INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

**ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

UMI[®]

Implementation of Location Detection, Home Agent Discovery and Registration of Mobile IPv6

By
Yang Xu. B.Sc

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

August 20, 2001

© Copyright
2001, Yang Xu. B.Sc

**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-66872-X

Canada

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis,

Implementation of Location Detection, Home Agent Discovery and
Registration of Mobile IPv6

submitted by

Yang Xu. B.Sc

in partial fulfilment of the requirements for
the degree of Master of Computer Science

Dr. Frank Dehne
(Director, School of Computer Science)

Dr. Michel Barbeau
(Thesis Supervisor)

Carleton University
August 29, 2001

Abstract

With the technological growth of computer and wireless terminals, it is necessary to enhance the IP suite to cope with the roaming of IP hosts. Mobile IP (MIP) is proposed to support the mobility of IP hosts, from network to network, without changing their IP address. The goal of Mobile IP is to let an IP host continually receive IP packets sent to its IP address, keep the undergoing IP connections and make the movement of IP hosts transparent up to transport layer protocols and applications. Due to the differences between IPv4 and IPv6, IETF defines separate protocol standards and system structures for Mobile IPv4 and Mobile IPv6. Mobile IPv6 is currently a work in process and needs to be validated over different Operating System and network environment. The implementation of Mobile IP is substantially dependent on the Operating System(OS) that is used by an IP host. Even with the same OS, the approaches of implementation can be substantially different.

The thesis focuses on the implementation of Mobile IPv6 over Linux. Its goal is to verify and provide feedback for the current IETF proposal of Mobile IPv6. This thesis attempts one of the approaches of Mobile IPv6 implementation over Linux. In this approach, part of the implementation code stays in the Linux kernel space. The remaining code stays in the Linux application space. The system partially implements the Mobile IPv6 functionality. During the implementation, the author also met some issues about the network programming over Linux, such as the data exchange between the kernel and application space. This thesis discusses approaches and solutions for these issues.

Acknowledgments

I would like to dedicate this thesis to the people who died in China's democracy campaign in 1989 and their families. To the students who forever lost the chance to finish their education and to all the people who cannot see tomorrow's sunrise. To the children, husbands, wives, and parents who lost their loved ones.

"Let tomorrow remember today's outcry. Let the whole world see the wound of history." - Anonymous

It is my honour to be supervised by Prof. Michel Barbeau with his kind help and direction. As a part-time student, it would be impossible for me to finish this thesis without the support and patience of my wife, Marina Zhang. I greatly appreciate Prof. Barbeau and Marina Zhang's effort and contribution to this thesis.

Contents

Abstract	iii
Acknowledgments	iv
1 Introduction	1
1.1 IP networking	1
1.2 Mobile IP	3
1.3 Motivation	4
1.4 Objective	5
1.5 Overview	6
2 Mobile IP via IPv6	7
2.1 IPv6 Overview	7
2.2 Mobile IP Network and Components	10
2.3 Data Process and Flow	12
2.4 Mobile Node Location Detection	15
2.5 Care-of Address Obtaining	16
2.6 Home Agent Discovery	17
2.7 Binding Management	19
2.8 Traffic Handling	21
2.9 Security	22
2.10 Review of Current Research and Development	23

3	Mobile IPv6 via Linux	27
3.1	Linux Overview	27
3.1.1	IP Traffic Flow in Linux	28
3.1.2	Process and Thread	29
3.1.3	Data Exchange between Applications and Kernel	30
3.2	Models of Mobile IPv6 Implementation in Linux	31
3.2.1	Application Layer Model	31
3.2.2	Kernel Layer Model	32
3.2.3	Application and Kernel Layer Model	34
4	Implementation and Experiment	36
4.1	Design Model and Environment	36
4.2	System Functionality	37
4.3	Threads	38
4.4	ICMPv6 Message Capture	38
4.5	Care-of Address Configuration	41
4.6	Security	42
4.7	Binding Message Capture	43
4.8	ICMPv6 Message Sending	48
4.9	Binding Message Sending and Structure	49
4.10	Data Exchange between Application Space and Kernel Space	53
4.11	Mobile Node	54
4.11.1	System Structure	54
4.11.2	Data Structure	56
4.11.3	Control Flow	57
4.12	Home Agent	60
4.12.1	System Structure	60
4.12.2	Data Structure	62
4.12.3	Control Flow	64
4.13	Correspondent Node	66
4.13.1	System Structure	66

4.13.2 Data Structure	67
4.13.3 Control Flow	67
4.14 Experiment	68
4.14.1 Testing Environment	68
4.14.2 Test Procedure and Result	69
5 Conclusion	72
5.1 Summary	72
5.2 Contributions	73
5.3 Future Work	74
Bibliography	77

List of Figures

1.1	Internet Protocol Model	1
2.1	IPv6 Header Link	7
2.2	Router Solicitation	9
2.3	Router Advertisement	9
2.4	Sample of Network	12
2.5	Mobile IPv6 Data Flow	15
2.6	ICMPv6 Home Agent Discovery Request	18
2.7	ICMPv6 Home Agent Discovery Reply	19
2.8	Binding Update	20
2.9	Binding Acknowledge	20
2.10	Binding Request	21
2.11	Authentication Header	23
3.1	IP stack	28
3.2	Application Layer Model	32
3.3	Kernel Layer Model	33
3.4	Application and Kernel Layer Model	34
4.1	Mobile Node Structure	55
4.2	Mobile Node: Process Binding Request	58
4.3	Mobile Node: Process Binding Acknowledgement	59
4.4	Mobile Node: Process ICMPv6 Router Advertisement	60
4.5	Mobile Node: Process ICMPv6 Home Agent Discovery Reply	61

4.6	Mobile Node: Trace Router Advertisement	62
4.7	Mobile Node: Trace Binding Update List	63
4.8	Mobile Node: Trace Binding Acknowledge	64
4.9	Home Agent: Process Binding Update	65
4.10	Home Agent: Process Home Agent Discovery Request	66
4.11	Home Agent: Trace Binding Registration	67
4.12	Correspondent Node Structure	68
4.13	Testing Environment	69

Chapter 1

Introduction

1.1 IP networking

The Internet uses a layered set of protocols to facilitate the communication between IP hosts. The Internet protocols are modeled in four layers:

Figure 1.1: Internet Protocol Model

Application layer is a user process cooperating with another process on the same or a different host in application layer, such as TELNET (a protocol for remote terminal connections), FTP (File Transfer Protocol), and SMTP (Simple Mail Transfer

Protocol). Transport layer provides the end-to-end data transfer. Example protocols are TCP (connection-oriented) and UDP (connectionless).

Internetwork layer also called the internet layer or the network layer. It provides the virtual network image of the Internet. Internet Protocol (IP) is the most important protocol in this layer. It is a connectionless protocol which doesn't assume reliability from the lower layers. Each IP entity is identified by a unique IP address. An IP address consists of two portions: network prefix and host ID/number. The network prefix portion defines the network on which an address resides. IP does not provide reliability, flow control or error recovery. These functions must be provided at a higher layer, either at the Transport layer by using TCP as the transport protocol, or at the Application layer if UDP is used as the transport protocol.

The network interface is also called the link layer or the data-link layer, the network interface layer is the interface to the actual network hardware. This interface may or may not provide reliable delivery, and may be packet or stream oriented. In fact, IP does not specify any protocol here, but can use almost any network interface available, which illustrates the flexibility of the IP layer. Examples are IEEE 802.2, X.25 (which is reliable in itself), ATM, FDDI, Packet Radio Networks (such as the AlohaNet).

A message unit in an IP network is called an IP packet. IP packets transmit the information being delivered over the Internet. Each IP packet contains the IP addresses of this packet's source and destination hosts. These addresses are used by Internet routers to direct the IP packet toward the destination. When an IP packet travels over the Internet, its route selection is controlled by routing protocol. The routers, receiving an IP packet, use the network prefix portion of the packet's destination IP address to decide its routing. This means that IP routing is network prefix based instead of host-specific based.

There are currently two versions of IP, IPv4 and IPv6. IPv4 is the version that is in use today. But it has some disadvantages such as the relatively small IP address space. IPv6 has some enhancements over IPv4 one of which is 128 bits for each

IP address rather than 32 bits that IPv4 uses. Other enhancements in IPv6 include enhanced routing capabilities, simplified header format, improved support for options, flow label, and transmission mechanisms.

1.2 Mobile IP

This section briefly describe Mobile IP and its motivation.

As we discussed in the previous section, IP routing is network prefix based instead of host-specific based. When an IP host is connected to a network, whose network prefix is different from the host's home network, all IP packets are still forwarded to the IP host's home network by routers according to the network prefix portion of its IP address, instead of to the network to which the host currently attaches. In this situation, the IP host cannot receive the IP packets that are destined for it. To continue the Internet communication, this IP host needs to reconfigure its IP address. When changing the IP address automatically or manually, the following problems have to be considered:

- Some application configurations rely on the host IP address. When changing the host IP address, the applications need to be reconfigured.
- During reconfiguration, the IP connections related to the previous IP address are lost.
- All correspondent nodes must be informed of the reconfiguration.
- When the host moves often in the Internet, such as changing Internet attachments every five minutes, manual re-configuration becomes impossible.

These problems, and many others not mentioned here, make it impossible for a mobile host to change its IP address while it is roaming, without a mechanism that supports the mobility in IP.

Mobile IP specifies enhancements that allow transparent routing of IP traffic to mobile nodes in the Internet without changing the configuration of applications. With Mobile IP, each mobile node is always identified by its home address, regardless of its current point of attachment to the Internet, even when it moves out from its home network and roams in the Internet. The roaming or location of the mobile node is transparent to the upper-layer applications, such as Web browser. The applications do not need to be aware of the roaming and location change of the mobile node. With Mobile IP, the IP sessions can be maintained in spite of the physical movements between points of attachment to the Internet and the frequency of the movements. Overall, the goal of Mobile IP is to make the mobility of IP hosts transparent to the transport layer of the IP stack.

The Mobile IP proposal was approved by the Internet Engineering Steering Group (IESG) in 1996 and was produced by the IP Routing for Wireless/Mobile Hosts working group of the Internet Engineering Task Force (IETF). IETF has defined separate standards for Mobile IPv6 and Mobile IPv4, which have totally different system structures and protocols. Currently, the Mobile IPv6 is still a work in process [11].

1.3 Motivation

This thesis selects Mobile IPv6 as the research direction and Linux as the platform for Mobile IPv6 implementation.

Since Mobile IPv6 proposal is still a work in process, IETF is asking comments about it. The proposal needs to be verified and validated in different environments by independent party. Compared to Mobile IPv4, Mobile IPv6 is a relative newcomer and less work is being done in this area.

With the advantages of IPv6, Mobile IPv6 is further optimized compared to Mobile IPv4. Due to the larger IP address space in IPv6, it becomes possible for a mobile node to auto-config its care-of address. In Mobile IPv6, control messages are carried in IPv6 destination headers. With this mechanism, control messages can be transmitted

in any IP packet. Some other system behaviors are also enhanced in Mobile IPv6, such as built-in route optimization and security support. All these features need to be verified in different network and Operating System(OS) environment.

The implementation of Mobile IP is substantially related to the OSs used by Mobile IP nodes and varies with different OS platforms. Within the same Operating System, different approaches of implementation have different aspects such as system structure and performance.

Due to the advantages of Unix, such as good network programming support, Unix or Unix clones, such as Free BSD and Linux, are often selected as networking experiment platforms. Although Unix network program APIs in the application space are almost standard, different Unix systems have very different internal structures in the kernel space. Since the implementation of Mobile IP on Unix cannot avoid enhancing the kernel behavior, the implementation approaches on Unix systems are substantially different depending on the Unix internal structure. Linux is a clone of Unix and the most popular Unix OS on PCs and laptops. Linux provides the standard Unix network programming APIs in the application space. The OS source code is free and open. This advantage makes it easy for programmers to enhance the Linux kernel for the implementation of Mobile IP.

1.4 Objective

The main objective of the thesis is to design an approach to Mobile IPv6 over Linux, partially implement the Mobile IPv6 features, solve some technical issues related to the implementation and provide a basis for future work.

There are many possible approaches to Mobile IPv6 implementations over Linux. We intend to describe and compare their system and data structure, select one of the approaches as the implementation model, and provide a sample of Mobile IPv6 implementation.

This thesis focuses on the control part of the Mobile IPv6, including location

detection, binding management, and home agent discovery. This thesis also intends to solve some technical issues for implementation such as binding message capture. Thus it establishes a basis for future research, such as IP traffic tunneling/forwarding.

The work presented in this thesis is based on the current Mobile IPv6 standard draft [11]. Linux 2.2.14 was used for stability considerations.

1.5 Overview

To fulfill the objective, Chapter 2 describes the Mobile IPv6 network, protocol, data flow and message formats. It helps explain the Mobile IP system and its implementation. We also briefly review work done by other people on Mobile IPv6. Chapter 3 introduces some concepts and mechanisms of Linux network programming, such as IP flow in Linux. We also compare and analyse some other possible approaches of implementation of Mobile IPv6 over Linux. After that, Chapter 4 selects one of the approaches described in Chapter 2 and describes the implementation in detail. We give solutions of some technical concerns for implementation. We also describe the testing procedure, testing environment, and testing result for our implementation. Chapter 5 contains the conclusion of this thesis and suggestions for future work.

Chapter 2

Mobile IP via IPv6

2.1 IPv6 Overview

This section highlights some characteristics of IPv6 related to Mobile IPv6 and lays the groundwork for later discussion of Mobile IPv6.

In the base header of an IPv6 packet, there is a field pointing to extension headers or higher-layer protocol headers. Each extension header also has a similar field pointing to other headers. Several extension headers can appear in a single IPv6 packet. These headers form a header chain in an IPv6 packet. Figure 2.1 shows this structure.

Figure 2.1: IPv6 Header Link

Among these extension headers, there are three types of extension headers related to Mobile IPv6: destination option header, authentication header, and routing header. A destination option header contains the information that is examined only by the ultimate destination. Mobile IPv6 uses this portion to carry control messages. Authentication header provides authentication and protection, integrity checking, and non-repudiation of the payload. Mobile IPv6 uses this portion to protect some of the control messages. The Routing header is used by an IPv6 source to list one or more intermediate nodes to be visited on the way to a packet's destination. This portion is used in Mobile IP to forward or tunnel IP packets to mobile nodes.

According to IPv6 Addressing Architecture [RFC 1884], there are three types of IPv6 addresses: unicast, multicast and anycast address. A packet with a unicast address is delivered to the node which has the unicast address assigned to its interface. A packet with a multicast address is delivered to every node that is a member of the group identified by the multicast address. An anycast packet is delivered to any one of several possible nodes that are identified by the address, typically the node that is closest to the sender of the packet.

The main purpose of the Internet Control Message Protocol (ICMP) is to provide error notification and some low-level network testing mechanisms. ICMP messages are generated by a host or router on the route of an IP packet and sent to the packet's source if there are some problems related to this packet. Some new Internet protocols, such as the ICMP neighbour discovery protocol, extend the use of ICMP and transfer their messages over ICMP channels. IPv6 neighbour discovery contains a set of functions that relate to how nodes discover and interact with neighbours on their current link. It includes two important functions used by Mobile IPv6, router discovery and address auto-configuration.

The ICMP router discovery protocol is used by IP hosts to discover neighboring routers. There are two types of the ICMP router discovery messages: Router Advertisement and Router Solicitation. Each router periodically multicasts a Router Advertisement message from each of its multicast interfaces, announcing the IP address(es) of that interface. This message includes corresponding information about

that router, such as its IP address and life time. IP hosts discover their neighboring routers simply by listening to Router Advertisement messages. A host may multicast a Router Solicitation to ask for immediate Router Advertisement messages, rather than waiting for the next ones to arrive. Figure 2.2 and Figure 2.3 show these message formats.

Type = 133	Code = 0	Checksum
Reserved		
Options		

Figure 2.2: Router Solicitation

Type = 134	Code = 0			Checksum		
Hop Limit	M	O	Reserved		Router Lifetime	
Reachable Time						
Retrans Time						
Type=3	Length=4			Prefix Length	L	A Reserved
Valid Lifetime						
Preferred Lifetime						
Reserved						
Network Prefix						
Options						

Figure 2.3: Router Advertisement

The Dynamic Host Configuration Protocol (DHCP) is designed to centralize network administration by automatically assigning valid IP addresses and other configuration parameters to IP hosts. When a dynamically configured IP host connects to the network, it can get an IP address from the DHCP server in this network via DHCP messages. When this address is no longer used by the host, it is returned to the DHCP server to be reused. Although this mechanism does not solve Mobile IP

problems, such as retaining any ongoing connection while roaming, it still can be used by a mobile node to get its care-of address.

IP encapsulation is an Internet standard for forwarding or tunnelling IP packets. At the entry point of forwarding or tunnelling, the IP node at the entry point encapsulates an IP packet, headers and payload into another IP packet's payload. This process is called encapsulation. The new packet is sent to the exit point of forwarding or tunnelling. At the exit point, this packet is de-encapsulated [8]. In some cases, Mobile IPv6 uses this mechanism to forward or tunnel IPv6 packets to a mobile node.

2.2 Mobile IP Network and Components

For a mobile node, the network that has a network prefix matching this mobile node's home address, is called this mobile node's home network. Any other networks are considered to be this mobile node's foreign network.

The three important components in Mobile IPv6 are the mobile node, home agent and correspondent node.

- Mobile Node (MN)

A mobile node, which roams in the Internet, can be an wired device or a wireless terminal, such as a laptop, a cellular phone, or a Palm Pilot. These types of hosts may have several link layer interfaces to connect to the Internet. At any time, only one of these interfaces works as the primary interface for Mobile IP. The primary interface is used to determine the Internet attachment of a host. All incoming and outgoing control messages, such as binding messages, go through the primary interface. The tunnelled IP packets are sent to the primary interface until the interface is no longer valid.

From a mobile node's point of view, the network, whose network prefix equals the network prefix of this mobile node's home address, is defined as this mobile node's home network. Any other networks are defined as this mobile node's

foreign network.

- **Home Agent (HA)**

A home agent, which stays in the home network of a mobile node, manages the mobile node's location and other information. In a network there may be several hosts simultaneously working as home agents. Each home agent can predefine the mobile nodes that it is going to manage or provide service. It is also possible that a mobile node not is pre-assigned to a home agent. The mobile node can dynamically discover its home agent. In this case, a mechanism needs to be developed to synchronize the operation between the home agents. Anytime, only one home agent works for one mobile node.

In some Mobile IP implementations, the home agent is implemented on a router. This approach makes it easier to capture the incoming and outgoing traffic. It also reduces the amount of IP traffic going to the network. A home agent, however, is not necessarily a router and can be any host in the home network of a mobile node.

- **Correspondent Node (CN)**

A correspondent node, which can be anywhere in the Internet, sends IP packets to a mobile node. The packets sent by the correspondent node, which are ignorant of the mobile node's care-of address, are routed to the mobile node's home network. If the mobile node is out of its home network, these packets are intercepted and tunneled to the mobile node's care-of address by the mobile node's home agent. If the correspondent node knows the mobile node's care-of address, the IP packets are sent to the mobile node's care-of address directly.

Figure 2.4 shows an example of a Mobile IP network.

Figure 2.4: Sample of Network

2.3 Data Process and Flow

A mobile node traces its current location by listening to the ICMPv6 Router Advertisement messages. These messages are periodically multicasted by IPv6 routers. the message includes the network prefix of the network that the router exists and the mobile node attaches to. If a mobile node does not want to wait, it can multicast an ICMPv6 Router Solicitation message to force the routers on the network to reply ICMPv6 Router Advertisement messages immediately. If a mobile node finds that the network prefix in the Router Advertisement is different from the network prefix of

the network to which the mobile node attaches, it can be considered that the mobile node attaches or moved to a new network. If the new network prefix matches the previous network prefix, the Router Advertisement message can be used to update the router data that is stored in the mobile node, such as the validation time of the router. In some cases, there are several routers in one network. The mobile node may need to store the information for each router.

If a mobile node attaches to a new foreign network, this mobile node needs to form or obtain a care-of address with IPv6 or Mobile IP mechanism. There are several ways to achieve this goal: stateful, stateless, and static way. If a mobile node returns to its home network, it should start using its home address and its previous care-of address becomes invalid. A mobile node registers this type of change in its home agent by use of Binding Update message and it also informs selected correspondent nodes after getting confirmation of the Binding Update message from its home agent.

If a mobile node has no pre-assigned home agent, the mobile node needs to find out the home agent that is in its home network when the mobile node initially connects to the network. The mobile node can find out the home agent by requesting a Home Agent Discovery Request. Its destination address is an anycast address with a home network prefix. One of the home agents in the home network for this mobile node will send back an ICMPv6 Home Agent Discovery Reply message to inform the mobile node of the corresponding information. The message includes the information of all home agents that can act as a home agent for this mobile node.

When a home agent receives a Binding Update message, the home agent will decide whether or not to allow the mobile node to register. The confirmation needs to be sent back to the mobile node via a Binding Acknowledge message.

If a mobile node receives the Binding Acknowledge message and the message indicates that the home agent has refused the mobile node's binding registration, the mobile node needs to re-send its binding registration to an other home agent. If the mobile node receives the Binding Acknowledge message and the message indicates that its binding registration is accepted by the home agent, the mobile node needs to

send a Binding Update message to other correspondent nodes to inform them of its new location.

IPv6 packets sent by correspondent nodes, which are ignorant of the mobile node's care-of address, are routed to the mobile node's home network with the mobile node's home address, where the home agent tunnels them with IPv6 Encapsulation to the care-of address of the mobile node. If a correspondent node knows the mobile node's care-of address, IPv6 packets sent by this correspondent node are sent directly to the mobile node using an IPv6 Routing header, which specifies the mobile node's care-of address as an intermediate destination. In the reverse direction, packets sent by a mobile node are routed directly to their destination with the mobile node's care-of address or home address used as the source address in the packet.

After a mobile node returns to its home network, it uses a Binding Update message to de-register from its home agent and also notify other correspondent nodes with Binding Update messages.

Figure 2.5 shows the data flow and messages between the mobile node, home agent, correspondent node and router.

Among the messages transmitted between Mobile IPv6 components, Router Advertisement and Solicitation messages, Home Agent Discovery Request and Reply messages are transmitted via ICMPv6. The binding control messages (Binding Update, Binding Acknowledge, and Binding Request) are carried in the IPv6 Destination headers. Currently, there are no other IPv6 protocols or applications using the IPv6 Destination header to carry information. Since binding control messages are sent in IPv6 Destination headers, they can be in any IPv6 packets with other payload. To support Mobile IP, a Mobile IP host (mobile node, home agent, and correspondent node) has to check all the IPv6 packets and intercept those that have binding messages in IPv6 destination header.

Figure 2.5: Mobile IPv6 Data Flow

2.4 Mobile Node Location Detection

A mobile node detects its network attachment or location by listening to or soliciting an ICMPv6 Router Discovery message. ICMPv6 Router Discovery has two types of messages, Router Solicitation and Router Advertisement.

Router Advertisement is anycasted periodically by routers in the ICMPv6 format. The mobile node can listen to this type of message and check the network prefix option in the message. If the network prefix option matches the Home address prefix of the mobile node, the mobile node attaches to its home network. Otherwise the mobile

node attaches to a foreign network. A mobile node needs to keep track of the network prefix of the network to which it attaches. If the new network prefix does not match the previous one, the network to which the mobile node attaches is changed.

If the mobile node does not get any Router Advertisement for a certain time, the mobile node can consider its primary link interface is detached from the network. It needs to send a Router Solicitation message to force the routers to send Router Advertisements messages immediately instead of waiting until next period. When a mobile node is just powered on, it may keep sending Router Solicitation messages until it get a Router Advertisement message.

2.5 Care-of Address Obtaining

There are several ways for a mobile node to form or obtain its care-of address. From the mobile node's point of view, they are stateful, stateless, and static ways.

For some mobile nodes, there are several link layer interfaces that may attach to different networks at the same time. For every interface, there may be one unique care-of address. But there is only one that works as the primary care-of address for this mobile node at any time.

- Stateful

After a mobile node detects that it is attached to a foreign network, it can get a new care-of address via DHCP6 from the DHCP Server on the new network. When a mobile node no longer needs this address, it should return the address to the DHCP server or just let it expire.

- Stateless

A mobile node detects its location via a Router Advertisement message on the network to which it attaches. This Router Advertisement message includes the network Prefix Information Option. The mobile node can form a new care-of address with this information and the identical number of the link interface by

which it connects to this network. For example, this link identical number can be Ethernet address of the interface for a PC or laptop.

A new care-of address formed in that way may be duplicated in the network. Thus, the mobile node may need to perform Duplicate Address Detection. This process may take some time to be done and cause the delay causing the mobile node to be unable to use the new address. On the other side, the possibility of duplication for the new address, which is formed by the network prefix and link layer identity, is very unlikely. For example, the ethernet link address is supposed to be identical in the worldwide. If the the ethernet link address is used as a part of the new care-of address, this address should be also identical unless there is one defect ethernet device or interface in the same network. It is a kind of trade-off. The Mobile IPv6 allows a mobile node to use this new address without performing a Duplicate Address Detection.

- Static

In some cases, a mobile node may have been assigned an IPv6 address to be used only while it attaches to a foreign network. The Mobile IPv6 allows the mobile node use this pre-assigned IPv6 address instead of forming a new care-of address. In this case, the mobile node's home network and the foreign network to which the mobile node is connecting, must support host-specific routing for that care-of address.

2.6 Home Agent Discovery

There are two ways for a mobile node to obtain a home agent address: pre-assigned or dynamic discovery. These two methods may also be combined to form a new mechanism.

Under normal circumstances, a mobile node can be pre-assigned a home agent. This may cause problems if the home network or the home agent is reconfigured when the mobile node is away from the home network. When they are reconfigured,

the pre-assigned home agent becomes invalid and the new home agent needs to be re-assigned to the mobile node.

To avoid a pre-assigned home agent becoming invalid when the home network is reconfigured, the mobile node can send an ICMPv6 Home Agent Discovery Request message to its home network. The host, which can act as the home agent in the home network for this mobile node, can respond to this request by sending back an ICMPv6 Home Agent Discovery Reply message to inform the mobile node relevant information including all home agents that can act as a home agent for this mobile node.

There are two types of ICMPv6 Home Agent Discovery messages: ICMPv6 Home Agent Discovery Request and ICMPv6 Home Agent Discovery Reply. ICMPv6 Home Agent Discovery Request is anycasted from a mobile node to its home network. ICMPv6 Home Agent Discovery Reply is sent from the mobile node's home network to the mobile node. It may be sent by one of the home agents or hosts that manage the home agents in the mobile node's home network. Figures 2.6 and 2.7 show this message format.

Type	Code=0	Checksum
Identifier		
Reserved		
Home Address		

Figure 2.6: ICMPv6 Home Agent Discovery Request

In the Home Agent Discovery Reply message, the field Home Agent Address field is a list of addresses of home agents on the home link for the mobile node. The number of addresses present in the list is indicated by the remaining length of the IPv6 packet carrying the Home Agent Address Discovery Reply message.

Type	Code=0	Checksum
Identifier		
Reserved		
Home Agent Address		
More Home Agent Address		

Figure 2.7: ICMPv6 Home Agent Discovery Reply

2.7 Binding Management

The binding messages are sent between a mobile node and its home agent or correspondent node(s). These messages are sent in IPv6 destination headers along with any other messages in an IP packet. There are three types of binding messages: Binding Update, Binding Acknowledge, and Binding Request.

Binding Update is sent from a mobile node to its home agent and correspondent nodes. Figure 2.8 shows this message format. The Acknowledge (A) bit is set to request a Binding Acknowledgment be returned upon receipt of the Binding Update. The Home Registration (H) bit is set to request the receiving node to act as this node's home agent. The Router (R) bit, when set, indicates that the mobile node is a router. This bit is only valid when the Home Registration (H) bit is also set. The Duplicate Address Detection (D) bit is set to request the mobile node's home agent to perform Duplicate Address Detection on the mobile node's home link for the home address in this binding. The last part in a Binding Update message is its sub-options. One of these sub-options is the Alternate Care-of Address sub-option. It contains an address to be used as the care-of address for the binding. If a Binding Update does not contain an Alternate Care-of Address sub-option, the receiver can consider the source address of this IP packet as the new care-of address.

A Binding Acknowledge is sent as a confirmation of Binding Update. Normally, it

Next Hdr	Hdr Len	Option type = 198	Option Length
A H R D	Prefix Length	Sequence Number	
Life Time			
Sub-Options ...			

Figure 2.8: Binding Update

is from a home agent to a mobile node. It is only sent by the Binding Update receiver when the bit A is set in the Binding Update message. Figure 2.9 shows the message format. Values less than 128 in the Status field indicate whether or not the Binding Update was accepted by the receiving node. If the Binding Update is rejected, it also indicates the reason, such as insufficient resource or home registration not supported. A Binding Acknowledge message also includes the values of the Lifetime, in seconds, for which this node will attempt to retain the entry for this mobile node in its binding cache. After the value of the Refresh time, in seconds, the mobile node should send a new Binding Update to this node, the Binding Acknowledge message sender, in order to refresh the mobile node's binding information in this node. There is no additional sub-option information is requested in a Binding Acknowledge message.

Next Hdr	Hdr Len	Reserved	Option type = 7
Option Length	Status	Sequence Number	
Life Time			
Refresh			
Sub-Options ...			

Figure 2.9: Binding Acknowledge

A Binding Request is sent from a home agent or correspondent node to a mobile node. It is used to request a mobile node's binding from the mobile node. Normally, it is sent when the binding information in the sender is going to expire. A mobile node sends a Binding Update message as a reply for the Binding Request message. Figure 2.10 shows the message format. There is no additional sub-option information requested in a Binding Request message.

Next Hdr	Hdr Len	Reserved
Option type = 8	Option Length	Sub-Options ...

Figure 2.10: Binding Request

2.8 Traffic Handling

When a mobile node is away from its home network, it can use its home address or care-of addresses as the source address when it sends IP packets. Mobile IPv6 has no strict rule on this issue. Since the home address' network prefix does not match the network prefix of the foreign network to which the mobile node is attaching, the packets with the mobile node's home address as the source address, may not pass the routers with some restrict filters on the foreign network.

Before sending any packet, a correspondent node checks whether or not it has binding information for the destination address to which the packet is being sent. This binding information includes the care-of address. If the sending node has binding information for this address, the correspondent node uses a routing header to route the packet to this mobile node. The destination address in the packet is set to the mobile node's care-of address. The routing header is initialized to contain a single route segment with an address of the mobile node's home address. If the correspondent node has no binding information for the destination address to which the packet is being sent, the correspondent node simply sends the packet with no Routing header. If the destination node is not a mobile node or is a mobile node that is currently at home, the packet will be delivered directly to this node and processed normally. If the destination node is a mobile node that is currently away from the home network, the packet will be intercepted by the mobile node's home agent and tunnelled using IPv6 encapsulation to the mobile node's care-of address. The mobile node will then send a Binding Update to the correspondent node to update or register its binding information in the correspondent node.

After a mobile node registers in its home agent, the home agent intercepts all

packets on the home network addressed to the mobile node's home address and tunnels these packets to the mobile node using IPv6 encapsulation. The tunnel entry point node is the home agent and the tunnel exit point node is the primary care-of address. The home agent cannot use a Routing header to forward these intercepted packets to the mobile node as a correspondent node does, since the home agent cannot modify a packet in flight without invalidating any existing IPv6 authentication header or encapsulating payload header present in the packet.

When a packet arrives at a mobile node, the mobile node will process the packet before the packet goes to the IP stack. The mobile node checks whether or not the packet is in IPv6 encapsulation. If it is, the packet is de-encapsulated. If it is not, the destination address of the packet is set to the mobile node's home address. After this processing, the packet is put into the IP stack and processed by other protocols and applications.

2.9 Security

Security is always a concern for Internet communication. The components of security are: confidentiality, authentication, integrity checking, and non-repudiation. Confidentiality means data can be decoded only by authorized parties. Authentication means proving or disproving the entity claiming the identity. Integrity checking ensures that data cannot be modified without such modification being detectable. Non-repudiation proves that the source of some data did in fact send the data while the sender may deny the fact later. [9]

There are some existing protocols used to achieve security goals, such as Authentication header and IP Encapsulating Security Payload. In an IPv6 packet, there is an extension header named IP Authentication header. It is placed between the IP header and upper layer headers to protect the entire packet against being modified during transmission and to authenticate the sender. It can be used to implement the security for binding message. Figure 2.11 shows the authentication header format.

The value of the Security Parameters Index field tell the receiver which of many possible security associations is used to compute the value of the Authentication field. In IPv6, the Authentication Data field is calculated over the whole IPv6 packet using a message digest algorithm, such as MD5. For the Authentication header and other fields that are modified during transit, such as the IPv6 Hop Limit field, the values are replaced by 0 for the calculation purpose.

Figure 2.11: Authentication Header

In Mobile IPv6, the Binding Update message will result in IP packets, which are addressed to a mobile node, being delivered to the mobile node's care-of address. This ability to change the routing of these IP packets could be a significant vulnerability if any packet containing a Binding Update option were not authenticated. With Binding Acknowledgment messages, a hacker could trick a mobile node into believing a different outcome from a registration attempt to its home agent.

For the above reasons, the Mobile IPv6 requires that all packets, including Binding Update and Binding Acknowledge messages, have authentication protection. The purpose is to prevent any malicious Binding Updates and Acknowledgments. This protection can be provided by the IPv6 Authentication header.

2.10 Review of Current Research and Development

Since the Mobile IP proposal was approved by IETF, some academic institutes and industry companies started to enhance and implement the Mobile IP protocols and

systems in wired and wireless networks. Mobile IPv6 is a new proposal in process and some issues are still discussed currently. The new dimensions of protocol functionality, resource control, Quality-of-Service(QoS) support, and AAA(Authentication, Authorization, and Accounting) features extended by charging schemes are still to be integrated effectively.

Traditionally, wireless cellular communication systems have been engineered for voice. With the explosive growth of Internet applications and users, there is an increasing demand for providing Internet services to mobile users based on the voice-oriented cellular network. Third-Generation(3G) mobile systems are driven by the vision of information at any time, at any place and in any form. The system will combine a wide range of applications, such as voice, high speed data and Internet access, and video conferencing. To meet the requirements for the IP communication, the Mobile IP is adopted for the 3G system [15]. There are still many open issues under investigation such as AAA(Authentication, Authorization, and Accounting) in mobile network.

The implementations of Mobile IP varies by different OS systems and design models such as Linux, Free BSD, Solaris. The following is part of the current Mobile IP projects and implementations over wired network.

- Helsinki University of Technology

The project started in 1999. The implementation platform is Linux. The implementation partially covers Mobile IPv4 such as basic Mobile IPv4. The system misses some extensional features, such as Mobile IPv4 Route Optimization. One of the characteristics of the system is the code stays in the Linux kernel. But this approach introduces the problem of code portability. Since this system code change many of Linux kernel structures and different Linux kernel releases vary very much, lots of work needs to be done when transplanting the system to other Linux kernel releases. In the latest release of the implementation, some system interfaces, such as socket and ioctl, are extended to make the system more flexible for the application's interaction.

The Web site is <http://www.cs.hut.fi/Research/Dynamics/index.html>

- National University of Singapore

This project is concerned with the introduction of mobile computing within the global Internet. The focus is on the performance analysis of the Mobile IP Internet Draft, issued by IETF, which specifies an architecture and procedures for the support of mobile nodes in the Internet. The aim of the study is to characterize the efficiency of the Mobile IP Draft and to uncover any performance bottlenecks in the architecture. The study is carried out simultaneously by means of mathematical modeling and analysis, computer simulation and experimentation.

The project started in 1996. The implementation partially covers Mobile IPv4 and Mobile IPv6 over Linux and MS Window and includes Route Optimization for Mobile IPv4, Bi-tunnelling, Multiple simultaneous mobility bindings, and Fast Handoff. In their system, there is no security protection for binding messages.

The Web site is <http://mip.ee.nus.edu.sg/>

- Carnegie Mellon University

The Monarch Project is held by the School of Computer Science at CMU. The implementation platform is Free BSD. The implementation partially covers the idea of Mobile IPv4. Additionally, some tools and simulators are used to test and measure the performance of the mobile network. This project is led by one of the Mobile IP RFC authors, David Johnson.

The Web site is <http://www.monarch.cs.cmu.edu/>

- Sun Microsystems Laboratory

The implementation platform is Solaris system. The implementation partially covers the idea of Mobile IPv4. This project was originally led by one of the Mobile IP RFC authors, Charles Perkins. Since this project started in 1996 and had no updates since 1999, the system did not reflect current Mobile IP

RFC, such as Mobile IPv4 Route Optimization. One of the characteristics of the system is that all code of implementation stays in the application space and does not need to patch the kernel. It is the only one with this characteristic among the Mobile IP implementations over Unix and its clone OSs.

The Web site is <http://playground.sun.com/pub/mobile-ip/index.html>

- **POLITECHNICA University of Bucharest**

The implementation platform of the implementation is Microsoft Windows NT. When the project started, the fact that most laptop computers use MS Windows as their OS was considered. Based on this fact, MS Window is selected as implementation platform. The implementation partially covers Mobile IPv4. Simultaneous bindings and broadcast are not supported.

The Web site is <http://mip-nt.aii.pub.ro/>.

Chapter 3

Mobile IPv6 via Linux

3.1 Linux Overview

Linux is a clone of Unix and a multitasking, multi-user, 32bit Posix-compatible operating system. It can run on various processors including Intel, Alpha, Sun Sparc, and PowerPC. It provides most standard Unix network programming APIs and mechanisms. As other Unix systems, the Linux running space be divided into two parts: kernel space and application space. The source code of the Linux kernel is free and open for use by anyone. Users are free to change and patch the Linux kernel. To implement Mobile IPv6, the Linux kernel portion may need some enhancement. Linux provides the facility to do this.

The socket interface is designed to be a generic communication programming interface for applications. It was first introduced by the 4.2 BSD Unix. It has become an industry standard for Unix and its clones. Linux provides and fully implements the socket interface such as raw socket.

The socket interface is differentiated from the different services that are provided. The stream socket defines a reliable connection-oriented service, such as TCP. The datagram socket defines a connectionless and non-reliable service, such as UDP. The

raw socket allows direct access to lower layer protocols, such as IP and ICMP. This interface is often used for testing new protocol implementation since a raw socket lets application read/write the portions, which Linux kernel does not know, in an IP packet.

Figure 3.1: IP stack

A Linux kernel is a single, large program where all the functional components of the kernel have access to all of the internal data and routines. To avoid the size of the kernel increasing too much due to the growth of functions the kernel supports, Linux kernel allows dynamically loading and unloading OS components(modules). This mechanism keeps the kernel size to a minimum and makes the kernel flexible.

3.1.1 IP Traffic Flow in Linux

When a network data frame that contains an IPv6 packet arrives in a Linux system, a network device driver, such as an Ethernet interface driver, collects the packet of data and builds an *sk_buff* structure. After the *sk_buff* structure is built, the function *netif_rx* in `/usr/src/linux/net/core/dev.c` puts this structure, which contains the IP packet, into the system queue for processing.

The *net_bh* function in the `/usr/src/linux/net/core/dev.c` processes the data in the

system queue. The *net_bh* function uses the protocol field of the *sk_buff* structure to find the corresponding handler for the packet. The handler examines the IP header and retrieves the IP protocol number. Using the IP protocol number, the handler looks up the protocol registration table and retrieves the IP protocol entry point. For IPv6, the IP protocol entry point is *ipv6_rcv* in */usr/src/linux/net/ipv6/ip6_input.c*. During the IPv6 packet is processed by the IPv6 module, the packet reaches the function *ipv6_iput* in */usr/src/linux/net/ipv6/ip6_input.c*. The function *ipv6_iput* examines the IPv6 packet content, such as the Destination Header and the Authentication Header, and sends the packet to the sockets, which are listening to this packet. From the socket's point of view, the format and type of IP packet that can be received by the socket is restricted. The socket cannot receive an IP packet with a format and type that the Linux kernel does not support.

In the reverse direction, the upper layer application sends the IP packet to the kernel via the socket. The socket builds an *sk_buff* structure with the user data and passes it to the corresponding module in the kernel according to the type of socket and the socket's attributes. For IPv6, the data is passed to the IPv6 module. One of the components in the IPv6 module is */usr/src/linux/net/ipv6/ip6_output.c*. The IPv6 Module forms the IPv6 packet with the user data and passes the new packet to the device driver. Applications can form and send out any format and type of IP packets even if the Linux kernel does not support them.

3.1.2 Process and Thread

In the traditional Unix model, a process only has one control point. When an application needs to handle multiple tasks, it has to fork a process for each task. When using the process, the following issues need to be addressed.

The process model is expensive in system resources. Since it is difficult to share data between processes, processes store duplicated data and structures. For each process, Unix creates a Process Control Block to store system information about this process. The storage requires extra memory and address space. When data

is transferred from an originator to the destination between processes, the system control point is transferred from the originator process to the destination process. Under normal circumstances this context switch requires a lot of system operations. This context switch requires more time than the normal function call within a process.

A thread is a code sequence within a process. Each thread has one control point. One process may have multiple threads. All threads within a process share the same global memory. Therefore, the thread avoids problems associated with the process. Since multiple threads may simultaneously access the same data, conflict may occur. The p-thread library provides some mechanisms to coordinate the access to the shared data.

3.1.3 Data Exchange between Applications and Kernel

In Linux, the kernel and application processes have different space addresses. A process in the application space cannot directly access the data in the kernel and vice versa. There is no inter-process communication mechanism between the kernel and application processes.

System calls are a method by which the application process requests kernel space service. A system call passes some parameters to the kernel, such as any type of data and the system call number. The system call number is used by the kernel to find the corresponding entry in the system call table and trigger a corresponding operation.

By definition, a system call can be used to transmit user data from the application space to kernel space. If an application needs to transmit some data with user-defined structures to the kernel and trigger some operations in the kernel space, there are two ways to achieve this goal: creating a new system call and extending the existing system call.

Creating a new system call requires a change in the system structure, such as the system call table. The change needs to follow the system call structure and it requires more knowledge about the system. The caller can pass any number and structure of

data to the kernel space under the system restriction. Extending an existing system call does not need to change the system structure and requires less knowledge about the system. Overall, extending an existing system needs to follow the rules of the existing system call. Creating a new system call is more complicated but is more flexible.

3.2 Models of Mobile IPv6 Implementation in Linux

Due to the advantages of Linux described in section 3.1, many research groups have selected Linux as the platform for research on the Mobile IP. Linux already supports all IPv6 features that Mobile IPv6 requires, such as the IPv6 Destination Header. However, Linux does not support some Mobile IPv6 messages, such as binding messages. There are several ways to implement the Mobile IPv6 in Linux. This thesis discusses the following three approaches: application layer model, kernel layer model, and application and kernel layer model. These approaches are categorized according to the OS space in which the implementation code exists.

3.2.1 Application Layer Model

Figure 3.2 shows the structure of the application model, in which all of the implementation code exists in the application space. The implementation of this model does not change the Linux kernel that makes installation and debugging easy and convenient. It is also easy for management purposes, such as activating and deactivating the function.

The Mobile IP module listens to all IPv6 packets via a raw socket to capture the control message of the Mobile IPv6. The implementation changes and affects the kernel behavior, such as traffic tunneling, via system calls that the kernel supports.

In the development of this approach, a general knowledge of the Linux/Unix socket interface is required. This approach is also easy to apply to other Linux releases and

Figure 3.2: Application Layer Model

even other Unix OS because most of the APIs in Unix application space are standard.

This approach, however, may cause performance problems since all IPv6 packets are transferred between the application processes and the kernel. Since this approach needs to control the kernel behavior via a system call, this approach relies on how the system lets the applications control the kernel behavior. Some features of Mobile IPv6 have to be done in kernel space, such as capturing binding message in IPv6 Destination Headers and Linux does not support some these features currently. After analyzing the application layer model approach, it is insufficient for the purposes of this thesis. So far there is no implementation of Mobile IPv6 over Linux using the application layer model.

3.2.2 Kernel Layer Model

Figure 3.3 shows the structure of the kernel layer model, in which all of the implementation code exists in the kernel space. The Mobile IP module plugs into an IPv6

Figure 3.3: Kernel Layer Model

stack and monitors and controls IPv6 packets. It also needs to interact with the data link layers to control system behavior, such as traffic tunneling.

This approach works efficiently, especially with large amounts of control and traffic messages going on, since there are no control and traffic messages being transferred between the application space and the kernel space for Mobile IPv6 management.

The implementation requires the Linux kernel to be enhanced. Experience and strong knowledge of the Linux kernel is required. Because all the code exists in the kernel space, it also makes the size of the kernel larger if there is no dynamically loaded mechanism. Since the Linux kernel varies between releases, more work is required to adapt to the different Linux releases. During implementation, more work is required to reduce the re-writing and porting of code to the different Linux Kernel releases.

The Mobile IP implementation by Helsinki University of Technology adopted this model. The code works as a dynamically loaded kernel module.

Figure 3.4: Application and Kernel Layer Model

3.2.3 Application and Kernel Layer Model

In this approach, part of the implementation code exists in the application space and part of the codes exists in the kernel space. Figure 3.4 shows the structure of this model. The module in the kernel space controls Mobile IP traffic management, such as traffic tunneling. It can reduce the number of IP packets being transferred to the application space. The module in the application space takes care of the control message portion, such as binding management. Since there are not many control messages, there are few performance problems for this portion. The modules in the kernel space and application space need to exchange data, such as binding information.

The implementation also needs to enhance the Linux kernel. It can be considered as a trade-off of the two afar-mentioned approaches. This model's implementation requires less knowledge and experience about the kernel compared to the Kernel Layer Model. Less work is required when the implementation needs to be applied to the different Linux releases.

The Mobile IPv6 implementation by the National University of Singapore adopted

this model.

Chapter 4

Implementation and Experiment

In this section, we will describe our MIPv6 implementation and related technical issues. The experimental procedures used to test the implementation are listed also.

4.1 Design Model and Environment

As mentioned in the last chapter, different models of Mobile IPv6 implementation have their own advantages and disadvantages. After a comparison of the different models, the application and kernel layer approach is adopted in our implementation and experiment. In this implementation, the control management portion exists in the Linux application space and the traffic management portion exists in the Linux kernel space.

Our implementation focuses on the control portion of the Mobile IPv6, including Home Agent Discovery, Location Detection, and binding management on the mobile node, home agent, and correspondent node. Linux 2.2.14 was used as the experimental platform since it was the most stable release when this project started.

4.2 System Functionality

In our implementation, the following features are fulfilled according to the Mobile IPv6 RFC.

Mobile Node:

- Home agent discovery

Obtain or find a home agent for a mobile node via the ICMPv6 Home Agent Discovery message.

- Moving detection

Detect a mobile node's location by listening to and soliciting the ICMPv6 Router Discovery message.

- Obtain Care-of Address

Form a care-of address for a mobile node when the mobile node is out of its home network.

- Binding management

Manage the binding information and process corresponding messages, such as Binding Request and Binding Update messages, for a mobile node.

Home Agent:

- Home agent management

Provide the home agent service to mobile nodes. Process and respond to the Home Agent Discovery messages for mobile nodes.

- Binding management

Manage the binding information and process corresponding messages. It registers or de-registers mobile nodes in the binding cache via the Binding Update messages from mobile nodes.

Correspondent Node:

- **Binding management**

Manage the binding information and process corresponding messages. It registers or de-registers mobile nodes in the binding cache via the Binding Update messages from mobile nodes.

4.3 Threads

In our implementation, there are multiple tasks running simultaneously in a Mobile IP host, such as a mobile node. These tasks share common data but their functions are independent. For example, one of the tasks in a mobile node is location detection. This task needs to listen to and capture ICMPv6 Route Advertisement messages. In the meantime, the binding management module needs to listen to and capture binding messages. These tasks run simultaneously and the binding management module uses the data from the Location Detection module.

In our implementation, p-thread library is used to implement multiple tasks. The code in the Linux application space is running in one process with multiple threads. The Linux p-thread library is a library for multi-threaded programming. It implements the Posix 1003.1c API for threads. It runs on any Linux system with kernel 2.0.0 or more recent versions, and a suitable C library.

4.4 ICMPv6 Message Capture

The following messages are transmitted via ICMPv6: Router Discovery and Solicitation, and Home Agent Discovery Request and Reply. Linux currently supports these ICMPv6 messages. The Mobile IPv6 system creates, sets, and listens to a raw socket to capture these ICMPv6 messages.

First, the application needs to initialize a raw socket, which includes creating and setting the attributes. When an application listens to a raw socket to capture ICMPv6 messages, the Linux IP stack, as the default, will pop up all ICMPv6 messages to this socket, even though the application is interested in only some types of these ICMPv6 messages. To avoid unnecessary operations, the raw socket should be set to pass useful messages and block others. In the meantime, some other raw socket attributes, such as validating the checksum of an IP packet, need to be set. The combination of these attributes may confuse programmers during development.

The following is a sample of code to catch the ICMPv6 Router Advertisement and Home Agent Discovery Reply messages. Exception handling is omitted.

```
1. struct icmp6_filter filter;
2. int val;
3. router_socket = socket(AF_INET6, SOCK_RAW, IPPROTO_ICMPV6);
4. val = 1;
5. setsockopt(router_socket, IPPROTO_IPV6, IPV6_PKTINFO, &val, sizeof(int));
6. val = 2;
7. setsockopt(router_socket, IPPROTO_IPV6, IPV6_CHECKSUM, &val, sizeof(int));
8. val = 1;
9. setsockopt(router_socket, IPPROTO_IPV6, IPV6_UNICAST_HOPS, &val, sizeof(int));
10. val = 1;
11. setsockopt(router_socket, IPPROTO_IPV6, IPV6_HOPLIMIT, &val, sizeof(int));
12. ICMP6_FILTER_SETBLOCKALL(&filter);
13. ICMP6_FILTER_SETPASS(ND_ROUTER_ADVERT, &filter);
14. ICMP6_FILTER_SETPASS(HA_DISCOVERY_REPLY, &filter);
15. setsockopt(router_socket, IPPROTO_ICMPV6, ICMP6_FILTER, &filter, sizeof(filter));
```

Details of the code and its motivation are as follows:

Line 1 Declare a filter for ICMPv6 messages. The structure is defined in `<netinet/icmp6.h>`.

Line 3 Create a raw socket to receive ICMPv6 messages.

Line 4-12 Set the attributes for the socket.

Line 12 Set the filter to block all ICMPv6 messages passed to this socket.

Line 13-14 Set the filter to enable the Router Advertisement and Home Agent Discovery Reply messages to be passed to this socket.

Line 15 Associate the filter with the socket.

IPV6_PKTINFO specifies that the following two pieces of information for datagrams are returned as ancillary data of `recvmsg`: the destination IPv6 address and arriving interface index. IPV6_CHECKSUM specifies two bytes offset into the user data of where the checksum field is located. IPV6_UNICAST_HOPS specifies the default hop limit for an outgoing datagram sent on to the socket. IPV6_HOPLIMIT specifies that the received hop limit field is returned as ancillary data by `recvmsg`.

After that, the application just needs to listen to this socket:

```
1. char recvbuf[1500];
2. struct msghdr mhdr;
3. struct iovec iov;
4. int len;
5. struct nd_router_advert *ra_ptr;
6. while (1)
7. {
8.     iov.iov_len = 1024;
9.     iov.iov_base = recvbuf;
10.    mhdr.msg_iov = &iov;
11.    mhdr.msg_iovlen = 1;
12.    len = recvmsg(router_socket, &mhdr, 0);
13.    ra_ptr = (struct nd_router_advert *) recvbuf;
```

```
14.     if (ra_ptr->nd_ra_hdr.icmp6.type == ND_ROUTER_ADVERT)
15.         recv_router_advert(mhdr);
16.     else if (ra_ptr->nd_ra_hdr.icmp6.type == HA_DISCOVERY_REPLY)
17.         recv_ha_discovery_reply(mhdr);
18. }
```

Details of the code and its motivation are as follows:

Line 8-11 Initialize the data buffer. An ICMPv6 message is returned in a system data structure `msg_hdr`. The main part of the ICMPv6 message is stored in `recvbuf`.

Line 12 Block and listen to the ICMPv6 message.

Line 13-17 Check the type of ICMPv6 message and trigger the corresponding functions. For mobile nodes, they are the Router Advertisement handler (line 15) and the Home Agent Discovery handler (Line 17). The system data structure `nd_router_advert` is used to check the type of ICMPv6 message. Although the ICMPv6 Home Agent Discovery message has a different data format from the Router Advertisement message, the system data structure `nd_router_advert` still can be used to check its common part ICMPv6 header, which contains the type of the ICMPv6 message.

4.5 Care-of Address Configuration

As mentioned in section 2.5, there are three ways for a mobile node to get or form a care-of address: stateful, stateless, and static.

In our implementation, the stateless way is adopted for a mobile node to form the care-of address. After it gets a Router Advertisement message, a mobile node uses the network prefix option in this message and the Ethernet address of the corresponding link level interface to form a new care-of address. In Linux, the file `/proc/net/if_inet6` contains the link level address for all link interfaces.

The care-of address formed in the stateless way is as follows:

Care of address = <sub-network prefix> + <0...0> + <Ethernet address>

A care-of address formed in this way may be duplicated in the network. But with a Ethernet address, which has 48 bits, and a sub-network prefix, the possibility of address duplication is very low. There is an assumption that the new care-of address formed in that way is unique. Since the Mobile IPv6 allows a mobile node to use a new care-of address without performing Duplicate Address Detection, the mobile node in this implementation uses a new care-of address without performing Duplicate Address Detection.

4.6 Security

In our implementation, the IP Authentication Header is used to implement the security for the Binding Update and Binding Acknowledge messages. The encapsulation algorithm is the Message Digest 5 (MD5). The implementation of this algorithm is from RFC 1321 [2].

As defined in RFC 1321, the MD5 algorithm takes as input a message of arbitrary length and produces as output a 128-bit ‘fingerprint’ or ‘message digest’ of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given pre-specified target message digest. The MD5 algorithm is intended for digital signature applications, where a large file must be ‘compressed’ in a secure manner before being encrypted with a private (secret) key under a public-key crypt system.

In our system, mobile hosts (mobile node, home agent and correspondent node) share a common key. After all portions of a Binding Update or Binding Acknowledge message except the Authentication Header portion are ready, the whole IP packet, including all IP headers, is passed to the MD5 algorithm with the Authentication Header set to 0. The result is set into the Authentication Header. On the receiver side, the whole IP packet is passed to the MD5 algorithm again to verify the data

integrity.

4.7 Binding Message Capture

In Mobile IPv6, binding messages are transmitted in the Destination Header of any IPv6 packet. The binding messages include Binding Update, Binding Acknowledge, and Binding Request messages. Linux supports the IPv6 Destination Header. Linux, however, does not recognize the content of this header. The standard socket API does not have a mechanism for applications to capture the IPv6 packets, which contain a binding message in the Destination Header portion. To allow a process in the application space to get the binding messages, the Linux kernel must be enhanced to intercept these IPv6 packets and pop them up to the application space via the socket API. There are many approaches to achieve this goal.

In our implementation, some codes are inserted in the trail of an IP packet in the Linux IP stack. The objective is to check the sub-option number in the Destination Header of IP packets, find any IPv6 packets with binding messages in Destination Headers, and send these packets to the application space via the raw socket mechanism.

In Linux, before an IPv6 packet is discarded or sent to the application space, Linux checks its content using the function `ip6_input()`. One of its tasks is to parse the IPv6 extension headers. The function of parsing the IPv6 extension headers is `ipv6_parse_exthdrs()`.

In the file `exthdrs.c`, there is one entry for each type of extension header including the Destination Header and Authentication Header. Since Linux assumes no protocol using the Destination Header, the entry for the Destination Header in `exthdrs.c` is empty. To capture the Destination Header, some code is added into `exthdrs.c`.

The following is the code used in our approach:

```
1. static int dstopt.binding_operation(struct sk_buff *skb, u8 *ptr)
```

```

2. {
3.     return 1;
4. }

5. /*****
6. Destination options header.
7. *****/
8. struct tlvtype_proc tlvprocdestopt_lst[] = {
9.     {7,     dstopt_binding_operation},    // Binding Ack
10.    {8,     dstopt_binding_operation},    // Binding Req
11.    {198,    dstopt_binding_operation},    // Binding Update
12.    {201,    dstopt_binding_operation},    // Binding Home
13.    {-1,     NULL}
14. };

15. /*****
16. Parse destination options header.
17. *****/
18. static u8 *ipv6_dest_opt(struct sk_buff **skb_ptr, u8 *nhptr)
19. {
20.     struct sk_buff *skb=*skb_ptr;
21.     struct inet6_skb_parm *opt = (struct inet6_skb_parm *)skb->cb;
22.     struct ipv6_destopt_hdr *hdr = (struct ipv6_destopt_hdr *) skb->h.raw;
23.     if (opt->dst0)
24.         opt->dst1 = (u8*)hdr - skb->nh.raw;
25.     else
26.         opt->dst0 = (u8*)hdr - skb->nh.raw;
27.     if (ip6_parse_tlv(tlvprocdestopt_lst, skb, nhptr)) {
28.         skb->h.raw += ((hdr->hdrlen+1)<<3);
29.         return &hdr->nexthdr;
30.     }
31.     return NULL;

```



```

32. }
33. /*****
34. Deliver destination options header to socket
35. *****/
36. static int deleiver_dstopt_raw(struct sk_buff *skb)
37. {
38.     struct in6_addr *saddr;
39.     struct in6_addr *daddr;
40.     struct sock *raw_sk;
41.     __u8 hash;
42.     saddr = &skb->nh.ipv6h->saddr;
43.     daddr = saddr + 1;
44.     hash = NEXTHDR_DEST & (MAX_INET_PROTOS - 1);
45.     raw_sk = raw_v6_htable[hash];
46.     while ((raw_sk = raw_v6_lookup(raw_sk, NEXTHDR_DEST, daddr, saddr))) {
47.         struct sk_buff *buff;
48.         buff = skb_clone(skb, GFP_ATOMIC);
49.         if (buff)
50.             rawv6_rcv(raw_sk, buff, 512);
51.         raw_sk = raw_sk->next;
52.     }
53.     return 1;
54. }
55. /*****
56. Parse ipv6 extension header.
57. *****/
58. u8 *ipv6_parse_exthdrs(struct sk_buff **skb_in, u8 *nhptr)
59. {
60.     struct hdrtype_proc *hdrt;
61.     u8 nexthdr = *nhptr;
62.     u8 has_dest_opt = 0;

```

```

63.     if (nexthdr == NEXTHDR_DEST)
64.         has_dest_opt = 1;
65.     restart:
66.     for (hdrt=hdrproc_1st; hdrt->type >= 0; hdrt++) {
67.         if (hdrt->type == nexthdr) {
68.             if ((nhptr = hdrt->func(skb_in, nhptr)) != NULL) {
69.                 nexthdr = *nhptr;
70.                 if (nexthdr == NEXTHDR_DEST)
71.                     has_dest_opt = 1;
72.                 goto restart;
73.             }
74.             if (has_dest_opt)
75.                 deleiver_dstopt_raw(*skb_in);
76.             return NULL;
77.         }
78.     }
79.     if (has_dest_opt)
80.         deleiver_dstopt_raw(*skb_in);
81.     return nhptr;
82. }

```

Details of the code and its motivation are as follows:

Line 1-4 When the system finds a Destination Header option in an IPv6 packet, the system checks the content of the Destination Header and calls this function for each sub-option type, which is defined in the following sub-option table. The return value of this function decides whether or not the system processes this sub-option. If the return value is 0, the system skips this sub-option. Otherwise, the system stores the sub-option in corresponding fields in the system data. In our experiment, this function's return value is always 1. So all IPv6 packets, which contain the binding messages, are processed by the system.

Line 8-14 Sub-option table for IPv6 Destination Header. In the original Linux code, this table is empty because Linux assumes that no protocol is using the IPv6 Destination Header. The system skips this portion. In our experiment, several entries are added for binding messages. For each entry, the first parameter is the value of a sub-option and the second parameter is the function to process this sub-option.

Line 18-32 Parse the Destination Header and store the data in corresponding fields of system data structures.

When the system finds the IPv6 Destination Header, this function is called. Lines 20-22 cast the raw data to system data structures. Lines 23-26 store the IPv6 Destination Header in corresponding fields of system data structures. If this IPv6 Destination Header is the first Destination Header in this packet, the data is stored into `opt->dst0`. If this IPv6 Destination Header is the second in this packet, the data is stored into `opt->dst1`. Lines 27-31 call the corresponding functions according to the sub-option in this Destination Header. The return value of this function decides whether or not the system processes this sub-option.

The original Linux code assumes that there is just one Destination Header in an IPv6 packet. The data is always written into the field `opt->dst1`. If there are two Destination Headers, the second Destination Header will overwrite the first one. In our experiment, the Binding Update and the Home Address messages are sent together in one single IPv6 packet. Each of them is included in one Destination Header. This means that there are two Destination Headers in an IPv6 packet. After this enhancement, the kernel stores two Destination Headers in `opt->dst0` and `opt->dst1`.

Line 36-54 Deliver an IPv6 packet containing binding messages to a raw socket.

In the original Linux code, there is no such function. If there are Destination Headers in an IPv6 packet, this function is called. Lines 42-43 get the source address and destination address from this IP packet. Lines 44-45 get the raw socket stack which has all opened sockets. Line 46 searches the raw socket stack and finds the socket whose port number is `NEXTHDR_DEST`. If the system finds any such socket, lines

47-51 deliver this IPv6 packet to that listener. In this way, the process in the application space can get all IPv6 packets that contain binding messages in Destination Headers.

Line 58-82 Parse the extension header in an IPv6 packet. In the original Linux code, there is no special data processing for IPv6 packets that contain Destination Headers. This function is enhanced in this experiment. Lines 63-64 and lines 70-71 mark the flag `has_dest_opt` if the next header is a Destination Header while the system processes the IPv6 header options. If Destination Headers are found in this IPv6 packet, lines 74-75 and lines 79-80 call function `deleiver_dstopt_raw` and deliver the packet to the application space via a raw sock.

4.8 ICMPv6 Message Sending

There are two types of ICMPv6 messages used by Mobile IPv6 hosts: the Router Solicitation and the Home Agent Discovery (Request and Reply). In Linux, the kernel can calculate the checksum for an ICMPv6 packet instead of the application.

To send an ICMPv6 message, the application creates an ICMPv6 raw socket and sets the corresponding attribute for this raw socket, such as the checksum flag. Since the socket is bi-directional, the application can use the same ICMPv6 raw socket, which is used to capture the incoming ICMPv6 messages, for the outgoing ICMPv6 messages.

When sending an ICMPv6 Router Solicitation, a mobile node sets the destination address as the link level anycast address `{0xff,0x02,0,0,0,0,0,0,0,0,0,0,0,0,0,1}`. With this address, the ICMPv6 Router Solicitation message will be anycasted along the sub-network to which the mobile node attaches. The ICMPv6 Router Solicitation structure is defined in `<netinet/icmp6.h>`.

When sending an ICMPv6 Home Agent Discovery Request message, the message is multicasted or anycasted to the mobile node's home network. According to the description of the ICMPv6 Home Agent Discovery message in subsection 2.6, the

message has the following structure.

```
struct ha_discovery_request
{
    uint8_t      icmp6_type; // = 190
    uint8_t      icmp6_code; // = 0
    uint16_t     icmp6_cksum; // icmp ckecksum
    uint16_t     identifier; // identifier for this request
    uint8_t      reserve[10];
    struct in6_addr home_addr[16]; // MN's home address
};
```

The ICMPv6 Home Agent Reply has the following format:

```
struct ha_discovery_reply
{
    uint8_t      icmp6_type; // = 191
    uint8_t      icmp6_code; // = 0
    uint16_t     icmp6_cksum; // icmp ckecksum
    uint16_t     identifier; // identifier for this request
    uint8_t      reserve[10];
    struct in6_addr ha_addr[16]; // home agent address
};
```

Since IETF is applying the ICMPv6 type of Home Agent Discovery Request and Reply from Internet Assigned Numbers Authority (IANA), we temporarily define their values in our experiment as 190 for Home Agent Discovery Request and 191 for Home Agent Discovery Reply.

4.9 Binding Message Sending and Structure

There are three types of binding message used by the Mobile IPv6 hosts: Binding Request, Binding Update, and Binding Acknowledge.

To send a binding message, the application needs to create a raw socket and set the corresponding attributes for this socket, such as the destination header flag and authentication header [23].

```
1. int init.binding_socket()
2. {
3.     int val;
4.     binding_socket = socket(AF_INET6, SOCK_RAW, IPPROTO_DSTOPTS);
5.     val = 1;
6.     setsockopt(binding_socket, IPPROTO_IPV6, IPV6_PKTINFO, &val, sizeof(int));
7.     val = 255;
8.     setsockopt(binding_socket, IPPROTO_IPV6, IPV6_UNICAST_HOPS, &val, sizeof(int));
9.     val = 255;
10.    setsockopt(binding_socket, IPPROTO_IPV6, IPV6_HOPLIMIT, &val, sizeof(int));
11.    val = 1;
12.    setsockopt(binding_socket, IPPROTO_IPV6, IPV6_DSTOPTS, &val, sizeof(int));
13.    val = 1;
14.    setsockopt(binding_socket, IPPROTO_IPV6, IPV6_AUTHHDR, &val, sizeof(int));
15. }
```

IPV6_PKTINFO specifies the following two pieces of information about a received IPv6 datagram that is to be returned as ancillary data by `recvmsg`: the destination IPv6 address and the arriving interface index. IPV6_UNICAST_HOPS specifies the default hop limit for an outgoing datagram. The IPV6_DSTOPTS option specifies that any received IPv6 destination header options are to be returned as ancillary data by `recvmsg`. IPV6_HOPLIMI specifies that the received hop limit field is returned as ancillary data by `recvmsg`. IPV6_AUTHHDR specifies the receiving of IPv6 authentication information.

To send the binding message, the application needs to form the packet of the binding message.

According to the description in subsection 2.7, the Binding Update message has the following data structure:

```
#define BIND_UPDATE_TYPE    198
#define BIND_UPDATE_LEN    8
struct bind_update
{
    u_int8_t    option_type; // = 198
    u_int8_t    option_len; // = 8
    u_int8_t    flag; // attributes for this message.
    u_int8_t    prefix_len; // the prefix length of the care-of addr.
    u_int8_t    sequence[2]; // sequence number of this message
    u_int8_t    lifetime[4]; // life time of this message
};
```

The Binding Update message includes the Unique Identifier and Alternate Care-of Address sub-option with the following data structure:

```
#define UNIQ_ID_TYPE    2
#define UNIQ_ID_LEN    2
struct uniq_id
{
    u_int8_t    option_type; // = 2
    u_int8_t    option_len; // = 2
    u_int8_t    uniq_id[2]; // unique id for this message.
};

#define ALT_CAREOF_ADDR_TYPE    4
#define ALT_CAREOF_ADDR_LEN    16
struct alt_careof_addr
{
    u_int8_t    option_type; // = 4
    u_int8_t    option_len; // = 16
    u_int8_t    care_addr[16]; // MN's care-of address
    u_int8_t    pads[6];
};
```

In our experiment, a Binding Update message along the Home Address option is

sent in Destination Header. The Home Address and Authentication options have the following data structure:

```
#define BIND_HOME_TYPE    201
#define BIND_HOME_LEN    16
struct bind_home
{
    u_int8_t    option_type; // = 201
    u_int8_t    option_len; // = 16
    u_int8_t    home_addr[16]; // MN's home addr
    u_int8_t    pads[4];
};

struct auth_data
{
    u_int8_t    option_type;
    u_int8_t    option_len;
    u_int8_t    reserved[2];
    u_int8_t    spi[4];
    u_int8_t    auth_data[16];
};
```

In our experiment, the key is pre-defined and shared between the mobile nodes, home agents, and correspondent nodes.

According to the description in section 2.7, the Binding Acknowledge message has the following data structure:

```
#define BIND_ACK_TYPE    7
#define BIND_ACK_LEN    16
struct bind_ack
{
    u_int8_t    option_type; // = 7
    u_int8_t    option_len; // = 16
    u_int8_t    status; // status of this message
    u_int8_t    sequence[2]; // sequence number of this message
    u_int8_t    lifetime[4]; // life time of the binding registration.
    u_int8_t    refresh[4]; // refresh interval of the binding registration.
```



```

        u_int8_t    pads[1];

};

```

4.10 Data Exchange between Application Space and Kernel Space

In our implementation, there are two binding caches in a home agent or correspondent node. One is in the application space and the other is in the kernel space. They keep exactly the same data for the binding management. To synchronize the two binding caches, any change in the binding cache in the application space is reflected on the cache in the kernel space. There are also two copies of Binding Update lists in a mobile node. They are subject to the same situations and requirements as the binding caches in a home agent or correspondent node.

In our Mobile IP implementation model, the binding management part sits in the application space. The binding management updates the binding data in the application space first and then passes the data to the kernel space to update the data in kernel space.

Since the data transmitted between the kernel and the applications is not complex in this experiment, an existing system call is extended to exchange data between the kernel and the applications.

The following sample of code shows how the data is transmitted from the applications to the kernel. The system call `ioctl()` is extended to transfer the binding management data from the application space to the kernel space.

Extend the function `inet6_ioctl` in the kernel source file `af_inet6.c`.

```

static int inet6_ioctl(struct socket *sock, unsigned int cmd, unsigned long
arg)
{
    ...
    switch(cmd)

```

```

    {
    ...
        case SIOCADDBC:
        case SIOCUPDBC:
        case SIOCDELBC:
            return (mip6.bindingcache.ioctl(cmd,(void *)arg));
    ...
    }
}

```

The SIOCADDBC, SIOCUPDBC and SIOCDELBC are user-extended commands. The `mip6.bindingcache.ioctl` is a user extended function in the kernel space. The parameter `arg` is the data from the application space.

When an application in the application space needs to pass some data to the kernel space, the application opens an IPv6 socket and calls the system call `ioctl()`. The kernel will pass the parameters to the corresponding handler according to the type of device. Since the device `sd` is an `AF_INET6` socket, the parameters will be passed to the `inet6.ioctl`.

```

sd = socket(AF_INET6, SOCK_RAW, 0);

ioctl(sd, SIOCADDBC, &arg);

```

The `arg` is the data being transmitted to the kernel.

4.11 Mobile Node

4.11.1 System Structure

As shown in Figure 4.1, in the mobile node, there are three threads running in parallel: the ICMPv6 listener thread, the binding management thread, and the timer thread.

The home agent of a mobile node sends Home Agent Discovery Reply messages to the mobile node via ICMPv6 whenever the home agent receives a Home Agent

Figure 4.1: Mobile Node Structure

Discovery Request or the home agent is re-configured. The router periodically sends Router Advertisement messages via ICMPv6. So the mobile node must keep listening to and capturing the Agent Discovery Reply and Router Advertisement ICMPv6 messages. Since these two types of messages are all carried in ICMPv6, the ICMPv6 listener thread is created to listen to and capture these messages.

The home agent and the correspondent node send Binding Acknowledge and Binding Request messages to a mobile node whenever they are required. The mobile node also needs to keep listening to and capturing these messages. For that reason, the binding management thread is created to listen and capture these messages.

System data that has time stamp needs to be checked periodically. Each entry in the Binding Update list has time stamp information, such as lifetime. To monitor these data, the timer thread is created. The timer thread also triggers some events, such as repeating binding messages.

4.11.2 Data Structure

In a mobile node, the most important data is the Binding Update list. It includes data about the home agent and the correspondent nodes. The ICMPv6 listener, binding management, and timer threads interact with the data in the Binding Update list. The Binding Update list is implemented as a link list. Each node or entity in this link list represents a home agent or correspondent node. It has the following structure:

```

struct binding_update_node
{
    struct in6_addr    dest_addr;
    struct in6_addr    home_addr;
    struct in6_addr    care_addr;
    int                init_lifetime;
    int                left_lifetime;
    int                refresh_time;
    int                sequence;
    int                last_send_time;
    int                acknowledge;
    int                if_home_agent;
    u_int8_t           spi[4];
    u_int8_t           auth_data[16];
    struct binding_update_node *next;
};

```

There are two Binding Update lists in the system. One exists in the application space and the other in the kernel space. They have exactly the same data. The Binding Update list in the application space is the main Binding Update list. At any time, when the data in this Binding Update list is changed, it should exactly reflect the Binding Update list in the kernel space.

The reason to have two Binding Update lists is to avoid any potential inconvenience of data operation and performance issue. In our implementation, the control management and traffic management modules exist in different OS spaces at running time: one exists in the application space and the other exists in the kernel space.

After an IP packet arrives or before an IP packet is sent out, the traffic management module checks the corresponding information in the Binding Update list. If there is only one Binding Update list in the application space, the traffic management module, which exists in the kernel space, has to retrieve data from the application space for every IP packet. It is inconvenient and may cause performance issues with heavy IP traffic. After a binding message arrives at a mobile node or before a binding message is sent out, the control management module also checks the corresponding information in the Binding Update list. If there is only one Binding Update list in the kernel space, the control management module, which exists in the application space, has to retrieve data from the kernel space for every binding message. It is also inconvenient and may cause performance issues with heavy binding messages.

The binding management updates the data in the Binding Update list according to the binding messages it receives. For example, after receiving a Binding Acknowledge message from a home agent, the system needs to find the node in the Binding Update list for the home agent and set the attribute acknowledge to true.

The ICMPv6 listener also takes care of the data in the Binding Update list. For example, when the system receives getting a Home Agent Discovery Reply message, it needs to update the data for the home agent in the Binding Update list. If there is no such node in the Binding Update list, it creates one for this home agent.

The timer checks the Binding Update list periodically and updates some attributes of the nodes, such as time to live, in the Binding Update list. It also triggers some operations according to the value of the attribute. For example, when an entry in the Binding Update list expires, it will use the function in the binding management to send the Binding Update messages again.

4.11.3 Control Flow

When Mobile IPv6 starts, the system initializes some data and then generates three threads: binding management, ICMPv6 listener, and timer.

The binding management listens to the Binding Request and Binding Acknowledge messages. After receiving a Binding Request message from a home agent, the mobile node needs to send a Binding Update message and set the flag of acknowledge in the Binding Update list. This flag is for the timer to trace the Binding Acknowledge message from the home agent. After receiving a Binding Acknowledge message, the binding management checks whether or not it is authenticated. If a Binding Acknowledge message is from a home agent, the mobile node needs to send the Binding Update message to all correspondent nodes to update their binding caches. After receiving a Binding Request message, the binding management returns a Binding Update message. The control flow is shown in Figure 4.2 and 4.3.

Figure 4.2: Mobile Node: Process Binding Request

The ICMPv6 listener listens to the Home Agent Discovery Reply and Router Advertisement messages. These types of messages are transmitted by the ICMPv6 messages. When the ICMPv6 listener gets a Router Advertisement message, it finds out whether or not it is from the same router as before. If it is, the ICMPv6 listener updates the router data. If not, the ICMPv6 listener compares the prefix option in the ICMPv6 packet with the one the mobile node received previously. If it is different, it indicates the mobile node has moved and attached to a new sub-network. A new care-of address needs to be formed and sent to the home agent with a Binding Update message. When the ICMPv6 listener gets a Home Agent Discovery Reply message, it triggers the mobile node to send a Binding Update message to the home agent. The

Figure 4.3: Mobile Node: Process Binding Acknowledgement

control flow is shown in Figure 4.4 and 4.5.

The timer thread periodically checks the Binding Update list and some other system data. If the mobile node has not received a Router Advertisement message for a certain time or if the router is expired, the timer sends a Router Solicitation message. If the mobile node has not got a home agent, the timer will keep sending a Home Agent Discovery Request message. The timer also checks the Binding Update list and validates the data for each node in the Binding Update list. The control flow is shown in Figure 4.6, 4.7 and 4.8.

Figure 4.4: Mobile Node: Process ICMPv6 Router Advertisement

4.12 Home Agent

4.12.1 System Structure

In a home agent, there are also three threads running in parallel as a mobile node: ICMPv6 listener thread, binding management thread and timer thread. Their functions are substantially different, however, from the threads in a mobile node.

A mobile node sends an ICMPv6 Home Agent Discovery Request message to its home network via ICMPv6 whenever the mobile node is powered on or its home agent data is invalid or expired. The home agent must keep listening to and capturing the

Figure 4.5: Mobile Node: Process ICMPv6 Home Agent Discovery Reply

ICMPv6 Home Agent Discovery Request messages. The ICMPv6 listener thread is created to listen to and capture these messages. Compared to the mobile node, this thread does not need to listen to the ICMPv6 Router Advertisement messages.

Mobile nodes send Binding Update messages whenever they are required. A home agent needs to keep listening to and capturing these messages. The binding management thread is created to listen to and capture these messages, update some corresponding information in the binding caches with these messages.

There is some system data that has a time stamp and needs to be checked periodically and consequently triggers some actions. To monitor and manage these data, the timer thread is created.

A home agent has almost the same system structure as a mobile node shown in Figure 4.1, but the function inside each component is substantially different.

Figure 4.6: Mobile Node: Trace Router Advertisement

4.12.2 Data Structure

In a home agent, the binding cache includes the corresponding data about all mobile nodes. The binding management and timer modules interact with the data in these binding caches. A binding cache is implemented as a link list. Each node or entity in this link list represents a mobile node. It has the following structure.

```

struct binding_cache_node
{
    struct in6_addr    home_addr;
    struct in6_addr    care_addr;
    int                init_lifetime;
    int                left_lifetime;
    int                sequence;
    int                prefix_len;
    int                if_home_agent;
    u_int8_t           flag;
    u_int8_t           spi[4];
}

```

Figure 4.7: Mobile Node: Trace Binding Update List

```

u_int8_t          auth_data[16];
struct binding_cache_node *next;

};

```

There are also two binding caches in the system. One exists in the application space and the other exists in the kernel space. The purpose of these two binding caches is the same as the Binding Update lists in a mobile node, although the use and structures are substantially different.

The binding management thread updates the data in the binding cache according to the binding messages it receives. For example, after receiving a Binding Update message from a mobile node, the system finds the entry in the binding cache for the mobile node and sets the value of the attribute `care_addr`. If there is no entry for this mobile node in the binding cache, it creates one for this mobile node.

The timer periodically checks and updates the attributes of the nodes and entries

Figure 4.8: Mobile Node: Trace Binding Acknowledge

in the binding caches. It also triggers some operations according to the value of the attribute. For example, when a node or entry in the binding cache expires, the timer thread removes the node from the binding caches.

4.12.3 Control Flow

When the Mobile IPv6 in a home agent starts, the system initializes some system data and then generates three threads: binding management, ICMPv6 listener, and timer threads.

The binding management listens to the Binding Update message. The binding management checks the authentication for each Binding Update message received. If the node, which sends this message, requests the destination node to work as its

home agent, a Binding Acknowledge message is returned. The control flow is shown in Figure 4.9.

Figure 4.9: Home Agent: Process Binding Update

The ICMPv6 listener listens to Home Agent Discovery Request messages from the mobile nodes. These messages are transmitted by ICMPv6 messages. When the ICMPv6 listener receives a Home Agent Discovery Request message, the home agent checks whether or not it can act as a home agent for this mobile node. If it can, the home agent sends the Home Agent Discovery Reply message to the mobile node. The control flow is shown in Figure 4.10.

The timer periodically checks the binding cache. If a node within the binding cache is expired, the timer will send a Binding Request to the mobile node and remove the corresponding entry from the binding cache. The control flow is shown in Figure 4.11.

Figure 4.10: Home Agent: Process Home Agent Discovery Request

4.13 Correspondent Node

4.13.1 System Structure

As it is shown in Figure 4.12, in the correspondent node, there are two threads running in parallel: binding management thread and timer thread. Compared to a home agent and mobile node, there is no ICMPv6 listener thread, since there are no ICMPv6 messages for a correspondent node.

Mobile nodes send Binding Update messages whenever they are required. The correspondent node needs to listen to and capture these messages. Thus, the binding management thread is created to listen to and capture these messages. As a home agent, there is some data that has time stamps and needs to be checked periodically in a correspondent node. To monitor and manage these data, the timer thread is created.

Figure 4.11: Home Agent: Trace Binding Registration

4.13.2 Data Structure

In the correspondent node, there are also two binding caches. One exists in the application space and the other exists in the kernel space. The purpose and operation of these binding caches are the same as the binding caches in a home agent. They include data about all mobile nodes. Binding management and timer threads interact with the data in the binding cache. The binding caches in a correspondent node have the same structure as a home agent.

4.13.3 Control Flow

When Mobile IPv6 starts, the system initializes some system data and then generates two threads: binding management and timer threads.

The binding management listens to the Binding Update message. The binding

Figure 4.12: Correspondent Node Structure

management checks the authentication for each Binding Update message. It has the same structure as the binding management in a home agent but sends different Binding Acknowledge messages with the home agent.

The timers will periodically check the binding cache. If a node in the binding cache expires, the timer will send a Binding Request to this mobile node and remove the corresponding entry from the binding cache. It has the same structure and function as the timer in the home agent.

4.14 Experiment

4.14.1 Testing Environment

In the Mobile IPv6 system, a mobile node detects its network attachment only by tracing the Router Advertisement messages sent periodically by routers. The time

interval between two consequent Router Advertisement messages may be seconds or minutes. To test our system, a mobile node has to roam in the Internet. It means the mobile node needs to connect to different networks. It is inconvenient and difficult for development. After analysis, we designed a pseudo router. Its main function is to send the fake Router Advertisement messages according to user-defined data, such as network prefix and lifetime. When a mobile node needs to change its network attachment to a different network, instead of physically changing the network attachment of the mobile node, we can change the network prefix portion in the fake Router Advertisement messages to be sent by this pseudo router. The mobile node receives this fake Router Advertisement message and thinks its network attachment is changed because the network prefix is different in the new fake Router Advertisement messages.

Figure 4.13 shows our testing environment.

Figure 4.13: Testing Environment

4.14.2 Test Procedure and Result

Our test focused on system functionality and performance. There are many factors that help determine the system performance. Among these factors, the network

latency and internal data processing are the most important. The network latency is determined by many other factors, such as bandwidth and network hops. It is unpredictable somehow. In our testing environment, we assume the network hop is equal to 1 and network latency is none.

- Mobile node's network attachment change

First, we change the network prefix portion in the fake Router Advertisement messages sent by the pseudo router. The fake network prefix must be a foreign network prefix. The mobile node receives this message and believes that its network attachment has changed. The mobile node forms a new care-of address with the information in the Router Advertisement message. Then it sends a Binding Update message to inform and register in its home agent. The home agent sends back a Binding Acknowledge message as confirmation of the Binding Update message. After receiving this message, the mobile node sends Binding Update messages to inform other correspondent nodes about its new care-of address.

This test passed successfully and the whole process took around 2 seconds.

- Home agent discovery:

First, the pseudo router is set to continue sending the fake Router Advertisement messages, which have the network prefix portion as a foreign network. After that, the Mobile IPv6 system in a mobile node is started. First, it actively sends Router Solicitation messages to force the pseudo router to send a Router Advertisement message. Then, it repeats the procedure in the last test case until it forms a new care-of address. After that, it sends a Home Agent Discovery Request message to its home network. The home agent gets this message and sends back a Home Agent Discovery Reply message. The mobile node continues the rest of steps in the last test case.

This test passed successfully and the whole process took around 3 seconds.

- Router detachment

After the mobile node finishes the binding procedure, the pseudo router is stopped. The mobile node starts to send Router Solicitation messages because it has not received Router Advertisement messages for a certain time.

This test passed successfully.

- Router detachment

After a certain time, a binding entry in a mobile node, home agent and correspondent node has expired. The home agent and correspondent node just simply remove the corresponding entry in their binding caches. In the meantime, the mobile node starts another binding registration process.

This test passed successfully.

- Security

When a mobile node sends a Binding Update message to its home agent, it uses incorrect security key to encrypt the message. After the home agent receives and verifies this message, it abandons this message. So the binding registration is failed.

This test passed successfully.

With the above test, it is verified that our implementation fulfill the design expectation including functionality and performance.

Chapter 5

Conclusion

5.1 Summary

Mobile IP lets the IP nodes (hosts and routers) use either IPv4 or IPv6 to roam seamlessly among IP sub-networks. Mobile IP supports the transparency of mobility to the layers above the IP layer, including the maintenance of active TCP connections and UDP port bindings. Mobile IP has different protocols and system structures for IPv4 and IPv6.

The implementation of Mobile IP is substantially dependent on the OS used by the IP host. For different OSs, implementations of Mobile IP vary considerably. Even with the same OS, the approaches of implementation may be substantially different since they may have different architectures and patterns. In this thesis, Linux is selected as the platform for the experiment. There are several system models for the implementation of Mobile IP over Linux. Each of them has its advantages and disadvantages. One of these models has been selected and implemented in our experiment. In this model, some of the functions are executed in the Linux kernel space and others in the application space. The implementation partially implements the Mobile IPv6 over Linux.

In our model, the Linux kernel is enhanced to capture the IPv6 packets containing binding messages in the IPv6 Destination Headers. Some pieces of code are inserted into the IPv6 module in the Linux kernel space to achieve this goal and send these packets to the application space via raw sockets. The control part, such as the binding management module and the ICMPv6 listener module, exists in the application space.

In our implementation, a Mobile IPv6 host (mobile node, home agent or correspondent node) listens to a particular raw socket and receives the IP packets containing binding messages. In the Mobile IPv6 node, two binding caches are used to store all binding information. One exists in the application space and the other exists in the kernel space. They have exactly the same data and are shared by the modules in the application space and the kernel space. The one in the kernel space can be used by modules such as traffic management. A mobile node or home agent listens to the ICMPv6 messages, which carry Home Agent Discovery messages and Router Discovery messages, via a raw socket. With these messages, a mobile node can detect its movements, obtain a care-of addresses, and perform the home agent discovery. In our implementation, a care-of address is formed by the sub-network's prefix and the mobile node's MAC address.

5.2 Contributions

Mobile IPv6 is an Internet proposal in process and is asking for comments. Our system partially implements the control portion of Mobile IPv6. It includes Home Agent Discovery, Location Detection, and Registration. Location Detection includes capturing the Router Advertisement messages, sending Router Solicitation messages, and obtaining care-of addresses. The Registration includes sending and receiving Binding Update, Binding Request, and Binding Acknowledge messages. This thesis implements these features and describes the approach in detail.

As mentioned before, there are a number of approaches for implementing the Mobile IPv6 on Linux. This thesis has considered the different approaches and selected

one of the approaches as the model for the implementation. In this approach, part of the functions stays in the Linux kernel space and the other part stays in the Linux application space. With experiments, our implementation proves the approach is suitable for implementing the Mobile IPv6 on Linux. This thesis describes the system and data structure of the implementation in detail. It also answers some technical concerns for the implementation.

Linux, as a clone of Unix, provides all standard Unix APIs of network programming, such as the Socket API. The use of these APIs varies with the different applications. This thesis provides some examples of code to explain some uses of these APIs to implement the Mobile IPv6.

Binding messages for Mobile IPv6 are transmitted in the Destination Header of any IPv6 packet. Linux partially supports the IPv6 Destination Header. Linux, however, does not recognize the content in this portion. There is no mechanism in Linux to capture the incoming IPv6 packets, which contain the binding messages in the Destination Header portion, and send them to the application space. To implement this feature, the system needs to enhance the kernel. One of the approaches is inserting some code in the Linux IP modules that process the IPv6 packets in the Linux kernel space. This thesis describes this approach and gives some samples of code.

In our implementation, the application needs to exchange some user-defined data between the application and the kernel. There are a number of ways to achieve this requirement. One approach is to extend an existing system call.

5.3 Future Work

There are many approaches for implementing the Mobile IPv6 in Linux. These approaches may be divided into several models according to where the implementation code exists in the OS space. This thesis has selected the approach that part of the code exists in the kernel space and the other part of the code exists in the application

space. Among these options, the approach, which all code exists in the Linux kernel space, is also a potential implementation model of Mobile IPv6. This approach works efficiently when there are large amounts of IP traffic. However, this approach has its disadvantages, one of which is code portability. Compared with the approach used in our experiment, more work may be required to implement this approach.

Since our implementation focuses on the control portion of Mobile IPv6, future research may include traffic management for this approach. The traffic management may include traffic tunnelling and other corresponding operations in home agent, correspondent node and mobile node. In our implementation, there are two binding caches and binding update lists. One is in the Linux kernel space and another one is in the Linux application space. They contain all the registration and management information that may be used by traffic management, such as the care-of address of mobile nodes.

Our implementation omitted the exception handling for some cases. In general, our system abandons or repeats the operation when an exception happens in our system or network. Some exceptions, however, must be considered since the abandoning or repeating operation may not solve the exception in some circumstances. For example, in our system, a mobile node forms a new care-of address by stateless way, in which the care-of address consists of the network prefix of a foreign network and the Ethernet address of this mobile node. The care-of address formed in that way may be duplicated in network. Since the Mobile IPv6 allows the use of this care-of address without performing the Duplicate Address Detection, this possibility is omitted in our system. The Mobile IPv6 RFC already gave some suggestions about processing the Duplicate Address Detection.

In our experiment, there is an assumption that there is only one home agent in a network for all mobile nodes. This home agent needs to handle the binding management and traffic management for all mobile nodes. If there are many mobile nodes in the home network, the home agent workload may be too heavy and cause some latency. One of the solutions is assigning this task to multiple home agents. Some work needs to be done for this mechanism such as the coordination structure

of multiple home agents. An example of this mechanism is the hierarchal structure of multiple home agents.

When a mobile node moves to a new network, the mobile node needs to finish some steps before it is able to send or receive IPv6 packets continually. The router discovery and the new care-of address obtaining are involved in this handover process. This process may cause some latency especially when a mobile node frequently changes its network attachment. To minimize the time of the handover process, IETF recently issued a new proposal: Fast Handovers for Mobile IPv6. This protocol enhances the base Mobile IPv6 and is intended to minimize the time during which a mobile node is unable to send or receive IPv6 packets. Future work may include the analysis and implementation of this protocol.

Bibliography

- [1] Randall Atkinson. IP authentication header. IETF RFC 1826, August 1995.
- [2] R. Rivest. The MD5 Message-Digest Algorithm IETF RFC 1321, April 1992.
- [3] Michel Barbeau. Wireless networks and protocols. W3C Note, May 2000.
- [4] A. Conta and S. Deering. Internet control message protocol (ICMP6) for the Internet protocol version 6 (IPv6) spec. IETF RFC 1885, December 1995.
- [5] E. Allen D. Haskin. IP version 6 over PPP. IETF RFC 2023, October 1996.
- [6] Charles Perkins. IP Encapsulation within IP. IETF RFC 2003, October 1996.
- [7] R. Droms. Dynamic Host Configuration Protocol IETF RFC 1541, October 1993.
- [8] Alex Conta and Stephen Deering. Generic Packet Tunneling in IPv6 Specification. IETF RFC 2473, December 1998.
- [9] R. Atkinson. Security Architecture for the Internet Protocol. IETF RFC 1825, August 1995.
- [10] S. Deering. ICMP router discovery messages. IETF RFC 1256, September 1991.
- [11] David Johnson and Charles Perkins. Mobility support in IPv6. IETF Internet Draft, November 2000.
- [12] G.Dommety. Fast Handovers for Mobile IPv6. IETF Internet Draft, March 2001.

- [13] C. Perkins and P. Calhoun. AAA Keys for Mobile IP. IETF Internet Draft, January 2001.
- [14] Tom Hiller. CDMA2000 Wireless Data Requirements for AAA. IETF Internet Draft, September 2000.
- [15] Yingchun Xu. Mobile IP Based Micro Mobility Management Protocol in The Third Generation Wireless Network. IETF Internet Draft, November 2000.
- [16] Stephen Kent and Randall Atkinson. IP authentication header. IETF RFC 2402, November 1998.
- [17] Scott Maxwell. *Linux Core Kernel Commentary*. Coriolisopen Press, 2000.
- [18] Charles Perkins. IP mobility support. IETF RFC 2002, October 1996.
- [19] Charles Perkins. *Mobile IP Design Principles and Practices*. Addison Wesley Longman, 1997.
- [20] Stephen T. Satchell and H.B.J. Clifford. *Linux IP Stack Commentary*. Coriolisopen Press, 2000.
- [21] James Solomon. *Mobile IP - The Internet Unplugged*. Prentice-Hall Canada, 1999.
- [22] Richard Stevens. *Advanced Programming in the UNIX Environment*. Prentice Hall, Upper Saddle River, NJ, 1997.
- [23] Richard Stevens. *Unix Network Programming*. Prentice Hall, Upper Saddle River, NJ, 1998.
- [24] E. Nordmark T. Narten. Neighbor discovery for IP version 6 (IPv6). IETF RFC 1970, August 1996.
- [25] E. Nordmark T. Narten and W. Simpson. Neighbor discovery for IP version 6 (IPv6). IETF RFC 2461, December 1998.

- [26] Thomson and T. Narten. IPv6 stateless address auto-configuration. IETF RFC 2462, August 1998.
- [27] Bakre, A and Badrinath B. *I-TCP: Indirect TCP for Mobile Hosts*. Proc. 15th International Conf. on Distributed Computing Systems(ICDCS), May 1995.
- [28] Dayem R. *Mobile Data and Wireless Lan Technologies*. Prentice Hall, Upper Saddle River, NJ, 1997.
- [29] Yu Wang. *Support IP Multicast for Mobile Hosts*. Mobile Networks and Applications. January, 2001.
- [30] Xinhua Zhao. *Flexible Network Support for Mobile Hosts*. Mobile Networks and Applications. March, 2001.
- [31] Thomas Nielson. *IPv6 for Future wireless Network*. Wireless Personal Communications. June, 2001.
- [32] Jason Jue and Dipak Ghosal. *Design and analysis of a replicated architecture for supporting IP-host mobility*. Cluster Computing. 1998
- [33] Vineet Chikarmane. *Multicast support for mobile hosts using Mobile IP: Design issues and proposed architecture..* Mobile Networks and Applications. 1998.
- [34] Vipal Gupta. *Secure and Mobile Networking*. Mobile Networks and Applications. 1998.
- [35] Anne Fladenmuller. *The effect of Mobile IP handoffs on the performance of TCP*. Mobile Networks and Applications. 1999.
- [36] A O'Neill and G Tsirtsis. *Edge Mobility Architecture - Routing and Handoff*. BT Technology Journal. January, 2001