strm_list_ptr);
            }

    FRET( mobile_strm_list_ptr );
    }

List*
mobile_rte_sup_table_load (file_name)
    char                    *file_name;
    {
    List*                   line_list_ptr;
    char                    err_str [256];

    /* Read in a routing table from an ascii */
    /* file adhering to format defined above. */
    FIN (mobile_rte_sup_table_load (file_name))

    /* Open and read the file into the list. */
    line_list_ptr = op_prg_gdf_read (file_name);
```

```
    /* Test for error in reading. */
    if (line_list_ptr == OPC_NIL)
        {
        sprintf (err_str, "File Name: %s", file_name);
        op_sim_end ("Package  : mobile_rte_sup",
                    "Error    : Unable to read routing table file",
                    err_str, "");
        }

    /* Return the list of text lines. */
    FRET (line_list_ptr)
    }


List*
mobile_rte_sup_list_parse (line_list_ptr)
    List*                       line_list_ptr;
    {
    List*                       mobile_stream_list_ptr;
    mobile_rte_table*mrt_ptr;
    int                         i, num_lines;
    char*                       line;
    List                        *field_list_ptr;


    /* Extract information from the lines of an ascii routing table */
    /*  and construct a corresponding routing table structure which */
    /* contains routing instructions. */
    FIN (mobile_rte_sup_list_parse (line_list_ptr))

    /* Allocate a routing table structure. */
    mrt_ptr = (mobile_rte_table*) op_prg_mem_alloc (sizeof (mobile_rte_table));

    /* Allocate a temporary table for holding lists. */
    mobile_stream_list_ptr = op_prg_list_create ();

    /* Scan through each of the lines, one at a time. */
    num_lines = op_prg_list_size (line_list_ptr);
    for (i = 0; i < num_lines; i++)
    {
        /* Obtain the i_th line. */
        line = op_prg_list_access (line_list_ptr, i);

        /* Decompose it into fields (field boundaries are */
        /* indicated by spaces, tabs,  slashes, or commas. */
        field_list_ptr = op_prg_str_decomp (line, " ,/\t");

        /* Format for a line is as follows: */
        /* <output_stream> <mtu>*/
        /* Incomplete lines are skipped. */
        if (op_prg_list_size (field_list_ptr) < 4)
            continue;

        /* Create a routing instruction structure. */
        mrt_ptr = (mobile_rte_table*)
            op_prg_mem_alloc (sizeof (mobile_rte_table));

        /* Transfer the parsed fields into the structure */
        /* First obtain the destination stream and mtu fields. */
        mrt_ptr->stream = atoi (
            op_prg_list_access (field_list_ptr, MOBILE_TBL_OUTSTREAM));
        mrt_ptr->mtu = atoi (
            op_prg_list_access (field_list_ptr, MOBILE_TBL_MTU));
```

```
            mrt_ptr->careof.net = atoi (
                    op_prg_list_access (field_list_ptr, MOBILE_TBL_CAREOF_NET));

            mrt_ptr->careof.node = atoi (
                    op_prg_list_access (field_list_ptr, MOBILE_TBL_CAREOF_NODE));

            if (mrt_ptr->mtu <= 0)
                    mrt_ptr->mtu = 0x7FFFFFFF;

            if (mrt_ptr->careof.net  == ADDRESS_UNDEFINED &&
                mrt_ptr->careof.node == ADDRESS_UNDEFINED )
            {
                    /* careof address not implemented */
                    mrt_ptr->condition = CONDITION_ENABLED;
            }
            else
            {
                    /* disable all condition flags initially */
                    mrt_ptr->condition = CONDITION_DISABLED;
            }

            /* Append the routing instruction to the temporary list. */
            op_prg_list_insert (mobile_stream_list_ptr, mrt_ptr, OPC_LISTPOS_TAIL);

    }

FRET( mobile_stream_list_ptr )
}


void
mobile_rte_sup_table_print (mobile_stream_list_ptr)
List*mobile_stream_list_ptr;
    {
    mobile_rte_table*table_entry;
    int                         i,size;
    char                        dne_str [128], dno_str [128];
    char                        nne_str [128], nno_str [128], str0 [512];

    /* Print the contents of a routing table. */
    FIN (mobile_rte_sup_table_print (mobile_stream_list_ptr))

    size = op_prg_list_size( mobile_stream_list_ptr );
    if ( size == 0 )
            {
            op_prg_odb_print_major ("Routing table is empty", VOS_NIL);
            }
    else{
            op_prg_odb_print_major ("Routing table contents :", VOS_NIL);
            for (i = 0; i < size; i++)
                    {
                    table_entry = (mobile_rte_table*)
                                  op_prg_list_access( mobile_stream_list_ptr, i );
                    sprintf( str0, "Stream (%d): mtu (%d)"
                            , table_entry->stream, table_entry->mtu );
                    op_prg_odb_print_minor (str0, VOS_NIL);
                    }
            }


    FOUT
    }


Compcode
```

```
    mobile_rte_sup_route_select ( route_table, mobile_table, pkptr, ici_ptr
                                , mobile_agnt_flag, pk_id, ttl
                                , HA_bind_ptr, FA_bind_ptr, MH_bind_ptr
    /*                          , home_net0, home_net1, home_net2 )*/
                                , ip0, ip1, ip2 )
    ip_rte_table        *route_table;
    List                        *mobile_table;
    Packet                      *pkptr;
    Ici                         *ici_ptr;
    int                         mobile_agnt_flag;
    int                         *pk_id;
    int                         ttl;
    List                        *HA_bind_ptr, *FA_bind_ptr, *MH_bind_ptr;
    IP                          *ip0, *ip1, *ip2;
    /*int                       home_net0, home_net1, home_net2;*/
    {
    int                         i, j, num_bind, num_multi_bind;
    IP                          dest;
    HA_mobility_binding*home_entry;
    FA_mobility_binding*visitor_entry;
    multi_binding       *multi_bind_entry;
    Compcode            status;

        FIN (mobile_rte_sup_route_select( route_table, mobile_table, pkptr, ...))

        op_pk_nfd_get( pkptr, "dest_net", &dest.net );
        op_pk_nfd_get( pkptr, "dest_node", &dest.node );

        /* Select a route from the routing table which matches the */
        /* requested destination network and node. */
        if ( !memcmp( &dest, ip0, sizeof(IP) ) ||
             !memcmp( &dest, ip1, sizeof(IP) ) ||
             !memcmp( &dest, ip2, sizeof(IP) ) )
                FRET ( OPC_COMPCODE_FAILURE );

        if ( send_via_FA( pkptr, MH_bind_ptr, mobile_table, ici_ptr, dest, mobile_agnt_flag )
                        == OPC_COMPCODE_SUCCESS )
                FRET ( OPC_COMPCODE_SUCCESS )
        else if ( forward_to_visitor( mobile_table, FA_bind_ptr, pkptr, ici_ptr, dest ) ==
OPC_COMPCODE_SUCCESS )
                FRET ( OPC_COMPCODE_SUCCESS )
        else if ( encap_packet( route_table, HA_bind_ptr, pkptr, ici_ptr, ip0, dest, pk_id, ttl )
== OPC_COMPCODE_SUCCESS )
                FRET ( OPC_COMPCODE_SUCCESS )
        else
                FRET ( OPC_COMPCODE_FAILURE )

    }



    Compcode
    send_via_FA( pkptr, bind_list_ptr, mobile_table, ici_ptr, dest, agnt_flag )
    List                        *bind_list_ptr, *mobile_table;
    Ici                         *ici_ptr;
    Packet                      *pkptr;
    IP                          dest;
    int                         agnt_flag;
    {
    int                         i, size;
    int                         copy = false;
    int                         next_net, next_node, stream, mtu;
    Packet                      *cp_pkptr;
    Compcode            status= OPC_COMPCODE_SUCCESS;
    MH_FA_binding       *fa_entry;
```

```
mobile_rte_table*table_entry;
char                    str0[80], str1[80];

FIN( send_via_FA( pkptr, bind_list_ptr, mobile_table, ici_ptr, dest ) )

    /* First of all, check if packet is for any enabled streams */
    size = op_prg_list_size( mobile_table );
    for ( i=0; i<size; ++i )
    {
           table_entry = (mobile_rte_table *) op_prg_list_access( mobile_table, i);

           if ( IP_equal( table_entry->careof, dest ) &&
                table_entry->condition == CONDITION_ENABLED )
           {
                  stream = table_entry->stream;
                  mtu    = table_entry->mtu;

                  next_net  = ADDRESS_UNDEFINED;
                  next_node = ADDRESS_UNDEFINED;

                  deliver_packet( pkptr, ici_ptr
                         , next_net, next_node, stream, mtu );

                  FRET( OPC_COMPCODE_SUCCESS )
           }
    }


    /* Now, check to see if there is any mobility binding */
    size = op_prg_list_size( bind_list_ptr );

    if ( size == 0 && agnt_flag )/* no bindings and not a mobile node */
           FRET( OPC_COMPCODE_FAILURE )

    /* each enabled FA will receive a copy of the packet */

    for( i=0; i<size; ++i )
    {
           fa_entry = (MH_FA_binding *)
                         op_prg_list_access( bind_list_ptr, i );

           /* obtain the stream associated with the current binding */
           stream = fa_entry->stream;

           if ( chk_strm_condition( mobile_table, stream, &mtu ) == CONDITION_ENABLED )
           {
/*                mtu = get_strm_mtu( mobile_table, stream );*/
                  if ( mtu == MOBILE_STRM_NONEXISTENT )
                  {
                         sprintf (str0, "Discarding packet (%d) ", op_pk_id (pkptr));
                         sprintf (str1, "Stream non-existent" );
                         op_prg_odb_print_major (str0, str1, OPC_NIL);
                         op_pk_destroy( pkptr );
                  }
                  else
                  {
                         next_net = fa_entry->careof.net;
                         next_node = fa_entry->careof.node;
#if 1
                         deliver_packet( pkptr, ici_ptr
                                       , next_net, next_node, stream, mtu );
                         FRET( OPC_COMPCODE_SUCCESS )
#else
                         copy = true;
                         cp_pkptr = op_pk_copy( pkptr );
```

```
                        deliver_packet( cp_pkptr, ici_ptr
                                        , next_net, next_node, stream, mtu );
#endif
                }
            }
    }

    /* have to deallocate original packet since it is no longer needed */
    if ( copy )
    {
            op_pk_destroy( pkptr );
            FRET( status )
    }

FRET( OPC_COMPCODE_FAILURE )
}


Compcode
encap_packet( route_table, bind_list_ptr, pkptr, ici_ptr, current, dest, pk_id, ttl )
ip_rte_table        *route_table;
List                *bind_list_ptr;
Packet              *pkptr;
Ici                 *ici_ptr;
IP                  current, dest;
int                 *pk_id;
int                 ttl;
{
char        str0[512], str1[512];
Packet              *encap_pkptr;
Packet              *warn_pkptr;
Packet              *warn_ipptr;
int                 i, j, num_bind, num_multi_bind;
int                 next_net, next_node, outstrm, mtu;
IP                  orig;
int                 data_len;
int                 copy = false;
HA_mobility_binding*home_entry;
multi_binding       *multi_bind_entry;
Compcode            status = OPC_COMPCODE_FAILURE;

FIN( encap_packet( route_table, ...) )

    num_bind = op_prg_list_size( bind_list_ptr );

    if ( num_bind == 0 )
            FRET( OPC_COMPCODE_FAILURE );

    for (i=0; i<num_bind; ++i)
    {
            home_entry = (HA_mobility_binding *)
                    op_prg_list_access( bind_list_ptr, i );

            /* check for matches in HA mobility binding */
            if (!memcmp( &dest,&home_entry->home_addr, sizeof(IP)))
            {
                    status = OPC_COMPCODE_SUCCESS;
                    num_multi_bind = op_prg_list_size( home_entry->multi_bind_list );
                    for( j=0; j<num_multi_bind; ++j )
                    {
                            multi_bind_entry = (multi_binding *)
                                    op_prg_list_access( home_entry->multi_bind_list, j );
/*                          encap_pkptr = op_pk_copy( pkptr );*/

                            data_len = op_pk_total_size_get(pkptr)/8;
```

```
                encap_pkptr = op_pk_create_fmt( "ip_dgram" );
                op_pk_bulk_size_set( encap_pkptr, data_len*8 );

                copy = true;
                op_pk_nfd_set( encap_pkptr, "data"
                             , op_pk_copy(pkptr) );
                op_pk_nfd_set( encap_pkptr, "protocol", PROTOCOL_ENCAP );
                op_pk_nfd_set( encap_pkptr, "src_net",
                             current.net );
                op_pk_nfd_set( encap_pkptr, "src_node",
                             current.node );
                op_pk_nfd_set( encap_pkptr, "dest_net",
                             multi_bind_entry->careof.net );
                op_pk_nfd_set( encap_pkptr, "dest_node",
                             multi_bind_entry->careof.node );

                op_pk_nfd_set( encap_pkptr,"orig_len",data_len);
                op_pk_nfd_set( encap_pkptr,"frag_len",data_len);

                op_pk_nfd_set( encap_pkptr,"ident", (*pk_id)++);
                op_pk_nfd_set( encap_pkptr,"frag", 0 );
                op_pk_nfd_set( encap_pkptr, "ttl", ttl);

                /* Determine the output stream on which to route it. */
                if (ip_rte_sup_route_select (route_table
                        , multi_bind_entry->careof.net
                        , multi_bind_entry->careof.node
                        , &next_net, &next_node, &outstrm, &mtu)
                                        == OPC_COMPCODE_FAILURE)
                {
                        /* If no route is provided, destroy the packet. */
                        if (op_prg_odb_ltrace_active ("ip_errs"))
                        {
                                sprintf (str0, "Discarding unroutable packet (%d)"
                                             , op_pk_id (encap_pkptr));
                                sprintf (str1, "Destination: net (%d), node (%d)"
                                             , multi_bind_entry->careof.net
                                             , multi_bind_entry->careof.node);
                                op_prg_odb_print_major (str0, str1, OPC_NIL);
                        }
                        op_pk_destroy (encap_pkptr);
                }
                else
                {
                        deliver_packet(encap_pkptr, ici_ptr
                                     , next_net, next_node
                                     , outstrm, mtu );
                }
        }

#if 0
        /* At this stage, it is assumed that binding exists
             * for mobile node. Therefore send binding warning
         * message to original sender */
        if ( route_optim == true )
        {
                warn_pkptr = op_pk_create_fmt( "bind_warn" );
                op_pk_nfd_set( warn_pkptr, "home_addr_net"
                             , home_entry->home_addr.net );
                op_pk_nfd_set( warn_pkptr, "home_addr_node"
                             , home_entry->home_addr.node );

                data_len = op_pk_total_size_get(warn_pkptr)/8;
                warn_ipptr = op_pk_create_fmt( "ip_dgram" );
                op_pk_nfd_set( warn_ipptr, "data", warn_pkptr);
```

```
                                op_pk_bulk_size_set(warn_ipptr,data_len*8);

                                op_pk_nfd_set( warn_ipptr, "src_net", current.net );
                                op_pk_nfd_set( warn_ipptr, "src_node", current.node );
                                op_pk_nfd_get( pkptr, "src_net", &orig.net );
                                op_pk_nfd_get( pkptr, "src_node", &orig.node );
                                op_pk_nfd_set( warn_ipptr, "dest_net", orig.net );
                                op_pk_nfd_set( warn_ipptr, "dest_node",orig.node );
                                op_pk_nfd_set( warn_ipptr,"orig_len",data_len);
                                op_pk_nfd_set( warn_ipptr,"frag_len",data_len);
                                op_pk_nfd_set( warn_ipptr,"frag", 0 );
                                op_pk_nfd_set( warn_ipptr,"ident", (*pk_id)++);
                                op_pk_nfd_set( warn_ipptr,"protocol", PROTOCOL_UDP );
                                deliver_packet( warn_ipptr,
                        }
    #endif
                }
        }

        if ( copy )
                op_pk_destroy( pkptr );

    FRET( status )
    }


    void
    deliver_packet( pkptr, ici_ptr, next_net, next_node, outstrm, mtu )
    Packet                      *pkptr;
    Ici                         *ici_ptr;
    int                         next_net, next_node, outstrm, mtu;
    {
    char        str0[512], str1[512];
    int         i, len;
    int         header_size, frag_size, data_size;
    int         dest_net, dest_node;
    int         ttl;
    int         frag_accum, frag, num_frags;
    Packet      *ip_pkptr, *data_pkptr, *frag_ptr;

    FIN( deliver_packet( pkptr, ... ) )

        /* obtain packet's destination */
        op_pk_nfd_get( pkptr, "dest_net", &dest_net );
        op_pk_nfd_get( pkptr, "dest_node", &dest_node );

        /* Decrement the packet's time-to-live field. If zero is reached, */
        /* discard the packet rather than send it on. */
        op_pk_nfd_get (pkptr, "ttl", &ttl);
        ttl--;
        if (ttl == 0)
        {
                /* In debug mode, indicate that a packet is destroyed */
                /* due to an expired ttl. */
                if (op_prg_odb_ltrace_active ("ip_errs"))
                {
                        sprintf (str0, "Discarding packet (%d) with expired TTL", op_pk_id (pkp-
tr));
                        sprintf (str1, "Destination: net (%d), node (%d)", dest_net, dest_node);
                        op_prg_odb_print_major (str0, str1, OPC_NIL);
                }
                op_pk_destroy (pkptr);
        }
        else{
                /* Assign the new decremented value of ttl. */
```

```
op_pk_nfd_set (pkptr, "ttl", ttl);

/* In debug mode, trace the routing action. */
if (op_prg_odb_ltrace_active ("mobile-ip_rte"))
{
        sprintf (str0, "Routing towards (%d, %d)", dest_net, dest_node);
        sprintf (str1, "Next hop (%d, %d), output stream (%d)",
                                next_net, next_node, outstrm);

        op_prg_odb_print_major (str0, str1, OPC_NIL);
}

/* Install an Ici indicating to the lower layer what the */
/* address of the next (intermediate) node is. */
op_ici_attr_set (ici_ptr, "next_node", next_node);
op_ici_install (ici_ptr);

/* Obtain the size in bytes of the fragment. */
frag_size = op_pk_total_size_get (pkptr) / 8;

/* Obtain the number of bytes of data carried in this fragment */
op_pk_nfd_get (pkptr, "frag_len", &data_size);

/* Also obtain the difference between the packet size */
/* and the length field: this is the size of the header. */
header_size = frag_size - data_size;

/* If it is smaller than the maximum transfer unit, send it as is. */
if (frag_size <= mtu)
{
        op_pk_send (pkptr, outstrm);
}
else{
        /* Otherwise, break it into fragments */
        /* Each fragment can contain up to (mtu - header_size) bytes of data */
        num_frags = (data_size + mtu - header_size - 1) / (mtu - header_size);

        /* In debug mode, indicate the fragmentation. */
        if (op_prg_odb_ltrace_active ("ip_frag"))
        {
                sprintf (str0, "Breaking datagram into (%d) fragments", num_frags);
                op_prg_odb_print_major (str0, OPC_NIL);
        }

        /* If the fragment is carrying the original datagram given to IP, */
        /* extract it before copies are made. Only one fragment can carry */
        /* the original packet for the reassembly model to work properly. */
        if (op_pk_nfd_is_set (pkptr, "ip_dgram"))
                op_pk_nfd_get (pkptr, "ip_dgram", &ip_pkptr);
        else ip_pkptr = OPC_NIL;

        /* If the packet is carrying any encapsulated data (normally this */
        /* would happen only for a packet fragmented for the first time), */
        /* extract this data packet so that it will not appear in each */
        /* fragment generated by copying. */
        if (op_pk_nfd_is_set (pkptr, "data"))
                op_pk_nfd_get (pkptr, "data", &data_pkptr);
        else data_pkptr = OPC_NIL;

        /* Loop through and create the fragments . */
        for (frag_accum = 0, i = 0; i < num_frags; i++)
        {
                /* Make a copy of the original packet. */
                frag_ptr = op_pk_copy (pkptr);
```

```
                        /* Indicate that the copy is a fragment */
                        op_pk_nfd_set (frag_ptr, "frag", 1);

                        /* for all but the last fragment, the size is the mtu. */
                        /* and the encapsulated ip packet is not included. */
                        if (i < num_frags - 1)
                        {
                              op_pk_nfd_set (frag_ptr, "frag_len", mtu - header_size);
                              op_pk_total_size_set (frag_ptr, 8 * mtu);
                              frag_accum += (mtu - header_size);
                        }
                        else{
                              len = data_size - frag_accum;
                              op_pk_nfd_set (frag_ptr, "frag_len", len);
                              op_pk_total_size_set (frag_ptr,8 * (header_size + len));

                              /* If the original packet was not a fragment, encapsulate it
*/
                              /* into the last fragment created here. */
                              op_pk_nfd_get (pkptr, "frag", &frag);
                              if (!frag)
                              {
                                    /* If the packet contained encapsulated data (i.e.,
from the */
                                    /* transport), that data will have been removed to
avoid */
                                    /* its duplication in the fragments. The data should
now be */
                                    /* reinserted into the original packet. */
                                    if (data_pkptr != OPC_NIL)
                                          op_pk_nfd_set (pkptr, "data", data_pkptr);

                                    /* In either case the original packet is */
                                    /* encapsulated in thhe fragment. */
                                    op_pk_nfd_set (frag_ptr, "ip_dgram", pkptr);
                              }

                              /* Otherwise the packet can be discarded. */
                              else{
                                    op_pk_destroy (pkptr);

                                    /* Also, if the packet included the original datagram
*/
                                    /* from which it was generated, transfer that data-
gram */
                                    /* into the last fragment created here. */
                                    /* Note that it is possible, in the case where a
fragment */
                                    /* is itself being fragmented, that none of the cre-
ated */
                                    /* fragments will contain the original datagram. */
                                    if (ip_pkptr != OPC_NIL)
                                    {
                                          op_pk_nfd_set (frag_ptr, "ip_dgram",
ip_pkptr);
                                    }
                              }
                        }

                        /* Forward the datagram fragment. */
                        op_pk_send (frag_ptr, outstrm);
                  }
            }
      }
   FOUT;
```

```
}



Compcode
forward_to_visitor( mobile_table, bind_list_ptr, pkptr, ici_ptr, dest )
List                *bind_list_ptr, *mobile_table;
Packet              *pkptr;
Ici                 *ici_ptr;
IP                  dest;
{
char                str0[80], str1[80];
Compcode    status = OPC_COMPCODE_FAILURE;
int                 i, size;
int                 next_net, next_node, stream, mtu;
int                 copy = false;
FA_mobility_binding*visitor_entry;
Packet              *cp_pkptr;

FIN( forward_to_visitor( mobile_table, bind_list_ptr, pkptr, ici_ptr, dest ) )

    size = op_prg_list_size( bind_list_ptr );

    if ( size == 0 )/* list does not exist */
            FRET ( OPC_COMPCODE_FAILURE )

    for( i=0; i<size; ++i )
    {
            visitor_entry = (FA_mobility_binding *)
                            op_prg_list_access( bind_list_ptr, i );

            if ( !memcmp( &dest, &visitor_entry->home_addr, sizeof(IP) ) )
            {
                    /* There is a match */
                    if ( visitor_entry->home_agnt.net != ADDRESS_UNDEFINED &&
                         visitor_entry->home_agnt.node != ADDRESS_UNDEFINED )
                    {
                            status = OPC_COMPCODE_SUCCESS;
                            next_net = visitor_entry->home_addr.net;
                            next_node = visitor_entry->home_addr.node;
                            stream = visitor_entry->stream;
                            mtu = get_strm_mtu( mobile_table, stream );
                            if ( mtu == MOBILE_STRM_NONEXISTENT )
                            {
                                    sprintf (str0, "Discarding packet (%d) ", op_pk_id (pkptr));
                                    sprintf (str1, "Stream non-existent" );
                                    op_prg_odb_print_major (str0, str1, OPC_NIL);
                                    op_pk_destroy( pkptr );
                            }
                            else
                            {
                                    copy = true;
                                    cp_pkptr = op_pk_copy( pkptr );
                                    deliver_packet( cp_pkptr, ici_ptr
                                                    , next_net, next_node, stream, mtu );
                            }
                    }
            }
    }

    if ( copy )
            op_pk_destroy( pkptr );
FRET( status );
}
```

```
int
get_strm_mtu( strm_ptr, strm )
List*strm_ptr;
int strm;
{
int                i, size;
mobile_rte_table*strm_entry;

FIN( get_strm_mtu( strm_ptr, strm ) )

    size = op_prg_list_size( strm_ptr );

    for( i=0; i<size; ++i )
    {
           strm_entry = (mobile_rte_table *) op_prg_list_access( strm_ptr, i );
           if ( strm_entry->stream == strm )
                  FRET( strm_entry->mtu );
    }

FRET( MOBILE_STRM_NONEXISTENT );
}



int
chk_strm_condition( mobile_table, stream, mtu )
List       *mobile_table;
int        stream;
int        *mtu;
{
int                i, size;
mobile_rte_table*table_entry;
FIN( chk_strm_condition( mobile_table, stream, mtu ) )

    size = op_prg_list_size( mobile_table );

    for( i=0; i<size; ++i )
    {
           table_entry = ( mobile_rte_table * )
                          op_prg_list_access( mobile_table, i );

           if( table_entry->stream == stream )
           {
                  *mtu = table_entry->mtu;
                  FRET( table_entry->condition )
           }
    }


    /* stream does not exist */
    *mtu = MOBILE_STRM_NONEXISTENT;

FRET( CONDITION_DISABLED )
}

void
get_careof_addr( mobile_table, iciptr )
List       *mobile_table;
Ici        *iciptr;
{
int                i, size;
int                stream;
IP                 careof;
```

```
mobile_rte_table*table_entry;
FIN( get_careof_addr( mobile_table, iciptr ) )

    op_ici_attr_get( iciptr, "stream", &stream );

    size = op_prg_list_size( mobile_table );
    for( i=0; i<size; ++i )
    {
        table_entry = ( mobile_rte_table * )
                      op_prg_list_access( mobile_table, i );

        if( table_entry->stream == stream )
        {
            op_ici_attr_set( iciptr, "careof_net", table_entry->careof.net );
            op_ici_attr_set( iciptr, "careof_node", table_entry->careof.node );
            op_ici_attr_set( iciptr, "status", OPC_COMPCODE_SUCCESS );
            FOUT
        }
    }


    /* stream does not exist */
    op_ici_attr_set( iciptr, "careof_net", ADDRESS_UNDEFINED );
    op_ici_attr_set( iciptr, "careof_node", ADDRESS_UNDEFINED );
    op_ici_attr_set( iciptr, "status", OPC_COMPCODE_FAILURE );

FOUT
}


void
set_strm_condition( mobile_table, iciptr )
List*mobile_table;
Ici *iciptr;
{
IP          careof;
int         stream;
int         mode;
int         i, size;
Compcodestatus = OPC_COMPCODE_FAILURE;
mobile_rte_table*list_ptr;
FIN( set_strm_condition( iciptr ) )

    op_ici_attr_get( iciptr, "mode", &mode );
    op_ici_attr_get( iciptr, "stream", &stream );

    size = op_prg_list_size( mobile_table );
    for( i=0; i<size; ++i )
    {
        list_ptr = (mobile_rte_table *)
                   op_prg_list_access( mobile_table, i );

        switch( mode )
        {
        case DISABLE_ALL:
            list_ptr->condition = CONDITION_DISABLED;
            status = OPC_COMPCODE_SUCCESS;
            break;

        case DISABLE_ALL_EXCEPT_THIS:
            if( list_ptr->stream != stream )
            {
                list_ptr->condition = CONDITION_DISABLED;
                status = OPC_COMPCODE_SUCCESS;
            }
```

```
                break;

        case DISABLE_THIS_ONLY:
                if( list_ptr->stream == stream )
                {
                        op_ici_attr_set( iciptr, "careof_net", list_ptr->careof.net );
                        op_ici_attr_set( iciptr, "careof_node", list_ptr->careof.node );
                        list_ptr->condition = CONDITION_DISABLED;
                        status = OPC_COMPCODE_SUCCESS;
                }
                break;

        case ENABLE_ALL:
                list_ptr->condition = CONDITION_ENABLED;
                status = OPC_COMPCODE_SUCCESS;
                break;

        case ENABLE_ALL_EXCEPT_THIS:
                if( list_ptr->stream != stream )
                {
                        list_ptr->condition = CONDITION_ENABLED;
                        status = OPC_COMPCODE_SUCCESS;
                }
                break;

        case ENABLE_THIS_ONLY:
                if( list_ptr->stream == stream )
                {
                        op_ici_attr_set( iciptr, "careof_net", list_ptr->careof.net );
                        op_ici_attr_set( iciptr, "careof_node", list_ptr->careof.node );
                        list_ptr->condition = CONDITION_ENABLED;
                        status = OPC_COMPCODE_SUCCESS;
                }
                break;

        default:
                status = OPC_COMPCODE_FAILURE;
                break;
        }
    }

    op_ici_attr_set( iciptr, "status", status );
FOUT
}

void
hop_to_next_strm( mobile_table, iciptr )
List        *mobile_table;
Ici         *iciptr;
{
int             j, i, size;
int             stream;
mobile_rte_table*table_entry;
FIN( hop_to_next_strm( mobile_table, iciptr ) )

    op_ici_attr_get( iciptr, "stream", &stream );

    size = op_prg_list_size( mobile_table );

    for ( i=0; i<size; ++i )
    {
        table_entry = ( mobile_rte_table * )
                        op_prg_list_access( mobile_table, i );

        if ( table_entry->stream == stream )
```

```
            {
                    if ( ( j = i + 1 ) == size )
                            j = 0;

                    table_entry = ( mobile_rte_table * )
                            op_prg_list_access( mobile_table, j );

                    op_ici_attr_set( iciptr, "stream"
                                    , table_entry->stream );

                    op_ici_attr_set( iciptr, "careof_net"
                                    ,table_entry->careof.net);

                    op_ici_attr_set( iciptr, "careof_node"
                                    ,table_entry->careof.node);

                    /* Now, enabling this stream */
                    table_entry->condition = CONDITION_ENABLED;

                    op_ici_attr_set( iciptr, "status"
                                    , OPC_COMPCODE_SUCCESS );

                    FOUT
            }
    }
    op_ici_attr_set( iciptr, "status", OPC_COMPCODE_FAILURE );
FOUT
}
```

Appendix B. Supplementary Source Code of the IMHP Model

```
/* mobile_rte_sup.ex.c */
/* Routing support procedures for the Mobile IP example model */

#include <opnet.h>
#include "mobile-ip.h"
#include "mobile_rte_sup.h"
#include "ip_rte_sup.h"
#include "protocol.h"


/* Functions called by Process Module */

List*
mobile_rte_sup_table_setup (file_name)
    char                    *file_name;
    {
    List*                   mobile_strm_list_ptr;
    List*                   line_list_ptr;
    mobile_rte_table*table_ptr;

    /* Provides comprehensive routing table loading and parsing */
    /* services for Mobile IP module. */
    FIN (mobile_rte_sup_table_setup (file_name, net0, node0, net1, node1, net2, node2))

    /* Load the list of text lines from the specified file. */
    /* Note: this procedure will quit the simnulation if */
    /* file cannot be loaded, so it is assumed that there */
    /* are no problems upon returning. */
    line_list_ptr = mobile_rte_sup_table_load (file_name);

    /* Parse the contents of the obtained list into */
    /* a routing-instruction table. */
    mobile_strm_list_ptr = mobile_rte_sup_list_parse (line_list_ptr);

    /* In debug mode, if tracing is active, print the table */
    if (op_prg_odb_trace_active ())
            {
            mobile_rte_sup_table_print (mobile_strm_list_ptr);
            }

    FRET( mobile_strm_list_ptr );
    }

List*
mobile_rte_sup_table_load (file_name)
    char                    *file_name;
    {
    List*                   line_list_ptr;
    char                    err_str [256];

    /* Read in a routing table from an ascii */
    /* file adhering to format defined above. */
    FIN (mobile_rte_sup_table_load (file_name))

    /* Open and read the file into the list. */
    line_list_ptr = op_prg_gdf_read (file_name);

    /* Test for error in reading. */
```

```
    if (line_list_ptr == OPC_NIL)
            {
            sprintf (err_str, "File Name: %s", file_name);
            op_sim_end ("Package  : mobile_rte_sup",
                        "Error    : Unable to read routing table file",
                        err_str, "");
            }

    /* Return the list of text lines. */
    FRET (line_list_ptr)
    }


List*
mobile_rte_sup_list_parse (line_list_ptr)
    List*                       line_list_ptr;
    {
    List*                       mobile_stream_list_ptr;
    mobile_rte_table*mrt_ptr;
    int                         i, num_lines;
    char*                       line;
    List                        *field_list_ptr;


    /* Extract information from the lines of an ascii routing table */
    /*  and construct a corresponding routing table structure which */
    /* contains routing instructions. */
    FIN (mobile_rte_sup_list_parse (line_list_ptr))

    /* Allocate a routing table structure. */
    mrt_ptr = (mobile_rte_table*) op_prg_mem_alloc (sizeof (mobile_rte_table));

    /* Allocate a temporary table for holding lists. */
    mobile_stream_list_ptr = op_prg_list_create ();

    /* Scan through each of the lines, one at a time. */
    num_lines = op_prg_list_size (line_list_ptr);
    for (i = 0; i < num_lines; i++)
    {
            /* Obtain the i_th line. */
            line = op_prg_list_access (line_list_ptr, i);

            /* Decompose it into fields (field boundaries are */
            /* indicated by spaces, tabs,  slashes, or commas. */
            field_list_ptr = op_prg_str_decomp (line, " ,/\t");

            /* Format for a line is as follows: */
            /* <output_stream> <mtu>*/
            /* Incomplete lines are skipped. */
            if (op_prg_list_size (field_list_ptr) < 4)
                    continue;

            /* Create a routing instruction structure. */
            mrt_ptr = (mobile_rte_table*)
                    op_prg_mem_alloc (sizeof (mobile_rte_table));

            /* Transfer the parsed fields into the structure */
            /* First obtain the destination stream and mtu fields. */
            mrt_ptr->stream = atoi (
                    op_prg_list_access (field_list_ptr, MOBILE_TBL_OUTSTREAM));
            mrt_ptr->mtu = atoi (
                    op_prg_list_access (field_list_ptr, MOBILE_TBL_MTU));

            mrt_ptr->careof.net = atoi (
                    op_prg_list_access (field_list_ptr, MOBILE_TBL_CAREOF_NET));
```

```
            mrt_ptr->careof.node = atoi (
                    op_prg_list_access (field_list_ptr, MOBILE_TBL_CAREOF_NODE));

            if (mrt_ptr->mtu <= 0)
                    mrt_ptr->mtu = 0x7FFFFFFF;

            if (mrt_ptr->careof.net  == ADDRESS_UNDEFINED &&
                mrt_ptr->careof.node == ADDRESS_UNDEFINED )
            {
                    /* careof address not implemented */
                    mrt_ptr->condition = CONDITION_ENABLED;
            }
            else
            {
                    /* disable all condition flags initially */
                    mrt_ptr->condition = CONDITION_DISABLED;
            }

            /* Append the routing instruction to the temporary list. */
            op_prg_list_insert (mobile_stream_list_ptr, mrt_ptr, OPC_LISTPOS_TAIL);

    }

FRET( mobile_stream_list_ptr )
}


void
mobile_rte_sup_table_print (mobile_stream_list_ptr)
List*mobile_stream_list_ptr;
    {
    mobile_rte_table*table_entry;
    int                         i,size;
    char                        dne_str [128], dno_str [128];
    char                        nne_str [128], nno_str [128], str0 [512];

    /* Print the contents of a routing table. */
    FIN (mobile_rte_sup_table_print (mobile_stream_list_ptr))

    size = op_prg_list_size( mobile_stream_list_ptr );
    if ( size == 0 )
            {
            op_prg_odb_print_major ("Routing table is empty", VOS_NIL);
            }
    else{
            op_prg_odb_print_major ("Routing table contents :", VOS_NIL);
            for (i = 0; i < size; i++)
                    {
                    table_entry = (mobile_rte_table*)
                                  op_prg_list_access( mobile_stream_list_ptr, i );
                    sprintf( str0, "Stream (%d): mtu (%d)"
                            , table_entry->stream, table_entry->mtu );
                    op_prg_odb_print_minor (str0, VOS_NIL);
                    }
            }


    FOUT
    }


Compcode
mobile_rte_sup_route_select ( mobile_table, pkptr, ici_ptr, objid
                            , agnt_flag, pk_id, ttl
```

```
                                , HA_bind_ptr, FA_bind_ptr
                                , CA_bind_ptr, MH_bind_ptr
                                , ip0, ip1, ip2, route_optim, trash_unsent_pk )
    List                        *mobile_table;
    Packet                      *pkptr;
    Ici                         *ici_ptr;
    Objid                       objid;
    int                         agnt_flag;
    int                         *pk_id;
    int                         ttl;
    List                        *HA_bind_ptr, *FA_bind_ptr;
    List                        *CA_bind_ptr, *MH_bind_ptr;
    IP                          ip0, ip1, ip2;
    int                         route_optim;
    int                         trash_unsent_pk;
    {
    char                        str0[100];
    Packet                      *inner_pkptr;
    int                         optim;
    int                         protocol;
    int                         i, j, num_bind, num_multi_bind;
    IP                          src, dest;
    IP                          home_agnt;
    HA_mobility_binding*home_entry;
    FA_mobility_binding*visitor_entry;
    multi_binding       *multi_bind_entry;

        FIN (mobile_rte_sup_route_select( mobile_table, pkptr, ...))

        op_pk_nfd_get( pkptr, "src_net", &src.net );
        op_pk_nfd_get( pkptr, "src_node", &src.node );

        op_pk_nfd_get( pkptr, "dest_net", &dest.net );
        op_pk_nfd_get( pkptr, "dest_node", &dest.node );

        /* Select a route from the routing table which matches the */
        /* requested destination network and node. */
        if ( IP_equal( dest, ip0 ) || IP_equal( dest, ip1 ) || IP_equal( dest, ip2 ) )
        {
                op_pk_nfd_get( pkptr, "protocol", &protocol );

                /* Not encapsulated ( not for a visitor ) */
                if ( protocol != PROTOCOL_ENCAP )
                        FRET ( MOBILE_RTE_TO_IP )

                op_pk_nfd_get( pkptr, "data", &inner_pkptr );
                op_pk_destroy( pkptr );

                pkptr = inner_pkptr;
                op_pk_nfd_get( pkptr, "dest_net", &dest.net );
                op_pk_nfd_get( pkptr, "dest_node", &dest.node );

              if ( forward_to_visitor( mobile_table, FA_bind_ptr, pkptr, ici_ptr, dest ) ==
OPC_COMPCODE_FAILURE )
                {
                        if ( route_optim )
                        {
                              check_ca_list( CA_bind_ptr, dest, &home_agnt );
                              if ( home_agnt.net == ADDRESS_UNDEFINED
                                 && home_agnt.node == ADDRESS_UNDEFINED )
                              {
                                    home_agnt = src;
                              }
                              generate_bind_warning( home_agnt, objid, dest, src );
                        }
```

```
                        if ( trash_unsent_pk )
                        {
                                if (op_prg_odb_ltrace_active ("handoff"))
                                {
                                        sprintf (str0, "Trashing pk(%d) at (%d,%d)"
                                        , op_pk_id (pkptr), ip0.net, ip0.node);
                                        op_prg_odb_print_major (str0, OPC_NIL);
                                }

                                encap_pk_destroy( pkptr );
                                FRET( MOBILE_RTE_FAILURE )
                        }
                        else
                        {
                                if (op_prg_odb_ltrace_active ("handoff"))
                                {
                                        sprintf (str0, "Resending pk(%d) at (%d,%d)"
                                        , op_pk_id (pkptr), ip0.net, ip0.node);
                                        op_prg_odb_print_major (str0, OPC_NIL);
                                }
                                encap_pk_send( pkptr );
                                FRET( MOBILE_RTE_ENCAP )
                        }
                }
                else
                {
                        /* packet sent to visitor */
                        FRET( MOBILE_RTE_SUCCESS )
                }
        }

        if ( IP_equal( src, ip0 ) || IP_equal( src, ip1 ) || IP_equal( src, ip2 ) )
                optim = false;
        else
                optim = route_optim;

        if ( send_via_FA( pkptr, MH_bind_ptr, mobile_table, ici_ptr, dest, agnt_flag ) ==
OPC_COMPCODE_SUCCESS )
                FRET ( MOBILE_RTE_SUCCESS )
        else if ( forward_to_visitor( mobile_table, FA_bind_ptr, pkptr, ici_ptr, dest ) ==
OPC_COMPCODE_SUCCESS )
                FRET ( MOBILE_RTE_SUCCESS )
        else if ( encap_packet( HA_bind_ptr, CA_bind_ptr, pkptr, ici_ptr, ip0, dest, pk_id, ttl,
optim, objid ) == OPC_COMPCODE_SUCCESS )
                FRET ( MOBILE_RTE_ENCAP )
        else
                FRET ( MOBILE_RTE_TO_IP )

    }


    Compcode
    send_via_FA( pkptr, bind_list_ptr, mobile_table, ici_ptr, dest, ma_flag )
    List                *bind_list_ptr, *mobile_table;
    Ici                 *ici_ptr;
    Packet              *pkptr;
    IP                  dest;
    int                 ma_flag;
    {
    int                 i, size;
    int                 copy = false;
    int                 next_net, next_node, stream, mtu;
    Packet              *cp_pkptr;
```

```
Compcode            status= OPC_COMPCODE_SUCCESS;
MH_FA_binding       *fa_entry;
mobile_rte_table*table_entry;
char                        str0[80], str1[80];

FIN( send_via_FA( pkptr, bind_list_ptr, mobile_table, ici_ptr, dest, ma_flag ) )

    /* First of all, check if packet is for any enabled streams */
    size = op_prg_list_size( mobile_table );
    for ( i=0; i<size; ++i )
    {
            table_entry = (mobile_rte_table *) op_prg_list_access( mobile_table, i);

            if ( IP_equal( table_entry->careof, dest ) &&
                 table_entry->condition == CONDITION_ENABLED )
            {
                    stream = table_entry->stream;
                    mtu    = table_entry->mtu;

                    next_net  = ADDRESS_UNDEFINED;
                    next_node = ADDRESS_UNDEFINED;

                    deliver_packet( pkptr, ici_ptr
                            , next_net, next_node, stream, mtu );

                    FRET( OPC_COMPCODE_SUCCESS )
            }
    }


    /* Now, check to see if there is any mobility binding */
    size = op_prg_list_size( bind_list_ptr );

/*  if ( size == 0 && ma_flag )/* no bindings and not a mobile node */
    if ( size == 0 )/* no bindings and not a mobile node */
            FRET( OPC_COMPCODE_FAILURE )

    /* each enabled FA will receive a copy of the packet */

    for( i=0; i<size; ++i )
    {
        fa_entry = (MH_FA_binding *)
                        op_prg_list_access( bind_list_ptr, i );

            /* obtain the stream associated with the current binding */
            stream = fa_entry->stream;

            if ( chk_strm_condition( mobile_table, stream, &mtu ) == CONDITION_ENABLED )
            {
                    status = OPC_COMPCODE_SUCCESS;

                    if ( mtu == MOBILE_STRM_NONEXISTENT )
                    {
                            sprintf (str0, "Discarding packet (%d) ", op_pk_id (pkptr));
                            sprintf (str1, "Stream non-existent" );
                            op_prg_odb_print_major (str0, str1, OPC_NIL);
                            encap_pk_destroy( pkptr );
                    }
                    else
                    {
                            next_net = fa_entry->careof.net;
                            next_node = fa_entry->careof.node;
#if 0
                            deliver_packet( pkptr, ici_ptr
                                        , next_net, next_node, stream, mtu );
```

```
                                FRET( OPC_COMPCODE_SUCCESS )
    #else
                                copy = true;
                                cp_pkptr = op_pk_copy( pkptr );
                                deliver_packet( cp_pkptr, ici_ptr
                                                , next_net, next_node, stream, mtu );
    #endif
                        }
                }
        }

        /* have to deallocate original packet since it is no longer needed */
        if ( copy )
        {
                encap_pk_destroy( pkptr );
                FRET( status )
        }

    FRET( OPC_COMPCODE_FAILURE )
    }


    Compcode
    encap_packet( HA_bind_ptr, CA_bind_ptr, pkptr, ici_ptr, current, dest, pk_id, ttl,
route_optim, reg_objid )
    List                        *HA_bind_ptr;
    List                        *CA_bind_ptr;
    Packet                      *pkptr;
    Ici                         *ici_ptr;
    IP                          current, dest;
    int                         *pk_id;
    int                         ttl;
    int                         route_optim;
    Objid                       reg_objid;
    {
    char        str0[512], str1[512];
    Packet                      *encap_pkptr;
    int                         i, j, num_bind, num_multi_bind;
    int                         next_net, next_node, outstrm, mtu;
    IP                          orig;
    int                         data_len;
    int                         copy = false;
    HA_mobility_binding*home_entry;
    CA_mobility_binding*ca_entry;
    multi_binding       *multi_bind_entry;
    Compcode            status = OPC_COMPCODE_FAILURE;

    FIN( encap_packet( HA_bind_ptr, ...) )

        num_bind = op_prg_list_size( HA_bind_ptr );

        for (i=0; i<num_bind; ++i)
        {
                home_entry = (HA_mobility_binding *)
                        op_prg_list_access( HA_bind_ptr, i );

                /* check for matches in HA mobility binding */
                if (!memcmp( &dest,&home_entry->home_addr, sizeof(IP)))
                {
                        status = OPC_COMPCODE_SUCCESS;
                        num_multi_bind = op_prg_list_size( home_entry->multi_bind_list );
                        for( j=0; j<num_multi_bind; ++j )
                        {
                                multi_bind_entry = (multi_binding *)
                                        op_prg_list_access( home_entry->multi_bind_list, j );
```

```
                        data_len = op_pk_total_size_get(pkptr)/8;
                        encap_pkptr = op_pk_create_fmt( "ip_dgram" );
                        op_pk_bulk_size_set( encap_pkptr, data_len*8 );

                        copy = true;
                        op_pk_nfd_set( encap_pkptr, "data"
                                , op_pk_copy(pkptr) );
                        op_pk_nfd_set( encap_pkptr, "protocol", PROTOCOL_ENCAP );
                        op_pk_nfd_set( encap_pkptr, "src_net",
                                current.net );
                        op_pk_nfd_set( encap_pkptr, "src_node",
                                current.node );
                        op_pk_nfd_set( encap_pkptr, "dest_net",
                                multi_bind_entry->careof.net );
                        op_pk_nfd_set( encap_pkptr, "dest_node",
                                multi_bind_entry->careof.node );

                        op_pk_nfd_set( encap_pkptr,"orig_len",data_len);
                        op_pk_nfd_set( encap_pkptr,"frag_len",data_len);

                        op_pk_nfd_set( encap_pkptr,"ident", (*pk_id)++);
                        op_pk_nfd_set( encap_pkptr,"frag", 0 );
                        op_pk_nfd_set( encap_pkptr, "ttl", ttl);

                        if (op_prg_odb_ltrace_active ("encap_pk"))
                        {
                                sprintf (str0, "Encapsulating pk(%d) sent"
                                        , op_pk_id (encap_pkptr));
                                sprintf (str1, "Destination: net (%d), node (%d)"
                                        , multi_bind_entry->careof.net
                                        , multi_bind_entry->careof.node );
                                op_prg_odb_print_major (str0, str1, OPC_NIL);
                        }

                        encap_pk_send( encap_pkptr );
                }

                /* At this stage, it is assumed that HA binding exists
                     * for mobile node. Therefore send binding warning
                 * message to original sender */
                if ( route_optim == true )
                {
                        op_pk_nfd_get( pkptr, "src_net", &orig.net );
                        op_pk_nfd_get( pkptr, "src_node", &orig.node );

                        generate_bind_warning( orig, reg_objid
                                , home_entry->home_addr, current );
                }
        }
    }

    if ( status == OPC_COMPCODE_SUCCESS )
    {
            if ( copy )
                    encap_pk_destroy( pkptr );
            FRET( status )
    }

#if 0
    if ( route_optim == false )
            FRET( OPC_COMPCODE_FAILURE )
#endif

    num_bind = op_prg_list_size( CA_bind_ptr );
```

```
    for( i=0; i<num_bind; ++i )
    {
            ca_entry = (CA_mobility_binding *)
                        op_prg_list_access( CA_bind_ptr, i );

            if ( IP_equal( dest, ca_entry->home_addr ) )
            {
                    status = OPC_COMPCODE_SUCCESS;

                    data_len = op_pk_total_size_get( pkptr )/8;
                    encap_pkptr = op_pk_create_fmt( "ip_dgram" );
                    op_pk_bulk_size_set( encap_pkptr, data_len*8 );

                    op_pk_nfd_set( encap_pkptr, "data", pkptr );
                    op_pk_nfd_set( encap_pkptr, "protocol"
                                   , PROTOCOL_ENCAP );
                    op_pk_nfd_set( encap_pkptr, "src_net", current.net );
                    op_pk_nfd_set( encap_pkptr, "src_node", current.node );
                    op_pk_nfd_set( encap_pkptr, "dest_net"
                                   , ca_entry->careof.net );
                    op_pk_nfd_set( encap_pkptr, "dest_node"
                                   , ca_entry->careof.node );
                    op_pk_nfd_set( encap_pkptr, "orig_len", data_len );
                    op_pk_nfd_set( encap_pkptr, "frag_len", data_len );
                    op_pk_nfd_set( encap_pkptr, "ident", (*pk_id)++ );
                    op_pk_nfd_set( encap_pkptr, "frag", 0 );
                    op_pk_nfd_set( encap_pkptr, "ttl", ttl );

                    /* Now schedule packet for transmission */
                    encap_pk_send( encap_pkptr );
                    FRET( OPC_COMPCODE_SUCCESS )
            }
    }

    if ( copy )
            encap_pk_destroy( pkptr );

FRET( status )
}


void
deliver_packet( pkptr, ici_ptr, next_net, next_node, outstrm, mtu )
Packet                  *pkptr;
Ici                     *ici_ptr;
int                     next_net, next_node, outstrm, mtu;
{
char        str0[512], str1[512];
int         i, len;
int         header_size, frag_size, data_size;
int         dest_net, dest_node;
int         ttl;
int         frag_accum, frag, num_frags;
Packet      *ip_pkptr, *data_pkptr, *frag_ptr;

FIN( deliver_packet( pkptr, ... ) )

    /* obtain packet's destination */
    op_pk_nfd_get( pkptr, "dest_net", &dest_net );
    op_pk_nfd_get( pkptr, "dest_node", &dest_node );

    /* Decrement the packet's time-to-live field. If zero is reached, */
    /* discard the packet rather than send it on. */
    op_pk_nfd_get (pkptr, "ttl", &ttl);
```

```
	ttl--;
	if (ttl == 0)
	{
		/* In debug mode, indicate that a packet is destroyed */
		/* due to an expired ttl. */
		if (op_prg_odb_ltrace_active ("ip_errs"))
		{
			sprintf (str0, "Discarding packet (%d) with expired TTL", op_pk_id (pkp-
tr));
			sprintf (str1, "Destination: net (%d), node (%d)", dest_net, dest_node);
			op_prg_odb_print_major (str0, str1, OPC_NIL);
		}
		op_pk_destroy (pkptr);
	}
	else{
		/* Assign the new decremented value of ttl. */
		op_pk_nfd_set (pkptr, "ttl", ttl);

		/* In debug mode, trace the routing action. */
		if (op_prg_odb_ltrace_active ("mobile-ip_rte"))
		{
			sprintf (str0, "Routing towards (%d, %d)", dest_net, dest_node);
			sprintf (str1, "Next hop (%d, %d), output stream (%d)",
					next_net, next_node, outstrm);

			op_prg_odb_print_major (str0, str1, OPC_NIL);
		}

		/* Install an Ici indicating to the lower layer what the */
		/* address of the next (intermediate) node is. */
		op_ici_attr_set (ici_ptr, "next_node", next_node);
		op_ici_install (ici_ptr);

		/* Obtain the size in bytes of the fragment. */
		frag_size = op_pk_total_size_get (pkptr) / 8;

		/* Obtain the number of bytes of data carried in this fragment */
		op_pk_nfd_get (pkptr, "frag_len", &data_size);

		/* Also obtain the difference between the packet size */
		/* and the length field: this is the size of the header. */
		header_size = frag_size - data_size;

		/* If it is smaller than the maximum transfer unit, send it as is. */
		if (frag_size <= mtu)
		{
			op_pk_send (pkptr, outstrm);
		}
		else{
			/* Otherwise, break it into fragments */
			/* Each fragment can contain up to (mtu - header_size) bytes of data */
			num_frags = (data_size + mtu - header_size - 1) / (mtu - header_size);

			/* In debug mode, indicate the fragmentation. */
			if (op_prg_odb_ltrace_active ("ip_frag"))
			{
				sprintf (str0, "Breaking datagram into (%d) fragments", num_frags);
				op_prg_odb_print_major (str0, OPC_NIL);
			}

			/* If the fragment is carrying the original datagram given to IP, */
			/* extract it before copies are made. Only one fragment can carry */
			/* the original packet for the reassembly model to work properly. */
			if (op_pk_nfd_is_set (pkptr, "ip_dgram"))
				op_pk_nfd_get (pkptr, "ip_dgram", &ip_pkptr);
```

```
else ip_pkptr = OPC_NIL;

/* If the packet is carrying any encapsulated data (normally this */
/* would happen only for a packet fragmented for the first time), */
/* extract this data packet so that it will not appear in each */
/* fragment generated by copying. */
if (op_pk_nfd_is_set (pkptr, "data"))
       op_pk_nfd_get (pkptr, "data", &data_pkptr);
else data_pkptr = OPC_NIL;

/* Loop through and create the fragments . */
for (frag_accum = 0, i = 0; i < num_frags; i++)
{
       /* Make a copy of the original packet. */
       frag_ptr = op_pk_copy (pkptr);

       /* Indicate that the copy is a fragment */
       op_pk_nfd_set (frag_ptr, "frag", 1);

       /* for all but the last fragment, the size is the mtu. */
       /* and the encapsulated ip packet is not included. */
       if (i < num_frags - 1)
       {
              op_pk_nfd_set (frag_ptr, "frag_len", mtu - header_size);
              op_pk_total_size_set (frag_ptr, 8 * mtu);
              frag_accum += (mtu - header_size);
       }
       else{
              len = data_size - frag_accum;
              op_pk_nfd_set (frag_ptr, "frag_len", len);
              op_pk_total_size_set (frag_ptr,8 * (header_size + len));

              /* If the original packet was not a fragment, encapsulate it
*/
              /* into the last fragment created here. */
              op_pk_nfd_get (pkptr, "frag", &frag);
              if (!frag)
              {
                     /* If the packet contained encapsulated data (i.e.,
from the */
                     /* transport), that data will have been removed to
avoid */
                     /* its duplication in the fragments. The data should
now be */
                     /* reinserted into the original packet. */
                     if (data_pkptr != OPC_NIL)
                            op_pk_nfd_set (pkptr, "data", data_pkptr);

                     /* In either case the original packet is */
                     /* encapsulated in thhe fragment. */
                     op_pk_nfd_set (frag_ptr, "ip_dgram", pkptr);
              }

              /* Otherwise the packet can be discarded. */
              else{
                     op_pk_destroy (pkptr);

                     /* Also, if the packet included the original datagram
*/
                     /* from which it was generated, transfer that data-
gram */
                     /* into the last fragment created here. */
                     /* Note that it is possible, in the case where a
fragment */
```

```
                                        /* is itself being fragmented, that none of the cre-
ated */
                                        /* fragments will contain the original datagram. */
                                        if (ip_pkptr != OPC_NIL)
                                        {
                                            op_pk_nfd_set (frag_ptr, "ip_dgram",
ip_pkptr);
                                        }
                                    }
                                }

                                /* Forward the datagram fragment. */
                                op_pk_send (frag_ptr, outstrm);
                            }
                        }
                    }
                FOUT;
                }



                Compcode
                forward_to_visitor( mobile_table, bind_list_ptr, pkptr, ici_ptr, dest )
                List                *bind_list_ptr, *mobile_table;
                Packet              *pkptr;
                Ici                 *ici_ptr;
                IP                  dest;
                {
                char                str0[80], str1[80];
                Compcode    status = OPC_COMPCODE_FAILURE;
                int                 i, size;
                int                 next_net, next_node, stream, mtu;
                FA_mobility_binding*visitor_entry;

                FIN( forward_to_visitor( mobile_table, bind_list_ptr, pkptr, ici_ptr, dest ) )

                    size = op_prg_list_size( bind_list_ptr );

                    if ( size == 0 )/* list does not exist */
                            FRET ( OPC_COMPCODE_FAILURE )

                    for( i=0; i<size; ++i )
                    {
                            visitor_entry = (FA_mobility_binding *)
                                    op_prg_list_access( bind_list_ptr, i );

                            if ( !memcmp( &dest, &visitor_entry->home_addr, sizeof(IP) ) )
                            {
                                    /* There is a match */
                                    if ( visitor_entry->home_agnt.net != ADDRESS_UNDEFINED &&
                                         visitor_entry->home_agnt.node != ADDRESS_UNDEFINED )
                                    {
                                            status = OPC_COMPCODE_SUCCESS;
                                            next_net = visitor_entry->home_addr.net;
                                            next_node = visitor_entry->home_addr.node;
                                            stream = visitor_entry->stream;
                                            mtu = get_strm_mtu( mobile_table, stream );
                                            if ( mtu == MOBILE_STRM_NONEXISTENT )
                                            {
                                                    sprintf (str0, "Discarding packet (%d) ", op_pk_id (pkptr));
                                                    sprintf (str1, "Stream non-existent" );
                                                    op_prg_odb_print_major (str0, str1, OPC_NIL);
                                                    op_pk_destroy( pkptr );
                                            }
```

```
                        else
                        {
                                deliver_packet( pkptr, ici_ptr
                                                , next_net, next_node, stream, mtu );
                        }
                }
        }

FRET( status );
}


int
get_strm_mtu( strm_ptr, strm )
List*strm_ptr;
int strm;
{
int             i, size;
mobile_rte_table*strm_entry;

FIN( get_strm_mtu( strm_ptr, strm ) )

    size = op_prg_list_size( strm_ptr );

    for( i=0; i<size; ++i )
    {
            strm_entry = (mobile_rte_table *) op_prg_list_access( strm_ptr, i );
            if ( strm_entry->stream == strm )
                    FRET( strm_entry->mtu );
    }

FRET( MOBILE_STRM_NONEXISTENT );
}


int
chk_strm_condition( mobile_table, stream, mtu )
List        *mobile_table;
int         stream;
int         *mtu;
{
int             i, size;
mobile_rte_table*table_entry;
FIN( chk_strm_condition( mobile_table, stream, mtu ) )

    size = op_prg_list_size( mobile_table );

    for( i=0; i<size; ++i )
    {
            table_entry = ( mobile_rte_table * )
                            op_prg_list_access( mobile_table, i );

            if( table_entry->stream == stream )
            {
                    *mtu = table_entry->mtu;
                    FRET( table_entry->condition )
            }
    }


    /* stream does not exist */
    *mtu = MOBILE_STRM_NONEXISTENT;
```

```
FRET( CONDITION_DISABLED )
}

void
get_careof_addr( mobile_table, iciptr )
List        *mobile_table;
Ici         *iciptr;
{
int             i, size;
int             stream;
IP              careof;
mobile_rte_table*table_entry;
FIN( get_careof_addr( mobile_table, iciptr ) )

    op_ici_attr_get( iciptr, "stream", &stream );

    size = op_prg_list_size( mobile_table );
    for( i=0; i<size; ++i )
    {
            table_entry = ( mobile_rte_table * )
                        op_prg_list_access( mobile_table, i );

            if( table_entry->stream == stream )
            {
                    op_ici_attr_set( iciptr, "careof_net", table_entry->careof.net );
                    op_ici_attr_set( iciptr, "careof_node", table_entry->careof.node );
                    op_ici_attr_set( iciptr, "status", OPC_COMPCODE_SUCCESS );
                    FOUT
            }
    }


    /* stream does not exist */
    op_ici_attr_set( iciptr, "careof_net", ADDRESS_UNDEFINED );
    op_ici_attr_set( iciptr, "careof_node", ADDRESS_UNDEFINED );
    op_ici_attr_set( iciptr, "status", OPC_COMPCODE_FAILURE );

FOUT
}


void
set_strm_condition( mobile_table, iciptr )
List*mobile_table;
Ici *iciptr;
{
IP          careof;
int         stream;
int         mode;
int         i, size;
Compcodestatus = OPC_COMPCODE_FAILURE;
mobile_rte_table*list_ptr;
FIN( set_strm_condition( iciptr ) )

    op_ici_attr_get( iciptr, "mode", &mode );
    op_ici_attr_get( iciptr, "stream", &stream );

    size = op_prg_list_size( mobile_table );
    for( i=0; i<size; ++i )
    {
            list_ptr = (mobile_rte_table *)
                        op_prg_list_access( mobile_table, i );

            switch( mode )
            {
```

```
		case DISABLE_ALL:
			list_ptr->condition = CONDITION_DISABLED;
			status = OPC_COMPCODE_SUCCESS;
			break;

		case DISABLE_ALL_EXCEPT_THIS:
			if( list_ptr->stream != stream )
			{
				list_ptr->condition = CONDITION_DISABLED;
				status = OPC_COMPCODE_SUCCESS;
			}
			break;

		case DISABLE_THIS_ONLY:
			if( list_ptr->stream == stream )
			{
				op_ici_attr_set( iciptr, "careof_net", list_ptr->careof.net );
				op_ici_attr_set( iciptr, "careof_node", list_ptr->careof.node );
				list_ptr->condition = CONDITION_DISABLED;
				status = OPC_COMPCODE_SUCCESS;
			}
			break;

		case ENABLE_ALL:
			list_ptr->condition = CONDITION_ENABLED;
			status = OPC_COMPCODE_SUCCESS;
			break;

		case ENABLE_ALL_EXCEPT_THIS:
			if( list_ptr->stream != stream )
			{
				list_ptr->condition = CONDITION_ENABLED;
				status = OPC_COMPCODE_SUCCESS;
			}
			break;

		case ENABLE_THIS_ONLY:
			if( list_ptr->stream == stream )
			{
				op_ici_attr_set( iciptr, "careof_net", list_ptr->careof.net );
				op_ici_attr_set( iciptr, "careof_node", list_ptr->careof.node );
				list_ptr->condition = CONDITION_ENABLED;
				status = OPC_COMPCODE_SUCCESS;
			}
			break;

		default:
			status = OPC_COMPCODE_FAILURE;
			break;
		}
	}

	op_ici_attr_set( iciptr, "status", status );
FOUT
}

void
hop_to_next_strm( mobile_table, iciptr )
List	*mobile_table;
Ici	*iciptr;
{
int		j, i, size;
int		stream;
mobile_rte_table*table_entry;
FIN( hop_to_next_strm( mobile_table, iciptr ) )
```

```
    op_ici_attr_get( iciptr, "stream", &stream );

    size = op_prg_list_size( mobile_table );

    for ( i=0; i<size; ++i )
    {
          table_entry = ( mobile_rte_table * )
                        op_prg_list_access( mobile_table, i );

          if ( table_entry->stream == stream )
          {
                if ( ( j = i + 1 ) == size )
                      j = 0;

                table_entry = ( mobile_rte_table * )
                      op_prg_list_access( mobile_table, j );

                op_ici_attr_set( iciptr, "stream"
                              , table_entry->stream );

                op_ici_attr_set( iciptr, "careof_net"
                              ,table_entry->careof.net);

                op_ici_attr_set( iciptr, "careof_node"
                              ,table_entry->careof.node);

                /* Now, enabling this stream */
                table_entry->condition = CONDITION_ENABLED;

                op_ici_attr_set( iciptr, "status"
                              , OPC_COMPCODE_SUCCESS );

                FOUT
          }
    }
    op_ici_attr_set( iciptr, "status", OPC_COMPCODE_FAILURE );
FOUT
}

void
process_binding_warning( pkptr, iciptr, rcv_iciptr, ip_objid )
Packet          *pkptr;
Ici             *iciptr;
Ici             *rcv_iciptr;
Objid           ip_objid;
{
IP          home_addr, rem, target, home_agnt;
int         rem_port;
Packet            *bind_pkptr;
Compcode    status;
Ici               *bind_iciptr;
FIN( process_binding_warning( pkptr, iciptr, rcv_iciptr, ip_objid ) )

    op_pk_nfd_get( pkptr, "home_addr_net", &home_addr.net );
    op_pk_nfd_get( pkptr, "home_addr_node", &home_addr.node );

    op_pk_nfd_get( pkptr, "target_net", &target.net );
    op_pk_nfd_get( pkptr, "target_node", &target.node );

    op_ici_attr_get( rcv_iciptr, "rem_net",  &rem.net );
    op_ici_attr_get( rcv_iciptr, "rem_node", &rem.node );
    op_ici_attr_get( rcv_iciptr, "rem_port", &rem_port );

    /* creating binding for Mobile-ip query */
```

```
	bind_iciptr = op_ici_create( "binding_command" );

	op_ici_attr_set( bind_iciptr, "command", READ_HA_BINDING );
	op_ici_attr_set( bind_iciptr, "home_net", home_addr.net );
	op_ici_attr_set( bind_iciptr, "home_node", home_addr.node );

	op_ici_install( bind_iciptr );
	op_intrpt_force_remote( BINDING_MAINTENANCE, ip_objid );
	op_ici_install( OPC_NIL );

	op_ici_attr_get( bind_iciptr, "status", &status );

	if ( status == OPC_COMPCODE_SUCCESS )
	{
		/* send binding warning if home addr is a registered MH */
		bind_pkptr = op_pk_create_fmt( "bind_warn" );
		op_pk_nfd_set( bind_pkptr, "home_addr_net", home_addr.net );
		op_pk_nfd_set( bind_pkptr, "home_addr_node", home_addr.node );
		op_pk_nfd_set( bind_pkptr, "target_net", target.net );
		op_pk_nfd_set( bind_pkptr, "target_node", target.node );

		udp_app_send( iciptr, bind_pkptr, REG_REQUEST_PORT
				, REG_REQUEST_PORT, target.net, target.node );
	}
	else
	{
#if 1
		/* first of all, check whether home addr is on cache list */
		op_ici_attr_set( bind_iciptr, "command", READ_CA_BINDING );

		op_ici_install( bind_iciptr );
		op_intrpt_force_remote( BINDING_MAINTENANCE, ip_objid );
		op_ici_install( OPC_NIL );

		op_ici_attr_get( bind_iciptr, "status", &status );

		if ( status == OPC_COMPCODE_SUCCESS )
		{
			/* Home addr on cache list, send binding request to home addr */
			op_ici_attr_get( bind_iciptr, "home_agnt_net", &rem.net );
			op_ici_attr_get( bind_iciptr, "home_agnt_node", &rem.node );

			bind_pkptr = op_pk_create_fmt( "bind_req" );
			op_pk_nfd_set( bind_pkptr, "home_addr_net", home_addr.net );
			op_pk_nfd_set( bind_pkptr, "home_addr_node", home_addr.node );

			/* send the registration packet */
			udp_app_send( iciptr, bind_pkptr, REG_REQUEST_PORT
				, REG_REQUEST_PORT, rem.net, rem.node );
		}
		else
		{
			/* no binding found, send it to target */
			bind_pkptr = op_pk_create_fmt( "bind_req" );
			op_pk_nfd_set( bind_pkptr, "home_addr_net", home_addr.net );
			op_pk_nfd_set( bind_pkptr, "home_addr_node", home_addr.node );

			/* send the registration packet */
			udp_app_send( iciptr, bind_pkptr, REG_REQUEST_PORT
				, REG_REQUEST_PORT, target.net, target.node );
		}
#else
		bind_pkptr = op_pk_create_fmt( "bind_req" );
		op_pk_nfd_set( bind_pkptr, "home_addr_net", home_addr.net );
		op_pk_nfd_set( bind_pkptr, "home_addr_node", home_addr.node );
```

```
            /* send the registration packet */
            udp_app_send( iciptr, bind_pkptr, REG_REQUEST_PORT
                    , rem_port, rem.net, rem.node );
#endif
    }

    /* deallocate resources after usage */
    op_ici_destroy( bind_iciptr );
FOUT
}


void
process_binding_request( pkptr, iciptr, rcv_iciptr, objid )
Packet          *pkptr;
Ici             *iciptr;
Ici             *rcv_iciptr;
Objid           objid;
{
Packet      *bind_pkptr;
IP          home_addr;
IP          careof;
IP          rem;
int         rem_port;
Ici         *ip_iciptr;
int         status;
int         lifetime;
FIN( process_binding_request( pkptr, iciptr, rcv_iciptr, objid ) )

    op_pk_nfd_get( pkptr, "home_addr_net", &home_addr.net );
    op_pk_nfd_get( pkptr, "home_addr_node", &home_addr.node );

    op_ici_attr_get( rcv_iciptr, "rem_net",  &rem.net );
    op_ici_attr_get( rcv_iciptr, "rem_node", &rem.node );
    op_ici_attr_get( rcv_iciptr, "rem_port", &rem_port );

    ip_iciptr = op_ici_create( "binding_command" );
    op_ici_attr_set( ip_iciptr, "command", READ_HA_BINDING );
    op_ici_attr_set( ip_iciptr, "home_net", home_addr.net );
    op_ici_attr_set( ip_iciptr, "home_node", home_addr.node );

    op_ici_install( ip_iciptr );
    op_intrpt_force_remote( BINDING_MAINTENANCE, objid );
    op_ici_install( OPC_NIL );

    op_ici_attr_get( ip_iciptr, "status", &status );
    if ( status == OPC_COMPCODE_FAILURE )
    {
            careof = home_addr;/* no binding exists */
            lifetime = 0;
    }
    else
    {
            op_ici_attr_get( ip_iciptr, "careof_net", &careof.net );
            op_ici_attr_get( ip_iciptr, "careof_node", &careof.node );
            op_ici_attr_get( ip_iciptr, "lifetime", &lifetime );
    }

    /* deallocate ici pointer after use */
    op_ici_destroy( ip_iciptr );

    bind_pkptr = op_pk_create_fmt( "bind_update" );
    op_pk_nfd_set( bind_pkptr, "home_addr_net", home_addr.net );
```

```
    op_pk_nfd_set( bind_pkptr, "home_addr_node", home_addr.node );
    op_pk_nfd_set( bind_pkptr, "careof_net", careof.net );
    op_pk_nfd_set( bind_pkptr, "careof_node", careof.node );
    op_pk_nfd_set( bind_pkptr, "lifetime", lifetime );

    /* send binding request via UDP port */
    udp_app_send( iciptr, bind_pkptr, REG_REQUEST_PORT, rem_port, rem.net, rem.node );

FOUT
}

void
process_binding_update( pkptr, iciptr, rcv_iciptr, objid )
Packet          *pkptr;
Ici             *iciptr;
Ici             *rcv_iciptr;
Objid           objid;
{
Packet     *bind_pkptr;
IP         home_addr;
IP         careof;
IP         rem;
int        rem_port;
Ici        *ip_iciptr;
int        lifetime, ack, status;
FIN( process_binding_update( pkptr, iciptr, rcv_iciptr, objid ) )

    op_pk_nfd_get( pkptr, "home_addr_net", &home_addr.net );
    op_pk_nfd_get( pkptr, "home_addr_node", &home_addr.node );
    op_pk_nfd_get( pkptr, "careof_net", &careof.net );
    op_pk_nfd_get( pkptr, "careof_node", &careof.node );
    op_pk_nfd_get( pkptr, "lifetime", &lifetime );
    op_pk_nfd_get( pkptr, "ack", &ack );

    op_ici_attr_get( rcv_iciptr, "rem_net",  &rem.net );
    op_ici_attr_get( rcv_iciptr, "rem_node", &rem.node );
    op_ici_attr_get( rcv_iciptr, "rem_port", &rem_port );

    ip_iciptr = op_ici_create( "binding_command" );

    if ( (careof.net == 0 && careof.node == 0) || lifetime == 0 )
    {
           op_ici_attr_set( ip_iciptr, "command", KILL_CA_BINDING );
           op_ici_attr_set( ip_iciptr, "home_net", home_addr.net );
           op_ici_attr_set( ip_iciptr, "home_node", home_addr.node );

           op_ici_install( ip_iciptr );
           op_intrpt_force_remote( BINDING_MAINTENANCE, objid );
           op_ici_install( OPC_NIL );

    }
    else
    {
           op_ici_attr_set( ip_iciptr, "command", EDIT_CA_BINDING );
           op_ici_attr_set( ip_iciptr, "home_net", home_addr.net );
           op_ici_attr_set( ip_iciptr, "home_node", home_addr.node );
           op_ici_attr_set( ip_iciptr, "home_agnt_net", rem.net );
           op_ici_attr_set( ip_iciptr, "home_agnt_node", rem.node );
           op_ici_attr_set( ip_iciptr, "careof_net", careof.net );
           op_ici_attr_set( ip_iciptr, "careof_node", careof.node );
           op_ici_attr_set( ip_iciptr, "lifetime", lifetime );

           op_ici_install( ip_iciptr );
           op_intrpt_force_remote( BINDING_MAINTENANCE, objid );
           op_ici_install( OPC_NIL );
```

```
            if ( ack == true )
            {
                    bind_pkptr = op_pk_create_fmt( "bind_ack" );
                    op_pk_nfd_set( bind_pkptr, "home_addr_net", home_addr.net );
                    op_pk_nfd_set( bind_pkptr, "home_addr_node", home_addr.node );

                    /* send binding request via UDP port */
                    udp_app_send( iciptr, bind_pkptr, REG_REQUEST_PORT
                                    , rem_port, rem.net, rem.node );
            }
    }

    /* deallocate ici pointer after use */
    op_ici_destroy( ip_iciptr );

FOUT
}

void
encap_pk_destroy( pkptr )
Packet      *pkptr;
{
Packet      *outer_pkptr, *inner_pkptr;
int         i, num_fds, data_present;
char        fd_name[40];
FIN( encap_pk_destroy( pkptr ) )

    outer_pkptr = pkptr;


    while ( 1 )
    {
            num_fds = op_pk_fd_max_index( outer_pkptr );

            for ( data_present=false,i=0; i< num_fds; ++i )
            {
                    op_pk_fd_index_to_name( outer_pkptr, i, fd_name );
                    if ( strcmp( fd_name, "data" ) == 0 )
                    {
                            op_pk_nfd_get( outer_pkptr, "data", &inner_pkptr );
                            op_pk_destroy( outer_pkptr );

                            data_present = true;
                            break;
                    }
            }

            /* no fields with name "data" */
            if ( data_present == false )
                    break;

            outer_pkptr = inner_pkptr;
    }

    op_pk_destroy( outer_pkptr );
FOUT
}


void
generate_bind_warning( dest, reg_objid, home_addr, target )
IP                  dest, home_addr, target;
Objid               reg_objid;
{
```

```
Ici        *warn_iciptr;
FIN( generate_bind_warning( home_addr, ) )

    warn_iciptr = op_ici_create("bind_warn_ici");
    op_ici_attr_set( warn_iciptr, "home_addr_net", home_addr.net );
    op_ici_attr_set( warn_iciptr, "home_addr_node", home_addr.node );

    op_ici_attr_set( warn_iciptr, "dest_net", dest.net );
    op_ici_attr_set( warn_iciptr, "dest_node",dest.node );

    op_ici_attr_set( warn_iciptr, "target_net", target.net );
    op_ici_attr_set( warn_iciptr, "target_node",target.node );

    op_ici_install( warn_iciptr );
    op_intrpt_force_remote( BINDING_WARN_TYPE, reg_objid );
    op_ici_install( OPC_NIL );

    op_ici_destroy( warn_iciptr );
FOUT
}


void
encap_pk_send( pkptr )
Packet     *pkptr;
{
FIN( encap_pk_send( pkptr ) )

    /* insert encapsulated packet at the beginning of the queue */
    op_subq_pk_insert( 0, pkptr, OPC_QPOS_HEAD );
FOUT
}


void
check_ca_list( list_ptr, dest, home_agnt )
List       *list_ptr;
IP         dest;
IP         *home_agnt;
{
CA_mobility_binding*ca_entry;
int                i, num_bind;
FIN( check_ca_list( list_ptr, dest, home_agnt ) )

        num_bind = op_prg_list_size( list_ptr );
    for( i=0; i<num_bind; ++i )
    {
        ca_entry = (CA_mobility_binding *)
                                op_prg_list_access( list_ptr, i );

        if ( IP_equal( dest, ca_entry->home_addr ) )
        {
                home_agnt->net  = ca_entry->home_agnt.net;
                home_agnt->node = ca_entry->home_agnt.node;
                FOUT
        }
    }
    home_agnt->net  = ADDRESS_UNDEFINED;
    home_agnt->node = ADDRESS_UNDEFINED;
FOUT
}
```

Appendix C. Supplementary Source Code of the Handoff Enhanced Model

```
/* mobile_rte_sup.ex.c */
/* Routing support procedures for the Mobile IP example model */

#include <opnet.h>
#include "mobile-ip.h"
#include "mobile_rte_sup.h"
#include "ip_rte_sup.h"
#include "protocol.h"



/* Functions called by Process Module */

List*
mobile_rte_sup_table_setup (file_name)
    char                    *file_name;
    {
    List*                   mobile_strm_list_ptr;
    List*                   line_list_ptr;
    mobile_rte_table*table_ptr;


    /* Provides comprehensive routing table loading and parsing */
    /* services for Mobile IP module. */
    FIN (mobile_rte_sup_table_setup (file_name, net0, node0, net1, node1, net2, node2))

    /* Load the list of text lines from the specified file. */
    /* Note: this procedure will quit the simnulation if */
    /* file cannot be loaded, so it is assumed that there */
    /* are no problems upon returning. */
    line_list_ptr = mobile_rte_sup_table_load (file_name);

    /* Parse the contents of the obtained list into */
    /* a routing-instruction table. */
    mobile_strm_list_ptr = mobile_rte_sup_list_parse (line_list_ptr);

    /* In debug mode, if tracing is active, print the table */
    if (op_prg_odb_trace_active ())
           {
           mobile_rte_sup_table_print (mobile_strm_list_ptr);
           }

    FRET( mobile_strm_list_ptr );
    }

List*
mobile_rte_sup_table_load (file_name)
    char                    *file_name;
    {
    List*                   line_list_ptr;
    char                    err_str [256];

    /* Read in a routing table from an ascii */
    /* file adhering to format defined above. */
    FIN (mobile_rte_sup_table_load (file_name))

    /* Open and read the file into the list. */
    line_list_ptr = op_prg_gdf_read (file_name);
```

```
    /* Test for error in reading. */
    if (line_list_ptr == OPC_NIL)
            {
            sprintf (err_str, "File Name: %s", file_name);
            op_sim_end ("Package  : mobile_rte_sup",
                        "Error    : Unable to read routing table file",
                        err_str, "");
            }

    /* Return the list of text lines. */
    FRET (line_list_ptr)
    }


List*
mobile_rte_sup_list_parse (line_list_ptr)
    List*                       line_list_ptr;
    {
    List*                       mobile_stream_list_ptr;
    mobile_rte_table*mrt_ptr;
    int                         i, num_lines;
    char*                       line;
    List                        *field_list_ptr;


    /* Extract information from the lines of an ascii routing table */
    /*  and construct a corresponding routing table structure which */
    /* contains routing instructions. */
    FIN (mobile_rte_sup_list_parse (line_list_ptr))

    /* Allocate a routing table structure. */
    mrt_ptr = (mobile_rte_table*) op_prg_mem_alloc (sizeof (mobile_rte_table));

    /* Allocate a temporary table for holding lists. */
    mobile_stream_list_ptr = op_prg_list_create ();

    /* Scan through each of the lines, one at a time. */
    num_lines = op_prg_list_size (line_list_ptr);
    for (i = 0; i < num_lines; i++)
    {
            /* Obtain the i_th line. */
            line = op_prg_list_access (line_list_ptr, i);

            /* Decompose it into fields (field boundaries are */
            /* indicated by spaces, tabs,  slashes, or commas. */
            field_list_ptr = op_prg_str_decomp (line, " ,/\t");

            /* Format for a line is as follows: */
            /* <output_stream> <mtu>*/
            /* Incomplete lines are skipped. */
            if (op_prg_list_size (field_list_ptr) < 4)
                    continue;

            /* Create a routing instruction structure. */
            mrt_ptr = (mobile_rte_table*)
                    op_prg_mem_alloc (sizeof (mobile_rte_table));

            /* Transfer the parsed fields into the structure */
            /* First obtain the destination stream and mtu fields. */
            mrt_ptr->stream = atoi (
                    op_prg_list_access (field_list_ptr, MOBILE_TBL_OUTSTREAM));
            mrt_ptr->mtu = atoi (
                    op_prg_list_access (field_list_ptr, MOBILE_TBL_MTU));
```

```
            mrt_ptr->careof.net = atoi (
                    op_prg_list_access (field_list_ptr, MOBILE_TBL_CAREOF_NET));

            mrt_ptr->careof.node = atoi (
                    op_prg_list_access (field_list_ptr, MOBILE_TBL_CAREOF_NODE));

            if (mrt_ptr->mtu <= 0)
                    mrt_ptr->mtu = 0x7FFFFFFF;

            if (mrt_ptr->careof.net  == ADDRESS_UNDEFINED &&
                mrt_ptr->careof.node == ADDRESS_UNDEFINED )
            {
                    /* careof address not implemented */
                    mrt_ptr->condition = CONDITION_ENABLED;
            }
            else
            {
                    /* disable all condition flags initially */
                    mrt_ptr->condition = CONDITION_DISABLED;
            }

            /* Append the routing instruction to the temporary list. */
            op_prg_list_insert (mobile_stream_list_ptr, mrt_ptr, OPC_LISTPOS_TAIL);

    }

FRET( mobile_stream_list_ptr )
}


void
mobile_rte_sup_table_print (mobile_stream_list_ptr)
List*mobile_stream_list_ptr;
    {
    mobile_rte_table*table_entry;
    int                         i,size;
    char                        dne_str [128], dno_str [128];
    char                        nne_str [128], nno_str [128], str0 [512];

    /* Print the contents of a routing table. */
    FIN (mobile_rte_sup_table_print (mobile_stream_list_ptr))

    size = op_prg_list_size( mobile_stream_list_ptr );
    if ( size == 0 )
            {
            op_prg_odb_print_major ("Routing table is empty", VOS_NIL);
            }
    else{
            op_prg_odb_print_major ("Routing table contents :", VOS_NIL);
            for (i = 0; i < size; i++)
                    {
                    table_entry = (mobile_rte_table*)
                                  op_prg_list_access( mobile_stream_list_ptr, i );
                    sprintf( str0, "Stream (%d): mtu (%d)"
                            , table_entry->stream, table_entry->mtu );
                    op_prg_odb_print_minor (str0, VOS_NIL);
                    }
            }


    FOUT
    }


Compcode
```

```
mobile_rte_sup_route_select ( mobile_table, pkptr, ici_ptr, objid
                            , agnt_flag, pk_id, ttl
                            , HA_bind_ptr, FA_bind_ptr
                            , CA_bind_ptr, MH_bind_ptr
                            , ip0, ip1, ip2, route_optim, trash_unsent_pk
                            , buffer_ptr, buffer_life )
List                        *mobile_table;
Packet                      *pkptr;
Ici                         *ici_ptr;
Objid                       objid;
int                         agnt_flag;
int                         *pk_id;
int                         ttl;
List                        *HA_bind_ptr, *FA_bind_ptr;
List                        *CA_bind_ptr, *MH_bind_ptr;
IP                          ip0, ip1, ip2;
int                         route_optim;
int                         trash_unsent_pk;
List                        *buffer_ptr;
double                      buffer_life;
{
char                        str0[100];
Packet                      *inner_pkptr;
Packet                      *buffer_pkptr;
int                         optim;
int                         protocol;
int                         i, j, num_bind, num_multi_bind;
IP                          src, dest;
IP                          home_agnt;
HA_mobility_binding*home_entry;
FA_mobility_binding*visitor_entry;
multi_binding       *multi_bind_entry;

    FIN (mobile_rte_sup_route_select( mobile_table, pkptr, ...))

    op_pk_nfd_get( pkptr, "src_net", &src.net );
    op_pk_nfd_get( pkptr, "src_node", &src.node );

    op_pk_nfd_get( pkptr, "dest_net", &dest.net );
    op_pk_nfd_get( pkptr, "dest_node", &dest.node );

    /* Select a route from the routing table which matches the */
    /* requested destination network and node. */
    if ( IP_equal( dest, ip0 ) || IP_equal( dest, ip1 ) || IP_equal( dest, ip2 ) )
    {
            op_pk_nfd_get( pkptr, "protocol", &protocol );

            /* Not encapsulated ( not for a visitor ) */
            if ( protocol != PROTOCOL_ENCAP )
                    FRET ( MOBILE_RTE_TO_IP )

            op_pk_nfd_get( pkptr, "data", &inner_pkptr );
            op_pk_destroy( pkptr );

            pkptr = inner_pkptr;
            op_pk_nfd_get( pkptr, "dest_net", &dest.net );
            op_pk_nfd_get( pkptr, "dest_node", &dest.node );

            if ( !trash_unsent_pk )
            {
/*                  buffer_pkptr = op_pk_copy( pkptr ); */
                    if( insert_pk_buffer( buffer_ptr, buffer_life, dest, pkptr )
                                    == OPC_COMPCODE_SUCCESS )
                    {
                            if (op_prg_odb_ltrace_active ("pk_buffer"))
```

```
                                {
                                        sprintf (str0, "Buffering pk(%d) at (%d,%d)"
                                                , op_pk_id (pkptr), ip0.net, ip0.node);
                                        op_prg_odb_print_major (str0, OPC_NIL);
                                }
                                FRET( MOBILE_RTE_SUCCESS )
                        }
                        else
                        {
    /*                          op_pk_destroy( buffer_pkptr ); */
                        }
                }

                if ( forward_to_visitor( mobile_table, FA_bind_ptr, pkptr, ici_ptr, dest ) ==
OPC_COMPCODE_FAILURE )
                {
                        if ( route_optim )
                        {
                                check_ca_list( CA_bind_ptr, dest, &home_agnt );
                                if ( home_agnt.net == ADDRESS_UNDEFINED
                                   && home_agnt.node == ADDRESS_UNDEFINED )
                                {
                                        home_agnt = src;
                                }
                                generate_bind_warning( home_agnt, objid, dest, src );
                        }

                        if ( encap_packet( HA_bind_ptr, CA_bind_ptr, pkptr, ici_ptr, ip0
                                        , dest, pk_id, ttl, optim, objid )
                                == OPC_COMPCODE_SUCCESS )
                        {
                                FRET( MOBILE_RTE_ENCAP )
                        }
                        else
                        {
                                if ( trash_unsent_pk )
                                {
                                        if (op_prg_odb_ltrace_active ("pk_buffer"))
                                        {
                                                sprintf (str0, "Trashing pk(%d) at (%d,%d)"
                                                        , op_pk_id (pkptr), ip0.net, ip0.node);
                                                op_prg_odb_print_major (str0, OPC_NIL);
                                        }
                                        op_pk_destroy( pkptr );
                                }
                                FRET( MOBILE_RTE_FAILURE )
                        }
                }
                else
                {
                        /* packet sent to visitor */
                        FRET( MOBILE_RTE_SUCCESS )
                }
        }

        if ( IP_equal( src, ip0 ) || IP_equal( src, ip1 ) || IP_equal( src, ip2 ) )
                optim = false;
        else
                optim = route_optim;

        if ( send_via_FA( pkptr, MH_bind_ptr, mobile_table, ici_ptr, dest, agnt_flag ) ==
OPC_COMPCODE_SUCCESS )
                FRET ( MOBILE_RTE_SUCCESS )
        else if ( forward_to_visitor( mobile_table, FA_bind_ptr, pkptr, ici_ptr, dest ) ==
OPC_COMPCODE_SUCCESS )
```

```
                FRET ( MOBILE_RTE_SUCCESS )
        else if ( encap_packet( HA_bind_ptr, CA_bind_ptr, pkptr, ici_ptr, ip0, dest, pk_id, ttl,
optim, objid ) == OPC_COMPCODE_SUCCESS )
                FRET ( MOBILE_RTE_ENCAP )
        else
                FRET ( MOBILE_RTE_TO_IP )

    }


    Compcode
    send_via_FA( pkptr, bind_list_ptr, mobile_table, ici_ptr, dest, ma_flag )
    List                    *bind_list_ptr, *mobile_table;
    Ici                     *ici_ptr;
    Packet                  *pkptr;
    IP                      dest;
    int                     ma_flag;
    {
    int                     i, size;
    int                     copy = false;
    int                     next_net, next_node, stream, mtu;
    Packet                  *cp_pkptr;
    Compcode            status= OPC_COMPCODE_SUCCESS;
    MH_FA_binding       *fa_entry;
    mobile_rte_table*table_entry;
    char                    str0[80], str1[80];

    FIN( send_via_FA( pkptr, bind_list_ptr, mobile_table, ici_ptr, dest, ma_flag ) )

        /* First of all, check if packet is for any enabled streams */
        size = op_prg_list_size( mobile_table );
        for ( i=0; i<size; ++i )
        {
                table_entry = (mobile_rte_table *) op_prg_list_access( mobile_table, i);

                if ( IP_equal( table_entry->careof, dest ) &&
                     table_entry->condition == CONDITION_ENABLED )
                {
                        stream = table_entry->stream;
                        mtu    = table_entry->mtu;

                        next_net  = ADDRESS_UNDEFINED;
                        next_node = ADDRESS_UNDEFINED;

                        deliver_packet( pkptr, ici_ptr
                                , next_net, next_node, stream, mtu );

                        FRET( OPC_COMPCODE_SUCCESS )
                }
        }


        /* Now, check to see if there is any mobility binding */
        size = op_prg_list_size( bind_list_ptr );

    /*  if ( size == 0 && ma_flag )/* no bindings and not a mobile node */
        if ( size == 0 )/* no bindings and not a mobile node */
                FRET( OPC_COMPCODE_FAILURE )

        /* each enabled FA will receive a copy of the packet */

        for( i=0; i<size; ++i )
        {
                fa_entry = (MH_FA_binding *)
```

```
                        op_prg_list_access( bind_list_ptr, i );

                /* obtain the stream associated with the current binding */
                stream = fa_entry->stream;

                if ( chk_strm_condition( mobile_table, stream, &mtu ) == CONDITION_ENABLED )
                {
                        status = OPC_COMPCODE_SUCCESS;

                        if ( mtu == MOBILE_STRM_NONEXISTENT )
                        {
                                sprintf (str0, "Discarding packet (%d) ", op_pk_id (pkptr));
                                sprintf (str1, "Stream non-existent" );
                                op_prg_odb_print_major (str0, str1, OPC_NIL);
                                op_pk_destroy( pkptr );
                        }
                        else
                        {
                                next_net = fa_entry->careof.net;
                                next_node = fa_entry->careof.node;
    #if 0
                                deliver_packet( pkptr, ici_ptr
                                                , next_net, next_node, stream, mtu );
                                FRET( OPC_COMPCODE_SUCCESS )
    #else
                                copy = true;
                                cp_pkptr = op_pk_copy( pkptr );
                                deliver_packet( cp_pkptr, ici_ptr
                                                , next_net, next_node, stream, mtu );
    #endif
                        }
                }
        }

        /* have to deallocate original packet since it is no longer needed */
        if ( copy )
        {
                op_pk_destroy( pkptr );
                FRET( status )
        }

    FRET( OPC_COMPCODE_FAILURE )
    }


    Compcode
    encap_packet( HA_bind_ptr, CA_bind_ptr, pkptr, ici_ptr, current, dest, pk_id, ttl,
route_optim, reg_objid )
    List                        *HA_bind_ptr;
    List                        *CA_bind_ptr;
    Packet                      *pkptr;
    Ici                         *ici_ptr;
    IP                          current, dest;
    int                         *pk_id;
    int                         ttl;
    int                         route_optim;
    Objid                       reg_objid;
    {
    char        str0[512], str1[512];
    Packet                      *encap_pkptr;
    int                         i, j, num_bind, num_multi_bind;
    int                         next_net, next_node, outstrm, mtu;
    IP                          orig;
    int                         data_len;
    int                         copy = false;
```

```
HA_mobility_binding*home_entry;
CA_mobility_binding*ca_entry;
multi_binding      *multi_bind_entry;
Compcode           status = OPC_COMPCODE_FAILURE;

FIN( encap_packet( HA_bind_ptr, ...) )

    num_bind = op_prg_list_size( HA_bind_ptr );

    for (i=0; i<num_bind; ++i)
    {
           home_entry = (HA_mobility_binding *)
                  op_prg_list_access( HA_bind_ptr, i );

           /* check for matches in HA mobility binding */
           if (!memcmp( &dest,&home_entry->home_addr, sizeof(IP)))
           {
                  status = OPC_COMPCODE_SUCCESS;
                  num_multi_bind = op_prg_list_size( home_entry->multi_bind_list );
                  for( j=0; j<num_multi_bind; ++j )
                  {
                         multi_bind_entry = (multi_binding *)
                                op_prg_list_access( home_entry->multi_bind_list, j );

                         data_len = op_pk_total_size_get(pkptr)/8;
                         encap_pkptr = op_pk_create_fmt( "ip_dgram" );
                         op_pk_bulk_size_set( encap_pkptr, data_len*8 );

                         copy = true;
                         op_pk_nfd_set( encap_pkptr, "data"
                                       , op_pk_copy(pkptr) );
                         op_pk_nfd_set( encap_pkptr, "protocol", PROTOCOL_ENCAP );
                         op_pk_nfd_set( encap_pkptr, "src_net",
                                       current.net );
                         op_pk_nfd_set( encap_pkptr, "src_node",
                                       current.node );
                         op_pk_nfd_set( encap_pkptr, "dest_net",
                                       multi_bind_entry->careof.net );
                         op_pk_nfd_set( encap_pkptr, "dest_node",
                                       multi_bind_entry->careof.node );

                         op_pk_nfd_set( encap_pkptr,"orig_len",data_len);
                         op_pk_nfd_set( encap_pkptr,"frag_len",data_len);

                         op_pk_nfd_set( encap_pkptr,"ident", (*pk_id)++);
                         op_pk_nfd_set( encap_pkptr,"frag", 0 );
                         op_pk_nfd_set( encap_pkptr, "ttl", ttl);

                         if (op_prg_odb_ltrace_active ("encap_pk"))
                         {
                                sprintf (str0, "Encapsulating pk(%d) sent"
                                       , op_pk_id (encap_pkptr));
                                sprintf (str1, "Destination: net (%d), node (%d)"
                                       , multi_bind_entry->careof.net
                                       , multi_bind_entry->careof.node );
                                op_prg_odb_print_major (str0, str1, OPC_NIL);
                         }

                         encap_pk_send( encap_pkptr );
                  }

                  /* At this stage, it is assumed that HA binding exists
                        * for mobile node. Therefore send binding warning
                   * message to original sender */
                  if ( route_optim == true )
```

```
                {
                        op_pk_nfd_get( pkptr, "src_net", &orig.net );
                        op_pk_nfd_get( pkptr, "src_node", &orig.node );

                        generate_bind_warning( orig, reg_objid
                                , home_entry->home_addr, current );
                }
        }
    }

    if ( status == OPC_COMPCODE_SUCCESS )
    {
        if ( copy )
                op_pk_destroy( pkptr );
        FRET( status )
    }

#if 0
    if ( route_optim == false )
        FRET( OPC_COMPCODE_FAILURE )
#endif

    num_bind = op_prg_list_size( CA_bind_ptr );

    for( i=0; i<num_bind; ++i )
    {
        ca_entry = (CA_mobility_binding *)
                        op_prg_list_access( CA_bind_ptr, i );

        if ( IP_equal( dest, ca_entry->home_addr ) )
        {
                status = OPC_COMPCODE_SUCCESS;

                data_len = op_pk_total_size_get( pkptr )/8;
                encap_pkptr = op_pk_create_fmt( "ip_dgram" );
                op_pk_bulk_size_set( encap_pkptr, data_len*8 );

                op_pk_nfd_set( encap_pkptr, "data", pkptr );
                op_pk_nfd_set( encap_pkptr, "protocol"
                                , PROTOCOL_ENCAP );
                op_pk_nfd_set( encap_pkptr, "src_net", current.net );
                op_pk_nfd_set( encap_pkptr, "src_node", current.node );
                op_pk_nfd_set( encap_pkptr, "dest_net"
                                , ca_entry->careof.net );
                op_pk_nfd_set( encap_pkptr, "dest_node"
                                , ca_entry->careof.node );
                op_pk_nfd_set( encap_pkptr, "orig_len", data_len );
                op_pk_nfd_set( encap_pkptr, "frag_len", data_len );
                op_pk_nfd_set( encap_pkptr, "ident", (*pk_id)++ );
                op_pk_nfd_set( encap_pkptr, "frag", 0 );
                op_pk_nfd_set( encap_pkptr, "ttl", ttl );

                /* Now schedule packet for transmission */
                encap_pk_send( encap_pkptr );
                FRET( OPC_COMPCODE_SUCCESS )
        }
    }

    if ( copy )
        op_pk_destroy( pkptr );

FRET( status )
}
```

```
void
deliver_packet( pkptr, ici_ptr, next_net, next_node, outstrm, mtu )
Packet                  *pkptr;
Ici                     *ici_ptr;
int                     next_net, next_node, outstrm, mtu;
{
char        str0[512], str1[512];
int         i, len;
int         header_size, frag_size, data_size;
int         dest_net, dest_node;
int         ttl;
int         frag_accum, frag, num_frags;
Packet      *ip_pkptr, *data_pkptr, *frag_ptr;

FIN( deliver_packet( pkptr, ... ) )

    /* obtain packet's destination */
    op_pk_nfd_get( pkptr, "dest_net", &dest_net );
    op_pk_nfd_get( pkptr, "dest_node", &dest_node );

    /* Decrement the packet's time-to-live field. If zero is reached, */
    /* discard the packet rather than send it on. */
    op_pk_nfd_get (pkptr, "ttl", &ttl);
    ttl--;
    if (ttl == 0)
    {
           /* In debug mode, indicate that a packet is destroyed */
           /* due to an expired ttl. */
           if (op_prg_odb_ltrace_active ("ip_errs"))
           {
                  sprintf (str0, "Discarding packet (%d) with expired TTL", op_pk_id (pkp-
tr));
                  sprintf (str1, "Destination: net (%d), node (%d)", dest_net, dest_node);
                  op_prg_odb_print_major (str0, str1, OPC_NIL);
           }
           op_pk_destroy (pkptr);
    }
    else{
           /* Assign the new decremented value of ttl. */
           op_pk_nfd_set (pkptr, "ttl", ttl);

           /* In debug mode, trace the routing action. */
           if (op_prg_odb_ltrace_active ("mobile-ip_rte"))
           {
                  sprintf (str0, "Routing towards (%d, %d)", dest_net, dest_node);
                  sprintf (str1, "Next hop (%d, %d), output stream (%d)",
                                        next_net, next_node, outstrm);

                  op_prg_odb_print_major (str0, str1, OPC_NIL);
           }

           /* Install an Ici indicating to the lower layer what the */
           /* address of the next (intermediate) node is. */
           op_ici_attr_set (ici_ptr, "next_node", next_node);
           op_ici_install (ici_ptr);

           /* Obtain the size in bytes of the fragment. */
           frag_size = op_pk_total_size_get (pkptr) / 8;

           /* Obtain the number of bytes of data carried in this fragment */
           op_pk_nfd_get (pkptr, "frag_len", &data_size);

           /* Also obtain the difference between the packet size */
           /* and the length field: this is the size of the header. */
           header_size = frag_size - data_size;
```

```
/* If it is smaller than the maximum transfer unit, send it as is. */
if (frag_size <= mtu)
{
        op_pk_send (pkptr, outstrm);
}
else{
        /* Otherwise, break it into fragments */
        /* Each fragment can contain up to (mtu - header_size) bytes of data */
        num_frags = (data_size + mtu - header_size - 1) / (mtu - header_size);

        /* In debug mode, indicate the fragmentation. */
        if (op_prg_odb_ltrace_active ("ip_frag"))
        {
                sprintf (str0, "Breaking datagram into (%d) fragments", num_frags);
                op_prg_odb_print_major (str0, OPC_NIL);
        }

        /* If the fragment is carrying the original datagram given to IP, */
        /* extract it before copies are made. Only one fragment can carry */
        /* the original packet for the reassembly model to work properly. */
        if (op_pk_nfd_is_set (pkptr, "ip_dgram"))
                op_pk_nfd_get (pkptr, "ip_dgram", &ip_pkptr);
        else ip_pkptr = OPC_NIL;

        /* If the packet is carrying any encapsulated data (normally this */
        /* would happen only for a packet fragmented for the first time), */
        /* extract this data packet so that it will not appear in each */
        /* fragment generated by copying. */
        if (op_pk_nfd_is_set (pkptr, "data"))
                op_pk_nfd_get (pkptr, "data", &data_pkptr);
        else data_pkptr = OPC_NIL;

        /* Loop through and create the fragments . */
        for (frag_accum = 0, i = 0; i < num_frags; i++)
        {
                /* Make a copy of the original packet. */
                frag_ptr = op_pk_copy (pkptr);

                /* Indicate that the copy is a fragment */
                op_pk_nfd_set (frag_ptr, "frag", 1);

                /* for all but the last fragment, the size is the mtu. */
                /* and the encapsulated ip packet is not included. */
                if (i < num_frags - 1)
                {
                        op_pk_nfd_set (frag_ptr, "frag_len", mtu - header_size);
                        op_pk_total_size_set (frag_ptr, 8 * mtu);
                        frag_accum += (mtu - header_size);
                }
                else{
                        len = data_size - frag_accum;
                        op_pk_nfd_set (frag_ptr, "frag_len", len);
                        op_pk_total_size_set (frag_ptr,8 * (header_size + len));

                        /* If the original packet was not a fragment, encapsulate it
*/
                        /* into the last fragment created here. */
                        op_pk_nfd_get (pkptr, "frag", &frag);
                        if (!frag)
                        {
                                /* If the packet contained encapsulated data (i.e.,
from the */
                                /* transport), that data will have been removed to
avoid */
```

```
                                        /* its duplication in the fragments. The data should
now be */
                                        /* reinserted into the original packet. */
                                        if (data_pkptr != OPC_NIL)
                                                op_pk_nfd_set (pkptr, "data", data_pkptr);

                                        /* In either case the original packet is */
                                        /* encapsulated in thhe fragment. */
                                        op_pk_nfd_set (frag_ptr, "ip_dgram", pkptr);
                                }

                                /* Otherwise the packet can be discarded. */
                                else{
                                        op_pk_destroy (pkptr);

                                        /* Also, if the packet included the original datagram
*/
                                        /* from which it was generated, transfer that data-
gram */
                                        /* into the last fragment created here. */
                                        /* Note that it is possible, in the case where a
fragment */
                                        /* is itself being fragmented, that none of the cre-
ated */
                                        /* fragments will contain the original datagram. */
                                        if (ip_pkptr != OPC_NIL)
                                        {
                                                op_pk_nfd_set (frag_ptr, "ip_dgram",
ip_pkptr);
                                        }
                                }
                        }

                        /* Forward the datagram fragment. */
                        op_pk_send (frag_ptr, outstrm);
                }
        }
    }
FOUT;
}



Compcode
forward_to_visitor( mobile_table, bind_list_ptr, pkptr, ici_ptr, dest )
List                *bind_list_ptr, *mobile_table;
Packet              *pkptr;
Ici                 *ici_ptr;
IP                  dest;
{
char                str0[80], str1[80];
Compcode    status = OPC_COMPCODE_FAILURE;
int                 i, size;
int                 next_net, next_node, stream, mtu;
FA_mobility_binding*visitor_entry;

FIN( forward_to_visitor( mobile_table, bind_list_ptr, pkptr, ici_ptr, dest ) )

    size = op_prg_list_size( bind_list_ptr );

    if ( size == 0 )/* list does not exist */
            FRET ( OPC_COMPCODE_FAILURE )

    for( i=0; i<size; ++i )
```

```
        {
                visitor_entry = (FA_mobility_binding *)
                                op_prg_list_access( bind_list_ptr, i );

                if ( !memcmp( &dest, &visitor_entry->home_addr, sizeof(IP) ) )
                {
                        /* There is a match */
                        if ( visitor_entry->home_agnt.net != ADDRESS_UNDEFINED &&
                             visitor_entry->home_agnt.node != ADDRESS_UNDEFINED )
                        {
                                status = OPC_COMPCODE_SUCCESS;
                                next_net = visitor_entry->home_addr.net;
                                next_node = visitor_entry->home_addr.node;
                                stream = visitor_entry->stream;
                                mtu = get_strm_mtu( mobile_table, stream );
                                if ( mtu == MOBILE_STRM_NONEXISTENT )
                                {
                                        sprintf (str0, "Discarding packet (%d) ", op_pk_id (pkptr));
                                        sprintf (str1, "Stream non-existent" );
                                        op_prg_odb_print_major (str0, str1, OPC_NIL);
                                        op_pk_destroy( pkptr );
                                }
                                else
                                {
                                        deliver_packet( pkptr, ici_ptr
                                                        , next_net, next_node, stream, mtu );
                                }
                        }
                }
        }

    FRET( status );
    }


    int
    get_strm_mtu( strm_ptr, strm )
    List*strm_ptr;
    int strm;
    {
    int                 i, size;
    mobile_rte_table*strm_entry;

    FIN( get_strm_mtu( strm_ptr, strm ) )

        size = op_prg_list_size( strm_ptr );

        for( i=0; i<size; ++i )
        {
                strm_entry = (mobile_rte_table *) op_prg_list_access( strm_ptr, i );
                if ( strm_entry->stream == strm )
                        FRET( strm_entry->mtu );
        }

    FRET( MOBILE_STRM_NONEXISTENT );
    }



    int
    chk_strm_condition( mobile_table, stream, mtu )
    List        *mobile_table;
    int         stream;
    int         *mtu;
    {
```

```
int                 i, size;
mobile_rte_table*table_entry;
FIN( chk_strm_condition( mobile_table, stream, mtu ) )

    size = op_prg_list_size( mobile_table );

    for( i=0; i<size; ++i )
    {
            table_entry = ( mobile_rte_table * )
                            op_prg_list_access( mobile_table, i );

            if( table_entry->stream == stream )
            {
                    *mtu = table_entry->mtu;
                    FRET( table_entry->condition )
            }
    }


    /* stream does not exist */
    *mtu = MOBILE_STRM_NONEXISTENT;

FRET( CONDITION_DISABLED )
}

void
get_careof_addr( mobile_table, iciptr )
List        *mobile_table;
Ici         *iciptr;
{
int                 i, size;
int                 stream;
IP                  careof;
mobile_rte_table*table_entry;
FIN( get_careof_addr( mobile_table, iciptr ) )

    op_ici_attr_get( iciptr, "stream", &stream );

    size = op_prg_list_size( mobile_table );
    for( i=0; i<size; ++i )
    {
            table_entry = ( mobile_rte_table * )
                            op_prg_list_access( mobile_table, i );

            if( table_entry->stream == stream )
            {
                    op_ici_attr_set( iciptr, "careof_net", table_entry->careof.net );
                    op_ici_attr_set( iciptr, "careof_node", table_entry->careof.node );
                    op_ici_attr_set( iciptr, "status", OPC_COMPCODE_SUCCESS );
                    FOUT
            }
    }


    /* stream does not exist */
    op_ici_attr_set( iciptr, "careof_net", ADDRESS_UNDEFINED );
    op_ici_attr_set( iciptr, "careof_node", ADDRESS_UNDEFINED );
    op_ici_attr_set( iciptr, "status", OPC_COMPCODE_FAILURE );

FOUT
}


void
set_strm_condition( mobile_table, iciptr )
```

```
List*mobile_table;
Ici *iciptr;
{
IP          careof;
int         stream;
int         mode;
int         i, size;
Compcodestatus = OPC_COMPCODE_FAILURE;
mobile_rte_table*list_ptr;
FIN( set_strm_condition( iciptr ) )

    op_ici_attr_get( iciptr, "mode", &mode );
    op_ici_attr_get( iciptr, "stream", &stream );

    size = op_prg_list_size( mobile_table );
    for( i=0; i<size; ++i )
    {
           list_ptr = (mobile_rte_table *)
                        op_prg_list_access( mobile_table, i );

           switch( mode )
           {
           case DISABLE_ALL:
                  list_ptr->condition = CONDITION_DISABLED;
                  status = OPC_COMPCODE_SUCCESS;
                  break;

           case DISABLE_ALL_EXCEPT_THIS:
                  if( list_ptr->stream != stream )
                  {
                        list_ptr->condition = CONDITION_DISABLED;
                        status = OPC_COMPCODE_SUCCESS;
                  }
                  break;

           case DISABLE_THIS_ONLY:
                  if( list_ptr->stream == stream )
                  {
                        op_ici_attr_set( iciptr, "careof_net", list_ptr->careof.net );
                        op_ici_attr_set( iciptr, "careof_node", list_ptr->careof.node );
                        list_ptr->condition = CONDITION_DISABLED;
                        status = OPC_COMPCODE_SUCCESS;
                  }
                  break;

           case ENABLE_ALL:
                  list_ptr->condition = CONDITION_ENABLED;
                  status = OPC_COMPCODE_SUCCESS;
                  break;

           case ENABLE_ALL_EXCEPT_THIS:
                  if( list_ptr->stream != stream )
                  {
                        list_ptr->condition = CONDITION_ENABLED;
                        status = OPC_COMPCODE_SUCCESS;
                  }
                  break;

           case ENABLE_THIS_ONLY:
                  if( list_ptr->stream == stream )
                  {
                        op_ici_attr_set( iciptr, "careof_net", list_ptr->careof.net );
                        op_ici_attr_set( iciptr, "careof_node", list_ptr->careof.node );
                        list_ptr->condition = CONDITION_ENABLED;
                        status = OPC_COMPCODE_SUCCESS;
```

```
                }
                break;

        default:
                status = OPC_COMPCODE_FAILURE;
                break;
        }
    }

    op_ici_attr_set( iciptr, "status", status );
FOUT
}

void
hop_to_next_strm( mobile_table, iciptr )
List        *mobile_table;
Ici         *iciptr;
{
int             j, i, size;
int             stream;
mobile_rte_table*table_entry;
FIN( hop_to_next_strm( mobile_table, iciptr ) )

    op_ici_attr_get( iciptr, "stream", &stream );

    size = op_prg_list_size( mobile_table );

    for ( i=0; i<size; ++i )
    {
        table_entry = ( mobile_rte_table * )
                      op_prg_list_access( mobile_table, i );

        if ( table_entry->stream == stream )
        {
            if ( ( j = i + 1 ) == size )
                j = 0;

            table_entry = ( mobile_rte_table * )
                op_prg_list_access( mobile_table, j );

            op_ici_attr_set( iciptr, "stream"
                        , table_entry->stream );

            op_ici_attr_set( iciptr, "careof_net"
                        ,table_entry->careof.net);

            op_ici_attr_set( iciptr, "careof_node"
                        ,table_entry->careof.node);

            /* Now, enabling this stream */
            table_entry->condition = CONDITION_ENABLED;

            op_ici_attr_set( iciptr, "status"
                        , OPC_COMPCODE_SUCCESS );

            FOUT
        }
    }
    op_ici_attr_set( iciptr, "status", OPC_COMPCODE_FAILURE );
FOUT
}

void
process_binding_warning( pkptr, iciptr, rcv_iciptr, ip_objid )
Packet          *pkptr;
```

```
Ici             *iciptr;
Ici             *rcv_iciptr;
Objid           ip_objid;
{
IP          home_addr, rem, target, home_agnt;
int         rem_port;
Packet          *bind_pkptr;
Compcode    status;
Ici             *bind_iciptr;
FIN( process_binding_warning( pkptr, iciptr, rcv_iciptr, ip_objid ) )

    op_pk_nfd_get( pkptr, "home_addr_net", &home_addr.net );
    op_pk_nfd_get( pkptr, "home_addr_node", &home_addr.node );

    op_pk_nfd_get( pkptr, "target_net", &target.net );
    op_pk_nfd_get( pkptr, "target_node", &target.node );

    op_ici_attr_get( rcv_iciptr, "rem_net",  &rem.net );
    op_ici_attr_get( rcv_iciptr, "rem_node", &rem.node );
    op_ici_attr_get( rcv_iciptr, "rem_port", &rem_port );

    /* creating binding for Mobile-ip query */
    bind_iciptr = op_ici_create( "binding_command" );

    op_ici_attr_set( bind_iciptr, "command", READ_HA_BINDING );
    op_ici_attr_set( bind_iciptr, "home_net", home_addr.net );
    op_ici_attr_set( bind_iciptr, "home_node", home_addr.node );

    op_ici_install( bind_iciptr );
    op_intrpt_force_remote( BINDING_MAINTENANCE, ip_objid );
    op_ici_install( OPC_NIL );

    op_ici_attr_get( bind_iciptr, "status", &status );

    if ( status == OPC_COMPCODE_SUCCESS )
    {
            /* send binding warning if home addr is a registered MH */
            bind_pkptr = op_pk_create_fmt( "bind_warn" );
            op_pk_nfd_set( bind_pkptr, "home_addr_net", home_addr.net );
            op_pk_nfd_set( bind_pkptr, "home_addr_node", home_addr.node );
            op_pk_nfd_set( bind_pkptr, "target_net", target.net );
            op_pk_nfd_set( bind_pkptr, "target_node", target.node );

            udp_app_send( iciptr, bind_pkptr, REG_REQUEST_PORT
                          , REG_REQUEST_PORT, target.net, target.node );
    }
    else
    {
#if 1
            /* first of all, check whether home addr is on cache list */
            op_ici_attr_set( bind_iciptr, "command", READ_CA_BINDING );

            op_ici_install( bind_iciptr );
            op_intrpt_force_remote( BINDING_MAINTENANCE, ip_objid );
            op_ici_install( OPC_NIL );

            op_ici_attr_get( bind_iciptr, "status", &status );

            if ( status == OPC_COMPCODE_SUCCESS )
            {
                    /* Home addr on cache list, send binding request to home addr */
                    op_ici_attr_get( bind_iciptr, "home_agnt_net", &rem.net );
                    op_ici_attr_get( bind_iciptr, "home_agnt_node", &rem.node );

                    bind_pkptr = op_pk_create_fmt( "bind_req" );
```

```
                op_pk_nfd_set( bind_pkptr, "home_addr_net", home_addr.net );
                op_pk_nfd_set( bind_pkptr, "home_addr_node", home_addr.node );

                /* send the registration packet */
                udp_app_send( iciptr, bind_pkptr, REG_REQUEST_PORT
                        , REG_REQUEST_PORT, rem.net, rem.node );
        }
        else
        {
                /* no binding found, send it to target */
                bind_pkptr = op_pk_create_fmt( "bind_req" );
                op_pk_nfd_set( bind_pkptr, "home_addr_net", home_addr.net );
                op_pk_nfd_set( bind_pkptr, "home_addr_node", home_addr.node );

                /* send the registration packet */
                udp_app_send( iciptr, bind_pkptr, REG_REQUEST_PORT
                        , REG_REQUEST_PORT, target.net, target.node );
        }
#else
        bind_pkptr = op_pk_create_fmt( "bind_req" );
        op_pk_nfd_set( bind_pkptr, "home_addr_net", home_addr.net );
        op_pk_nfd_set( bind_pkptr, "home_addr_node", home_addr.node );

        /* send the registration packet */
        udp_app_send( iciptr, bind_pkptr, REG_REQUEST_PORT
                , rem_port, rem.net, rem.node );
#endif
    }

    /* deallocate resources after usage */
    op_ici_destroy( bind_iciptr );
FOUT
}


void
process_binding_request( pkptr, iciptr, rcv_iciptr, objid )
Packet          *pkptr;
Ici             *iciptr;
Ici             *rcv_iciptr;
Objid           objid;
{
Packet      *bind_pkptr;
IP          home_addr;
IP          careof;
IP          rem;
int         rem_port;
Ici         *ip_iciptr;
int         status;
int         lifetime;
FIN( process_binding_request( pkptr, iciptr, rcv_iciptr, objid ) )

    op_pk_nfd_get( pkptr, "home_addr_net", &home_addr.net );
    op_pk_nfd_get( pkptr, "home_addr_node", &home_addr.node );

    op_ici_attr_get( rcv_iciptr, "rem_net",  &rem.net );
    op_ici_attr_get( rcv_iciptr, "rem_node", &rem.node );
    op_ici_attr_get( rcv_iciptr, "rem_port", &rem_port );

    ip_iciptr = op_ici_create( "binding_command" );
    op_ici_attr_set( ip_iciptr, "command", READ_HA_BINDING );
    op_ici_attr_set( ip_iciptr, "home_net", home_addr.net );
    op_ici_attr_set( ip_iciptr, "home_node", home_addr.node );
```

```
    op_ici_install( ip_iciptr );
    op_intrpt_force_remote( BINDING_MAINTENANCE, objid );
    op_ici_install( OPC_NIL );

    op_ici_attr_get( ip_iciptr, "status", &status );
    if ( status == OPC_COMPCODE_FAILURE )
    {
           careof = home_addr;/* no binding exists */
           lifetime = 0;
    }
    else
    {
           op_ici_attr_get( ip_iciptr, "careof_net", &careof.net );
           op_ici_attr_get( ip_iciptr, "careof_node", &careof.node );
           op_ici_attr_get( ip_iciptr, "lifetime", &lifetime );
    }

    /* deallocate ici pointer after use */
    op_ici_destroy( ip_iciptr );

    bind_pkptr = op_pk_create_fmt( "bind_update" );
    op_pk_nfd_set( bind_pkptr, "home_addr_net", home_addr.net );
    op_pk_nfd_set( bind_pkptr, "home_addr_node", home_addr.node );
    op_pk_nfd_set( bind_pkptr, "careof_net", careof.net );
    op_pk_nfd_set( bind_pkptr, "careof_node", careof.node );
    op_pk_nfd_set( bind_pkptr, "lifetime", lifetime );

    /* send binding request via UDP port */
    udp_app_send( iciptr, bind_pkptr, REG_REQUEST_PORT, rem_port, rem.net, rem.node );

FOUT
}

void
process_binding_update( pkptr, iciptr, rcv_iciptr, objid )
Packet              *pkptr;
Ici                 *iciptr;
Ici                 *rcv_iciptr;
Objid               objid;
{
Packet      *bind_pkptr;
IP          home_addr;
IP          careof;
IP          rem;
int         rem_port;
Ici         *ip_iciptr;
int         lifetime, ack, status;
FIN( process_binding_update( pkptr, iciptr, rcv_iciptr, objid ) )

    op_pk_nfd_get( pkptr, "home_addr_net", &home_addr.net );
    op_pk_nfd_get( pkptr, "home_addr_node", &home_addr.node );
    op_pk_nfd_get( pkptr, "careof_net", &careof.net );
    op_pk_nfd_get( pkptr, "careof_node", &careof.node );
    op_pk_nfd_get( pkptr, "lifetime", &lifetime );
    op_pk_nfd_get( pkptr, "ack", &ack );

    op_ici_attr_get( rcv_iciptr, "rem_net",  &rem.net );
    op_ici_attr_get( rcv_iciptr, "rem_node", &rem.node );
    op_ici_attr_get( rcv_iciptr, "rem_port", &rem_port );

    ip_iciptr = op_ici_create( "binding_command" );

    if ( (careof.net == 0 && careof.node == 0) || lifetime == 0 )
    {
           op_ici_attr_set( ip_iciptr, "command", KILL_CA_BINDING );
```

```
            op_ici_attr_set( ip_iciptr, "home_net", home_addr.net );
            op_ici_attr_set( ip_iciptr, "home_node", home_addr.node );

            op_ici_install( ip_iciptr );
            op_intrpt_force_remote( BINDING_MAINTENANCE, objid );
            op_ici_install( OPC_NIL );

        }
        else
        {
            op_ici_attr_set( ip_iciptr, "command", EDIT_CA_BINDING );
            op_ici_attr_set( ip_iciptr, "home_net", home_addr.net );
            op_ici_attr_set( ip_iciptr, "home_node", home_addr.node );
            op_ici_attr_set( ip_iciptr, "home_agnt_net", rem.net );
            op_ici_attr_set( ip_iciptr, "home_agnt_node", rem.node );
            op_ici_attr_set( ip_iciptr, "careof_net", careof.net );
            op_ici_attr_set( ip_iciptr, "careof_node", careof.node );
            op_ici_attr_set( ip_iciptr, "lifetime", lifetime );

            op_ici_install( ip_iciptr );
            op_intrpt_force_remote( BINDING_MAINTENANCE, objid );
            op_ici_install( OPC_NIL );

            if ( ack == true )
            {
                bind_pkptr = op_pk_create_fmt( "bind_ack" );
                op_pk_nfd_set( bind_pkptr, "home_addr_net", home_addr.net );
                op_pk_nfd_set( bind_pkptr, "home_addr_node", home_addr.node );

                /* send binding request via UDP port */
                udp_app_send( iciptr, bind_pkptr, REG_REQUEST_PORT
                              , rem_port, rem.net, rem.node );
            }
        }

        /* deallocate ici pointer after use */
        op_ici_destroy( ip_iciptr );

    FOUT
    }

    #if 0
    void
    encap_pk_destroy( pkptr )
    Packet      *pkptr;
    {
    Packet      *outer_pkptr, *inner_pkptr;
    int         i, num_fds, data_present;
    char        fd_name[40];
    FIN( encap_pk_destroy( pkptr ) )

        outer_pkptr = pkptr;


        while ( 1 )
        {
            num_fds = op_pk_fd_max_index( outer_pkptr );

            for ( data_present=false,i=0; i< num_fds; ++i )
            {
                op_pk_fd_index_to_name( outer_pkptr, i, fd_name );
                if ( strcmp( fd_name, "data" ) == 0 )
                {
                    op_pk_nfd_get( outer_pkptr, "data", &inner_pkptr );
                    op_pk_destroy( outer_pkptr );
```

```
                        data_present = true;
                        break;
                }
        }

        /* no fields with name "data" */
        if ( data_present == false )
                break;

        outer_pkptr = inner_pkptr;
    }

    op_pk_destroy( outer_pkptr );
FOUT
}
#endif


void
generate_bind_warning( dest, reg_objid, home_addr, target )
IP                  dest, home_addr, target;
Objid               reg_objid;
{
Ici         *warn_iciptr;
FIN( generate_bind_warning( home_addr, ) )

    warn_iciptr = op_ici_create("bind_warn_ici");
    op_ici_attr_set( warn_iciptr, "home_addr_net", home_addr.net );
    op_ici_attr_set( warn_iciptr, "home_addr_node", home_addr.node );

    op_ici_attr_set( warn_iciptr, "dest_net", dest.net );
    op_ici_attr_set( warn_iciptr, "dest_node",dest.node );

    op_ici_attr_set( warn_iciptr, "target_net", target.net );
    op_ici_attr_set( warn_iciptr, "target_node",target.node );

    op_ici_install( warn_iciptr );
    op_intrpt_force_remote( BINDING_WARN_TYPE, reg_objid );
    op_ici_install( OPC_NIL );

    op_ici_destroy( warn_iciptr );
FOUT
}


void
encap_pk_send( pkptr )
Packet      *pkptr;
{
FIN( encap_pk_send( pkptr ) )

    /* insert encapsulated packet at the beginning of the queue */
/*  op_subq_pk_insert( 0, pkptr, OPC_QPOS_HEAD );*/
    op_subq_pk_insert( 0, pkptr, OPC_QPOS_HEAD );
FOUT
}


void
check_ca_list( list_ptr, dest, home_agnt )
List        *list_ptr;
IP          dest;
IP          *home_agnt;
{
```

```
CA_mobility_binding*ca_entry;
int                i, num_bind;
FIN( check_ca_list( list_ptr, dest, home_agnt ) )

        num_bind = op_prg_list_size( list_ptr );

        for( i=0; i<num_bind; ++i )
        {
                ca_entry = (CA_mobility_binding *)
                                op_prg_list_access( list_ptr, i );

                if ( IP_equal( dest, ca_entry->home_addr ) )
                {
                   home_agnt->net  = ca_entry->home_agnt.net;
                   home_agnt->node = ca_entry->home_agnt.node;
                   FOUT
                }
        }

    home_agnt->net  = ADDRESS_UNDEFINED;
    home_agnt->node = ADDRESS_UNDEFINED;
FOUT
}


Compcode
insert_pk_buffer( buffer_ptr, buffer_life, dest, pkptr )
List       *buffer_ptr;
double     buffer_life;
IP         dest;
Packet     *pkptr;
{
int                      i, size;
packet_buffer*entry_ptr;
List               *packet_list;
Ici                      *iciptr;
FIN( insert_pk_buffer( buffer_ptr, buffer_life, dest, pkptr ) )

    if ( buffer_life <= 0.0 )
           FRET( OPC_COMPCODE_FAILURE )

    size = op_prg_list_size( buffer_ptr );
    for ( i=0; i<size; ++i )
    {
        entry_ptr = (packet_buffer *) op_prg_list_access( buffer_ptr, i );
        if ( IP_equal( dest, entry_ptr->dest ) )
        {
               packet_list = entry_ptr->cache_list;
               op_prg_list_insert( packet_list, pkptr, OPC_LISTPOS_TAIL );

               FRET( OPC_COMPCODE_SUCCESS )
        }
    }

    /* no buffer for this destintation yet */
    FRET( OPC_COMPCODE_FAILURE )
}
```