NAIST-IS-DD0761203

Doctoral Dissertation

Design of A Research and Development Platform for Mobile IPv6 based Protocols

Keiichi Shima

March 17, 2009

Department of Information Systems
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Keiichi Shima

Thesis Committee:

Professor Hideki Sunahara	(Supervisor)
Professor Suguru Yamaguchi	(Co-supervisor)
Associate Professor Kazutoshi Fujikawa	(Co-supervisor)
Associate Professor Youki Kadobayashi	(Co-supervisor)
Associate Professor Keisuke Uehara	(Keio University)

Design of A Research and Development Platform for Mobile IPv6 based Protocols*

Keiichi Shima

Abstract

The Internet is growing by combining various communication media. In the future, it is considered that the wireless connection, rapidly advancing recently, will be the core component of the Internet. The research goal is to design and evaluate the mobility platform for the mobile-centric future Internet. The platform adopts IPv6 and Mobile IPv6 as the base protocols because they are the only realistic options to support the large number of nodes in the future. There are two core points in the design of the mobility platform. One is the separation of signal processing and packet processing to make new protocol support easy. The other is the definition of a common communication interface between processing modules to help smooth information exchange. Using the platform, four different mobility extension protocols are implemented and evaluated. By implementing and operating them through the experiments using the live network have shown the adaptability of the proposed platform design. Third party researchers also use the platform, which objectively indicates that the platform is useful even for other researchers. Furthermore, a completely new protocol that did not exist at the design phase of the platform has been implemented and evaluated. With these results, the conclusion is possible that the proposed mobility platform design can be the future research and development platform for the mobile Internet technology.

Keywords:

IPv6, Mobile IPv6, Network Mobility, Extensible platform design, Mobility research infrastructure

*Doctoral Dissertation, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0761203, March 17, 2009.

Mobile IPv6ベースプロトコル用研究開発プラットホームの設計*

島 慶一

内容梗概

インターネットは最新の通信技術を取り込みながら拡大を続けている。今後は、急速に発展している無線通信技術を利用した、移動通信が接続の主流になっていくと考えられる。本研究では、移動通信を主体とした将来のインターネットに対応できるプロトコル研究開発基盤の実装設計とその評価を行う。莫大なノードを収容できるIPv6基本プロトコル、およびその上で動作するMobile IPv6を採用し、それぞれのプロトコルをBSDオペレーティングシステム用に設計、実装、運用することで基本環境の正当性を評価した。本基盤環境の特徴は2つある。ひとつは、シグナル処理とパケット処理の分離による、将来の拡張への柔軟性の提供、もうひとつは処理プログラム間に共通の情報伝達手段を提供してスムーズな処理を実現することである。本基盤環境を用いて4つの異なる拡張プロトコルが実装され、実際のインターネットを利用した実験を実施することで基盤環境の柔軟性が示された。本基盤環境は、他の組織での研究開発にも利用された実績があり、客観的な有用性を持っていると評価できる。また最後には、設計当時に存在しなかった拡張プロトコルを基盤環境上に実装できることを実証することで、本提案基盤が将来の移動通信研究開発活動の基礎となりうることを示した。

キーワード

IPv6, Mobile IPv6, ネットワークモビリティ, 拡張可能な実験基盤設計, 移動通信用研究開発基盤

*奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 博士論文, NAIST-IS-DD0761203, 2009年3月17日.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1. Background

When the communication resources were precious, it was natural to design a special method for better utilization of the precious resources. For this reason, many information network infrastructure providers were used to develop their own network designs and protocols for a long time. Recent wide deployment of the Internet Protocol (IP) technology provides a simple communication framework for any kind of information infrastructure and it is integrating all information infrastructures into one protocol, IP.

At the beginning, the evolution occurred for the wired infrastructure because the wired networks had a faster communication property than the wireless networks. Even though many wired network infrastructures moved from their dedicated network designs and protocols to the generic IP based system, the wireless infrastructures remained their own designs. The wireless infrastructure which had a slower communication property could not accept the overhead of the generic protocol, even though the idea of having a common protocol had many benefits, such as interoperability, simplicity, and cost performance.

Recently however, the progress of the wireless communication technology gave us much wider broadband infrastructures in the wireless environment than before. The IEEE 802.11-based technology is now providing 600Mbps communication speed, IEEE 802.16 (WiMax) technology provides over 70Mbps with about 50km communication range, and IEEE 802.16e (Mobile WiMax) provides over 20Mbps

communication speed for moving nodes. There is no doubt that the future wireless technology will give us much faster communication properties. They are still narrower than of the wired communication devices, however, the overhead of using IP protocol over them is now not a big issue any more.

2. Research Objectives

Considering the current wireless infrastructure and progress of the future communication infrastructure, a new mobility infrastructure which suits the all-IP world, is needed. In the future, we will see much more mobile/non-mobile Internet devices in the world. All the computing devices will be connected to the Internet. Also, not only computers and PDAs are connected, but also many computer parts such as CPUs, memory units, input/output devices, sound devices are expected to be Internet nodes in the future. Non-computer devices like papers or foods may be Internet nodes through some kind of proxy devices. The future network needs to support tons of Internet devices. As the base infrastructure of these entire situations, a new IP protocol which can connect as many devices as we need is necessary. The Internet Protocol Version 4 (IPv4) [57], which has led the current growth of the Internet, has a too small addressing space. The ideal and only solution is to deploy a new protocol which provides a wider addressing space. The Internet Protocol Version 6 (IPv6) [11] is the answer. In chapter 4, the work for investigation of the IPv6 specification and implementation as the base system of all the following works is discussed.

To support the mobility infrastructure, a mobility platform for research and development of new mobility functions should be provided. In chapter 3, a conceptual design of the platform is presented. The design proposes separation of protocol signal processing and packet processing, and provides an interface layer between these processing modules. Although it is not clear which mobility protocol will be deployed in the future network, there will be one of the base mobility protocols in the future. On the base protocol, many extension protocols and ideas to enhance the base technology will be developed. It can be assumed that as long as the base protocol is same, the core packet processing will not be very different even though new extension protocols are proposed. The separation will make it

easy to develop protocol signal handling code, and will keep separate function (the packet processing function) untouched. These two function blocks will be connected by the interface layer through which each block can access the other block. Although the proposed design can be applied to any mobility protocols, this dissertation will focus on Mobile IPv6 as an concrete application of the platform. The design will be evaluated by implementing Mobile IPv6 as one of the example platform of mobility. On top of it some related extension protocols will be implemented to show the extensibility and practicability of the platform.

Mobile IPv6 is the mobility support protocol for IPv6 used in this dissertation. As already mentioned, any kind of mobility protocol can be an example protocol for the evaluation of the platform; however Mobile IPv6 was chosen in this work. The important point when providing a mobility framework is that many mobile service operators must adopt the same framework. To achieve the point, the protocol must be an open standard protocol and must be easily available to anyone who wants to utilize the implementation. It is well-known that the existence of freely available implementations of a new protocol stack sometimes accelerates the deployment of the new protocol. For example, the TCP/IP protocol stack implementation distributed with the BSD operating system by University of California Berkeley played a big role in deployment of the IP technology. The IPv6 protocol stack developed by the KAME project [86], which is the successor of the implementation discussed in chapter 4, contributed to distribute the IPv6 protocol stack to the world with the BSD operating systems. From these studies, we have learnt that implementing and distributing a high quality standard mobility stack is a mandatory activity for realizing the future mobile infrastructure. In this dissertation, the Mobile IPv6 protocol stack which is the main mobility protocol discussed at the Internet Engineering Task Force (IETF) is focused on because of the reasons listed in section 3. This decision will make the platform more practical. The result of the designing and development of Mobile IPv6 is discussed in chapter 5.

When designing a future protocol, it is important to support the legacy protocols. In the real world, the widely deployed infrastructure cannot be replaced to the new system in one night. Even though the IPv6 and Mobile IPv6 are deployed widely, we will have to use many IPv4 applications as the heritage of

the legacy world. The important point is that some kind of gradual transition scenario for migration to the new infrastructure should be provided. As one of the experiments for the platform evaluation, this backward compatibility problem and the evaluation of solution using the platform are discussed. Chapter 6 proposes a transition scenario and mechanism based on the Mobile IPv6-based network mobility protocol. The solution does not focus on interoperability between IPv4 and IPv6. It is considered that both IPv4 and IPv6 will be used for many years simultaneously. The usage scenario proposed here is called as the dual-stack scenario. The proposal provides an easy introduction mechanism for the IPv6 protocol to existing IPv4 networks, and at the same time, a simple multihoming and fault tolerant property as an additional feature. Such an incentive will encourage existing IPv4 users to introduce Mobile IPv6-based technology and will make the protocol transition easier.

The next topic discussed in this dissertation is a performance problem. Mobile IPv6 was originally designed as a mobility mechanism for terminal nodes. However after the network mobility mechanism, which is an extension of Mobile IPv6, was developed, many people realized that the mechanism could also be used as a mobility mechanism for an entire network infrastructure. There is a performance problem when a moving node changes its attachment point from one place to another place. While the node is disconnected from the previous attachment point and trying to connect to the new attachment point, the network connectivity is lost. Using multiple network interfaces can ease the problem. The multiple interfaces can be used as a pseudo mobility network interface. This idea is similar to multihoming. For the wider deployment of mobility technology, the advertisement of such a usage scenario will be important. Chapter 7 discusses how the multihoming mechanism was implemented on the mobility platform, and the experiment which evaluates the validity of the mechanism. Through this experiment, the extensibility and practicability of the platform is evaluated.

For the wide range mobility service deployment, a global scale mobility operation mechanism is mandatory. In the future, more and more mobile devices will be connected to the Internet and the communication range will be expanded to the world scale. Since Mobile IPv6 highly depends on the home agent, which is a kind of proxy node, it causes a single point of failure problem and redundant

route problem between a mobile node and its home agent by nature. When the operation scale is small enough, the basic Mobile IPv6 specification can serve the mobility function well. However, when considering the operation which serves a large number of mobile nodes and mobile service networks which connecting all the continentals, problems arise. In chapter 8, the problems are explained and the solution for the problems is evaluated using the mobility platform. The solution was implemented as an extension function of the proposed mobility platform, and a pilot operation was performed as a part of the core network services in one of the largest network events in Japan to prove the solution is feasible. Through this work on the above solution, the adaptability of the platform to newly invented protocols is evaluated in addition to its extensibility and practicability.

Proposing a new infrastructure is easy. The difficult part is how to convince the players that the proposal is actually useful and operable. Even though the proposal is good, it is useless if it cannot be used in the real environment. To prove that a proposal is valid, the proposal has to be operated on the real Internet. Chapter 9 introduces the status of the mobility service operation on the proposed mobility platform and the status of integration approach of the mobility protocol stack to the widely distributed open source operating systems. These two activities are efforts to show that it is possible to deploy the mobility service using the proposed mobility platform implementation, and to make this implementation more available. The final goal of this activity is to prepare an environment where anyone can construct mobility service operation environment only using the widely distributed operation environment, without any additional software download or patching to existing software.

Finally, beyond the future mobility framework we need to seek next generation mobility environment. The IETF standard mobility protocol is chosen for the platform in this dissertation. The selection was correct from the engineering and deployment points of view. No real business players want to change the existing infrastructure drastically and no one wants to use a well-designed but completely incompatible protocol with the existing infrastructure. However, when considering next generation of the future environment, choosing conservative approach is not always the best way. As a final part of the future mobility platform research activity, more autonomous mobility mechanism than the Mo-

bile IPv6-based mechanism is considered. Since such a new idea cannot be easily deployed in a real world, the scenario discussed in this dissertation is limited to a small closed environment. The chosen environment is a disaster recovery environment. In such an environment, we need to establish a network infrastructure from scratch and to operate some information gathering devices for the rescue activity. The devices used in this scenario are rescue robots. Since robots are moving, a kind of mobility mechanism is necessary, however it is impossible to assume there is a stable network infrastructure as the Mobile IPv6 assumes. In chapter 10, the problems in the disaster environment are discussed and the requirements for such a new mobile environment are proposed as examples of the possible future directions of this research area.

3. Comparison of Mobility Protocols

In this dissertation, Mobile IPv6 is focused throughout the document. However, there are other mobility-related mechanisms for IP. In this section two related protocols are introduced and compared to Mobile IPv6.

The first one is the Host Identity Protocol (HIP) [44]. Similar to Mobile IPv6, HIP uses two different kinds of addresses to locate and identify a host. To locate a host, HIP uses an IP address as an entity to determine the topological location of the node (locator). To identify a host, it uses a Host Identity (HI), which is a public key of the node, as an entity to determine the identity of the node (identifier). All the communication between HIP nodes uses HIs as communication end points. Since HI does not change even though a node changes its point of attachment (locator), communication between nodes can survive the node movement. In real communication, HI is transformed into a HI Tag (HIT) using a one-way hash function. Since HIT consists of a 128bit number, it can be used as the source or destination address of an IPv6 packet. All the applications use HITs as their communication endpoints. The binding between HIT and the current locator is managed by the rendezvous mechanism. All the HIP nodes register their locator information to their rendezvous servers, whenever the locator information changes. When a node contacts to another node, it first retrieves the peer node's HIT and rendezvous server by using DNS. After that, it asks

the rendezvous server the latest locator information related to the peer node's HIT. The address conversion from HIT to locator is done in the HIP protocol-processing layer implemented in each HIP node, and hidden from the application programs. When the node moves and the locator of the node changes, it notifies the new locator to the peer node so that the peer node can update the binding information. If the notification fails, the peer node will ask rendezvous server for the latest locator information.

The benefit of HIP compared to Mobile IPv6 is that it has less communication path overhead. In Mobile IPv6, the communication path between a mobile node and its peer node may be redundant, since all the packets pass the home agent of the mobile node. Mobile IPv6 has a route optimization mechanism to solve the redundant path problem. However, since it requires the peer node to modify to support Mobile IPv6, it introduces an other restriction. In the sense of the redundant path problem, HIP does not have the problem in nature, since it starts the communication between locators of peers. The communication path is always the shortest path in the sense of routing metrics. The drawback is that HIP requires both nodes to support HIP. HIP itself does not have compatibility to the existing legacy IPv6.

Another interesting protocol is SHIM6 (Level 3 Multihoming Shim Protocol for IPv6) [49]. Actually, SHIM6 is a multihoming technology rather than a mobility protocol, as its protocol name represents. SHIM6 is originally designed to solve the IPv6 multihoming problem. When a node is connected to the Internet through multiple ISPs, the host will have multiple prefixes assigned by its ISPs. The reason why a node subscribes to multiple ISPs is to achieve the redundant connectivity in case of network failure. If a node has multiple external links for connectivity, even when one of the ISPs fails, the node can continue its communication using another subscribed ISP. The question is which prefix the node should use. Since a prefix is tightly bound to an ISP, one prefix used in a certain ISP cannot be used within another ISP. If the node is using ISP-A's prefix and ISP-A fails, the node can start a new connection using another ISP's prefix, but it cannot use ISP-A's prefix. SHIM6 is aimed to solve this problem. In SHIM6, a node uses one of its IP addresses as a fixed identifier of it. If the selected IP address becomes unavailable, for example due to the ISP failure, another avail-

able IP address is used as a locator. The applications on the node can use the identifier even though the ISP which gave the prefix information of the identifier is not available any more. The SHIM6 layer implemented in the node translates the identifier to one of the available locators when sending packets to destination nodes. On the reverse direction, the locator is translated to the identifier before packets are passed to applications. Apparently, this mechanism requires both peers to support SHIM6 protocol.

SHIM6 is not actually a mobility protocol. The initial address resolution in SHIM6 is done by the DNS system. That means that some of locators of the node must be registered to the DNS system, and at least one of them must be reachable before making a connection. Otherwise, the peer node cannot make an initial contact to the destination SHIM6 node. Because of this restriction, SHIM6 nodes are not expected to move to different subnets. Of course, this design is not a drawback of SHIM6, since its purpose is to provide multihoming function to nodes. But since SHIM6 can map the identifier and locators of a node, it can be used as a mobility protocol, once the solution of initial contact problem is provided.

Mobile IPv6 is more conservative compared to the above protocols. Similar to the above protocols, Mobile IPv6 has two entities to identify a node. One is the home address which is used as a node identifier; the other is the care-of address which is used as a locator of the node. Unlike HIP, the address format of the identifier is same as ordinary IPv6 addresses, and the addresses are Internet routable. While HIP uses non-routable identifiers (HIT) to make a connection, Mobile IPv6 usually uses routable addresses as its identifiers (home addresses). This decision provides a backward compatibility with the legacy IPv6. Considering the fact that both HIP and SHIM6 require both nodes involved in the conversation to support each protocol, the deployment hurdle will be low when Mobile IPv6 is used.

The drawback of Mobile IPv6 when compared to HIP is that the communication path may be redundant as discussed above. When compared to SHIM6, the same discussion can be applied. Although Mobile IPv6 has a mechanism to optimize the route, the mechanism will drop the backward compatibility feature from Mobile IPv6. Mobile IPv6 provides both path optimization and backward

compatibility as a kind of trade-off.

Table 1.1 shows the summary of comparison of the above three protocols. The core idea of recent IP mobility technology is separation of host identifier and host locator. In that sense, all three protocols discussed are supporting the idea. The difference is that the point which each protocol emphasizes. In HIP, the most important point is to provide secure communication between HIP nodes. To do that, all the IPsec mechanism utilizing the public key information embedded in the address encapsulates all the HIP communication. Since the identifier used in HIP is a public key of a HIP node, the addresses used in HIP (HIT) use a completely different routing space from the normal IPv6 address space. This caused the backward compatibility problem. In SHIM6, the primary goal is to provide the multihoming solution to IPv6 nodes. To do that, SHIM6 allows an IPv6 node to choose an arbitrary address as the identifier of the SHIM6 node. Since SHIM6 nodes use IP addresses assigned statically from the subscribed ISPs, the SHIM6 nodes cannot move to the outside networks which other ISPs manage. Mobile IPv6 considers connectivity to the existing IPv6 nodes as much as possible. The drawback is that it requires a home agent, which acts like the rendezvous server in HIP. Compared to the rendezvous server in HIP, the home agent in Mobile IPv6 has a bigger role. While the HIP rendezvous server basically manages the binding between HIT and locator, the Mobile IPv6 home agent proxies packets to/from mobile nodes in addition to the binding management. This feature is sometimes taken as a weak point of Mobile IPv6 and it is true, however, recent research shows that several supportive mechanisms to make home agents redundant will be possible [82, 75].

After more than 10 years have passed since the first IPv6 RFC was published, the world is now moving to the IPv6-based network. There are already many IPv6 enabled/capable nodes in the world. Considering this situation, providing the backward compatibility will be a benefit in the future mobile networking world. With this observation, Mobile IPv6 has been chosen as the core mobility protocol in the framework proposed in this dissertation.

Table 1.1. Comparison of HIP, SHIM6, and Mobile IPv6

	HIP	SHIM6	Mobile IPv6
Path redundancy	No	No	Yes (if route optimization is not used)
Address resolution mechanism	DNS and Rendezvous server	DNS	DNS
Possible single point of failure	DNS and Rendezvous server (can be redundant)	DNS (can be redundant)	DNS and Home agent (can be redundant)
Encapsulation overhead	Payload encapsulation (e.g. ESP encapsulation)	SHIM6 protocol header	IPv6-in-IPv6 encapsulation (for communication through home agent), or Routing header and Destination options header (for communication route optimized path)
Backward compatibility to normal IPv6 nodes	No	No	Yes (if route optimization is not required)

Chapter 2

Mobility Platform for Future Mobile Operation Environment

In this chapter, an objective explanation of the results achieved through this research activity is provided.

1. Common Communication Protocol

The number of devices connected to network will continue growing. The computers and global network connectivity have been available only to the people in developed countries for a long time. Though recently, because the cost of computing devices is going lower, the number of people who can access to the computing resource is growing. The advance of networking technology also has helped solving the digital divide problem. The cheaper network access methods makes it possible to connect rural regions or developing countries to the Internet. In addition to the computers such as desktop/notebook computers, we will see a lot of intelligent networking devices in the future such as network sensors.

IPv4 succeeded to connect all the computer networks that were scattered around the world. The researchers and engineers in IETF had already noticed that IPv4 would not be able to support the future growth of the Internet in early 1990's. Thus, the effort began to specify the new Internet protocol which can be used instead of IPv4 to support the future Internet. IPv6 was produced based on the discussion held in the IP Next Generation (IPng) working group at IETF.

Although IPv6 intended to solve several forecasted problems (or the points believed as problems), the most important enhancement compared to IPv4 was the extension of the address space. The other problems, such as the security function, autoconfiguration function, route information aggregation, were eventually concluded that just introducing IPv6 could not solve them. For example, autoconfiguration function is provided in IPv4 using the DHCP protocol, the security function such as IPsec is also available for IPv4, and the route information aggregation problem is still a problem in IPv6. In short, IPv6 is basically provides almost the same functions as IPv4. Even though, the extended address space is beneficial where a lot of networking devices connect to the Internet. In this context, it was the right decision to focus on IPv6.

To prepare the future explosive deployment of IP devices, IPv6 is required. At the time when the first IPv6 specification was published, the importance is recognized to start the activity to develop and verify the new protocol as the fundamental infrastructure for the future communication technologies. Since there was no other IPv6 implementation at that time, the first protocol stack had to be carefully designed so that the stack could be a reference implementation of following IPv6 implementations. The quality of the protocol stack provided at the initial stage of the protocol deployment sometimes affects the speed of the deployment and influences the level of understanding of protocol developers and users. To design a good IPv6 protocol stack and evaluate through the real implementation was the most important task to be done first.

2. Mobility Support Protocol

With more than 20 years of efforts, IPv6 is finally going to be deployed in the real world. The reason of the deployment is not because IPv6 has many advanced features, but because it has a huge IP address space. At the time of 2008, it was noticed that the IPv4 addresses would exhaust by 2011 highly prospectively [20]. That means that new IP devices can no longer acquire an IPv4 address once the address pool exhausts. The growth of the Internet is still on going. Since the future devices should be more mobile friendly owing to the advance of the communication and downsizing technologies, some kind of mobility support on

top of the Internet technology is necessary.

The research history of this area is long. In the middle of 1990's, the mobility protocol for IPv4 was already proposed and it was standardized around 2002. The protocol has not spread over the consumer devices, however, it has been deployed to provide a mobility solution for the network service infrastructure of a mobile data service (for example, the data communication service for mobile handsets) operators. The reason why the mobility protocol for IPv4 was not widely deployed to consumer devices is because the networking infrastructure for mobile devices was not matured yet. For long time, the wireless communication infrastructure had been used mostly for the voice communication and could not provide sufficient bandwidth as a computer network infrastructure. The connection of a computing device to the Internet through such a wireless service was happened only in a kind of emergency situation and was not a usual style. In this sense, it is possible to say that even though there was a wireless communication infrastructure, most of the networking devices were not 'mobile'. Most of the time, they were stick to one location, and when they were moving, the devices were disconnected from the network.

The situation is now changing. The IEEE802.11 wireless communication technology for computer networks has replaced the wire cables connecting computers to the Internet with radio waves. The communication speed of IEEE802.11 was initially only 2Mbps, but it was not very slow compared to the wired networking speed, which was 10Mbps (using Ethernet) at that time. Now the IEEE802.11 technology is going to provide 600Mbps bandwidth for mobile computers. Though The communication bandwidth is still lower than the wired networking devices (in 2008, 1Gbps is the most popular Ethernet communication speed for computers), it is no longer a slow connection. Out of offices, the wireless communication service technology is also advancing. The mobile voice communication carriers have replaced their core networks using IP-based technology and the replacement makes it possible to provide the high speed digital communication for their customers. Now more than 7Mbps bandwidth using the mobile telephone carrier networks (such as High Speed Packet Access, HSPA) are available at many places. The speed continues to advance. As a complement technology of these two wireless communication mechanisms, WiMax (IEEE802.16) recently attracts attention of

mobile service operators. Its cover range is smaller than HSPA but it gives faster connectivity. The IEEE802.16e, which is designed for mobile terminals, provides 21Mbps connectivity speed for moving terminals.

These technologies are not just replacement of wire cables. By combining these communication methods based on the current location and radio wave condition, the network connectivity is always gained. This is important since IP mobility function can show its potential only with the always-connected environment. The more the Internet becomes important in business and private, the more people demand to be connected to the Internet. To disconnect people from the wire cables and re-connect with radio waves is the only way to satisfy their demands.

The mobility support protocol in IPv6 (Mobile IPv6) was initially proposed in 1996. Since the dissertation focuses on IPv6 as a common infrastructure for the future Internet that can support tons of networking nodes, it is quite natural to choose IPv6-based mobility protocol to support the small mobile networking nodes in the future environment. When the work about Mobile IPv6 in this dissertation started, the revision of the draft specification was 13, and people at IETF were still discussing the specification. To move forward to prepare the future mobile Internet, the first action required in this area was to stabilize the specification discussion. Getting involved in the discussion at IETF to stabilize the specification on one hand, it is required to design the stack implementation and show it to other researchers/protocol stack implementers on the other hand, so that everyone can understand how the protocol works and how realistic the protocol specification was. Since the specification was drastically changing while it was being discussed, it was important to provide the implementation which supported latest functions.

From the implementation design's point of view, there was a question whether the stack should be implemented in the kernel space or in the user space. Each method has its own benefits and drawbacks. This dissertation compares and evaluates both methods to provide better understanding to the stack developers and researchers.

3. Importance of Publicly Aavailable Protocol Stack

It is said that the freely available open source software sometimes accelerate the deployment of the new technologies. One of the reasons of the successful IPv4 deployment was that UC-Berkeley distributed the BSD UNIX operating system source code with their own TCP/IP stack. The operating system attracted many academic people, and as the operating system was spread over the world, the TCP/IP stack became popular either. The similar thing happened for the IPv6 protocol stack that was developed by the KAME project, which is the successor of the IPv6 implementation produced through this dissertation activity. The source code provided by the KAME project was adopted by four BSD distributions (FreeBSD, NetBSD, OpenBSD, and BSD/OS) and spread to the world with these distributions. It was a symbolical event, because the operating systems, which derived from the Berkeley Software Distribution (BSD) which provided the initial TCP/IP code, adopted the next generation IP code developed by the people grown with the BSD operating systems, and re-distributed the next generation IP code to the world again.

As a part of important aspects of the activity to build the future mobility research and development platform, the freely available open source software will be a core part of the components.

4. Transition Scenario

The deployment of a new protocol always has a transition problem. The deployment burden of IPv6 is a good example. IPv6 has a big advantage of the huge IP address space. However, it will not be deployed until people face the real problem, that is, the exhaustion of IPv4 address. Although the goal of this dissertation is to provide the future mobility research and development platform, the deployment scenario is worth to consider. The infrastructure which will never be deployed has no value. Fortunately, the mobility technology can be also a good supportive program for the IPv6 deployment. One of the problems in IPv6 is its lack of multihoming function. In IPv4, multihoming is achieved by advertising the routing information of the provider independent (PI) address space to the global Internet

through multiple ISPs. This mechanism works, however, it increases the size of the global routing table and throws a serious problem of memory exhaustion of the Internet core backbone routers. IPv6 initially tried to solve this problem by posing the strict routing information aggregation, however, the attempt failed.

The essence of the mobile technology is to provide the fixed address space independent of the attachment points of nodes or routers. This feature makes it possible to assign a fixed address space to a router, which can be considered an entire site which is covered by the same routing prefix, while the router can change its point of attachment. If multiple points of attachment, which are the entrances to the Internet, are established through different ISPs, then the router can 'move' from one ISP to another ISP when there is a problem in the former ISP. Although the solution still have a single point of failure which derives from the nature of the Mobile IPv6 base protocol, it can be used as a cheap multihoming solution for small sites. This will attract the existing customers who want the multihoming solution with small amount of investment. Though the solution proposed in this dissertation is based on the IPv6 network mobility function, IPv4 traffic can also be handled, because Mobile IPv6-based mobility architecture is a kind of overlay networking architecture using IP-in-IP tunneling. Both IPv4 and IPv6 can be used on top of the IPv6 mobility technology. This accelerates mobility deployment, and at the same time, encourages people to move to IPv6-based networking service.

5. Operation Technique

This dissertation tries to avoid just making an experimental system when proposing the mobility research and development platform. The real platform should have a real service operation and real users to gain the realistic result. Building and operating the service is one of the core activities of this entire research activity. From the operation, operation experience will be achieved and some operational issues may be found, which will be useful in designing the mobility infrastructure and the future service operation.

6. Global Scale Operation Technology

The final goal of infrastructure building is to create a mobility world. To construct a real service, the involvement of ISPs to the mobile service area is necessary. The Mobile IPv6 base specification depends on the home agent, which is the central forwarding agent, for its service availability. This poses a serious problem that it can be a single point of failure. In addition, since the forwarding agent is responsible for all the traffic sent to/from mobile nodes, the communication path sometimes longer than in the case not using mobile service, especially when the service is provided worldwide. The solution idea for the problems is simple. Because the problems occur at the home agent, duplicating home agent is the only solution. Similar redundancy mechanism is used in other various technologies. For example, to make routers redundant, the Virtual Router Redundancy Protocol (VRRP [25]) is used. For Mobile IPv6, the Home Agent Reliability Protocol [82] is proposed for local redundancy. For this case, in addition to the local redundancy, the global scale redundancy mechanism is required. For global scale redundancy mechanism, the cache-based mechanism is usually used. The Domain Name System (DNS) is one of the oldest global scale databases which have redundant property in their mechanisms. The records provided by the DNS system are distributed and cached worldwide, and even though some of the data holders stop working, another data holder caching the original data takes over the broken server. Another example is the web cache system. The cache servers are distributed over the world and keep registered contents. The web contents retrieval request is properly directed to the nearest cache server by using the location-based DNS response. If one of the cache servers goes down, then the DNS response is changed to point another working cache server.

For the mobility service, a similar approach can be taken. The point is that the redundancy mechanism must not break the Mobile IPv6 base specification. For example, the cache mechanism may not be a proper solution, since that kind of system usually has a synchronization delay. Because mobile nodes moves more frequently than the DNS record update interval, the location information of a mobile node must be quickly updated at all the distributed home agents, otherwise data packets sent to the mobile node will not be delivered to the correct locations. Also locating the correct home agent from distributed home agents is

another challenge. A mobile node must reach to the nearest home agent, and when the mobile node moves to another location where another home agent exists, the mobile node should move to that home agent. This kind of load balancing and fault tolerance mechanism, and automatic path coordination mechanism are necessary when deploying a large-scale business operation.

Chapter 3

Platform Design and Experiment Plan

This chapter provides the design of a mobility platform and its evaluation plan. Although the actual platform focuses on Mobile IPv6, the conceptual design can be applied to any other mobility protocols. In section 1, the goals and requirements of the platform are provided. Based on the requirements, the design of the mobility platform is proposed in section 2. Section 3 explains the experiment plan to evaluate the platform implemented using the proposed design. Some real problems happening around Mobile IPv6 and the solutions to solve them are focused. Implementation and experiment plan for each solution is defined in this chapter, and the platform is evaluated based on the result of the implementation procedures and experiments in later chapters.

1. Requirements for Mobility Platform

In this section, the requirements for the mobility platform is clarified. The speed of the advance of wireless communication technology and downsizing technology will bring us a mobile-centric world. In such an environment, there will be various mobility-related communication technologies. The mobility platform should be able to adapt these new technologies. The goals of the platform are as described below.

Goal 1. The platform must provide the environment where it is possible to build and evaluate ideas quickly.

Goal 2. The extension process should not introduce a drastic modification to the base platform.

Goal 3. The platform and the extension protocols build on top of the platform should not introduce a degrade of communication performance.

Goal 4. The platform must be reusable by other researchers and developers to test their ideas on the platform.

The technology development speed becomes faster and faster. Especially, the protocols used in the Internet world are usually developed quicker than other well-organized protocols defined by internationally organized committees. Because the benefits and usefulness of the Internet technologies are often proved by the running code, providing a development environment where new ideas can be evaluated quickly is important. The same discussion can be applied to the mobility protocols too. Since the mobile communication networks are also moving to the all-IP based networks, the platform must support the trend.

When deploying a new scenario, there is usually one base infrastructure protocol. In the Internet case, the *Internet Protocol* is the base protocol. In the mobility platform, Mobile IPv6 is chosen as discussed in section 3. So the base platform supports Mobile IPv6 as its fundamental function. All the extension protocols will be build on top of the Mobile IPv6 base platform. In this case, the developers should be able to implement their ideas on the platform with the minimum modification of the platform. What the developers and researchers want to evaluate is their new ideas, not the platform itself. If the extension process requires a lot of modification of the base platform, it will cause more burden than benefit. They may have to handle problems introduced by modifying the base platform, which are out of their interest.

Although the platform is a testbed for the new ideas and protocols, it cannot be a reason of that the platform does not need to provide better performance. The platform should also be a realistic environment for the real usage, such as for the daily working environment, or the commercial service. This requires that the

mechanism implemented in the platform should not introduce packet processing performance degradation because of the extensions.

The platform demonstrates its value when it is used as the base system of many research and development activities. It is easy for the designer to use his platform by himself since he knows everything about it; however just using the platform by its designer does not help accelerate the research/development activity. The platform must be widely published and have an easy interface for the third party developers and researchers to use it.

To achieve the above goals, the following requirements in platform designing arise.

Design Requirement 1. Separation of signal processing and packet processing (for Goal 1 and 3)

Design Requirement 2. User space implementation of signal processing and kernel space implementation for packet processing (for Goal 2 and 3)

Design Requirement 3. Communication interface between function blocks and users (for Goal 1, 2, and 4)

The first requirement is separation of the packet processing function block and the signal processing function block. This design requirement is necessary to achieve the goal 1 and 3. The packet handling is performed based on the routing and forwarding information. The information is calculated by the signal processing function block and injected to the packet processing function block. Even though a new protocol extension is introduced, the packet handling part is usually not affected. Developers of new protocol extensions need to take care of the signal processing block only by separating these function blocks. Because they can concentrate on the protocol signal handling part, the development will go smooth. The separation keeps the packet handling part isolated and the developers do not need to touch the block. Since the packet processing part is not modified by the new protocol developers, it can keep the performance regardless of the introduction of new signal processing mechanisms.

The second requirement mentions the location of the function blocks. The signal processing function should be placed in the user space, and the packet

processing function should be placed in the kernel. This requirement achieves the goal 2 and 3. The programs running in the user space do not have strong privilege, while the kernel can do anything. With this requirement, the signal processing part, which is the biggest part in developing a new protocol extension, is placed in the user space. Because the user space programs do not have rights to access important parts required for the stable operating system operation, the entire system can be stable even though the new signal processing function is not mature or has some bugs. This will make the development and testing much easier, and will save the system in operation when there are some bugs which cannot be detected while debugging.

The third requirement achieves the goal 1, 2, and 4, by providing a common communication interface between function blocks and users. Such an interface is necessary, for example, when putting routing/forwarding information from the signal processing function block to the packet processing function block. The other direction is also necessary to check the current routing information. The communication mechanism is used not only between the signal processing part and the packet processing part, but also between the modules in the signal processing space. There may be multiple signal processing programs in the user space based on the functions. For example, there may be a mobile node program which only handles signals used by moving nodes, and a network movement detection program which dedicates monitoring the current network attachment status and notifies the status to the mobile node program. In this case, the communication interface is used to exchange information between them. This interface is also important for the third party developers and researchers. They may want to implement their own signal handling programs but may want to use the existing packet processing function. By making the interface standard and open, the platform can be re-usable not only for the designers but also for many other mobility researchers and developers.

In summary, the following three points are the requirements for the mobility platform.

1. Separation of signal processing and packet processing
2. User space implementation of signal processing and kernel space implementation for packet processing

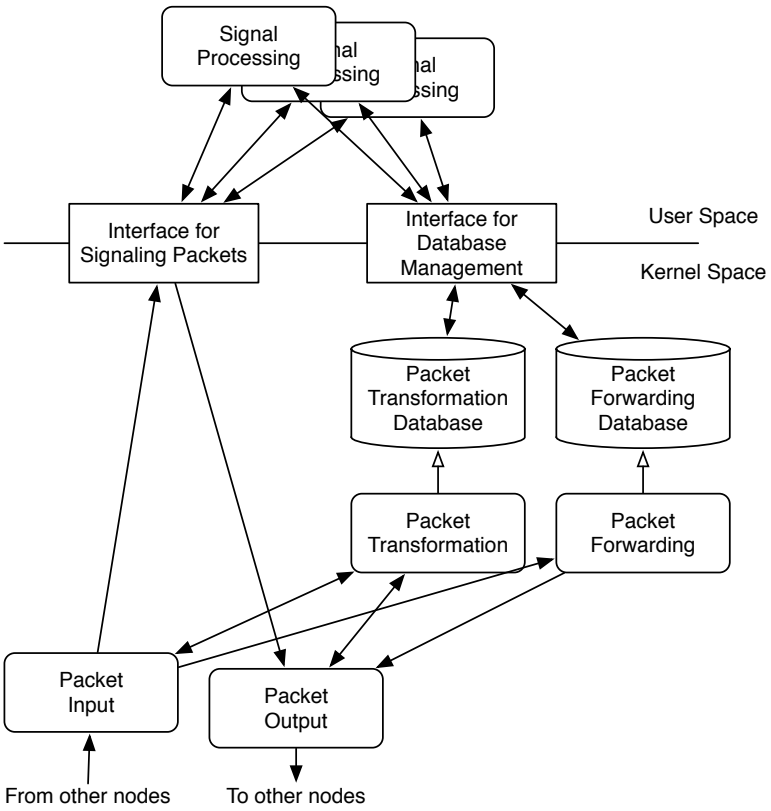


Figure 3.1. The mobility platform design diagram

3. Communication interface between function blocks and users

2. Design of Mobility Platform

Based on the requirements, the system is designed as shown in figure 3.1. In the user space, the signal processing programs for mobility operation are located. These programs send and receive mobility protocol signaling messages through the interface for signaling packets. In the beginning of the specification discussion of Mobile IPv6, there is an implicit assumption that the kernel should handle the signal processing. However, when the specification was being sophisticated through the discussion at IETF, the possibility appeared that signal processing can be done in the user space. In this design, that approach is taken as discussed in the previous section. To do that, a mechanism to send or receive mobility

signaling packets is necessary. As a part of the output of this dissertation, a contribution has been made for the standardization procedure of the Extension to Socket API for Mobile IPv6 [6], to create the mechanism. The implementation explained in this dissertation is one of the first implementations of the socket interface.

In addition to the signal packet interface, there is another interface for mobility function management. This interface is used to set or get packet transformation information and packet forwarding information in the kernel. Packet transformation is required when sending out packets. The mobility-related information is added to the outgoing packets. In Mobile IPv6, the care-of address information and/or routing header information are added if necessary. On the incoming case, this information are also understood by the packet transformation module and passed to the upper layer in the node, or forwarded to another node. The transformation module also performs IP-in-IP encapsulation between a proxy node and a mobile node. The packet forwarding database is looked up when a packet being processed needs to be passed to another node. Such transformation and forwarding information are calculated based on the exchange of mobility signaling packets. When the user space programs complete the signal exchange procedure, and new information is acquired, the information is updated through the management interface whenever necessary.

The design in not limited to the Mobile IPv6-based architecture. Whenever the target protocol requires the same requirements, the similar design can be applied. For example, HIP or SHIM6 also be able to be implemented in the same design. However, in this dissertation, the focus is Mobile IPv6 and the actual implementation only supports Mobile IPv6 and its family protocols. Since the actual packet transformation varies in each mobility support protocol, one generic processing module cannot support all the mobility protocols.

3. Focused Problems and Experiment Plans

The focus of this dissertation is to design a mobility research and development platform. To evaluate the platform, the following problems is focused in this document.

1. IPv4 network transition problem
   In this topic, the extendability of the mobile platform will be evaluated.

2. Service disruption problem during handover
   In this topic, the extendability of the mobile platform is evaluated, and the applicability of a mobile network technology is explained.

3. Home agent redundancy problem
   In this topic, the extendability of the mobile platform is evaluated.

The first problem is that the NEMO Basic Support, which is an extension of Mobile IPv6 to support mobile network, does not support IPv4 network prefix. That means that if a user introduced NEMO BS protocol to make his network mobile, he has to use the IPv6 only network. This limitation prevents people from using the mobility technology. The author proposed the idea to extend the NEMO BS protocol to support IPv4 mobile network in [64]. To solve this problem by using the mobility platform, it is evaluated if the platform can be extendable to support a newly proposed protocol in this dissertation.

The second problem is the service disruption problem when a mobile router is moving one network to another network. The solution of this problem is proposed in [81] and [45]. In the proposals, Mobile IPv6 is extended to support the multiple care-of addresses registration function. In the basic specification, a mobile node can only register a local address at once. Because of this, when a mobile node moves from one network to another network, there appears a duration where the node cannot access to the Internet. By registering multiple local addresses, a node can prepare a new network before detaching from the old network. During the experiment, the multiple care-of addresses extension support is added to the platform, which will show the extendability of the platform. In this experiment, the applicability of the network mobility technology is also shown. The network mobility technology can be applied to various scenario, such as the Personal Area Network (PAN), the Vehicle Area Network (VAN), the Transportation Networks (Train Car Network, Ship Network, and Airplane Network), and the Site-Level network. Each usage scenarios has each assumed number of network nodes inside. For example, in the PAN case, personal devices such as computers and PDAs are assumed and the number of devices is a few to 10. For the train network, each

car will carry 80 to 100 people. If every passenger has one's own computer, then the assumed number of network nodes will be around 100. In the experiment, a conference network which has more than 200 people in it is used as a test case, which will be realistic size as a middle-sized mobile network (for example, the train network case).

The third problem is the service redundancy problem. Since Mobile IPv6-based protocol is a kind of tunneling protocol, the tunnel server (called as a home agent in Mobile IPv6) can be a single point of failure. The solution which is now being discussed at IETF is to locate multiple home agents around the Internet and connect the home agents each other with IP-in-IP tunnels. Using the tunnels, the home agents establish an overlay network for inter home agent communication. The nearest home agents accommodate a mobile node. Once the serving home agent fails, the other home agents in the same home agent overlay network will start serving. During the experiment, one prototype idea of the home agent redundancy mechanism is implemented, which shows the extensibility of the platform consequently.

Table 3.1 summarizes the experiment plan.

Table 3.1. The experiment plan for evaluation of mobility platform.

Problem	Evaluation topics	Experiment plan
Lack of IPv4 network support	Extendability of the proposed mobility platform	Extend the base mobility platform to support IPv4 mobile network [64], and evaluate the mechanism in a laboratory scale network.
Service disruption while moving	Extendability of the proposed mobility platform, and evaluation of the network mobility applicability for the middle-sized mobile network	Extend the base platform to support the multiple care-of addresses registration mechanism [81], and evaluate the mechanism with a conference network which contains more than 200 people.
Home agent redundancy	Extendability of the proposed mobility platform	Extend the base platform to support the home agent distribution mechanism discussed in chapter 8, and evaluate the mechanism at one of the largest network vendors/service providers exhibitions [27].

Chapter 4

Implementation Design of the Internet Protocol Version 6 (IPv6) and its Evaluation

1. Background

The growth of the Internet caused the exhaustion of the IPv4 address space. As the long term solution of the address exhaustion, the first version of protocol specification for the new Internet Protocol was designed in 1995 as IPv6. In this chapter, the Internet Protocol is described with a version number explicitly (for example, 'IPv4' or 'IPv6') when necessary to distinguish IP version, otherwise the protocol is described just as 'IP'.

The following list is the benefits of the IPv6 expressed originally. However, during the deployment process of the IPv6, some of the features have been proved to be not significant benefits compared to the IPv4. At this moment, it is widely understood that the biggest and only benefit of the IPv6 is its huge address space.

1. Huge address space: The node identifier of the IPv6 has the 128bits address space. The total address space is $2^{96} \simeq 8 \times 10^{28}$ times bigger than that of the IPv4, since IPv4 node identifier uses the 32bits address space.

2. Simplified header format: Based on the operation experience of IPv4 over 20 years, some of the header fields which are not used frequently were removed.

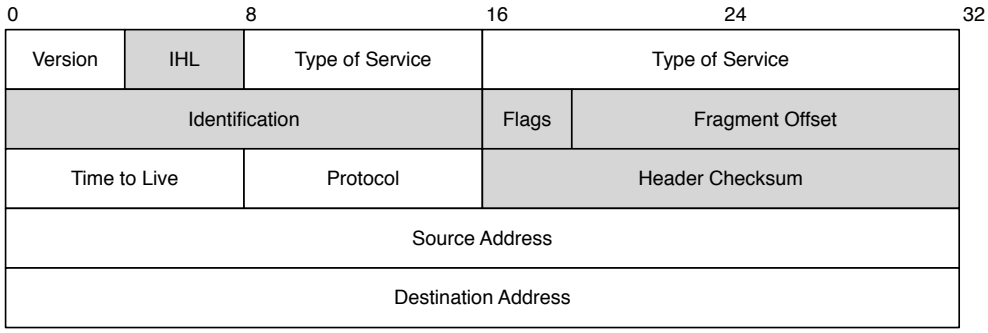


Figure 4.1. The header format of the IPv4

Table 4.1. The header fields in IPv4 and IPv6 which have the same meaning with different names.

The name in the IPv4 header	The name in the IPv6 header
Type of Service	Traffic Class
Total Length	Payload Length
Time to Live	Hop Limit

Figure 4.1 shows the header format of the IPv4 and figure 4.2 shows that of IPv6. In figure 4.1, the shaded fields were removed in the IPv6 design.

Note that some of the IPv4 header fields were renamed and kept in the IPv6 header. Table 4.1 shows the list of the header fields that have the same meaning but different names. Although the size of the IPv6 address is 4 times larger than that of IPv4, the total header size of the IPv6 is suppressed 2 times larger than the IPv4 header size. This is the effect of the removal of uncommon header fields of IPv4.

3. Generalization of option headers: In the IPv4 specification, the option headers are defined as a part of the IPv4 header. The option header format is a form of the TLV (Type Length Value) format, which increases processing overhead especially when forwarding packets on an IPv4 router. In addition, the maximum length of the IPv4 option is limited to the maximum size of the IPv4 header, which is 64 bytes. The size limit prevents to define an option header which may require a large data part. In the IPv6

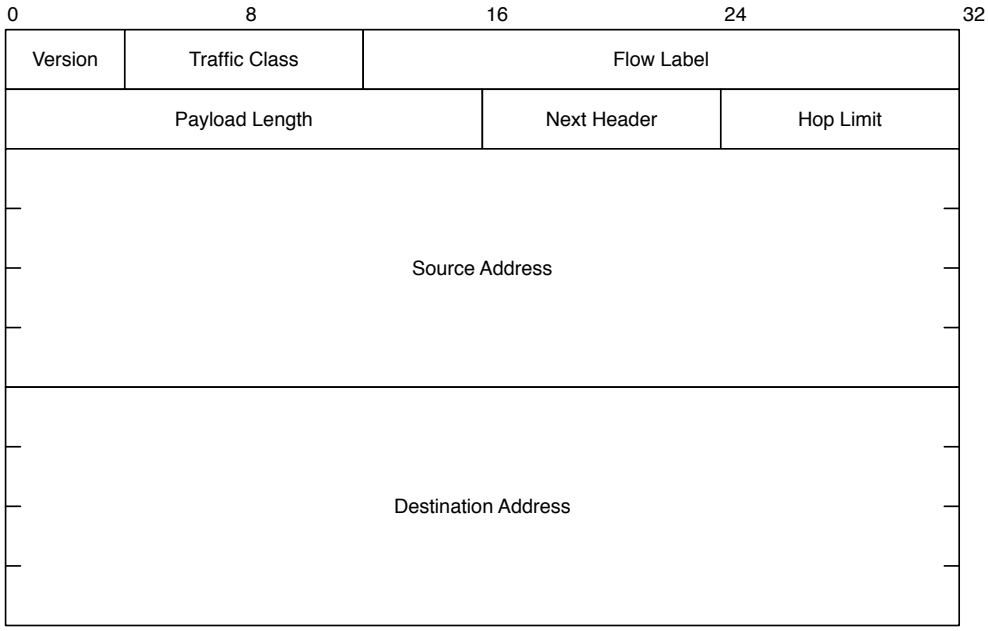


Figure 4.2. The header format of the IPv6

specification, the option header field is removed from the core header part and each option defines its own option header format as a separate header. The design policy of option header at the beginning was to specify a fixed length header for each option to make the processing cost of option header low. However, at this time of writing, some option headers length are variable. For example, the Destination options header has many sub-options in the header, and as a result, the total option header length cannot be fixed any more. However, because the processor speed has increased and the implementation technology of hardware-driven header processing has been improved, the variable length option header is not a problem recently. Even the situation changes, the idea to split the core header and option headers is still effective, since it is possible to use as many option headers as necessary as long as the total packet size does not exceed one IP datagram unit, making future extensions easier than IPv4.

Figure 4.3 depicts the general format of the option header.

4. Flowlabel: The flowlabel field is intended to group a series of packets as

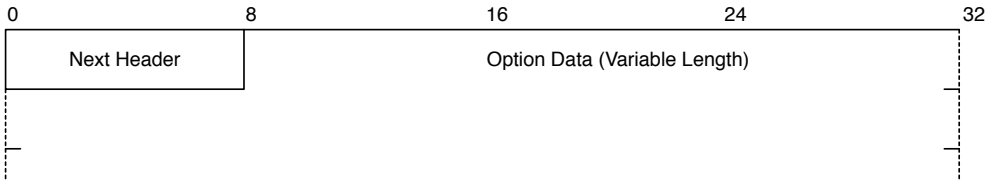


Figure 4.3. The option header format of the IPv6 specification

one continuous data stream. In certain conditions, the flowlabel field can be used as a key value in looking up the forwarding information table to determine the outgoing interface on a routing device. RFC1809 [51] proposed a usage of the flowlabel field, however, at this moment this field is not widely recognized nor used effectively.

5. Authentication and Confidentiality: Although the IP security (IPsec) mechanism was widely adopted and used in the recent Internet, the mechanism is not deployed in a real operation environment at the time when the IPv6 was being discussed. The IPsec mechanism [34], IP Authentication mechanism [32] and IP encryption mechanism [33] were defined as mandatory features of the IPv6. This feature encouraged spreading the IPsec implementation with the IPv6 base protocol stack, even though it was not used at the initial stage of the IPv6 deployment. The enforcement of providing security function within the base specification was a good idea.

2. Purpose of Designing and Implementing IPv6

Designing and implementing IPv6 has the following purposes.

1. Propose the best practice of the IPv6 protocol implementation design and prove the interoperability of the IPv6 specification by implementing the protocol.

2. Discover issues in the IPv6 specification.

3. Build the IPv6 research/development environment for further activities.

In the protocol standardization process at the IETF, a new protocol specification is required to have more than 3 different implementations which are interoperable each other; otherwise the new protocol cannot be published as a standard protocol. Providing at least one implementation in the beginning of the protocol standardization process is one of the important activities.

Implementing protocol stack sometimes reveals mistakes in the protocol specification. In the Internet standardization process, nobody tries to complete a perfect specification before making an implementation. Instead, historically the approach has been taken to work on the running code and make updates based on the real code. In such a realistic research and development process, it is important to check the specification does not have any serious issues by implementing and operating the protocol.

More than 10 years have passed since the first revision of the IPv6 specification was published. Now the exhaustion of the IPv4 address space is expected and many people believes that the true solution is to deploy IPv6, which has a huge address space. It is no doubt that the IPv6 is the next generation of the IP protocol which supports the Internet infrastructure. Building the base research and development environment by providing a working protocol stack has been important, and now it becomes more and more important considering the Internet is now facing the migration period from IPv4 to IPv6.

This chapter consists of the following sections. In section 3, the design and implementation details of the IPv6 stack are proposed. Section 4 discusses ambiguous points in the protocol specification found through the implementation and operation work, and proposes solutions for the issues. Section 5 reports the result of the interoperability test done with three Japanese organizations (Murai Laboratory of Keio University, Hitachi Corporation and Sony Corporation). The section also reports the result of the international interoperability test held by the Interoperability Laboratory at the New Hampshire University, U.S.A. from February 5th to 10th, 1996. Section 6 evaluates this activity and section 7 summarizes the design and implementation work of IPv6.

3. Implementation of the IPv6 stack

This section describes the design of the implementation of the proposed IPv6 stack, based on the function blocks listed below.

- Interface layer
- Network layer
- Transport layer
- Socket layer

The stack was implemented on the BSD/OS operating system provided by BSDI Inc.[1] which is based on the 4.4BSD Lite implementation. Working on this task, BSD/OS was the most stable BSD-based operating system with commercial support. At that time, there were other BSD-based operating systems such as FreeBSD and NetBSD. However these operating systems were still in their initial development stages and sometimes introduced unexpected unstability which was not derived from the networking stack. For the initial reference implementation, stability of the operating system was important to concentrate the development of the network part. The implementation developed as a result of this dissertation work was taken over by the KAME project [86] later, supporting other BSD operating systems (FreeBSD, NetBSD, OpenBSD).

3.1 Implementation Design

Even though the IPv6 packet processing flow is similar to that of the IPv4, IPv6 is not compatible with IPv4. As we can see in figure 4.1 and 4.2, their header formats are completely different and they do not have a shared part for backward compatibility. When implementing a new protocol whose header format is the only difference and the other parts (the upper layer protocols processing) are similar to the existing one, we can choose either one of the following two approaches.

1. Extend the IPv4 packet processing code to support the IPv6 header format.

[1]The company does not exist now.

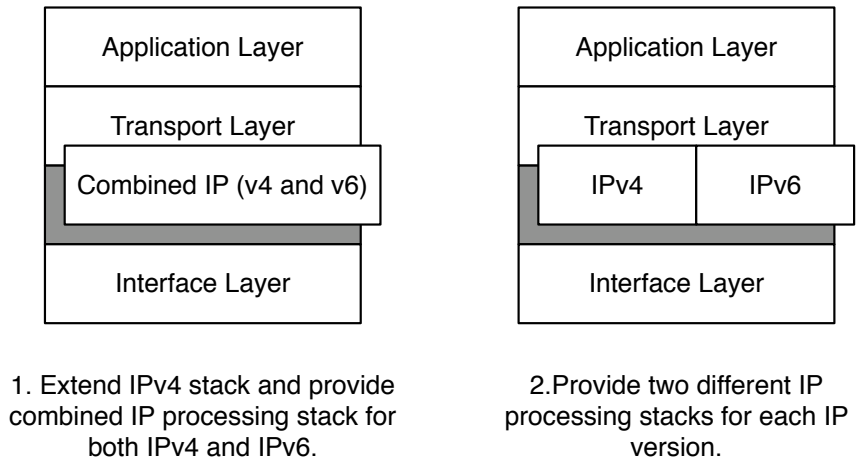


Figure 4.4. Two approaches to implement a new protocol stack

2. Build a new IPv6 packet processing code.

Figure 4.4 shows the two ideas.

In this activity, the second approach was chosen. IPv6 is considered to take over the role to be the infrastructure of the Internet from IPv4. Since one of the intention of this research was to provide a free reference code for the future research and development, separating the IPv4 legacy code and the future code will help people to have clear understanding of how the protocol is implemented. The drawback is that some problems may happen when implementing a cooperative action between IPv6 and IPv4. For example, the IPv4-mapped IPv6 address usage described in RFC2553 [21] which enables programmers to use IPv6 socket for both IPv4 and IPv6 communications is one of such cooperation scenarios. It was considered that it would be easier to implement such a function if the IP stack was implemented as a hybrid stack of IPv4 and IPv6, rather than separating IPv4 and IPv6 stacks. However, the later study proved that it is not necessarily be a hybrid stack, and as a proof, most of the recent IP stack has separate IPv4 and IPv6 stacks. Even though these protocol stacks are implemented separately, most of them provide the IPv4-mapped IPv6 address function. Considering the result, the conclusion is possible that the direction which was taken when designing the IPv6 stack to have a separate IPv6 was correct.

3.2 Interface Layer

The role of the interface layer is to receive a data frame from a network device (such as an Ethernet device) and pass it to the network layer (such as the IP layer), and vice versa. In the implementation, the Ethernet interface adaptation module and the loopback interface module for IPv6 were designed and implemented.

The input data from a network device is usually processed using the hardware interruption mechanism. That is, when the data arrives at the network device, an interrupt signal is raised and the function prepared for the interrupt processing is called to handle the data. In the IP stack case, the data is queued in the IP packet input queue.

Basically the protocol input queue is prepared per protocol. When implementing an IPv6 protocol stack, the following two patterns could be possible.

1. Share one protocol input queue between IPv4 and IPv6.

2. Arrange a dedicated queue for IPv6.

Since IPv4 and IPv6 are both 'the Internet protocols,' it is not strange to share one protocol queue. As we can see in figure 4.1 and figure 4.2, the IP headers has a header field indicating the version number of the protocol. Even though sharing one protocol queue, it is possible to demultiplex packets based on their version number. At the time of this implementation, the IPv6 research team at INRIA built an IPv6 stack [18] on the NetBSD operating system whose protocol input queue was shared by the IPv4 and IPv6 stacks. In the implementation build in this dissertation, however, the strategy having separate queues for each IPv4 and IPv6 was taken because of the following reasons.

1. Code readability
   Although IPv6 is successor of IPv4, it does not have any backward compatibility with IPv4. Catching packets based on the different protocols in the same queue and demultiplexing later in the layer 3 processing code is confusing.

2. The difference of Ethernet frame type number
   The Ethernet frame type for IPv6 (0x86dd) is different from that of IPv4

(0x0800), even though both are defined as 'the Internet protocols'. This means even at the Ethernet level, these two protocols are treated differently. It is natural to keep the difference in the input queues and pass packets to the upper layers.

3.3 Network Layer

The IPv6 communication stack requires the following components to be implemented.

- IPv6 core processing
- ICMPv6 (Internet Control Message Protocol for IPv6) processing
- NDP (Neighbor Discovery Protocol) processing

ICMPv6 [9] is the IPv6 version of ICMP (Internet Control Message Protocol) [56]. The protocol is not just an adaptation of ICMP to IPv6, but also it has some new features which the ICMP does not have. The implementation design of ICMPv6 is discussed in section 3.5.

NDP [46] is a generalized version of ARP (Address Resolution Protocol) [54]. NDP is used to map the addresses used in a data link layer and the addresses used in a network layer. NDP is discussed in section 3.6.

3.4 Implementation of IPv6

The remarkable change in the IPv6 specification from the IPv4 specification is the size of the IP address and the option header processing method. In other words, there is only a little difference between the IPv4 and IPv6 in the rest of the protocol processing. During implementing the IPv6 stack, many IPv4 packet processing code are referred as sample code. In this section, the option header processing is discussed first, and the design of the protocol control block (PCB) for IPv6 which is the core design work when making a network layer protocol stack is discussed later.

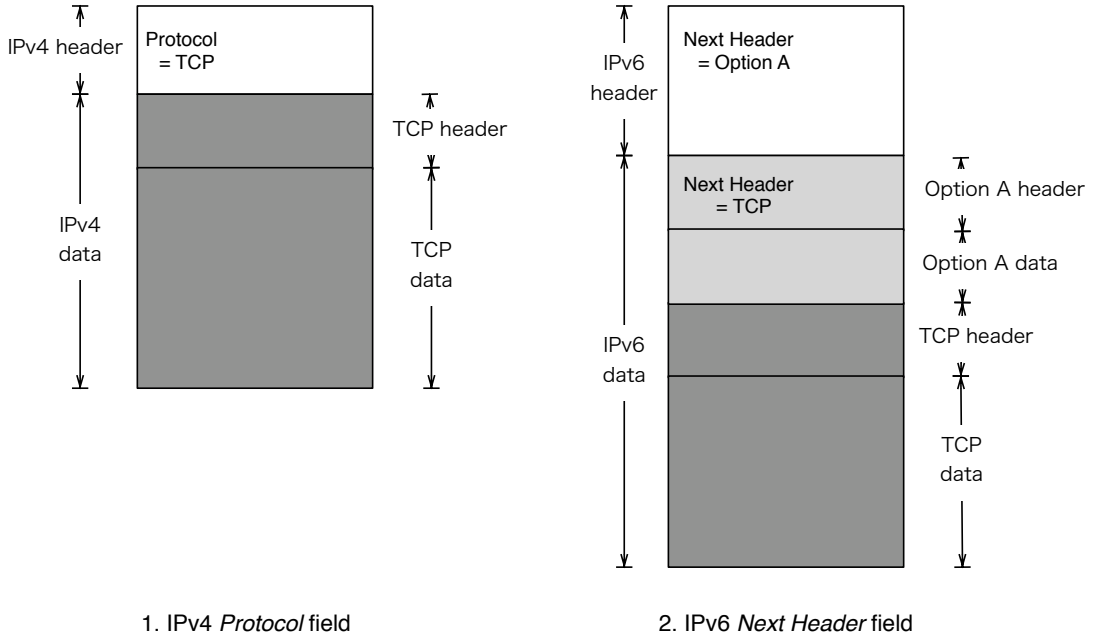


Figure 4.5. The Protocol field in the IPv4 header and the Next Header field in the IPv6 header

Option Header Processing

As explained in section 1, the option headers in IPv6 are separated from the core IPv6 header. It is different from that the option headers in IPv4 are embedded as a part of the IPv4 header. In IPv6, an option header is treated in the same way as the upper layer headers are treated. And, each option header has its own protocol number as the upper layer protocols (like TCP [58] or UDP [55]) have their own numbers. The Next Header field placed in each header identifies the following header block. The Next Header field is an equivalent to the Protocol field in the IPv4 header. The reason of the name change is due to the meaning change of the field. The field is not only indicating upper layer protocol, but also indicating following header including option headers.

Figure 4.5 shows the usage of the Protocol field in the IPv4 header and the Next Header field in the IPv6 header.

In the IPv4 protocol processing, packets are processed in the following sequence of procedures.

1. Check if there are any options in the IPv4 header.
2. Process options if any exist in the IPv4 header processing code.
3. Pass the payload to an upper layer processing code (for example, the payload will be passed to the TCP processing code in figure 4.5 case).

On the other hand, IPv6 does not have option headers in its base header. The header processing code can focus on the predictable header information with no exception. As we can see from figure 4.5, the structure of the IPv6 header and its option headers is similar to that of the IPv4 header and its upper layer protocol header. Since the headers are chained by the Next Header fields, the packet processing code can just process the current header and pass the rest to the protocol processing code of the next header.

1. Start with the IPv6 header processing code.
2. Pass the content to the protocol processing code specified by the Next Header field.
3. Return 2.

Finally, the upper layer protocol processing code (usually the transport protocol processing code, like TCP or UDP) processes the last part of the packet and the packet processing procedure completes.

Design of the Protocol Control Block

The protocol control block (PCB) is a set of parameters of the communication status. In the BSD networking code, each protocol has its own PCB structure. As discussed before, the difference between the IPv4 and IPv6 is mainly the size of address. As a result, the difference of the PCB structures of these protocols is small. When designing the PCB for the IPv6, two possible ways exist.

1. Extend the existing IPv4 PCB and share the same structure. (Figure 4.6 left)
2. Define a new PCB structure dedicated to the IPv6. (Figure 4.6 right)

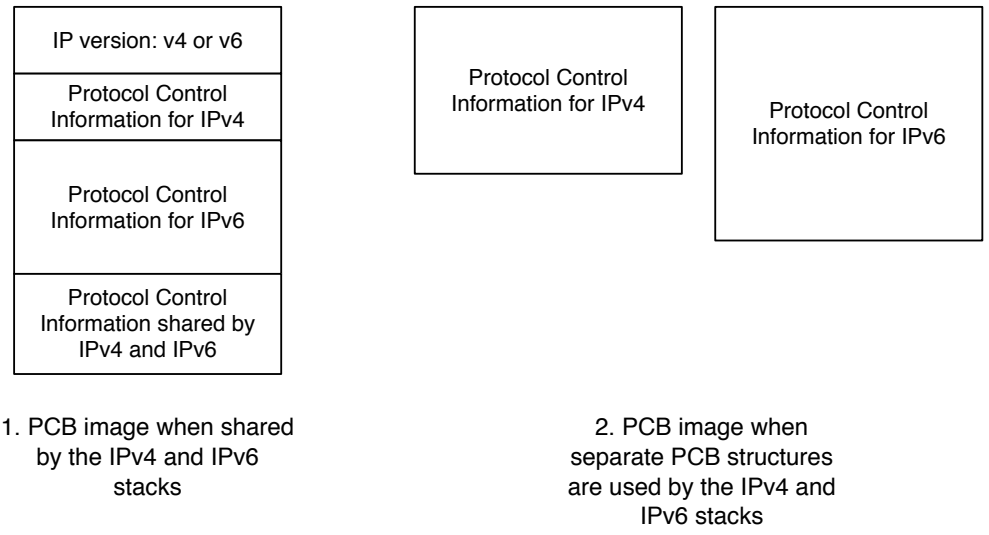


Figure 4.6. The design of the protocol control block

The method 1 has the following benefits.

a) The PCB processing code can be shared.

b) IPv4 PCB and IPv6 PCB can be transformed to the other version's PCB with less overhead compared to having separate PCB structures for each protocols.

Especially the benefit b) is expected to make the socket transformation mechanism between IPv4 and IPv6 easy. Such a transformation mechanism is defined in the Socket API documentation [21]. Figure 4.7 shows the relationship between the socket structure and the PCB structure in 4.4BSD Lite.

As we can see from figure 4.7, the socket structure and the PCB structure is doubly linked. One socket is usually represents one communication path. Considering the design, it looks the method 1 in figure 4.6 looks better, when IPv6 communication using the socket created for IPv4 communication is used, because when changing the IP version, it is possible just to change the type of the PCB. On the other hand, the method 2 has to detach the IPv4 PCB and attach the IPv6 PCB when changing communication from IPv4 to IPv6 using the socket created for IPv4 communication.

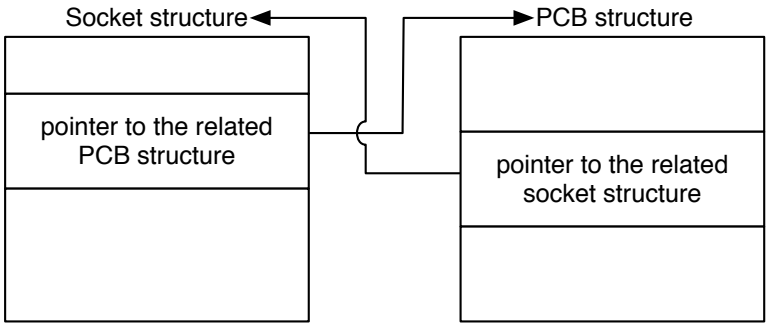


Figure 4.7. The relationship between the socket structure and the PCB structure in 4.4BSD Lite

The method 2 has the following benefits.

c) The code is easy to understand.

d) Debugging the code will be easier than the method 1.

For the researchers who read the code, the method 2 makes it easier to understand the code if the code of the IPv6 PCB is independent from that of IPv4 PCB. This is important since one of the purposes of the dissertation work is to provide the fundamental research/development infrastructure for IPv6 as described in section 2. In addition, there are sometimes unexpected side effects when adding new features to the existing stable code. It is necessary to avoid causing problems in the IPv4 domain, which are out of the research focus.

Based on these analyses, the separate PCB design (method 2) was chosen for the proposed IPv6 implementation.

3.5 Implementation of ICMPv6

The ICMPv6 provides following functions.

- Transmission of error reports
- Multicast group membership management
- Transmission of NDP packets

Each function will be described in the following sections in detail. Note that the NDP protocol, the successor of the ARP protocol, is considered as a different protocol from the ICMPv6. The reason why the NDP protocol is listed as a function of the ICMPv6 is that, in the IPv6 framework, the NDP protocol uses the ICMPv6 protocol as its transport mechanism. In this section, just the overview of the NDP protocol implementation is provided. The detailed discussion of the NDP implementation will be done in section 3.6.

Transmission Error Report

When the following error occurs during the packet transmitting process, the ICMPv6 process code generates an ICMPv6 packet and sends it to the packet originator.

- No route information towards the destination address is found.
- The IPv6 header contains an invalid part.
- The Hop Limit value becomes 0.
- The reassemble procedure of the fragmented packets has failed.

These error reports are almost the same as those used in ICMP for IPv4. In the proposed implementation, the ICMPv6 error reporting mechanism is just a porting work of the related code of ICMP. Therefore, the detailed explanation of the implementation is omitted. One important difference between ICMP and ICMPv6 is that the Path MTU Discovery mechanism [42] is defined as a mandatory function in ICMPv6, where it is not the case in ICMP. Since the implementation of the Path MTU Discovery in the BSD operating system needs attention, the topic is discussed in section 4 later.

Multicast Group Membership Management

In IPv4, IGMP (Internet Group Membership Protocol) [12] manages multicast group membership. In IPv6, the mechanism is handled by the MLD (Multicast Listener Discovery) protocol [10] built on top of ICMPv6. In this implementation, MLD was not implemented because the multicast function was considered as an

advanced function to be implemented at the initial stage of the IPv6 protocol implementation work.

Transmission of NDP Packets

NDP binds the addresses in the data link layer and the addresses in the network layer. NDP is build on top of the ICMPv6 mechanism. From the network layer's point of view, NDP messages can be handled as ICMPv6 packets.

3.6 Implementation of NDP

NDP is a protocol which binds the addresses in the data link layer and the addresses in the network layer. In IPv4, the ARP protocol is used for that purpose. At the beginning of the ARP designing process, the target data link layer was the Ethernet protocol. As the IPv4 became more popular, the protocol was extended to support other data link mechanisms such as HIPPI [61] and FDDI [31]. NDP extends the design of ARP to support more data link mechanisms in a generic processing mechanism. In addition to the data link address resolution mechanism, NDP provides a router discovery mechanism of the same local subnet. Also, the packet redirection mechanism which is a part of the ICMP functions in IPv4 becomes a function of NDP.

The router discovery mechanism and the redirection mechanism were not implemented, since the first priority goal was to establish and implement bi-directional communication between two IPv6 hosts on the same local area network to provide a base communication mechanism.

At the time working on this task, the NDP specification was still under discussion to achieve the final form, which had been published as RFC1970 [50] in 1998.[2] During the implementation process, the following two issues exist.

1. The abstraction model of the address resolution mechanism
2. A specification mistake in the address resolution procedure

These issues is discussed in section 4 in detail.

[2]Later, the NDP specification was revised two times and the latest specification is available as RFC4861 [46].

3.7 Transport Layer

Just providing the IPv6 stack does not encourage people using IPv6. For the real use, it is necessary to provide transport layer protocols, which are directly used by the user-level applications. In this implementation, the TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) protocols, which are widely deployed transport protocols in the IPv4 world, are focused. Although they are called as 'transport' protocols, their protocol designs are tightly integrated with the IPv4 design. Because of these designs, TCP or UDP cannot be used by just implementing the IPv6 stack in the network layer. In the implementation, the entire TCP and UDP protocol code is copied and modified to support the IPv6 protocol. This results in having two same code fragments for the two different network protocols. This was not a good way as the final solution. However, since the goal was to provide a working IPv6 stack, it is not necessary to focus on implementing generic TCP and UDP transport protocols those support both IPv4 and IPv6 seamlessly. The better TCP and UDP code was finally provided by the KAME project later.

3.8 Socket Layer

The socket mechanism is used as a bridge between the application programs and communication protocols implemented in the transport layer and the layers below it. Because the socket system is widely deployed to implement various application programs, the modification to support the IPv6 must be straightforward and must not make a big difference from the original usage of the socket.

Fortunately, in the socket framework, the addresses of communication protocols are managed by the socket address structure (the *sockaddr* structure), which is the abstract structure to indicate the endpoint of a communication path. Because of this abstraction, the socket framework can accept a new communication protocol by adding a new structure which represents the communication endpoint of the new protocol under the abstract structure.

In this implementation, the socket address structure for the IPv6 was newly defined based on the BSD socket API as shown in figure 4.8. The structure is similar to that of the IPv4. The differences are the newly defined field to keep

```
struct sockaddr_in6 {
    u_char  sin6_len;           /* size of struct sockaddr_in6 */
    u_char  sin6_family;        /* protocol number */
    u_short sin6_port;          /* upper layer port number */
    u_long  sin6_flowinfo;      /* IPv6 flow information */
    struct  in6_addr sin6_addr; /* IPv6 address */
};
```

Figure 4.8. IPv6 socket address structure

```
struct sockaddr_in {
    u_char  sin_len;            /* size of struct sockaddr_in */
    u_char  sin_family;         /* protocol number */
    u_short sin_port;           /* upper layer port number */
    struct  in_addr sin_addr;   /* IPv4 address */
};
```

Figure 4.9. IPv4 socket address structure

the IPv6 flow information, and the size of the address field.

The system calls shown in table 4.2 are used with the socket address structure. Since the system calls use the abstraction of the socket address (shown as *struct sockaddr*), it is possible to use both the IPv4 socket address structure (*struct sockaddr_in*) and the IPv6 socket address structure (*struct sockaddr_in6*) in the same manner.

Table 4.2. System calls using the socket address structure

System call name
int bind(*int*,*struct sockaddr **,*int*)
int connect(*int*,*struct sockaddr **,*int*)
int sendto(*int*,*char **,*int*,*int*, *struct sockaddr **,*int*)
int recvfrom(*int*,*char **,*int*,*int*, *struct sockaddr **,*int **)

4. Problems of IPv6

In this section, the problems found through the implementation work are discussed, and the solutions of the problems are proposed. The following 3 points are raised in section 3.

- The Path MTU Discovery (ICMPv6)
- The abstraction model of the address resolution mechanism (NDP)
- A specification mistake in the address resolution procedure (NDP)

These topics are discussed in section 4.1, section 4.2 and section 4.3 respectively.

4.1 Path MTU Discovery (ICMPv6)

Path MTU Discovery is the mechanism to determine the MTU between two nodes, specified in RFC1191 [42] by Stephan E. Deering. This section briefly introduces the mechanism and discusses the implementation design for the 4.4BSD Lite networking stack, which is the target platform for the IPv6 code implemented in the dissertation.

Overview of Path MTU Discovery

Various kinds of communication media are used today, for example, Ethernet, FDDI, pseudo tunnel interface, and so on. Every communication medium has its own limitation of the maximum transmission unit size which is called MTU (Maximum Transmission Unit). In the above examples, the MTU of the Ethernet is 1492 (for Ethernet 802.3), 1500 (for Ethernet v2) or larger for some kinds of fast Ethernet media such as Gigabit Ethernet. 4500 is used in the FDDI case. The tunnel interface can have an arbitrary MTU size by the interface configuration. When sending an Ethernet frame whose size is 2000 bytes over the Ethernet v2 media, we need to divide the frame to two frames whose sizes are 1500 and 500 bytes respectively. If all the media communicating each other are the same between two nodes, we can coordinate the packet size so that the size will not exceed the MTU size by considering the MTU of the local medium. However, since many links which construct the Internet have various MTU sizes, it is usually

not possible to determine the proper packet size by just investigating the MTU size of the local link. Since the packet fragmentation is known to decrease the performance, it is important to choose the minimum MTU when sending data.

Path MTU Discovery is the algorithm to find the minimum MTU size defined as follows.

- At the sending node:
    1. Send a packet with the initial size pre-defined usually (for example, 1500 bytes).
    2. If an error is reported from the intermediate router which indicates the packet is too big for the intermediate link, the sender resends the packet by reducing the size of the packet to the size indicated by the reporting router.
    3. After a fixed period has passed since the sender received the last MTU changing notification, the sender resets the MTU size to the default value.
- At the intermediate router:
    1. If the link MTU is smaller than the packet size which is being forwarded, the router generates an ICMP error message with the information of the proper size of the packet, and sends it to the sender node.

Consideration on Path MTU Discovery Implementation

Since it is waste of the bandwidth resource to discover the Path MTU every time when communication occurs with a certain node, the MTU value should be cached for a short period of time. In the 4.4BSD Lite networking stack, the MTU size discovered by the Path MTU Discovery mechanism is kept with the route information in the routing table. This approach has following merit and demerit.

- Merit: No need to introduce a new table for MTU information. The MTU value can be achieved at routing information lookup time.

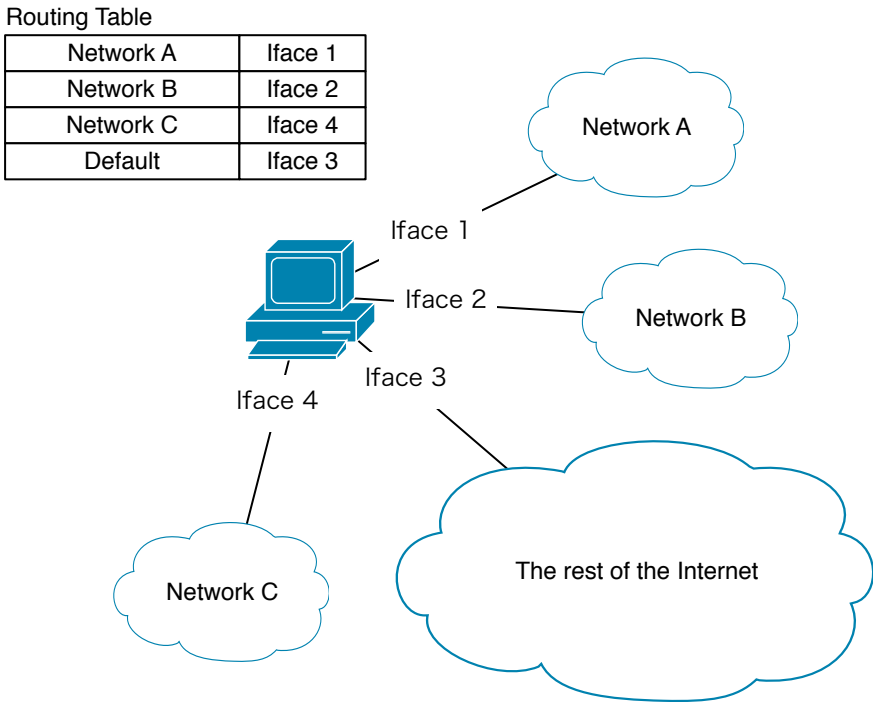


Figure 4.10. The default route

- Demerit: The value may not reflect the proper MTU value when the routing lookup results in the aggregated route entry (for example, the default route).

The default route entry has the last resort information for the routing table lookup as shown in figure 4.10. If a route information entry for a certain destination address does not exist in the routing table, the information in the default route entry is used.

In the 4.4BSD Lite networking stack, only one MTU value can be recorded in the default route entry. In a leaf node, most of the destination addresses match the default route entry when looking up the routing table. Since many different nodes share the default route entry, the recorded MTU size may be a smaller value for some nodes. The correct way is to have an independent entry for each destination address to keep the MTU value for each.

The implementation made in this dissertation did not provide the final solution, however most of the recent BSD networking codes have a similar mech-

anisms. Although they do not have a separate MTU table, the route entry management mechanism is extended to keep different MTU values for different destination addresses even though the route lookup procedure results in the same entry.

4.2 Abstraction Model of the Address Resolution Mechanism (NDP)

In this section, it is discussed that the packet output routine problem in the 4.4BSD Lite networking stack due to the data link address resolution mechanism for IPv6, which was not a problem in the IPv4 stack.

In IPv6, the address resolution of the data link layer address is handled by the ICMPv6 protocol. Thanks to defining the abstraction model of data link address resolution using ICMPv6, IPv6 can manage various kinds of data link media uniformly within the ICMPv6 framework. Since ICMPv6 is one of the upper layer protocols of IPv6, the output routine of the ICMPv6 stack uses the IPv6 output routine. As defined in the NDP specification, the ICMPv6 message for the address resolution is multicasted. In that case, the destination address of the ICMPv6 packet is an IPv6 multicast address. However, the multicast routing has been historically treated as an advanced function, and most of the node does not have multicast routing table. This causes a routing table lookup problem when finding the output interface for the packet whose destination is a multicast address.

Note that this problem never happens in IPv4, since the address resolution mechanism (ARP) does not use IPv4 packets. Instead of it, ARP uses a separate protocol format. Query messages will be directly sent out to the data link layer.

Locating the address resolution mechanism over the layer 3, despite the fact that the mechanism uses the layer 3 protocol, caused this problem. In the implementation, the static route entries are predefined for the multicast destination addresses used by the NDP protocol. In IPv4, such a predefined routes are not necessary, since the ARP protocol does not use IPv4 routing information. This solution is adopted by the successors of the implementation provided from the output of the dissertation (for example, the KAME project). In the KAME code,

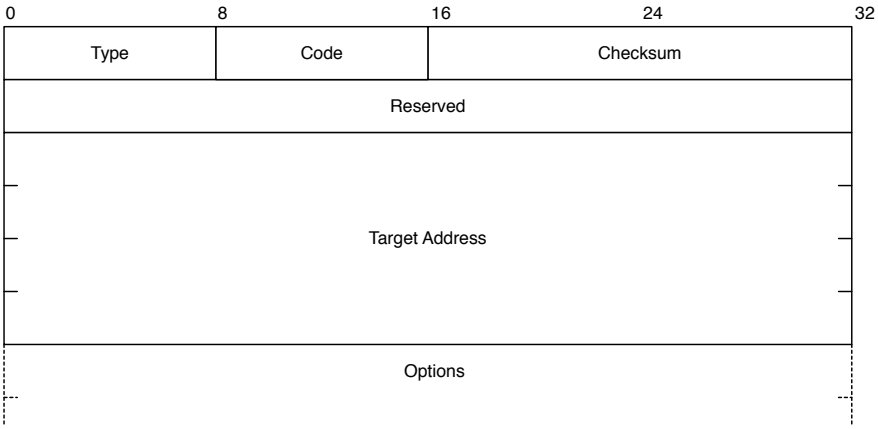


Figure 4.11. Neighbor Solicitation packet format

such a route information entry is injected when an interface which supports IPv6 becomes active. In that sense, it can be concluded that the approach taken in the implementation to solve this problem is feasible.

4.3 Address Resolution (NDP)

In this section, one serious problem in the NDP specification is mentioned that there is a case that the address resolution procedure never completes. Note that the problem has already been fixed based on the problem report, and the latest specification does not have this problem.

In the NDP procedure, an ICMPv6 packet called *Neighbor Solicitation* is multicasted to find the associated data link layer address of the target IPv6 address. The node which receives the Neighbor Solicitation packet replies to the sender node with a unicast ICMPv6 packet called *Neighbor Advertisement* including its data link layer address.

Figure 4.11 shows the format of the Neighbor Solicitation message.

The first 64bit is an ICMPv6 header. The Target Address field contains the IPv6 address whose data link layer address is being solved. In the Options field, additional information can be included. The *Source link-layer address* option is one of such options, which includes the data link layer address of the node which sent a Neighbor Solicitation packet.

In the old specification, this option was defined as an *optional* option. This caused the problem. When a node received a Neighbor Solicitation message without the Source link-layer address option, the receiving node needed to send a Neighbor Solicitation message before sending a Neighbor Advertisement message, since the receiving node did not know the data link layer address which was necessary to send the Neighbor Advertisement message as a unicast message. If the receiving node did not include the Source link-layer address option in the Neighbor Solicitation message either, the address resolution procedure never completed.

Apparently, the usage definition of the Source link-layer address option is the source of the problem. The old NDP specification describe the option as follows.

> "The sender SHOULD include its link-layer address (if it has one) in the solicitation as a Source Link-Layer Address option."

As we can see, the option is treated as an optional requirement. By fixing the option to be a mandatory option, this problem can be solved. As mentioned already, the latest NDP specification has already been fixed based on the problem report.

5. Interoperability Tests

The implementation produced as a part of the result of this dissertation was tested to validate its interoperability at the following three interoperability tests.

- WIDE first interoperability test
  - At Keio University, Kanagawa, Japan
  - 14th and 15th December, 1995
- WIDE second interoperability test
  - At Keio University, Kanagawa, Japan
  - 29th and 30th January, 1996
- The first IPv6 interoperability test

– At the Interoperability Laboratory of New Hampshire University, U.S.A.

– 5th to 10th February, 1996

In this section, the results of these three tests is described.

5.1 WIDE First Interoperability Test

In this subsection, the procedure and result of the WIDE first interoperability test, held from 14th December to 15th December 1995, at Keio University in Japan, is reported. The following 4 organizations joined the interoperability test.

- Nara Institute of Science and Technology (NAIST)
- Keio University
- Hitachi Corporation
- Sony Corporation

Purpose and Test Cases

Since IPv6 is a layer 3 protocol, each node has to discover the data link layer address (layer 2 address) of the target node before sending any packets to the target node. The binding process between the layer 3 address and the layer 2 address is called the address resolution procedure. In IPv6, the NDP protocol is used for this purpose. During the interoperability test, the following two test items were performed to check the behavior of each NDP implementation.

1. Monitor the network traffic and confirm that correct NDP packets were transmitted.
2. Make sure that any two nodes could perform the `ping` program which sent an ICMPv6 Echo Request packet and received an ICMPv6 Echo Reply message in response to the request packet.

Test Procedure

The test procedures of the two test items described in the previous subsection are as follows.

1. Monitor the network traffic and confirm that correct NDP packets were transmitted.

   The data link layer address resolution was performed as described below in IPv6.

   (a) When a node is going to send a packet to a certain destination node, the source node sends a Neighbor Solicitation message if the data link layer address of the target node is unknown.

   (b) A node which receives a Neighbor Solicitation message replies to the source node with a Neighbor Advertisement message including its data link layer address in the Source link-layer option.

   (c) The address resolution process completes when the source node receives the Neighbor Advertisement message.

   To verify the NDP address resolution procedure was performed as described above, all the NDP packets on the network were monitored using the `tcpdump` program.

2. Make sure that any two nodes could perform the `ping` program.
   If the procedure described in 1 went well, the two nodes should be able to ping each other since both nodes had already achieved the data link layer address of their peer. Once the procedure 1 was verified, an ICMPv6 Echo Request message was sent by using the `ping` program from one node, and an ICMPv6 Echo Reply message was monitored to check if the other node replied to the sender node.

Result of the Tests

Figure 4.12 shows the result of the interoperability test.

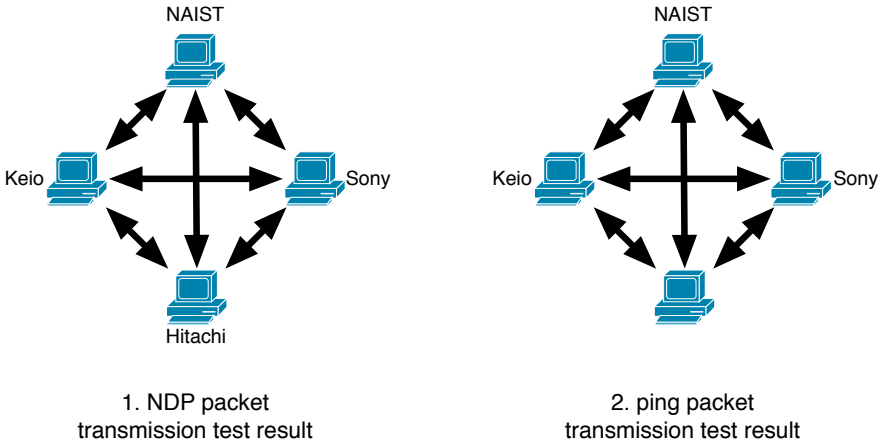


Figure 4.12. Test result of NDP packets transmission and ping conversation

The left figure in figure 4.12 shows the result of NDP packets transmission test. The arrows in the figure indicate the direction of the first NDP packet (the Neighbor Solicitation packet).

Here, an example scenario is used to describe the correct packet transmission procedure between two nodes. In the example, the node from NAIST has the IPv6 address `fe80::1b3d:9293` and the node from Keio has `fe80::20:af75:6d49`. In this condition, when the NAIST node is going to send a packet to the Keio node, the packet transmission as shown in figure 4.13 should happen.

Figure 4.13 is taken from the output of the `tcpdump` program. The first packet is a Neighbor Solicitation packet, requesting the data link layer address of the IPv6 address `fe80::20:af75:6d49`. The second packet is a reply message (Neighbor Advertisement message) in response to the solicitation packet. As we can see in the figure, the packet contains the related data link address to the IPv6 address `fe80::20:af75:6d49`. When these two packets are seen, an arrow from the NAIST node to the Keio node is drawn. All the combination between any two nodes that joined the interoperability test were checked, and it was confirmed that all the nodes could exchange correct NDP packets.

The right side figure shows the result of the `ping` test. The arrows have the same meaning as the NDP packets case. An arrow is drawn from the node that sends an ICMPv6 Echo Request to the receiver node, when the correct exchange

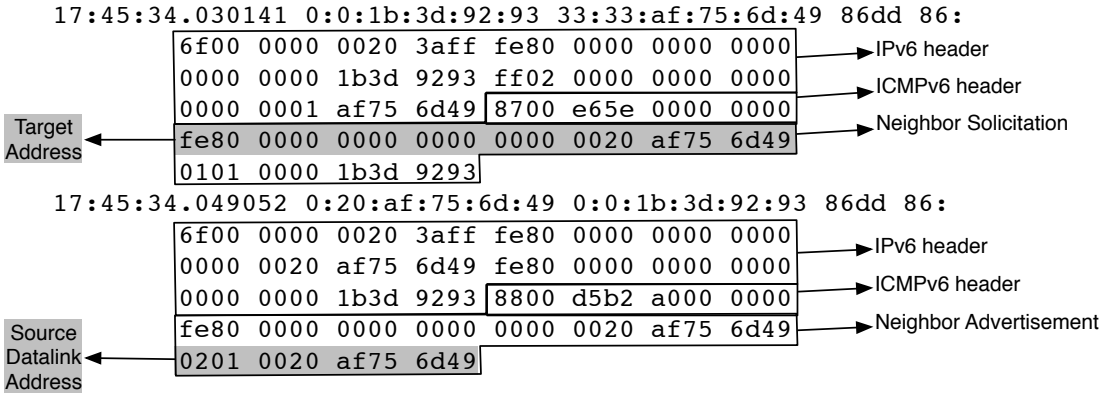


Figure 4.13. NDP packets transmission

procedure of the request and reply messages can be observed. As shown in the figure, it was confirmed that all the implementations could process both request and reply messages correctly.

Consideration

The purpose of the WIDE first interoperability test was to confirm the address resolution mechanism, which was mandatory function when implementing the IPv6 stack. The IPv6 and NDP protocol stacks based on the specification were implemented and confirmed to work as expected. The implementation was checked its interoperability with other implementations which were built completely separately. Every independent implementation could communicate with each other. With the result, it can be concluded that the test proved that the specification was well coordinated and could be usable in the real industrial areas.

In addition, a problem, that the NDP protocol might fail to resolve data link addresses in a certain condition as described in 4.3, was found. The problem was reported to the authors of the specification and has been fixed.

5.2 WIDE Second Interoperability Test

This section describes the WIDE second interoperability test held from 29th to 30th January 1996 at Keio University in Japan. The following organizations

joined the interoperability test.

- Nara Institute of Science and Technology (NAIST).
- Hitachi Corporation.
- Sony Corporation.

Purpose and Test Items

In the WIDE first interoperability test, it was confirmed that the NDP protocol, which is a mandatory function to make an IPv6 implementation, was specified correctly and feasible to implement. In the second interoperability test, two upper layer protocols, TCP and UDP over IPv6 were tested.

The following two items were tested.

1. Verify TCP over IPv6 works by logging into a remote host using the `telnet` program.
2. Extend the Internet super daemon (the `inetd` program) to support IPv6 and verify UDP over IPv6 with the UDP echo protocol, which was implemented inside the `inetd` program.

Test Procedure

To verify the two test items listed in the previous subsection, the following two procedures were performed.

1. Verification of TCP over IPv6 using `telnet`
   BSD/OS 2.0 (based on 4.4BSD), which was the base operating system used, came with the `telnet` program. The program was modified to support IPv6 and the testers tried to log in another IPv6 node on the same local area network.
2. Verification of UDP over IPv6 using `inetd`
   The UDP port 7 is reserved for the echo service, which simply receives a datagram and send the same contents back to the sender. The `inetd`

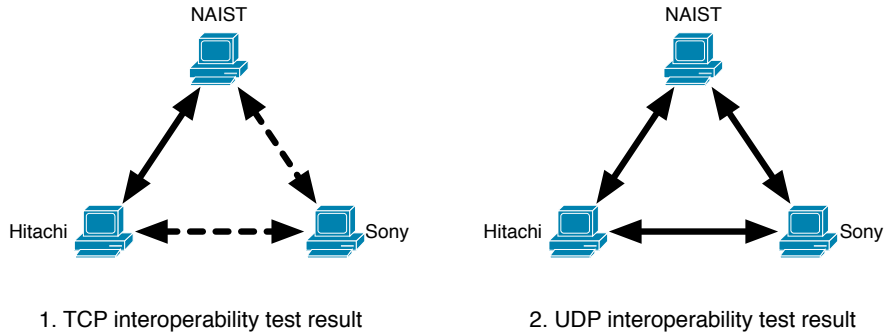


Figure 4.14. The results of TCP and UDP over IPv6 interoperability tests.

program came with BSD/OS 2.0 supported the UDP echo function. The `inetd` program was modified to support IPv6 and the testers sent a UDP echo request from an IPv6 node to another IPv6 node on the same local area network.

Test Results

Figure 4.14 shows the test results of the second interoperability test.

The left side figure is the result of the TCP interoperability test. The arrow means the direction of the telnet connection. The beginning of the arrow is a telnet client node, and the end of the arrow is a telnet server. The NAIST node and the Hitachi node could communicate each other using the `telnet` program. The TCP urgent data transmission could not be verified with the Sony nodes, since at that time the Sony implementation did not support the urgent data transmission mechanism. Except the urgent data, the TCP transmission could be verified with the Sony nodes.

The right side figure is the result of the UDP interoperability test. The arrow means the direction of UDP echo request. In the UDP case, all the implementations could communicate each other using UDP over IPv6.

Consideration

The WIDE second interoperability test verified the function of the IPv6 protocol when used as a network layer protocol which carries transport layer protocols.

Two transport protocols, TCP and UDP, were implemented over IPv6 and tested with other implementations developed separately. All the implementations from NAIST, Keio and Sony could communicate each other with normal TCP traffic. Between the NAIST and Keio implementations, it is also confirmed that the TCP urgent data transmission worked over IPv6. For the UDP case, all the three implementations could communicate without any problems. Thus, by verifying the protocol activity using different three implementations, it was concluded that the IPv6 protocol has no problem when used as a network layer protocol.

5.3 IOL Interoperability Test

This section describes the interoperability test held from 5th to 10th February 1996 at the Interoperability Laboratory (IOL) of New Hampshire University in U.S.A. The following organizations joined the test event.

- WIDE Project
  - Nara Institute of Science and Technology (NAIST)
  - Keio University
  - Hitachi Corporation
- Bay Networks, Inc.
- Digital Equipment Corporation
- FTP Software, Inc.
- INRIA Rocquencourt
- Mentat, Inc.
- Process Software Corporation
- SICS
- Sun Microsystems, Inc.
- Xerox/Palo Alto Research Center

Purpose of the Test

The purpose of this interoperability test was same as those of WIDE interoperability tests (first and second). Every implementer brought its own implementation developed separately based on the specification issued by the IETF, and verified whether each implementation could communicate with other implementations. During the interoperability test event, the NDP protocol activity and the TCP over IPv6 function were investigated.

Test Items

The following test items were prepared by the interoperability test team at IOL.

1. NDP protocol verification
    - Neighbor Solicitation/Advertisement
    - Router Solicitation/Advertisement
    - Redirect
2. Upper layer protocol verification
    - ICMPv6 (using the `ping` program)
    - TCP over IPv6 (using the `telnet` program)

Note that since the NAIST implementation had only implemented stand alone host functions, and had not implemented any interaction mechanisms with IPv6 routers, only host related functions were tested. Based on the implementation status, the following tests were performed on the implementation.

1. NDP state transition verification
2. ICMPv6 Echo Request/Reply (with the `ping` program)
3. TCP over IPv6 (with the `telnet` program)

Figure 4.3 shows the NDP state transition diagram.

Test Results

The test was performed based on the instruction document prepared by the IOL test group. The test scenarios and related results are shown below.

Scenario 1 Send an ICMPv6 Echo Request from the tester node to the target node. The target node sends a Neighbor Solicitation packet to send back an ICMPv6 Echo Reply to the tester node. If the tester node does not reply to the Neighbor Solicitation packet, the target node resends solicitation messages 3 times, and removes the NDP state entry. This series of actions is defined as the number 1, 2 and 3 in table 4.3. After verifying the packet transmission pattern, the target node's NDP entry is checked if the related entry is correctly removed or not.

Result 1 The scenario 1 was verified. From this result, it was concluded that the state transition implementation of the number 1, 2 and 3 in table 4.3 worked.

Scenario 2 While the target node is retrying to send a Neighbor Solicitation packet as in scenario 1, the tester node sends a Neighbor Advertisement packet with the *Solicited* flag set to 0 and the *Override* flag set to 0 or 1. If the target node is implemented correctly, the NDP entry state transits to the *STALE* state based on the number 4 in table 4.3. If the target node replies an ICMPv6 Echo Reply message while the related NDP entry is in the *STALE* state, the state changes to the *DELAY* state based on the number 12 in table 4.3. After the *DELAY* state times out, the state becomes the *PROBE* state. In this state, the target node starts sending Neighbor Solicitation messages to verify the data link layer address (the tester node's data link address in this case) recorded in the NDP entry. If the tester node does not reply the target node with Neighbor Advertisement messages, the NDP entry is removed. In this test, the above packet exchange was monitored and the target node was investigated not to have the related NDP entry at the end of the test.

Result 2 The scenario 2 was verified. From this result, it was concluded

that the state transition implementation of the number 4, 12, 13, 14 and 15 in table 4.3 worked.

Scenario 3 While the target node is retrying to send a Neighbor Solicitation packet as in scenario 1, the tester node sends a Neighbor Advertisement packet with the *Solicited* flag set to 1 and the *Override* flag set to 0 or 1. If the target node is implemented correctly, the NDP entry state transits to the *REACHABLE* state based on the number 5 in table 4.3. When the target node receives an ICMPv6 Echo Request packet from the tester node, it returns an ICMPv6 Echo Reply packet. The packet exchange was monitored to verify the above process was done correctly.

Result 3 The scenario 3 was verified. From this result, it was concluded that the state transition implementation of the number 5 in table 4.3 worked.

Scenario 4 Make the NDP entry state of the target node to the *STALE* state by using the same procedure as described in the scenario 2. After that, send a Neighbor Advertisement message with the *Solicited* flag set to 1 and the *Override* flag set to 0 from the tester node. If the target implementation is correct, the NDP state becomes the *REACHABLE* state based on the number 6 of table 4.3. Then, verify that the state is in the *REACHABLE* state using the latter part of the procedure described in the scenario 3.

Next, make the NDP state of the target node to the *DELAY* state, instead of the *STALE* state by using the same procedure as described in the scenario 2. Then follow the above process. The result should be the same.

Next, make the NDP state of the target node to the *PROBE* state, instead of the *DELAY* state by using the same procedure as described in the scenario 2. Then follow the above process. The result should be the same.

Next, make the NDP state of the target node to the *REACHABLE* state, instead of the *PROBE* state by using the same procedure as described in the scenario 3. Then follow the above process. The result should be the same.

Result 4 The scenario 4 was verified. From this result, it was concluded that the state transition implementation of the number 6 in table 4.3 worked.

Scenario 5 This is similar to the scenario 4. The difference is the *Solicited* flag is set to 1 and the *Override* flag is set to 1. The result should be the same as that of the scenario 4.

Result 5 The scenario 5 was verified. From this result, it was concluded that the state transition implementation of the number 7 in table 4.3 worked.

Scenario 6 Make the NDP entry state of the target node to the *STALE* state by using the same procedure as described in the scenario 2. After that, send a Neighbor Advertisement message with the *Solicited* flag set to 0 and the *Override* flag set to 0 from the tester node. If the target implementation is correct, the NDP state becomes the *STALE* state based on the number 8 of table 4.3. After that, verify the NDP state is in the *STALE* state using the procedure described in the latter of the scenario 2.

Next, make the NDP state of the target node to the *DELAY* state, instead of the *STALE* state by using the same procedure as described in the scenario 2. Then follow the above process. The result should be the same.

Next, make the NDP state of the target node to the *PROBE* state, instead of the *DELAY* state by using the same procedure as described in the scenario 2. Then follow the above process. The result should be the same.

Next, make the NDP state of the target node to the *REACHABLE* state, instead of the *PROBE* state by using the same procedure as described in the scenario 3. Then follow the above process. The result should be the same.

Result 6 The scenario 6 was verified. From this result, it was concluded that the state transition implementation of the number 8 in table 4.3 worked.

Scenario 7 This is similar to the scenario 6. The difference is the *Solicited* flag is set to 0 and the *Override* flag is set to 1. The result should be the same as that of the scenario 6.

Result 7 The scenario 7 was verified. From this result, it was concluded that the state transition implementation of the number 9 in table 4.3 worked.

Scenario 8 Send an ICMPv6 Echo Request to other implementations using the `ping` program.

Result 8 Correct response messages (ICMPv6 Echo Reply messages) were received from the following organizations.

- Bay Networks, Inc.
- Digital Equipment Corporation
- INRIA Rocquencourt
- Mentat, Inc.
- Process Software Corporation
- Sun Microsystems, Inc.

Scenario 9 Try to log in remotely to the other implementation using the `telnet` program.

Result 9 It was succeeded to log in to the IPv6 hosts provided by the following organizations.

- Digital Equipment Corporation
- INRIA Rocquencourt
- Process Software Corporation
- Sun Microsystems, Inc.

The result was different from the ping test result. The reasons were as follows.

- Bay Networks, Inc did not implement the telnet service.
- Mentat, Inc. had a problem in their TCP packet processing.

Consideration

This interoperability test verified the detailed NDP state transition procedures and TCP over IPv6 implementation. Since the NAIST implementation did not provide routing function and the interaction mechanism with IPv6 routers, it could not be verified if the implementation worked as a router and the specification related to IPv6 routing mechanism worked or not. However, from the test results, it could be verified that the specification was fine for the host case and the NAIST implementation was interoperable with various other IPv6 implementations developed separately. The result matched the results of previous two WIDE interoperability test results. It can be concluded that IPv6 is designed interoperable and the specification is feasible to implement from the results.

6. Evaluation

The purpose of this chapter is to verify the specification of IPv6, to find any problems in the specification, and to provide the infrastructure for the future IPv6 research/development activities. The following subsections evaluate these purposes.

6.1 Verification of IPv6 Interoperability

The IPv6 stack evaluated in this chapter was build on top of the BSD/OS 2.0 operating system. The documents referred during the implementation were only the RFC and Internet-Drafts, which are publicly distributed documents by IETF. The referred code while implementing the IPv6 stack was the source code provided with the BSD/OS 2.0. No proprietary information was used, and any information and the actual source code were not exchanged with other organizations working on implementation of IPv6 stacks. The code was purely made by using publicly available information only.

The following organizations joined the WIDE interoperability tests, also had implemented their code with publicly available information only.

- Keio University

- Hitachi Corporation
- Sony Corporation

In the standardization procedure defined at the IETF, three interoperable independent implementations are required when making a certain protocol to be a standard protocol (more precisely, to be a draft standard protocol). Considering the conditions posed in this implementation and verification activity, it can be concluded that three independent IPv6 implementations could interoperate with each other and that the IPv6 standardization process should be ascended to the next level.

6.2 Finding Problems of Specification

The following problems were found.

1. Implementation hurdle in the Path MTU Discovery mechanism for 4.4BSD-Lite system
2. Abstraction model of the address resolution mechanism
3. The possibility in the case of address resolution process failure

In 4.4BSD-Lite, the MTU information for a specific node was kept in a routing entry. Because of this reason, all the MTU information to many different nodes could not be kept if the destination addresses matched the default route. Because the Path MTU Discovery for IPv4 was not implemented in 4.4BSD-Lite, this implementation design was not a problem in 4.4BSD-Lite. Moreover, since IPv4 allows packet fragmentation at the intermediate routers, the Path MTU Discovery mechanism can be an optional feature. However, in IPv6, the intermediate routers cannot fragment packets. Because of this, the Path MTU Discovery becomes a mandatory feature in IPv6, and the design has to be enhanced. The implementation provided in this dissertation did not handle this problem, however, the KAME project IPv6 stack, which was a successor of the implementation has extended the routing entry structure to keep MTU information to each different destination node as suggested in this chapter.

The second problem was a kind of a layering problem, that is, IPv6 uses IPv6 itself to resolve data link layer addresses of the nodes in the local area network. In IPv6, some specific multicast routing information is required before starting IPv6 communication. In IPv4, since the address resolution mechanism is designed as a specific function to each data link media, this kind of problem does not exist. The proposed implementation solved this problem by installing necessary static route information before starting the IPv6 function. The design has been inherited to the successors of the implementation, such as the KAME project's IPv6 stack and BSD operating systems.

The last problem was found through the interoperability test. When implementing the IPv6 stack based on the specification, the *Source link-layer address* option in the NDP specification was specified as optional. However, as described in section 4.3, this specification causes infinite NDP packet exchanges. The option was implemented as a mandatory option to solve the problem. This problem was reported to the authors of the NDP specification and the specification was updated based on our solution.

6.3 Providing an IPv6 Research/Development Infrastructure

The reason why IPv4 was widely accepted is that the source code of the IPv4 stack was distributed as open source software with BSD operating systems. Providing the base IPv6 code as open source software similar to IPv4 should contribute the future IPv6 research and development activities as well. In fact, the source code of the implementation produced in this dissertation was inherited by the KAME project later, and the code provided by the project was widely distributed as an open source software. All the BSD operating systems are now using the code provided by the KAME project, and many research activities and products are produced from them. In that sense, the result of this chapter made the initial step of the IPv6 research/development infrastructure.

7. Summary

The rapid growth of the Internet causes the exhaustion of IPv4 addresses. More and more devices will be connected to the Internet in the future and the Internet must accept those nodes to provide the seamless communication layer among all networking devices. To extend the IP address space and make the Internet affordable for more number of Internet devices, a new protocol whose IP address is represented with a 128bit integer value, while a 32bit integer value is used in IPv4, was proposed as IPv6.

The purpose of this chapter was the following three.

1. Verify the specification of IPv6.
2. Find any problems in the IPv6 specification.
3. Provide the infrastructure for the future IPv6 research/development activities.

To achieve the goals, the IPv6 protocol stack was implemented from scratch based on the specification. The protocol specification was confirmed feasible to be implemented and was evaluated that it was interoperable with other implementations, which were also developed from scratch based on the same specification.

Three problems were found through the implementation. The Path MTU information management problem is caused by the traditional 4.4BSD-Lite MTU management mechanism, which keeps only one MTU information per one routing entry. This mechanism does not work when using a default route entry. As a solution, a proposal to keep independent MTU information per destination nodes was provided. The function was not provided by the implementation shown in this chapter; however, the successor implementation of the code finally provided the same concept. The second problem was a link-local multicast route information problem. The route information is necessary to operate the NDP protocol. For this problem, the solution to install static route information when a node boots up was proposed. The idea was widely accepted and most of the current IPv6 stacks take similar approach proposed in this work. The last problem was a specification mistake, which caused an infinite NDP packet exchange loop in the

address resolution phase. This problem was reported to the authors of the NDP specification and fixed.

The obtained code was adopted by the successor project, the KAME project, which aimed to provide a high quality IPv6 protocol stack for BSD operating systems. The KAME code was finally adopted by all the BSD operating systems. Thanks to the KAME effort, IPv6 on BSD operating systems can be utilized without any additional configuration currently. In that sense, it can be concluded that the proposed implementation and related activities constructed the base of the following IPv6 development and research activities.

Table 4.3. NDP state transition diagram

Rule #	State	Event	Action	New state
1	–	Packet to send	Create entry, Send multicast NS, Start retransmission timer.	INCOMPLETE
2	INCOMPLETE	Retransmission timeout, less than N retransmission	Retransmit NS, Start retransmission timer.	INCOMPLETE
3	INCOMPLETE	Retransmission timeout, N or more retransmission	Discard entry, Send ICMPv6 error.	–
4	INCOMPLETE	NA with Solicited=0 and Override=any	Record link-layer address.	STALE
5	INCOMPLETE	NA with Solicited=1 and Override=any	Record link-layer address.	REACHABLE
6	INCOMPLETE	NA with Solicited=1 and Override=0	–	REACHABLE
7	INCOMPLETE	NA with Solicited=1 and Override=1	Record link-layer address.	REACHABLE
8	INCOMPLETE	NA with Solicited=0 and Override=0	–	STALE
9	INCOMPLETE	NA with Solicited=0 and Override=1	Record link-layer address.	STALE
10	INCOMPLETE	Upper-layer reachability confirmation	– REACHABLE	
11	REACHABLE	Timeout, more than N seconds since reachability confirmation	–	STALE
12	STALE	Packet to send	Start delay timer.	DELAY
13	DELAY	Delay timeout	Send unicast NS, Start retransmission timer.	PROBE
14	PROBE	Retransmission timeout, less than N retransmission	Retransmit NS.	PROBE
15	PROBE	Retransmission timeout, N or more retransmission	Discard entry.	–

Chapter 5

Implementation Design and Evaluation of Mobile IPv6/NEMO Basic Support

This chapter discusses the mobility protocol designed on top of the IPv6 protocol. As discussed in chapter 2, a mobility function is considered as a mandatory function in the future Internet environment. To provide the design reference of the mobility function and the workable implementation as a part of the future mobility research/development environment, Mobile IPv6 and NEMO Basic Support are focused in this dissertation.

Mobile IPv6 and NEMO Basic Support are IETF standard mobility protocols for IPv6. It is said that freely available implementations play a big role in the deployment of new protocols. To accelerate the deployment of IPv6 mobility, two mobility protocol stacks for the BSD operating systems were implemented. The two different implementations are based on different design policies. The first one is an in-kernel implementation and the other is a user space implementation. The former design makes it easy to use kernel information necessary for mobility operation, but it is difficult to implement and extend features than the latter. The latter design needs to have extra mechanisms to retrieve or inject kernel information from/to the user space, but in most cases developing user space programs is easier than developing codes in the kernel. In this chapter, the design policies and implementation details of these two stacks are discussed.

1. Introduction

The rapid growth of the IPv4 Internet has raised a problem of the exhaustion of the IPv4 address space. IPv6 was designed as the essential solution to this problem. We are now in the transition period from the IPv4 Internet to the IPv6 Internet. As a result of the transition, a vast number of IPv6 devices connected to the Internet using various communication technologies will appear in the future. The devices will not only be computers and PDAs but also cars, mobile phones, sensor devices and so on. Since many devices will potentially move around changing their points of attachment to the Internet, mobility support for IPv6 is considered necessary. The IETF has discussed the protocol specification and finally standardized two IPv6 mobility protocols, Mobile IPv6 [30] for host mobility and Network Mobility Basic Support (NEMO BS) [13] for network mobility.

When deploying a new protocol, it is often efficient to provide the protocol stack as an open source software. The developers of the protocol stack can get feedback from users worldwide and thereby enhance their implementation. For example, the IPv6 protocol stack developed by the KAME project [86] accelerated the implementation of IPv6 in various operating systems. I and my colleagues at the WIDE project intended to do the same thing for the mobility protocols. We implemented the two IPv6 mobility stacks: the KAME Mobile IPv6 stack and the SHISA [88, 69] mobility stack. In this chapter, the design and implementation details of these two models are discussed.

2. Overview of Mobile IPv6 and NEMO BS

Mobile IPv6 is a protocol which adds a mobility function to IPv6. In Mobile IPv6, a moving node (*Mobile Node*, *MN*) has a *Home Address* (*HoA*), which is its permanently fixed address. The HoA is assigned to the MN by the network to which the MN is originally attached. This network is called the *Home Network*, and all other networks are referred as *Foreign Networks*. When the MN moves to an other networks, the MN sends a message to bind its HoA and address assigned at the foreign network, *Care-of Address* (*CoA*). The message is called

a *Binding Update* (*BU*) message. This message is sent to a special node, the *Home Agent* (*HA*), which is located in the home network. The HA replies to the MN with a *Binding Acknowledgement*[1] (*BA*) message to confirm the request. A bi-directional tunnel between the HA and the CoA of the MN is established after the binding information has been successfully exchanged. All packets sent to the HoA of the MN are routed to the home network by the standard Internet routing mechanism. The HA intercepts the packets and forwards them to the MN using the tunnel. The MN also sends packets using the tunnel when communicating with other nodes. The communicating nodes, *Correspondent Nodes* (*CN*), do not need to know the actual location of the MN, since they communicate to the MN through its home network attachment. Figure 5.1 illustrates the operation of Mobile IPv6.

In figure 5.1, the communication path between the MN and its peer node is redundant since every traffic is forwarded via the HA. Mobile IPv6 provides an optimized way to communicate with an IPv6 node which is aware of the Mobile IPv6 protocol. An MN can send a BU message to a CN. When sending the BU message, the MN must perform a simple address ownership verification procedure, the *Return Routability* (*RR*) procedure. The MN sends two messages, *Home Test Init* (*HoTI*) and *Care-of Test Init* (*CoTI*) messages, to the CN, one from its HoA and the other from its CoA. The CN replies to these two messages by generating two separate cookies and send them in a *Home Test* (*HoT*) and *Care-of Test* (*CoT*) messages. The MN then generates a secret key using these two cookies received via the different paths and sends a BU message protected cryptographically by the secret key. Once the CN accepts the BU message, the MN can start sending packets to the CN from its CoA. To provide the HoA information to the CN, the MN stores its HoA in a Destination Options Header as the *Home Address option* (*HAO*). The option was newly defined in the Mobile IPv6 specification. The CN can also send a packet directly to the MN using the *Type 2 Routing Header* (*RTHDR2*), a newly defined routing header type. This direct communication is called *Route Optimized* (*RO*) communication.

NEMO BS is an extension of Mobile IPv6. The basic operation of a moving

[1]Throughout this paper, the word 'acknowledgement' is used instead of 'acknowledgment' following the wording convention used in the Mobile IPv6 specification.

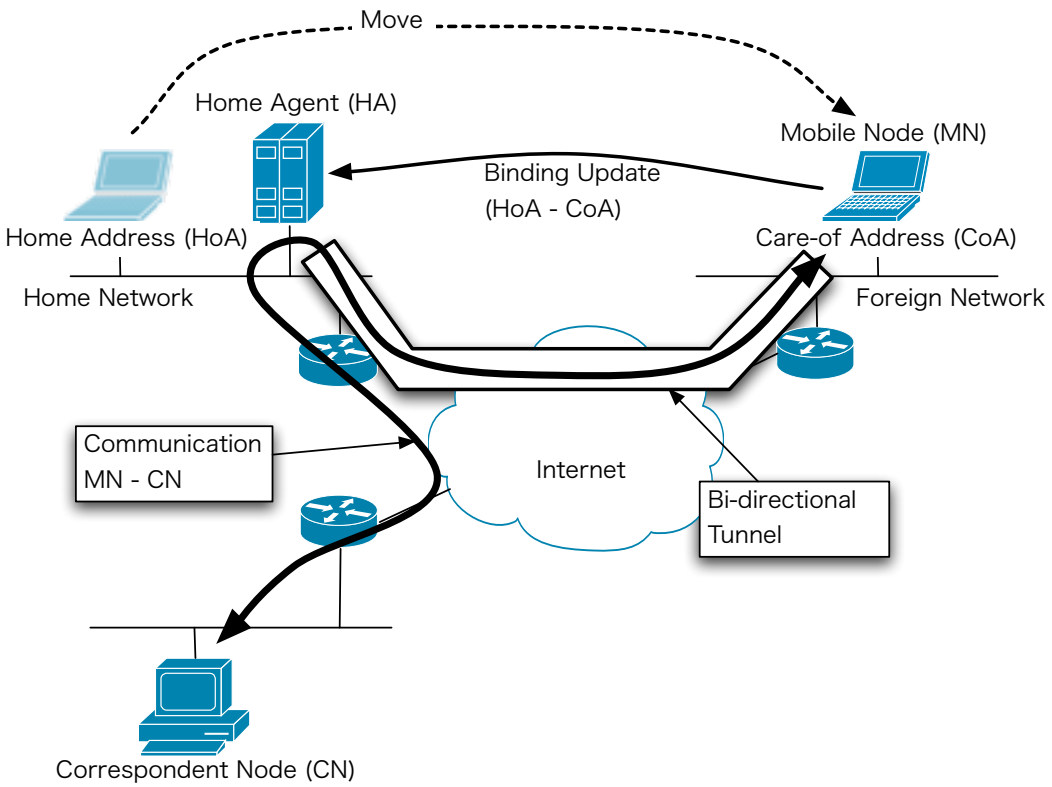


Figure 5.1. Basic Operation of Mobile IPv6

router, *Mobile Router* (*MR*), is same as that of an MN except the MR has a network (*Mobile Network*) behind it. The network prefix is called a *Mobile Network Prefix* (*MNP*). A node in the mobile network, *Mobile Network Node* (*MNN*), can communicate with the rest of the Internet as if it were attached to its home network, thanks to the tunneling between the HA and the MR. NEMO BS does not provide the RO feature. Figure 5.2 depicts the operation of NEMO BS.

3. KAME Mobile IPv6

The development of the KAME Mobile IPv6 stack started around June 2001 as a part of the KAME project activity. At that time, the KAME IPv6 stack [29] already had a Mobile IPv6 protocol stack contributed by Ericsson, however no one in the KAME project was maintaining the contributed code. To keep the code

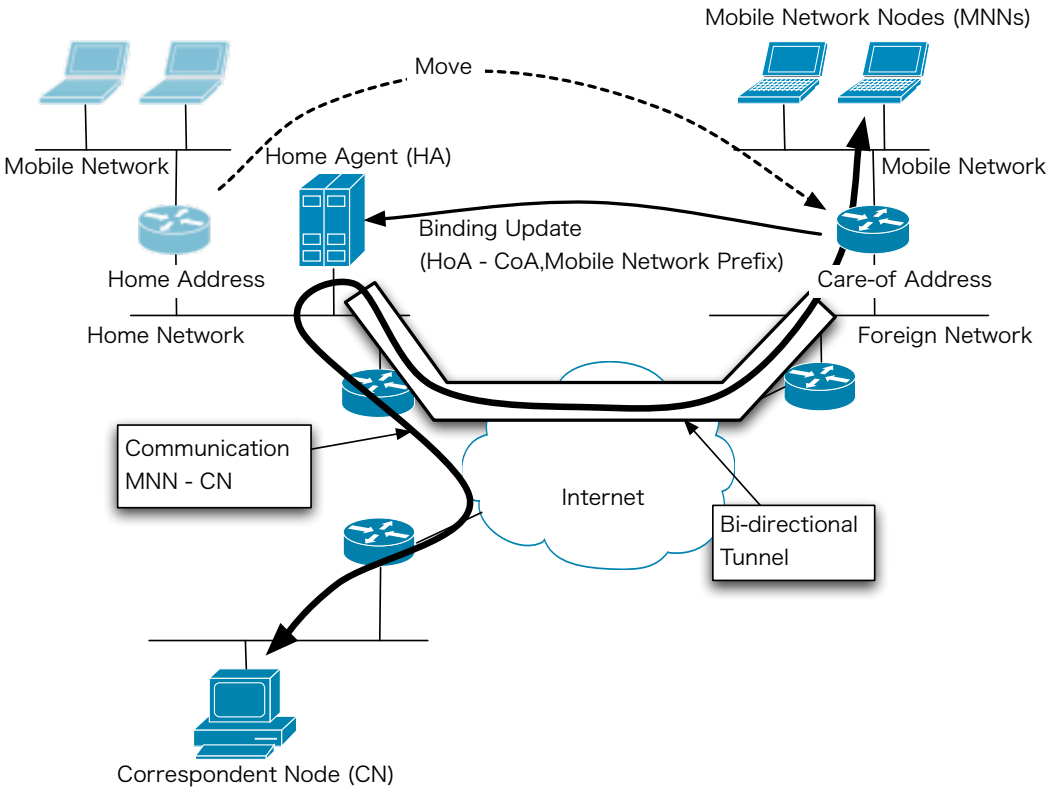


Figure 5.2. Basic Operation of NEMO BS

up-to-date to the specification and enhance the quality of the code, it was started that implementing the new Mobile IPv6 stack on top of the KAME IPv6 stack. This code was developed and maintained until April 2004 when the development of the new mobility code discussed in section 4 was started.

3.1 KAME Mobile IPv6 Design

When designing the KAME Mobile IPv6, we defined the goals of the stack as follows.

- Providing a simple configuration architecture to use Mobile IPv6
- Node type based coding to reduce the object size

Table 5.1. ICMPv6 messages handling

ICMPv6 message type	Sent from	Processed by
Dynamic Home Agent Address Discovery Request	kernel	user space
Dynamic Home Agent Address Discovery Reply	user space	kernel
Mobile Prefix Solicitation	kernel	user space
Mobile Prefix Advertisement	user space	kernel
Router Advertisement	user space	kernel

The KAME Mobile IPv6 is designed as a part of the kernel, whose design is the same as Ericsson's code used before the KAME Mobile IPv6. All types of the Mobility Header, which is a newly defined IPv6 extension header for carrying mobility related signaling messages, are processed in the kernel. The exception is the handling of a few ICMPv6 messages extended in RFC3775. In the KAME IPv6 protocol design, the ICMPv6 messages are handled both in the kernel space and user space (for example, the ICMPv6 Echo Request message is sent from the `ping6` program in user space, and the ICMPv6 Echo Reply message is handled in the kernel). The Raw socket mechanism provides user space programs access to ICMPv6 messages. This mechanism is utilized to reduce kernel code size. Table 5.1 shows the ICMPv6 messages related to Mobile IPv6 which are handled in user space.

The reason why the stack was implemented in the kernel is that the specification of the Mobile IPv6 was using the Destination Options Header for mobility signaling messages when the implementation started. The Destination Options Header is one of the IPv6 extension headers and the basic processing code had already been implemented in the kernel as a part of the KAME IPv6 stack. Implementing the Mobile IPv6 stack in the kernel extending the Destination Options Header processing code was natural choice at that time. The IETF Mobile IP Working Group later changed the message container from the Destination Options Header to the Mobility Header, which can be handled in user space. It was possible to switch the stack from a kernel implementation to a user space implementation, however this change was delayed until the SHISA stack was de-

signed. Since the change would require an entire stack re-designing, possibly causing quality problems, the kernel implementation was continued to keep the code stable.

The main problem with implementing all functions in the kernel is that it increases kernel size. In fact, since most users are not going to use the mobility function, providing it in the kernel is a waste of the code space for such users. This code bloat could be somewhat addressed by the decision to have the KAME Mobile IPv6 code designed as a supplemental function to the existing IPv6 code and the goal not to modify the existing kernel code as much as possible. The code is also divided into several parts based on the type of the node defined in the Mobile IPv6 specification (MN, CN and HA) and users of the stack can easily drop unnecessary code, based on their usage requirements. This design helps to reduce the total size of the code, when code size is considered as an important problem, such as embedded platforms.

3.2 KAME Mobile IPv6 Implementation

Figure 5.3 shows the module relationship of the KAME Mobile IPv6. In this design, the kernel has most of the Mobile IPv6 processing modules. Only the `had` and `rtadvd` are the user space programs which handle ICMPv6 messages shown in table 5.1. The ICMPv6 messages are exchanged between user space and kernel space using ICMPv6 sockets.

The *Neighbor Discovery module* is modified to notify the *Movement Detection* and *Prefix Management modules* of received prefix information. The Movement Detection module detects MN's movement by the current prefix information and its currently assigned address status. The Prefix Management module keeps prefix information for the MN's home network and foreign networks. The *Address Management module* is extended to support HoAs, the permanent addresses used by an MN.

The *Return Routability module* processes signaling messages for the RO communication and puts binding information into the *Binding Management module*. The Binding Management module keeps the binding information between the HoA and CoA of the MN. The module manages a Binding Update List database (in an MN) and/or a Binding Cache database (in an HA/CN). The *Forward-*

ing module and the *Tunneling module* look up these databases when sending or receiving packets, and process every packet based on the binding information.

The *Destination Options Header module* processes the HAO options of the incoming packets. If the HoA included in the incoming HAO is invalid, the packet must be dropped. The *Routing Header module* processes the Type 2 Routing Header introduced by Mobile IPv6.

3.3 Problems of KAME Mobile IPv6

Several problems were found in the KAME Mobile IPv6 stack. The first problem is its extensibility. Since almost all of the codes are implemented as kernel functions, they highly depend on the internal design of the kernel, therefore this makes it hard to extend some specific parts of the module.

The second problem is the number of third party developers. The number of the kernel developers is relatively smaller than that of user space application developers. As a result, the total amount of feedback of the code might be smaller than it should be.

Third, because the code is tightly integrated to the kernel, it is difficult to replace some parts of the mobility function. For example, movement detection may require code to handle specific events in the environment of the operation scenarios of mobile service carriers. However, to do it with KAME Mobile IPv6 implementation, it is necessary to modify the kernel code related to movement detection code.

Lastly, the fact that the KAME Mobile IPv6 needs large modification in the kernel is also a problem when considering integration of the code into the original BSD distributions. For example, we implemented the RR procedure in the kernel. The RR procedure requires several messages to exchange and a timeout/retry management. This introduces the complex state management and timer handling on a per message basis in the kernel, which should be avoided as much as possible. For these reasons, redesigning the entire stack from scratch was decided.

4. SHISA

The development of the SHISA mobility stack started in April 2004 to overcome the problems found in the KAME Mobile IPv6 stack. The latest SHISA stack supports NEMO BS, Multiple Care-of Address Registration [81] and Dual Stack support [64, 73] in addition to Mobile IPv6.

4.1 SHISA Design

SHISA is designed to achieve the following goals.

- Separation of signaling processing layer and forwarding processing layer:
  The operation of Mobile IPv6 and NEMO BS is basically IP packet routing (forwarding or tunneling). To get better performance, packet processing of normal traffic should be done in kernel space, while the signal processing should be done in user space, since the signal processing is complex and it is easier to modify/update user space programs than the kernel. This separation will give the stack both good performance and increased efficiency in stack development.

- Adaptability to various movement scenarios:
  The mechanism required for movement detection and performance varies based on the demands of operators. The signaling mechanism and movement detection mechanism must be independent, because the signaling procedure usually never changes unless the protocol specification changes. Movement detection code must be replaceable in response to operator's requirements.

- Extensibility:
  IPv6 mobility technology is one of most active areas in Internet research. The design of the stack must provide an easy framework to support several future yet undefined functions.

- Minimum modification on existing kernel functions:
  The mobility function will be a core function in future operating systems. To integrate the implementation to the target operating system (in the

implementation discussed in this dissertation, the BSD operating systems), it is necessary to minimize the modification for existing kernel functions.

In the following sections, the implementation decisions are discussed based on the above requirements.

4.2 SHISA Implementation

SHISA is implemented on top of the KAME IPv6 stack [29]. The fact that the KAME IPv6 stack had covered most of the IPv6 functions and API functions made it easier for us to implement the mobility stack on it. What to implement was the mobility-related functions only, and there was no need to implement other IPv6 core functions.

4.3 Supported Features

The SHISA implementation provides the following functions.

- Mobile IPv6 functions for MN, HA and CN
- NEMO BS functions for MR and HA
- Multiple Care-of Address Registration support [81]
- IPv4 Mobile Network Prefix support [64]

4.4 Program Organization

SHISA consists of several user space programs and the modified kernel. Table 5.2 shows the programs used by the SHISA stack and table 5.3 lists the necessary program modules used by each node type.

Based on the node type, one or more SHISA programs can run on a node. In addition, a user can choose to drop or replace functions by stopping or changing the relevant programs. For example, if one does not need the CN functions when operating an MN, he can stop the `cnd` program to save CPU resources or storage

Table 5.2. SHISA programs

Program	Description
`mnd`	Provides the MN functions
`had`	Provides the HA functions
`cnd`	Provides the CN functions
`babymdd`	A simple movement detector of MN
`mrd`	Provides the MR functions
`nemonetd`	Provides the tunnel setup functions for NEMO BS

Table 5.3. SHISA programs categorized by the node types

Node type	Used programs
Mobile node	`mnd`, `babymdd`, `cnd`
Mobile router	`mrd`, `nemonetd`, `babymdd`
Home agent	`had`, `cnd`
Correspondent node	`cnd`

footprint. One can also replace the `babymdd` program to another movement detection program, which may be specialized to the user's network and optimized to the devices used in his network environment.

Figure 5.4 shows the system configuration of the SHISA stack. On top of the stack there are six programs as listed in table 5.2. The `mnd` program and the `mrd` program manage signaling processing of the MN and MR sides respectively. The `mnd` program processes the RR procedure signaling messages for the RO communication. The `mrd` handles the NEMO BS extension in addition to the Mobile IPv6 signaling with the program `nemonetd`. The binding information held on the MN side is stored in the Binding Update database and managed by the `mnd` or `mrd` program.

The `babymdd` program is a simple movement detection program. The movement detection algorithm is discussed in section 4.6.

The `cnd` program manages the signaling processing for the RR procedure on the CN side. The detailed state management of the binding information exchange will be discussed in section 4.7.

The `had` program manages the signaling processing on the HA side. If the HA

serves MRs in addition to MNs, it also processes NEMO BS signaling messages with the `nemonetd` program.

The Neighbor Discovery module and Address Management module provide the same functionality as the KAME Mobile IPv6. The difference is in the method used for event notification. In the KAME Mobile IPv6, these modules send the events directly to the Movement Detection module as shown in figure 5.3. In SHISA, the events are sent to the user space programs via the *Mobility Socket*.

The Mobility Socket is a newly designed socket interface to exchange mobility related information between the kernel and user space programs, and also between user space programs. The Mobility socket is discussed in section 4.5.

The Binding Management module in SHISA manages the binding information database in the kernel. These databases are subset of the master database managed by the `mnd`/`mrd` or `had` programs. The binding databases in the kernel only keeps the minimum information necessary for packet input/output processing. The detailed packet processing is discussed in section 4.8.

The Destination Options Header module and Routing Header module provide the same functionality as those of the KAME Mobile IPv6.

The *Routing Table Management module* is the existing module and used to manage routing entries for tunnel interfaces used by the NEMO BS function.

4.5 Mobility Socket

The operation of Mobile IPv6 and NEMO BS is similar to the existing routing operation. When the design of the SHISA stack was started, the existing BSD Routing Socket mechanism [70] was referred. In the routing mechanism, the role of the kernel is to keep the routing table and forward packets based on that table. The kernel itself does not exchange any routing information. The information exchange is done by routing daemon programs running in user space. This design has several benefits. For example, it is possible to avoid implementing a complex routing algorithm in kernel space, where a trivial mistake sometimes causes a serious problem like a kernel panic. Also debugging user space programs is easier than debugging a kernel, because it is possible to utilize many advanced debugging programs.

A similar approach to the one used by the routing mechanism was taken.

The kernel keeps tables of binding information and transmits packets based on the tables. The update of the tables is performed by the user space programs running on the MN, HA and CN. We designed a new socket, the *Mobility Socket* (*MIPSOCK*) [43], to exchange the mobility-related information between the kernel and user space programs, and between two or more user space programs. It might be possible to reuse the Routing Socket to exchange mobility information, however this approach was not chosen for the following two reasons.

- To minimize the impact to the existing programs which rely on the Routing Socket
- Different information semantics from routing information

There already are several programs which use the Routing Socket. The routing function is one of the core functions for IP nodes to connect to the Internet. It must be avoided causing problems by adding new messages to the Routing Socket. One of the goals is to merge the produced code to the original BSD distributions. Importing a big change to an existing stable function is usually difficult. Designing a separate mechanism for a new feature seems to be more reasonable. Also, not all the information exchanged using MIPSOCK between SHISA programs and the kernel are routing related information. For example, the kernel sends a hint message to user space programs to notify that the node has received a tunneled packet. Another example is a hint message from the kernel to user space programs to insist on sending a protocol error message. In this sense, using the Routing Socket does not seem to be appropriate.

A subset of MIPSOCK messages is listed in table 5.4. The most important point here is that, these messages are not only used for communications between user space programs and the kernel, but also used among user space programs to exchange information. This mechanism provides an easy way to get the current situation of the mobility stack asynchronously and also provides extensibility to other programs related to mobility. For example, `MIPM_MD_INFO` is issued by the `babymdd` program and received by the `mnd` or `mrd` program. Other examples are `MIPM_BUL_ADD` and `MIPM_BC_ADD` messages. These messages are issued by the `mnd` and `had` programs to add binding information. To add the NEMO BS feature to the SHISA stack, we implement `nemonetd` to monitor these messages

Table 5.4. The subset of the MIPSOCK messages

Message	Description
`MIPM_BC_ADD`	Add/Update binding information on HA or CN.
`MIPM_BC_REMOVE`	Remove binding information on HA or CN.
`MIPM_BUL_ADD`	Add/Update binding information on MN or MR.
`MIPM_BUL_DELETE`	Remove binding information on MN or MR.
`MIPM_MD_INFO`	Notify movement with a new CoA to MN or MR.
`MIPM_HOME_HINT`	Notify returning home to MN or MR.
`MIPM_RR_HINT`	Notify there is tunneled traffic to MN.
`MIPM_BE_HINT`	Notify a protocol error occurred in the kernel and insist to send an error message to a peer node.

and establish a bi-directional tunnel. Thanks to MIPSOCK, the modification to the `mnd` or `had` programs to support the NEMO BS feature could be minimized.

4.6 Movement Detection

Movement detection in our Mobile IPv6 stack is based on the result of an unreachability detection of the routers of the network with which the MN is currently connected. When an MN receives a Router Advertisement message, it always sends a Router Solicitation message, which changes all router's reachability status to the *PROBE* state. The *PROBE* state is defined to require a reachability confirmation procedure in the Neighbor Discovery specification [46]. Thus, the MN checks the reachability state of all routers after it receives a Router Advertisement message.

The next step in the movement detection is the validation of the address currently used as a CoA. The KAME IPv6 stack places a strong relationship between the reachability of a router and the prefix advertised by the router. If a router becomes unreachable, the prefixes advertised by the router are marked as *detached* prefixes [28], which mean to be still valid but no longer preferred. As

a result, the address generated from the prefix becomes a non-preferred address. The MN checks its CoA at this point, and changes it to one of the other preferred addresses if the CoA currently used is detached. This algorithm covers most of the general movement scenarios in the real usage.

Movement detection is the most difficult part of a mobility stack implementation. Although the movement detection should be ideally done in the IP layer, the performance of the detection is often slow, if relying only on the IP layer information to detect movement. Several seconds may be needed to detect movement because of many required processes, such as configuring a CoA by receiving a Router Advertisement message, making sure that the address is not duplicated, checking every routers' status to invalidate the current CoA, and processing Mobile IPv6 signal messages. This delay is sometimes critical for real-time applications.

It is known that if using the layer 2 information, it is possible to detect movement much faster [74]. In the KAME Mobile IPv6, the movement detection module was implemented in the kernel and tightly integrated to it. Thus, it was hard to modify the detection algorithm or replace the module with another scenario specific version of the movement detection algorithm. In the SHISA stack, we designed it as a replaceable module. We provided a simple movement detection program (`babymdd`) which performs almost the same process as the KAME Mobile IPv6 movement detection program, however the user of the SHISA stack can implement a special movement detection module based on his local scenario and background layer 2 technologies.

4.7 Binding Management

The *binding update list entry* is managed by an MN or MR to represent the binding between a CoA, HoA, and peer address, to perform the RR procedure and RO communication. The entry was implemented as a finite state machine as shown in figure 5.5 and 5.6. The circles in the figures mean states and the texts written in a bold font mean the events of the state machine. When an event occurs, the corresponding action (written in parenthesis) of the event is performed and the state is changed based on the arrow. Table 5.5 lists all the states of the entry and table 5.6 lists all the events of the state machine.

The state machine is a two-layered state machine, one for the binding informa-

Table 5.5. The list of status values of a binding update list entry

Value	Description
IDLE	The initial state of a primary state machine
RRINIT	Performing the RR procedure. No valid binding exists.
RRREDO	Performing the RR procedure for re-registration. A valid binding exists.
RRDEL	Performing the RR procedure for de-registration
WAITA	Waiting for an Binding Acknowledgement (BA) message. No valid binding exists.
WAITAR	Waiting for an BA message for re-registration. A valid binding exists.
WAITD	Waiting for an BA message for de-registration
BOUND	Valid binding exists.
START	The initial state of a secondary state machine
WAITHC	Waiting for a Home Test and Care-of Test messages
WAITH	Waiting for a Home Test message
WAITC	Waiting for a Care-of Test message

tion registration procedure and the other for the RR procedure. The latter state machine runs when the state of the first state machine is in one of the RRINIT, RRREDO or RRDEL states.

When the entry reaches the BOUND state, the tunnel link between the MN and its HA is established, if the entry's peer node is the HA. While this entry stays in the BOUND state, the MN can send or receive packets addressed to its HoA using the tunnel link as explained in section 2. If the entry's peer node is a CN, then the state means that the MN can perform RO communication with the CN. The detailed packet processing is discussed in section 4.8.

The *binding cache entry* is managed by the HA or CN, which represents the binding between the node and the communicating MN's HoA and CoA. The binding cache entry is implemented as a simple finite state machine as described in figure 5.7. Table 5.7 lists all the states of the entry and table 5.8 lists all the events of the state machine.

When the state machine receives a valid BU message, it immediately changes its state to the BOUND state. The state changes to the WAITB and WAITB2

Table 5.6. The list of events of a binding update list entry

Name	Description
MOVEMENT	Moved to other foreign network.
RETURNING_HOME	Returned to home.
REVERSE_PACKET	Received a bi-directional packet.
RR_DONE	The return routability procedure has been completed.
BA	Received a BA message.
TIMEOUT	A retransmission timer expired.
START_RR	The RR procedure is initiated.
START_HOME_RR	The RR procedure for returning home is initiated.
STOP_RR	The RR procedure is aborted.
HOT	Received a Home Test message.
COT	Received a Care-of Test message.

Table 5.7. The list of status values of a binding cache entry

Value	Description
BOUND	Valid binding exists.
WAITB	Waiting for a BA message for de-registration
WAITB2	Waiting for a BA message for de-registration

state before the lifetime of the binding information expires. If the entry does not receive any BU message in these states, the entry is removed. To extend the lifetime, the state machine of the entry sends a Binding Refresh Request message in the WAITB2 state. The message is one of the Mobile IPv6 signaling messages which requests an MN to send a BU message. While the entry exists the node uses Type 2 Routing Headers when sending packets to the HoA bound in the cache entry. If the node is an HA, then it also installs a proxy Neighbor Discovery entry to intercept all traffic destined to the MN related to the binding cache entry and establishes a tunnel entry to the MN.

Similar to the binding update list entry, the BOUND state means that the HA can use its tunnel link established between the HA and its MN.

Table 5.8. The list of events of a binding cache entry

Name	Description
BU	Received a BU message.
TIMEOUT	A timer expired.

4.8 Packet Input/Output

The performance of the forwarding module impacts the total performance of the stack. While all signaling processing is implemented in user space programs, the normal packet input/output processing is implemented in the kernel to avoid performance degradation. As shown in figure 5.4, the kernel has a subset of the binding information managed by the user space programs. When sending or receiving packets, the stack checks the binding information in addition to the routing table.

There are eight paths with regard to the packet processing in a mobility context.

1. A MN outputs packets using a bi-directional tunnel.
2. A MN inputs packets using a bi-directional tunnel.
3. A MN outputs packets using the RO communication.
4. A MN inputs packets using the RO communication.
5. A HA intercepts packets for an MN and forwards them.
6. A HA receives tunneled packets from an MN.
7. A CN outputs packets using the RO communication.
8. A CN inputs packets using the RO communication.

Figure 5.8 shows the call flow of the case 1. A packet generated in an MN reaches to the `nd6_output()`. The `nd6_output()` checks if the source address of the packet is the HoA of the MN and the status of the binding information. If a valid binding exists, the packet is passed to the `mip6_output()` function to tunnel it to the HA of the MN.

The left side shows the packet flow of a mobile network when the MN is acting as an MR. A packet generated in the mobile network is received by the MR and reaches the `nd6_output()` via the `ip6_input()`, `ip6_forward()` and `ip6_output()`. In the MR case, the packet is passed to the `nemo_output()` function which is the input function of NEMO BS tunneling created by the `nemonetd` program. The `nemo_output()` function sends the packet using the tunnel link established between the MR and the HA.

Figure 5.9 shows the case 2. An MN receives a tunnel packet from either the `nemo_input()` function or `mip6_tunnel_input()` function. The former is used for NEMO BS and the latter is used for the Mobile IPv6 processing. If the packet is addressed to the mobile network, it is passed to `ip6_forwarding()`, otherwise it is passed to the upper layer input routine of the node.

For historical reasons, The SHISA stack has two tunnel mechanisms. When we implemented the SHISA stack, it only supported Mobile IPv6 and used the `mip6_output()` and `mip6_tunnel_input()` functions. These functions were not designed to process forwarding cases. Because of this, we added another tunneling function which supports forwarding cases for NEMO BS. These designs should be reviewed.

Figure 5.10 shows cases 3 and 7. When a packet is processed in the `ip6_output()` function, the function calls two other functions: `mip6_create_rthdr2()` and `mip6_create_hoa_opt()`. The `mip6_create_rthdr2()` function looks up the binding cache database and checks if there is a valid entry for the destination node. If an entry exists it creates a Type 2 Routing Header and sends the packet directly to the destination MN (case 7). The `mip6_create_hoa_opt()` function looks up the binding update database and creates a Destination Options header with a HAO option. The packet is not tunneled to the HA rather it is sent directly to the peer node (case 3).

Figure 5.11 shows cases 4 and 8. The route-optimized packet has a Destination Option Header (the HAO option) or Type 2 Routing header. The IPv6 stack has already defined the processing routines for the Destination Options Header and Routing Header. In SHISA, these header processing routines are extended to support Mobile IPv6-related data, which is the HAO option and the Type 2 Routing Header. These headers are processed in the `dest6_input()` and

`route6_input()` functions. The two functions; the `dest6_swap_hao()` and the `dest6_mip6_hao()`, swap the HoA stored in the HAO option and the source address field of the input IPv6 header. By swapping the source address (CoA) and HoA, upper layer protocols do not need to handle the temporal location of the MN. The code has two different positions to swap these addresses for security, as discussed in section 4.9.

Figure 5.12 shows the case 5. A HA intercepts packets destined to the HoAs of MNs for which it is serving. These packets are input to the `ip6_input()` function by the proxy Neighbor Discovery mechanism. The packets are passed to the `ip6_forward()` function because the destination address is not one of the HA's addresses. If the packet is addressed to the HoA of one of the MNs, the HA calls the `mip6_encapsulate()` function to send the packet to the tunnel link between the HA and the MN which is the owner of the HoA. If the packet is addressed to the mobile network, the `nemo_output()` function is used instead.

Figure 5.13 shows the case 6. The packet tunneled from an MN is passed to either `nemo_input()` or `mip6_tunnel_input()` function. The packet is passed to the `ip6_input()` function again and forwarded to the destination node of the packet. For the same reason as discussed in Figure 5.9, two tunnel input functions are provided; one for Mobile IPv6 and the other for NEMO BS.

4.9 Home Address Verification

When an MN sends a route-optimized packet to other nodes, it inserts an HAO option into the packet. The node which receives such a packet swaps the HoA contained in the HAO and the source address of the packet. The upper layer stacks and applications do not notice the actual location of the MN thanks to this swap procedure. However, if the procedure is performed without any verification of the HoA, any malicious node can steal arbitrarily HoAs of other legitimate nodes.

The verification is based on the existence of binding information related to the HoA. If the binding entry exists, the HoA is accepted. The address stored in the HAO option and the address in the IPv6 source address field can be swapped immediately (see the `dest6_swap_hao()` function in figure 5.11).

The problem arises during the processing algorithm of the initial message (a

BU message) which contains an HAO option. Although the message contains an HAO option, there is no valid binding information before accepting the first BU message. Thus, the algorithm described above cannot be used.

For a BU message, the validity check of the HAO option is done by a cryptographic method. The Mobile IPv6 specification defines that an MN and an HA must protect the BU/BA messages by the IPsec mechanism. This means, if the IPsec header protects the packet, the packet with a HAO option can be accepted. However, the solution is not so easy. The IPsec processing is performed on the logical format of the packet. That is, the IPsec header assumes that the source address is a HoA. It is necessary to swap the HoA in the HAO and IPv6 source address before performing IPsec verification. In the SHISA implementation, the swap operation is delayed until the IPsec header appears in the input packet. In figure 5.11, the `dest6_mip6_hao()` function is called from the `ip6_output()` function. The function is called every time the `ip6_output()` function processes the header chain of the input packet. The `dest6_mip6_hao()` function swaps addresses if the next header to be processed is either the Authentication Header or the Encapsulation Security Payload. If the receiving node has a valid IPsec security association, the BU/BA message passes the IPsec verification; otherwise it is dropped.

For a BU message sent to a CN, the same mechanism cannot be used, because a BU message sent to a CN is not usually protected by the IPsec mechanism. The specification defines that a BU message to a CN must be cryptographically protected by the RR procedure. Thus, the stack accepts a BU/BA message even if it does not have IPsec headers. The `dest6_mip6_hao()` function checks the next header to be processed, and if the next header is a Mobility Header and the message type is BU, the function swaps the HoA in the HAO and the source address of the IPv6 header. The BU message is validated in the processing routine of a BU message with the RR procedure. If the BU message has been sent as a result of the proper RR procedure, the message passes the cryptographic verification; otherwise it is dropped.

4.10 Multiple Care-of Address

In the Mobile IPv6/NEMO BS specifications, it is impossible for MN/MR to register multiple CoAs at the same time. However, considering the recent progress of wireless technologies, it is getting more common for mobile devices to have multiple network interfaces. Using multiple interfaces enables efficient usage of network property/bandwidth and increases fault tolerance in case of network problems [16]. The Multiple Care-of Address support protocol [81] was implemented as one of the solution for the problems. Currently, this function is available only for MR. The extension is implemented by adding a unique identifier field to the `MIPM_BUL_ADD`, `MIPM_BUL_ADD` and `MIPM_MD_INFO` messages and adding a field to keep the identifier in the binding databases. The identifier is bound to each CoA assigned to the MR. Usually the HA routes packets based on the HoA of each MR. In this case, the HA cannot distinguish the binding between (HoA, CoA1) and (HoA, CoA2). In this extension, the pair of its HoA and an identifier specifies each MR. In the SHISA implementation, the identifier is mapped to a tunnel interface. If MR has two network interfaces for CoAs and registers these two CoAs at the same time, the MR and its HA have two tunnel interfaces between them, each tunnel is bound to each identifier assigned to the network interfaces of the MR. The MR and the HA can utilize these tunnels based on local policy. For example, they can use one tunnel as a primary interface and the other for backup. Or, if they have two interfaces whose properties are different, for example, one interface is low-bandwidth and low-latency, and the other is high-bandwidth and high-latency, then they may use the former for urgent messages and the latter for data transmission. Since this mechanism has been implemented as a tunnel interface, it is possible to use the basic packet filtering mechanism, such as IP Filter [60], to distribute traffic.

4.11 IPv4 Mobile Network Prefix

When NEMO BS was specified, most people involved in the discussion did not think they would need IPv4 NEMO support. However, considering the current rate of IPv6 deployment, it will require more time than originally expected, therefore some kind of IPv4 support is necessary. I proposed a mechanism to carry

IPv4 traffic over a tunnel interface created by NEMO BS mechanism [64]. With this mechanism, MR can have an IPv4 MNP in addition to an IPv6 MNP. The benefit of this mechanism is that users of an MR, which supports this extension, can operate IPv4 networks over an IPv6 only infrastructure. This kind of operation encourages the existing IPv4 users to move IPv6 infrastructure [65], since NEMO BS can provide fault tolerance and load-balance or traffic engineering using the Multiple CoA support as discussed in section 4.10. SHISA is extended to keep IPv4 MNPs in its binding database and extended to forward IPv4 packets, whose prefix is registered as a part of MNP, using the tunnel interface established between the MR and its HA.

4.12 SHISA Problems

The main problem for SHISA is its tunneling mechanism. As discussed in section 4.8, SHISA has two different tunnel mechanisms, one for Mobile IPv6 traffic, and the other for NEMO BS traffic. The former is implemented as an in-kernel tunneling mechanism and the latter is implemented as a pseudo network interface. As described in section 4.10, the Multiple Care-of Address Registration function is only provided for MRs. The reason why it was not possible to provide the same function to Mobile IPv6 comes from the difference in the tunnel design. To distribute traffic to multiple tunnel interfaces, the IP Filter mechanism was used. The mechanism works with the pseudo interfaces but does not work with the in-kernel tunneling mechanism. This problem has to be fixed.

4.13 SHISA Applicability

Various experiments and demonstrations were performed using the SHISA stack which proved the applicability of the stack. In 2005, the stack was used to provide transparent network service in an actual conference network infrastructure [67]. In 2006, a similar operation at a conference using the Multiple CoA Registration mechanism was performed to provide a smoother handover experience [68], both experiments are discussed in chapter 7. Some demonstration activities were also made to advertise the IPv6 mobility technology, such as at the First IPv6 Summit in Thailand [66], and the CEATEC 2006 exhibition which is the largest consumer

electronics exhibition in Japan. The SHISA stack is also used as the base mobility service system by the Home Agent Web User Interface activity [19, 3] in the Nautilus6 project [47].

5. Discussion

The implementation of the Mobile IPv6 stack was started as a part of the kernel function initially. In the beginning, it seemed to be a better and faster way to create a working stack, because implementing all functions in the kernel is similar to adding new functions to a single program. Even though the size of the kernel code is huge, it is a single program. Since the author already had a good foundation in kernel programming through the KAME project activity, implementing the stack in the kernel shortened the development time. The first version of the code was released in October 2001, about four months after beginning of the development. After the specification became stable around June 2003, when the final draft of the Mobile IPv6 specification was published, consideration of the addition of new features to the platform was started.

At this point, the decision was made to redesign the entire stack to solve problems which the kernel implementation had as described in section 3.3. The new code, SHISA, was implemented mostly as the user space programs with minimum kernel support. In fact, the initial Mobile IPv6 specification was hard to implement in user space because of the following two reasons. One is that the signal message is designed as a Destination Option of the IPv6 extension header. The specification requires protecting the signal messages by IPsec, however, protecting a part of the IP packet is not possible by the IPsec mechanism. The second reason is that there was no interface to send or receive mobility signal message from user space. Inserting a Destination option with the arbitrary timing is not possible with the current IPv6 architecture. These two problems were solved when the specification was completed, and there arose a possibility to re-design the mobility platform.

The first release of SHISA was December 2004, eight months after beginning. It took twice the time to achieve the initial release of the stack, however SHISA provided not only Mobile IPv6 but also NEMO BS from the beginning. Thanks

Table 5.9. The code size of each mobility stack (in line numbers).

	KAME Mobile IPv6	SHISA
kernel (core)	15750	5470
kernel (Mobiliy Socket)	n/a	811
user space	3000	17078

to the re-designing, NEMO BS could be implemented easily on top of the SHISA stack. Also, as discussed in section 4.10 and 4.11, some new features were also added to SHISA later without big architectural changes.

The code size of each implementation is shown in table 5.9. The modified code size of the SHISA kernel is reduced to 40% of that of the KAME Mobile IPv6 kernel. The smaller the modification size is, the easier to merge the code to the original BSD distribution is. Opposite to the kernel size, the user space program size of SHISA became about six times larger than that of the KAME stack. This is understandable given that the size of the KAME Mobile IPv6 kernel and the size of the user space program of SHISA is almost the same. Since most of the functions of mobility processing were moved from the kernel to user space, it is not surprising that the size of the user space programs is almost equal to the old kernel code. Even if most of the functions has moved to user space, the SHISA kernel still has about 6000 lines for mobility processing. This is because the subset of binding database management code is necessary to leave in the kernel and the interface layer (the Mobility Socket mechanism described in section 4.5) between the user space and the kernel is needed to exchange mobility information. The code size of the interface layer is 811 lines occupying about 12% of the kernel mobility code.

The total code size of the SHISA stack will be larger than that of the KAME Mobile IPv6. However, despite of this drawback, keeping the mobility protocol implementation in user space is better, because of its ease of maintenance and extensibility when importing more advanced features defined in the future. Through this design process, the socket API specification for sending/receiving mobility signal messages were being discussed at IETF [6]. Without the API, the current platform design cannot exist. The platform presented in this dissertation played a role as the first working stack of the API, which contributed the

standardization process of the API.

When moving the signal processing part to user space, the entire function was divided into small pieces. Six user space programs were provided in the current implementation as listed in table 5.2, based on the processed signal types. Each program can be replaced with another equivalent program, as long as it does not change the information exchange rules among other running programs. As discussed in section 4.6, this is useful when designing a movement detection mechanism. If the network supports a special notification mechanism at the layer 2 level, then the movement detection speed is enhanced by utilizing the notification, which will finally shorten the service disruption time when a node is moving from one network to another network.

Because the signal processing code was moved to user space, some performance degradation in signal processing was foreseen. In the KAME Mobile IPv6, the signaling packet is handled in the kernel. In the kernel implementation case, creation and update of binding information are done while the packet is being processed. In contrast, the SHISA escalates signaling packets to user space and processes them there. The binding information stored in the kernel is created or updated from the user space using the Mobility Socket mechanism. Eventually, the context switching between the kernel and user space has to be performed twice to handle one signaling packet.

To evaluate the effect, the processing time of a Binding Acknowledgement packet was measured with both the KAME Mobile IPv6 and SHISA stack. In the KAME Mobile IPv6, the processing of the Binding Acknowledgement packet is completed in one function, `mip6_ip6ma_input()`. Thus, the time difference before and after the function call was measured. In the SHISA stack, the Binding Acknowledgement message is received by the general signal processing function `mip6_input()` and passed to the IPv6 raw socket mechanism to deliver the message to the user space. Once the packet is processed, the SHISA user space program updates the related binding information using the Mobility Socket mechanism. Thus, the time, from when the Binding Acknowledgement message is received by the `mip6_input()` function, was recorded, and it was compared to the time when the binding information was updated. Figure 5.14 shows the measurement points and table 5.10 shows the results.

Table 5.10. Comparison of processing time (in microseconds) of signaling packets between the KAME Mobile IPv6 and the SHISA stack

	KAME Mobile IPv6	SHISA
Average of 20 messages	25.85	62.15
Standard Deviation	5.55	19.01

The processing time of a Binding Acknowledgement message of the SHISA stack is about 2.4 times slower than that of the KAME Mobile IPv6. Examining the different values for the standard deviation it is understood that the KAME Mobile IPv6 processing times are more predictable. These results show that the SHISA stack has performance degradation. However, the conclusion here is that the degradation caused by the processing of signal information can be ignored. The Binding Acknowledgement message exchange occurs at most every 4 seconds, usually less. Suppose if sending VoIP traffic, the voice packet size is IPv6 header (40 bytes) + UDP header (8 bytes) + RTP header (12 bytes) + payload (10 bytes in G.729 codec) = 70 bytes. Since the voice packets are sent every 20ms, the total size in 4 seconds will be 14000 bytes. The packet size of a Binding Acknowledgement message is 52 bytes. Even if we send Binding Acknowledgement messages at the most frequent rate possible (every 4 seconds), the occupation ratio will be 0.37%. In a realistic situation, the message will be exchanged less frequently (in the SHISA implementation, the default interval is 20 seconds). In addition, as the data traffic size becomes larger (for example, file transfer or video communication), the less dominant the occupation ratio becomes. Therefore, the increased processing time of the signaling messages in the SHISA stack can be negligible.

Moving the signal processing function to the user space arise other problem, that is the loss of signal due to the heavy load on the node. Of course, the same problem occurs even though the signal processing is implemented in the kernel. However, in the kernel implementation case, the signal packet processing is invoked as a part of the packet input interrupt processing, which has usually higher priority than the user space programs. In the user space implementation, the signal packets are once queued in the socket buffer. If the socket processing

is delayed due to other high priority tasks, then the signal loss may occur. This effect is not investigated deeply yet. The more work on the packet processing miss in the worst environment should be done.

6. Summary

The IPv6 mobility stacks based on the IETF standard mobility specifications was implemented and released to accelerate the deployment of the IPv6 mobility technology. The first version of the stack (KAME Mobile IPv6) was implemented in the kernel, but the kernel stack design and its implementation had problems in extensibility and ease of development. To solve the problems, the implementation is totally revised based on the mobility platform design proposed in chapter 3. In the new implementation, the signal processing code is moved to user space was proposed. To keep the performance of the normal packet processing, the packet input/output processing code was kept in the kernel. The new stack (SHISA) can be easily extended to support new features. Thanks to this design, the NEMO BS extension, Multiple Care-of Addresses Registration extension, and IPv4 mobile network support can be implemented. These extensions are discussed in the following chapters. Also the SHISA stack attracts third party development activities [79, 78] more readily thanks to the user space implementation which is easier to develop than the kernel development.

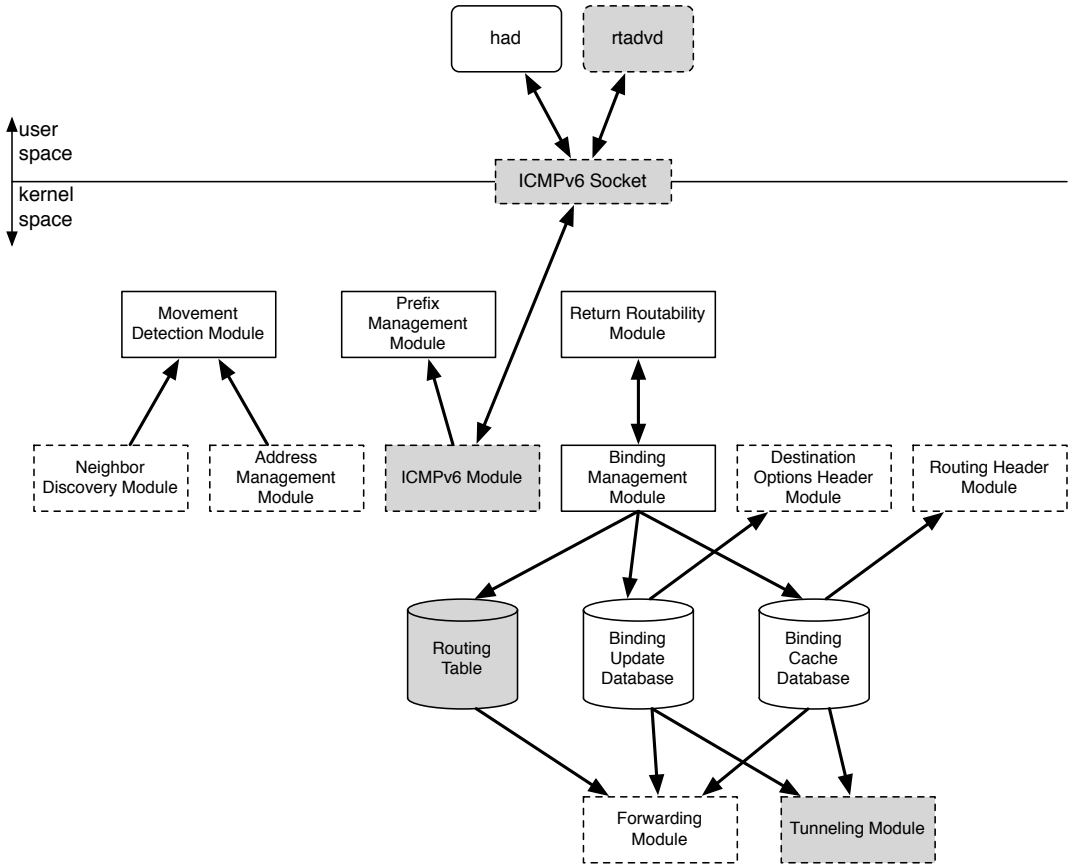


Figure 5.3. The module layout of the KAME Mobile IPv6. The dotted boxes are modules which exist in the base KAME IPv6 protocol stack. The shaded boxes are modules which did not require any modification to support Mobile IPv6. The boxes with solid lines are newly introduced modules for Mobile IPv6.

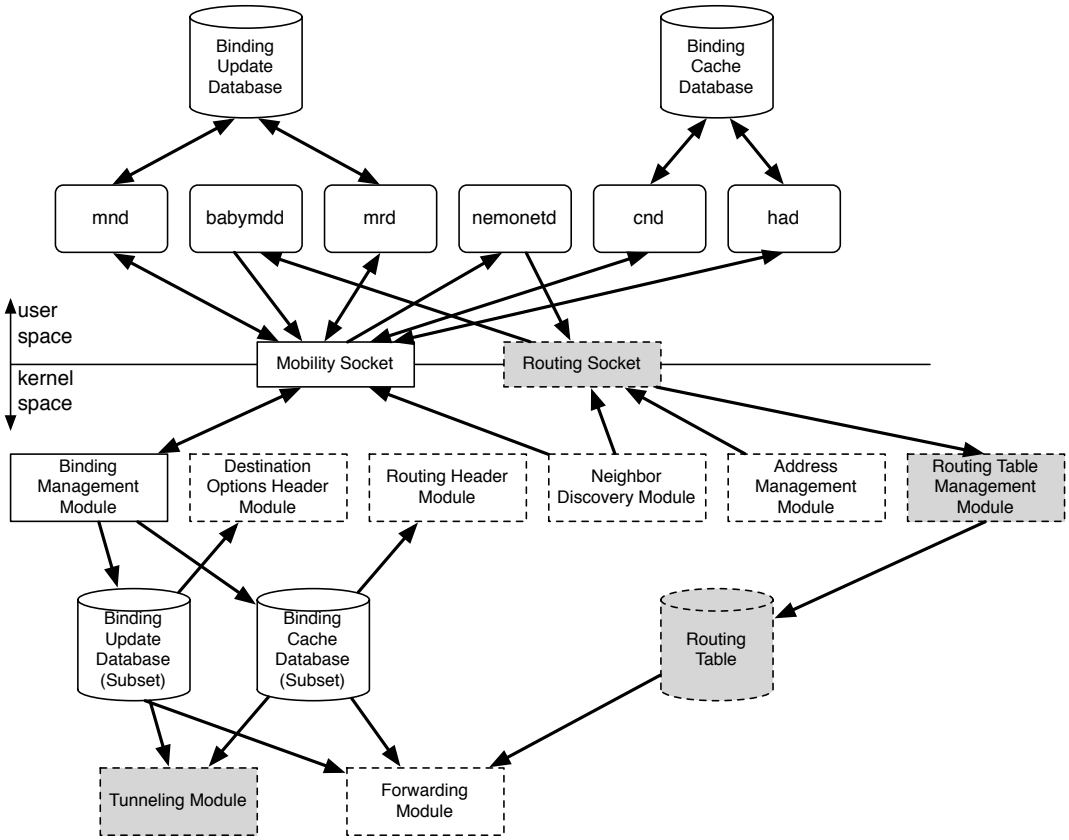


Figure 5.4. The SHISA system configuration. Same as figure 5.3, the dotted boxes are modules which exist in the base KAME IPv6 protocol stack. The shaded boxes are modules which did not require any modification to support Mobile IPv6. The boxes with solid lines are newly introduced modules for Mobile IPv6.

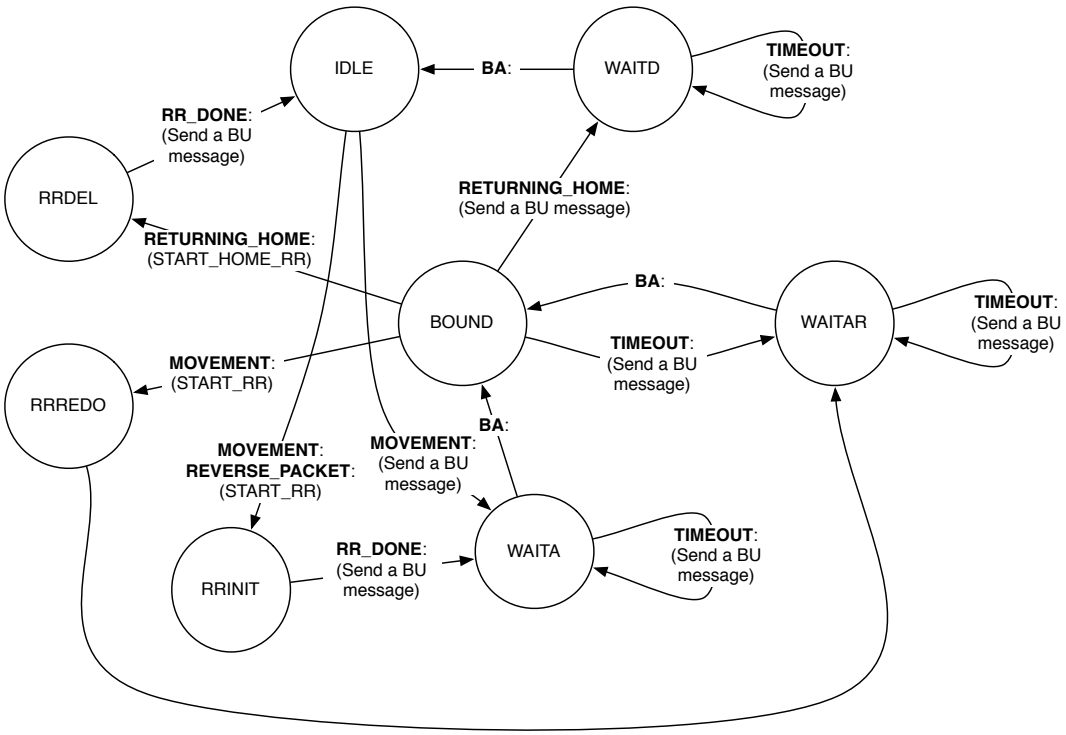


Figure 5.5. The primary state machine of a binding update list entry

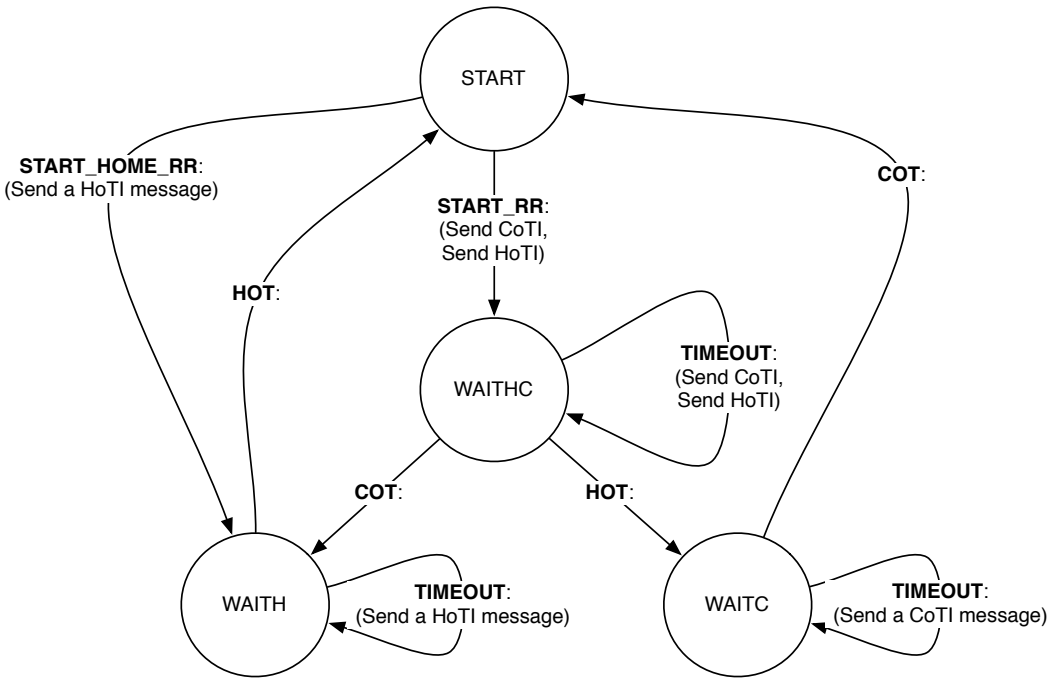


Figure 5.6. The secondary state machine of a binding update list entry

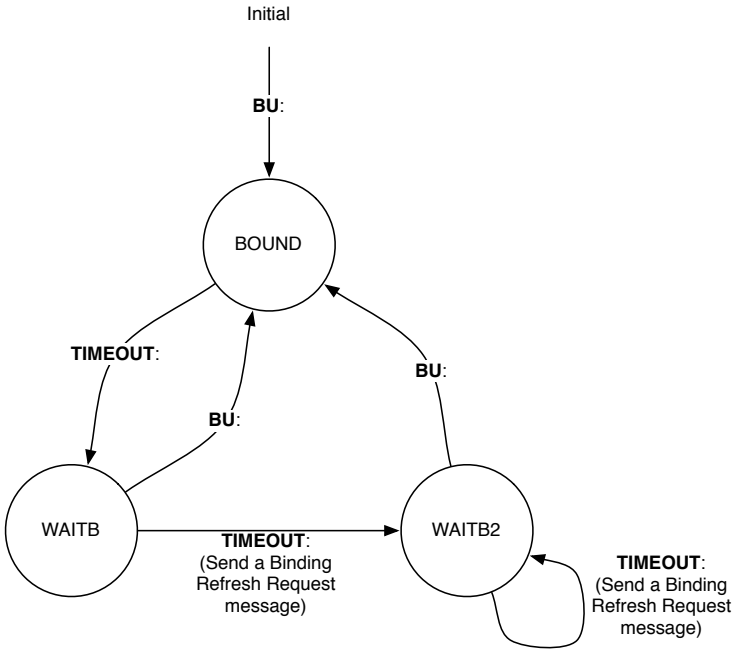


Figure 5.7. The state machine of a binding cache entry.

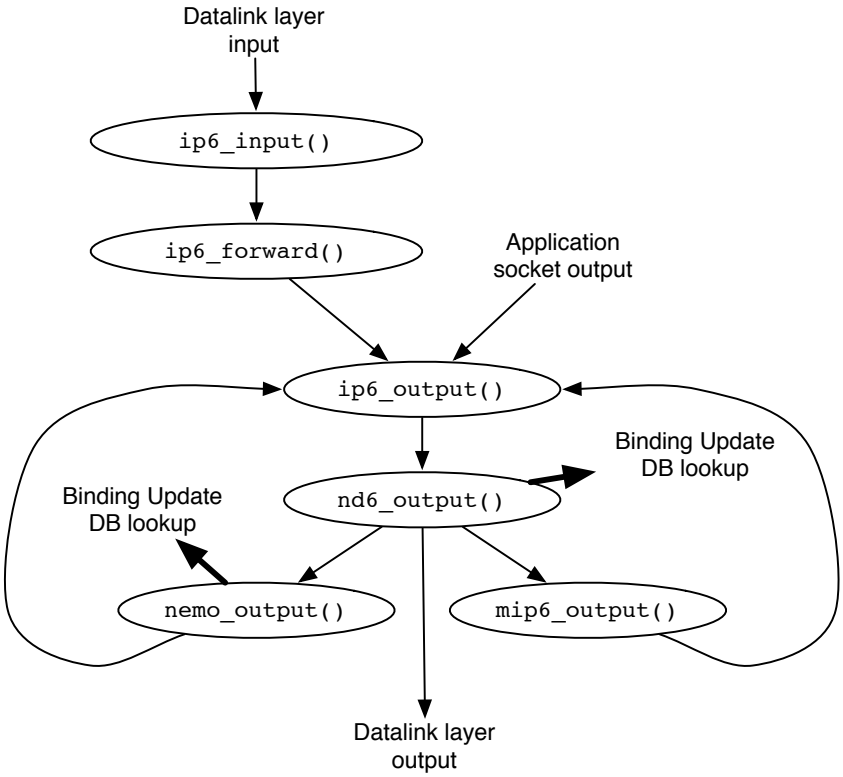


Figure 5.8. Bi-directional output on MN.

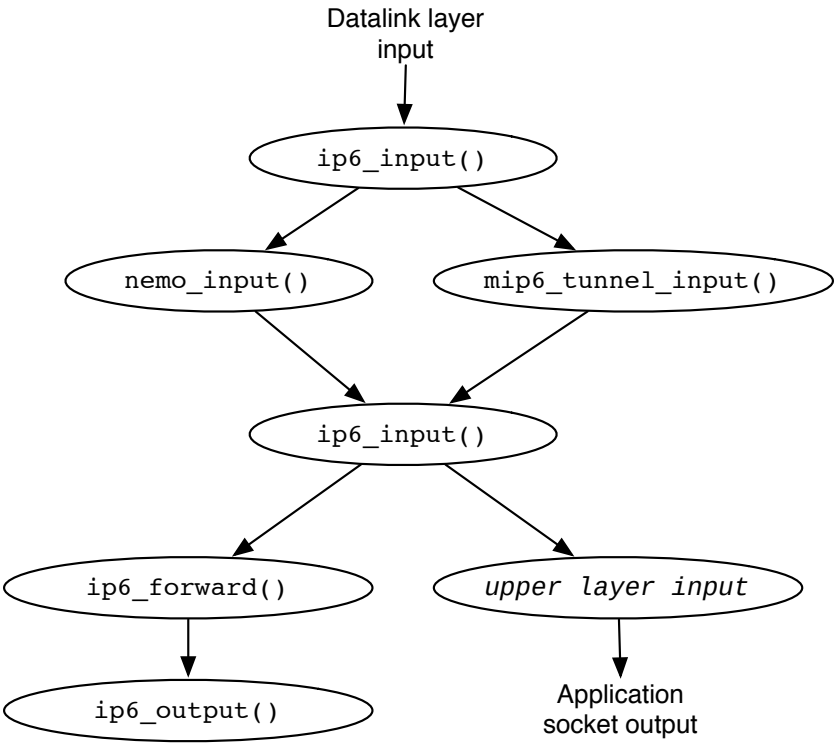


Figure 5.9. Bi-directional input on MN.

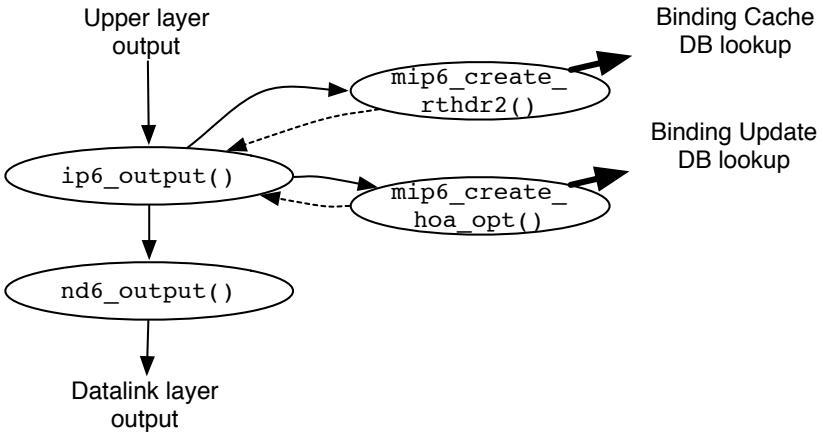


Figure 5.10. Route-optimized output on MN

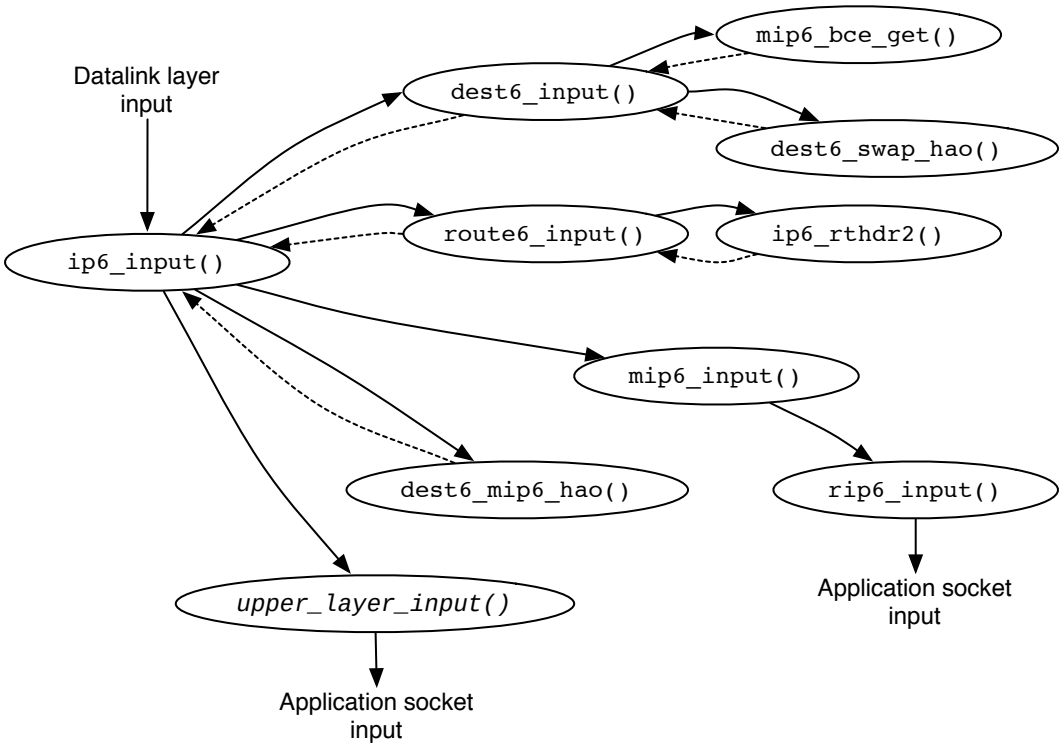


Figure 5.11. Route-optimized input on MN.

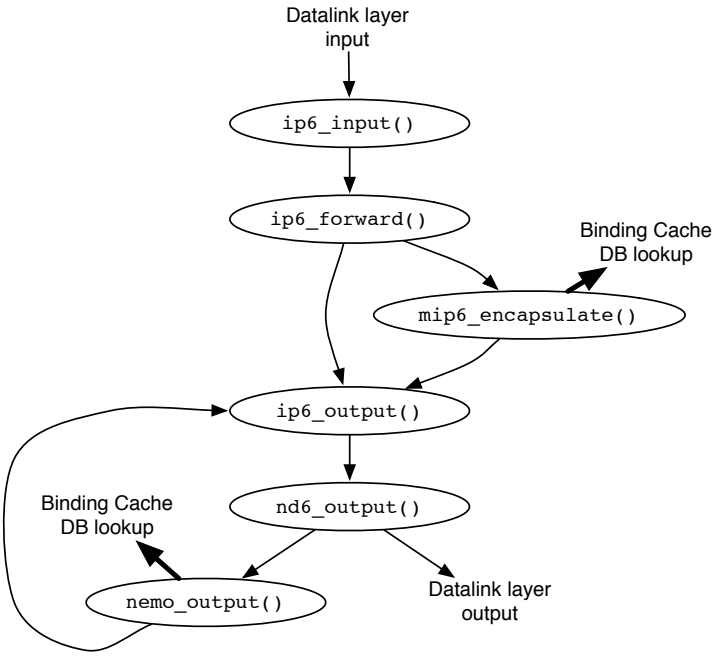


Figure 5.12. Bi-directional output on HA

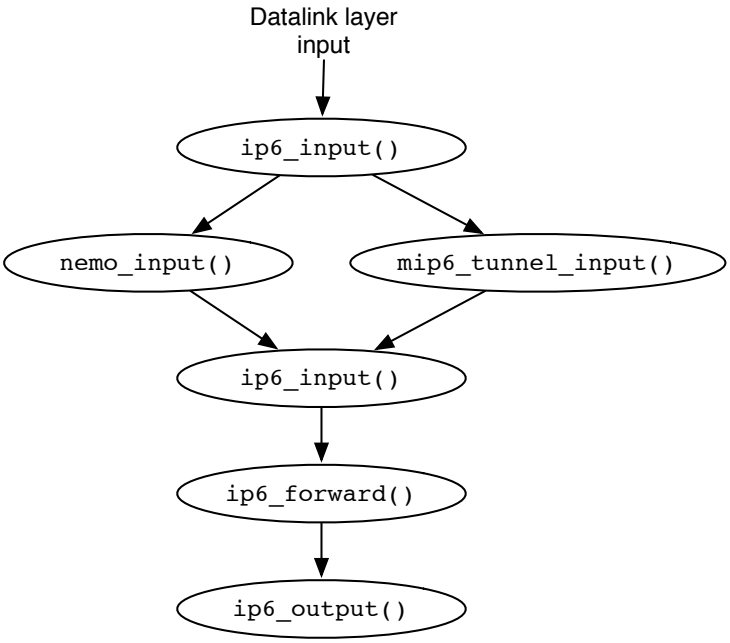


Figure 5.13. Bi-directional input on HA.

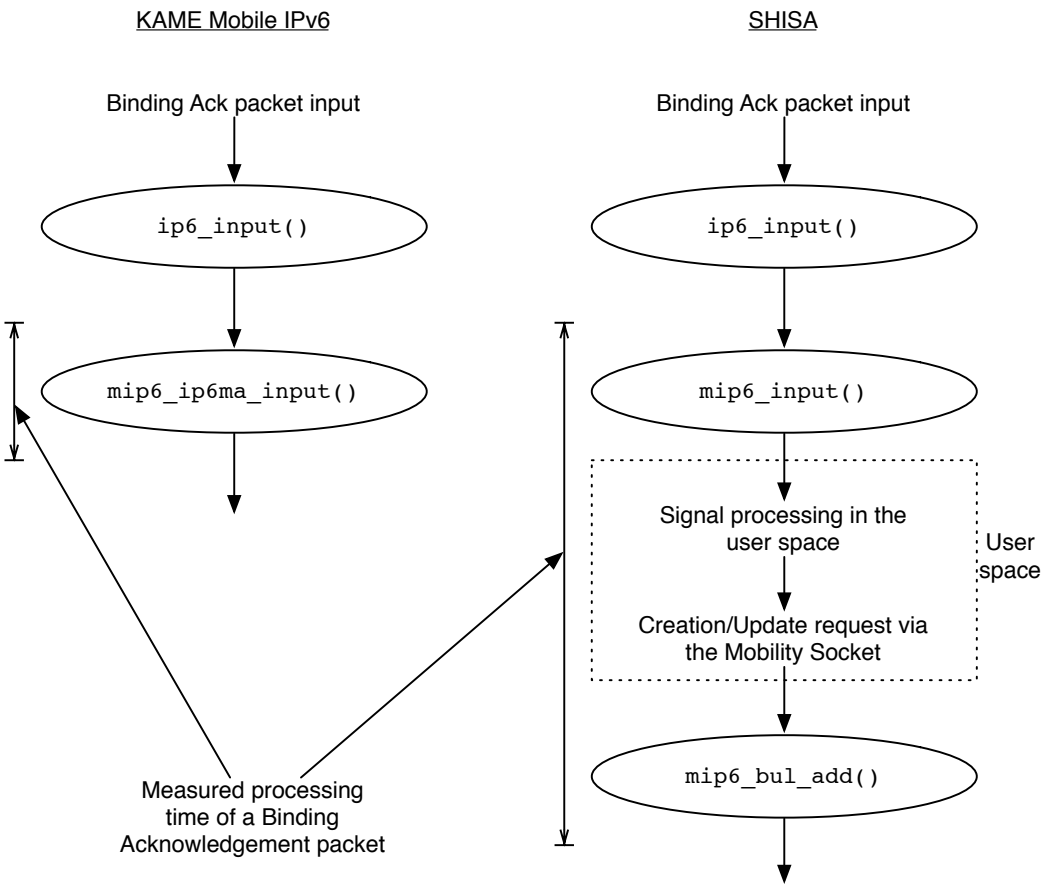


Figure 5.14. Measurement points of a Binding Acknowledgement message processing time for the KAME Mobile IPv6 and the SHISA stack

Chapter 6

Proposal of Smooth Transition Scenario from IPv4 to IPv4/IPv6 Dual Stack Operation using Mobility Technology

This chapter discusses a mechanism which enables network mobility (NEMO) in both IPv4 and IPv6. Providing this kind of technology will encourage IPv4 people to adopt IPv6, which finally makes the infrastructure proposed in this dissertation more useful for them as a base testbed system for their operation or services.

As already discussed in chapter 5, there is an IETF standard NEMO protocol, NEMO Basic Support (NEMO BS). However, it is designed only for IPv6 mobile networks. The basic concept of NEMO BS is a kind of dynamic tunnel configuration protocol. NEMO BS assumes that only IPv6 packets are passed over the tunnel. The proposal explained in this chapter permits to forward IPv4 packets over the configured tunnel created by NEMO BS too, and adds a mechanism to exchange IPv4 network information between a mobile router and its home agent. With this mechanism, it is possible to obtain a dual-stack mobile network even if the mobile router does not have access to an IPv4 access network. A mobile router can move around the IPv6 Internet keeping IPv4 connectivity. The proposal will provides a mobility function to IPv4 nodes accommodated under the

mobile router without changing any IPv4 subsystems. The benefit is important during the transition period from IPv4 to IPv6. The idea has been implemented and confirmed that the proposed mechanism enables an IPv4/IPv6 dual-stack network over IPv6 only network.

1. Problem of IPv6 and mobility deployment and its solution

Adding mobility function to the Internet Protocol (IP) has been discussed for a long time. There are two kinds of mobility protocols in IP area. One is a host mobility protocol, which is specified as Mobile IP (MIPv4) [52] for IP version 4 (IPv4) and Mobile IPv6 (MIPv6) [30] for IP version 6 (IPv6). The other is a network mobility (NEMO) protocol specified as NEMO Basic Support (NEMO BS) [13] for IPv6. Although there are host mobility protocols both for IPv4/IPv6, there is no standard network mobility protocol for IPv4. When the standardization activity of NEMO started, many people thought that there was no need to specify a protocol for IPv4. Everyone thought IPv6 should be the next generation IP and IPv4 would be replaced by IPv6 finally. Considering the situation it is not always a wrong way to focus only on IPv6. However, with regard to the network mobility function, now people have realized that supporting IPv4 is important because of the delay of IPv6 service deployment. It is recently noticed that the shift to IPv6 needs more time than expected initially. It is now forecasted that there will be a long period that requires a dual-stack (IPv4/IPv6) network operation before IPv6 is fully deployed. During the long transition period, most people would need not only IPv6 mobile networks but also IPv4 mobile networks.

In this chapter, a mechanism to realize a dual-stack NEMO is proposed. The proposal does not provide a new mechanism for IPv4, since the IP infrastructure is changing from IPv4 to IPv6. Instead, the proposal provides the mechanism as an extension to NEMO BS which works only on the IPv6 infrastructure. Anyone who is interested in NEMO but has not introduced the technology because of lack of IPv4 support, can use this mechanism without changing the existing IPv4 infrastructure. All they have to do is to prepare the IPv6 access infrastructure and allocate their own IPv4 and IPv6 address blocks for their mobile networks. All

packets from their mobile networks are transmitted to IPv4/IPv6 Internet using NEMO BS with the proposed extension which runs on the IPv6 infrastructure. The IPv6 access infrastructure may be a little difficult to prepare at this time, however, the difficulty will become smaller as the IPv6 deployment progresses. Moreover, it may accelerate the transition from the IPv4 infrastructure to the IPv6 infrastructure, because IPv4 users can use their IPv4 services over the IPv6 only infrastructure with our proposed mechanism.

2. NEMO Basic Support overview

NEMO BS adds a mobility function to IPv6 routers. An entire IPv6 network served by a *mobile router (MR)* which supports NEMO BS, can be a *mobile network*. The nodes inside the mobile network can use static IPv6 addresses which never change regardless of the attachment point of the MR.

Every MR has a home network. A home network is a network to which a MR is originally attached. A MR has a mobile network behind it. A mobile network can attach anywhere in the Internet thanks to the MR. A mobile network has a fixed network prefix (*MNP, mobile network prefix*). MNP never changes regardless of the location of the MR.

A network which is not a home network, is called a foreign network. When a MR attaches to a foreign network, it configures an address of its interface connected to the foreign network by some means, usually by the IPv6 address auto-configuration mechanism. The address assigned to the interface on a foreign network is called a *care-of address (CoA)*. On the other hand, the address assigned while a MR is on its home network is called a *home address (HoA)*. To track the location of the MR, a *home agent (HA)* needs to manage the information of a pair of HoA and CoA. The information is called *binding*. A HA is a special node located on a home network. When the binding information of a MR changes, the MR provides the new binding information to its HA. After the procedure, the MR and its HA configure an IPv6 tunnel between a CoA and HA. While exchanging the binding information, the MR and its HA also provides the MNP to the HA. Routing information of the MNP will be advertised from the HA so that all packets to the nodes behind the MR are routed to the HA. Those packets

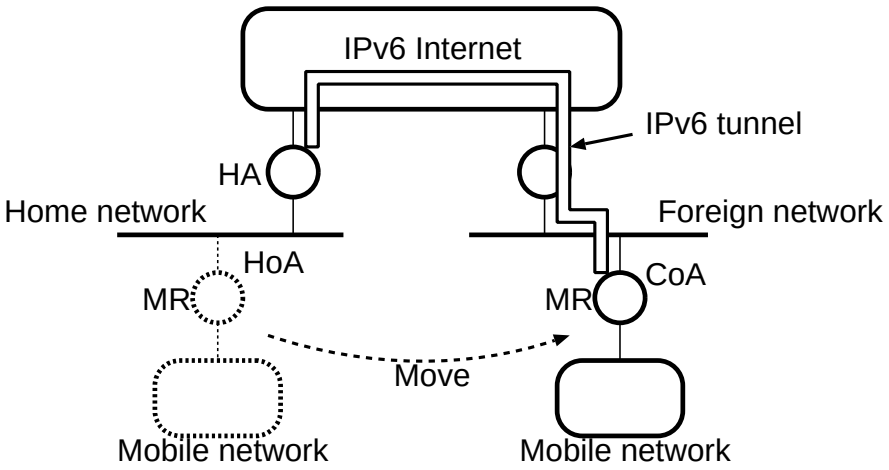


Figure 6.1. The concept of the NEMO technology

are intercepted by the HA and forwarded to the MR using the IPv6 tunnel. On the contrary, all packets generated by the nodes inside the mobile network are tunneled by the MR to its HA. All nodes can communicate with the nodes inside the mobile network, as if they are located on the home network. They even do not notice that the communicating nodes are in a mobile network. Figure 6.1 describes the concept of the NEMO BS protocol.

3. Comparison with other similar proposals

MIPv4 mentions an idea to operate a MIPv4 node as a router to provide an IPv4 mobile network in [52]. This mechanism has one problem. The nodes inside a mobile network send their packets directly from the mobile network. Such packets will be topologically incorrect because the routing information of mobile network is advertised from the home network of the mobile router. This kind of traffic is recently not recommended because of security concerns.

NEMOv4 [39] proposes an extension for MIPv4 similar to NEMO BS. This proposal does not have the problem of the above proposal, since this mechanism can use a bi-directional tunnel like NEMO BS.

The above two proposals can support IPv4 NEMO, however these proposals

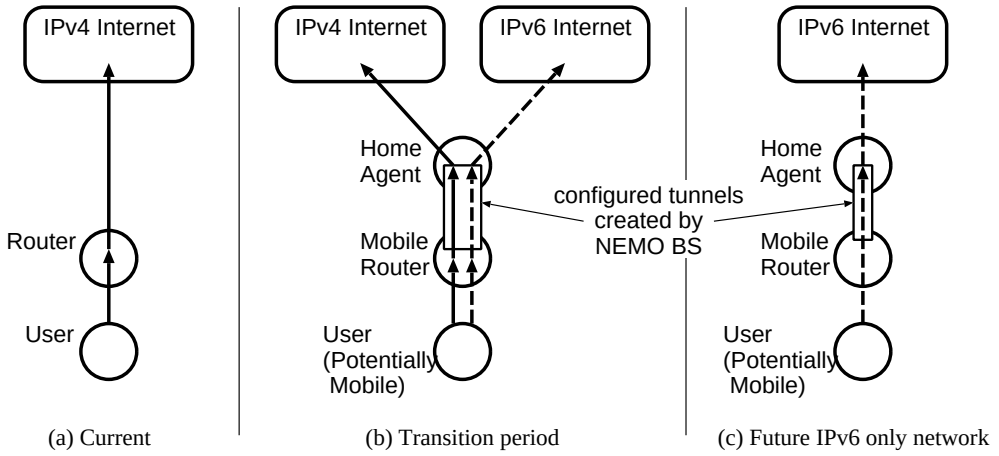


Figure 6.2. Transition from IPv6 to IPv4 with our proposed mechanism

are not thought suitable when considering transition to the future IPv6 Internet. These proposals are both based on IPv4. To provide a dual-stack mobile network which is necessary during transition, both IPv4 NEMO and IPv6 NEMO have to be operated independently. Such a parallel operation increases the operational overhead. In addition, it may cause unstability because of the asynchronous behavior between IPv4 NEMO and IPv6 NEMO.

The proposal does not have such problems, since it is based on IPv6 NEMO only. All packets are sent in a topologically correct manner. There is no asynchronous behavior since IPv4 mobile network operation is performed simultaneously with that of IPv6 in one atomic operation. Moreover, the proposal does not require any new IPv4 technology. Figure 6.2 depicts the transition scenario using the proposed mechanism. End users can use their IPv4 service (a) in the same manner during transition period (b) without any modification on their IPv4 subsystems. In addition, IPv4 users can benefit from mobility of both IPv4/IPv6. When transition has completed, what to do is just to disable IPv4 (c), without changing any of the already deployed IPv6 network.

Dual Stack Mobile IPv6 [14] is the best resembling mechanism to the proposal discussed in this chapter. It discusses a protocol which enables IPv4 HoA and IPv4 MNP as a part of the protocol. The idea is almost the same with the one discussed in this chapter. However the draft originally focused on IPv4 host

mobility and lacked to describe detailed procedures to operate NEMO BS for IPv4. With the discussion at IETF with the authors and working group members, the latest specification of [14] has merged the idea proposed in this chapter.

4. Usage scenarios

When considering providing NEMO services to, for example, buses or trains, it is unrealistic to provide only IPv6 NEMO, since most people are still using IPv4. The proposed mechanism can provide a dual-stack NEMO in such situations. The access lines to buses or trains need to be IPv6 [17], and it is not very difficult. Most of advanced ISPs are already IPv6-ready and it is easy to provide dedicated IPv6 lines for such specialized purposes [26]. From the user's point of view, it is not important if the access line is IPv6 only or not, as long as users can use IPv4. The benefit of this scenario is that IPv4 can be provided transparently to users in addition to IPv6. Also the access lines to IPv6 can be changed transparently. This mechanism can be applied even to a static network, such as small SOHO site. Thanks to NEMO BS, such a dual-stack static network can provide the access line redundancy by subscribing multiple IPv6 ISPs. If one of the ISPs goes down, the site can use another ISP's address as a new CoA. This may be a good reason to transit to the IPv6 infrastructure for users keeping the current IPv4 alive.

5. Proposed mechanism

NEMO BS provides IPv6 connectivity over a tunnel connection created between a MR and a HA. To provide IPv4 connectivity simultaneously, it is needed to transmit IPv4 traffic over the tunnel. The NEMO BS specification does not mention the upper layer protocol carried over the tunnel, although the specification assumes IPv6 implicitly. Technically speaking, it is possible to carry any kind of layer 3 protocols over the tunnel as long as there are proper routing entries. The following new operation rules to NEMO BS are introduced.

1. A MR can have an IPv4 network on its internal interface.

2. The MR notifies the IPv4 network information which it has on its internal interface with its HA.

3. The HA advertises the IPv4 MNP information which is located behind the MR.

4. The MR forwards IPv4 packets from its internal interface to its HA using the tunnel created between them.

5. The HA forwards IPv4 packets whose destination prefix is the MNP behind the MR to the tunnel created between them.

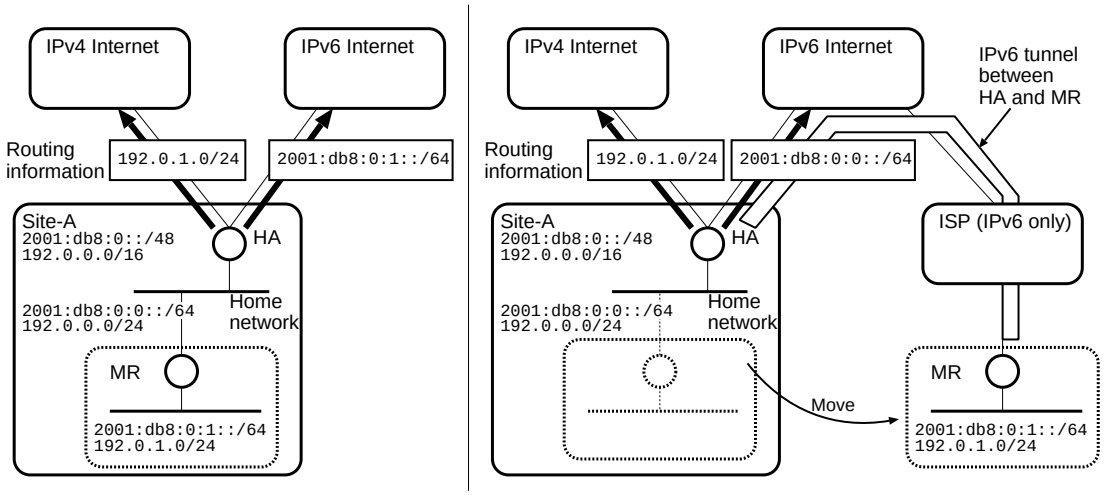
Figure 6.3 depicts a sample network configuration. Site-A is a dual-stack site and has a home network whose prefixes are `2001:db8:0:0::/64` and `192.0.0.0/24`. There is a HA and a MR attached to the home network. The MR has the MNPs of `2001:db8:0:1::/64` and `192.0.1.0/24`. The HA advertises the route information of MNPs to the Internet. Figure 6.3.(a) describes the logical topology of the mobile network. The MR is logically attached to its home network. Figure 6.3.(b) is the actual network topology when the MR is in a foreign network. The MR only needs IPv6 connectivity to provide both IPv4/IPv6 accesses to its mobile network. When the MR attaches to a foreign network, it registers the IPv6 CoA assigned to its HA on the foreign network.

During the registration procedure, the MR notifies the IPv4 MNP which is attached to the MR to its HA in addition to normal NEMO BS operation. There are two possible mechanisms to exchange the IPv4 MNP.

1. Configuring the MNP value on the HA and the MR in advance

2. Notifying the MNP value using NEMO BS signaling messages

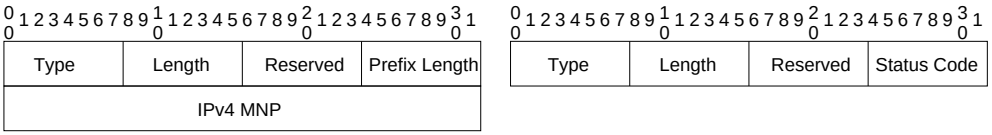
In the first method, the HA and the MR already know the MNP value, in this case `192.0.1.0/24`. After finishing the registration, the HA installs a routing entry, which indicates all packets sent to `192.0.1.0/24` will be forwarded via a tunnel interface created between the HA and the MR. Similarly, the MR installs a forwarding entry which indicates that every traffic from its internal interface will be forwarded via the tunnel.



(a) Logical network topology

(b) Actual network topology

Figure 6.3. Network topologies



(a) IPv4 MNP Option

(b) IPv4 MNP Registration Status Option

Figure 6.4. The option format to exchange IPv4 network information

The second method is to notify IPv4 MNP using a Binding Update message and a Binding Acknowledgment message, which are used by the NEMO BS signaling procedure to carry binding information of a MR. New options for the messages were defined to carry IPv4 MNP information. Figure 6.4 depicts the option formats.

The IPv4 MNP Option (figure 6.4.(a)) is put in a Binding Update message sent from a MR to a HA. The IPv4 MNP field contains the IPv4 MNP assigned to the internal interface of the MR. In the sample case, `192.0.1.0` is put in the field. The prefix length of the mobile network, 24 in this case, is put in the Prefix Length field. A HA replies a Binding Acknowledgment message with the IPv4 MNP Registration Status Option (figure 6.4.(b)) if it recognizes the IPv4

MNP Option. If the HA does not know the option, it simply ignores the option. A MR can know weather its HA supports IPv4 NEMO or not by checking if the received Binding Acknowledgment has the IPv4 MNP Registration Status Option. If the HA does not support the function, the MR gives up to provide IPv4 connectivity to its nodes in its internal network. The Status Code field of the IPv4 MNP Registration Status indicates whether the information exchange of the IPv4 network succeeded or failed. Once the exchange has been succeeded, a HA and a MR install routing entries so that the nodes inside the mobile network can send and receive their IPv4 packets, similar to the first method.

The detailed specification of the protocol extension is described in [64].

6. Implementation details

The mechanism is implemented in the SHISA MIPv6/NEMO BS protocol stack [88]. SHISA is a part of the KAME IPv6 protocol stack provided by the KAME project [86] and freely available from the home page of the project.

7. SHISA overview

SHISA consists of several function components. The packet forwarding and extension header manipulation for normal packets are handled in the kernel. For NEMO BS signal packet processing, SHISA provides 4 daemon programs in user space. The first program is `mrd`, which processes NEMO BS signaling messages to be handled on a MR. The second program is `had`, which processes NEMO BS signaling messages to be handled on a HA. The third program is `babymdd`, which detects movement of a MR and notifies the event to `mrd` program. The last program is `nemonetd`, which manages bi-directional tunnel connections between a MR and a HA. Each program communicates with other programs using a dedicated socket interface [43]. For example, when `babymdd` detects movement of a MR, it broadcasts the event to other programs. When `mrd` receives the event, it start the registration procedure since the movement means change of the binding information. After `mrd` completes the registration procedure, it broadcasts the event to other programs. `nemonetd` updates a tunnel interface and routing entries

```
interface mip0 {
    prefixtable {
        2001:240:1:280::beef    2001:240:1:281::/64    explicit;
        2001:240:1:280::beef    10.0.0.0/24            explicit;
    };
};
ipv4-dummy-tunnel {
    nemo0    169.254.0.1    169.254.0.2;
};
```

Figure 6.5. MNP database file

based on the registration information.

8. Implementation of the proposed mechanism

The following two functions are necessary to be extended.

1. An IPv4 MNP exchange mechanism
2. A mechanism to install IPv4 route entries

The IPv6 MNP is managed by `mrd` and `had` when NEMO BS is operated for IPv6 NEMO. The prefix management database has been extended to handle IPv4 MNP. `mrd` and `had` may exchange the MNP information explicitly. As discussed in the previous section, new options to exchange the IPv4 MNP information have been defined. `mrd` and `had` programs were also extended to send and receive the configured IPv4 MNP in the form of options described in figure 6.4.

`nemonetd` configures a tunnel interface between a MR and a HA once the NEMO BS registration message processing has been completed successfully. IPv6 route information for the mobile network is installed at this tunnel setup time. Our extended `nemonetd` program also installs IPv4 route entries on the same configured tunnel at the same time on both MR and HA.

Figure 6.5 shows the sample database file for MNP definition. In the original SHISA, only IPv6 MNP can be specified in a `prefixtable` section. The format has been extended to accept both IPv4/IPv6 MNPs by changing the internal

data storage from `in6_addr{}` structure, which can only keep IPv6 address, to `sockaddr_storage{}` structure which can store any kind of layer 3 address.

The NEMO BS implementation on SHISA uses the unnumbered tunnel mechanism for the tunnel created between a MN and a HA. The implementation forwards both IPv4 and IPv6 traffic using the tunnel. However, BSD systems have a problem concerning to this design. BSD systems do not have a mechanism for the unnumbered tunnel for IPv4. Some addresses have to be assigned on the tunnel interfaces to inject IPv4 route information. `ipv4-dummy-tunnel` specifies IPv4 addresses assigned to a tunnel interface. The addresses are taken from the IPv4 address space reserved for link-local use in this example. This implementation decision works in most cases except one case. When a MR itself initiates traffic to an IPv4 node, the address assigned to the outgoing interface is used as a source address. In the example case, a link-local IPv4 address is chosen but the MR cannot receive reverse traffic. To solve the problem, it is required to implement the unnumbered tunnel for IPv4. All other nodes connected to the mobile network do not have any problem.

9. Verification of the implementation

Figure 6.6 is the topology used to verify the function. We used 3 IPv6 networks and two IPv4 networks. `2001:240:0:280::/64` is a home network, `2001:240:0:200::/64` is a foreign network and `2001:240:0:281::/64` is a mobile network for IPv6. `192.168.64.0/25` is an IPv4 network for the home link and `10.0.0.0/24` is an IPv4 mobile network. A MR is logically attached to the home network (`2001:240:0:280::/64`). When the MR moves to the foreign network, it registers two MNPs, one is `2001:240:0:281::/64` and the other is `10.0.0.0/24` using the proposed mechanism.

As a result of the protocol operation, the MR and the HA have a routing tables as shown in figure 6.7. There is a route entry to `10.0.0.0/24` on the HA. The traffic is routed to the `nemo0` interface, which is a tunnel interface between the HA and the MR. Also, the MR has an IPv4 default route entry whose destination is `nemo0`. This means that all IPv4 traffic on the MR will be forwarded to the HA through the `nemo0` interface.

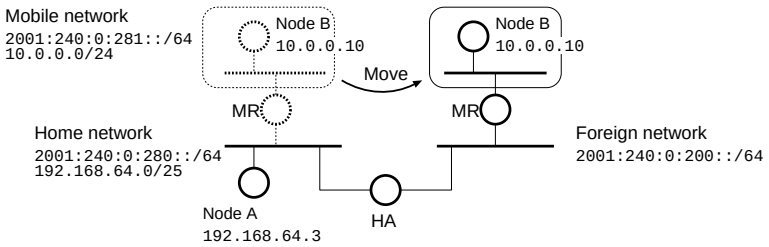


Figure 6.6. The topology used to verify the proposed mechanism

The following IPv4 applications are confirmed to work with the implementation.

- ping from the node A to the node B
- SSH remote login from the node A to the node B
- Accessing to the Web service running on the node A from the node B

The result means that the node A, which is behind the MR, can use its IPv4 applications without any modification thanks to the MR and our proposed mechanism, even if the MR is connected to the Internet only with the IPv6 infrastructure.

10. Summary

NEMO BS provides the mobility function to IPv6 network, which is necessary in the coming ubiquitous IPv6 Internet where tons of moving nodes connect to the Internet with variety of access mechanisms. However, because of transition problems, both IPv4 and IPv6 will be used for a long time. In this chapter, the dual-stack NEMO technology which depends only on the IPv6 infrastructure was proposed. There are other proposals to enable IPv4 mobile network as described in [52] and [39]. Although it is possible to operate them and NEMO BS in parallel to provide a dual-stack NEMO, however, such a combination causes the problem of generating topologically incorrect packets or the unstability problem due to asynchronous behaviors of two mobility protocols. The proposed mechanism does not have such problems and is more efficient when considering the transition

Routing table on HA (NetBSD2.0.2 + SHISA with our extension)

```
Internet:
Destination        Gateway            Flags     Refs     Use    Mtu  Interface
10/24              127.0.0.1          UGS          0       0      -  nemo0
192.168.64/25      link#1             UC           3       0      -  fxp0
192.168.64.1       00:02:b3:3a:8a:6f  UHLc         1       2      -  fxp0

Internet6:
Destination          Gateway           Flags     Refs     Use    Mtu  Interface
2001:240:1:280::     00:02:b3:3a:84:ac UHL         0        0      -  lo0 =>
2001:240:1:280::/64  link#2            UC          0        0      -  fxp1 =>
2001:240:1:280::/64  link#2            UC          0        0      -  fxp1 =>
2001:240:1:280::/57  ::1               UR          0        0      -  lo0
2001:240:1:280::1    00:02:b3:3a:84:ac UHL         0        0      -  lo0
2001:240:1:280::beef 00:02:b3:3a:84:ac UHLS2       2        0      -  fxp1
2001:240:1:281::/64  ::1               UGS         0        0      -  nemo0
```

Routing table on MR (FreeBSD5.4-RELEASE + SHISA with our extension)

```
Internet:
Destination        Gateway            Flags    Refs      Use  Netif Expire
default            127.0.0.1          UGS         0        0  nemo0
10.0.0.1           10.0.0.1           UH          0        0    lo0
10.0.1/24          link#1             UC          0        0    em0
10.0.1.1           00:11:25:32:d9:8c  UHLW        0        2    lo0
169.254.0.2        169.254.0.1        UH          0        0  nemo0

Internet6:
Destination                       Gateway           Flags      Netif Expire
default                           ::1               UGS       nemo0
2001:240:1:200:202:b3ff:fe3a:87d9 00:02:b3:3a:87:d9 UHL         em0
2001:240:1:200:211:25ff:fe32:d98c 00:11:25:32:d9:8c UHL         lo0
2001:240:1:280::beef              link#2            UHL         lo0
2001:240:1:281::/64               ::1               U           lo0
2001:240:1:281::1                 link#3            UHL         lo0
```

Figure 6.7. Routing tables on MR and HA

scenario from IPv4 to IPv6. Using the base mobility platform, the proposed dual-stack mechanism was built on the SHISA stack and it was evaluated to work as expected.

Chapter 7

Mobile Network with Lower Packet Loss Rate using MCoA Extension

NEMO Basic Support (NEMO BS) adds a mobility function to IPv6 routers, and such router is called as mobile router. The network behind the mobile router (mobile network) becomes logically static. This function is considered useful when a network has a lot of nodes which do not have a mobility function but move with the network. The typical usage of this technology is the network provided in a transportation system such as bus or airplane. Other example is a small office network which wants to function as multihomed network using mobility technology as discussed in chapter 6.

In the base specification of NEMO BS, a mobile router can use one IPv6 address as its attachment point at one time. Therefore, the mobile router and its mobile network will face service disruption of network connectivity while the mobile router is moving from one network to other network. In this chapter, two operational experiments of network mobility are explained. One is an experiment using only the base NEMO BS function. The other is an experiment of seamless handover using multiple network interfaces to solve the problem. In these experiments, the mobile networks were actual conference networks which held a few hundred people. The result shows that the usage of multiple interfaces provides the one-third to one-tenth packet loss rate compared to the case in which the

technology is not used. Also most attendees of the conference did not notice the movement of the networks, which means that the technology can be used for a realistic solution for the seamless handover.

1. Background

With Network Mobility Basic Support, an entire IPv6 network served by a mobile router can be a mobile network. The nodes inside the mobile network can use static IPv6 addresses which never change regardless of the attachment point of the mobile router.

One of possible applications of this technology is to provide a network in moving entities, such as buses, trains, airplanes, and so on. Such a moving entity may connect various network access points depending on its location. The address assigned to the mobile router from the router of each access point will differ depending on each access point, since IPv6 addresses are usually assigned based on the network topology. The NEMO BS technology hides the address change of the mobile router and provides transparent access to the Internet from the nodes inside the mobile network.

Although the specification has been completed, more experience is required to deploy the technology widely. The more experience of the real operation is necessary to prove the usefulness of the technology and to find any kind of problems which is not noticeable from the specification. Based on this policy, we performed two experiments of a mobile network which accommodated the entire conference network under a mobile router in the WIDE [87] 2005 autumn meeting and the WIDE 2006 spring meeting. The mobile router changed its point of attachment from time to time, providing transparent access to the nodes in the meeting network. The number of attendees of each meeting was almost 250 and most of them used their laptop computers to connect to the meeting network. This network can be thought as a kind of moving entity which carries many people connecting to the inside network. It can be considered as a realistic example of transportation systems, like a train.

In this chapter, the related technologies of these experiments are briefly explained in section 2. In section 3, the experiment done at the WIDE 2005 autumn

camp meeting using only the base NEMO BS function is explained. Section 4 explains the second experiment done at the WIDE 2006 spring camp using multiple network interfaces to solve the problem which was found in the first experiment. Section 5 compares the two experiments and section 6 concludes this chapter.

2. Related Works

In this chapter, the approach and the result of the experiments performed to obtain a mobile network with smooth handover are shown. There are several approaches to make the movement seamless. One of such technologies is FMIPv6 [37].

FMIPv6 defines extensions to both a mobile node and access routers which provide foreign networks to the mobile node. The mobile node supporting FMIPv6 initiates movement by using some prediction mechanisms of its movement. The prediction mechanisms are not defined in the specification. Usually the mechanism is implemented using the layer 2 information. The mobile node requests new access router information which provides a new foreign network to which the mobile node is going to move. The mobile node notifies its movement by sending a message to the access router to which the mobile node connects currently. After receiving the message, the access router creates a bi-directional tunnel with the new access router and starts forwarding packets addressed to the mobile node. The packets tunneled from the current access router to the new access router are buffered at the new access router until the mobile node attaches to the new foreign network. When the mobile node moves to the new foreign network, it notifies the new access router that it has completed the movement. The new access router then delivers the buffered packets to the mobile node.

Even if the mobile node could not send the message that indicates the node was going to move from its current access router, the mobile node can send the same message after it has moved to the new foreign network. In this case, some of the packets sent to the mobile node while it is moving may be lost. However, if the notification time of the movement to its old access router is smaller enough than the registration time of the mobile node to its home agent, the loss will be smaller than the case not using FMIPv6.

FMIPv6 requires changes to the infrastructure network. The access routers which provide foreign networks to the mobile node must support the protocol. Also, access routers have to be authorized by each other because they create bi-directional tunnels which forward packets of the mobile node. FMIPv6 can be used if the entire network that the mobile node attaches is managed by one operational entity (e.g. one ISP). However if we need to utilize multiple Internet service providers to get the access to the Internet, it is difficult to deploy FMIPv6.

Another approach was chosen in this dissertation which utilizes multiple connections concurrently while performing handover from one foreign network to an other network. Although this mechanism requires additional equipment to the mobile node (that is, additional network interfaces), it does not require any modification to the existing infrastructure network. It requires extensions to the mobile node and its home agent. This approach is easy to deploy considering that the Internet is provided by the several different access service providers. The implicit assumption of this approach is that the mobile node needs to be covered by at least two access networks at the same time. Otherwise, the node loses the connectivity and some packets during handover. However, considering the recent progress of wireless Internet access services, this assumption seems feasible.

3. Moving Network using Basic Function Only

Our first experiment of NEMO BS in a real environment was the WIDE 2005 autumn camp meeting. At the meeting, a wireless network was provided to the attendees. The network was designed as a mobile network. In this meeting, the MR had two network interfaces; one was used for external connectivity and the other was used to provide a mobile network. The mobility protocol used at this meeting was NEMO BS with no extension.

3.1 Network Topology

Figure 7.1 depicts the topology used. The network infrastructure created at the camp meeting site had a network `2001:200:0:ffff::/64`, which was a mobile network. The MR provided the mobile network to users who participated in the meeting. The home address of the MR was `2001:200:0:fffe::4649` and the MR

connected to two different networks while the meeting was being held. One network was extended from the WIDE Nara NOC (Network Operation Center), located in Nara prefecture, Japan. The network prefix was `2001:200:0:8ff::/64`. The other network was extended from the WIDE K2 NOC, located in Kanagawa prefecture, Japan. The network prefix was `2001:200:0:80bb::/64`. The MR acquired two different IPv6 addresses as CoAs based on its attached network. Each time the MR changed its CoA, the MR sent a message to its HA to notify that the current attachment point had been changed. The HA was placed at the WIDE Fujisawa NOC, Kanagawa prefecture. The address was `2001:200:0:fffe::1000`. The routing information for the mobile network (`2001:200:0:ffff::/64`) was advertised from the HA so that all traffic to the mobile network was routed to the HA. The HA forwards all traffic addressed to the mobile network to the tunnel interface to the MR. On the other hand, all traffic generated from the IPv6 nodes inside the mobile network were forwarded by the MR to the HA using the tunnel connection.

The IPv6 connections between two WIDE NOCs and access routers at the meeting site were created using IPv6 over IPv4 tunnels, because native IPv6 connection services could not be got at the meeting place.

The physical network topology is shown in Figure 7.2. The dotted objects mean that they are parts of IPv4 network. The IPv4 connections to the meeting place were provided by two different ISPs, ISP-1 and ISP-2. These two ISPs and WIDE are inter-connected at one of the Internet Exchange Points in Japan. The tunnel connections, which were used to create two IPv6 networks, were created over the two IPv4 ISP connections respectively. The reason why this configuration was chosen is to use different ISPs over those two tunnels as many as possible. If using only one ISP service to get IPv4 access and created two IPv6 over IPv4 tunnels over the one single IPv4 ISP connection, the most part of the tunnel path would be same, since the WIDE network and ISPs in Japan usually share only one exchange point. The IPv4 path from the meeting place to the exchange point was same even if we had two different IPv6 over IPv4 tunnels. As figure 7.2 shows, almost different tunnel paths could be achieved by using two different ISPs for the IPv4 connections.

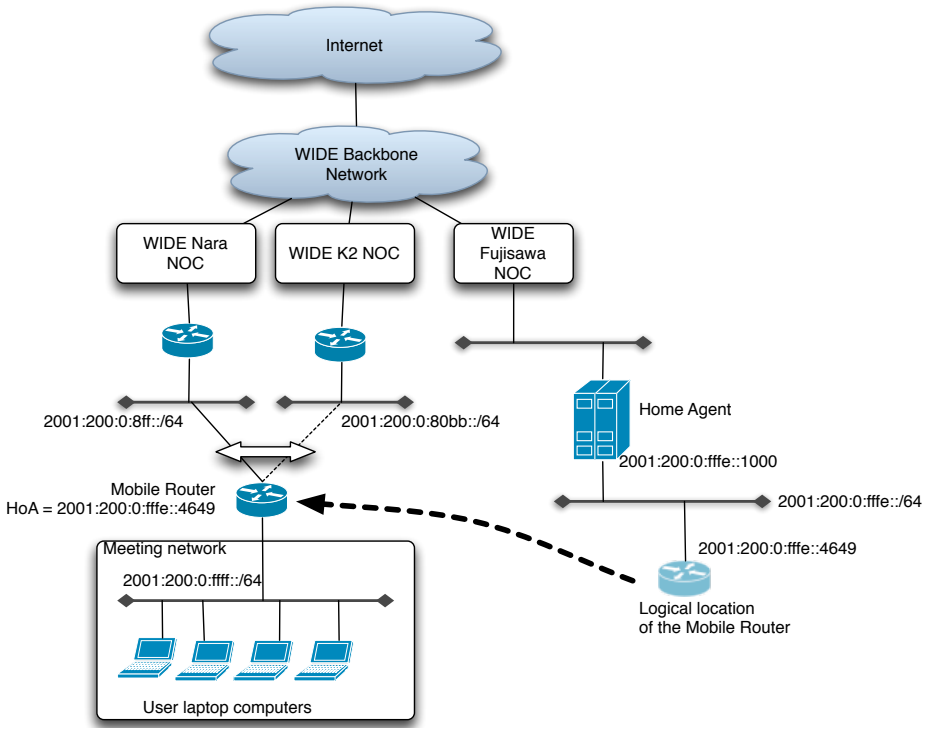


Figure 7.1. The network topology of the experiment at the WIDE 2005 autumn meeting

3.2 System Configuration

The SHISA Mobile IPv6/NEMO BS protocol stack [88] was used in the experiment. The hardware specification and software configuration are described in table 7.1.

The SHISA stack can be available from the KAME project [86] home page as a part of the KAME IPv6 protocol stack software. In this experiment, FreeBSD5.4-RELEASE with the KAME snapshot was chosen as the MR and NetBSD2.0.2 with the KAME snapshot as the HA. To advertise the routing information of the MR (`2001:200:0:fffe::/64`) and the MR's mobile network (`2001:200:0:ffff::/64`), GNU Zebra [92] and ospf6d on the HA were used.

The MR was configured not to perform the Duplicate Address Detection (DAD)

Table 7.1. System configuration

Node	Hardware	Software
MR	Pentium4 1.8G, 256MB memory	FreeBSD5.4-RELEASE + SHISA 20050905 snapshot
HA	Pentium4 1.8G, 256MB memory	NetBSD2.0.2 + SHISA 20050829 snapshot, GNU Zebra ospf6d

to make its movement quicker. Since the node which attached to the access networks was only the MR, the DAD procedure could safely omitted, which usually requires 1 second to complete. The MR detected the movement of itself by receiving Router Advertisement (RA) messages from access routers. In this experiment, the access routers sent the messages every 3 seconds which is the maximum rate defined in the Neighbor Discovery specification [46].

3.3 Movement Frequency

The meeting was held for 3 days. During the meeting, the movement frequency was dynamically changed based on the user request. Figure 7.3 is the user interface used at the meeting. The movement frequency is changed every 10 minutes based on the total number of users who requested more frequent movement and less frequent movement. If the number of people who requested more frequency were greater than the number of people who requested less frequency, the movement frequency would be increased. If the comparison was opposite, the frequency would be decreased.

At the initial stage of the demonstration, the minimum frequency was set to every 20 seconds and the maximum frequency to every 60 seconds with 5 seconds as a step. However, from the day 2 of the camp meeting, the minimum frequency was changed to every 60 seconds and the maximum frequency to 300 seconds with 60 seconds as a step, because of the problem discussed in the next section.

Table 7.2. Packet loss rate at the second day of the WIDE 2005 autumn camp meeting

From	Sent/Received	Loss rate
Node inside	10842/7408	31.7%
MR	2280/2249	1.4%

3.4 Result of the First Experiment

Figure 7.4 depicts the transition of the movement frequency during the meeting. As we mentioned in the previous section, the frequency was very high in the first day of the meeting. It can be seen that the frequency is changing dynamically and randomly, which means that the participants voted their requests to increase or decrease the frequency based on their feeling of the speed of the network movement. The data of the latter half of the third day is not correct because of the misconfiguration of the switch which changes the point of attachment of the MR. The movement frequency request mechanism was manually stopped for trouble shooting of the network, and the movement frequency became more than 300 seconds, which could not be set by the frequency voting system.

The range of the frequency that the users can change was changed from the day 2, since we got a lot of request that the service disruption time was too long when changing the point of attachment of the MR. In fact, there was about 20 seconds connection disruption during the MR was changing its attached network. Table 7.2 shows the packet loss rate using ICMPv6 echo request/reply messages from a node inside the mobile network and from the MR itself taken at the second day. The echo request messages were sent every one second to the destination node which was one of the IPv6 nodes located in the WIDE K2 NOC.

As shown in figure 7.4, the movement interval was almost 60 seconds at the second day. The mean time for the MR to detect its movement is 1.5 second, since we configured the MR not to perform the DAD procedure and the access routers to send RA messages every 3 seconds. In theory, the echo request messages are lost during detecting movement. The theoretical packet less rate is as follows.

$$(DetectionTime/MovementInterval) \times 100 \tag{7.1}$$

In this case, the theoretical loss rate is $(1.5/60) \times 100 = 2.5\%$. The actual loss rate from the MR was 1.4% which was better result compared to the theoretical value. However, the packet loss rate from the internal nodes was 31.7% which was high enough to make the communication of internal nodes unstable. From Equation (7.1), the loss rate implies the internal nodes lost their connectivity almost 19 seconds on every movement. Considering that the packet loss rate was similar to the theoretical value when the echo request was sent from the MR, the service disruption during the movement affected the forwarding function of the MR. That means, if the disruption during handover can be reduced, the forwarding performance will be better.

4. Moving Network using Multiple CoAs

We had another network mobility experiment at the WIDE 2006 spring camp meeting. This time, I tried to solve the problem observed at the WIDE 2005 autumn meeting. The Multiple CoAs Registration mechanism [81] was used as a solution which makes it possible to use multiple network interfaces concurrently. In the previous experiment, the MR had one network interface only. Because of this, it was impossible to avoid the service disruption during handover. If it is possible to use multiple network interfaces concurrently, a network interface can be prepared on the new foreign network to which the MR is going to move, before disconnecting from the old foreign network.

4.1 Multiple CoAs Overview

The Multiple CoAs Registration mechanism is an extension of Mobile IPv6/NEMO BS. In the base specifications of Mobile IPv6/NEMO BS, a mobile host (MH) or a mobile router can only register one CoA at one time. Considering the recent progress of wireless communication technology and small devices, it can be expected that many devices will have multiple communication mechanisms in the future. The support for simultaneous use of multiple network interfaces is required. In the Multiple CoAs mechanism, a MH or MR assigns unique identifiers to its network interfaces. When the MH or MR registers its CoA, it also sends the unique identifier assigned to the network interface on which the CoA is assigned.

The MH, MR and HA use the pair of the HoA and the unique identifier to identify the particular network interface of the MH or MR. Using this technology, the traffic between the HA and the MH or MR can be distributed over the multiple bi-directional tunnels established between them. Figure 7.5 shows the idea of the Multiple CoAs.

In figure 7.5, the MR attaches to two foreign networks provided by the Access Router1 and the Access Router2 respectively. The MR registers its CoAs (CoA1 and CoA2) at the same time to its HA. As a result, two bi-directional tunnels are established between them. The traffic can be distributed between the two tunnels. The distribution policy is not defined in the specification and it depends on the local policy of the network operator.

4.2 Network Topology

Figure 7.6 shows the network topology used at the WIDE 2006 spring meeting. The home network was located at the WIDE K2 NOC (`2001:200:0:8430::/64`). The home address of the MR was `2001:200:0:8430::4649` and the address of its home agent was `2001:200:0:8430::1000`. The mobile network prefix was `2001:200:0:8470::/56`. This time, we allocated /56 network prefix for the meeting network, since the program committee of the meeting required more than one IPv6 subnets. Foreign networks were provided from the WIDE Fujisawa NOC. The prefixes of the foreign networks were `2001:200:0:ff60::/64`, `2001:200:0:ff61::/64` and `2001:200:0:ff62::/64`. Unlike the WIDE 2005 autumn camp meeting, three foreign networks were provided at this camp meeting. Two of them were provided using T1 dedicated lines and the other network was provided using a satellite link.

Similar to the WIDE 2005 autumn camp meeting, the IPv6 foreign networks were provided using IPv6 over IPv4 tunnels. However, unlike the network configuration of the previous experiment, different physical paths for the two T1 lines were not allocated. The reason was that there was only one ISP that could provide the Internet connection to the meeting place.

Table 7.3. System configuration for the WIDE 2006 spring meeting

Node	Hardware	Software
MR	Pentium4 1.8G, 256MB memory	NetBSD2.0.2 + SHISA 20060320 snapshot
HA	Pentium4 2.4G, 1GB memory	NetBSD2.0.2 + SHISA 20060320 snapshot

4.3 System Configuration

The hardware specification and software configuration are described in table 7.3.

In the second experiment, NetBSD2.0.2 with the KAME snapshot was used for both the MR and the HA. The routing information for the meeting network (`2001:200:0:8470::/59`) was distributed by `route6d` running on the HA.

The traffic distribution policy for outgoing packets was to use the network attached to the MR most recently. Although we had three different networks at this meeting, we did not use them as a way for load-balancing, but focused on the smooth handover.

The MR was configured to perform the DAD procedure. This time, the DAD procedure should not make a big effect on movement detection because the MR could complete the movement procedure before disconnecting from the old network. The interval of RA messages sent from access routers was 3 seconds.

4.4 Movement Frequency

The meeting was held for 3 days. Different from the previous meeting, the movement frequency was fixed to every 60 seconds. This time, to utilize the multiple network interfaces equipped to the MR, the connection at one network interface was configured to last 90 seconds. Figure 7.7 shows the strategy of the movement. The upper graph shows the strategy when the MR uses two network interfaces and the lower graph shows the case where there are three network interfaces.

The last 30 seconds of each connection was used as a migration period to the next connection. During this period, the MR sent a BU message to its HA. After the MR and the HA completed the registration procedure, every traffic to and from the meeting network was sent over the new bi-directional tunnel established

Table 7.4. Packet loss rate at the WIDE 2006 spring meeting

	Sent/Received	Loss rate
1st time	20472/19808	3.2%
2nd time	39601/35811	9.6%

using the newly attached link.

4.5 Result of the Second Experiment

ICMPv6 echo request/reply statistics was taken from MNN located on the meeting network to the node located at the WIDE K2 NOC. Table 7.4 shows the result.

The transition graphs of sequence numbers of the TCP stream were also drawn. Figure 7.8 shows the transition of the sequence numbers when the MR was not moving. The X-axis indicated the elapsed time from the beginning of the TCP stream in seconds and the Y-axis shows the relative sequence number of the TCP stream. When the MR was not moving, the sequence numbers drew a clear graph.

Figure 7.9 shows the transition of sequence numbers when the MR was moving between two T1 links. There are small gaps when the MR moved. Figure 7.10 is the zoomed image of the first movement in figure 7.9. The TCP stream was suspended for about four seconds during handover. The similar suspension occurred on each handover.

Figure 7.11 is the result of a TCP stream when the MR was moving among three links (two T1 links and the Satellite link). It is able to see that the throughput of the TCP stream was lower when the MR was using the Satellite link than when the MR was using either of the T1 links. There are two large gaps in the graph. The first gap is at the handover time when the MR moved from the Satellite link to the T1 (T1-a) link. The second gap is when the MR moved from the T1-a link to the T1-b link.

5. Comparison and Consideration

We provided a mobile network to a large scale realistic meeting network that had over 250 people in it. The MR moved between two or three different networks without any serious problems and we could provide a moving network to the participants of the meetings. In the first experiment held at the WIDE 2005 autumn camp meeting, we saw a serious service disruption problem during handover. As the result of ICMPv6 echo request/reply messages shows in table 7.2, there was 31.7% packet loss from the nodes inside the meeting network to the nodes located outside of the meeting network.

In the second experiment held at the WIDE 2006 spring meeting, the multiple network interfaces at the MR was introduced and the Multiple CoAs Registration mechanism was utilized. By using the Multiple CoAs mechanism, the registration procedure could be performed for the new CoA which the MR got on the new foreign network before it left the old foreign network. The MR could safely disconnect from the old foreign network after all traffic was routed to the bi-directional tunnel established on the new foreign network. The ICMPv6 echo request/reply result shown in table 7.4 indicates that the mechanism is useful to reduce the packet loss rate. In the second experiment, we only saw 3.2% to 9.6% packet loss from the nodes inside the meeting network. However, considering the fact that the MR could use the old bi-directional tunnel until it established a new bi-directional tunnel, the packet loss rate could be smaller than the result.

At the second experiment, the TCP performance was measured by recording the transition of the sequence numbers of TCP stream. The total performance of TCP sessions was basically good. The result can be confirmed by comparing figure 7.8 which depicts the performance of a TCP stream when the MR was not moving and figure 7.9 which depicts the performance when the MR was moving between two T1 links. However, when looking at the performance during handover in detail, it is possible to see that the transmission was suspended for about 4 seconds. Assuming that there were 4 seconds suspension for every 60 seconds, which was the movement interval in the second experiment, four ICMPv6 reply packets were lost while sending 60 ICMPv6 echo request packets. The loss rate will be $(4/60) \times 100 = 6.7\%$. The reason of the packet loss shown in table 7.4 is thought to be considered related to the reason of the result of the

Table 7.5. The result of how frequently people felt the network movement

Frequency	Number of answers
Did not notice	102
Every 10 minutes	6
Every 5 minutes	3
Every 2 minutes or more	3

TCP performance.

At the second meetings, we asked participants of the camp meetings how frequently they noticed the movement during they were attending the meeting. The result is shown in table 7.5. The result shows that 89.5% of people did not notice the network movement.

6. Summary

When considering the deployment of a new protocol, there are two important things. One is to implement the specification to find specification problems which cannot be easily found during designing it. The other thing is to use the implemented protocol stack in a real environment to find any operational issues, to gain experience of protocol operation and to advertise the new protocol to people. These experiments are tries to deploy the mobility technology, especially the network mobility technology based on NEMO BS. The benefit of NEMO BS is that the nodes inside the moving network served by a mobile router are not required to change their protocol stacks to support the moving function. The nodes inside the moving network can be normal IPv6 nodes. The typical application of this technology is to provide the Internet access to transportation systems such as buses and trains.

Two experimental operations of mobile network using NEMO BS were made using the WIDE 2005 autumn camp meeting and the WIDE 2006 spring meeting. These meetings had the actual conference networks used by the meeting attendees. The number of attendees was over 250 people for each meeting, which could be considered as a practical moving entity like a train and its passengers. The success of the experiment means that the technology can be used for the real

transportation systems.

At the first experiment, only the basic functions provided by the base NEMO BS specification were used. The short report of the first experiment was also published as [67]. In the base specification, a mobile router can only use one network interface at one time. It means that when the mobile router moves from one foreign network to another foreign network, the mobile router must have duration of disconnection. Because of this problem, a large packet loss rate (31.7%) from the nodes inside the mobile network provided by the mobile router was observed. In the second experiment, an extension of NEMO BS which enables the concurrent use of multiple network interfaces was used. In the second experiment, the mobile router was equipped three network interfaces. The movement policy used in this experiment is to connect to a new network before leaving from an old network. The result of the packet loss was improved to 3.2% in the best case. Also, the questionnaires asked to the meeting attendees revealed that most of users did not notice the network movement. From the result, it can be concluded that the Multiple Care-of Address Registration mechanism is useful for seamless handover of the mobile network and the mobile network is practically usable as a moving entity.

However, as the results of the ICMPv6 packet loss measurement (table 7.4) and the TCP performance measurement (figure 7.10) show, the packet loss still happens and causes TCP transmission suspension problems even with the Multiple CoAs mechanism. Further research is necessary to find the solution of these problems.

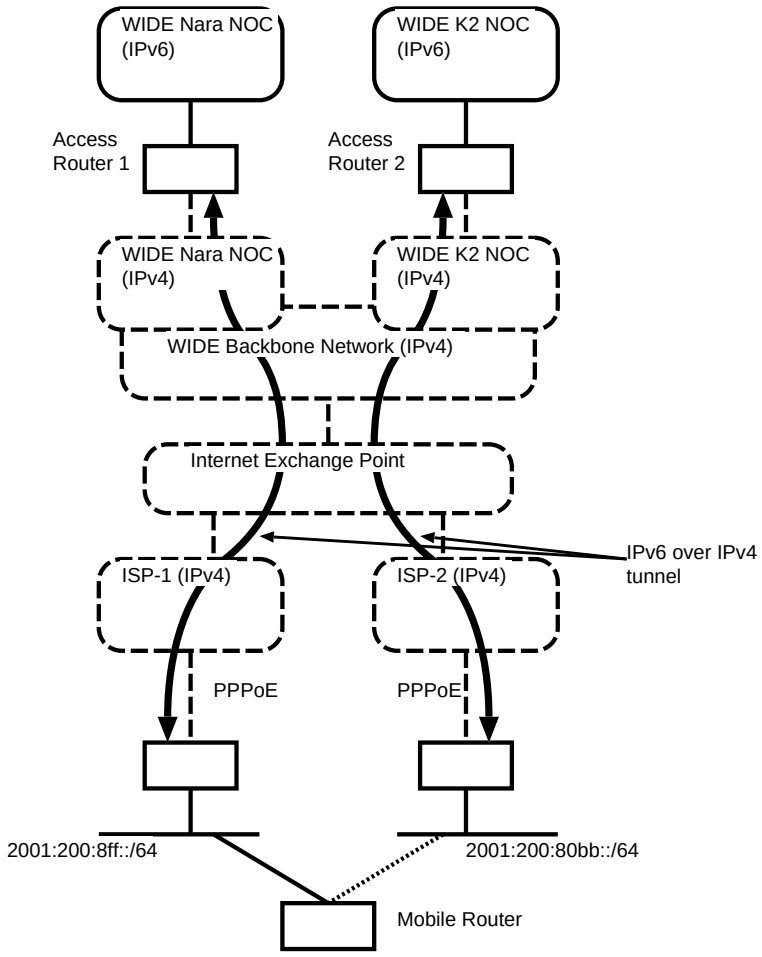


Figure 7.2. The physical network topology of the experiment at the WIDE 2005 autumn camp meeting

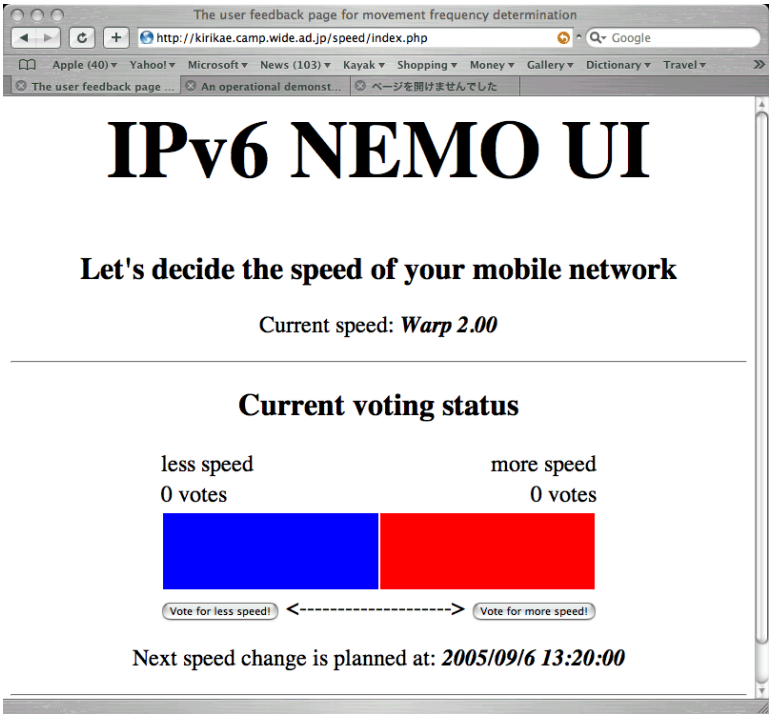


Figure 7.3. The user interface to vote the movement frequency used at the WIDE 2005 autumn camp meeting

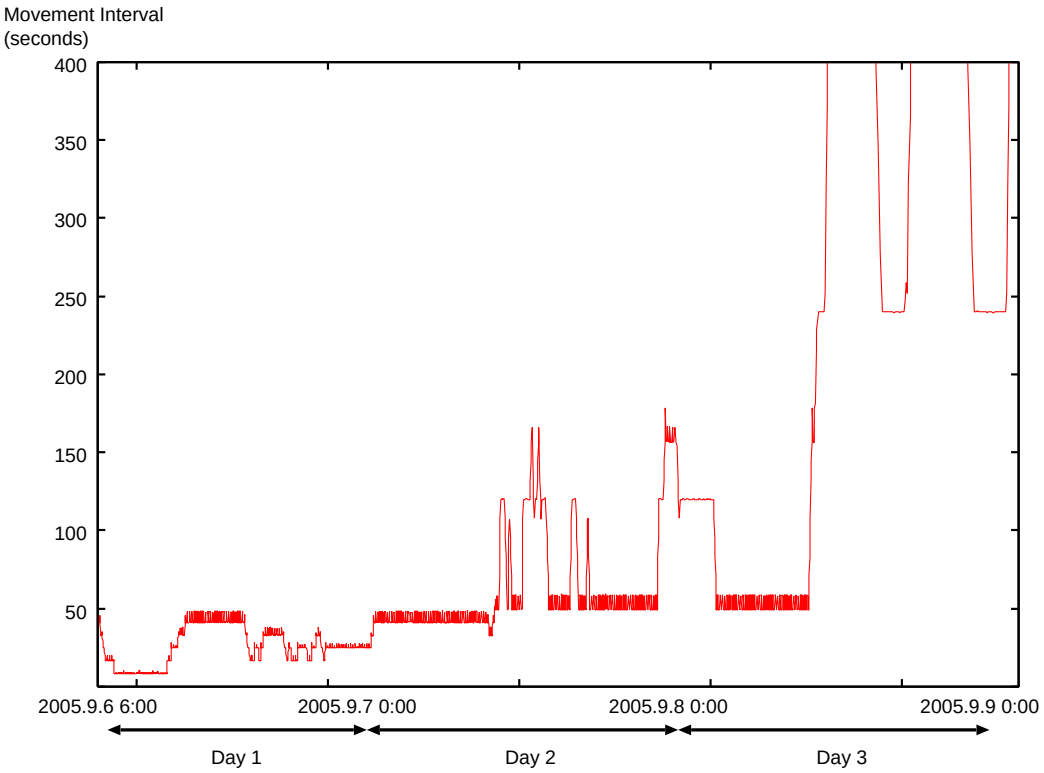


Figure 7.4. Transition of movement frequency at the WIDE 2005 autumn camp meeting

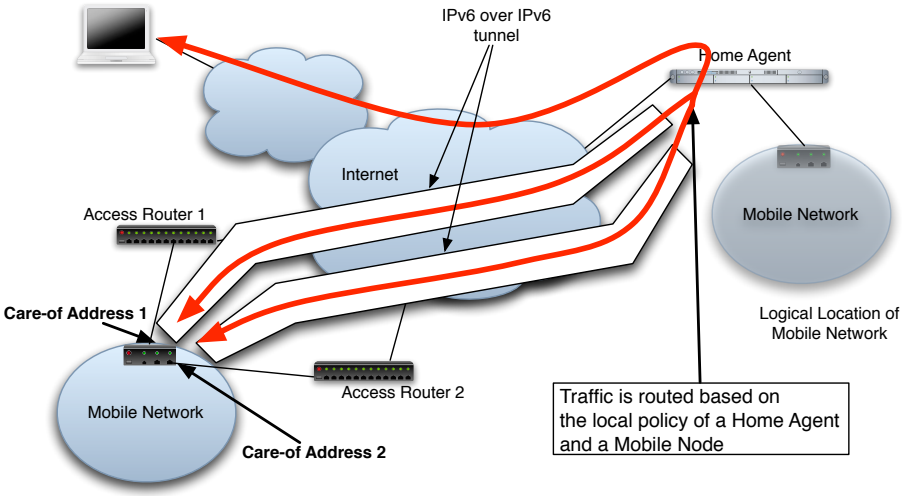


Figure 7.5. Multiple Care-of Address Registration and Traffic Distribution

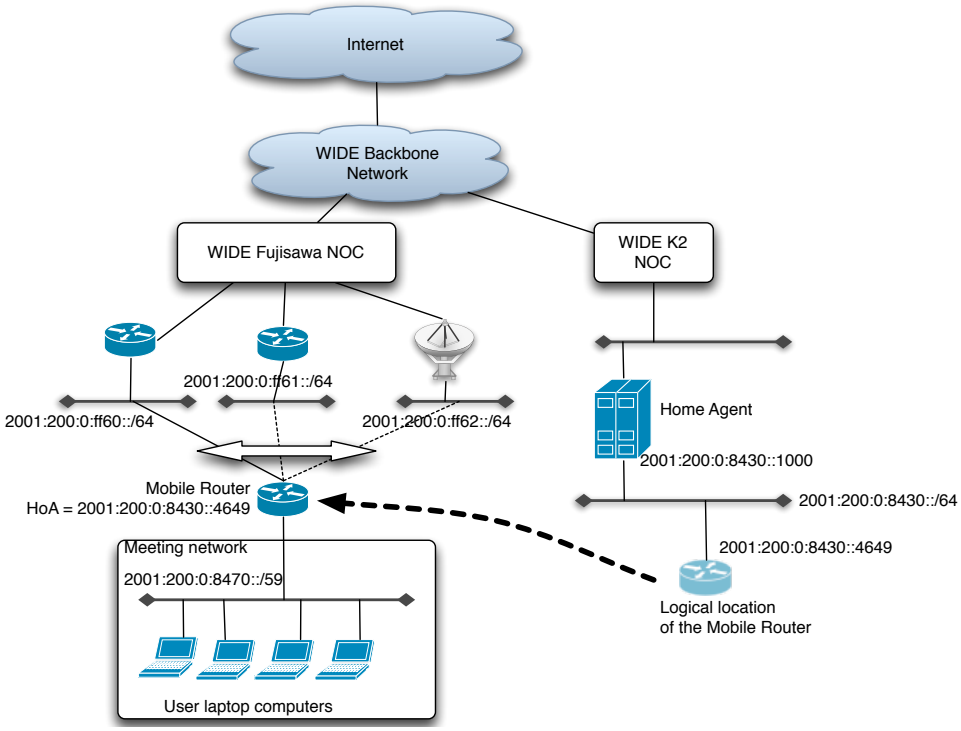


Figure 7.6. The network topology of the experiment at the WIDE 2006 spring meeting

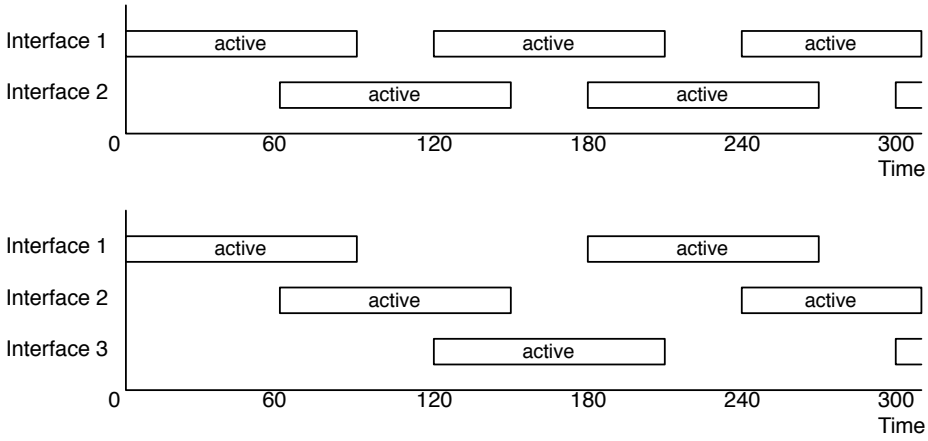


Figure 7.7. Movement Strategy at the WIDE 2006 spring meeting

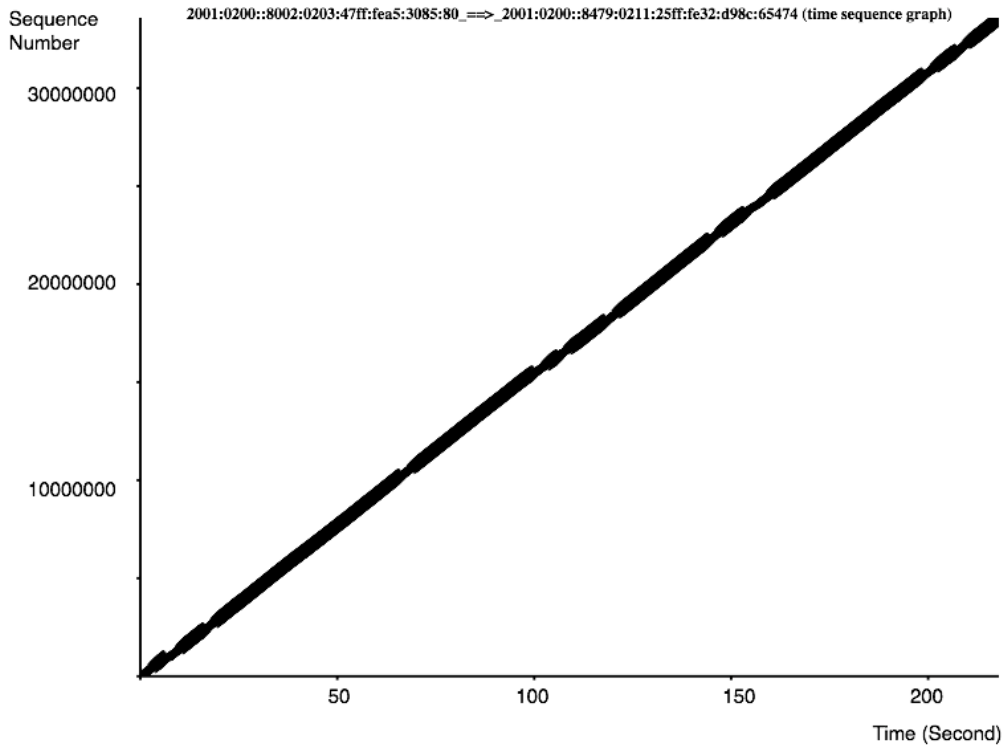


Figure 7.8. Transition of sequence numbers when the MR was not moving

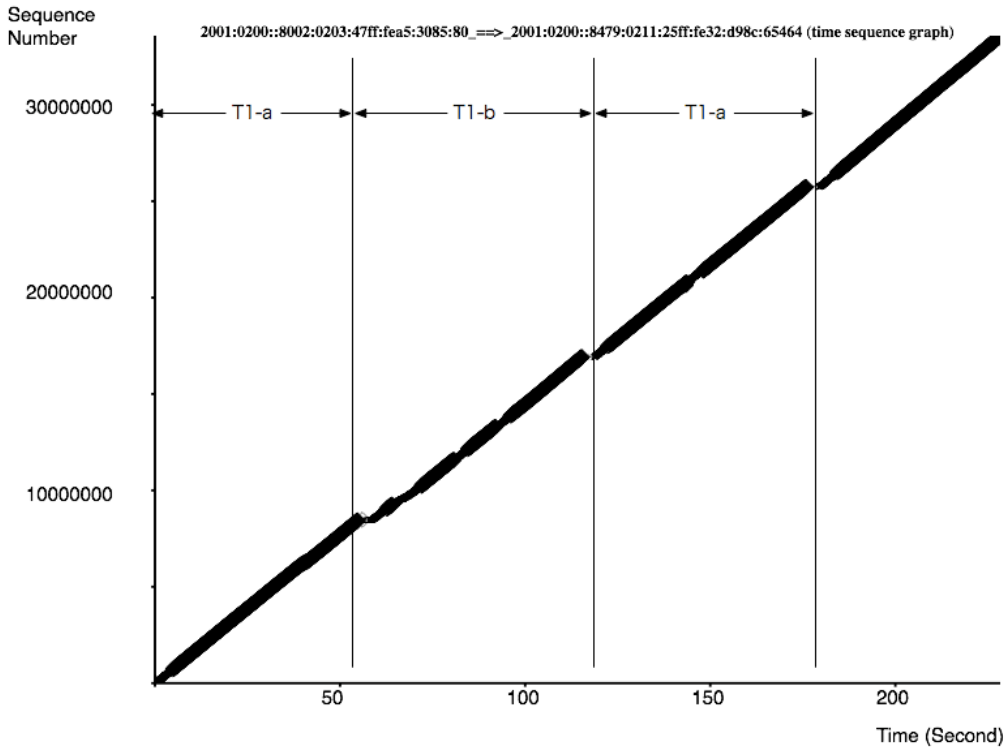


Figure 7.9. Transition of sequence numbers when the MR was moving between the two T1 links

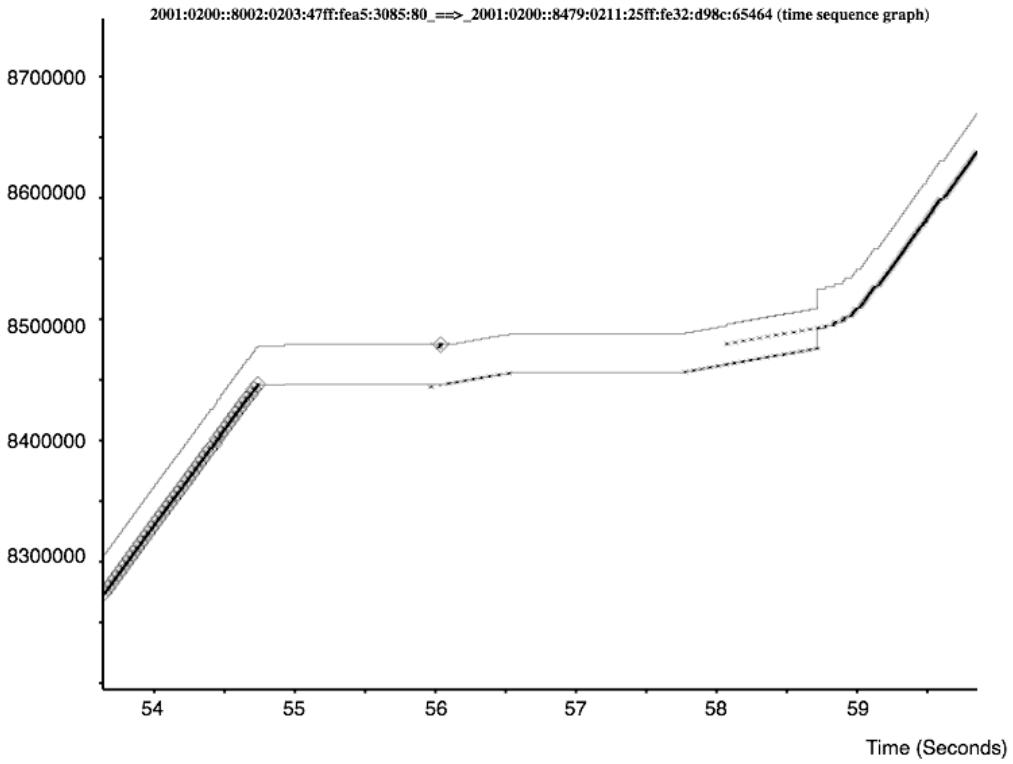


Figure 7.10. Transition of sequence numbers around the first movement in Figure 7.9

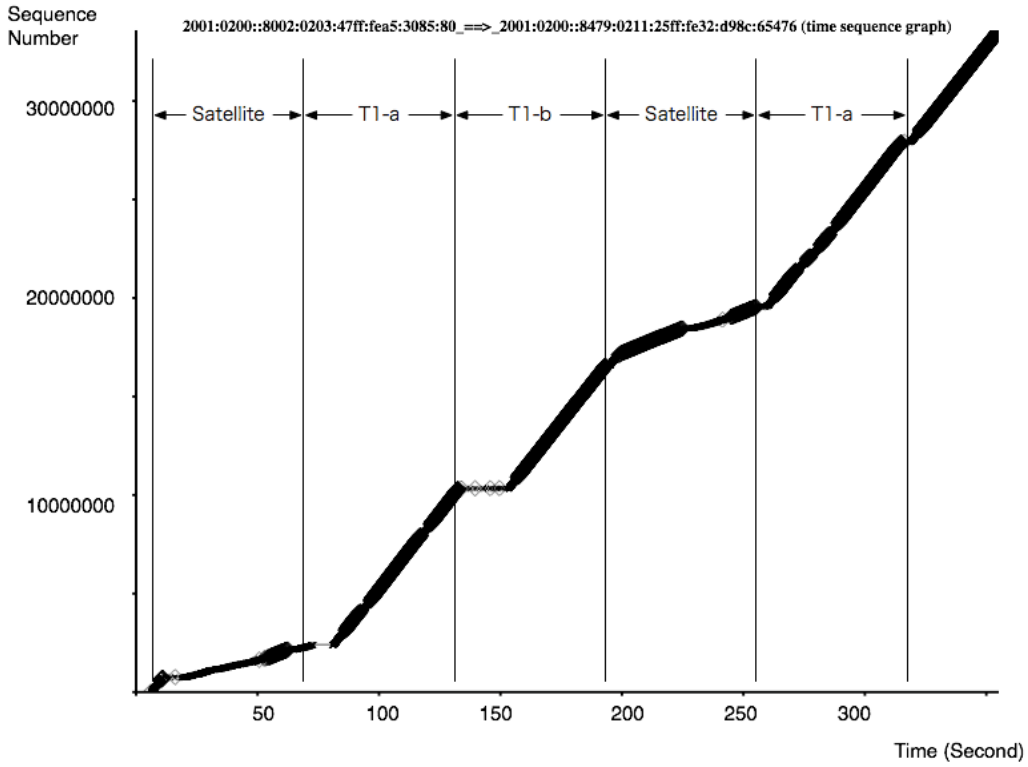


Figure 7.11. Transition of sequence numbers when the MR was moving between the two T1 and the Satellite links

Chapter 8

Global Scale Mobility Operation

Mobile IPv6 does not have a global home agent distribution mechanism in its base specification, which is important when considering global deployment of Mobile IPv6. In this chapter, a global home agent distribution mechanism is proposed based on the anycast routing and bi-directional tunnel mechanism among home agents, and the experiment result performed in the Interop ShowNet network is provided. The experiment result shows the proposed mechanism is feasible and works in a real operation network.

1. Introduction

Mobile IPv6 [30] and NEMO Basic Support [13] are the IETF standard protocols for IPv6 mobility function. Many service operators are considering these protocols or their variant protocols like Proxy Mobile IPv6 [23] for their future mobility services. For the global-scale mobility deployment, a mechanism to distribute home agents is mandatory. With distributed home agents, redundant service host operation which enhances fault tolerant property, load balancing of mobile traffic, and shorten redundant communication path length caused by the Mobile IPv6 tunnel mechanism can be achieved.

As earlier research works, the idea of using the OSPF anycast routing mechanism and binding information synchronization between multiple home agents for home agent redundancy was proposed in [83]. The experiment using the global Internet and simulated mobile nodes was also reported in [85]. A proposal to

apply the technology to the existing Mobile IPv6 based protocols was mentioned in [76].

On top of the previous studies, the home agent distribution mechanism [75], called the Global HAHA mechanism, was designed and implemented. The implementation was tested at the Interop Tokyo 2008 [27]. The Interop Tokyo is an international event of the networking products and services. There were more than 300 exhibitors and about 150,000 visitors in the 2008 exhibition. In the event, the Network Operation Center (NOC) team builds an experimental advanced network called *ShowNet* as the backbone network. The network was connected to several Internet exchange points by more than 120Gbps links in 2008. The Global HAHA experiment was served as a part of the ShowNet.

In this chapter, first the ShowNet overview and the experiment scenario are described, then the system architecture, implementation, and measurement results are explained.

2. ShowNet As An Experimental Network

The ShowNet network is a showcase of the cutting edge technologies of the latest products, and at the same time it is used as a residential network infrastructure for the vendors participated in the event. Figure 8.1 shows the overview of the ShowNet topology . In 2008, the network had 11 exit points, through which it was connected to the Internet. The network was roughly divided into two large segments, *noc.hall2* and *noc.hall4*. In the event, 5 halls (hall 1 to hall 5) were provided to the exhibitors. Exhibitors' segments were extended from noc.hall2 or noc.hall4 depending on the hall number of the their booth. If a booth was located in the hall 1, then its network was extended from noc.hall2, and so on. The servers which provided common services were located in the server segment. For the conferences and tutorials, a separate subnet was extended from noc.hall2 and noc.hall4 to the conference building besides the exhibition halls. The detailed topology can be found in the Interop Tokyo web page [27].

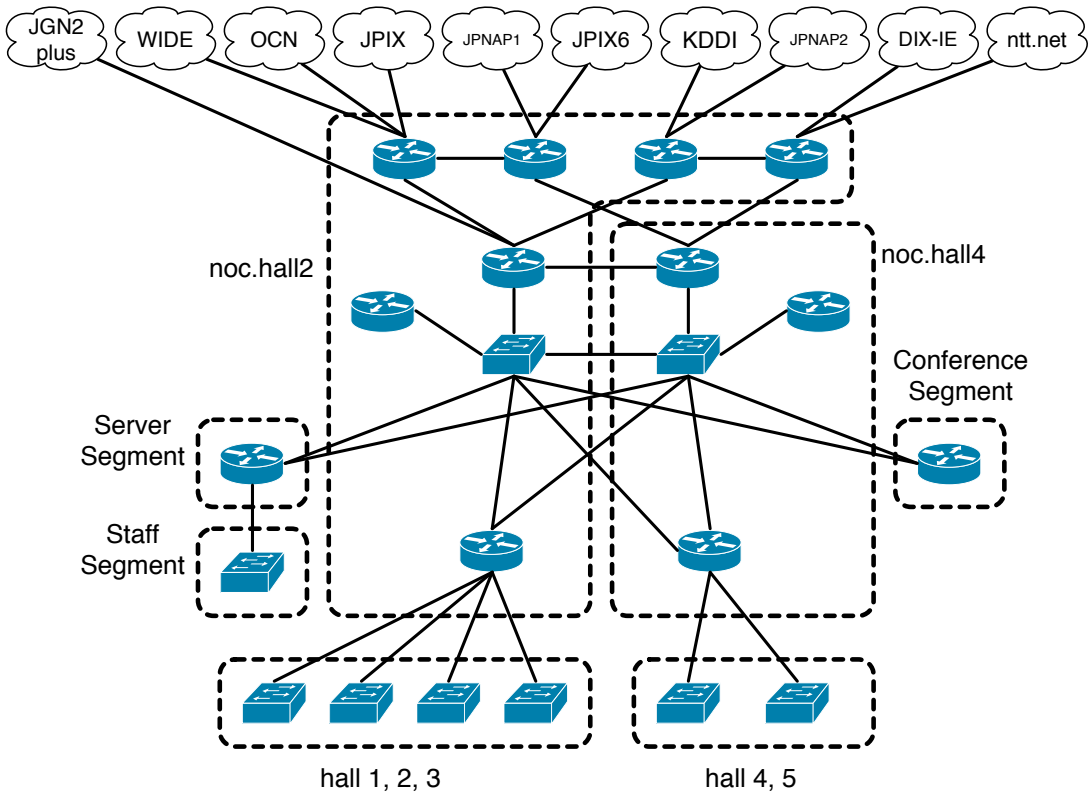


Figure 8.1. The ShowNet topology. The network had 11 exit points to the Internet. The core network was roughly divided into two parts, noc.hall2 and noc.hall4. Server computers were mainly placed in the server segment. The conference building had its own small network segment.

3. Experiment Scenario

During the event, we had an experiment of Global HAHA operation aiming to confirm that the architecture proposed was valid and feasible, through the real operation in the event network. In the ShowNet, two home agents (HA1 and HA2) were set up. Figure 8.2 shows the location of each home agent. HA1 was located in the server segment. HA2 was located in the hall 4. The segment to which HA2 attached was extended from noc.hall4 similarly to the exhibitors' segments in the hall 4 and 5.

The IPv6 network prefix allocated for the Interop Tokyo event was `2001:3e8::/32`. `2001:3e8:ff55::/48` was used as an extended home network prefix for the ex-

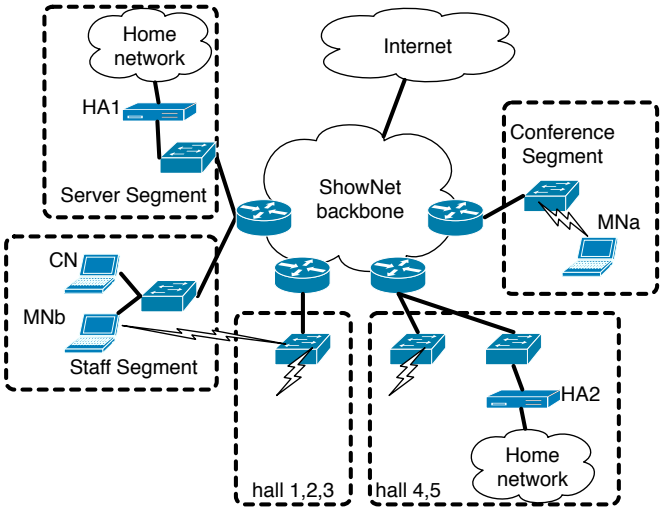


Figure 8.2. The location of home agents

periment. The routing information of the extended home network was advertised from the upstream routers of both home agents. Since the same routing information was advertised from different routers in different sites, the traffic addressed to the extended home network went to the nearest network because of the routing information. In the experiment, the home agent were not involved in the routing management, because they sometimes reboot for maintenance and code updates. The upstream routers advertise the extended home network prefix used in the experiment on behalf of home agents for the stable routing advertisements to the ShowNet.

The home network was `2001:3e8:ff55::/64`, taken from the extended home network. Both home agents were configured with the same home network information.

The mobile nodes had their own home addresses derived from the `2001:3e8:ff55::/64` address block. In the exhibition halls, we could access three wireless networks. The first one was the wireless network served in the halls 1 to 3 (and staff room), called *shownet123* network. The second one was served in the halls 4 and 5, called *shownet45* network. The last one was served in the conference building, called *shownet* network. When a mobile node attaches to one of these networks, the registration request will arrive at the nearest home agent. Several mobile nodes moving between these networks were operated to verify if the Global HAHA mech-

anism was working and to measure the round trip time (RTT) and throughput between nodes using the Global HAHA mechanism.

4. System Architecture

Home agents attach to different networks located in different places. Each home agent has a local IP address on the network interface attached to the local segment, called the care-of address of the home agent. Home agents establish their overlay network with bi-directional tunnels among them, whose endpoint addresses are the care-of addresses of them. In a real deployment, this overlay network can be constructed per mobile service operator depending on their underlying network infrastructure. Operators can interconnect their overlay networks by sharing physical points. This architecture can be a future mobile routing architecture, which consists of connectivity providers as the lower layer and mobility providers as the upper layer.

When a mobile node attaches to a network, it configures its care-of address and registers the care-of address by sending a binding update message to its home agent. Suppose that a mobile node attaches to the hall 1,2,3 wireless segment, and tries to register itself. In the ShowNet network (figure 8.2), the staff segment and the hall 1,2,3 were configured to be near to the server segment (HA1). The conference segment and the hall 4,5 are near to HA2. The binding update message sent from the mobile node which attaches to the hall 1,2,3 is routed to HA1. When HA1 accepts the binding update message, it sends another message to HA2 which indicates HA1 has started serving the mobile node. This notification is done with a *binding migration* message, a newly defined message for this purpose. The message includes the home address of the mobile node. When HA2 receives the message, it then sets up a host route entry for the home address indicating that all the traffic to the home address should be routed to the tunnel interface configured between HA1 and HA2.

If a node near HA2, for example a node in the hall 4,5, tries to send a packet to the home address, the packet from the node is routed to HA2 because HA2 is closer than HA1 from the hall 4,5. HA2 receives the packet addressed to the home address; however, it cannot process it because HA2 is not managing the

binding cache entry for the mobile node. HA2 tunnels the packet to HA1 based on the host route entry installed by the migration message. The packet is finally delivered to the mobile node using a Mobile IPv6 tunnel between HA1 and the mobile node. For the reverse direction, a packet sent from the mobile node is tunneled to HA1 and directly forwarded to the node in the hall 4,5. Different from the opposite direction, the tunnel between HA1 and HA2 is not used in this case.

By combining with the route advertisement mechanism, this system can work as a fault tolerant or traffic engineering (TE) system. If one of the home agents breaks, all the traffic can be routed to other home agents by stopping route advertisement from the broken site. For TE, controlling mobile traffic is possible by stopping route advertisement intentionally. Also, the home agent overlay topology can be designed arbitrary based on the traffic flow requirements.

When the mobile node moves from the hall 1,2,3 to the hall 4,5, the following procedure is performed. The mobile node sends a binding update message to its current (far) home agent HA1, however, because of route information, the binding update message is delivered to the nearer home agent HA2. At this point, HA2 just forward the packet to HA1 through the bi-directional tunnel pre-established between HA2 and HA1. HA1 can then notice that the mobile node is nearer to HA2 than HA1, since the binding update message is delivered from HA2. HA1 sends a *home agent switch* message [24] to the mobile node to suggest that the nearer home agent is HA2. The mobile node then registers its care-of address to the suggested home agent, which is HA2 in this case.

Since the traffic to/from mobile nodes is distributed based on the route information, we can naturally distribute mobile traffic reducing congestion on a single home network.

5. Implementation

NetBSD-current (2008-1-28 version) was used for home agents. The additional code was implemented on top of the SHISA package [88]. Two functions were added to support the Global HAHA protocol described in section 4.

The first function is a sending mechanism of the home agent switch message.

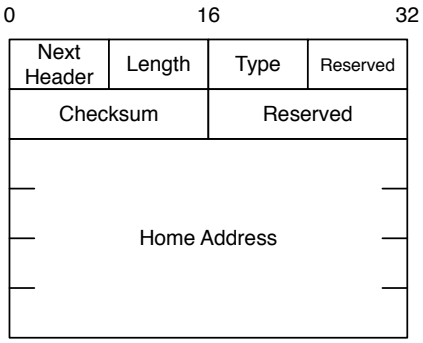


Figure 8.3. The binding migration message format. This message includes the home address of a mobile node.

The bi-directional tunnel between home agents uses the gif generic tunnel mechanism provided by the NetBSD operating system. As described in the previous section, when a binding update message reaches the nearer home agent and is forwarded from the home agent to the farther (currently registered) home agent, the binding update message is transmitted in the gif tunnel. When a home agent receives a binding update message from a gif interface, it knows that its mobile node has moved to far site, and sends a home agent switch message.

The second function is sending and receiving mechanisms of the binding migration message. When a home agent receives a binding update message from a mobile node, it has to notify the other home agent that it will start serving the mobile node. Otherwise, the packets for the mobile node delivered to the other home agent will be dropped. A binding migration message is sent to other home agents when a binding update message is successfully processed. The message format is shown in figure 8.3. When a home agent receives the migration message, it creates a host route entry for the home address of the mobile node specified in the migration message to forward the packets addressed to the home address to the home agent serving the mobile node.

For the mobile nodes, the same version of the NetBSD/SHISA system and the Ubuntu Linux system were used. A receiving and processing mechanism of the home agent switch message is added to the systems. When a mobile node receives the message, it updates the current home agent address information stored in its binding update list entry, and re-initiate the registration procedure

to immediately change its home agent.

6. Interaction with Routing Plane

When we were testing the implementation before the event began, we faced a loop problem of a mobile node registration procedure. When we attached a mobile node to the conference segment, it tried to register its address to HA2. Soon, HA2 sent back a home agent switch message indicating HA1 is nearer than HA2. The mobile node then tried to register to HA1, however, HA1 also sent back a switch message indicating HA2. The problem was caused by the routing configuration and an intermediate load balancer box. The routing costs from the conference segment to HA1 and HA2 were equal by chance. Usually the route is fixed to one of the possible paths based on the routing algorithm, however, in our case, the load balancer distributed binding update messages to different paths. As a result, the binding update messages arrived at HA1 and HA2 alternatively, causing the registration loop problem. We solved this problem by installing static route information from the conference segment to HA2, bypassing the load balancer. This accident indicates that our proposed mechanism requires careful configuration of the underling network topology and interaction between the overlay plane and routing plane is mandatory.

7. Mobile Nodes Movement Results

In the experiment, we operated the simplest case, that is, one mobility operator with two home agents distributed different locations in the ShowNet network. For mobile nodes, we operated six NetBSD based mobile nodes[1].

Since this experiment required modifying the participants' operating systems, it was not opened to the public users. Only the NOC members joined the experiment. Basically, the mobile nodes were located at the staff segment (see figure 8.2), and moved to other segments with operators. The exhibition started

[1]We also prepared Linux mobile nodes, however, they were provided only as virtual machine images and their statistics data is not mentioned in this document.

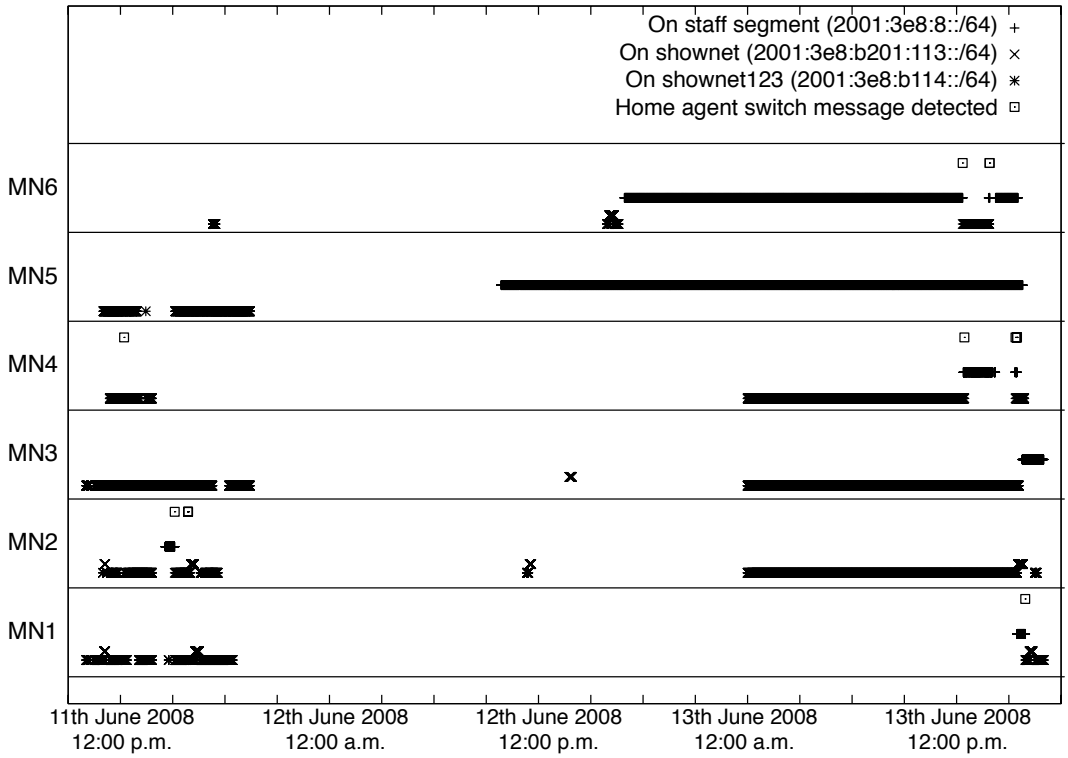


Figure 8.4. The movement timeline of the six mobile nodes. The plus, times and star marks represent the location (attached network) of the mobile node. The square mark means a home agent switch message is sent to the mobile node. The x-axis represents the time, starting from 9:00 a.m. on 11th to 6:00 p.m. on 13th.

at 10:00 a.m. on 11th June 2008 and closed at 5:00 p.m. 13th June 2008. Figure 8.4 shows the movement patterns of mobile nodes during the event.

The graph was drawn based on the log data taken on HA1 and HA2. MN1 to MN6 represent six mobile nodes. From the log data, the mobile nodes seem to attach three different networks during the event. The first network is the staff segment (`2001:3e8:8::/64`). The second network is the shownet123 wireless network (`2001:3e8:b114::/64`) in the halls 1 to 3. The last network is the shownet wireless network (`2001:3e8:b201:113::/64`) in the conference building.

It is possible to see that most of the mobile nodes were attached to and kept staying at the shownet123 on the 1st day. On the 2nd day, most of the mobile nodes could not register its location or were offline. On the final day, all the mobile

nodes started working again. Especially just before the end of the event, some of them were frequently moving between networks. This is because measurement tests were being performed using these nodes. The measurement result will be discussed in section 8.

The square mark drawn in the graph indicates a home agent switch message. From the graph, it can be verified that the home agents sent home agent switch messages when a mobile node moved from one network to another network, which indicates the location detection of the mobile node and suggestion of the nearer home agent worked. Mobile nodes were correctly re-associated to the nearer home agent. However, at the same time, it was also found that the switch message was not sent in many movement cases. This happens when a mobile node took long time to roam from one network to another network. In this case, the binding update list entry expires before attaching to a new foreign network, and the corresponding binding cache entry on the home agent disappears. The registration request sent when the mobile node attached to the new network was treated as a fresh registration procedure. Even in such a case, the mobile node could eventually register to the nearest home agent. Thus, the node distribution and load distribution were still working correctly.

8. Measurement Results

During the experiment, two kinds of traffic measurement were performed; one is the RTT measurement and the other is the throughput measurement.

8.1 RTT Measurement

RTT values between a mobile node (MNa) and a static node (CN) were measured. Figure 8.2 shows the location of the nodes. Two parameters for the measurement were chosen. One is if the packet is fragmented or not. Packet fragmentation sometimes affects the performance of the path. It was thought as important to check if there was any significant difference when the packets were fragmented. The other parameter is if Mobile IPv6 is used or not. If Mobile IPv6 is not used, then the path between the mobile node and static node will be optimal. The purpose is to check the difference in these cases.

Three different packet size values were used in the measurement. 1240 and 1280 bytes were used for the non-fragmented case, and 1548 bytes was used for the fragmented case. Since the MTU size of the bi-directional tunnel between a mobile node and its home agent were configured as 1280 bytes in our implementation, the MTU size must not exceed 1280 bytes when sending/receiving packets from/to mobile nodes to avoid fragmentation. The reason why it was required to use two different sizes for the non-fragmented case is that, when using Mobile IPv6, the size of the payload is reduced by 40 bytes because of the IP-in-IP encapsulating. A 1240 bytes packet was used when Mobile IPv6 was applied, and 1280 bytes was used when two nodes were directly communicating.

MNa sent 100 ICMPv6 echo request packets to CN. In the non-fragment case, MNa sent 1240 bytes packets to CN if MNa was using Mobile IPv6. In this case, the packet was routed to its home agent by the IP-in-IP encapsulation mechanism (during encapsulation, the total size of the packet would become 1280 bytes). When MNa was not using Mobile IPv6, the packet was sent from the care-of address of MNa and routed to CN directory without any intervention of the home agent.

In the fragmented case, MNa sent 1548 bytes packets to the static node. When MNa was using Mobile IPv6, a packet was divided into two packets whose sizes were 1280 and 268 bytes, and each packet was encapsulated to the home agent. If MNa was not using Mobile IPv6, the packet would be divided into 1500 and 48 bytes and sent directly to CN.

Table 8.1 shows the result of the measurement. The values are the mean value and standard deviation value calculated from the 100 times ICMPv6 transmission result in each case.

From the result, one apparent fact can be confirmed that fragmented cases take longer RTT values than non-fragmented cases. Any big difference cannot be seen between the MIP cases and non-MIP cases. It means that the Global HAHA architecture does not pose significant drawback as long as the core network's forwarding ability is high enough. Since the modification of the terminal nodes is small as discussed in section 5, it can be said that this architecture suites the backbone operator's use.

Table 8.1. The RTT measurement result. The shownet123 row is the case where MNa was attached to the wireless network provided in the hall 1,2,3. In the shownet row, MNa was attached to the conference hall wireless network.

Size	1240	1280	1548	1548 bytes
MIP used?	MIP	no MIP	MIP	no MIP
fragmented?	no frag	no frag	frag	frag
shownet123	7.62	6.31	10.1	9.51
	(7.16)	(5.80)	(12.2)	(13.4)
shownet	7.17	5.56	8.63	8.64
	(2.47)	(1.92)	(3.41)	(7.09)

upper value: mean
lower value: (standard deviation)
unit: milliseconds

8.2 Throughput Measurement

To compare the throughput difference in various Mobile IPv6 operation cases, throughput measurement was performed using the `netperf` utility. the following three patterns were performed to see the difference.

1. Simple Mobile IPv6 case: One mobile node is located in a foreign wireless network and one static node is located at the wired staff segment. (Middle path length case)

2. HAHA tunnel case: Tow mobile nodes are located in two different foreign networks (one wireless, the other wired) and each mobile node registers different home agents. (Longest path case)

3. RO case: Two mobile nodes are located as same as the case 2 and use the route optimization mechanism between them. (Shortest path case)

Table 8.2 shows the result of the above three patterns. In this case, the mobile node (MNa) was located at the conference hall network and the static node (CN) was located at the staff segment (see figure 8.2). Throughput measurement was performed five times for each case. The values in the table are the mean value and the standard deviation value of the five results.

Table 8.2. Throughput measurement result from the Conference building (using 802.11a) to the staff room wired segment

	MNa-CN	MNa-MNb	MNa-MNb w/ RO
TCP	5.24	5.72	0.884
	(1.09)	(1.12)	(0.900)
UDP	15.4	16.2	11.3
	(1.06)	(0.471)	(4.98)

upper value: mean
lower value: (standard deviation)
unit: Mbits/s

Table 8.3. Throughput measurement result from the staff wireless segment (using 802.11b) to the staff room wired segment.

	MNa-CN	MNa-MNb	MNa-MNb w/ RO
TCP	4.32	4.03	4.13
	(0.0840)	(0.211)	(0.199)
UDP	6.39	6.14	6.14
	(0.129)	(0.384)	(0.789)

upper value: mean
lower value: (standard deviation)
unit: Mbits/s

There are not big differences in these cases. Apparently, the top-right case (TCP w/ RO) is abnormal, maybe congestion was happening during the measurement period. It looks the UDP w/ RO case (bottom-right) gave worse value, however, as one can see from its standard deviation, measured values were not stable in this case. The max value of the case was actually 14.35, which is not bad compared to other cases.

The same measurement procedures were performed again by changing the location of MNa from the conference hall to the staff wireless segment. Table 8.3 shows the result. In this case, no big difference can be seen either. MN-MN cases and RO cases are slightly worse, maybe because of tunnel header processing and RO related processing overhead on the terminal nodes. As same as the RTT case, the difference came from the terminal node's processing ability. As long as the

core network is fast enough, the Global HAHA architecture does not pose a big drawback.

9. Summary

The global-scale mobility service deployment needs a new mobile routing architecture which covers the earth-scale mobile node distribution. In this chapter, one solution is proposed for the new operation architecture which consists of the anycast routing and tunnel-based home agent overlay network. The architecture provides service redundancy, load distribution property and possibility to future TE operation. This architecture can be extended to the world-scale interconnection mechanism of mobile service operators' networks. Similar to the current Internet, which is operated over the various L2 networks, the future mobile Internet will be operated over the various Layer 3 network services including static Internet.

The implementation was developed on top of the proposed mobility platform, and it was operated in the ShowNet, the one of the biggest live event networks in Japan. The finding is that the architecture can be implemented with small modification to the existing Mobile IPv6 architecture. It was also confirmed that the mechanism is feasible by actually providing the real implementation and operating in a real network. It was impossible to find any significant drawback with this mechanism from the measurement result. However it was also impossible to find any performance merit from the measurement result. Further experiments with a larger scale network are necessary to evaluate the idea.

The experiment reveals that the combination of the mechanism depending on routing infrastructure and an intelligent under layer box may cause a routing problem. This is a good finding for the future protocol design and operation network design.

The future work is to extend the operation to multiple mobile operators and operate the system on the IPv6 Internet using commercial networks in global-scale.

Chapter 9

Activity towards the Deployment of Mobility Technology

Just providing software or the protocol stack implementation is not usually enough to help quick deployment of the new technology provided by such a software. As the previous study shows, deployment of a new technology is accelerated if supportive activities exist. For a new protocol, the following two points are important for the deployment.

1. The real environment where the protocol is used/the users can interact with the protocol
2. Bundling the stack with the existing operating systems

In the following sections, the activities focused on the above points are described.

1. Public Mobile IPv6 Service Operation

One of the big problems with which a newly invented protocol faces is that there is no real environment in which the new protocol can be used before its deployment. Of course, during the development process, most of the protocol developers prepare their own small testbed for testing. There are also several interoperability testing events where many independent developers of the same new protocol

gather to verify the specification and interoperability of their protocol stacks. However, these environments are still specially crafted ones just for testing. Providing a real environment and operating the stacks in the real environment will encourage people to use the new service, and may reveal more problems than in the testbeds, for example, operation issues, interaction issues with the existing technologies, scale issues, and so on.

To achieve the goal to provide such an environment, a public Mobile IPv6 service operation was started in December 2006 by our research team.

1.1 Service Image

The service operation intended to become a simple Mobile IPv6 service for users, and at the same time, a testbed for the Mobile IPv6 implementation being developed. Also, it has been used for the testbed for the development of the service console application.

The service core consists of the following components.

- The base operating system
- The Mobile IPv6 stack to provide the home agent function
- The console application for the service administrator
- Live CD creation service

The base operating system can be BSD operating systems, where we developed as a result of this dissertation, or can be the Linux operating system. In the latter case, the Mobile IPv6 stack for Linux [89, 22] is used as the mobility stack. The service can be provided with these above two components. However, to attract more users and to reduce the operation complexity, the third component, the console application is also prepared.

The forth component is intended to accelerate the service deployment. Currently, no operating system supports Mobile IPv6 client service officially. It means that the users of this public service must prepare their Mobile IPv6 by themselves. The process includes the upgrade of the operating system kernel, in some cases patching source code and rebuilding the kernel. This kind of work is not difficult,

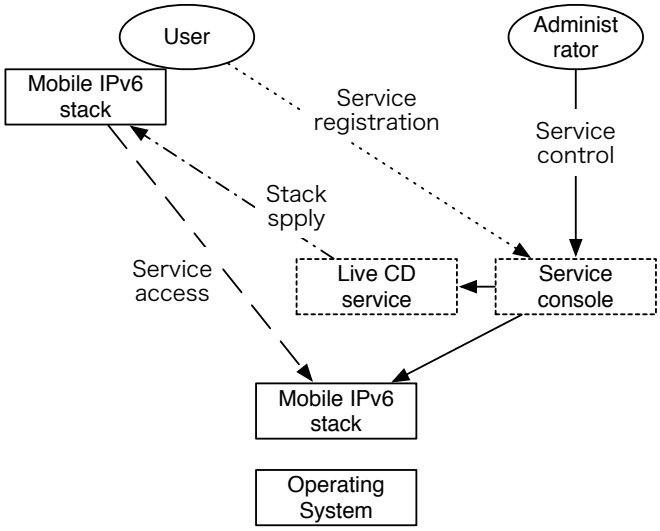


Figure 9.1. The relationship between operation service components and its players

but is something what users do not want to do. The Live CD service is a remedy for this problem. Once a user registers the service account, he can download a CD image which can be used immediately.

Figure 9.1 shows the relationship between components and players. The minimum components to operate Mobile IPv6 are the boxes with solid lines. The dotted boxes are additional software components to enhance user experience. The administrator can control the mobility service through the service console application. The user also use the service console application to register their account to the service and retrieve Mobile IPv6-related information such as home addresses and IPsec security parameter information. Users who do not have Mobile IPv6 stack implementation can use Live CD service through the service console application to generate their own CD image. With the Mobile IPv6 stack, which is either the user installed to their computer or booted from the Live CD, the user can access the mobility service.

Figure 9.2 and figure 9.3 show the interface screens of mobile node management for service users, and of home agent management for service administrators. The implementation detail are not discussed in this dissertation, since it is out of scope of this discussion. The reports are available as published papers as [19, 3].

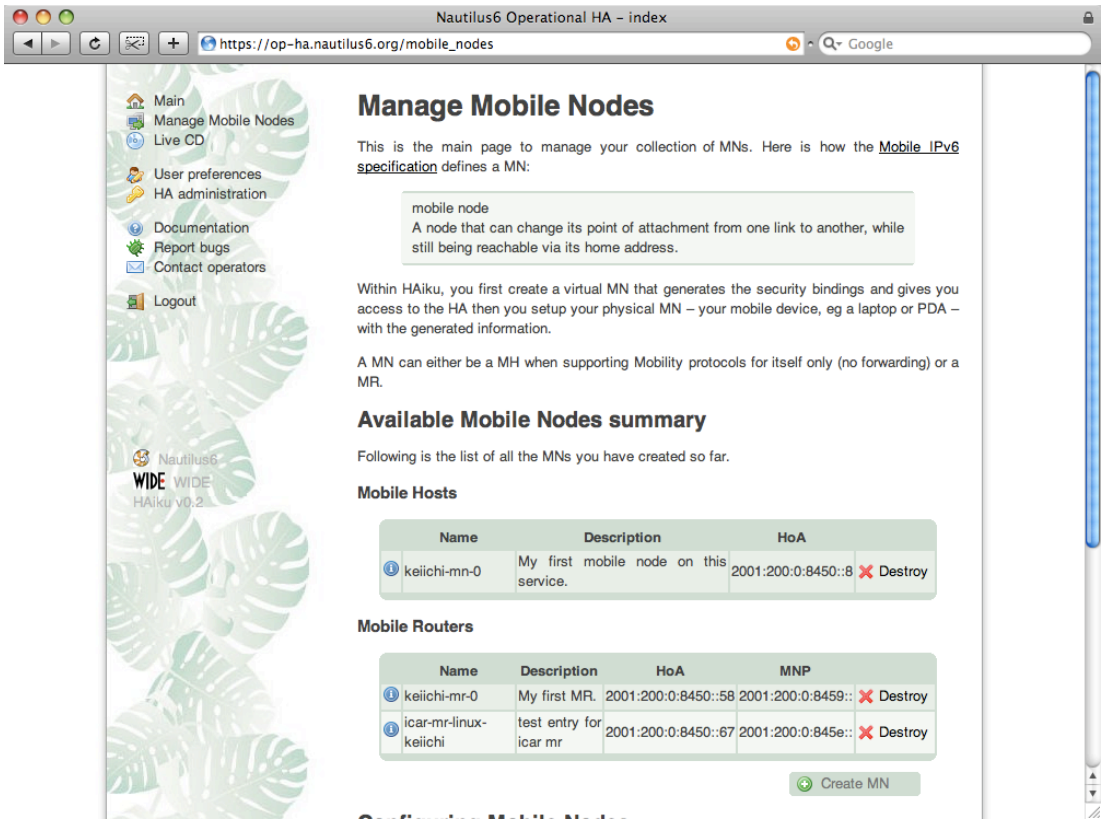


Figure 9.2. The management interface screen for service users to manage their mobile nodes.

1.2 Operation Network Topology

The service system is located in the Keio University K2 town campus, where one of the WIDE [87] network operation centers (WIDE K2 NOC) co-locates.

Figure 9.4 shows the topology graph of the service network. Two home agents (one for the service, the other for testing and development) are attached to one of the network (`2001:200:0:8400::/64`) allocated at the WIDE K2 NOC. As the home network for mobile nodes, `2001:200:0:8450::/64` is assigned to the network behind the home agent. The cloud drawn under the home network in figure 9.4 is the network prefix pool used by mobile routers registered through the public Mobile IPv6 operation service. In the beginning of the service operation, only the mobile node service is provided. The service is now supporting both the mobile node and mobile router services.

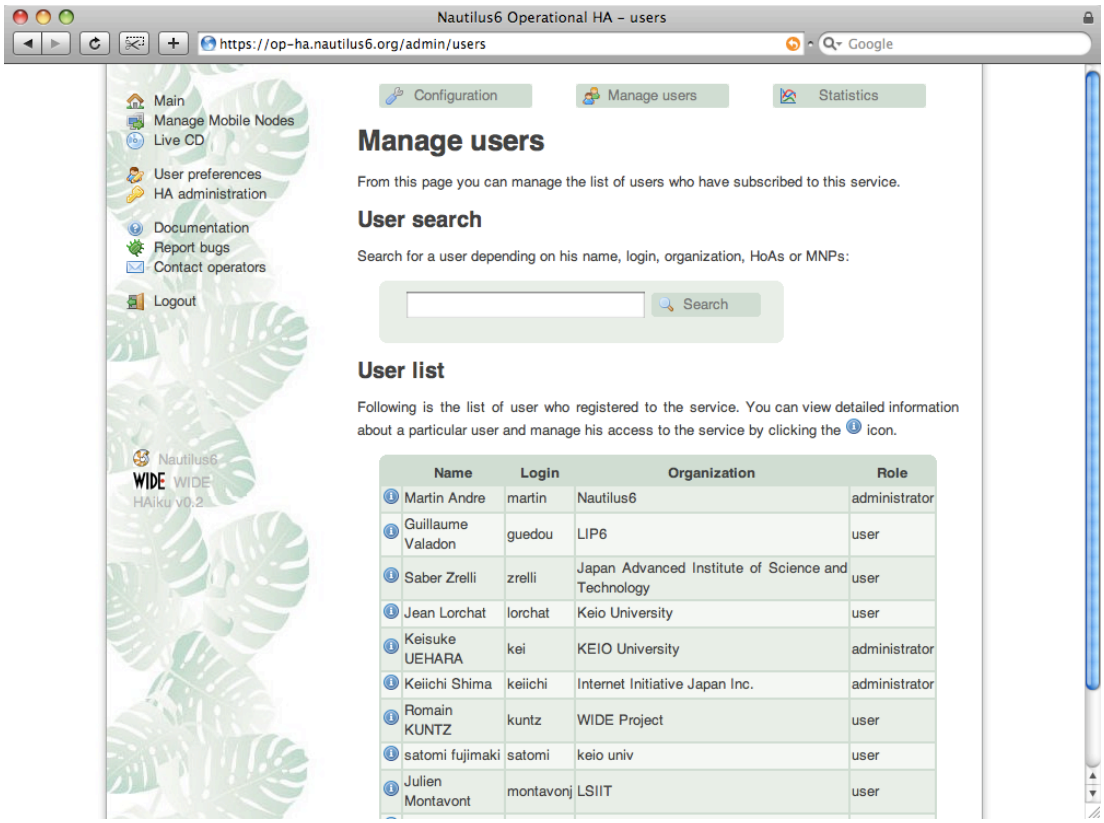


Figure 9.3. The management interface screen for service administrators to manage home agent.

1.3 Output From the Service

The service started in late 2006 and is in operation at the time of this writing, 2009. Through this activity, over 80 people have registered to the service. However, most of the people just made service registration and do not use the service everyday. One of the reasons why people do not/cannot use the service is that the service is available only for users who have IPv6 access. In IETF, the dual stack solution to solve the scarce IPv6 deployment problem is being discussed. The future service design should take into account supporting the dual stack feature.

As a result of the activity, several software products were developed and published as open source softwares. The following is the list of the freely available softwares which constitute the service.

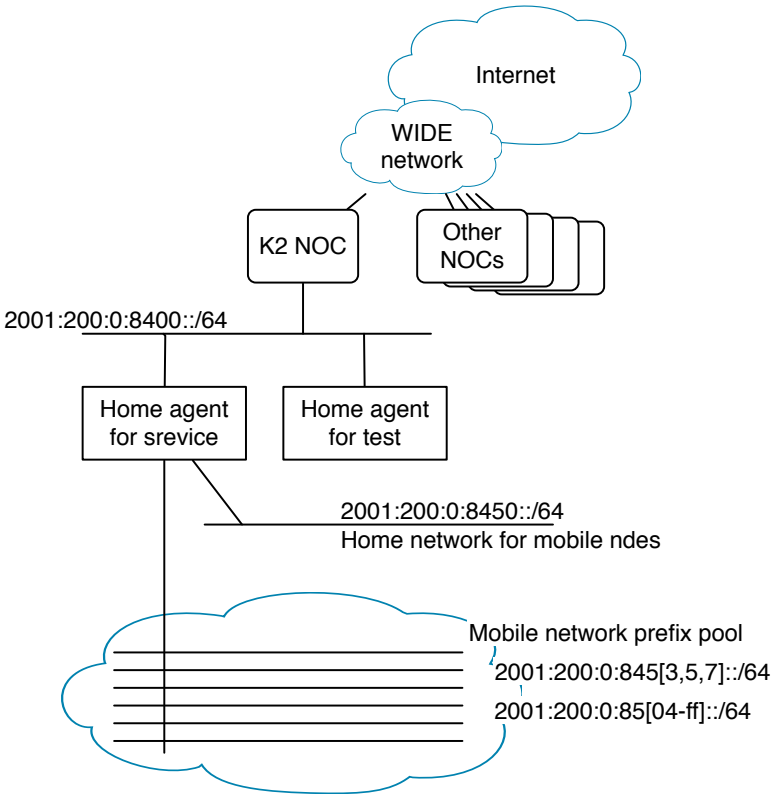


Figure 9.4. The topology of the Mobile IPv6 public service operation

- The service console application 'HAiku'
  - Available from http://software.nautilus6.org/HAiku/index.php
- The Live CD software 'Homeguy'
  - Available from http://software.nautilus6.org/homeguy/index.php

2. Integration to the NetBSD

Currently, people who want to use or operate the mobility service have to retrieve both the base operating system and related patches to add the mobility stack. This work is sometimes hard especially for those who do not have deep knowledge in operating system kernel. Since the goal of this activity is to provide the future mobile research/development infrastructure, preparing an easy installation

environment for developers and researchers is one of the important tasks in this activity. In this section, the plan and current status of the integration work of the SHISA mobility stack to the NetBSD operating system is reported.

2.1 Integration Strategy

SHISA was originally developed on top of the KAME IPv6 stack [29] for NetBSD 2.0 and FreeBSD 5.4. I ported SHISA to the NetBSD-current tree as the first step of porting effort. There are two reasons why NetBSD was chosen as the first platform for the porting work. The first reason is that it supports various kinds of architectures. The mobility functions are useful especially when it is integrated to moving entities such as PDAs, cars or trains, and so on. Such moving entities usually use architectures which run with limited resources. NetBSD supports many architectures which are suitable for embedded use, and the SHISA development members, including I, wanted to realize such small devices using their code. The other reason is the difference between the KAME tree (based on NetBSD 2.0) and the latest NetBSD is relatively small compared to other BSD variants which KAME supported. This makes it easier to port the SHISA code from KAME to NetBSD-current.

Since it is difficult to integrate a new protocol to the existing source tree directly, some kind of verification and approval phase are necessary. To do that, some of the SHISA developers, including I, became the NetBSD developers and started the integration work. The following is the plan for the integration.

1. Separate Mobile IPv6 part and NEMO BS part of the SHISA code.
2. Create a test branch on the NetBSD source code repository.
3. Import the Mobile IPv6 code to the test branch.
4. Verify the code and get approval from developers.
5. Import the Mobile IPv6 code from the test branch to the main branch.
6. Import the NEMO BS code to the test branch.
7. Do the same procedures as for the NEMO BS case.

As discussed in chapter 5, SHISA supports two IETF standard mobility protocols, Mobile IPv6 and NEMO BS. Since the NEMO BS protocol is based on the Mobile IPv6 protocol, they share most of the code, except the mobile network prefix handling code which is meaningful only when mobile router function is concerned. It should be avoided to import the entire code at once, because if the difference between the new code and old code is small, the required discussion for the integration will be small too. The first work is to separate the NEMO BS only code from the entire SHISA code.

The second work is to make a working branch for this integration task. The integration of the Mobile IPv6 function modifies the IP processing function of the kernel, it is necessary to check the integration does not cause any side effect. After creating the test branch, the integrated code will be put on the branch.

Once reviewing of the function in the branch completes, the code will be ported to the main branch. After this, the same procedure should be taken for the NEMO BS function.

2.2 Current Status and Availability

We have already done splitting Mobile IPv6 and NEMO BS code in the SHISA source code, and completed the integration of the Mobile IPv6 code to the test branch. Since the discussion with other developers has not completed yet, the code in the test branch has not been imported to the main branch.

Even though, the code is available to people who want to use the Mobile IPv6 code for NetBSD. The NetBSD operating system source code for the mobility function is put in the *keiichi-mipv6* branch and can be retrieved with the cvs command below.

```
$ cvs -d:pserver:anoncvs@anoncvs.netbsd.org:/cvsroot co \
  -D keiichi-mipv6 src
```

The above cvs command will get the entire *keiichi-mipv6* branck into the 'src' directory.

As discussed in chapter 5, SHISA consists of the kernel part and the user space programs part. The user space programs are available from the SourceForge.net,

the open source project repository. The following cvs command will retrieve the SHISA code.

```
$ cvs -d:pserver:anonymous@shisa.cvs.sourceforge.net:/cvsroot/shisa co \
  -P shisa
```

The integration work is still ongoing. It is necessary to continue the discussion with other NetBSD developers to make the mobility code as a standard function of the NetBSD distribution.

Chapter 10

Mobile Communication in the Disaster Environment

This chapter considers the next step of the future mobility environment. Up to the previous chapter, an infrastructure-based mobility technology, based on Mobile IPv6 and NEMO BS, was discussed. That technology is based on the IPv6, which belongs to the traditional IPv4 school, and there is no doubt that the current IP-based technology will be used as a main information infrastructure for many years from now. However, it is considered that such an infrastructure-based technology cannot solve all the problems seen now and will be seen in the future.

As a final discussion of this dissertation, one of the special environments which require mobility technology that is not based on the infrastructure-based technology is picked up. The situation focused in this chapter is a disaster environment. When a disaster happens in an area where people live and there are victims at there, a rescue team is organized and sent to save the victims. Traditionally, the rescue parties run a risk of their own lives to save them. Recent progress of robotics technologies and networking technologies can help the situation. Thus the idea of the autonomous network construction system and the remote investigation system of the disaster area by robots is proposed in this chapter. A new research area for the dynamically extended and autonomously maintained network by robots is proposed, and I with colleagues define the disaster situation assumed in this research area and state the requirements to realize the solution

for the new kind of network.

1. Introduction

The primary purpose of rescue activities in the area stricken by a disaster is to save victims left behind or unable to evacuate because of collapsed houses or fire caused by the disaster. Since such a stricken area is usually in danger, especially just after the disaster, it is necessary to take care not to cause secondary disasters when performing rescue activities. In addition to natural disaster, there are risks of artificial disasters like terrorism recently. In such cases, poisoned gas or bombs may be spread over the area and would be immediate causes of serious injuries on rescuers.

One possible way to avoid the secondary disaster is to send rescue robots instead of human parties. Rescue robots inspect the disaster area and collect information necessary for the later human rescue activities. By using such robots, rescuers can reduce the cost to search areas where no victims exist and can avoid too dangerous areas for human rescuers. Many laboratories are researching such rescue robots [62].

However, operators still have to go to the disaster area with robots because most of them are short range remote controlled robots and operators have to control robots beside or very near from them. Since the current robot control methods are using direct communication devices such as a wired line or a single wireless connection only, operators cannot control robots from a distant place yet.

It is possible to think that combining the robotics technologies and networking technologies we can provide a safer way to do rescue activities. If we have a network which covers whole the disaster stricken area, we can use the network to support robot operation with no need of operators to run a risk to get into the area. This raises one question; how such a network can be created in the stricken area. If the rescue area is outside, then we may be able to utilize wide-area wireless communication technologies to construct such kind of network. However if the disaster occurs indoor such as airports, underground shopping centers or stations, it is difficult to use wide-area communication devices. These areas are

usually complicated in shapes and radio wave cannot cover whole the area because they are divided by walls or obstacles. In such cases, robots can be used not only to inspect disaster areas but also to construct the network for rescue. The search area can be extended by enlarging the network using the network itself and robots.

In this chapter, the requirements for this kind of dynamic self-extendable network using robots are proposed . We named such a type of network as "*Robohoc network*". Section 2 shows the basic idea of how to construct and extend the Robohoc network. Section 3 discusses disaster situations and use cases of the Robohoc network, and then defines requirements for the Robohoc network. Section 4 introduces some related works on this area and section 5 concludes this chapter.

2. Robohoc Network Construction Scenario

The Robohoc network is required because the target area where victims exist may be dangerous for human. Before sending a human rescue team, it is necessary to examine the target area and collect information with robots using the Robohoc network.

With the Robohoc network, operators can initiate the rescue activity in the nearest safer place from the disaster stricken area. Robots and router nodes are launched from this base control point. In this document, we call the router node which constructs a Robohoc network is called as "*Robohoc-Router (RHR)*" and the base control point as "*Point Zero*". Robots are connected to the Robohoc network via wireless communication between a robot and the nearest RHR. Figure 10.1 describes the initial status of the Robohoc network. In the initial state, there are only one RHR (RHR1) and one robot (Robot1). RHR1 is located at the Point Zero and Robot1 can move only inside the wireless communication range of the RHR1.

Robot1 explores unexamined areas within the wireless access range to collect information of the neighbor area. For example, it checks the radio power of the RHR1 or existence of obstacles such as walls and doors. Operators can plan the following strategy of how to extend the Robohoc network efficiently. In some cases, Robot1 may be able to decide autonomously what it should do to extend

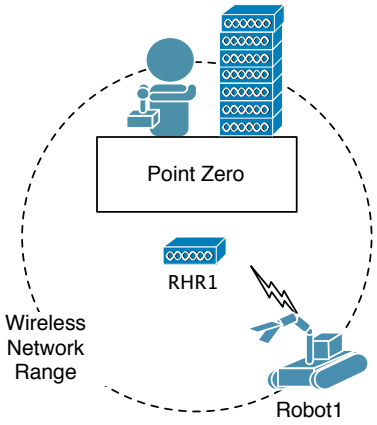


Figure 10.1. The initial Robohoc network topology

the network.

Based on the decision of the operators and robots, the Robohoc network is extended by Robot1. Robot1 puts another RHR (RHR2) at the boundary of the wireless network range of RHR1 (Figure 10.2). RHR1 and RHR2 communicate each other and extend the Robohoc network. At this point, the Robohoc network includes two RHRs (RHR1 and RHR2) and one robot (Robot1).

Robot1 changes its attachment point to the Robohoc network from RHR1 to RHR2 and puts another RHR to extend the Robohoc network and so forth. Figure 10.3 shows the fully expanded Robohoc network which can cover the whole disaster stricken area.

Routing information in the Robohoc network is maintained by the extended IP routing protocols (for example, [11, 40, 8, 53, 7]). Thanks to the nature of the Internet constructed by autonomous systems and maintained with distributed processing methods, it is easy to add a new access point to the existing network and can extend the infrastructure. What is important here is the robot activities also depend on the network which is constructed by the robots themselves.

The network construction scenario is similar to that of the mobile adhoc network (MANET). However the property of the Robohoc network is quite different from that of the MANET network. In the Robohoc network, most of the routers (RHRs) do not move. The main purpose of the Robohoc network is to provide the network infrastructure to the rescue robots. The RHRs are usually placed

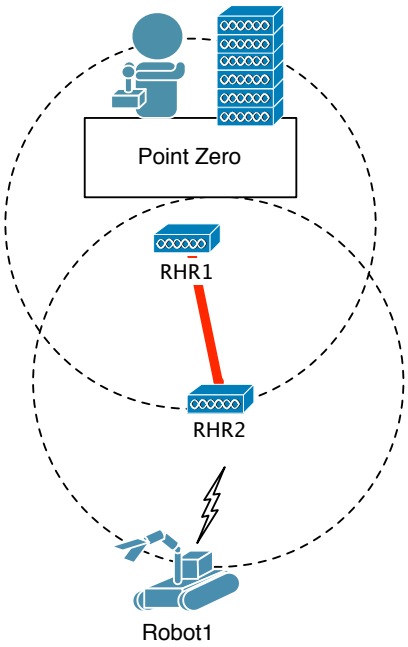


Figure 10.2. The second level rescue network topology

statically. The benefit of this approach is that the connectivity between RHRs is more stable than the MANET case in which the neighboring routers frequently change. With the statically placed routers, stable network delay and jitter can be achieved, the characteristics provides less possibility of topology change events, and more end-to-end bandwidth. Of course, it may be required to relocate RHRs to fix the network, for example, when a secondary disaster breaks some of core RHRs (see section 3.3, 3.6 and 3.10). To make the Robohoc network robust, it is necessary to prepare some moving RHRs in the Robohoc network. The proportion of the number of RHRs which can move out of all RHRs depends on the possibility of the network failure. More practice is necessary on how to define the proper value.

The number of robots is not restricted in this mechanism. Operators can launch any number of robots to extend a Robohoc network quickly and to search victims in a wider area. Figure 10.4 shows an example topology for wider operation. In figure 10.4, there are four concurrently operated robots. Two of them are extending the Robohoc network to reach frontier areas and the rest are looking

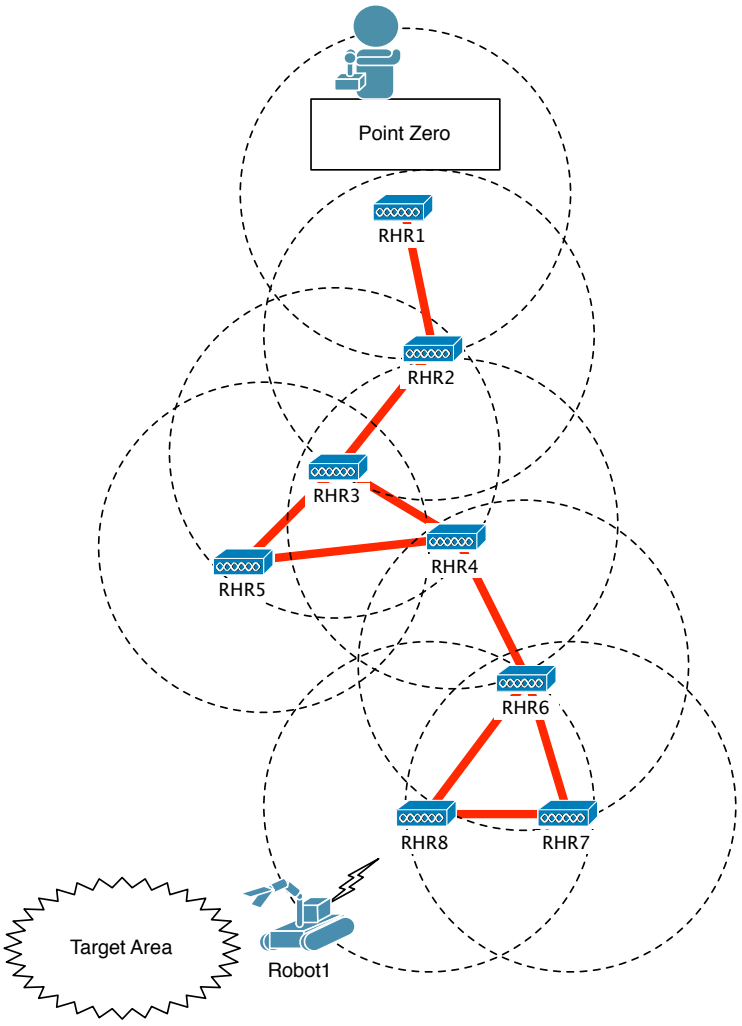


Figure 10.3. The fully expanded Robohoc network topology of this scenario

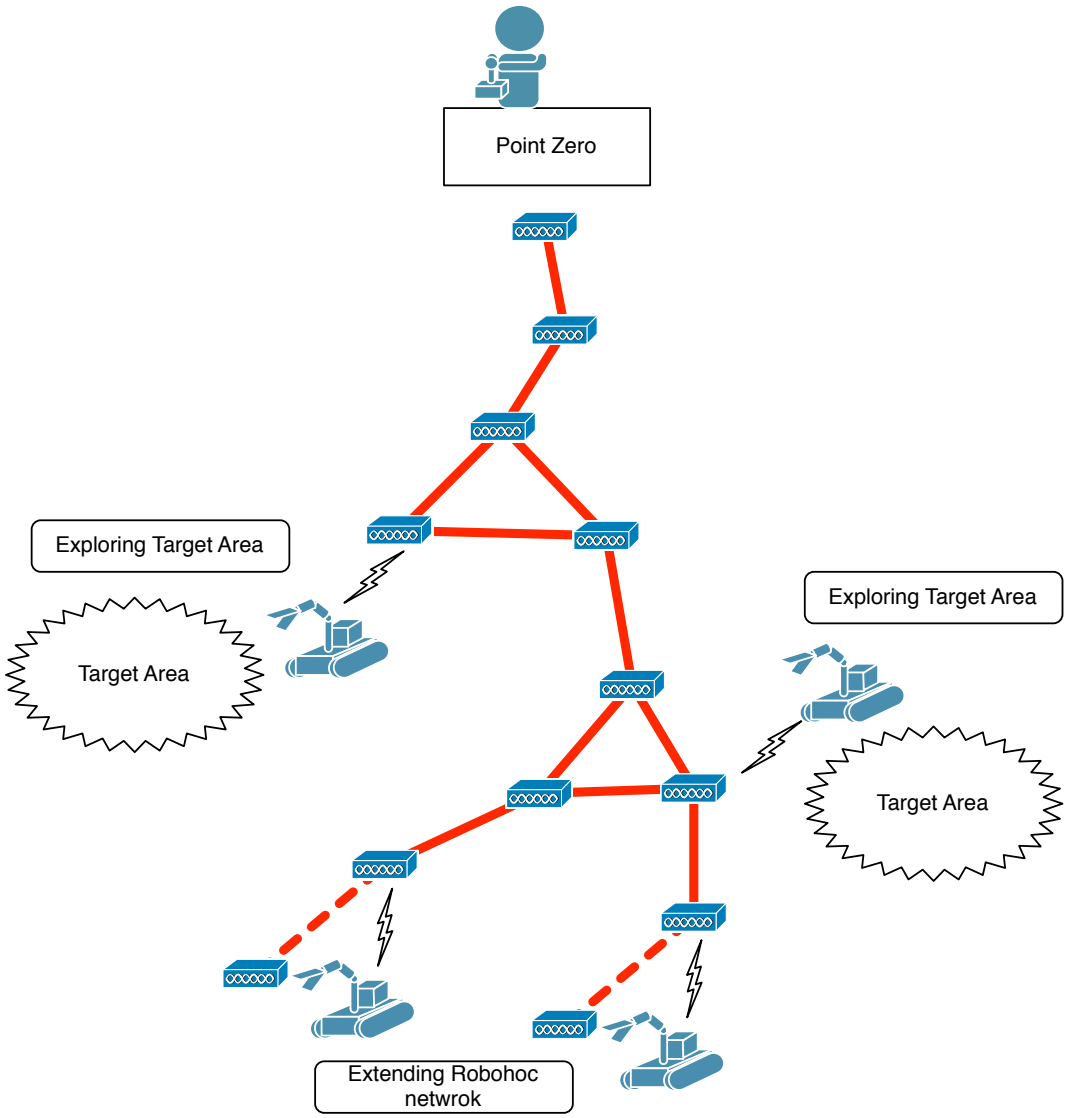


Figure 10.4. Concurrent search and expansion with multiple robots.

for victims or examining the target area to collect information. The collected information is delivered to the rescue team via the network and they will use it for further rescue actions.

After the rescuers get enough information from the robots, they can go to save victims with detailed information of the stricken area collected by the robots and it will make them safer than they go to there without any information.

The constructed Robohoc network is also useful during the rescue activities by human teams. For example, it can be used as a communication channel between teams, in the case where the existing communication infrastructure is damaged in a disaster situation. In addition, resident people may use the Robohoc network in a recovery period from the disaster as an alternative communication method until a more stable communication infrastructure becomes available.

3. Use Cases and Requirements

In this section, properties of the Robohoc network constructed as discussed in section 2 is described.

Before defining requirements for the Robohoc network, it is necessary to understand the assumed disaster situation and its base requirements. The disaster assumed here is the urban disaster, for example, the case a big earthquake hits a city or an explosion accident. In these situations rescuers have to explore collapsed buildings or underground stations where many partitioning walls and collapsed obstacles would be there. It cannot be assumed that a wide-range wireless communication device can be used in such a situation. A network needs to be constructed as a set of small wireless cells. When defining the size of the disaster area, we referred to the Yaesu underground shopping mall (about $73,000m^2$), which is one of famous underground shopping malls in Japan. The size fits most of the in-house disaster situation.

There are requirements which come from robot operation technologies too. The network delay gives a serious impact to a robot operator. The well-known fact is that one-second delay is the maximum delay to operate robots in real-time. If the delay is more than that, the real-time control becomes unrealistic, but as long as the robot knows the surrounding environment and the network

assures the fixed delay and jitter, the operator can still control the robot using the predictive control mechanism. Thus, robots need to have some interfaces to know the current environment (such as geographical information or building map information provided from the operator) and the network has to provide stable communication between robots and the operator.

3.1 RHR Distribution Property

The assumption of the research focus is that the Robohoc network constructed area is disaster stricken. Victims are usually isolated in broken houses/buildings or underground structures like underground shopping centers or subway stations. There should be many walls or objects broken by the disaster and inside aisles are not wide. For these reasons, access point routers cannot be located uniformly. Figure 10.5 illustrates the situation. When there are many obstacles between the Point Zero and the target area, the density and the path length between nodes have big deviation.

3.2 Communication Distance

The size of the disaster stricken area is usually unknown until the initial exploration completed. The distance between Point Zero and robots may vary from a few dozen of meters to a few kilometers. Moreover, this parameter depends on the disaster level. For a small size disaster such as a fire accident or explosion in a building, the distance would be a few dozen meters to a few hundreds meters. For a large disaster, like an earthquake in the metropolitan area, the collapsed area may spread a few dozens kilometers or more. The assumption in this case is that the distance between teleoperators and robots will be from a few hundreds meters to about 1 kilometer, because the exploration area can be divided into small pieces even in a large-scale disaster.

3.3 Network Partitioning

When exploring the stricken area, operators and rescuers ay face subsequent unexpected accidents. A RHR may stop working in the middle of an operation,

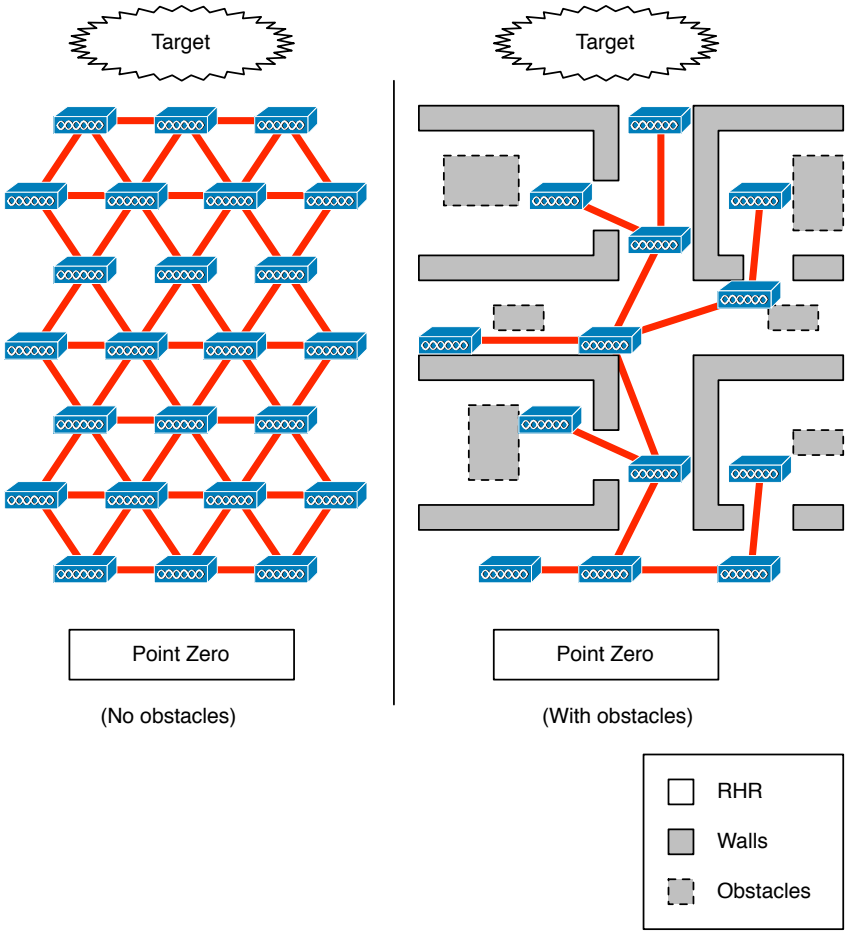


Figure 10.5. The network nodes cannot be located uniformly.

or it may be broken by an object fallen onto it since the surrounding environment is still unstable. In such cases, the constructed Robohoc network may divide into two or more sub-networks. The Robohoc network must have a property to recover from unexpected partitioning.

3.4 Real-time Robot Control

In the teleoperation situation, the quality requirements to the network vary depending on the control theory of the system including robots, controller and teleoperators. Moreover, the system designer should select the appropriate control theory according to the amount of network delay.

Generally some kind of compensation mechanism is required in the long latency environment, because the more the latency of network communication increases, the more the difficulty of teleoperation grows. To avoid the network latency problem in the teleoperation, various techniques, such as the prediction and preview display and the supervisory control system, are proposed and applied [63].

It is needed to consider the robot operation model for the Robohoc network to decide the network parameters.

A bilateral control theory is popular when the teleoperators require feedback from robots. It is generally known as difficult to make a stable controller in the case communication delay exists in the control/feedback loop. The scattering transform method partly solves the problem [2] if the delay is known and constant. However, the method cannot directly be applied to the Internet environment because the latency and jitter change dynamically, especially in the multi-hop wireless network. Some techniques are proposed ([38, 91]) for this problem, but it is still under addressing yet for the large fluctuation of jitter and long delay. Therefore, in the situation where the bilateral teleoperation is required, the network has to offer the communication assuring the upper bound of the jitter and latency.

If teleoperators need no bilateral feedback, some other control theories like a prediction display can be applied. The usability of this kind of teleoperation mechanism mainly depends on the skill and human characteristic of the teleoperator. With a well-designed teleoperation system, a teleoperator can manipulate

robots over a few-seconds delay of visual feedback [35]. It is well known that an ordinary person can operate up to one-second visual feedback delay.

Based on the previous facts and discussion, the target upper bound of latency was decided as $400ms$ (RTT: $800ms$). This value is delivered from the human factor of visual feedback delay. However, it is almost impossible to guarantee a small value against the upper bound of the latency because the Robohoc network consists of several RHRs and we cannot know the actual size of the network. There is a technology to control remote object even if the latency is not very small. In this case, though, the network must provide the constant or predictable communication latency.

3.5 Type of Service Support

In the Robohoc network, various types of data would be sent; for example, messages to control robots, live streaming movies from robots, location data of robots and RHRs collected by sensor devices equipped with robots and RHRs, or geographical data to create the up-to-date disaster area map. The requirements from each kind of data vary depending on the traffic property. For example, a live streaming data requires a high bandwidth channel but geographical data does not; control messages should require low latency and low bandwidth communication channel. Therefore, the Robohoc network has to support various communication properties based on the data types used in the network ([48, 59, 5]).

3.6 Topology Information Sharing and Storing

The Robohoc network has to know its topology information. To recover from a partitioned situation, both robots and teleoperators need the network and geometric topology information. Both network components (RHRs and robots) have to share this kind of information. For example, when a Robohoc network is partitioned into two sub-networks A and B, and one of them (A) is connected to the Point Zero, teleoperators can control inside of the sub-network A. The other sub-network (B) is not connected to the Point Zero and has to be reconnected to A. Because the robots can modify the Robohoc network, it may be possible to re-connect these sub-networks autonomously if B has some robots in it. To

recover the network, it has to inform robots of the broken link.

3.7 Bootstrap and Auto-Configuration

The rescue activity is a kind of fight against time. The construction of a Robohoc network and the collection of necessary data for the following rescue activities have to be done quickly. The RHRs and Robots need to be launched as soon as possible. To avoid that the configuration of these nodes become complex, every node has to be configured automatically with a minimal manual configuration ([90]).

3.8 Hop Counts

The assumption is that this technology is used in the urban disaster situation. The assumed size of the stricken area varies from one building/station to one town. The network communication tool should be a wireless network device since construction of a wired network is hard because of the weight of wire and possible obstacles on the path.

Considering that the range of one RHR is $50m$ (it is said that it is possible to obtain this range with the current 802.11a [1] technology without any hindrances) and the size of the stricken area is $300m \times 300m$, 36 nodes are needed when there are no radio waves disturbing objects. Because the wireless range will reduce when there are obstacles, it is required to add some margin for the range. At this point, we assume that we can enjoy the half range size of the best case. Of course it is necessary to verify the assumption in a real environment. About the size of the target area, the size of the Yaesu underground mall in Tokyo Japan is about $73,000m^2 \approx 270m \times 270m$ for example. The size of $300m \times 300m$ seems a good approximate value. Assuming that the range is $25m$, 144 RHRs are needed if they are put as a grid layout, and the maximum hop counts will be 24 hops. It means that the Robohoc network must work in a situation that there are more than 100 nodes and the hop counts between two nodes are more than 20 hops.

3.9 Layer 2 Information Utilization

When constructing a Robohoc network, it is required to adapt the environment of the target area. In the previous section, the assumption of the range of a RHR is about $25m$. However, it depends on the layout of obstacles and walls. To adapt to the actual environment, all RHRs and Robots need to have a function to monitor the radio wave quality. The routing protocol, which runs on RHRs and creates the logical Robohoc network topology, has to consider the advantage or weakness of the links to get a better performance and reduce service disruptions. This information has to be shared by all RHRs and Robots to optimize the total throughput of the network and to utilize the strategy to extend the network to reach further unexplored areas.

3.10 Fault Tolerance

The Robohoc network has to be fault tolerant in case of network failure. It is highly possible that the constructed network is broken by some accident when performing extension of the network or any rescue activity. In principle, the network must not have a single point of failure. The topology should be designed redundantly. However, it cannot be ensured that the network can always be redundant. In case of big accident which cannot be covered by the redundant topology, the Robohoc network should be recovered either by teleoperator's recovery procedures or automatically by robots. For example, if robots know the topology of the Robohoc network and they can get any failure information from the network, they may be able to move to the failure point to fix the broken link. This kind of approach requires the intelligent network, that is, the network has to know its topology and have a capability to detect the broken link in case of accident. In addition, the network and robots must have capabilities to exchange the location of the point and to re-construct a new topology to reconnect to the other partitioned network.

Table 10.1. The summary of the requirements for the Robohoc network

Required property	*Description*
RHR distribution	RHRs and robots cannot be located uniformly. The Robohoc network must support the non-flat node distribution. (Section 3.1)
Communication distance	The distance between teleoperators and robots is from a few hundred meters to about 1 kilometer. (Section 3.2)
Network partitioning	The Robohoc network may be partitioned while constructing the network or operating rescue activities. The network must have a property to recover from partitioning. (Section 3.3)
Real-time robot control	For real-time robot control, the network latency has to be less than $400ms$. Robots can be controlled even the latency is more than $400ms$, however, in that case, the latency has to be predictable and stable. (Section 3.4)
Type of service support	The Robohoc network must be able to provide different traffic properties for different contents, for example, the real-time delivery for the robot control and the wider bandwidth for the live streaming. (Section 3.4)
Topology information sharing and storing	When recovering from partitioning, teleoperators, RHRs and robots have to know the topology of the network to find the failure point. The topology information must be shared and stored in every node. (Section 3.6)
Bootstrap and auto-configuration	The network construction and rescue activities must be started as soon as possible. Every node must start with minimum manual configuration and must have an auto-configuration property. (Section 3.7)
Hop counts	The number of RHRs in a Robohoc network may be more than 100. The average hop count in this case would be more than 20. To support a wider area, the number of hops and average hop count will increase. (Section 3.8)
Layer 2 information utilization	The Robohoc network uses a wireless communication media to create the network. Each RHR has to monitor the link quality of its connection and utilize the information for better performance. (Section 3.9)
Fault Tolerance	The Robohoc network must not have a single point of failure. The network must be able to recover from partitioning either by the human intervention or by autonomous recovery actions of robots. (Section 3.10)

4. Related Works

[77] discusses an efficient algorithm to explore frontier areas by robots forming an adhoc network. In this proposal, robots are trying to avoid their network range conflict and spread to the frontier. [71] proposes a self-healing mechanism of a wireless network using cross-layer information exchange between the network layer and the application layer. These researches focus on completely autonomous system where human interaction is not necessary. In this research, the opposite approach was taken, that is, the human resources are utilized as much as possible since in rescue activities, the experience of rescue experts is sometimes useful. It is necessary to keep trying to give help to such people so that they can do better jobs than before.

5. Summary

Rescue teams are looking for new ways to perform rescue activities safer and more efficiently than those done only by human parties. Using helper robots for rescue activities is one of new approaches. Recently, high functional robots, can run over rough roads broken by disasters or which can go stairs up and down, are intensively being researched and developed. However, as long as the robots are controlled by short range communication devices, such as wires or direct wireless connections, it is impossible to expand the exploration range drastically. The networking technology can help on this problem. An idea of the new rescue network architecture (the Robohoc network) which can be dynamically extended and adapted by either human operated robots or autonomous robots was proposed in this chapter. Using the proposed network, one can construct a temporary rescue network adaptable to various disaster situations. Though the usefulness is clear, there have been few research activities in this area. This chapter showed the basic idea of the Robohoc network and the requirements to satisfy the properties necessary for rescue activities as a first step.

Chapter 11

Evaluation

In this chapter, the proposed mobility infrastructure design is evaluated based on the implementation activities and experiment result. The evaluation of the platform design is discussed in section 1, the adaptability evaluation of the platform is discussed in section 2, the evaluation of the performed experiments is discussed in section 3, and section 4 summarized the chapter.

1. Design Evaluation

In chapter 3, the requirements for the future mobility infrastructure are defined. The followings are the list of the requirements.

1. Separation of packet processing part and signal processing part
2. Standard procedures to interact the signal processing plane and packet processing plane

To achieve the first requirement, the conceptual system design was presented in figure 3.1 in chapter 3. The signal processing programs reside in the user space, handing all the signal packet message exchange and processing. After the signal processing, the result will be translated into the information which can be used to transform and forward packets. By splitting the mobility function into two parts, one is the signal processing part and the other is packet processing part, the system can keep packet forwarding performance. At the same time,

Table 11.1. The amount of code of the platform

Extension	Code increased	Kernel User space
(Base system)	(23359 lines)	(6281 lines) (17078 lines)
NEMO Basic Support	+1544 (+6.6%)	+22 (+0.35%) +1522 (+8.9%)
Multiple Care-of Addresses Registration	+969 (+4.1%)	+65 (+1.0%) +904 (+5.3%)
IPv4 Network Prefix Support	+337 (+1.4%)	+0 (+0%) +337 (+2.0%)
Global HA-HA	+852 (+3.6%)	+33 (+0.52%) +819 (+4.8%)

implementing signal processing mechanism, which is usually complex and needs more development resources, becomes easier than doing it in the kernel, thanks to the helpful debugging tools and environment provided in user space.

Based on the conceptual design, the actual mobility platform was designed and implemented as shown in figure 5.4 in chapter 5.

Table 11.1 shows the amount of code of the base system and the extended code for newly added functions in each experiment. As one can see from the table, all the extensions made through the implementation activities of this dissertation required only a small amount of additional code. The biggest extension is the NEMO Basic Support extension, which required 6.6% of code modification to the base system. It is difficult to judge if the amount is reasonable or not, however, at least the ratio of the additional code is smaller than the ratio of the amount of the protocol specification pages. As table 11.2 shows, although most extension specifications required 15% to 20% additional pages, the actual code modification required at most 6%. From this observation, it can be concluded that the proposed platform can be used as a base system of Mobile IPv6 based protocol implementation and it has enough extensibility for testing new extension protocols.

Table 11.2. Page count of each specification

Specification	Pages	Grow rate
Mobile IPv6 (RFC3775 [30])	165	–
NEMO Basic Support (RFC3963 [13])	33	+20%
Multiple Care-of Addresses Registration [81]	36	+21%
IPv4 Network Prefix Support [64]	25	+15%
Global HA-HA	(no spec yet)	–

Table 11.1 also shows that the kernel modification is smaller than the user space modification for all extensions. All the extensions required less than 1% modification to the kernel code. Compared to the kernel modification, the user space program needed to be modified more. This means that by separating the processing code into kernel and user space, and by reducing the modification of the kernel, most of the development of new protocols could be done in user space. This result shows that the original intention to provide an easy development platform to researchers and developers was achieved. Especially, in the IPv4 Prefix Support case, no kernel modification was necessary.

In the real implementation, several user space signal processing programs run in user space based on the mobility functions. Each program communicates with each other using the common interface to exchange information. Using the common interface makes it easier to replace some of the user space modules based on the operation requirements. For example, for the mobile terminal vendors, the necessary functions are mobile node's functions and they may not need home agent functions. In that case, they can choose moving node functions only. The other example is mobile network operators. The network operators may have an efficient support mechanism for mobile node handover. They may want to enhance the movement detection mechanism as they want. In this case, the movement detection program can be replaced without changing other mobility programs. This kind of inter-module communication is provided by the second

requirement. In the real implementation, the mobility socket is designed and implemented (section 4.5 in chapter 5). With this interface, each researcher or developer can enhance any user space signal processing programs, as long as he does not change the mobility socket interface protocol.

The interface was utilized during implementing extension protocols discussed in this dissertation. In the development of the NEMO Basic Support protocol, a new user space program which manages IP-in-IP tunneling function for mobile network traffic tunneling was developed separately from the other mobility signaling programs. The tunnel setup program monitors the signal packet exchange, and autonomously sets up or terminates mobile network tunnel connections. Thanks to the common interface, only small kernel modification was required as table 11.1 shows. In the Global HA-HA development case, two new signal messages need to be added. From the new message exchange, some additional packet forwarding rules are generated. The generated information is injected to the kernel using the common interface. Table 11.1 shows that the Global HA-HA requires 33 lines of kernel modification, however, all the 33 lines are the definition of the new message types. This means that, in the Global HA-HA case, almost no modification was required for the kernel. From these observations, it can be concluded that the mechanism to create common interface between user space and the kernel contributed to shift the development to user space, which is easier than the extension of kernel inside.

2. Platform Adaptability Evaluation

The implemented Mobile IPv6 platform was tested with four new extension protocols, NEMO BS, Multiple Care-of Addresses Registration, IPv4 Network Prefix, and Global HA-HA. In this section, the requirements for these protocols and the way that the platform was adapted to these requirements are briefly explained.

2.1 NEMO Basic Support

The NEMO Basic Support protocol is an extension of Mobile IPv6 to support network-level mobility. The main requirements of this protocol are shown below.

1. The protocol requires a tunnel connection between a moving network and its home agent.

2. A moving node acts as an IPv6 host for the external network interface to configure the external interface, and at the same time the node acts as an IPv6 router for the internal network interface.

The first requirement was solved by utilizing the common interface established between user space and the kernel. In the NEMO BS, a mobile network prefix is bound to a moving router. When the moving router registers its current location to its home agent, a mobile network tunnel has to be established together. The registration information is exchanged between the mobility signal processing program and the kernel using the mobility socket interface. The newly developed program for NEMO BS monitors the information exchange and establishes related tunnel connections when the registration occurs. The tunnel setup is done by the routing socket interface, which is a traditional common interface equipped with the BSD-based operating systems for routing information management.

The second requirement was handled by extending the kernel code. The modification of the kernel described in table 11.1 is mainly done to satisfy this requirement. The moving router is required to act differently from normal IPv6 host or router. It acts like an IPv6 host when it attaches to a foreign network, that is, it needs to obtain an IP address from the attached foreign network and has to configure a care-of address on its external network interface. This action is the same as that of normal IPv6 host node. On the other hand, it must provide routing function for the mobile network behind the internal network interface of the moving router. Usually, IPv6 routers do not configure its IP address automatically. The KAME IPv6 implementation, which is the base stack of the platform, does not allow such a behavior either. The kernel modification done in this extension makes it possible for IPv6 node to act both as a host on the external interface and as a router between the mobile network and its home agent.

By separating signal processing part from the other functions, the special requirements for the node behavior could be isolated in the kernel core.

2.2 Multiple Care-of Addresses Registration

When a moving host or router has several network interfaces, it is natural to have a demand to utilize the multiple network interfaces at the same time. Since the basic Mobile IPv6 specification does not allow registering multiple addresses at the same time, an extension was proposed at IETF. The requirements fro this extension are shown below.

1. Ability to keep multiple binding information for the same home address at the same time, on both home agent and mobile node
2. Packet filter mechanism to distribute traffic to multiple interfaces based on the traffic property

The binding information in the basic Mobile IPv6 protocol is distinguished by the home address of a mobile node, since the home address information has been the only unique information to distinguish nodes. However, when considering multiple concurrent registrations, the home address information is not unique anymore. The Multiple Care-of Address Registration protocol added the concept of *binding ID*. The binding information is distinguished by the pair of the binding ID and home address. This modification required update of the binding database schema. To implement this requirement, the binding databases in the kernel and user space are updated to keep the binding ID information. Because this extension required the kernel database modification, the size of the modified kernel code became largest among the four extensions developed in this dissertation. In user space, to support multiple interfaces, the movement detection program was updated to handle several interfaces at the same time. Even though the movement detection mechanism is enhanced, the interface to the mobility signal processing program has not changed. What is needed is to add a new message to inform binding ID information in conjunction to the movement notification information. Thus, in this implementation, it can be concluded that the common communication interface established among programs helps the development where the local modification does not affect to the wide range of the entire platform. The code amount required to support this extension also confirms the same conclusion as shown in table 11.1.

For traffic filtering mechanism, the implementation of Multiple Care-of Address Registration utilized the existing packet filtering mechanism. Since the tunnel between a mobile router and its home agent is implemented as a virtual IP-in-IP interface, a general traffic distribution mechanism can be used. In the experiment, the IP Filter mechanism [60] was used.

2.3 IPv4 Network Prefix

The original Mobile IPv6 and its extension NEMO BS do not support IPv4. Considering the situation that both IPv4 and IPv6 will be used together for a long time before networks can transit to IPv6 completely, IPv4 support is an important feature. The IPv4 network prefix support adds IPv4 mobile network function to NEMO BS. The requirements are as shown below.

1. When an IP-in-IP tunneling for a mobile network is established between a mobile router and its home agent, the IPv4 network prefix bound to the mobile network behind the mobile router has to be configured.

This extension is built on top of the NEMO BS extension. Since the NEMO BS extension already provides IPv6 tunneling mechanism for the mobile network, this IPv4 network prefix extension can utilize the mechanism. In this extension, when the routing information is set up for IPv6 mobile network prefix, IPv4 mobile network prefix bound to the mobile router is also set up. The extension did not require any kernel modification. Only the mobile network tunnel setup program had to be modified.

2.4 Global HA-HA

To solve the single point of failure problem cased by a home agent, and the redundant path problem which occurs when a mobile node and its correspondent node is topologically close but the home agent is topologically far, the home agent distribution mechanism is necessary. The specification has not been completed yet, and this extension requires the following functions.

1. Establish an overlay network of home agents.

2. Exchange binding information of mobile nodes between home agents.

3. Notify the nearest home agent information to mobile nodes.

The first requirement is implemented as a virtual tunnel interface between home agents. The home agent function program is extended to establish pre-configured virtual tunnels between home agents and sets up routing information between them.

For the second requirement, it is needed to define a new signaling message to send binding information from one home agent to other home agents. This requirement has also been solved in the user space program level, because all the signaling messages are processed by the user space program. When a home agent received binding information from a mobile node, it transfers the information to other home agents using the newly defined message. For this new message implementation, required kernel modification was just for defining the new message type structure.

For the third requirement, modification on both home agent and mobile node is necessary. When a home agent receives binding information from a mobile node via another home agent, it knows that the mobile node is closer to that another home agent than itself. In this case, the signal processing program for home agent sends the mobile node to re-register to the nearest home agent using a newly defined message. All processing is done in user space, and only the new message definition is required in kernel level.

In the Global HA-HA case, no additional packet transformation rules are defined. Thus, splitting signal processing and packet processing worked efficiently.

2.5 Other Examples

Other than the four extensions explained in this document, there are several examples which utilize the mobility platform proposed in this document.

Credit-Based Authorization for Concurrent Reachability Verification This protocol is proposed by Christian Vogt and Mark Doll as an Internet-Draft [79], and published as a part of the paper discussing the mechanism for efficient end to end mobility support [80]. The protocol proposes the way to

authorize two mobile nodes communicating each other in optimized path. If the basic Mobile IPv6 mechanism is used, these two nodes need to perform the return routability procedure to authorize each home address, which takes several message exchanges via their home agents. In the proposal, it is assumed that the communicating nodes are already authorized each other. The authorization level depends on the amount of traffic exchanged before. If the two nodes exchanged a lot of packets before moving, the two nodes are considered authorized each other and the return routability procedure can be omitted. The idea was evaluated by the mobility platform provided as a result of this dissertation work.

Dual Stack Mobile IPv6 Dual Stack Mobile IPv6 (DSMIPv6) [14] is an advanced version of IPv4 Network Prefix mechanism. It supports not only moving networks, but also moving nodes. With this mechanism, a DSMIPv6 node can attach either IPv4 or IPv6 networks, and it can use both IPv4 and IPv6 home addresses, in addition to the support of IPv4/IPv6 mobile networks. The implementation and evaluation work based on a slightly older version of the specification [73] was done by Koshiro Mitsuya and the result was reported at Asia BSD Conference 2007 [41].

Proxy Mobile IPv6 Proxy Mobile IPv6 (PMIP) [23] is a mechanism to provide mobility function to normal (non-mobile) IPv6 nodes. In PMIP, mobility function is implemented as a core network service, instead of a terminal function. The specification has already completed as RFC5213. Sawako Kiriyama implemented this specification and its content transfer mechanism on top of the mobility platform proposed in this paper, and evaluated the performance [36].

3. Experiment Evaluation

Using the mobility platform developed in this dissertation work, the following three experiments were performed based on the experiment plan proposed in chapter 3.

1. Proposal of smooth transition scenario from IPv4 to IPv4/IPv6 dual stack operation (Chapter 6)

Purpose Evaluate the extensibility of the proposed mobile platform.

2. Mobile Network handover with lower packet loss using Multiple Care-of Addresses Registration mechanism (Chapter 7)

   Purpose 1 Evaluate the extensibility of the proposed mobile platform

   Purpose 2 Evaluate the practicality of middle-scale mobile network.

3. Global HA-HA operation (Chapter 8)

   Purpose Evaluate the extensibility of the proposed mobile platform

3.1 Smooth Transition Scenario from IPv4 to Dual-Stack

The experiment discussed in chapter 6 aimed to evaluate the mechanism of IPv4 Network Prefix extension. The extension is implemented as discussed in section 2. The problem treated in this experiment is that the original NEMO BS protocol does not support IPv4 mobile network, and so it will discourage the shift to IPv6-based mobile world. In the effort to accommodate this problem, no large scale experiment was done. Instead of that, the laboratory scale small test network was used to verify the implementation build based on the proposed protocol. The result shows that the dual stack mobile network can be implemented on the proposed mobility platform with the proposed protocol.

3.2 Mobile Network Handover with Lower Packet Loss

When a mobile node moves from one network to another network, it has to detach from the previous link and then attach to the new link. During this period, the node loses the Internet connectivity. Several methods to reduce the impact of this problem have been proposed ([15, 72]). In the experiment discussed in chapter 7, the concurrent use of multiple network interfaces technology was focused. The implementation work in this experiment shows the extendability of the proposed mobile platform as discussed in section 2.

In the experiment, more than 200 people were attached to the mobile network. The scale of mobile networks varies depending on the scenario. For example, in the PAN scenario case, only a small number of devices equipped with a person

will be operated in a mobile network. In the transportation network case, such as a train network or an airplane network, a few dozen to a few hundred people may connect to one moving network. There are scenarios for larger mobile network usage, such as a site-multihoming case. In such a case, the network does not move actually, but logically changes its attachment point to the Internet.

The network focused in this experiment which covers 200 people was a fair assumption for the middle-class moving network, such as a transportation network. Two experiment were performed using conference networks. In the first experiment, the moving network was provided without multiple interfaces support. In the second experiment, the moving network utilized multiple interfaces when moving. The result shows the performance enhancement in one hand (figure 7.2 and 7.4 in chapter 7), and the practicability of the middle-class moving network by providing Internet connection for four days each during conferences.

3.3 Global HA-HA

Mobile IP-based mobility protocols rely on the proxy host, which is called as home agent, to forward packets to/from mobile nodes. When considering large scale deployment, or fault tolerant operation, a distribution mechanism of home agent is necessary. In the experiment of Global HA-HA, one of such ideas was implemented on the proposed mobility platform and tested at the network vendors/service providers' exhibition. The result shows the extendability of the mobility platform.

4. Evaluation Summary

To support the coming mobile Internet world, a mobility platform which can be used as a testbed of the new mobility protocols is needed. In this dissertation, two important requirements for such a platform are proposed. One is the separation of a packet processing part and a signal handling part. The other is to define a common interface among processing programs. The proposed design is evaluated by implementing Mobile IPv6 as a base mobility system, and four extension protocols, NEMO BS, Multiple Care-of Addresses Registration, IPv4 Network Prefix, and Global HA-HA. Section 1 and 2 show that how the

proposed design was used to support the extension protocols and that less than 7% code modification was required to implement new protocols. The mobility platform implementation was used in several experiments to show that the actual implementation works as a fundamental platform for mobility experiments. Four experiments were performed as a part of the dissertation, and three other experiments using the platform proposed in this dissertation have been done as long as I know. This shows the practicability of the proposed platform as a future mobility platform.

Chapter 12

Conclusion

Thanks to the worldwide deployment of the Internet, a common communication platform to connect various networking devices appeared. However, the protocol supporting the current Internet was designed more than 20 years ago and there are now many facts which the initial protocol designers could not forecast. One of such problems is the number of devices connecting to the Internet. Another issue is the improvement of wireless communication technology and downsizing technology, which can make every networking device be wireless. Considering the technology trend, the preparation is needed for the world in which tons of networking devices are connected to the Internet wirelessly and moving around. The activity discussed in this dissertation is to provide the future mobility research/development platform to prepare the approaching un-wired world.

In chapter 3, the design requirements are defined for such a mobility platform. The system is based on IPv6 and Mobile IPv6, which are the promised next generation Internet protocol. Because the platform must be able to support many new protocols proposed in the future, it must be extensible. The platform must be also easy to use for developers and researchers who work on the new ideas. The design proposed in this dissertation has two strong points. One is to separate the signal processing mechanism and the packet processing mechanism, and place them in user space and in the kernel respectively. With this design, the development of a new signal processing will be easier, while keeping the packet processing performance high. The other point is to define a common information exchange system between processing modules. Since many extension mechanisms

are constructed on the base mechanism, most extension proposals can utilize the existing base system. The common information exchange mechanism provides the way to add small modules which are required to support new functions, without changing the rest of the system drastically.

As the preparation to discuss the mobility, the base system is designed and evaluated in chapter 4 and 5. The IPv4, which is the major Internet Protocol at this moment, has 32bit identifier space. Recent research expects that the rapid growth of the Internet will consume all the usable addresses by the year 2011 if people keep the current address distribution scheme. IPv6 is designed to solve the problem, and I have been working on IPv6 from the beginning stage of its standardization process, which was started in around 1996. One of the first IPv6 stack implementations for the BSD operating system was designed as a common communication infrastructure for the future research activity. Through the work, the base design of the IPv6 implementation was proposed, and it was taken over by the KAME project and major BSD operating systems (FreeBSD, NetBSD, OpenBSD, and BSD/OS).

In the un-wired networking world, the mobility support protocol will be a key technology. It is no doubt that IPv6 would be the future Internet Protocol. The implemented base system focused on the IPv6-based mobility protocols, Mobile IPv6 and NEMO Basic Support. Same as the IPv6 implementation design, the implementation design of the IPv6 mobility protocols was done at first, and then its specification has been verified through the operation of the implementation. Through this work, two ways of the implementation, one is implementing in the kernel, and the other is implementing in user space, were compared. The comparison confirmed that the kernel implementation can process mobility signal processing more stable and fast than the user space implementation, however, as discussed in the design phase, the user space implementation can be more flexible. The derivative results from the user space implementation, such as Multiple Care-of Addresses Registration mechanism, IPv4 Mobile Network Prefix option mechanism, and Global Home Agent Distribution mechanism, show the flexibility.

Through the work to evaluate the platform flexibility, two experiments using the live Internet were performed. First one was related to the Multiple Care-of Addresses Registration mechanism. To accelerate transition to the IPv6 network

and deployment of Mobile IPv6-based technology, some application usages are proposed in chapter 6 and 7. In chapter 6, one usage of NEMO BS is proposed as a simple multihoming solution. This idea will help people who want multihoming function for fault tolerance but cannot afford to use the expensive BGP-based routing solution. In addition, as the solution is based on NEMO BS, IPv6 deployment may also be accelerated since NEMO BS runs on IPv6. In chapter 7, the other usage of NEMO BS as a smooth handover mechanism is proposed. I operated a small conference network under the mobile router running NEMO BS. By using the multiple care-of address registration technique, the network successfully moved from one external line to another external line. These results show that the implementation is ready to use for the real operation, and concurrently imply the possibility of mobility technology to apply non-mobile area.

The other experiment was related to the Global Home Agent Distribution mechanism. For the realistic deployment, business operation had to be considered. The Mobile IPv6 base specification has a single point of failure problem. In chapter 8, one of the solutions for the problem and the protocol implementation of it is proposed. The implementation of the idea was used at the Interop Tokyo 2008, which is one of the biggest networking exhibitions in Japan, and proved that the idea is feasible.

In chapter 9, the activity about operation experience and deployment effort is reported. A public Mobile IPv6 service was established in 2007 and has been operating for years by me and my research colleagues. Unfortunately, the number of users is not large (at the year 2009, only 80 registered users exist), but we believe that we could show operability of the service by doing the service actually for many years. On the other hand, the effort is continued to integrate the developed code to NetBSD operating system. Although the work is still ongoing, the initial step to make the working area in the NetBSD source code repository has already been completed. Once the code will be integrated to the original NetBSD distribution, more people can have easy access to the mobility code, which will accelerate the deployment of the function.

Finally, one question to future mobility researchers including us about the limitation of the Mobile IPv6-based technology has been thrown in chapter 10. The Mobile IPv6-based protocol assumes the firm infrastructure support. It re-

quires a stable network connectivity and operation of home agents. In some cases, though, we may need mobility function in an unstable networking environment, such as disaster recovery environment. In chapter 10, the requirements to build an information gathering network for rescue robots in a disaster recovery phase are discussed. In such a case, it is not possible to rely on the existing infrastructure. Rescuers need to build an ad-hoc network infrastructure and have to operate moving entities (robots) in that environment. Since it is difficult to define one general solution for such an environment, the requirements for the limited environment such as underground shopping mall, are discussed as the first step. This work needs more effort to investigate various scenarios and to generalize the requirements for the future infrastructure-less mobile environment.

In conclusion, in this dissertation, the design of the extensible mobility research and development platform is proposed, and evaluated by actually implementing new mobility extension protocols on it. From the work, it is shown that the platform can support at least four extension protocols discussed in this paper. In addition to them, three other research activities [79, 80, 41] were done by other researchers, which indicates that the platform is useful not only for the designer but also for other developers and researchers. The implemented protocols were used in the live Internet environment, such as in the conference networks and in the event network, and showed their capability to be combined with the existing infrastructure. With these results, it can be concluded that the proposed design of the mobility platform can be an infrastructure for the future mobility research and development.

References

[1] Std 802.11a 1999. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer in the 5GHz Band*. IEEE, r2003 edition, June 2003.

[2] Robert J. Anderson and Mark W. Spong. Bilateral Control of Teleoperators with Time Delay. *IEEE Transaction on Automatic Control*, 34(5):494–501, May 1989.

[3] Martin André. A Practical Evaluation of the Nautilus6 Operational Home Agent Service. In *IPv6 Today – Technology and Deployment (IPv6TD'07)*. International Academy Research and Industry Association, IEEE Computer Society, March 2007.

[4] Jari Arkko, Vijay Devarapalli, and Francis Dupont. *Using IPsec to Protect Mobile IPv6 Signaling Between Mobile Nodes and Home Agents*. IETF, June 2004. RFC3776.

[5] Bob Braden, Lixia Zhang, Steve Berson, Shai Herzog, and Sugih Jamin. *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*. IETF, September 1997. RFC2205.

[6] Samita Chakrabarti and Erik Nordmark. *Extension to Socket API for Mobile IPv6*. IETF, July 2006. RFC4584.

[7] Thomas Heide Clausen, Christopher M. Dearlove, and Philippe Jacquet. *The Optimized Link-State Routing Protocol version 2*. IETF, June 2006. draft-ietf-manet-olsrv2-02.

[8] Rob Coltun, Dennis Ferguson, and John May. *OSPF for IPv6*. IETF, December 1999. RFC2740.

[9] Alex Conta and Stephen Deering. *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. IETF, December 1998. RFC2463.

[10] Stephen E. Deering, William C. Fenner, and Brian Haberman. *Multicast Listener Discovery (MLD) for IPv6*. IETF, October 1999. RFC2710.

[11] Stephen E. Deering and Robert M. Hinden. *Internet Protocol, Version 6 (IPv6)*. IETF, December 1998. RFC2460.

[12] Steve Deering. *Host Extensions for IP Multicasting*. IETF, August 1989. RFC1112.

[13] Vijay Devarapalli, Ryuji Wakikawa, Alexandru Petrescu, and Pascal Thubert. *Network Mobility (NEMO) Basic Support Protocol*. IETF, January 2005. RFC3963.

[14] Hesham Soliman (Editor). *Dual Stack Mobile IPv6 (DSMIPv6) for Hosts and Routers*. IETF, December 2008. draft-ietf-mext-nemo-v4traversal-07.

[15] Rajeev Koodli (Editor). *Fast Handovers for Mobile IPv6*. IETF, June 2008. RFC5268.

[16] Thierry Ernst, Nicolas Montavont, Ryuji Wakikawa, ChanWah Ng, and Koojana Kuladinithi. *Motivations and Scenarios for Using Multiple Interface and Global Addresses*. IETF, February 2006. draft-ietf-monami6-multihoming-motivation-scenario-00.

[17] Thierry Ernst and Keisuke Uehara. Connecting Automobiles to the Internet. In *3rd International Workshop on ITS Telecommunications (ITST)*, November 2002.

[18] Francis Dupont. An implementation of IP version 6 for NetBSD, March 1996.

[19] Satomi Fujimaki, Keiichi Shima, Keisuke Uehara, and Fumio Teraoka. The Deployment of Mobility Protocols Based on the Home Agent Service with Easy Interface. In *Internet Conference 2006 (IC2006)*, October 2006.

[20] Geoff Huston. IPv4 Address Report. Web page, December 2008. http://www.potaroo.net/tools/ipv4/index.html.

[21] Robert E. Gilligan, Susan Thomson, Jim Bound, and Richard W. Stevens. *Basic Socket Interface for IPv6*. IETF, March 1999. RFC2553.

[22] Go-core project. MIPL Mobile IPv6, December 2006. http://www.mobile-ipv6.org/.

[23] Sri Gundavelli, Kent Leung, Vijay Devarapalli, Kuntal Chowdhury, and Bsavaraj Patil. *Proxy Mobile IPv6*. IETF, August 2008. RFC5213.

[24] Brian Haley, Vijay Devarapalli, James Kempf, and Hui Deng. *Mobility Header Home Agent Switch Message*. IETF, January 2008. RFC5142.

[25] Robert Hinden. *Virtual Router Redundancy Protocol (VRRP)*. IETF, April 2004. RFC3768.

[26] Impress Corporation. IPv6 Service in Japan. Web page, December 2006. http://www.ipv6style.jp/en/statistics/services/index.shtml.

[27] Interop Tokyo 2008, June 2008. http://www.interop.jp/.

[28] Tatuya Jinmei, Jun-ichiro Ito, and Munechika Sumikawa. Efficient Use of IPv6 Auto-Configuration in a Mobile Environment. In *The 7th Research Reporting Session*. Information Processing Society of Japan, SIG Mobile Computing, December 1998.

[29] Tatuya Jinmei, Kazuhiko Yamamoto, Jun-ichiro Hagino, Shoichi Sakane, Hiroshi Esaki, and Jun Murai. The IPv6 Software Platform for BSD. *IEICE Transactions on Communications*, E86-B(2):464–471, February 2003.

[30] David B. Johnson, Charles E. Perkins, and Jari Arkko. *Mobility Support in IPv6*. IETF, June 2004. RFC3775.

[31] Dave Katz. *Transmission of IP and ARP over FDDI Networks.* IETF, January 1993. RFC1390.

[32] Stephen Kent. *IP Authentication Header.* IETF, December 2005. RFC4302.

[33] Stephen Kent. *IP Encapsulating Security Payload (ESP).* IETF, December 2005. RFC4303.

[34] Stephen Kent and Karen Seo. *Security Architecture for the Internet Protocol.* IETF, December 2005. RFC4301.

[35] Won S. Kim and Anatal K. Bejczy. Demonstration of a high-fidelity predictive/preview display technique for telerobotic servicing in space. *IEEE Transaction on Robotics and Automation*, 9(5):698–702, October 1993.

[36] Sawako Kiriyama. A Context Transfer Mechanism to Improve Handover Performance in Proxy Mobile IPv6. Master's thesis, Keio University, Japan, 2009.

[37] R. (Editor) Koodli. *Fast Handovers for Mobile IPv6.* IETF, July 2005. RFC4068 (Obsoleted by RFC5268).

[38] Kazuhiro Kosuge and Hideyuki Murayama. Teleoperation via computer network. *Electrical Engineering in Japan*, 124(3):49–56, December 1998.

[39] Kent Leung, Gopal Dommety, Vidya Narayanan, and Alexandru Petrescu. *IPv4 Network Mobility (NEMO) Basic Support Protocol.* IETF, October 2005. draft-leung-nemov4-base-00.

[40] Gary Scott Malkin and Robert E. Minnear. *RIPng for IPv6.* IETF, January 1997. RFC2080.

[41] Koshiro Mitsuya, Ryuji Wakikawa, and Jun Murai. Implementation and Evaluation of the Dual Stack Mobile IPv6. In *Asia BSD Conference (AsiaBSDCon2007)*, March 2007.

[42] Jeffrey Mogul and Steve Deering. *Path MTU Discovery.* IETF, November 1990. RFC1191.

[43] Tsuyoshi Momose, Keiichi Shima, and Anti Tuominen. *The application interface to exchange mobility information with Mobility subsystem (Mobility Socket, AF_MOBILITY)*. IETF, June 2005. draft-momose-mip6-mipsock-00.

[44] Robert Moskowitz, Pekka Nikander, Petri Jokela, and Thomas R. Henderson. *Host Identity Protocol*. IETF, April 2008. RFC5201.

[45] Kenichi Nagami, Nobuo Ogashiwa, Ryuji Wakikawa, Hiroshi Esaki, and Hiroyuki Ohnishi. *Multi-homing for small scale fixed network Using Mobile IP and NEMO*. IETF, February 2005. draft-nagami-mip6-nemo-multihome-fixed-network-03.

[46] Thomas Narten, Erik Nordmark, William Allen Simpson, and Hesham Soliman. *Neighbor Discovery for IP Version 6 (IPv6)*. IETF, September 2007. RFC4861.

[47] Nautilus6 project, November 2008. http://www.nautilus6.org/.

[48] Kathleen Nichols, Steven Blake, Fred Baker, and David L. Black. *Definition of the Differenciated Service Field (DS Field) in the IPv4 and IPv6 Headers*. IETF, December 1998. RFC24274.

[49] Eric Nordmark and Marcelo Bagnulo. *Shim6: Level 3 Multihoming Shim Protocol for IPv6*. IETF, December 2006. draft-ietf-shim6-proto-11.

[50] Erik Nordmark and William Allen Simpson. *Neighbor Discovery for IP Version 6 (IPv6)*. IETF, August 1996. RFC1970.

[51] Craig Partridge. *Using the Flow Label Field in IPv6*. IETF, June 1995. RFC1809.

[52] Basavaraj Patil, Phil Roberts, and Charles E. Perkins. *IP Mobility Support for IPv4*. IETF, August 2002. RFC3344.

[53] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir R. Das. *Ad hoc On-Demand Distance Vector (AODV) Routing*. IETF, July 2003. RFC3561.

[54] David C. Plummer. *An Ethernet Address Resolution Protocol – or – Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*. IETF, November 1982. RFC826.

[55] Jonathan B. Postel. *User Datagram Protocol*. IETF, August 1980. RFC768.

[56] Jonathan B. Postel. *Internet Control Message Protocol*. IETF, September 1981. RFC792.

[57] Jonathan B. Postel. *Internet Protocol*. IETF, September 1981. RFC791.

[58] Jonathan B. Postel. *Transmission Control Protocol*. IETF, September 1981. RFC793.

[59] K. K. Ramakrishnan, Sally Floyd, and David L. Black. *The Addition of Explicit Congestion Notification (ECN) to IP*. IETF, September 2001. RFC3168.

[60] Darren Reed. IP Filter. Web page, August 2007. http://coombs.anu.edu.au/~avalon/.

[61] John K. Renwick and Andy Nicholson. *IP and ARP on HIPPI*. IETF, October 1992. RFC1374.

[62] Binoy Shah and Howie Choset. Survey on Urban Search and Rescue Robots. *Journal of the Robotics Society of Japan*, 22(5):582–586, 2004.

[63] Thomas B. Sheridan. Space teleoperation through time delay – Review and prognosis. *IEEE Transactions on Robotics and Automation (1042–296X)*, 9(5):592–606, October 1993.

[64] Keiichi Shima. *IPv4 Mobile Network Prefix Option for NEMO Basic Support Protocol*. IETF, October 2005. draft-shima-nemo-v4prefix-01.

[65] Keiichi Shima. The design and implementation of a dual-stack mobile network using IPv6 only network infrastructure. In *The First International Workshop on Network Mobility (WONEMO)*. ICOIN, January 2006.

[66] Keiichi Shima, Romain Kuntz, and Koshiro Mitsuya. Nautilus6 mobile technology demonstration at the First IPv6 Summit in Thailand. Technical Report wide-tr-nautilus6-mobility-demo-thai-ipv6-summit-00, WIDE Project, July 2006.

[67] Keiichi Shima, Yojiro Uo, Nobuo Ogashiwa, and Satoshi Uda. An operational demonstration of a mobile network with a fairly large number of nodes. In *The International Symposium on Applications and the Internet Workshops (SAINTW'06)*, pages 6–9. IEEE Computer Society and Information Processing Society of Japan, IEEE Computer Society, January 2006.

[68] Keiichi Shima, Yojiro Uo, Nobuo Ogashiwa, and Satoshi Uda. Operational Experiment of Seamless Handover of a Mobile Router using Multiple Care-of Address Registration. *Academy Publisher Journal of Networks*, 1(3):23–30, July 2006.

[69] Keiichi Shima, Ryuji Wakikawa, Koshiro Mitsuya, Tsuyoshi Momose, and Keisuke Uehara. SHISA: The IPv6 Mobility Framework for BSD Operating Systems. In *IPv6 Today – Technology and Deployment (IPv6TD'06)*. International Academy Research and Industry Association, IEEE Computer Society, August 2006.

[70] Keith Sklower. A Tree-based Packet Routing Table for Berkeley UNIX. In *Proceedings of the Winter 1991 USENIX Conference*, pages 93–103. USENIX Association, January 1991.

[71] Matthew Smith and Bernd Freisleben. Self-Healing Wireless Ad Hoc Networks Based On Adaptive Node Mobility. In *Proceedings of the First IFIP International Conference on Wireless and Optical Communications Networks*, pages 121–124. IFIP, June 2004.

[72] Hesham Soliman, Claude Castelluccia, Karim ElMalki, and Ludovic Bellier. *Hierarchical Mobile IPv6 (HMIPv6) Mobility Management*. IETF, October 2008. RFC5380.

[73] Hesham Soliman, George Tsirtsis, Vijay Deverapalli, James Kempf, Henrik Levkowetz, Pascal Thubert, and Ryuji Wakikawa. *Dual Stack Mobile IPv6*

(DSMIPv6) for Hosts and Routers. IETF, October 2006. draft-ietf-mip6-nemo-v4traversal-03.

[74] Fumio Teraoka, Kazutaka Gogo, Koshiro Mitsuya, Rie Shibui, and Koki Mitani. *Unified L2 Abstractions for L3-Driven Fast Handover*. IETF, September 2006. draft-irtf-mobopts-l2-abstractions-01.

[75] Pascal Thubert, Ryuji Wakikawa, and Vijay Devarapalli. *Global HA to HA protocol*. IETF, March 2008. draft-thubert-mext-global-haha-00.

[76] Guillaume Valadon, Ryuji Wakikawa, and Jun Murai. Extending Home Agent Migration To Mobile IPv6 based Protocols. In *Asian Internet Engineering Conference (AINTEC)*, pages 27–29, November 2007.

[77] Jose Vazquez and Chris Malcolm. Distributed Multirobot Exploration Maintaining a Mobile Network. In *Proceedings of 2nd IEEE International Conference on Intelligent Systems*, volume 3, pages 113–118, June 2004.

[78] Christian Vogt. *Early Binding Updates for Mobile IPv6*. IETF, August 2006. draft-vogt-mobopts-simple-ebu-00.

[79] Christian Vogt and Jari Arkko. *Credit-Based Authorization for Concurrent Reachability Verification*. IETF, August 2006. draft-vogt-mobopts-simple-cba-00.

[80] Christian Vogt and Mark Doll. Efficient End-to-End Mobility Support in IPv6. In *Wireless Communications and Networking Conference, 2006 WCNC 2006. IEEE*, volume 1, pages 575–580, April 2006.

[81] Ryuji Wakikawa, Thierry Ernst, and Kenichi Nagami. *Multiple Care-of Addresses Registration*. IETF, February 2006. draft-wakikawa-mobileip-multiplecoa-05.

[82] Ryuji Wakikawa et al. *Home Agent Reliability Protocol*. IETF, July 2008. draft-ietf-mip6-hareliability-04.

[83] Ryuji Wakikawa, Yasuhiro Ohara, and Jun Murai. Virtual Mobility Control Domain for Enhancements of Mobility Protocols. In *The 8th IEEE Global Internet Symposium 2005*, volume 4, pages 2792–2797, March 2005.

[84] Ryuji Wakikawa, Keisuke Uehara, Thierry Ernst, and Kenichi Nagami. *Multiple Care-of Addresses Registration*. IETF, June 2004. draft-wakikawa-mobileip-multiplecoa-03.

[85] Ryuji Wakikawa, Guillaume Valadon, and Jun Murai. Migrating Home Agents towards Internet-Scale Mobility Deployments. In *ACM 2nd CoNEXT Conference on Future Networking Technologies*, Dec 2006.

[86] WIDE project. KAME Working Group, March 2006. http://www.kame.net/.

[87] WIDE project, November 2008. http://www.wide.ad.jp/.

[88] WIDE project. SHISA, November 2008. http://www.mobileip.jp/.

[89] WIDE project. USAGI Working Group, November 2008. http://www.linux-ipv6.org/.

[90] Aidan Williams. *Requirements for Automatic Configuration of IP Hosts*. IETF, September 2002. draft-ietf-zeroconf-reqts-12.

[91] Yuh Yamashita and Hirokazu Nishitani. Compensation of transmission delay and packet-loss in nonlinear remote-control system. In *10th IFAC/IFORS/IMACS/IFIP Symposium on Large Scale Systems: Theory and Applications*, pages 627–632, July 2004.

[92] Zebra Project. GNU Zebra. http://www.zebra.org/.

Achievements

Papers

- Keiichi Shima, Koshiro Mitsuya, Tsuyoshi Momose, Shuichi Karino, Ryuji Wakikawa, Kazushi Sugyou, and Keisuke Uehara. "Designing and Implementing IPv6 Mobility stack on BSD Operating Systems." Computer Software, Vol. 24, No. 4, pp. 23–39, October 2007.

- Keiichi Shima, Yojiro Uo, Nobuo Ogashiwa, and Satoshi Uda. "Operational Experiment of Seamless Handover of a Mobile Router using Multiple Care-of Address Registration." *Academy Publisher Journal of Networks*, Vol. 1, No. 3, pp. 23–30, July 2006.

Mater's Thesis

- Keiichi Shima. "An empirical verification of interconnectivity of IP version 6." Mater's Thesis, Nara Advanced Institute of Science and Technology, 1996.

Books

- Qing Li, Tatuya Jinmei, and Keiichi Shima. "IPv6 Advanced Protocols Implementation." *Morgan Kaufmann*, 2007. ISBN 0123704790.

- Qing Li, Tatuya Jinmei, and Keiichi Shima. "IPv6 Core Protocols Implementation." *Morgan Kaufmann*, 2006. ISBN 0124477518.

Conference Papers (reviewed)

- Keiichi Shima, Yojiro Uo, and Sho Fujita. "Auto configuration and management mechanism for the robotics self extensible WiFi network." In *International Conference on Instrumentation, Control and Information Technology (SICE Annual Conference 2008)*, pp. 1648–1652. The Society of Instrument and Control Engineers, August 2008.

- Keiichi Shima, Koshiro Mitsuya, Ryuji Wakikawa, Tsuyoshi Momose, and Keisuke Uehara. "SHISA: The Mobile IPv6/NEMO BS Stack Implementation Current Status." In *Asia BSD Conference (AsiaBSDCon2007)*, pp. 67–78, March 2007.

- Keiichi Shima and Yojiro Uo. "Requirements for Quick Network Construction Mechanisms for the On-Site Rescue Activity." In *Internet Conference 2006 (IC2006)*, October 2006.

- Satomi Fujimaki, Keiichi Shima, Keisuke Uehara, and Fumio Teraoka. "The Deployment of Mobility Protocols Based on the Home Agent Service with Easy Interface." In *Internet Conference 2006 (IC2006)*, October 2006.

- Keiichi Shima, Ryuji Wakikawa, Koshiro Mitsuya, Tsuyoshi Momose, and Keisuke Uehara. "SHISA: The IPv6 Mobility Framework for BSD Operating Systems." In *IPv6 Today – Technology and Deployment (IPv6TD'06). International Academy Research and Industry Association, IEEE Computer Society*, August 2006.

- Keiichi Shima, Yojiro Uo, Nobuo Ogashiwa, and Satoshi Uda. "An operational demonstration of a mobile network with a fairly large number of nodes." In *The International Symposium on Applications and the Internet Workshops (SAINTW'06)*, pp. 6–9. *IEEE Computer Society and Information Processing Society of Japan*, *IEEE Computer Society*, January 2006.

- Keiichi Shima. "The design and implementation of a dual-stack mobile network using IPv6 only network infrastructure." In *The First International Workshop on Network Mobility (WONEMO). ICOIN*, January 2006.

Internet-Drafts (no review)

- Ryuji Wakikawa, Keiichi Shima, and Noriyuki Shigechika. "The Global HAHA Operation at the Interop Tokyo 2008." *IETF*, July 2008. draft-wakikawa-mext-haha-interop2008-00.

- Keiichi Shima. "IPv4 Mobile Network Prefix Option for NEMO Basic Support Protocol." *IETF*, October 2005. draft-shima-nemo-v4prefix-01.

- Tsuyoshi Momose, Keiichi Shima, and Anti Tuominen. "The application interface to exchange mobility information with Mobility subsystem (Mobility Socket, AF_MOBILITY)." *IETF*, June 2005. draft-momose-mip6-mipsock-00.

- Keiichi Shima. "Route Optimization hint option." *IETF*, October 2003. draft-shima-mip6-rohints-00.

Acknowledgments

The author thanks Professor Hideki Sunahara for giving me a precious chance to summarize my past activities of IPv6 and IPv6 mobility. Without his support, this dissertation would not be published.

This dissertation could not be made without my co-workers and colleagues. The author especially thanks Dr. Kazuhiko Yamamoto and Professor Keijiro Araki, who gave me a chance to work on IPv6. Masaki Minami, Atsushi Onoe, Yoshifumi Atarashi, Ken Watanabe, Shin'ichi Hamamoto, and Munechika Sumikawa gave precious discussions of IPv6 implementation design. Dr. Ryuji Wakikawa, Kazushi Sugyo, Shuichi Karino, Tsuyoshi Momose, Dr. Keisuke Uehara, and Dr. Koshiro Mitsuya are my war buddies at IETF when working on Mobile IPv6. The late Dr. Jun'ichiro Itojun Hagino, Dr. Tatsuya Jinmei, Shoichi Sakane, Shinsuke Suzuki, and other KAME project members gave me useful general comments in the IPv6 and Mobile IPv6 field. Hideaki Yoshifuji, Noriaki Takamiya, Kazunori Miyazawa, Masahide Nakamura, Masafumi Aramoto, Shinta Sugimoto, and other USAGI project members gave me useful information in Linux IPv6 and Mobile IPv6 area. Dr. Thierry Ernst, Romain Kuntz, Martin André, Sébastien Decugis, and other Nautilus6 project members gave me irreplaceable discussions on mobility technologies. The WIDE project and Interop Tokyo 2008 NOC members supported my several experiments by giving me the real Internet as a testbed, which increased the level of practicability of my works. Internet Initiative Japan allowed me to spend some time in working on the dissertation as a part of my company jobs.

Finally, the author thanks the late Professor Masaki Hirabaru who invited me to his laboratory and shed the light on the way to the Internet for me.

Appendix A

SHISA Document

SHISA is a Mobile IPv6-based IPv6 mobility protocol stack implementation for BSD operating systems. The stack supports Mobile IPv6 [30, 4] as its basic function. The stack is designed to be able to support extended functions based on Mobile IPv6. At this moment, SHISA supports NEMO Basic Support [13], Multiple Care-of Addresses Registration [81] and IPv4 Mobile Network Prefix Option for NEMO Basic Support Protocol [64] as extensions. It also supports Dual-Stack Mobile IPv6 (DSMIPv6) for Hosts and Routers [73] and the mechanism to distribute home agents globally as experimental functions.

In this appendix, the detailed implementation design and its explanation will be provided.

1. SHISA Availability

SHISA is available from the SourceForge repository as of March 2009. The URL is as follows:

```
http://sourceforge.net/projects/shisa/
```

The kernel source code which supports the SHISA function is available from the NetBSD source code repository, with the branch name *keiichi-mipv6*. As of March 2009, only NetBSD is the supported platform of SHISA.

In the following sections, the files in the SHISA project is referred as `${SHISA}/`*`referred-filena`* and the files in the NetBSD source code is referred as `${NETBSD}/`*`referred-filename`*.

2. SHISA Design

SHISA consists of two parts. One resides in the kernel to process packet transform and tunnel/forward processing. The other resides in the user program space to manage Mobile IPv6 (and related mobility) protocol signal message processing. The reason why the stack is divided into two parts is to achieve an easy development environment to protocol developers and researchers, while keeping fast packet processing performance. The packet processing in Mobile IPv6 is performed based on the information inserted from the user space signal processing program. Since the signal processing is sometimes complicated, and there are many proposals to extend Mobile IPv6, it is reasonable to move that part to user space. Such a complicated and/or unstable code for under-specifying protocols must have many bugs. Debugging in the kernel space is harder compared to debugging in user space. Because the kernel sometimes uses unfamiliar functions for basic processing such as memory management, and debugging tools for the kernel requires much resources such as remote debugging terminal. Thus, keeping the kernel simple will help development of the signal processing code of new protocols, which is the most important part in protocol development.

One additional component is required when splitting one protocol processing into the kernel part and the user space part, which is the communication mechanism between the two parts. For example, in the routing mechanism implementation, the *routing socket* is used for that purpose. The kernel only has a packet forwarding information (usually called as a routing table). Based on the information, the kernel process the incoming packets. The kernel does not involve the update of the forwarding information. The information is updated by the user space application program, usually called as routing daemon program. A routing daemon program implements the routing information exchange protocol specification and communicates with other routing daemon programs running on other nodes. The result of the routing information protocol exchange will be translated to the forwarding information and set to the kernel through the routing socket.

In SHISA system, the similar mechanism, called as *mobility socket*, is provided. Figure A.1 shows the role of the mobility socket mechanism. The mobility occurs both in the kernel and in user space. For example, the event of the binding information update occurs in user space as a result of the mobility signal message

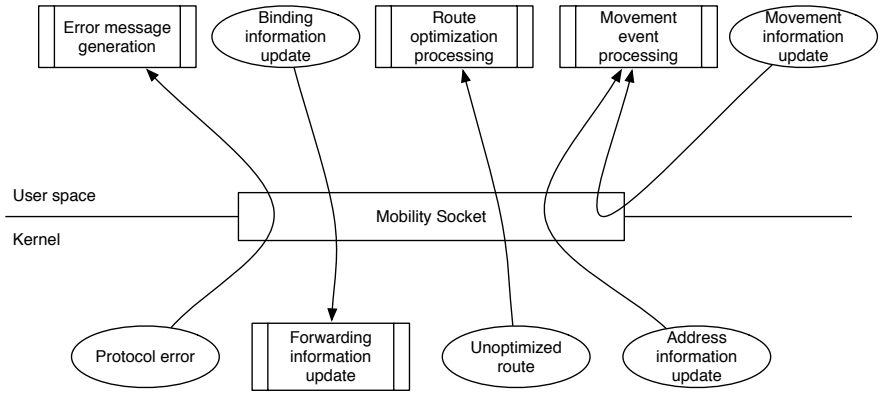


Figure A.1. The role of the Mobility Socket

processing. Another example is the event of the protocol error, like that an unverified home address is used in the incoming packet. This occurs in the packet input function of the kernel. Some of the events require additional processing in the opposite side, for example, the binding information update event requires the update of the forwarding information in the kernel. The mobility socket is used to exchange such information.

The mobility socket is also used between the user space modules, as shown in figure A.1. The movement information update event which occurs in user space triggers the mobility signal message exchange for the binding information exchange between a mobile node and its home agent. Since the binding information exchange is implemented in user space, the event should be notified to other user space modules. The mobility socket is used in this case too.

3. Mobility Socket

In the following sections, the detailed implementation of the mobility socket is discussed.

3.1 Message Structures

All the messages used in mobility socket communication are defined as mobility messages. Table A.1 shows the list of mobility socket messages and relevant struc-

Table A.1. Mobility messages

Message name	Msg #	Structure name	Description
MIPM_NODETYPE_INFO	1	mipm_nodetype_info{}	Change the node type.
MIPM_BC_ADD	2	mipm_bc_info{}	Add a binding cache entry.
MIPM_BC_REMOVE	4	mipm_bc_info{}	Remove a binding cache entry.
MIPM_BC_FLUSH	5	–	Remove all binding cache information.
MIPM_BUL_ADD	6	mipm_bul_info{}	Add a binding update list entry.
MIPM_BUL_REMOVE	8	mipm_bul_info{}	Remove a binding update list entry.
MIPM_BUL_FLUSH	9	–	Remove all binding update information.
MIPM_MD_INFO	10	mipm_md_info{}	Notify movement detection status.
MIPM_HOME_HINT	11	mipm_home_hint{}	Notify the attachment to the home network.
MIPM_RR_HINT	12	mipm_rr_hint{}	Notify the request to start the return routability procedure.
MIPM_BE_HINT	13	mipm_be_hint{}	Notify the necessity to send a Binding Error message.
MIPM_DAD	14	mipm_dad{}	Perform the Duplicate Address Detection procedure for the specified home address.

ture names defined in ${NETBSD}/sys/net/mipsock.h, ${NETBSD}/sys/netinet/ip6mh.h, and ${SHISA}/sys/netinet6/mip6.h.

All the structures have the common header defined as the mipm_msghdr{} structure shown in Figure A.2. The miph_msglen variable is the size of each message. The miph_version variable specifies the message version. The value 1 is used at this moment. The miph_type variable is one of the message number shown in table A.1. The miph_pid, miph_seq, and miph_errno are the process ID of the program which sent a message, message sequence number, and error code when processing error occurred respectively. These three fields are not used

```
struct mip_msghdr {
        u_int16_t miph_msglen;  /* message length */
        u_int8_t  miph_version; /* version: future binary compatibility */
        u_int8_t  miph_type;    /* message type */
        pid_t     miph_pid;
        u_int32_t miph_seq;     /* for sender to identify action */
        u_int32_t miph_errno;   /* why failed */
};
```

Figure A.2. The `mipm_msghdr{}` structure.

```
struct mipm_nodetype_info {
        struct mip_msghdr mipmni_hdr;
        u_int8_t mipmni_nodetype;
        u_int8_t mipmni_enable; /* set 1 to enable, 0 to disable */
};
```

Figure A.3. The `mipm_nodetype_info{}` structure

at this moment.

Figure A.3 shows the `mipm_nodetype_info{}` structure. This structure is used with the `MIPM_NODETYPE_INFO` type to enable or disable Mobile IPv6 function. The `mipmni_nodetype` value specifies the desired node type as listed in table A.2. The `mipmni_enable` is either set to 1 to enable the specified node function, or set to 0 to disable the function.

Figure A.4 shows the format of the `mipm_bc_info{}` structure. This structure

Table A.2. The `mipmni_nodetype` values

Name	Value	Description
`MIP6_NODETYPE_NONE`	0x00	The legacy IPv6 node without Mobile IPv6 support
`MIP6_NODETYPE_CORRESPONDENT_NODE`	0x01	The static node with Mobile IPv6 support
`MIP6_NODETYPE_HOME_AGENT`	0x02	The home agent
`MIP6_NODETYPE_MOBILE_NODE`	0x04	The mobile host
`MIP6_NODETYPE_MOBILE_ROUTER`	0x08	The mobile router

```
struct mipm_bc_info {
        struct mip_msghdr mipmci_hdr;
        int mipmci_lifetime;
        u_int16_t mipmci_flags;
        struct sockaddr mipmci_addr[0];
#define MIPC_HOA(mipc)  (&(mipc)->mipmci_addr[0])
#define MIPC_COA(mipc)  ((struct sockaddr *)((char *)(MIPC_HOA(mipc)) \
                                + (MIPC_HOA(mipc))->sa_len))
#define MIPC_CNADDR(mipc)       ((struct sockaddr *)((char *)(MIPC_COA(mipc)) \
                                + (MIPC_COA(mipc))->sa_len))
};
```

Figure A.4. The `mipm_bc_info{}` structure

Table A.3. The flags for `mipmci_flags`.

Name	Value	Description
`IP6_MH_BA_KEYM`	0x80	Key management compatibility specified in [30].

is used with the `MIPM_BC_ADD` and `MIPM_BC_REMOVE` messages to specify the binding cache information to be managed. The contents of this structure are almost the same as the contents of theoretical structure of the binding cache information. The `mipmci_lifetime` variable is the lifetime of the binding. The `mipmci_flags` is the flag value shown in table A.3. The `mipmci_addr[]` contains the home address, care-of address, and peer address of the binding cache information.

Figure A.5 shows the `mipm_bul_info{}` structure. This structure is used with the `MIPM_BUL_ADD` and `MIPM_BUL_REMOVE` messages to specify the binding update information to be managed. The `mipmui_flags` is flags shown in table A.4. The `mipmui_hoa_ifindex` and `mipmui_coa_ifindex` are the interface index of a home address and a care-of address respectively. The `mipmui_state` variable is used to keep internal state that indicates the entry is enabled or not. The `mipmui_addr[]` variable contains the home address, care-of address, and peer address of the relevant binding update list information.

Figure A.6 shows the `mipm_md_info{}` structure. This structure is used with the `MIPM_MD_INFO` message to inform other receivers of the mobility socket of the movement information of the node. The `mipmmi_command` variable specifies

```
struct mipm_bul_info {
        struct mip_msghdr mipmui_hdr;
        u_int16_t       mipmui_flags;
        u_int16_t       mipmui_hoa_ifindex;
        u_int16_t       mipmui_coa_ifindex;
        u_int8_t        mipmui_state;
        u_int8_t        mipmui_reserved[1];
        struct sockaddr mipmui_addr[0];
#define MIPU_HOA(mipu)  (&(mipu)->mipmui_addr[0])
#define MIPU_COA(mipu)  ((struct sockaddr *)((char *)(MIPU_HOA(mipu)) \
                                + (MIPU_HOA(mipu))->sa_len))
#define MIPU_PEERADDR(mipu)     ((struct sockaddr *)((char *)(MIPU_COA(mipu)) \
                                + (MIPU_COA(mipu))->sa_len))
};
```

Figure A.5. The `mipm_bul_info{}` structure

Table A.4. The flags for `mipmui_flags`

Name	Value (BE/LE)	Description
`IP6_MH_BU_ACK`	0x8000/0x0080	Binding Ack message required
`IP6_MH_BU_HOME`	0x4000/0x0040	Home registration.
`IP6_MH_BU_LLOCAL`	0x2000/0x0020	Link-local compatibility specified in [30].
`IP6_MH_BU_KEYM`	0x2000/0x0020	Key management compatibility specified in [30].

the type of movement event as shown in table A.5. The variable `mipmmi_hint` specifies the data type (table A.6) of additional information which follows just after this variable. The `mipmmi_ifindex` variable is valid only when `mipmmi_hist` is `MIPM_MD_INDEX`. It specifies the interface index of the network interface which achieved a new care-of address. The `mipmmi_addr[]` array contains the address information of the mobile node. The `MIPD_HOA()` points the home address of the mobile node, the `MIPD_COA()` points the care-of address of the mobile node (the care-of address will be the same address as the home address when de-registering), and the `MIPD_COA2()` points the local address of the mobile node, when the mobile node is de-registering from a foreign network. By receiving this message, the mobile node's signal processing program will perform the home

```
struct mipm_md_info {
        struct mip_msghdr mipmmi_hdr;
        u_char mipmmi_command;
        u_char mipmmi_hint;
        u_int16_t mipmmi_ifindex;
        struct sockaddr mipmmi_addr[0];
#define MIPD_HOA(mipd)  (&(mipd)->mipmmi_addr[0])
#define MIPD_COA(mipd)  ((struct sockaddr *)((char *)(MIPD_HOA(mipd)) \
                                + (MIPD_HOA(mipd))->sa_len))
#define MIPD_COA2(mipd) ((struct sockaddr *)((char *)(MIPD_COA(mipd)) \
                                + (MIPD_COA(mipd))->sa_len))
};
```

Figure A.6. The `mipm_md_info{} structure.`

Table A.5. The `mipmmi_command` values

Name	Value	Description
`MIPM_MD_REREG`	0x01	Registration or re-registration is required.
`MIPM_MD_DEREGHOME`	0x02	De-registration from home network is required.
`MIPM_MD_DEREGFOREIGN`	0x03	De-registration from foreign network is required.
`MIPM_MD_SCAN`	0x04	Request the kernel to check the reachability of link-local routers.

(de-)registration procedure.

Figure A.7 shows the `mipm_home_hint{}` structure. This is used with the `MIPM_HOME_HINT` type to notify the mobility socket listeners of the fact that the node has attached to the home network. The `mipmhh_ifindex` variable indicates the interface index from which the router advertisement message, including the home network prefix has been received. The `mipmhh_prefixlen` is the prefix length of the received home prefix. The `mipmhh_prefix` variable contains the home prefix. By receiving this message, the movement detection program will decide if it should send `MIPM_MD_INFO` for de-registration.

Figure A.8 shows the `mipm_rr_hint{}` structure. This is used with the `MIPM_RR_HINT` type to notify the mobility socket listeners of the fact the outgoing or incoming packet with a correspondent node is not route-optimized, that

Table A.6. The mipmmi_hint values

Name	Value	Description
MIPM_MD_INDEX	0x01	The mipmmi_ifindex variable contains a valid value.
MIPM_MD_ADDR	0x02	The address information is located in the mipmmi_addr[] array only.

```
struct mipm_home_hint {
        struct mip_msghdr mipmhh_hdr;
        u_int16_t mipmhh_ifindex;               /* ifindex of interface
                                                   which received RA */
        u_int16_t mipmhh_prefixlen;             /* Prefix Length */
        struct sockaddr mipmhh_prefix[0];       /* received prefix */
};
```

Figure A.7. The mipm_home_hint{} structure

means, the packet is sent or received using the IP-in-IP tunnel between the mobile node and its home agent. The mipmmrh_addr[] array contains the source and destination addresses of the packet. By receiving this message, the mobile host signaling processing program may start the return routability procedure.

Figure A.9 shows the mipm_be-hint{} structure. This is used with the MIPM_BE_HINT type to notify the mobility socket listeners of the fact that a mobility protocol processing error has happened in the kernel. In the SHISA system, all the mobility signaling messages are processed in user space. The protocol error occurred in the kernel has to be notified to the user space programs.

```
struct mipm_rr_hint {
        struct mip_msghdr mipmrh_hdr;
        struct sockaddr mipmrh_addr[0];
};
#define MIPMRH_HOA(mipmrh) ((mipmrh)->mipmrh_addr)
#define MIPMRH_PEERADDR(mipmrh)                         \
    ((struct sockaddr *)((char *)(MIPMRH_HOA(mipmrh))   \
    + (MIPMRH_HOA(mipmrh))->sa_len))
```

Figure A.8. The mipm_rr_hint{} structure

```
struct mipm_be_hint {
        struct mip_msghdr mipmbeh_hdr;
        u_int8_t mipmbeh_status;
        u_int8_t mipmbeh_reserved[3];
        struct sockaddr mipmbeh_addr[0];
};
#define MIPMBEH_PEERADDR(mipmbeh) ((mipmbeh)->mipmbeh_addr)
#define MIPMBEH_COA(mipmbeh)                                    \
    ((struct sockaddr *)((char *)(MIPMBEH_PEERADDR(mipmbeh))    \
    + (MIPMBEH_PEERADDR(mipmbeh))->sa_len))
#define MIPMBEH_HOA(mipmbeh)                                    \
    ((struct sockaddr *)((char *)(MIPMBEH_COA(mipmbeh)) \
    + (MIPMBEH_COA(mipmbeh))->sa_len))
```

Figure A.9. The `mipm_be_hint{}` structure

Table A.7. The `mipmbeh_status` values

Name	Value	Description
`IP6_MH_BES_UNKNOWN_HAO`	1	Unverified home address is used.
`IP6_MH_BES_UNKNOWN_MH`	2	Undefined Mobility Header type is used.

The `mipmbeh_status` variable indicates the error status defined as a part of the Binding Error message specified in RFC3775. The values are shown in table A.7. The `mipmbeh_addr[]` array includes the peer address, the care-of address, and the home address which involves in the communication which triggered the error.

Figure A.10 shows the `mipm_dad{}` structure. This is used with the `MIPM_DAD` type to trigger the Duplicate Address Detection (DAD) and receive the DAD

```
struct mipm_dad {
        struct mip_msghdr mipmdad_hdr;
        u_int16_t mipmdad_message;
        u_int16_t mipmdad_ifindex;
        struct in6_addr mipmdad_addr6;
};
```

Figure A.10. The `mipm_dad{}` structure

Table A.8. The mipmdad_message values. Values below 127 are sent from the user space to the kernel space, above 128 are used from the kernel space to the user space.

Name	Value	Description
MIPM_DAD_DO	0	Trigger the DAD procedure for the specified address.
MIPM_DAD_STOP	1	Stop the DAD procedure for the specified address.
MIPM_DAD_LINKLOCAL	2	Trigger the DAD procedure for the specified link-local address.
MIPM_DAD_SUCCESS	128	The DAD procedure succeeded.
MIPM_DAD_FAIL	129	The DAD procedure failed.

process status. The mipmdad_message variable includes the operation code or status code as shown in table A.8. The mipmdad_ifindex variable indicates the interface index of the address being tested. The mipmdad_addr6 variable is the address itself being tested.

3.2 Using Mobility Socket from User Space

The mobility socket is implemented as a variant of the raw socket. The code is implemented in ${NETBSD}/net/mipsock.h and ${NETBSD}/net/mipsock.c. To use the mobility socket, a user program need to open a raw socket with the AF_MOBILITY socket family as follows:

```
s = socket(AF_MOBILITY, SOCK_RAW, 0);
```

Once the socket is available, the user space program can send a mobility socket message as described in section 3.1. As already explained, some of the mobility socket messages are generated in the kernel. Similarly, some of the messages are generated by a user space program and delivered to other user space programs. These messages are delivered to all the mobility socket listeners asynchronously. The user space program must be ready for such asynchronous mobility-related message notification.

Table A.9. The mobility socket supportive functions for the kernel user

Name	Description
`void mips_notify_home_hint(` `u_int16_t` *ifindex*`, struct sockaddr *` *prefix*`, u_int16_t` *prefixlen*`)`	Send a `MIPM_HOME_HINT` message. *ifindex* is the interface index attached to the home network. *prefix* and *prefixlen* are the home prefix and prefix length respectively.
`void mips_notify_rr_hint(` `struct sockaddr *` *hoa*`,` `struct sockaddr *` *peeraddr*`)`	Send a `MIPM_RR_HINT` message. *hoa* is the home address. *peeraddr* is the address of the remote host.
`void mips_notify_be_hint(` `struct sockaddr *` *src*`,` `struct sockaddr *` *coa*`,` `struct sockaddr *` *hoa*`,` `struct sockaddr *` *status*`)`	Send a `MIPM_BE_HINT` message. *src* is the source address of the remote host that caused the protocol error, *coa* and *hoa* is the care-of address and home address of the mobile node. *status* is a status code of the Binding Error message shown in table A.7.

3.3 Using Mobility Socket from Kernel

Using a mobility socket from the kernel is similar to using a routing socket from the kernel. The kernel just calls the `raw_input()` function by specifying the mobility socket protocol family. The kernel programmer can prepare an mbuf containing the mobility socket message by himself, or he can use supportive function provided in `${NETBSD}/sys/net/mipsock.c` file. Table A.9 shows the support function list.

4. Kernel Functions

In this section, the kernel functions related to SHISA are explained. Since the mobility function affects many parts of the kernel core networking function, the code fragments are scattered over the wide range of the networking code.

4.1 Mobility Core Functions

Table A.10 shows the list of input/output functions. The `mip6_input()` function processes incoming mobility header messages. The function checks the validity of the input message and passes it to the user space program using the Exten-

Table A.10. The packet input/output functions in `${NETBSD}/sys/netinet6/mip6.c`.

Name
`int mip6_input(struct mbuf ** mp, int * offp, int proto)`
`int mip6_tunnel_input(struct mbuf ** mp, int * offp, int proto)`
`int mip6_encapsulate(struct mbuf * mm, struct in6_addr * osrc, struct in6_addr * odst)`

sion Socket API defined in RFC4584 [6]. The `mip6_tunnel_input()` function processes the packet sent from the home agent which is encapsulated using the IP-in-IP tunneling. The function decapsulates the packet, and re-input the internal packet to the IPv6 input function. The `mip6_encapsulate()` function is used when a mobile node is using the bi-directional tunnel connection between the node and its home agent. All the packets sent from the mobile node are encapsulated by the IP-in-IP tunneling mechanism.

Table A.11 shows the binding cache management functions. The `mip6_bce_get()` function searches an existing binding cache entry, or creates it if there is no matching entry, by calling the `mip6_bce_new_entry()` and `mip6_bc_list_insert()` functions. The `mip6_bce_update()` function updates the existing binding cache entry with the latest information. The `mip6_bc_proxy_control()` function adds or removes the proxy neighbor discovery entry to intercept the traffic sent to mobile nodes at the home agent. The *target* argument is the mobile node's address, the *local* argument is the home agent's address, and the *cmd* argument is either `RTM_DELETE` or `RTM_ADD` macro which is passed to the routing manipulation function. The `mip6_bce_remove_bc()` and `mip6_bce_remove_all()` functions remove the existing binding cache entries. The `mip6_bc_list_remove()` function is called as a sub-function on removal.

Table A.12 shows the binding update management functions. The `mip6_bul_get()` function finds an existing binding update entry or creates a new entry if there is no matching entry by calling the `mip6_bul_create()` function. The `mip6_bul_get_home_agent()` function is a special function to return only the binding update entry which is used for the home registration. The `mip6_bul_add()` function updates the existing binding update entry with the latest information. The `mip6_bul_remove()` and `mip6_bul_remove_all()` functions remove the binding update entry.

Table A.11. The binding cache management functions in ${NETBSD}/sys/netinet6/mip6.c

Name
struct mip6_bc_internal *mip6_bce_get(struct in6_addr * *hoa*, struct in6_addr * *cnaddr*, struct in6_addr * *coa*, u_int16_t *bid*)
static struct mip6_bc_internal *mip6_bce_new_entry(struct_in6_addr * *cnaddr*, struct in6_addr * *hoa*, struct in6_addr * *coa*, struct ifaddr * *ifa*, u_int16_t *bid*)
static void mip6_bc_list_insert(struct mip6_bc_internal * *mbc*)
int mip6_bce_update(struct sockaddr_in6 * *cnaddr*, struct sockaddr_in6 * *hoa*, struct sockaddr_in6 * *coa*, u_int16_t *flags*, u_int16_t *bid*)
int mip6_bc_proxy_control(struct in6_addr * *target*, struct in6_addr * *local*, int *cmd*)
int mip6_bce_remove_addr(struct sockaddr_in6 * *cnaddr*, struct sockaddr_in6 * *hoa*, struct sockaddr_in6 * *coa*, u_int16_t *flags*, u_int16_t *bid*)
void mip6_bce_remove_bc(struct mip6_bc_internal * *mbc*)
void mip6_bce_remove_all(void)
void mip6_bc_list_remove(struct mip6_bc_internal * *mbc*)

Table A.13 shows IPv6 header manipulation functions. The `mip6_create_rthdr2()` function prepares the Routing Header Type 2 when a correspondent node is communicating with a mobile node with route optimized path. The `mip6_create_hoa_opt()` function prepares the Home Address destination option which is used when a mobile node is communicating with a correspondent node with route optimized path.

Table A.14 shows functions related to event notification. The `mip6_notify_rr_hint()` function is used when the kernel notices input or output of the bi-directional packet between a mobile node and its home agent. The function eventually sends the `MIPM_RR_HINT` message to the mobility socket listeners. The `mip6_md_scan()` function checks the reachability of routers attached to the network interface specified by the *ifindex* argument. This is called when a user space program sends the `MIPM_MD_INFO` message with the scan option to the mobility socket. The `mip6_do_dad()`, `mip6_stop_dad()` functions start and stop the DAD procedure. The functions are called when a user space program sends the `MIPM_DAD` message. The `mip6_do_dad_lladdr()` function is a special function prepared for the DAD procedure of the link-local address. With this function, a user space program can

Table A.12. The binding update management functions in ${NETBSD}/sys/netinet6/mip6.c

Name
struct mip6_bul_internal *mip6_bul_get(struct in6_addr * *src*, struct in6_addr * *dst*, u_int16_t *bid*)
struct mip6_bul_internal *mip6_bul_get_home_agent(struct in6_addr * *src*)
static struct mip6_bul_internal *mip6_bul_create(struct in6_addr * *peeraddr*, struct in6_addr * *hoa*, struct in6_addr * *coa*, u_int16_t *flags*, u_int8_t *state*, struct mip_softc * *sc*, u_int16_t *bid*;
int mip6_bul_add(struct in6_addr * *peeraddr*, struct in6_addr * *hoa*, struct in6_addr * *coa*, u_short *hoa_ifindex*, u_int16_t *flags*, u_int8_t *state*, u_int16_t *bid*)
void mip6_bul_remove(struct mip6_bul_internal * *mbul*)
void mip6_bul_remove_all(void)

Table A.13. The header management functions in ${NETBSD}/sys/netinet6/mip6.c

Name
struct ip6_rthdr2 *mip6_create_rthdr2(struct in6_addr * *coa*)
u_int8_t *mip6_create_hoa_opt(struct in6_addr * *coa*)

start the DAD procedure of a link-local address by specifying the interface index, without knowing the exact link-local information.

SHISA provides a virtual interface used for the Mobile IPv6 operation, called *mip* virtual interface. The implementation files are located in ${NETBSD}/sys/net/if_mip.[hc]. The main role of the virtual interface is to keep home prefix information and home address information of the mobile node. The virtual interface also provides the function to encapsulate packets from a mobile node to its home agent, when the mobile node is communicating with a correspondent node which does not support Mobile IPv6. Table A.15 shows the featured functions provided by the virtual interface. The mip6_outout() function is an encapsulation function used to send packets to correspondent nodes via the home agent of the mobile node.

Table A.14. The notification functions in ${NETBSD}/sys/netinet6/mip6.c

Name
void mip6_notify_rr_hint(struct in6_addr * *dst*, struct in6_addr * *src*)
void mip6_md_scan(u_int16_t *ifindex*)
void mip6_do_dad(struct in6_addr * *addr*, int *ifidx*)
void mip6_stop_dad(struct in6_addr * *addr*, int *ifidx*)
void mip6_do_dad_lladdr(int *ifidx*)

Table A.15. The mip virtual interface functions

Name
int mip_output(struct ifnet * *ifp*, struct mbuf * *m*, struct sockaddr * *dst*, struct rtentry * *rt*)

4.2 Destination Options Functions

The Destination Options header processing is modified to support Mobile IPv6. Table A.16 shows functions related to Mobile IPv6 processing. The dest6_input() function is extended to support the processing of the Home Address destination option. The function checks if the input packet contains at least a home address destination option or not, and if it is a valid home address or not. If the packet is a Binding Update message, then the function passes the packet to the IPsec processing performed later to verify if the packet is valid or not. Otherwise, the function checks if the home address is a registered home address in the binding cache entry in the receiving node. If there is a related binding cache entry, then the home address is accepted, otherwise, the later home address processing code in the IPv6 input routine judges the packet. The dest6_swap_hao()

Table A.16. The support functions in ${NETBSD}/sys/netinet6/dest6.c

Name
int dest6_input(struct mbuf ** *mp*, int * *offp*, int *proto*)
static int dest6_swap_hao(struct ip6_hdr * *ip6*, struct ip6aux * *ip6a*, struct ip6_opt_home_address * *haopt*)
int dest6_mip6_hao(struct mbuf * *m*, int *mhoff*, int *nxt*)
static void mip6_notify_be_hint(struct in6_addr * *src*, struct in6_addr * *coa*, struct in6_addr * *hoa*, u_int8_t *status*)

Table A.17. The support functions in ${NETBSD}/sys/netinet6/route6.c

Name
`static int ip6_rthdr2(struct mbuf *` *m*`, struct ip6_hdr *` *ip6*`, struct ip6_rthdr2 *` *rh6*`,`

function swaps the source address of the incoming IPv6 packet and the address included in the home address option field if the home address is verified. By swapping these addresses, the change of the care-of address is hidden to the upper layer programs. The `dest6_mip6_hao()` function is used by the IPv6 input function to check if the home address option is valid or not. If it is not valid, the `mip6_notify_be_hint()` function is called to notice the user space signal processing program to handle the error case. If it is unsure, the packet processing is passed to the IPsec function. If the IPsec verification succeeds, the home address is considered verified.

4.3 Routing Header Functions

The Mobile IPv6 specification defines a new Routing Header type (type 2) for Mobile IPv6 route optimized communication. SHISA implements the handling code in `${NETBSD}/sys/netinet6/route6.c`. The `ip6_rthdr2()` function described in table A.17 processes the type 2 Routing Header. The type 2 Routing Header has the following limitations.

- All the addresses contained in the header must belong to the mobile node processing the header.
- The number of addresses contained in the header must be one.

The function checks the above conditions and discards the packet if the packet does not satisfy them.

4.4 IPv6 Functions

The core IPv6 processing part is also modified to support Mobile IPv6. Table A.18 shows features IPv6 functions modified for Mobile IPv6. The `ip6_input()` function handles the general IPv6 input work. For mobility support, the function

Table A.18. The IPv6 core function support

Name
`void ip6_input(struct mbuf * m)`
`int ip6_output(struct mbuf * m0, struct ip6_pktopts * opt, struct route * ro, int flags, struct ip6_moptions * im6o, struct socket * so, struct ifnet ** ifpp)`
`int nd6_output(struct ifnet * ifp, struct ifnet * origifp, struct mbuf * m0, struct sockaddr_in6 * dst, struct rtentry * rt0)`
`struct in6_addr *in6_selectsrc(struct sockaddr_in6 * dstsock, struct ip6_pktopts * opts, struct ip6_moptions * mopts, struct route * ro, struct in6_addr * laddr, struct ifnet ** ifpp, int * errorp)`
`void nd6_ra_input(struct mbuf m, int off, int imp6len)`
`static int prelist_update(struct nd_prefixctl * new, struct nd_defrouter * dr, struct mbuf * m, int mcast)`
`void pfxlist_onlink_check(void)`
`void ip6_forward(struct mbuf * m, int srcrt)`

checks the validity of the home address option while checking the extension headers by using the `dest6_mip6_hao()` function discussed in section 4.2. If the verification fails, the incoming packet is dropped. The `ip6_output()` function is extended to support Mobile IPv6 related extension headers. If a node has a valid binding update entry, and the outgoing packet matches the binding update entry, then the function adds a home address option as a destination options header by using the `mip6_create_hoa_opt()` function discussed in section 4.1. Similarly, if a node has a valid binding cache entry, and the outgoing packet matches the entry, then a type 2 routing header is added by the `mip6_create_rthdr2()` function discussed in section 4.1.

The `nd6_output()` function is extended to support the bi-directional encapsulation from a mobile node to its home agent. If the node is communicating with a correspondent node which does not support Mobile IPv6, the `nd6_output()` function calls the encapsulation `mip_output()` discussed in section 4.1.

The `in6_selectsec()` function supports the new source address selection rules which reflect the Mobile IPv6 policy. If a node has a valid home address, then the source address of the outgoing packet should be the home address. The function assigns the home address if the source address of the outgoing packet is unspecified.

The `nd6_ra_input()` function processes a router advertisement message. For movement detection, the function is extended to check the reachability of the on-link routers when the node receives a router advertisement. The `prelist_update()` and `pfxlist_onlink_check()` functions are subsequently called to update the on-link prefix status. If the prefix status changes, for example, if a prefix previously on-link becomes off-link, the notification is sent to the user space program through the routing socket. In the use space program, the notification is seen as the status change of the care-of address. If the care-of address in use becomes off-link, then the movement detection procedure starts.

The `ip6_forward()` function is extended to support the bi-directional tunnel from a home agent to mobile nodes. If a node is acting as a home agent and receives a packet sent to the mobile node which the home agent is serving, the packet is passed to the `ip6_forward()` function based on the normal IPv6 packet processing. In the function, the home agent checks if there is any binding cache entry matching the incoming packet, and if it is found the function calls the `mip6_encapsulate()` function discussed in section 4.1 to tunnel the packet to mobile node.

5. Mobile Host Program

The mobile host program consists of two user space programs. One is the Mobile IPv6 mobile host-related signal packet handing program `mnd`, and the other is a movement detection program `babymdd`. In this section, the former program is explained.

5.1 Structures

Figure A.11 is the `binding_update_list{}` structure used to keep binding update entry in a mobile node. There are some variables which are not in the conceptual binding update entry structure. The `bul_bu_lastsent` variable keeps the time when the latest binding update message was sent to the home agent. The `bul_hoainfo` variable keeps the home address information of the binding update entry. The `bul_home_ifindex` variable is the interface index through which the mobile node attaches to the home network when it returns to home. The

```
struct binding_update_list {
        LIST_ENTRY(binding_update_list) bul_entry;
        struct in6_addr     bul_peeraddr;   /* peer addr of this BU */
        struct in6_addr     bul_coa;        /* CoA */
        u_int16_t           bul_lifetime;   /* BU lifetime */
        u_int16_t           bul_refresh;    /* refresh frequency */
        u_int16_t           bul_seqno;      /* sequence number */
        u_int16_t           bul_flags;      /* BU flags */
        time_t              bul_bu_lastsent; /* The last time when mn sent the BU */
        mip6_cookie_t       bul_home_cookie;
        u_int16_t           bul_home_nonce_index;
        mip6_token_t        bul_home_token;   /* home keygen token */
        mip6_cookie_t       bul_careof_cookie;
        u_int16_t           bul_careof_nonce_index;
        mip6_token_t        bul_careof_token; /* careof keygen token */
        struct mip6_hoainfo *bul_hoainfo; /* backpointer to hoa_info */
        int                 bul_home_ifindex; /* ifindex of home network */
        u_int8_t            bul_reg_fsm_state;/* registration state */
        u_int8_t            bul_rr_fsm_state; /* rr state */
        CALLOUT_HANDLE      bul_retrans;      /* callout handle for retrans */
        u_int8_t            bul_retrans_time;
        CALLOUT_HANDLE      bul_expire;       /* callout handle for failure */
        u_int8_t            bul_state;        /* local status */
};
```

Figure A.11. The `binding_update_list{}` structure

`bul_reg_fsm_state` and `bul_rr_fsm_state` keep the state of the state machine as discussed in section 4.7 in chapter 5. The `bul_retrans` and `bul_expire` are the timer entry for the message retransmission and binding information expiration respectively. The `bul_retrans_time` is the current retransmission time in a unit of second. The `bul_state` variable indicates that the entry is valid or not.

Figure A.12 shows the `mip6_hoainfo{}` structure. The `hinfo_hoa` variable is the home address. The `hinfo_ifindex` variable is the interface index of the mip virtual interface on which the home address is assigned. The `hinfo_location` variable is the current location of the mobile node as shown in table A.19. The `hinfo_dhaad_id`, `hinfo_dhaad_lastsent`, `hinfo_mps_id`,

```
struct mip6_hoainfo {
        LIST_ENTRY(mip6_hoainfo) hinfo_entry;
        struct in6_addr hinfo_hoa; /* Home Address */
        /* if_index of mip virtual interface where HoA is assigned */
        u_int16_t hinfo_ifindex;
        u_int8_t  hinfo_location;   /* Location where mn is located */
        u_int16_t hinfo_dhaad_id;
        time_t    hinfo_dhaad_lastsent;
        u_int16_t hinfo_mps_id;
        time_t    hinfo_mps_lastsent;
        struct binding_update_list_head hinfo_bul_head;         /* Binding Updat
e List */
};
```

Figure A.12. The `mip6_hoainfo{}` structure

Table A.19. The `hinfo_location` values

Name	Value	Description
`MNINFO_MN_UNKNOWN`	0x00	Location is unknown.
`MNINFO_MN_HOME`	0x01	Locatin is home.
`MNINFO_MN_FOREIGN`	0x02	Location is foreign.

and `hinfo_mps_lastsent` are the identification number and last sent time of the Dynamic Home Agent Address Discovery message and Mobile Prefix Solicitation message respectively.

5.2 Behavior

When the `mnd` program is launched, it first collects all the home address information from the mip virtual interfaces. After that, the program waits for a message from the movement detection program to identify the current location of the node. If the current location is home, then the program stays until it moves to a foreign network. If the current location is a foreign network, then it triggers the movement procedure based on the message sent from the movement detection program.

The main loop of the `mnd` program is to listen opened sockets and process

Table A.20. The mobility header processing functions used by `mnd`

Name
`int mh_input_common(int` *fd*`)`
`int mh_input()` is an alias of `bul_kick_fsm_by_mh()`
`int bul_kick_fsm_by_mh(struct in6_addr *` *src*`, struct in6_addr *` *dst*`, struct in6_addr *` *hoa*`, struct in6_addr *` *rtaddr*`, struct ip6_mh *` *mh*`, int` *mhlen*`)`
`int bul_kick_fsm(struct binding_update_list *` *bul*`, int` *event*`, struct fsm_message *` *data*`)`
`int bul_reg_fsm(struct binding_update_list *` *bul*`, int` *event*`, struct fsm_message` *data*`)`
`int bul_rr_fsm(struct binding_update_list *` *bul*`, int` *event*`, struct fsm_message` *data*`)`
`int send_hoti(struct binding_update_list *` *bul*`)`
`int send_coti(struct binding_update_list *` *bul*`)`
`int send_bu(struct binding_update_list *` *bul*`)`
`int send_be(struct in6_addr *` *dst*`, struct in6_addr *` *src*`, struct in6_addr *` *home*`, u_int8_t` *status*`)`

the incoming messages from the sockets. The `mnd` program opens the following sockets.

- Mobility header socket
- Mobility socket
- ICMPv6 socket

The mobility header socket is used to send or receive the Mobile IPv6 signaling messages. The socket extension is defined in RFC4584 [6]. The mobility socket is the core interface provided by the SHISA system to exchange mobility related information among mobility programs. The ICMPv6 socket is used to receive ICMPv6 error messages necessary for Mobile IPv6 processing.

5.3 Mobility Header Processing

Table A.20 shows the mobility header processing functions. The `mh_input_comon()` function is a common mobility header message processing function. The function

performs the basic validation of the incoming message and extract data such as the source and destination addresses of the message, the home address option and the routing header address, and the mobility header message contents itself.

After the common processing, the message is passed to the `mh_input()` function, which is the same function as the `bul_kick_fsm_by_mh()` function in the `mnd` program case. The most of the mobile node data processing is done in the state machine. The state machine details can be checked in section 4.7 in chapter 5. The state machine is for the KAME Mobile IPv6 implementation, however, the state machine itself is working in the same logic in the SHISA version too. The `bul_kick_fsm_by_mh()` analyzes the mobility header contents and translates them to the event (table 5.6 in chapter 5) which can be recognized by the state machine. The `bul_kick_fsm()` function is the actual state machine implementation. The translated event is eventually put to the function. If the event is related to the binding registration processing, the `bul_reg_fsm()` function is called. If the event is related to the return routability procedure, then the `bul_rr_fsm()` is called instead. These fsm functions will call other functions, such as registering the binding information in the kernel, or sending another mobility signaling message based on the state machine transition diagram as shown in figure 5.5 and 5.6 in chapter 5.

The `send_hoti()` and `send_coti()` functions send a Home Test Init message and Care-of Test Init message respectively. They are called when the `mnd` program receives the `MIPM_RR_HINT` message through the mobility socket, to initiate the return routability procedure. The receive function of the Home Test and the Care-of Test messages are implemented as a part of the `bul_kick_fsm_by_mh()` function.

The `send_bu()` function sends a Binding Update message. If the binding update entry passed by the argument is a home registration entry, then the created Binding Update message has a home registration flag and sent to the home agent of the mobile node. If the binding update entry is for a correspondent node, then the authentication value is calculated based on the Home Test/Care-of Test message exchange, and sent to the correspondent node.

The `send_be()` function sends a Binding Error message. This function is called when a protocol error is found during the message processing, or when the

Table A.21. The mobility socket processing functions used by `mnd`

Name
`int mipsock_input(struct mip_msghdr *` *miphdr*`)` (defined in `${SHISA}/mnd.c`)
`int mipsock_recv_mdinfo(struct mip_msghdr *` *miphdr*`)`
`int mipsock_recv_rr_hint(struct mip_msghdr *` *miphdr*`)`
`int mipsock_bul_request(struct binding_update_list *` *bul*`, u_char` *command*`)`
`int mipsock_md_dereg_bul(struct in6_addr*` *hoa*`, struct in6_addr*` *coa*`, u_int16_t` *ifindex*`)`
`int mipsock_nodetype_request(u_int8_t` *nodetype*`, u_int8_t` *enable*`)`

`MIPM_BE_HINT` message is received through the mobility socket which indicates a protocol error occurred in the kernel space.

5.4 Mobility Socket Processing

Table A.21 shows the mobility socket-related functions. The `mipsock_input()` function is an entry point of the mobility socket messages or the `mnd` program. Note that the reader should not be confused with this name because the same name is used in other programs too.

The `mipsock_recv_mdinfo()` function processes the `MIPM_MD_INFO` message sent from the movement detection program. Based on the information, the `mnd` program starts the registration or re-registration procedure. In case that the movement information indicates that the node is home, then the de-registration procedure is triggered.

The `mipsock_bul_request()` function adds/updates or removes the kernel binding update information through the mobility socket. The *command* argument is either `MIPM_BUL_ADD` or `MIPM_BUL_REMOVE`.

The `mipsock_md_dereg_bul()` function actually does not send any mobility message. It processes the home de-registration tasks when the mobile node returns to home, and send the event indicating that the node is home to the state machine. Eventually, the state machine removes the related binding update information by the `mipsock_bul_request()` function.

Table A.22. The ICMPv6 processing functions used by `mnd`

Name
`int icmp6_input_common(int` *fd*`)`
`int send_haadreq(struct mip6_hoainfo *` *hoainfo*`, int` *hoa_plen*`, struct in6_addr *` *src*`)`
`int receive_hadisc_reply(struct mip6_dhaad_rep *` *dhrep*`, size_t` *dhrep_len*`)`
`int send_mps(struct mip6_hpfxl *` *hpfx*`)`
`int receive_mpa(struct mip6_prefix_advert *` *mpa*`, size_t` *mpalen*`, struct binding_update_list *` *bul*`)`

5.5 ICMPv6 Processing

Table A.22 shows the ICMPv6-related functions. The `icmp6_input_common()` function receives the incoming ICMPv6 message and performs the basic validation process. The function handles four ICMPv6 messages. The ICMPv6 Destination Unreach and ICMPv6 Parameter Problem messages are put into the state machine, since they affect the binding information status. The Dynamic Home Agent Address Discovery reply and Mobile Prefix Advertisement message are handled by the `receive_hadisc_reply()` and `receive_mpa()` functions respectively.

The `send_haadreq()` function is used when a mobile node sends a Dynamic Home Agent Address Discovery message. This message is sent in two cases. The first case is the initial boot time. If the mobile node does not know the address of the home agent, the state machine calls this function when it needs to perform the home registration procedure. The second case is when the home agent becomes unavailable. The state machine sends the message to find a new home agent, after 3 times trial to contact to the home agent recorded in the binding update entry.

The `send_mps()` function sends a Mobile Prefix Solicitation message to find/update the home network prefix information, while the mobile node is away from home.

6. Movement Detection

In this section the movement detection program `babymdd` is explained.

```
struct mdd_info {
        int debug;
        int dns;
        int nondaemon;
        int rtsock;
        int mipsock;
        int linkpoll;
        int linksock;
        int whereami;
        struct if_info *coaif;
        LIST_HEAD(, if_info) ifinfo_head;
        u_int16_t hoa_index;
        LIST_HEAD(, hoa_info) hoainfo_head;
};
```

Figure A.13. The `mdd_info{}` structure

Table A.23. The `whoami` values

Name	Value	Description
IAMHOME	1	The node is at home.
IAMFOREIGN	2	The node is at foreign network.

6.1 Structures

Figure A.13 shows the `mdd_info{}` structure. All the address information necessary to perform the movement detection procedure are kept or linked from this structure. The `debug`, `dns`, and `nodaemon` variables are for debugging purpose. The `rtsock` variable contains the socket descriptor of the routing socket to monitor address status changes. The `mipsock` variable contains the socket descriptor of the mobility socket. The `linkpoll` variable indicates if the `babymdd` program intensively checks the link status. If this is set to 1, the program periodically checks the reachability of the on-link routers. The `linksock` variable is a socket descriptor to receive link status information. The `whereami` variable indicates the current location of the node as shown in table A.23. The `coaif` variable indicates the interface information of the current chosen care-of address. The `ifinfo_head` variable contains the list of the candidate interfaces of care-of addresses. The `hoa_index` variable is the interface index attached to the home

```
struct hoa_info {
        LIST_ENTRY(hoa_info) hoainfo_entry;

        struct sockaddr_storage hoa;/* HoA */
};
```

Figure A.14. The `hoa_info{}` structure

```
struct if_info {
        LIST_ENTRY(if_info) ifinfo_entry;
        char ifname[IFNAMSIZ];
        u_char iftype;
        u_int16_t ifindex;
        struct sockaddr_storage coa;/* Current CoA */
        struct sockaddr_storage pcoa;/* Previous CoA */
        int priority;
        time_t lastsent;
        int linkstatus;
        int bid;
};
```

Figure A.15. The `if_info{}` structure

network when the mobile node is at home. The `hoainfo_head` variable contains the list of home addresses.

Figure A.14 shows the `hoa_info{}` structure. The `hoa` variable contains the home address.

Figure A.15 shows the `if_info{}` structure. This structure contains the candidate of the care-of address and some additional information per interface. The `coa` and `pcoa` variables keep the current candidate care-of address and previous candidate of care-of address of the interface. The `priority` variable is the interface priority used to determine which care-of address should be used when there are multiple available interfaces. The `lastsent` variable is the last time when the movement detection program sent the probe packet to the link. The `linkstatus` variable indicates the current link status (such as UP or DONW).

Table A.24. The routing socket related functions used by `babymdd`

Name
`int baby_rtmsg(struct rt_msghdr * rtm, int msglen)`
`void baby_getifinfo(strct if_info * ifinfo)`

6.2 Behavior

The `babymdd` program first gather all the home address information from the mip virtual interfaces and the information from all the physical interfaces, and put the information to the `mdd_info{}` structure.

In the main loop, the `babymdd` program listens to the routing socket and mobility socket for any incoming messages.

6.3 Routing Socket Processing

Table A.24 shows the main routing socket functions.

The `rtsock_rtmsg()` function handles three routing socket messages. One is the `RTM_NEWADDR` message, which is sent when a new address is assigned to an interface or the existing address information is changed. The second message is the `RTM_DELADDR` message sent when an address is removed from an interface. The third message is the `RTM_ADDRINFO` message, which is extended by SHISA to notify the address reachability status change.

When the function received the `RTM_NEWADDR` message, the `babymdd` program updates the address information kept in the `mdd_info{}` structure.

When the `RTM_DELADDR` message is received, the function checks if the removed address is a home address or not. If the mobile node is at home and moves to another foreign network, the home address is removed from the physical interface to the mip virtual interface. If the removed address is the home address, and the interface is a physical interface, then the program updates its location (the `whoami` variable of the `mdd_info{}` structure) from home to unspecified location. If the removed address is the care-of address in use, then the program invalidates the current care-of address information and update all the care-of address candidate addresses.

When the `RTM_ADDRINFO` message is received, the program checks if the status

Table A.25. The mobility socket related functions used by `babymdd`

Name
`int baby_mipmsg(struct mip_msghdr * mipm, int msglen)`
`int baby_md_reg(struct sockaddr * coa, int ifindex, u_int16_t bid)`
`int baby_md_home(struct sockaddr * hoa, struct sockaddr_in6 * coa, int ifindex)`
`int baby_md_scan(struct if_info * ifinfo)`

change affects the validity of the current care-of address in use. If it does, the program invalidate the current care-of address.

The new care-of address selection is done after returning from the `baby_rtmsg()` function.

The `baby_getifinfo()` function extract the address information from the routing socket message and copies the information to the *ifinfo* argument.

6.4 Mobility Socket Processing

Table A.25 shows the mobility socket functions used in `babymdd`. The `baby_mipmsg()` function handles two mobility socket messages. One is the `MIPM_HOME_HINT` message which indicates the node is attached to the home network. In this case the function invalidates the current care-of address and sets the location to home. The other message is the `MIPM_MD_SCAN` message. When the function receives this message, it calls `send_rs()` function to request the on-link routers to send a Router Advertisement message. On receipt of the Router Advertisement message, the kernel starts the reachability confirmation procedure for the on-link routers as discussed in section 4.4.

The `baby_md_reg()` function sends the `MIPM_MD_INFO` message to inform the mobile node signal processing program that there is a movement event. The `baby_md_home()` function sends the `MIPM_MD_INFO` message with de-registration option to inform the mobile node signal processing program that the node is at home now. These functions are called as a result of the care-of address selection discussed in section 6.6.

The `baby_md_scan()` function sends the `MIPM_MD_INFO` message with the scan option. The message is received by the `babymdd` program eventually, causing the

Table A.26. The interface scanning functions used by `babymdd`

Name
`static int send_rs(struct if_info * ifinfo)`

Table A.27. The care-of address selection function used by `babymdd`

Name
`void baby_selection(void)`

function call of the `send_rs()` function.

6.5 Interface Scanning

Table A.26 shows the interface scanning function `send_rs()`. The function is called as a result of the `MIPM_MD_INFO` with the scan option processing, or from the main loop if periodical interface scanning is specified. The function simply constructs a Router Solicitation message and sends it to the all-nodes multicast address of the specified interface.

6.6 Care-of Address Selection

Table A.27 shows the care-of address function `baby_selection()`. The function first checks the `whereami` variable of the `mdd_info{}` structure. If the location is home, then it calls the `baby_md_home()` function to send the `MIPM_MD_INFO` message indicating that the node returns to home. If the location is not home (that means, it is foreign or unknown), and the care-of address is not determined or invalidated by other functions, then the function checks all the interfaces and pick one which has the most highest priority value. Once the new care-of address has been chosen, the `baby_md_reg()` function is called to send the `MIPM_MD_INFO` message to indicate the node's care-of address is changed.

7. Home Agent Program

In this section the home agent program `had` is explained.

```
struct home_agent_list {
        LIST_ENTRY(home_agent_list) hal_entry;
        /* common for mobile node and home agent */
        struct in6_addr hal_lladdr;  /* Link-local address of HA */
        struct in6_addr hal_ip6addr; /* global IPv6 address of HA */
        int             hal_flag;
        /* HA exclusive field: it is used when ha receives RA */
        struct mip6_halist_ha_exclusive {
                u_int16_t halist_lifetime;   /* Remaining lifetime */
                u_int16_t halist_preference; /* Preference for this HA */
        } hal_for_ha;
        CALLOUT_HANDLE        hal_expire;   /* callout handle for expiration */
};
```

Figure A.16. The `home_agent_list{}` structure

Table A.28. The `hal_flag` values

Name	Value	Description
`MIP6_HAL_OWN`	0x01	The entry is the node itself.
`MIP6_HAL_STATIC`	0x02	Manually configured entry.
`MIP6_HAL_RA`	0x04	The entry created by a Router Advertisement message.

7.1 Structures

Figure A.16 shows the `home_agent_list{}` structure. This structure is also used by the `mnd` program to keep home agent information. The `hal_lladdr` and `hal_ip6addr` variables are the link-local address and global address of the home agent. The `hal_flag` variable indicates how the entry was created. Table A.28 shows the possible values of the flag. The `halist_lifetime` and `halist_preference` variables are taken from the home agent information option included in a Router Advertisement message sent from another home agent. These files are valid only when the entry is created as a result of the receipt of a Router Advertisement message. The `hal_expire` variable manages entry expiration timeout.

Figure A.17 shows the `binding_cache{}` structure. The meanings of the variables which do not exist in the conceptual binding cache entry are as shown below. The `bc_realcoa` variable is meaningful only when the Alternate Care-of

```
struct binding_cache {
        LIST_ENTRY(binding_cache) bc_entry;
        struct in6_addr       bc_hoa;       /* peer home address */
        struct in6_addr       bc_coa;       /* peer coa */
        struct in6_addr       bc_realcoa;
        struct in6_addr       bc_myaddr;    /* my addr */
        u_int8_t              bc_state;     /* state of this bce */
        u_int16_t             bc_flags;     /* recved BU flags */
        u_int16_t             bc_seqno;     /* recved BU seqno */
        u_int32_t             bc_lifetime;  /* recved BU lifetime */
        time_t                bc_expire;    /* expiration time of this BC. */
        CALLOUT_HANDLE        bc_refresh;   /* callout handle for retrans */
        u_int8_t              bc_refresh_count;
        void                  *bc_dad;      /* dad handler */
        struct binding_cache *bc_llmbc;
        struct binding_cache *bc_glmbc;
        u_int32_t             bc_refcnt;
};
```

Figure A.17. The `binding_cache{}` structure

Address option is used in a Binding Update message. In this case, `bc_realcoa` will keep the original care-of address stored in the source address field of the Binding Update message. The `bc_state` indicates the current status of the entry as shown in table A.29. The `bc_refresh_count` variable keeps the number of Binding Refresh Request messages sent since last Binding Update message, and used to calculate the backoff timeout of re-sending of a Binding Refresh Request message. The `bc_dad` variable keeps a pointer to the address structure, which is being investigated by the DAD process. The `bc_llmbc` and `bc_glmbc` are pointers

Table A.29. The `bc_state` values

Name	Value	Description
`BC_STATE_VALID`	0	The entry is valid.
`BC_STATE_UNDER_DAD`	1	The DAD procedure in progress.
`BC_STATE_UNDER_AUTH`	2	The authentication procedure in progress.
`BC_STATE_DEPRECATED`	4	The entry is deprecated.

to the other binding cache entries. When a mobile node registers a global address with the link compatibility option set, the home agent must create two binding cache entries, one for the global address, and the other is the link-local address where the interface ID is the same as the global address. These pointers connect these related entries. The `bc_refcnt` variable indicates the number of refers of the entry.

7.2 Behavior

When the `had` program is launched, it opens the following sockets and wait for incoming messages from these sockets.

- Mobility header socket
- Mobility socket
- ICMPv6 socket

The mobility header socket is used to send or receive Mobile IPv6 signaling messages. The mobility socket is used to interact with the kernel Mobile IPv6 function such as adding/removing binding cache entries. The ICMPv6 socket is used to receive Router Advertisement messages from other on-link routers, and to receive Mobile Prefix Solicitation message from mobile nodes.

7.3 Mobility Header Processing

Table A.30 shows the mobility header processing functions used by `had`. The `mh_input()` function processes the Binding Update message only in case of the home agent.

The processing of a Binding Update message is done by the `receive_bu()` function. In the home agent case, the Binding Update message is protected by the IPsec mechanism. Since the kernel drops illegal Binding Update message before delivering the message to user space, there is no further validation check necessary. If there is an existing binding cache entry on the home agent, then the `had` program updates the entry based on the message content. If this is the first Binding Update message from the mobile node, then a new binding cache entry

Table A.30. The mobility header processing functions used by `had`

Name
`int mh_input(struct in6_addr *` *src*`, struct in6_addr *` *dst*`, struct in6_addr *` *hoa*`, struct in6_addr *` *rtaddr*`, struct ip6_mh *` *mh*`, int` *mhlen*`)`
`int receive_bu(struct in6_addr *` *src*`, struct in6_addr *` *dst*`, struct in6_addr *` *hoa*`, struct in6_addr *` *rtaddr*`, ip6_mh_binding_update *` *bu*`, int` *mhlen*`)`
`int send_ba(struct in6_addr *` *src*`, struct in6_addr *` *coa*`, struct in6_addr *` *acoa*`, struct in6_addr *` *hoa*`, u_int16_t` *flags*`, mip6_kbm_t *` *kbm_p*`, u_int8_t` *status*`, u_int16_t` *seqno*`, u_int16_t` *lifetime*`, int` *refresh*`, u_int16_t` *bid*`)`
`send_brr(struct in6_addr *` *src*`, struct in6_addr *` *dst*`)`

is created though the mobility socket. The new binding cache entry creation also starts the neighbor discovery proxy function in the kernel level as discussed in section 4.1. The `had` program performs the DAD procedure before enabling a binding cache entry. The process is invoked through the mobility socket described in section 7.4.

If the Binding Update message is for de-registration, that is, the lifetime field of the message is zero, or the care-of address and the home address in the message is same, then the existing binding cache entry is removed, and the proxy function for the mobile node is stopped.

Once the Binding Update message is accepted, and the sender of the message requests an acknowledgment message, the `send_ba()` function is called to send a Binding Acknowledgement message. This message is sent after the DAD procedure completes.

As shown in figure A.17, each binding cache entry has a timer pointer, which is used to send a Binding Refresh Request message to the registrar to remind the peer of the fact that the binding cache is going to expire. In the mobile node case, these timeout handling are done in the state machine implementation. In the binding cache case, these timer handling are done on every update of a binding cache and handled as a part of the main loop of the `had` program.

The `send_brr()` function creates a Binding Refresh Request message and sends it to the specified destination address. If the `had` program does not receive

Table A.31. The mobility socket processing functions used by `had`

Name
`int mipsock_input(struct mip_msghdr * miphdr)`
`void mip6_dad_order(int message, struct in6_addr * addr)`
`void mip6_dad_done(int message, struct in6_addr * addr)`

a Binding Update message before expiring the binding cache entry, the entry will be removed.

7.4 Mobility Socket Processing

Table A.31 shows the mobility socket functions. The `mipsock_input()` function is an entry point of the message input. In the home agent case, the DAD-related message is handled.

The home agent has a special task to perform the DAD procedure on behalf of the mobile node. The `mip6_dad_order()` function initiates the procedure by sending the `MIPM_DAD` message to the mobility socket. The *message* argument is one of `MIPM_DAD_DO`, `MIPM_DAD_STOP`, or `MIPM_DAD_LINKLOCAL`. The function is called through the creation of a new binding cache entry, which has an address whose DAD verification has not been completed.

The `mip6_dad_done()` function is called when the `mipsock_input()` function receives the `MIPM_DAD` message from the kernel, which indicates the status of the DAD procedure started before. If the result is success, that is the returned command is the `MIPM_DAD_SUCCESS`, then the related binding cache entry is enabled and a Binding Acknowledgement message is sent to the mobile node. If it fails, that is the `MIPM_DAD_FAIL` command is returned, then the binding cache is removed and a Binding Acknowledgement message is returned with an error status.

7.5 ICMPv6 Socket Processing

Table A.32 shows main ICMPv6 processing functions used by the `had` program.

The `icmp6_input_common()` function handles two ICMPv6 messages. One is the Dynamic Home Agent Address Discovery request message, sent from a

Table A.32. The ICMPv6 processing functions used by `had`

Name
`int icmp6_input_common(int` *fd*`)`
`int send_haadrep(struct in6_addr *` *dst*`, struct in6_addr *` *anycastaddr*`, struct min6_dhaad_req *` *dhreq*`, u_short` *ifindex*`)`
`int send_mpa(struct in6_addr *` *dst*`, u_int16_t` *mps_id*`, u_short` *ifindex*`)`
`int receive_ra(struct nd_router_advert *` *ra*`, size_t` *ralen*`, int` *receivedifindex*`, struct in6_addr *` *in6_lladdr*`, struct in6_addr *` *in6_gladdr*`)`

mobile node to get the address of the home agent. In response to the message, the `send_haadrep()` function is called from the `icmp6_input_common()` function to send a Dynamic Home Agent Address Discovery reply message. The second message is the Mobile Prefix Solicitation message sent from a mobile node to query the home prefix information. Similar to the previous message, the `send_mpa()` function is called to send a Mobile Prefix Advertisement message in response to the solicitation message.

The `receive_ra()` function handles the Router Advertisement message. When the home agent replies a Dynamic Home Agent Address Discovery reply or Mobile Prefix Advertisement messages, it needs to know the information of the other home agents' addresses and the latest home network prefix information. This information is distributed by the Router Advertisement message. The Mobile IPv6 specification defines a new Router Advertisement option which includes home agent information such as addresses and lifetime. When the received Router Advertisement message contains another home agent's information, it is recorded as in the list of `home_agnet_list{}` structure. The home network prefix information is also received through the Router Advertisement message. The collected data is used when replying discovery messages from mobile nodes.

8. Correspondent Node Program

In this section the correspondent node program `cnd` is explained.

```
struct mip6_nonces_info {
        struct mip6_nonces_info *next, *prev;
        u_int16_t nonce_index;
        u_int8_t  nonce[MIP6_NONCE_SIZE];
        u_int8_t  node_key[MIP6_NODEKEY_SIZE];
        time_t    nonce_lasttime; /*  generated time */
        LIST_HEAD(, mip6_nonce_blockedbce) nb_head;
};
```

Figure A.18. The `mip6_nonce_info{}` structure

8.1 Structure

Figure A.18 shows the `mip6_nonces_info{}` structure which keeps the nonce information used when calculating authentication data to verify the Binding Update message. Since the correspondent node creates a binding cache entry, the same binding cache structure used by the home agent is used too.

8.2 Behavior

The role of the `cnd` program is to perform the return routability procedure. The listening sockets are the same as the other programs.

8.3 Mobility Header Processing

Table A.33 shows the list of mobility header-related functions used by the `cnd` program. The `mh_input()` function processes the mobility header messages related to the return routability procedure. One is the Home Test Init message, and the other is the Care-of Test Init message. On receiving a Home Test Init message from a mobile node, the `cnd` program returns a Home Test message by calling the `send_hot()` function. Similarly, a Care-of Test message is returned by the `send_cot()` function in response to the Care-of Test Init message input.

A correspondent node has a list of random values indexed by numbers. When the correspondent node returns the Home Test or Care-of Test messages, token values generated from the random values and mobile node's addresses (the care-of address for the Care-of Test message, the home address for the Home Test

Table A.33. The mobility header processing functions used by `cnd`

Name
`int mh_input(struct in6_addr *` *src*`, struct in6_addr *` *dst*`, struct in6_addr *` *hoa*`, struct in6_addr *` *rtaddr*`, struct ip6_mh *` *mh*`, int` *mhlen*`)`
`int send_hot(struct ip6_mh_home_test_init *` *hoti*`, struct in6_addr *` *dst*`, struct in6_addr *` *src*`)`
`int send_cot(struct ip6_mh_careof_test_init *` *coti*`, struct in6_addr *` *dst*`, struct in6_addr *` *src*`)`
`int receive_bu(struct in6_addr *` *src*`, struct in6_addr *` *dst*`, struct in6_addr *` *hoa*`, struct in6_addr *` *rtaddr*`, ip6_mh_binding_update *` *bu*`, int` *mhlen*`)`

message). The Home Test and Care-of Test messages include the index to the random values (called nonce index), and generated tokens respectively.

A mobile node which receives these two messages will generate a shared key from the received token values, and use the key to sign a Binding Update message to be sent to the correspondent node. Since the Home Test message and the Care-of Test message are delivered from the correspondent node to the mobile node using different paths, it is hard to steal both messages at the same time. This assumption is the key of the return routability procedure.

When the correspondent node receives the Binding Update message signed by the mobile node, the message is processed by the `receive_bu()` function. The `cnd` program extract two nonce indices from the Binding Update message, and re-generate the shared key. If the received Binding Update message can be verified with the re-generated shared key, then the message is accepted, otherwise, it is dropped.

8.4 Mobility Socket Processing

The mobility socket processing is almost same as that of home agent. When a Binding Update message is accepted, the `cnd` program issues the `MIPM_BC_ADD` message to the mobility socket to create a new binding cache entry in the kernel. When the lifetime of the binding cache entry expires, the kernel entry is removed by the `MIPM_BC_REMOVE` message.

Appendix B

Protocol Behavior

In this appendix, a brief explanation of the extension protocols implemented in this dissertation is provided.

1. Network Mobility Basic Support

Network Mobility Basic Support (NEMO BS) is an extension protocol of Mobile IPv6. While Mobile IPv6 focuses only the host mobility function, NEMO BS supports the router mobility function, either.

The protocol behavior of NEMO BS is the same as that of Mobile IPv6. The difference is as shown below.

- The binding information exchange messages are extended to be able to indicate that:
  1. The mobile node is acting as a mobile router.
  2. The home agent supports NEMO BS.
- A new option is added to notify the prefix information of the mobile network which a mobile router serves.

As same as a mobile host in Mobile IPv6, a mobile router has its logical fixed network at its home network. At the home network, the mobile router gets a home address and mobile network. When the mobile router moves to another

network, it sends a binding update message indicating that it is working as a router.

When a home agent receives the binding update message, it creates an IP-in-IP tunnel between itself and the mobile router same as the mobile host case. In the mobile host case, only the traffic for the mobile host is tunneled. However, in NEMO BS case, the traffic sent to the mobile network behind the mobile router is also tunneled in addition to the traffic sent to the mobile router. On the other direction, the traffic generated from the mobile network is tunneled from the mobile router to the home agent as well.

The network prefix information can be specified by the following 3 ways.

Implicit mode The mobile network prefix information is statically defined at both home agent and mobile router. In this case, the signaling message does not contain any network prefix information. When the registration of the care-of address has been successfully completed, both the home agent and the mobile router set up tunnel routing information based on the pre-defined mobile network prefix information.

Explicit mode The mobile network prefix information is configured at the home agent, and notified to the mobile router when registration. The benefit of this mode is that the mobile router does not need to know the prefix information before hand. The home agent can assign any network prefix taken from the prefix pool managed at the home network side.

Dynamic mode In this mode, the mobile router and its home agent exchange routing information of the mobile network prefix. The mobile router can have its own network prefix independent of the home network configuration, as long as the home agent spread the mobile network routing information to the Internet.

Figure B.1 is the Binding Update/Acknowledgement messages used in NEMO BS. The *Router flag* (*R flag*) is newly defined to indicate that the mobile node is acting as a router.

The status values used in the Binding Acknowledgement message is also extended to indicate mobile network-related status information. Table B.1 shows the extended status values.

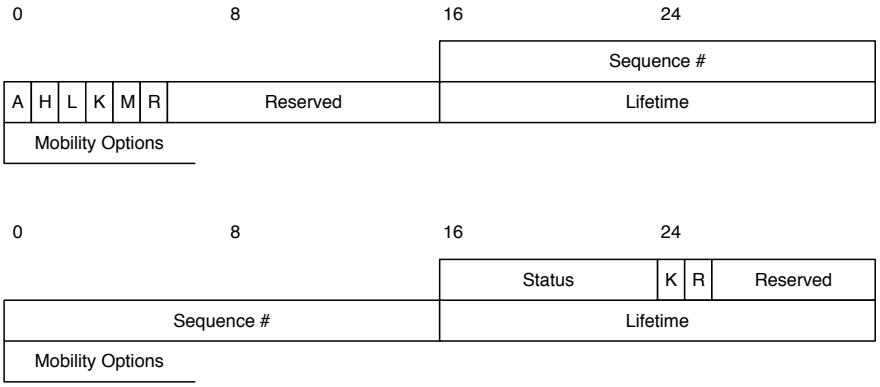


Figure B.1. The Binding Update (upper figure) and Binding Acknowledgement (lower figure) messages of NEMO BS

Table B.1. Extended status values for NEMO BS

Value	Description
140	Mobile router operation not permitted
141	Invalid prefix
142	Not authorized for the prefix
143	Forwarding setup failed (prefixes missing)

The prefix information is carried by the newly defined mobility option message when the explicit mode is used. Figure B.2 shows the option format. The type number for this option is 6.

2. IPv4 Network Prefix

The IPv4 Network Prefix support defines the operation rules to allow IPv4 mobile networks for NEMO BS, and specifies the option format to carry the IPv4 mobile network prefix information. If both home agent and mobile router supports this function, and the IPv4 mobile network prefix information bound to the mobile router is configured, then the routing information for the IPv4 network is configured over the tunnel link established between the home agent and the mobile router.

When used in the explicit mode, the IPv4 mobile network prefix is notified

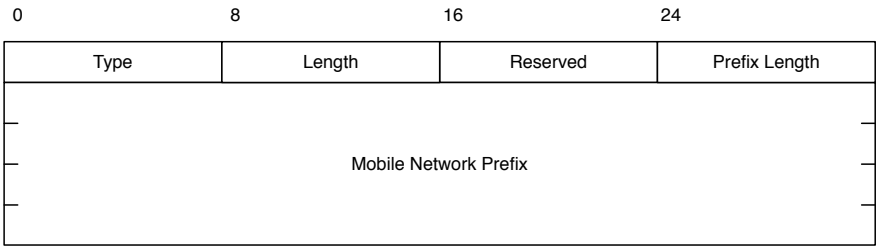


Figure B.2. The Mobile Network Prefix option format

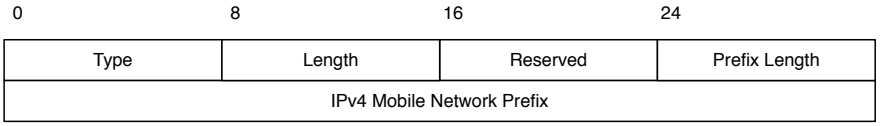


Figure B.3. The IPv4 Mobile Network Prefix option format

by the newly specified IPv4 Mobile Network Prefix mobility option shown in figure B.3.

The home agent supporting this extension and the mobile router are configured to use IPv4 mobile network, or the mobile router requests an IPv4 mobile network prefix by the IPv4 Mobile Network Prefix option, the home agent must return the operation result using the IPv4 Mobile Network Prefix Registration Status mobility option (figure B.4) with the Binding Acknowledgement message. The status values are shown in table B.2.

3. Multiple Care-of Addresses Registration

The Multiple Care-of Addresses Registration (MCoA) mechanism enables a mobile node to use multiple network interfaces concurrently. In the base Mobile IPv6 specification, each mobile node is recognized by its home address which is unique to each mobile node. MCoA introduces the binding identifier (BID) to

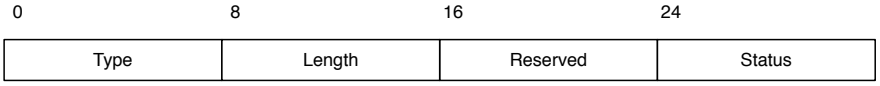


Figure B.4. The IPv4 Mobile Network Prefix Registration Status option format

Table B.2. IPv4 Mobile Network Prefix Registration Status values

Value	Description
0	Success
140	IPv4 prefix registration not permitted
141	Invalid prefix
142	Not authorized for the prefix
144	Forwarding setup failed

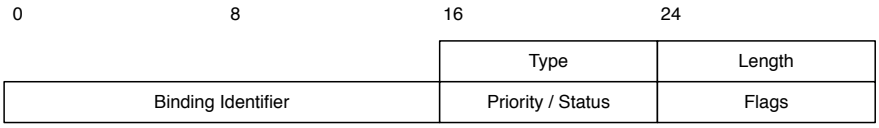


Figure B.5. The Multiple Care-of Addresses Registration option format

distinguish interfaces equipped on a mobile node. The home agent distinguishes the binding information by the pair of the home address of a mobile node and BID of the interface of the mobile node.

By using MCoA mechanism, a mobile node can distribute its outgoing traffic based on the property of the network interface. For example, the data which requires low latency can be sent to the interface with low latency but high cost network. The large amount of data may be sent to the interface that cost is low but the speed is slow.

Figure B.1 also shows the MCoA extended Binding Update message. The *MCoA flag* (*M flag*) indicates that the Binding Update message contains BID information. Figure B.5 shows the Multiple Care-of Addresses Registration option. The Binding Identifier field keeps the BID information. The Priority/Status field is used to specify the priority of the interface when used with the Binding Update message by a mobile node. It is also used to specify the processing status of the option when used with the Binding Acknowledgement message by a home agent as shown in table B.3. The flags field contains additional processing flags as specified in table B.4. The specification is still being discussed at IETF at the time of this writing, February 2009. The fields explained here is mostly compliant to the 5th version of the specification [84] which is implemented in the mobility platform (but not exactly the same), and not compliant to the latest version.

Table B.3. Multiple Care-of Addresses Registration option status values

Value	Description
0	The option is accepted.
128	The home agent does not support the option.

Table B.4. Multiple Care-of Addresses Registration option flag

Value	Description
0x8000	Stop using the specified interface.
0x4000	Can be disabled whenever necessary.

The traffic distribution policy is not the scope of this specification. This specification just sets up the multiple paths between a home agent and a mobile node. The rules of which BID uses what kind of flow is decided by other mechanisms.

4. Global Home Agent Distribution

The Global HA-HA is a mechanism to solve the following two problems which Mobile IPv6 has.

- A single point of failure problem of the home agent
- Redundant path problem which occurs when a mobile node and a correspondent node are located closely while its home agent is far away.

In the Global HA-HA mechanism, a home network is distributed to multiple locations in the Internet. Each home agent has two addresses. One address is taken from the home network, the other is taken from the external interface through which the home agent is attached to the Internet. The former address is sometimes called the home agent address, and the latter address is called the home agent's care-of address.

The routing information of the home network is advertised from all the home agents to the Internet routing system. This kind of routing technique is called the anycast routing. Since the same network prefix (the home network prefix) information is advertised from multiple locations, the packet sent to the home network traverses to the nearest home network based on the routing information.

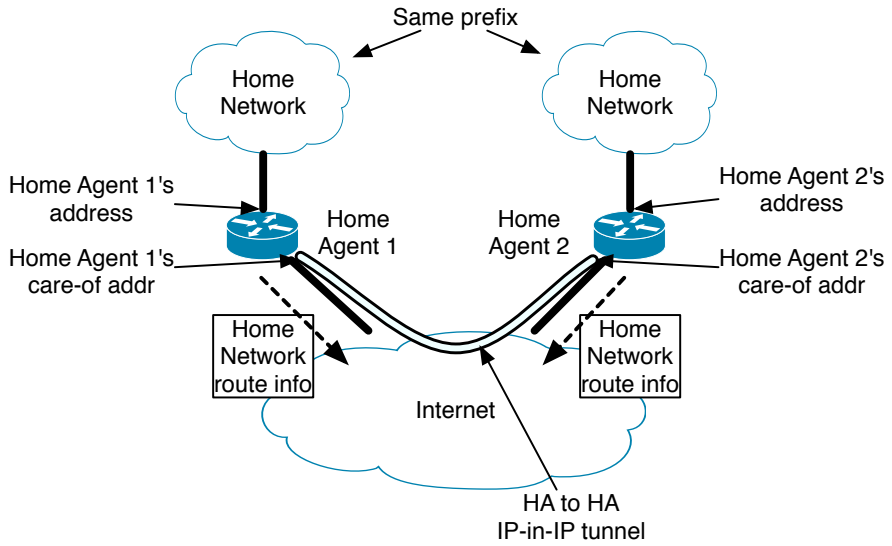


Figure B.6. The example network configuration of Global HA-HA

The home agents create an IP-in-IP overlay network between them using their home agent's care-of addresses. Using the home agent to home agent tunnel (HA-HA tunnel), they can exchange traffic delivered to the home network. The configuration example is shown in figure B.6.

When a mobile node boots up, it first picks one of the home agent addresses, and sends a binding update message to the address. The packet reaches to the nearest home agent based on the Internet routing information. If the home agent received the message is the actual destination of the message, that is, the destination address of the binding update message and the address of the home agent is same, then the normal Mobile IPv6 procedure starts.

If the received home agent is not the destination of the binding update message, the home agent forwards the binding update message to the right home agent using the HA-HA tunnel. By receiving a binding update message from the HA-HA tunnel, the right home agent can know that its mobile node is now registering the other home agent. In this case, the home agent notifies the mobile node to re-register to the nearest home agent using the Home Agent Switch message. Figure B.7 shows the message format.

Home agents share the binding information of the mobile nodes of the home network. When the binding information is successfully registered, the master

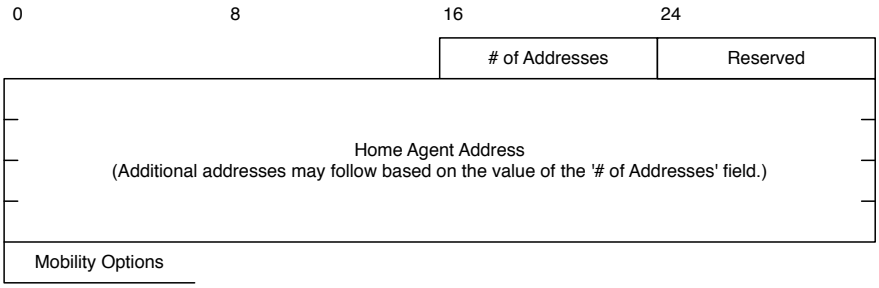


Figure B.7. The Home Agent Switch message format

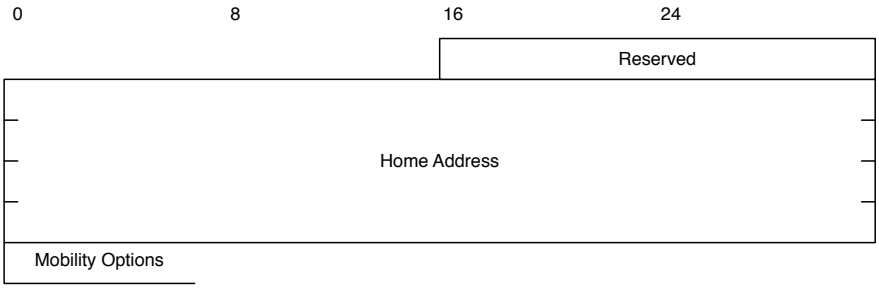


Figure B.8. The Binding Migration message format

home agent, that is handling the binding update message and acknowledgement message, notifies other home agents of the binding information using the Binding Migration message shown in figure B.8. A home agent which receives the binding migration message creates a slave binding information for the mobile node specified in the message.

When a mobile node sends a packet to a correspondent node, the normal Mobile IPv6 encapsulation procedure takes place. That is, the packet is encapsulated by IP-in-IP mechanism from the mobile node to the home agent. The home agent decapsulates the packet, and forwards it toward the correspondent node.

On the reverse side, the scenario differs depending on the location of the correspondent node. If the packets sent from the correspondent node are delivered to the master home agent, then the packet processing procedure is same as that of Mobile IPv6. If the correspondent node is located far from the mobile node, and the packets sent to the mobile node are received one of the slave home agents, then the slave home agent first encapsulates the packet toward the master home agent using the HA-HA tunnel. The master home agent decapsulates the packet,

and re-encapsulates it to deliver the packets to the mobile node.

When a mobile node moves to another network, the nearest home agent may change. If there is a nearer home agent from the new network, the re-registration message will be delivered to the nearest slave home agent, not to the master home agent. The binding update message is forwarded by the nearest slave home agent to the master home agent using the HA-HA tunnel. The master home agent recognizes that the mobile node has moved to farther network, and notifies the mobile node to re-register to the nearest slave home agent using the Home Agent Switch message. By this mechanism, the load of home agents is distributed based on the location of mobile nodes.

If a home agent stops working, then the similar process will occur. Since the routing information is stopped when a home agent stops, all the following messages sent to the home network are naturally re-directed to the next nearest home network based on the Internet routing system. With the same process as described above, a new master home agent is chosen and all the mobile nodes and correspondent nodes can resume their communication.