KUNGL TEKNISKA HÖGSKOLAN

25 January 1996

Implementation and Analyses of the Mobile-IP Protocol

Fredrik Broman and Fredrik Tarberg

Thesis report for a masters degree in Computer Science at the Department of Teleinformatics, Royal Institute of Technology, Stockholm, Sweden.

Abstract

This report is the result of a masters degree project conducted at the Department of Teleinformatics at the Royal Institute of Technology during the autumn 1995. The area investigated is the Mobile Internet Protocol, especially its implementation and efficiency.

The thesis work is divided into three areas. The first area includes the development and implementation of a Management Information Base for the Mobile-IP protocol. The second area deals with the porting of a Mobile-IP implementation for SunOS to MachOS and Solaris. The last area covers the tests done to measure the throughput and latency of the protocol.

"If you would not be forgotten, as soon as you are dead & rotten, either write things worth reading, or do things worth writing."
Benjamin Franklin

1.0 Introduction

1.1 Background

Mobile communication is an area which is rapidly developing. Cellular telephones have become, at least in Sweden, a common feature for many people as the price of cellular telephones have been subsidized by the telephone operators. Even though cellular phones are well suited to voice communication the provided bandwidth is too small to get an acceptable data transmission rate for computers. As computers become less expensive and smaller in size people will expect their mobile computer equipment to support communication mobility.

The Walkstation Project [4] is one of the research activities at the Department of Teleinformatics and Ericsson Radio Systems. The aim is to allow users of portable laptop computers to move around while retaining all possibilities that a fixed network connection provides. This is done by providing mobile users with a high capacity packet radio based cellular network.

How to efficiently support mobile wireless Internet access and how to provide the mobile users with transparent access to the Internet information are still open questions. The efforts to integrate mobile communication and support mobility on the Internet are being undertaken by the Internet Engineering Task Force (IETF) and is part of the Walkstation Project. This will result in a standard protocol called Mobile-IP in the near future. Mobile-IP is an abbreviation of “Mobile Internet Protocol”. It was originally proposed by Dr. John Ioannidis and Prof. Gerald Maguire Jr. from Columbia University (now professor at KTH). As there were multiple proposals for such a standard, an Internet Engineering Task Force (IETF) ‘Working Group for Mobile IP’ was formed in June 1992 to develop a single Mobile-IP protocol. The goal is to allow transparent routing of IP data packets to and from mobile hosts.

1.2 Problem statement and project specification

The title of this degree project is “Implementation and Analyses of the Mobile-IP Protocol”, and it has involved several different areas related to the Mobile-IP protocol. While the specification for the Mobile-IP protocol was not yet fixed by the IETF when this project started, one of the tasks was to examine the effects of implementation choices and test them. We also had to examine how the protocol works (for example “Is the functionality complete?”, “What is the performance limited by?”, “What extension should be made?”).

Our work included the following tasks:

- Understand the Mobile-IP Protocol draft and look at the implementation written by Anders Klements.
- Define a Mobile-IP Management Information Base (MIB).
- Incorporate support for Network Management (SNMP) into Anders Klemets’ Mobile-IP implementation.
- Port the implementation to:
  a. The MINT (Mobile INTernet) router and the MachOS
  b. Solaris
- Plan a series of experiments:
  a. Throughput and latency measurements - for a fixed basestation and stationary “mobile”.

b. Handoff measurements for a mobile between two basestations. This should not only explore the wireless traffic but also the traffic on the infrastructure.

- Make a series of measurements and do an analysis of these experiments.

2.0 The Walkstation II Project

The purpose of the Walkstation II project is to investigate a complete radio based cellular system for interactive mobile multimedia applications, and the project includes aspects of VLSI design, radio communication, network protocols and mobile applications. The three main components of the project are the Mobile Internet Router (MINT) [6, 7, 8], the Radio Transceiver and the Mobile-IP protocol [5].

2.1 The system

The cellular, wireless LAN that will be the result of the Walkstation II project consists of Base Stations and Mobile Stations. Every cell in the network is covered by a Base Station with a (Spread Spectrum) Radio Transceiver. The Base Station is also connected to a wired network. The Mobile Stations will be able to connect to the Internet through a Base Station. Even though this wireless LAN looks like any other cellular system there is one major difference. There is no top down hierarchy as in found in, for example, GSM and DECT where the Base Stations control the allocation of the radio resources. The radio channel is a shared media which can be used by any Station to send information whenever the channel is free, and it is immediately released afterwards. From this perspective the system more closely resembles an ethernet than a cellular mobile telephone communication system. It is also possible to send data from one Mobile Station to another within the same cell *without* going though a Base Station.

2.2 The Mobile INTernet Router (MINT)

One of the goals of the Walkstation II project was to incorporate all communication hardware and software into one device called the MINT. The advantages of this design decision are that the MINT can be made compatible with many different types of Mobile Hosts and no changes have to be made in the operating system of the Mobile Host. In addition, all computations necessary for communication can be done by the MINT and do not have to burden the Mobile Host. Also, by putting all mobile functionality into one device it is transparent to the user (and their computer).

The Mobile INTernet Router is designed to be a small, lightweight router that can be plugged into the back of any computer. The basic architecture of the MINT is shown in Figure 1. Today the size of a MINT is approximately 32x27x13 cm, but the intension is to make it fit on a PCMCIA card.

The MINT consists of a 25 MHz MC68030 processor, 1 Mbyte of ROM, 8 Mbyte of RAM, with two serial, one parallel, one SCSI and two Ethernet interfaces. In addition, there is a prototyping area with access to the processor bus and the two ethernet controllers. Basically the MINT can be viewed as a three component device; one connection to the host backbone (an ethernet network), one interface to a wireless LAN and a processing part in between to handle the communication protocol. This architecture was designed to be used both with Mobile Hosts and Base Stations.

The MINT hardware was developed in a pre-project in conjunction with HP Labs, Palo Alto. Today there are 8 MINTs up and running. The operating system currently used on the MINTs is a version of the Mach 3.0 operating system, ported by Anders Klemets[18].

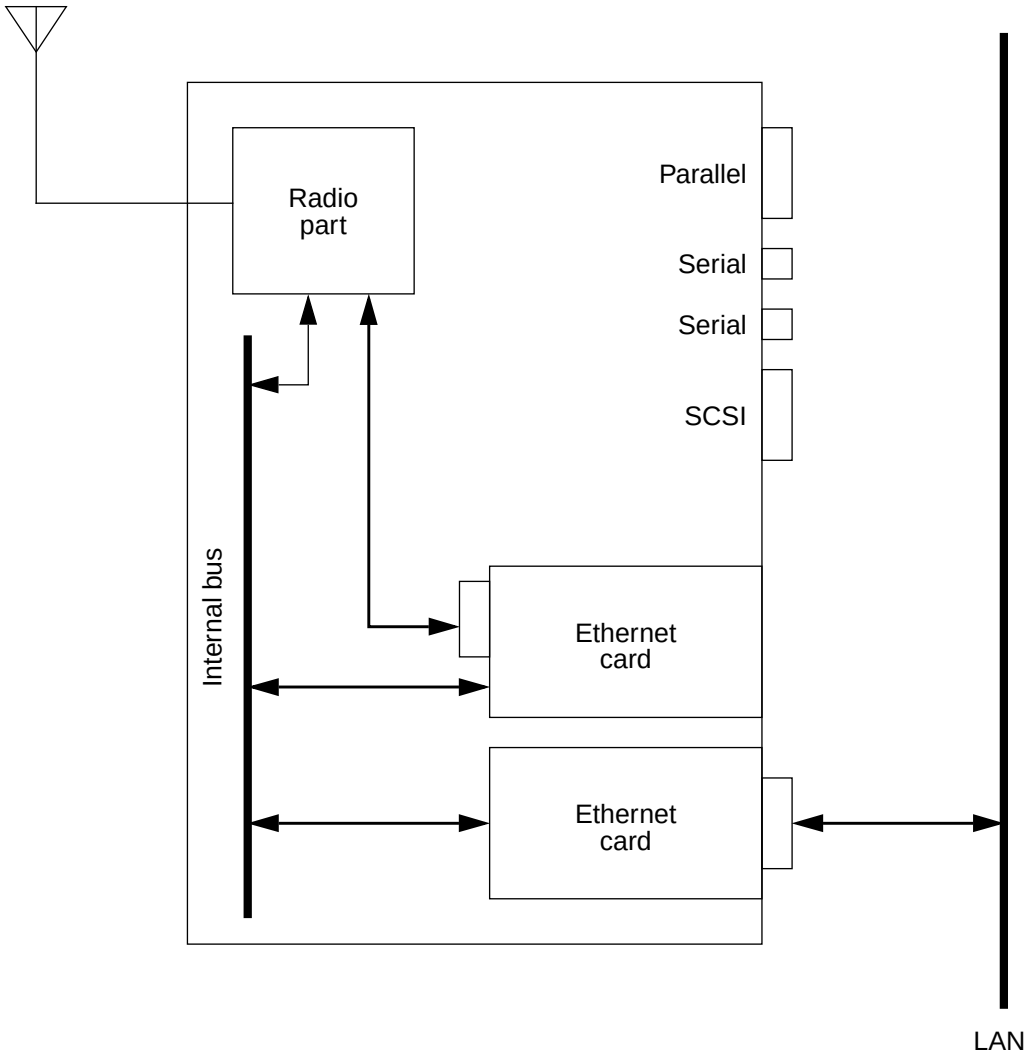

Figure 1. The MINT

3.0 The Mobile-IP protocol

When Mobile Nodes are to be introduced on the Internet, the present version of the network layer protocol (Internet Protocol version 4) is no longer enough. New functionality has to be implemented to handle the new situation. This problem was encountered in the Student Electronic Notebook project [12] and as a result a Mobile Internet Protocol (Mobile*IP [10]) was suggested. Today the work on standardizing Mobile-IP is conducted by the Mobile-IP Working Group of the Internet Engineering Task Force [5].

3.1 Introduction

The protocol used at the network layer on the Internet, the Internet Protocol (IP), was designed with two assumptions in mind. First that a node's point of attachment remains fixed, and secondly that a node's IP address identifies the network to which it is attached. Routing of datagrams is based on the network number portion of the node's IP address. For example, a datagram destined to the computer with the IP address 130.237.215.110 is sent to the network with the network number 130.237.215.

If a node could move around on the Internet without changing its IP address it would no longer be possible to correctly route datagrams to it. To overcome this problem, the Mobile Internet Protocol was introduced. There are basically three entities defined in the protocol. These are the Mobile Node (MN), the Home Agent (HA) and the Foreign Agent (FA). A Mobile Node is a host or a router that changes its point of attachment from one network or subnetwork to another without changing its IP address. The Mobile Node has one network called the home network. The Home Agent, which can be a host or a router, is located on the home network. The task of the Home Agent is to maintain a table of the current location of all its Mobile Nodes, and relay datagrams to their present location. All datagrams destined to a Mobile Node are caught on the home network by the Home Agent and sent to a Foreign Agent who relays the data to the Mobile Node. A Foreign Agent is a host or a router on the foreign network that takes care of the Mobile Node while it is away from home. When the Mobile Node wishes to make a connection to the Internet while away from home, it contacts the closest Foreign Agent who sends a registration request to the Mobile Node's Home Agent. When the Home Agent gets the request it knows where the Mobile Node is located at present and to which Foreign Agent it should relay the datagrams destined to the Mobile Node. The address of the Foreign Agent acts as a care-of address of the Mobile Node.

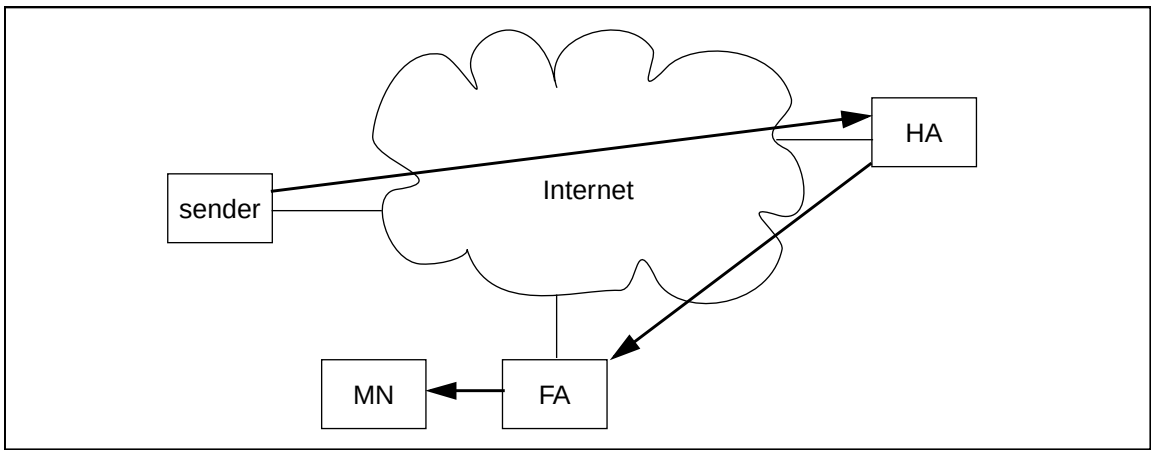

Figure 2. The entities in the Mobile-IP protocol

3.2 The Protocol

3.2.1 Requirements and Goals

In [5] the following aspects were considered important in designing the protocol.

- A Mobile Node using its home address shall be able to communicate with other nodes despite changing its point of physical attachment.
- Implementation of the protocol shall not cause a Mobile Node to be unable to communicate with other nodes that do not implement these mobility functions.
- No protocol enhancements are required in hosts or routers that are not providing any of the mobility functions.
- A Mobile Node shall provide authentication in its registration messages.

The link by which the Mobile Node is directly attached to the Internet is likely to be bandwidth limited and experience a higher rate of errors than traditional wired networks. Moreover, Mobile Nodes are more likely to be battery powered, and minimizing power consumption is important. Therefore, only a few administrative messages should be sent between a Mobile Node and an agent, and the size of these messages should be kept as short as possible.

3.2.2 Overview of Protocol Events

The following is a rough outline of the sequence of events that a Mobile Node goes through as given in [5]. See Figure 3.

- Mobility agents (Home Agents and Foreign Agents) advertise their presence via Agent Advertisements (see Section 3.2.3).
- A Mobile Node receives these advertisements and determines whether it is on its home subnet or a foreign subnet.
- The Mobile Node, if it detects that it has moved to a foreign subnet (either from its home subnet or from another foreign subnet), obtains a care-of address on the foreign subnet. The care-of address can either be obtained from the advertisements, or by some other assignment mechanism (for example DHCP [13]).
- The Mobile Node then registers its new care-of address with its Home Agent, possibly via a Foreign Agent (see Section 3.2.4).
- Packets sent to the Mobile Node’s home address are received by the Home Agent and relayed (possibly through a Foreign Agent) to the Mobile Node via encapsulation, using the care-of address as the new destination (see Section 3.2.5).

.

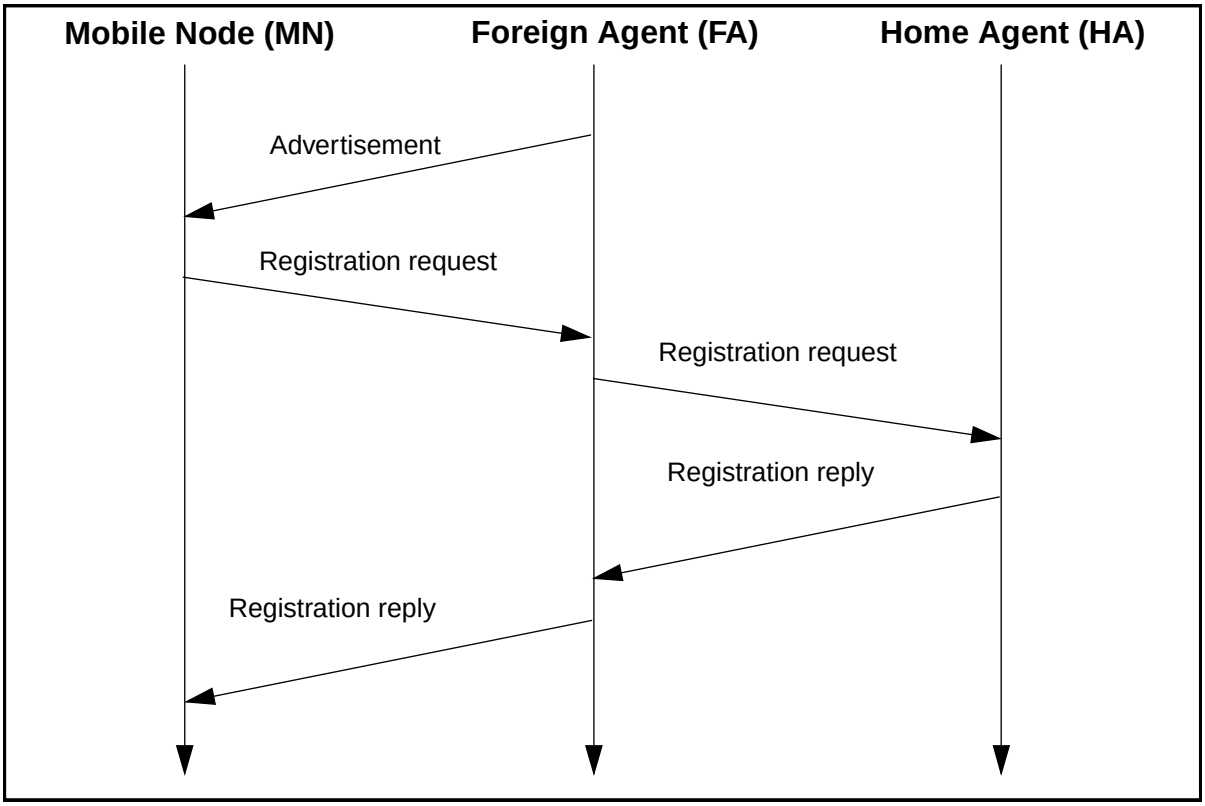

Figure 3. The messages sent when a Mobile Node register with a Home Agent through a Foreign Agent

3.2.3 Agent Discovery and Solicitation

To communicate with a Foreign or Home Agent, a Mobile Node must learn either the IP address or the link address of that agent. It is assumed that a link-layer connection has been established between the agent and the Mobile Node. The method used to establish such a link-layer connection is not specified in this protocol. After establishing a link-layer connection, the Mobile Node learns whether there are any agents available. If the address of any agent matches the Mobile Node's stored address for its Home Agent, the Mobile Node is at home.

The Mobile Node can get information about Mobile Agents either by receiving an ICMP Router Advertisement or by sending an ICMP Router Solicitation. To the ICMP Router Advertisement, information is added to indicate that the router serves as a mobility agent.

3.2.4 Registration

The registration function exchanges information between a Mobile Node and its Home Agent. The information sent from the Mobile Node is the new care-of address to which the Home Agent is supposed to send the datagrams destined to the Mobile Node. If a Mobile Node itself is assigned a care-of address, it can act without a Foreign Agent, and register or deregister directly with a Home Agent by the exchange of only 2 messages stated by the protocol:

- The Mobile Node sends a registration request to a Home Agent, asking it to provide service.

- The Home Agent sends a registration reply to the Mobile Node, granting or denying service.

When the care-of address is associated with a Foreign Agent, the Foreign Agent acts as a relay between the Mobile Node and the Home Agent. This extended registration process involves the exchange of 4 messages:

- The Mobile Node sends a registration request to the prospective Foreign Agent to begin the registration process.
- The Foreign Agent relays the request to the Home Agent, asking the Home Agent to register the Mobile Node at the Foreign Agent's care-of address.
- The Home Agent sends a registration reply to the Foreign Agent to grant or deny service.
- The Foreign Agent relays the registration reply to the Mobile Node to inform it of the disposition of its request.

The registration messages use the User Datagram Protocol (UDP) header [14]. A non-zero UDP checksum should be included in the header, and checked by each recipient. An administrative domain may require a visiting Mobile Node to register via a Foreign Agent. This facility is envisioned for service providers with packet filtering fire-walls, or visiting policies which require exchanges of authorization.

It is possible for a Mobile Node to have more than one care-of address at any given time. This can be useful when a Mobile Node moves within range of multiple cellular systems. When the Home Agent allows simultaneous bindings, it will encapsulate a separate copy of each arriving datagram to each care-of address, and the Mobile Node will receive multiple copies of its datagrams. This is not a problem since IP explicitly allows duplication of datagrams.

3.2.5 Forwarding Datagrams to the Mobile Node

The way in which IP packets are relayed from the Home Agent to the Mobile Node is by IP in IP encapsulation [15], also called "tunneling". This means that when the Home Agent receives an IP packet destined to a Mobile Node it puts the IP packet in a new IP packet (by adding an IP header) with the destination field set to the care-of address of that Mobile Node (see Figure 4).

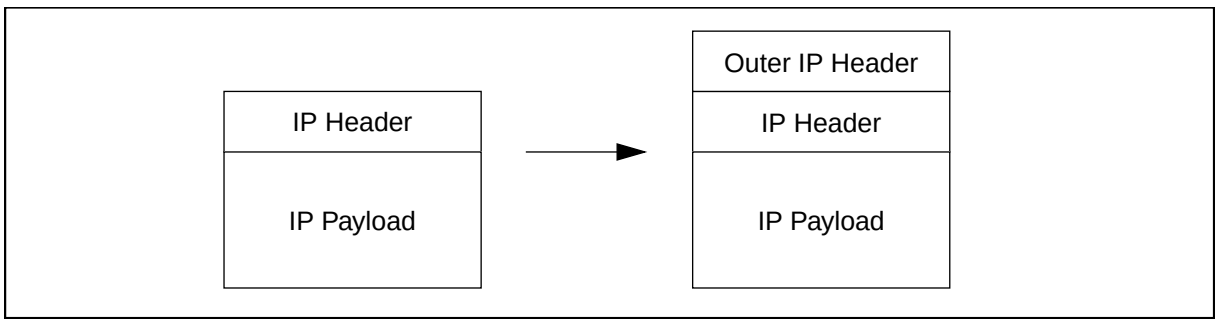

Figure 4. IP in IP encapsulation

When the Foreign Agent receives the IP packet it decapsulates it by removing the outer IP packet and sending the original packet to the Mobile Node. The general encapsulation case is shown in Figure 5.

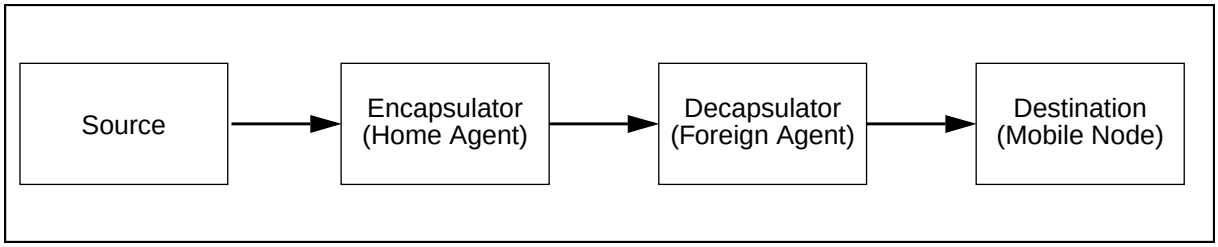

Figure 5. The general encapsulation case

Encapsulation is a way to re-address datagrams. Another method would be to use the IP Source Route option in the IP protocol (see [20]), which lets the sender specify a path that the IP datagram must follow. There are however several technical reasons to prefer encapsulation over source routing, according to the encapsulation draft [15]:

- There are unsolved security problems associated with the use of source routing.
- Current internet routers exhibit performance problems when forwarding packets which use the IP source routing option.
- Too many internet hosts process source routing options incorrectly.
- Firewalls may exclude source-routed packets.
- Insertion of an IP source route option may complicate the processing of authentication information by the source and/or destination of a datagram, depending on how the authentication is specified to be performed.
- It is considered impolite for intermediate routers to make modifications to the packets which they did not originate.

There are, of course, some disadvantages with the encapsulation technique too. For instance, encapsulated packets are normally longer than source routed packets.

4.0 Network Management

A computer network consists of many different components, for example routers, hubs, bridges and hosts. As the complexity of a system grows, the need for monitoring and controlling the different entities increases. This problem was recognized by vendors of network equipment, and as a result they developed strategies to manage their products. Of course different manufacturers came up with different solutions, and that led to problems for managers who administrated systems that consisted of equipment from different vendors. A standard was needed, and at the end of the eighties a standard was developed based on something called the *Internet-standard Network Management Framework*. This framework included a set of rules for describing management information, an initial set of managed objects, and a protocol used to exchange management information (the Simple Network Management Protocol [19]).

A network management systems consists of four components:

- one or more *managed nodes*, each containing an *agent* which runs the management software;
- at least one *network management station* (NMS) on which one or more network management applications (often called *managers*) reside;
- a network management *protocol* (for example SNMP) which is used by the manager and the agents to exchange management information; and
- a *Management Information Base* (MIB) that specifies what variables the network elements maintain.

The communication can happen in two ways: the manager asks the agent for a specific value, or the agent telling the manager that something important has happened. Also, the manager should be able to *set* variables in the agent in addition to reading values from it.

Note that the network management system can be thought of as a client/server system, where the management application (the manager) is the *client* which is sending questions and commands to the agent (the *server*).

4.1 SNMP

The Simple Network Management Protocol (SNMP) [19] emerged from the IETF (Internet Engineering Task Force) as a result of a recommendation from IAB (Internet Activity Board) concerning the standardization of network management. The philosophy behind the protocol was that it should be simple and focus on the areas of fault management and configuration management. However, the protocol proved to be very flexible and suitable for all kinds of network management.

In general, an SNMP interaction consists of a request of some kind, followed by a response. See Figure 6.

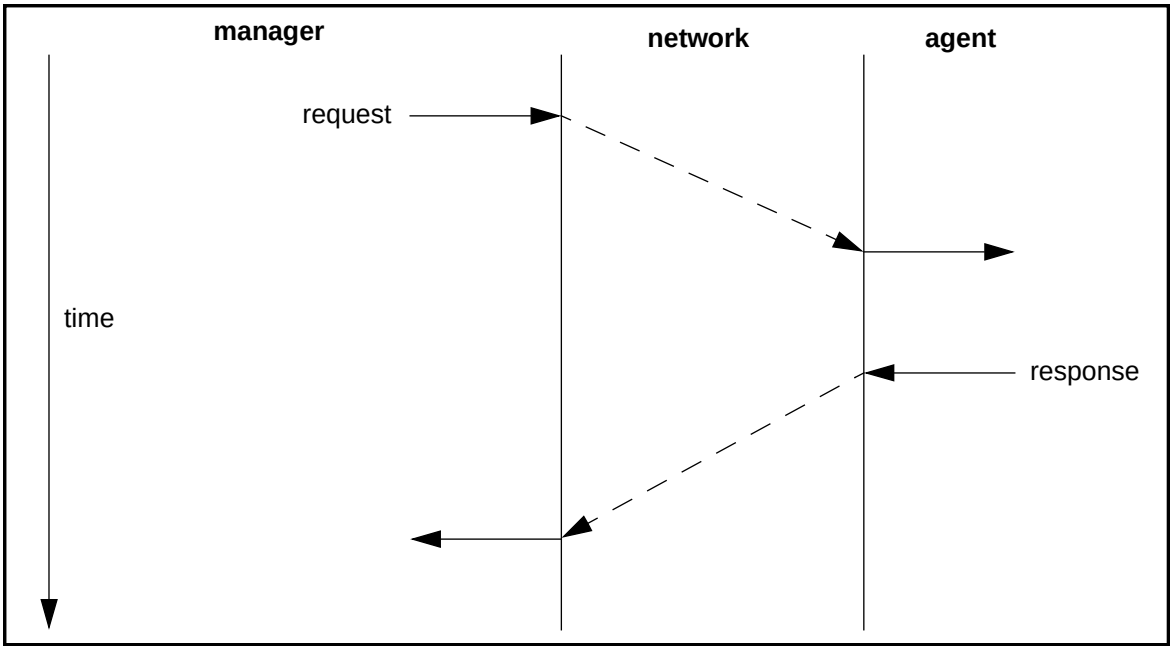

Figure 6. SNMP request-response interaction (based on [11] p. 243)

SNMP defines only five types of messages that are exchanged between the manager (the client) and an agent (the server):

1. Fetch the value of one or more variables: the `get-request` operator.
2. Fetch the next variable: the `get-next-request` operator.
3. Set the value of a variable: the `set-request` operator.
4. Return the value of a variable: the `get-response` operator. This is the message returned by the agent to the manager in response to the `get-request`, `get-next-request`, and `set-request` operators.
5. Notify the manager when something happens on the agent: the `trap` operator.

The first three messages are sent from the manager to the agent, and the last two are from the agent to the manager. The messages are sent in UDP packets.

4.2 MIB

The Management Information Base, or MIB, is a description of the information maintained by the agent that the manager can query or set. Each object in the MIB is given a name and is also defined by a unique sequence of integers separated by decimal points. This sequence of integers is called the *object identifier*, and it is allocated by some organization that has responsibility for a group of identifiers. The objects are arranged in a tree structure similar to a filesystem (see Figure 7). For example the name corresponding to 1.3.6.1.2.1.4 is iso.org.dod.internet.mgmt.mib-2.ip, and is the object identifier pointing to the Internet Protocol (IP) group.

A MIB definition resembles a definition of a data structure in a programming language. It is a description of which variables can be accessed and what data types they have. An example of a MIB definition can be found in Appendix G, which is a description of our MIB for the Mobile-IP protocol. MIBs are described using a subset of the Abstract Syntax Notation One (ASN.1).

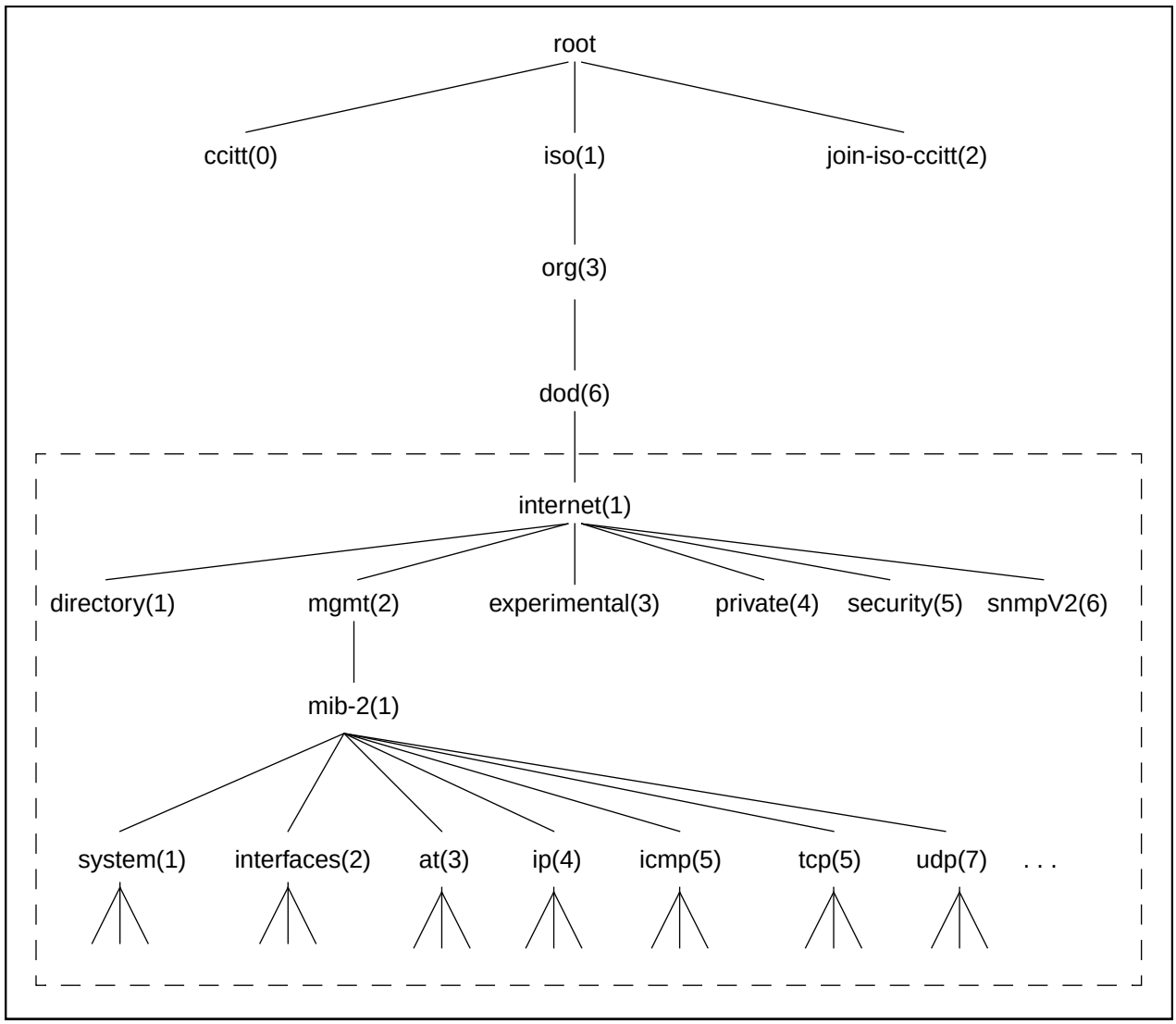

Figure 7. Object identifiers in the Management Information Base (based on [20] page 365)

5.0 The Mobile-IP MIB

One task of our degree project was to analyse the performance of the Mobile-IP protocol, and in order to do so we needed to get out certain variable values from the protocol implementation while it was running. This can be accomplished with a network management system. Unfortunately there did not exist any MIB for this new protocol, so our first task was to define a Mobile-IP MIB ourselves. We examined the protocol draft [5] and the implementation made by Anders Klemets [3] and tried to figure out which values we needed to make our measurements. The result is described in the figures and tables below, and the complete MIB definition described in Abstract Syntax Notation One (ASN.1) can be found in Appendix G.

Figure 8 shows some of the objects that we suggest should be in the Mobile-IP MIB, and these objects are further described in Table 1.

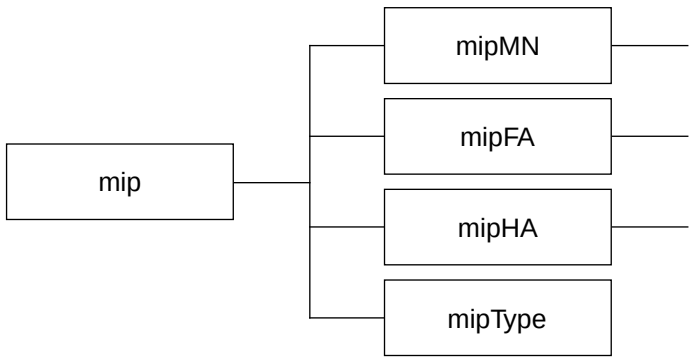

Figure 8. The Mobile-IP MIB

#	Field Name	MIB Object Label	Datatype	Description
1	Mobile Node	mipMN	OBJECT IDENTIFIER	The Mobile Node subgroup
2	Foreign Agent	mipFA	OBJECT IDENTIFIER	The Foreign Agent subgroup
3	Home Agent	mipHA	OBJECT IDENTIFIER	The Home Agent subgroup
4	Type	mipType	BIT STRING	A bit vector indicating whether this entity is acting as a Mobile Node, a Home Agent and/or a Foreign Agent

Table 1: The objects in the Mobile-IP MIB

5.1 The Mobile Node

The first subgroup in the Mobile-IP MIB is the Mobile Node. Figure 9 shows the Case Diagram for that object. A Case diagram (named after a Professor Case) is a simplified diagram that shows the flow of management information in a protocol layer. The horizontal lines represents counters.

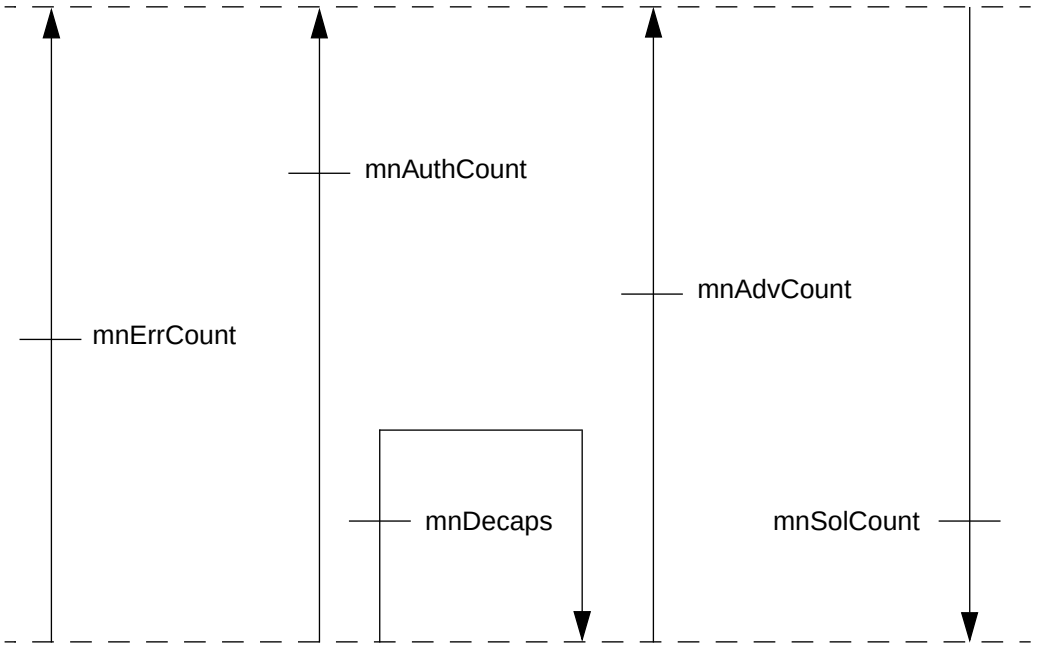

Figure 9. Case Diagram for the Mobile Node

Below is a picture of the objects that belong to the Mobile Node (Figure 10) and after that is a table which describes all the variables (Table 2). From the picture we can see that there are two tables defined in this subgroup; mnRegTable (a table of current registrations for the Mobile Node) and mnPendRegTable (a table of pending registrations). These tables are more carefully described in Table 4 and Table 5. The Mobile Node also has a list of potential Home Agents, mnHomeAgentList, which is described in Table 3.

The following sections in the chapter will follow the same pattern as this one; a Case diagram, a picture of the subgroup, a table describing the variables and separate tables for each of the table variables.

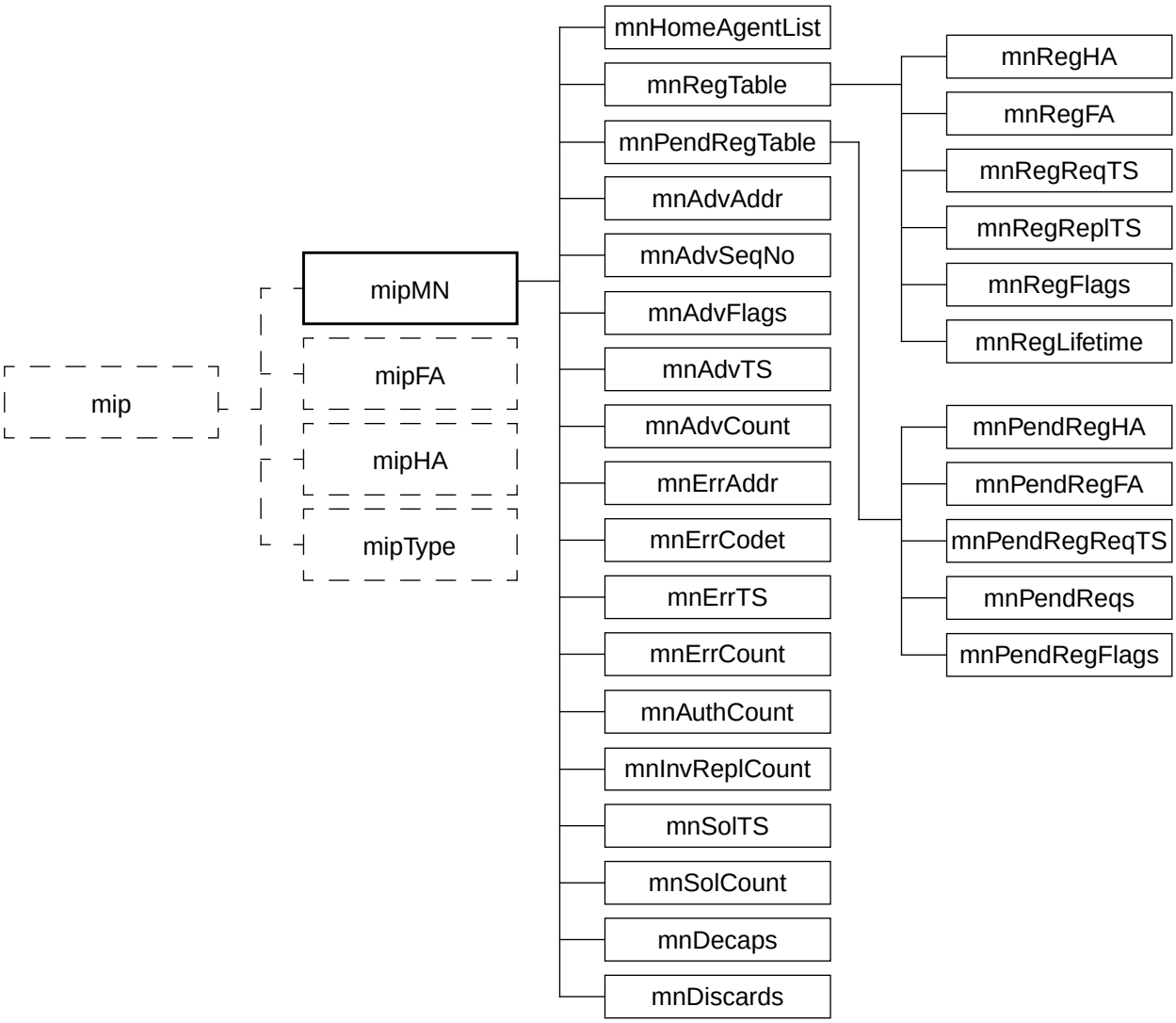

Figure 10. The Mobile Node subgroup

5.1.1 Mobile Node objects

#	Field Name	MIB Object Label	Datatype	Description
1	Home Agent List	mnHomeAgentList	SEQUENCE OF MNHAEntry	The Mobile Node's list of Home Agents. See Table 3
2	Registration Table	mnRegTable	SEQUENCE OF MnRegEntry	The Mobile Node's registration table. See Table 4
3	Pending Registration Table	mnPendRegTable	SEQUENCE OF MnPendRegEntry	The Mobile Node's pending registration table. See Table 5
4	Advertisement Address	mnAdvAddr	IpAddress	The IP address in the last received agent advertisement
5	Advertisement Sequence Number	mnAdvSeqNo	INTEGER (0..65535)	The sequence number in the last received agent advertisement
6	Advertisement Flags	mnAdvFlags	Bit String	The flags field in the last received agent advertisement
7	Advertisement Time Stamp	mnAdvTS	Counter	The time when the last agent advertisement was received
8	Advertisement Counter	mnAdvCount	Counter	The total number of agent advertisements received
9	Error Address	mnErrAddr	IpAddress	The IP address from which the last error message was received
10	Error Code	mnErrCode	INTEGER (0..255)	The error code in the last received error message
11	Error Time Stamp	mnErrTS	Counter	The time when the last error message was received
12	Error Counter	mnErrCount	Counter	The total number of error messages received
13	Authentication Exception Counter	mnAuthCount	Counter	The total number of authentication exceptions discovered in the MN
14	Invalid Reply Counter	mnInvReplCount	Counter	The total number of invalid replies
15	Solicitation Time Stamp	mnSolTS	Counter	The time when the last agent solicitation message was sent
16	Solicitation Counter	mnSolCount	Counter	The total number of agent solicitations sent
17	Decapsulations	mnDecaps	Counter	The number of IP-packets decapsulated at the Mobile Node
18	Discards	mnDiscards	Counter	The number of encapsulated packets discarded at the Mobile Node

Table 2: The objects in the Mobile Node subgroup

5.1.2 List of Home Agents

#	Field Name	MIB Object Label	Datatype	Description
1	Home Agent	mnHALAddr	IpAdddress	The IP address of a Home Agent

Table 3: The Mobile Node's list of Home Agents. Just a list of IP addresses.

5.1.3 Mobile Node Registration Table

#	Field Name	MIB Object Label	Datatype	Description
1	Home Agent	mnRegHA	IpAddress	The IP-address of the Home Agent
2	Foreign Agent	mnRegFA	IpAddress	The IP-address of the Foreign Agentt
3	Registration Request Time Stamp	mnRegReqTS	Counter	The time when the first registration request was sent
4	Registration Reply Time Stamp	mnRegReplTS	Counter	The time when the registration reply was received
5	Flags	mnRegFlags	Bit String	The flags field that was used in the request
6	Lifetime	mnRegLifetime	Integer	The remaining lifetime for this registration

Table 4: The Mobile Node's Registration Table

5.1.4 Mobile Node Pending Registration Table

#	Field Name	MIB Object Label	Datatype	Description
1	Home Agent	mnPendRegHA	IpAddress	The IP-address of the Home Agent
2	Foreign Agent	mnPendRegFA	IpAddress	The IP-address of the Foreign Agent
3	Registration Request Time Stamp	mnPendRegReqTS	Counter	The time when the first registration request was sent
4	Registration Requests	mnPendRegReqs	Integer	The number of registration requests sent
5	Flags	mnPendRegFlags	Bit String	The flags field that was used in the request

Table 5: The Mobile Node's Pending Registration Table

5.2 The Foreign Agent

Figure 11 shows the Case Diagram for the Foreign Agent.

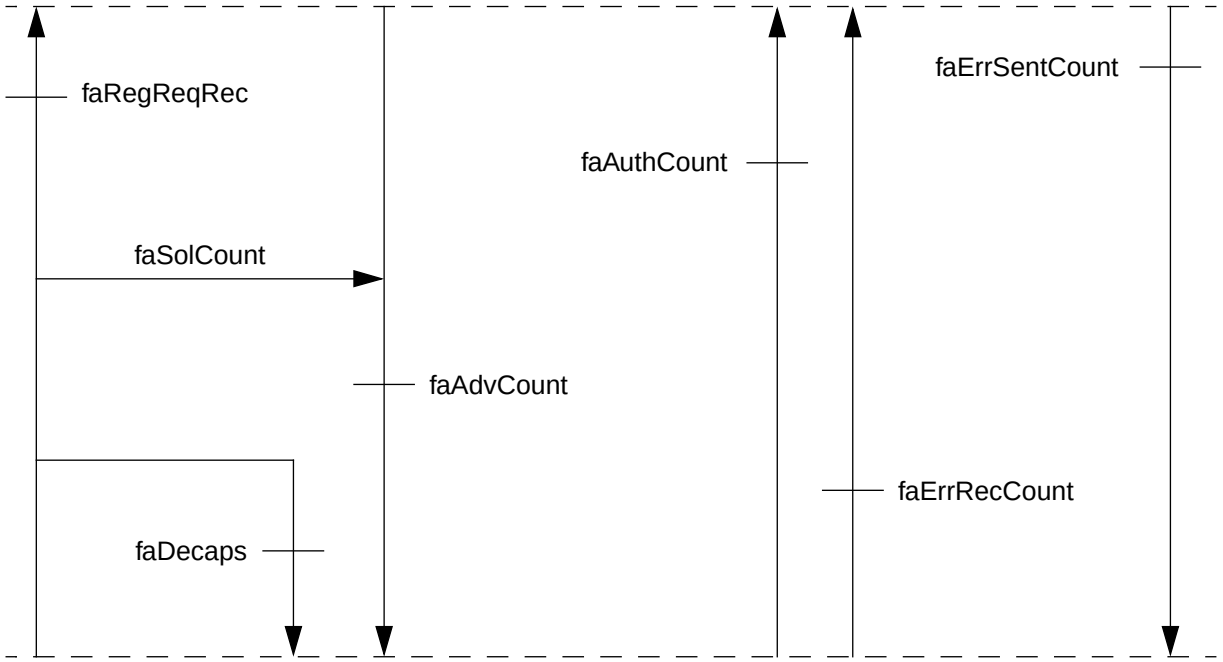

Figure 11. Case Diagram for the Foreign Agent

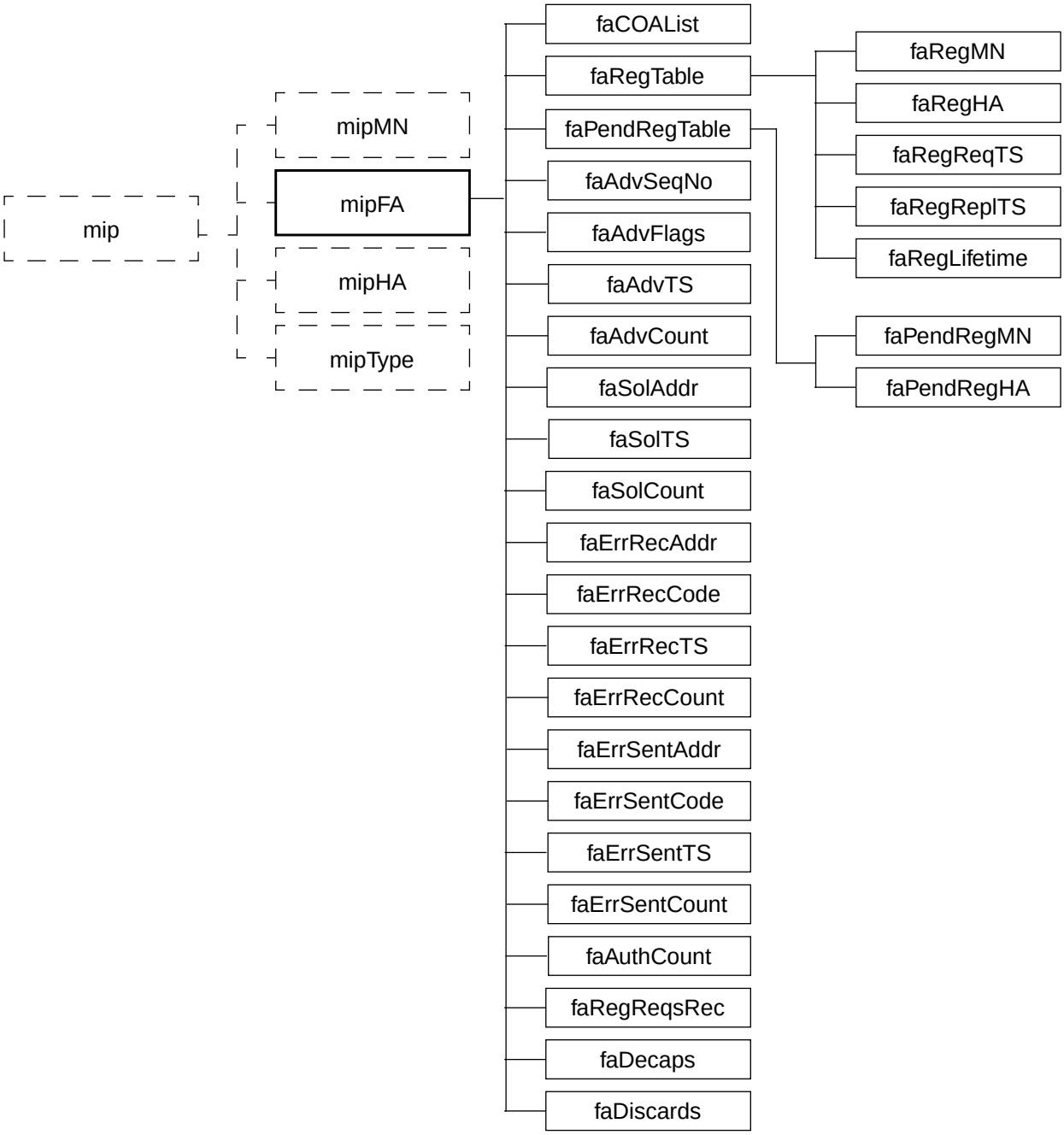

Figure 12. The Foreign Agent subgroup

5.2.1 Foreign Agent objects

#	Field Name	MIB Object Label	Datatype	Description
1	COA List	faCOAList	SEQUENCE OF FaCOAEntry	The Foreign Agent's list of care-of address, if any. See Table 7
2	Registration Table	faRegTable	SEQUENCE OF FaRegEntry	The Foreign Agent's registration table. See Table 8
3	Pending Registration Table	faPendRegTable	SEQUENCE OF FaPendRegEntry	The Foreign Agent's pending registration table. See Table 9
4	Advertisement Sequence Number	faAdvSeqNo	INTEGER (0..65535)	The sequence number in the last sent agent advertisement
5	Advertisement Flags	faAdvFlags	Bit String	The flags field in the last sent agent advertisement
6	Advertisement Time Stamp	faAdvTS	Counter	The time when the last agent advertisement was sent
7	Advertisement Counter	faAdvCount	Counter	The total number of agent advertisements sent
8	Solicitation Address	faSolAddr	IpAddress	The IP address in the last received agent solicitation message
9	Solicitation Time Stamp	faSolTS	Counter	The time when the last agent solicitation message was received
10	Solicitation Counter	faSolCount	Counter	The total number of agent solicitations received
11	Error Received Address	faErrorRecAddr	IpAddress	The IP address from which the last error message was received
12	Error Received Code	faErrRecCode	INTEGER (0..255)	The error code in the last received error message
13	Error Received Time Stamp	faErrRecTS	Counter	The time when the last error message was received
14	Error Received Counter	faErrRecCount	Counter	The total number of error messages received
15	Error Sent Address	faErrorSentAddr	IpAddress	The IP address to which the last error message was sent
16	Error Sent Code	faErrSentCode	INTEGER (0..255)	The error code in the last sent error message
17	Error Sent Time Stamp	faErrSentTS	Counter	The time when the last error message was sent
18	Error Sent Counter	faErrSentCount	Counter	The total number of error messages sent
19	Authentication Exception Counter	faAuthCount	Counter	The total number of authentication exceptions
20	Registration Requests Received	faRegReqsRec	Counter	The total number of registration requests received
21	Decapsulations	faDecaps	Counter	The number of IP-packets decapsulated at the Foreign Agent
22	Discards	faDiscards	Counter	The number of encapsulated packets discarded at the Foreign Agent

Table 6: The objects in the Foreign Agent subgroup

5.2.2 List of Care-of Addresses

#	Field Name	MIB Object Label	Datatype	Description
1	Care-Of Address	faCOAddr	IpAddress	The Care-of Address

Table 7: The Foreign Agent's list of Care-of addresses

5.2.3 Foreign Agent Registration Table

#	Field Name	MIB Object Label	Datatype	Description
1	Mobile Node	faRegMN	IpAddress	The IP-address of the visiting Mobile Node
2	Home Agent	faRegHA	IpAddress	The IP-address of the Mobile Node's Home Agent
3	Registration Request Time Stamp	faRegReqTS	Counter	The time when the first registration request was sent
4	Registration Reply Time Stamp	faRegReplTS	Counter	The time when the registration reply was received
5	Lifetime	faRegLifetime	Integer (0..65535)	The remaining lifetime for this registration

Table 8: The Foreign Agent's Registation Table

5.2.4 Foreign Agent Pending Registration Table

#	Field Name	MIB Object Label	Datatype	Description
1	Mobile Node	faPendRegMN	IpAddress	The IP-address of the visiting Mobile Node
2	Home Agent	faPendRegHA	IpAddress	The IP-address of the Mobile Node's Home Agent

Table 9: The Foreign Agent's Pending Registation Table

5.3 The Home Agent

Figure 13 shows the Case Diagram for the Home Agent.

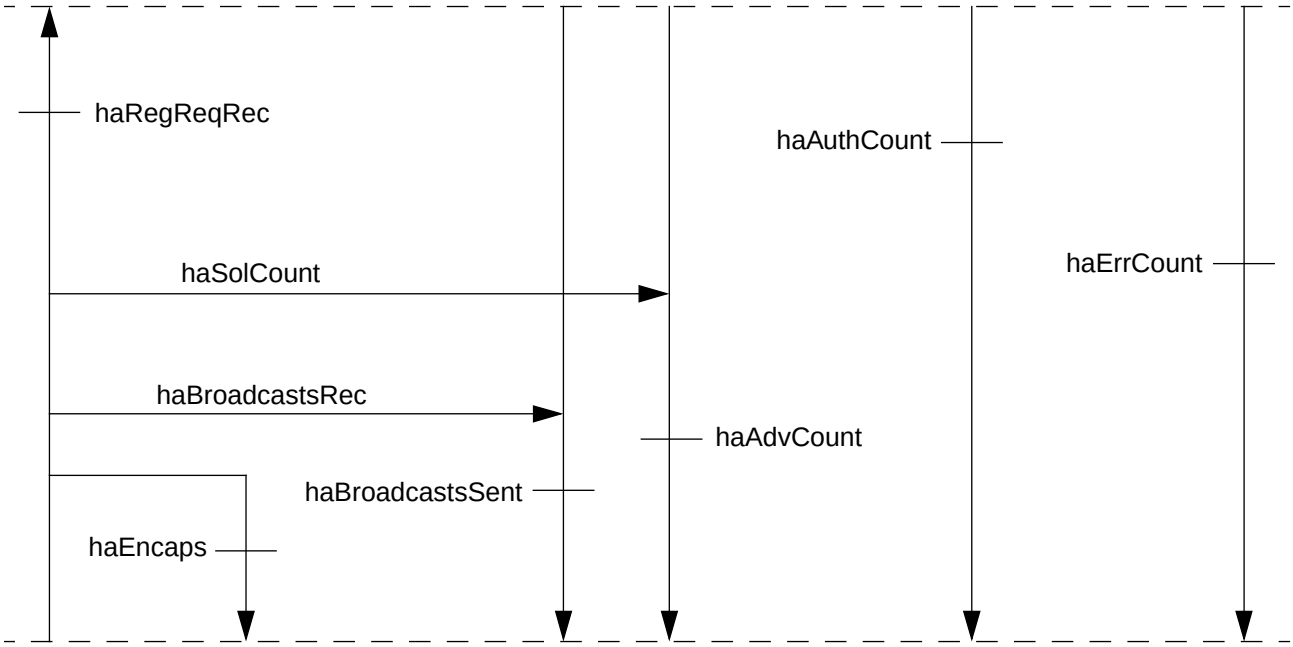

Figure 13. Case Diagram for the Home Agent

Figure 14. The Home Agent subgroup

5.3.1 Home Agent objects

#	Field Name	MIB Object Label	Datatype	Description
1	Mobility Binding Table	haBindingTable	SEQUENCE OF HaBindingEntry	The Home Agent's mobility binding table. See Table 11
2	Authorized Node List	haAuthNodeList	SEQUENCE OF HaANEntry	The Home Agent's list of authorized Mobile Nodes. See Table 12
3	Advertisement Sequence Number	haAdvSeqNo	Integer (0..65535)	The sequence number in the last sent agent advertisement
4	Advertisement Flags	haAdvFlags	Bit String	The flags field in the last sent agent advertisement
5	Advertisement Time Stamp	haAdvTS	Counter	The time when the last agent advertisement was sent
6	Advertisement Counter	haAdvCount	Counter	The total number of agent advertisements sent
7	Solicitation Address	haSolAddr	IpAddress	The IP address from which the last agent solicitation message was received
8	Solicitation Time Stamp	haSolTS	Counter	The time when the last agent solicitation message was received
9	Solicitation Counter	haSolCount	Counter	The total number of agent solicitation messages received
10	Error Address	haErrAddr	IpAddress	The IP address from which the last error message was sent
11	Error Code	haErrCode	INTEGER (0..255)	The error code in the last sent error message
12	Error Time Stamp	haErrTS	Counter	The time when the last error message was sent
13	Error Counter	haErrCount	Counter	The total number of error messages sent
14	Authentication Exception Counter	haAuthCount	Counter	The total number of authentication exceptions
15	Registration Requests Received	haRegReqsRec	Counter	The number of registration requests received at the Home Agent
16	Encapsulations	haEncaps	Counter	The number of IP-packets encapsulated at the Home Agent
17	Broadcasts Received	haBroadcastsRec	Counter	The number of broadcast packets received
18	Broadcasts Sent	haBroadcastsSent	Counter	The number of broadcast packets forwarded to Mobile Nodes

Table 10: The objects in the Home Agent subgroup

5.3.2 Home Agent Mobility Binding Table

#	Field Name	MIB Object Label	Datatype	Description
1	Mobile Node	haBindingMN	IpAddress	The home address of the Mobile Node
2	COA	haBindingCOA	IpAddress	The care-of address of the Mobile Node
3	Lifetime	haBindingLifetime	Integer (0.65535)	The lifetime for this registration
4	Flags	haBindingFlags	Bit String	The flags field for this registration

Table 11: The Home Agent's Mobility Binding Table

5.3.3 List of Authorized Nodes

#	Field Name	MIB Object Label	Datatype	Description
1	Authorized Node	haANAddr	IpAddress	The IP address of an authorized mobile Node

Table 12: The Home Agent's list of Authorized Mobile Nodes

6.0 Testing the SunOS implementation

Before we started to port Anders Klemets' implementation of the Mobile-IP protocol to the MINT and Solaris, we thought that it would be a good idea to see how it worked on its original platform: SunOS 4.1.

This chapter contains a quite detailed description of what we did, to make it possible for others to reproduce these tests.

6.1 Installation

We downloaded version 7 of Klemets' implementation from

```
ftp://sics.se/archive/mobile-ip/
```

and compiled it, just typing `make`. This produced an executable file called `xmipd`, which is the Mobile-IP daemon. This program is used to start both the Foreign Agent and the Home Agent, as well as the Mobile Node, but with different configuration files. For example, to start a Foreign Agent you type

```
xmipd fa.cfg
```

where `fa.cfg` is the configuration file for a foreign agent. More about the configuration files below. Note that the implementation uses the Network Interface Tap (`/dev/nit`), which requires you to have root access (to open the device).

6.2 The environment

In the lab we had two subnetworks and a couple of Sun SparcStations that we could use for these tests. The configuration is shown in Figure 15.

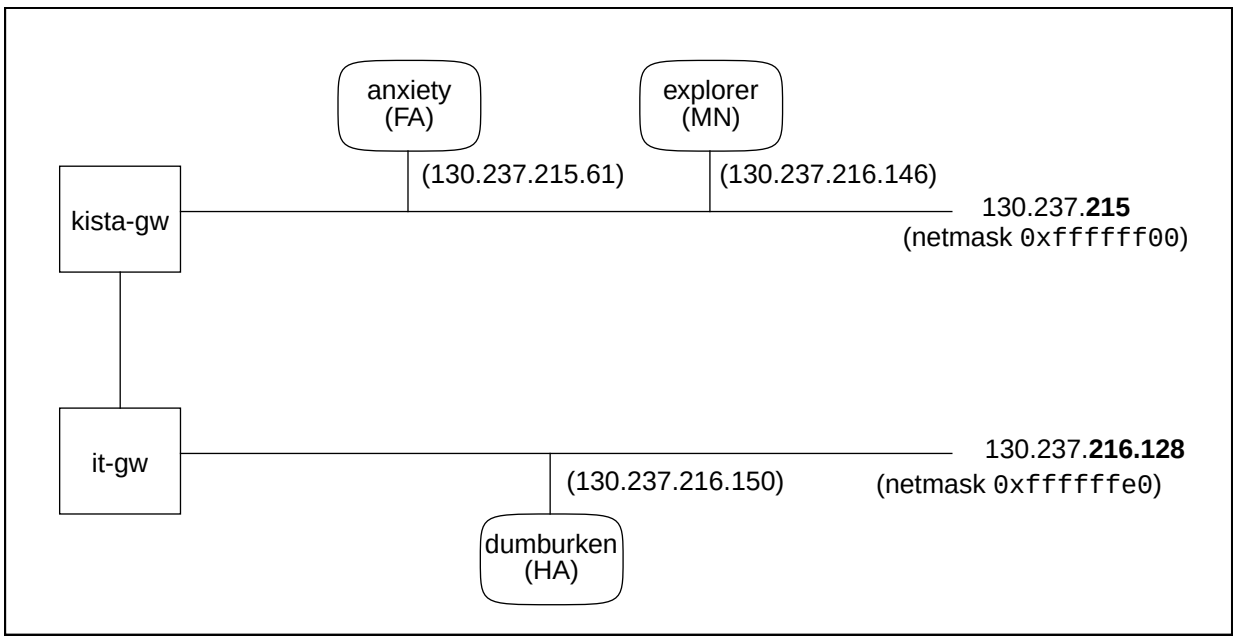

Figure 15. The subnetworks and the workstations in the lab

The netmask for the 215 subnet is `0xffffff00` (255.255.255.0), but for the 216 subnet it is `0xfffffe0` (255.255.255.224), which means that the 216 subnet is

itself subdivided into eight different sections. The three most significant bits in the last byte of the IP address determines which section we are dealing with (see below).

```
130.237.216.Y =
10000010.11101101.11011000.XXX*****
```

XXX	IP-addresses
000	0-31
001	32-63
010	64-95
011	96-127
100	128-159
101	160-191
110	192-223
111	224-255

We are using the subnet that has IP-addresses in the range 130.237.216.128 to 130.237.216.159.

6.3 Configuration

All three daemons (HA, FA and MN) have to be configured with IP addresses different from the real IP addresses on the computers where they are running, so we gave them addresses which were not used by any other computers in the department but still had the right network number. These addresses were then used in the configuration files for the different entities. The syntax of the configuration commands is described in the files `README` and `README.config` which are included in the Mobile-IP package by Anders Klemets. We will shortly describe the configuration setup that we used in our tests. Below is the configuration file for a Mobile Node.

```
myipaddr              130.237.216.146
ha                    130.237.216.150
key                   130.237.216.150 t n 12345 this_is_a_secret
key                   130.237.216.150 r n 13588 this_is_another_secret
lifetime 40
interface le0 8:1:20:1:2:3 130.237.216.146 255.255.255.224
route add default 130.237.215.1 1
```

The command `myipaddr` establishes the IP address of the Mobile Node. This IP address does not belong to any *fixed* computer. The actual IP address for the workstation “explorer”, where the Mobile Node is running, is 130.237.215.41.

The `ha` command specifies the IP address of the Home Agent. (This address should correspond to `myipaddr` in the configuration file for the Home Agent). This also indicates that this configuration file is describing a Mobile Node. Every configuration file must contain one of the commands `ha`, `fa` or `mh`.

The `key` command is used for authentication purposes. Here you specify the secret key to be used when communicating with a certain IP address, in our case the IP address of the Home Agent. The letter “t” indicates that this key will be used when

transmitting packets only, while a letter "r" means *receiving*. The third argument, which here is an "n", says that *nonces* will be used for replay protection, instead of *timestamps* ("t"). The fourth argument is a pseudo random Security Parameter Index and the last argument is the authentication key, given as a character string. More details about these arguments can be found in the `README.config` file mentioned above.

`Lifetime` sets the maximum value of the registration lifetime, in seconds. This value is used in the registration requests sent from this Mobile Node.

The `interface` command is used to configure a physical network interface. The first argument is the name of the real interface on our computer which is "le0", the ethernet interface. Next argument is the hardware address to use. Since we do not want to interfere with the normal IP traffic to our workstation, we use a made up ethernet address. This will make the packets that we are interested in to go to our separate protocol stack in user space. Any valid ethernet address can be used, as long as it is not used by another interface on the same network. As all Sun computers with Lance ethernet cards have ethernet addresses starting with 8:0:20, it was safe for us to use addresses starting with 8:1:20. The third argument is the IP address that is to be used on this interface. This should be the same value as the `myipaddr` value, and not the real IP address of the interface. The last argument is the netmask to be used with this IP address.

The last command in our configuration file is the `route add` command which adds an entry to the IP routing table. Here we specify the default router to be used from the Mobile Node.

Note, a much safer method of assigning new ethernet link addresses, that simply making up an ethernet address, is to apply for your own set of addresses. The Institute of Electrical and Electronics Engineers, Inc. has been designated by the ISO Council to act as the registration authority for the implementation of International Standards in the ISO/IEC 8802 series. For further details contact:

IEEE Registration Authority
IEEE Standards Department
445 Hoes Lane, P.O. Box 1331
Piscataway NJ 08844-1331
phone: (908)562-3813
Fax: (909)562-1571
Email: i.ringel@ieee.org

6.4 Running the system

We started up the different entities as described in Figure 15, that is the Home Agent at dumburken, the Foreign Agent at anxiety and the Mobile Node at explorer. To see that it really was working, we used the `ping` command to send a packet from a workstation on another subnet (the workstation artigonn on the subnet 130.237.213) to the IP address of the Mobile Node. The ping test was successful.

We later used the SunOS implementation and the configuration described here quite a lot when we were doing the performance tests. More about this in Section 10.0.

7.0 The SNMP implementation

To implement the Network Management functions that we needed in the Mobile-IP code, we used the cmu-snmp2.1.2 package. There were several reasons for choosing this tool. Firstly it is available for free from the Carnegie Mellon University (CMU) and secondly it is widely used.

7.1 The system

The environment we used to test our system was the same as the one described in Section 6.2 on page 30 with the difference that on each of the SparcStations we were running an SNMP agent as well (as the MobileIP code). To monitor the different components we used our own manager program, called `mipwatcher`, which is described in Section 7.5.

There are three types of entities in our system; the SNMP Manager, the SNMP Agent and the Mobile-IP daemon as shown in Figure 16. The communication protocol between the Manager and the Agent is SNMPv2. Between the Agent and the Mobile-IP daemon we created a simple request-reply protocol to run over UDP. In our system the Agent and the Mobile-IP daemon run on the same computer but they could as easily run on different ones as described in Section 7.4. The Agent is thus a proxy agent.

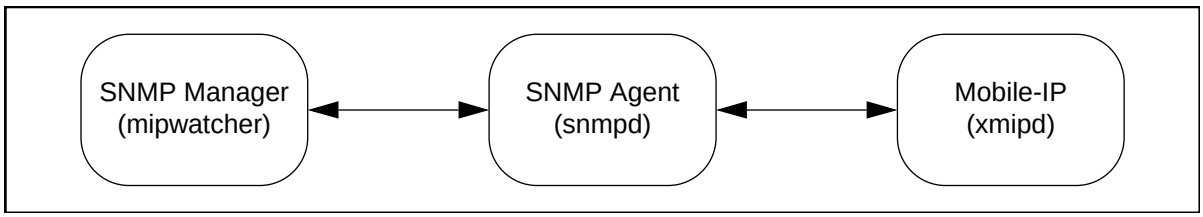

Figure 16. SNMP communication

7.2 The SNMP agent

The SNMP agent was implemented using the cmu-snmp2.1.2 package. The code in the agent is rather simple. For a basic understanding of the cmu-snmp2.1.2 see Appendix A.

7.2.1 The functions

There are a few functions worth mentioning. In `snmp.c` *connect_mipd* is called to open a UDP socket for the communication with the mipd. The address is given by *snmp_addr* and the port number by *snmp_port.* Both can be changed using switches on the command line as described in Section 7.4. Since all variables in the Mobile-IP MIB are very similar in type, there are only three lookup functions. *var_mip* takes care of all simple variables, *var_miptype* handles the miptype variable and *var_mipEntry* looks up all tables and lists. The *var_miptype* function could be included in *var_mip*, but is separately defined for improved readability. Basically all three lookup functions work as follows. First a check is made to see if we can handle this request. If so, a packet is sent to the Mobile-IP daemon (xmipd) asking it to retrieve the variable asked for, and then wait for an answer. If no answer is returned within *TimeOutTime* seconds, retransmit the request, but no more than *NumOfRetrans* times.

To send a packet to xmipd a call is made to the *SendReadReq* function. First, build a readpacket containing (in order) the R/W value, the magic value, the exact value and the clength value. If *current*, which is the matching prefix in the subtreelist, is not NULL include it next in the packet. If *current* is not NULL also put in the *rlength* and the requested OID (*request*). When the packet is done, send it to *MIPsock*.

After the request is sent, a call to *GetResp* is made to wait for a response from mipd. *GetResp* waits *TimeOutTime* seconds for a response, and if none is received *TIMEOUT* is returned. If a packet is returned from mipd, first check the *error* value. If it is non-zero, this is an error packet which only includes the error code and *GetResp* just returns the error code. However, if *error* is zero, the found value length is fetched and the value put into **value*. If the caller of *GetResp* is interested in the length of the value, that is *vlength* is not NULL, put the length into **vlength*. Next, get the length field of the found OID. If it is non zero, get the OID from the packet and update **olength* and **oidfound*.

7.2.2 The protocol

To communicate between the SNMP agent and the Mobile-IP daemon (xmipd), a simple request-reply protocol over UDP was implemented. A request sent from the agent to xmipd looks like this. The first byte indicates whether this is a read or write request. A read request is indicated by the value 1 and a write by 2. The next byte is the magic number hint found by the agent. Then the exact value occupies the third byte and the length of the *vp->name* field, given in bytes, the fourth. If xmipd is not interested in the *vp->name*, this last field can be set to zero. It is then automatically assumed that there is no more data in the packet. If the *vp->name* is to be included in the request, it follows the length field. After that, the length field of the requested OID, also in bytes, occupies one byte. If it is not zero, then the requested OID is put at the end of the packet.

A response from xmipd to the agent has the following structure. The first byte is the error code. If an error was encountered during the lookup of a variable, the error code should be set to a non-zero value and no more data is expected in the packet by the agent. If the lookup of a variable was a success, then the following byte is the length, in bytes, of the value found. After that the value itself is included. The next byte is the length, in bytes, of the OID returned. This value can be zero if the agent does not need the found OID, and then this is the last byte in the packet. Otherwise the OID occupies the last bytes in the packet.

7.3 Changes to the Mobile-IP implementation

To be able to get information concerning the state of a Mobile-IP daemon, code had to be added to collect statistics, store interesting values and to communicate with the SNMP agent. One design decision made was to run the Mobile-IP daemon and the SNMP agent as separate processes. This was to facilitate the porting of the Mobile-IP code to the MINT, but also to be able to run the SNMP agent on a different machine if so desired. To incorporate the agent into the Mobile-IP daemon should not be a difficult process.

7.3.1 Structures

To store the collected statistics, a global variable *mipstat* was created. Its definition can be found in Appendix B.2.3. Basically it has one entry for every simple variable in the Mobile-IP MIB.

Most of the data of interest to the tables and lists in the MIB were already contained in different structures in the code. Though some changes had to be made and they are pointed out in Appendix B.1.

7.3.2 Functions

The functionality added is basically a number of functions to get a request from the SNMP agent, fetch the variable asked for and send a reply. All new functions written are contained in the files *snmp.c* and *snmp_init.c.* Almost all other changes to the original Mobile-IP implementation involve simply updating the *mipstat* variable and thus can easily be found.

The order of events is roughly that in *ioListenForPackets* a call is made to *snmp_socket_init* to open a UDP socket to use for the communication to the SNMP agent. Then we wait for packets to arrive at that socket in the function *bsdNITInput* and when it does, *snmpHandleReq* is called to take care of the request. The only job of the *snmpHandleReq* function is to read the request from the socket and get the first byte to determine whether this is a read or write request and call the corresponding function. When a read request is received, it is up to *snmpHandleReadReq* to extract the rest of the information from the packet, determine which variable is asked for, fetch the value and call *snmpSend* to send an answer. If, during the variable lookup, an error is discovered, *snmpERROR* sends the corresponding error packet to the SNMP agent. The function *snmpSend* calls *snmpMakeReply* to make a reply packet, and then sends that packet. When done, we wait for more requests.

7.4 Configuration

One command has been added to the configure file for the xmipd program. If for some reason the port number used for listening for requests form the SNMP agent is occupied it can be changed by the command ‘Port number’, where number is the new port. The default port is 0xFFD3.

If you want the SNMP agent to run on a different machine than the Mobile-IP daemon you can start snmpd with the switch [-ma ipaddress], where ipaddress is the IP address of the computer on which the Mobile-IP daemon is running. Also, if the port on which the Mobile-IP daemon is listening for requests has been changed, the switch [-mp newport] tells the snmpd about it.

7.5 Mobile-IP Watcher

Mobile-IP Watcher, or `mipwatcher`, is an SNMP manager program that we have created to be able to monitor the different entities in the Mobile-IP protocol and to display it in a nice format. It is based on the `snmpwalk` application in the CMU SNMP package. Mipwatcher calls the snmpwalk program with certain parameters, and displays the formatted result in a window on the screen. The program is written in the script language Tcl/Tk, and needs the application `wish` to run.

The source code for our program can be found in Appendix F.

7.5.1 How to use `mipwatcher`

Before you start the program, make sure that you have set the environment variable MIBFILE to point to the MIB file that you want to use. The program will not start if the MIBFILE variable is not set.

To start the program, go to the directory where mipwatcher is located and simply type `mipwatcher`. If this does not work, it is probably because it cannot find the

`wish` program, which is assumed to be in the `/usr/local/bin` directory. This can be helped by changing the path on the first line in the `mipwatcher` file to point to the executable wish on the current system.

The first screen to appear when you start the program will look like Figure 17.

Figure 17. Main menu

Here you choose which entity you want to monitor. Simply press one of the buttons “Mobile Node”, “Foreign Agent” or “Home Agent”, or choose “Quit” if you want to exit the program. If you choose one of the first three alternatives, a new window will appear that asks you to enter the IP address of the snmp agent (Figure 18), which is probably the same address as where the corresponding mip daemon is located.

Figure 18. Enter the IP address

Enter the IP address (textual names works fine) and press return. Now the monitor window appears, which will look a bit different depending on which entity you have chosen to watch. Below is the screenshot from the Mobile Node (Figure 19).

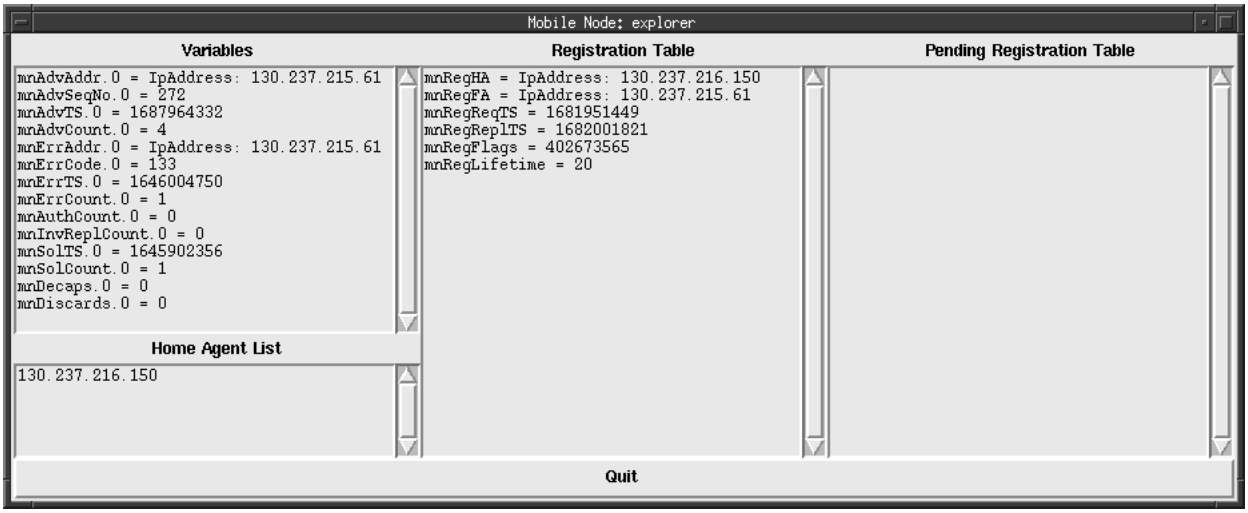

Figure 19. Mobile Node monitor window

Here you will see all the variables in the MIB which belongs to the selected entity. The scalar variables are listed in one window, and each table variable has a window of its own, which makes it easy to read the information. The information is by default updated every fifth second, but this interval can be changed by changing the variable `g(delay)` in the program source.

8.0 Program development for the MINT

This chapter will in detail describe how to proceed when making programs for use in the MINT environment, regarding both user programs and the operating system for the MINT.

8.1 The system

Here is a picture describing our working environment (Figure 20).

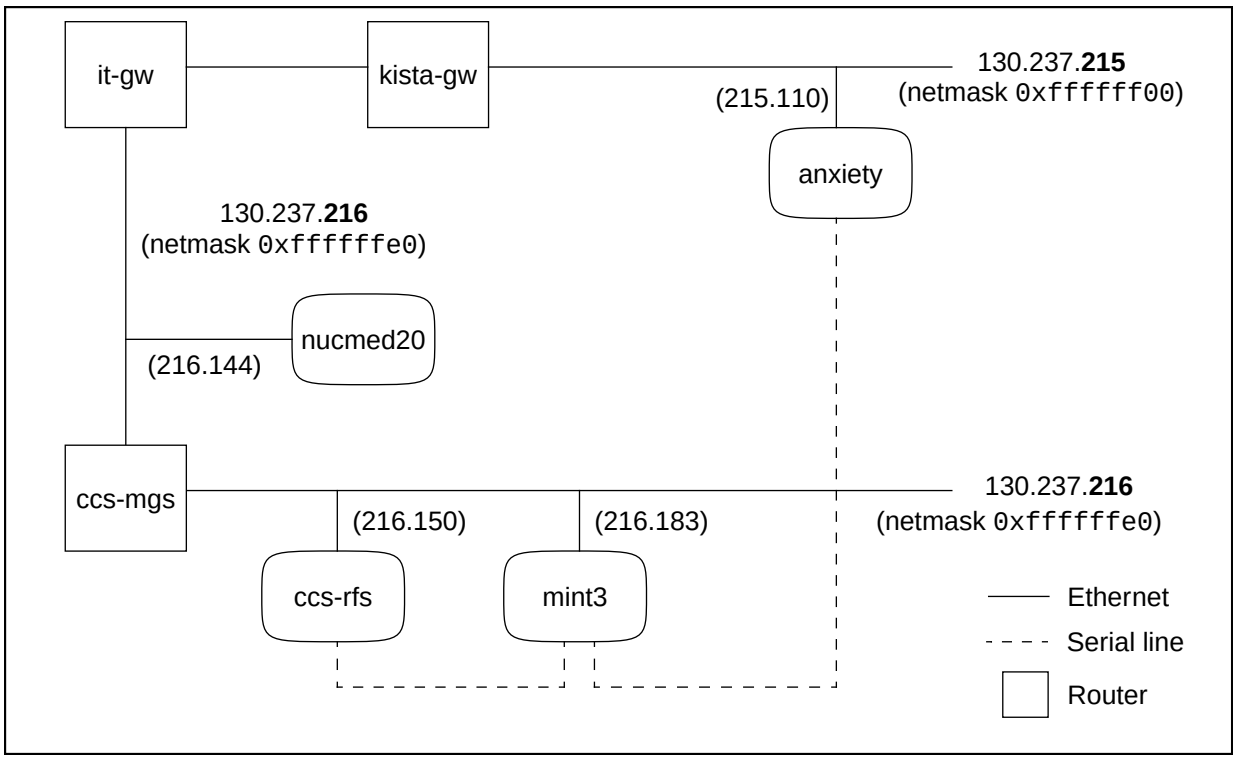

Figure 20. The subnetworks and the workstations in the lab

Anxiety is a SPARCstation 10 running SunOS 4.1.4, which is connected via a serial line to one of the serial ports on a MINT. This is used for remote debugging of programs running on the MINT. Kista-gw, it-gw and ccs-mgs are routers. Nucmed20 is a Hewlett Packard workstation that is acting as a boot server for the MINTs. When a MINT is booted, it fetches the programs from this computer using Bootp[26] and TFTP[25]. Ccs-rfs is a Toshiba PC running MachOS 2.6, which has a serial connection to the console port of the MINT that we are interested in (for example mint3). From this machine you can give commands to the built-in PROM monitor inside the MINT. This computer also acts as a file server for the MINTs. A MINT should be able to access the files on the Toshiba via RFS (Remote File Sharing).

8.2 Booting a MINT

This section will explain how to boot a MINT, assuming the working environment described in the section above (Section 8.1).

First you will have to log in to the Toshiba (ccs-rfs), which should be connected through a serial cable to the console port of the MINT you want to boot. The login

can be done locally at the machine, or remote via telnet, which can look like the example below (where all inputs from the user are printed in bold).

```
anxiety:~>telnet 130.237.216.164
Trying 130.237.216.164 ...
Connected to 130.237.216.164.
Escape character is '^]'.
ccs-rfs.electrum.kth.se TCP Telnet service.

2.6 MSD Mach (ccs-rfs.electrum.kth.se) (ttyP0)

login: d91-fta
Password:
```

When the login is done, you can start a kermit program, which lets you connect to a MINT via a serial link. Tell the program which line and speed to use by using the commands "set line" and "set speed":

```
% kermit
C-Kermit, 4F(077) 1 Apr 89, 4.2 BSD
Type ? for help
C-Kermit>set line /dev/tty02
Warning, read access to lock directory denied
C-Kermit>set speed 9600
/dev/tty02: 9600 baud
```

Now you can connect to the MINT by giving the "conn" command:

```
C-Kermit>conn
Connecting thru /dev/tty02, speed 9600.
The escape character is CTRL-\ (28).
Type the escape character followed by C to get back,
or followed by ? to see other options.
```

If the MINT has not been resetted before, now is the time to do that. By pushing the reset button on the MINT, the following text should appear on you screen:

```
MINT KTH/HPL, vers 2.3
@
```

If it does not appear on your screen, try hitting the return key once. The "@" character is the prompt. Now you can type commands to the built-in EPROM monitor. Typing a "?" will list the commands that are available in this version of the monitor:

```
@?
A -> ALTER bytes
B -> BOOT using TFTP
D -> DISPLAY bytes
```

```
G -> Go to address (LOADENTRY default)
I -> Re-INITIALIZE monitor
L -> LIST files in ramdisk
M -> Byte alter using LONG (32 bit) accesses
P -> PRINT environment variables
R -> Registers and flag display
S -> SET environment variables
T -> TRACE using remote GDB
U -> USE stored registers and go to addreess
W -> Byte alter using WORD (16 bit) accesses
```

The “p” command is quite useful. It displays the values of the environment variables, which you also can set with the “s” command.

```
@p
Debug                           0x0
GDBdebug                        0x0
Console                         0x0
LANCE                           A
Loadstart                       0x40000000
Loadentry                       0x40000000
Runflag                         0x1
Bootflag                        0x1
Bootfile
Bootdevice                      net
Availmem                        0x7c0000
EtheraddrA                      08:00:09:00:69:63
EtheraddrB                      08:00:09:03:04:c6
Hostname
IPaddr                          0.0.0.0
Subnetmask                      0.0.0.0
Gateway                         0.0.0.0
DNSserver                       0.0.0.0
```

Now you need to set the variable “bootfile” to the name of the file that you want to download. The files that you can download are presently stored in the directory `/usr5/tftpdir` on the machine nucmed20. To boot a MINT you can use the file `mach.boot`. After the variable “bootfile” is set, you type “b” to download the file.

```
@set bootfile mach.boot
@b

MINT bootp downloader:
Got Bootp reply from 130.237.216.144 (00:00:0c:00:29:94)
Our IP address is 130.237.216.183
Our subnet mask is 255.255.255.224
Our gateway is 130.237.216.163
Our DNS server is 130.237.212.6
Our hostname is mint3
TFTP server is 130.237.216.144 (00:00:0c:00:29:94)
Suggested boot file name is ‘/mintbootfile’
Downloading file ‘mach.boot’ from host 130.237.216.144
```

```
Image size: 0x232cfc bytes
Loading to: 0x40000000
Entry point: 0x40000000
|xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxx|
Downloaded 2305276 bytes
MINT Mach_3.0 VERSION(MK84): Thu Jan 18 11:29:16 MET 1996; kernel/STD+WS-
debug (anxiety.electrum.kth.se)

vm_page_bootstrap: 1228 free pages
pit0: at MC68901 timer 0, time = 0 secs : 0 nsecs, resolution = 10000000
nsecs
le0: at LANCE Ethernet 0: 8-0-9-0-69-63
le1: at LANCE Ethernet 1: 8-0-9-3-4-c6
ram0: at Ramdisk Controller 0 addr = 0x40041538, size = 1474560 bytes
ram1: at Ramdisk Controller 1 addr = 0xf0007448, size = 737280 bytes
Server directory? [ /dev/ram0/mach_servers ]
(default pager): Added paging file /dev/ram0/mach_servers/paging_file
(bootstrap): loading unix symbols from /dev/ram0/mach_servers/startup
(bootstrap): loading emulator symbols from /dev/ram0/mach_servers/emulator
(startup): server_dir(/mach_servers) on root.
(startup): emulator_path(/mach_servers/emulator)
(startup): first_program(/mach_servers/mach_init)
Mach_3.0 VERSION(UX42): Fri Dec 15 15:26:50 MET 1995; server/STD+WS
(anxiety.electrum.kth.se)

Availale memory = 4.58 megabytes
Unix tables: 1.10 megabytes
Unix buffer cache: 65 buffers 0.50 megabytes
flowat=126, fhiwat=226, ilowat=6, ihiwat=6
Base is Thu Jan 18 05:27:03 1996
Current time is Wed Dec 31 19:01:14 1969
This is strange -- CHECK AND RESET THE DATE!
Time is set to Thu Jan 18 05:27:03 1996
Automatic reboot in progress...
Thu Jan 18 05:27:11 EST 1996
Thu Jan 18 05:27:11 EST 1996
ufs_mount: file system not cleaned -- mounting anyways
flowat=29, fhiwat=58, ilowat=3, ihiwat=3
checking quotas: done.
starting system logger
standard daemons:.
starting network daemons: inetd.
starting local daemons:.
starting cmucs/mach daemons:.
Thu Jan 18 05:27:20 EST 1996


3.0 MACH (mint3.electrum.kth.se) (console)

login:
```

From the printout above you can see how the Mach operating system is loaded, and the Unix server is started. Then you can log in to the MINT, just as on any ordinary

Unix system. [Do you want to tell the reader what possible accounts there are? Or where they have to look to find out what accounts there are or to add more?]

8.3 Compiling programs for the MINT

If you want to develop programs for the MINT, there are several things you have to keep in mind. The MINT has a Motorola MC68030 processor, so the code you write must be compiled for that architecture. The normal procedure is to use a cross-compiler, and Anders Klemets has made a version of the Gnu Compiler (gcc) that runs under SunOS on a SPARCstation and produces machine code that can be run on a MINT. This compiler (together with a cross-assembler and a cross-linker) can be found in the `bin` directory under the mint root directory, which is `/afs/it.kth.se/misc/projects/walkstation/mint/`. Under this directory you will find almost all the files that are needed when working in the MINT environment, but it can sometimes be hard to find exactly what you are looking for, because there are about 27 000 files and subdirectories stored here. To help you find a particular file, you can look at the textfile called `files`, which is a listing of all the subdirectories and files under the mint directory. The best way to find something is to load the file '`files`' into emacs, and use the search-functions to find what you are looking for.

8.3.1 Stand-alone programs

A stand-alone program is a program that runs on the bare machine, without any support from an operating system. How such a program is compiled for a MINT is described by Anders Klemets in Appendix B in a master thesis report by Pascal Guerin[27].

8.3.2 Compiling the Operating System

To be able to run some standard applications on a computer, an operating system is needed. Mach 3.0 from Carnegie Mellon University (CMU) was the choice for the MINT. Why this OS was chosen, and how it was ported to the MINT is described in the paper "Mach 3.0 as an Operating System for the MINT" [18]. On top of Mach a Unix server (called UX) is run, which makes it possible to run ordinary BSD Unix programs on the MINT. In theory, all you have to do is to re-compile your favourite BSD Unix programs with the cross-compiler, and they should immediately work on the MINT. How this is done is described in Section 8.3.3.

The rest of this section describes how to compile the operating system itself. This is useful to know if you have to introduce changes in the operating system kernel or the unix server, but otherwise it can probably be skipped.

There are a number of different parts that are needed to be able to create a running Unix system on the MINT. The first of them is the Mach micro kernel, which can be compiled by using the script called `cckern` in the mint root directory (`/afs/it.kth.se/misc/projects/walkstation/mint/`). This script sets up a number of environment variables, and starts a special make program, called `odemake`. The program `odemake` works on two directory trees at the same time, one referred to as *basedir*, and the other as *masterbase*. Masterbase contains all the original source files, and in basedir you put the modified source files. Basedir will also contain all object files after the compilation. For the MINT environment, the following values will be true:

```
masterbase =  /afs/it.kth.se/misc/projects/walkstation/mint/mk-84/
basedir =     /afs/it.kth.se/misc/projects/walkstation/mint/mint-mk84/
```

You should never change any of the files in the mk-84 directory, but instead copy them to the corresponding location in the mint-mk84 directory and change them there.

Apart from the normal files that are needed to build the Mach kernel, a special file called `ram.o` is linked into the kernel when building it for the MINT. This file is an image of a small RAM file system, which is used as the root filesystem when starting the Unix server. This is necessary because the MINTs are diskless, and the Unix server needs to read and write several system files when booting. The file `ram.o` can be created by using a script called something like `ccramdisk` in the mint directory. This is a very ugly script that takes the contents of a floppy disk, which contains the file system and the necessary files, adds a small header and places the resulting file (`ram.o`) in the correct directory (which is `mint/objs`).

The files on the ramdisk must of course be compiled for the MINT architecture. Source codes and binaries for several standard Unix programs, e.g., `ls`, `mkdir` and `kill`, can be found under the directory `sup-i386`. A useful command if you do not know which architecture a program is compiled for, is the `file` command. For example, this is how it *should* look when executing the `file` command on an `ls` program compiled for the MINT:

```
anxiety:~/mint/sup-i386/src/bin>file ls
ls: mc68020 demand paged
```

Another part of the system which has to be compiled is the Unix server, called ux. This is done by using the `ccux` script. This script produces a file called `vmunix.UX42.STD+WS` which is places in the directory `special` under the mint root directory. This file should be copied to the `mach_servers` directory on the floppy disk containing the RAM filesystem, but it *must be renamed* to `startup`.

For the kernel to be able to start the Unix server, it needs a bootstrap program. This program is piggy-backed at the end of the mach kernel, and it can be compiled by using the script `ccbootstrap`.

The normal sequence of steps to compile a complete system for the MINT can be described as follows:

- Compile the Unix Server (with the `ccux` command). Copy the program to the floppy disk.
- Compile the Bootstrap program (`ccbootstrap`).
- Compile the ramdisk (`ccramdisk`). The file `ram.o` will now contain the contents of the floppy disk, which is your new Unix Server and some Unix commands.
- Compile the kernel (`cckern`). Apart from the kernel itself, the script produces a bootable file which contains the bootstrap program and the ramdisk. This file, called `mach.boot`, should be put in the bootdirectory for the MINT (`/usr5/tftpdir` on the computer nucmed20, which is the TFTP server), where it can be downloaded and used to a boot a MINT.

8.3.3 Compiling Unix Applications

When you compile Unix programs for the MINT, there are a few things that you must remember. Firstly, use the correct compiling tools, this means the cross-compiling versions of gcc (the compiler), ld (the linker) and as (the assembler).

Secondly, find the correct include files and library files. At present, there is no single `include` directory, but the include files are spread over at least four different directories. The situation is the same for the library files, but there is a directory called `libs` under the mint root directory which contains soft links to a few of the most commonly used library files. Here is an example of a simple Makefile that can be used when compiling Unix programs for the MINT:

```
MINT = /afs/it.kth.se/misc/projects/walkstation/mint

CC = $(MINT)/bin/gcc
AS = $(MINT)/bin/as
LD = $(MINT)/bin/ld

I1 = $(MINT)/mintmk84/export/sun4_mach_X_mint/include
I2 = $(MINT)/mach-i386/include
I3 = $(MINT)/mint-mach/src/ux/server
I4 = $(MINT)/sup-mk82/src/ux/server

LIB = $(MINT)/lib

CFLAGS = -Wa,-mc68030 -msoft-float -nostdlib -nostdinc -I$(I1) -I$(I2)\
 -I$(I3) -I$(I4) -Dmint

OBJS = hello.c

all: $(OBJS)
     $(CC) $(CFLAGS) -o hello $(OBJS) $(LIB)/crt0.o $(LIB)/libc.a
```

The flag `-Wa,-mc68030` tells the assembler that it should be prepared for mc68030 assembler code instructions. Since the MINT does not have a floating point unit, the `-msoft-float` flag tells the assembler to convert all floating point operations to calls to software library routines.

You must also tell the compiler not to use the standard include and library files (the switches `-nostdinc` and `-nostdlib`), and instead specify the correct library files to use. Almost all programs need the `crt0.o` and `libc.a` files.

8.4 Remote debugging using the GNU Debugger

When developing software there is one thing you can be sure of; your program will contain undesirable features, sometimes referred to as bugs. Luckily there are tools to help the programmer find these. One tool is the GNU DeBugger (GDB). Apart from being an excellent debugging program, GDB has the ability to remotely control the execution of a program, i.e. GDB can run on one machine and debug a program on another.

A version of GDB is provided for use with the MINT. With it comes an initialization file, *.gdbinit*, that defines a command called mint-restart which initializes the MINT for remote debugging. If the file *.gdbinit* is in the user's root directory, GDB will load it automatically and execute mint-restart when necessary.

8.4.1 Stand-alone programs

On our first trials using GDB we tried to use it to control the Mach kernel on the MINT. The reason for doing this was that the remote file system (RFS) for the

MINT was not working properly. Below is a step by step description on what to do to make GDB work.

Login on the MINT as described in Section 8.2. When you get a prompt, do the following:

```
@set bootfile mach.boot
@set runflag 0
@b
```

Setting *runflag* to 0 informs the monitor that it should NOT start the program when the downloading process has finished. Default is 1. When the file is downloaded you can start the program. The following command starts the program and generates a breakpoint on the first instruction which gives control to GDB.

```
@T
```

Next, start gdb on anxiety.

```
anxiety>gdb -b 9600 mach.boot
GDB is free software and you are welcome to distribute copies of it
 under certain conditions; type “show copying” to see the conditions.
There is absolutely no warranty for GDB; type “show warranty” for details.
GDB 4.11 (sparc-sun-sunos4.1.1 --target m68k-unknown-aout),
Copyright 1993 Free Software Foundation, Inc..
(gdb)target remote /dev/ttya
```

When GDB returns you should be all set to debug the program.

8.4.2 UNIX processes

Anders Klemets has made a version of GDB that should be executable on a MINT, but it is probably not a good idea to try to use that one because it is quite large, which means that there is not much space left in the memory for the program you want to debug. A better idea is to run GDB remotely (as mentioned above). GDB supports remote debugging, but there are some hardware specific routines that have to be written before you can start debugging your program. These routines are used to handle interrupts and to generate breakpoints, which will make GDB able to take control of the program execution. Fortunately, theses routines were already written for the MINT by Klemets, but for some reason we were never able to get it to work properly. The problem was that we were not able to find some essential include files when we tried to incorporate the GDB support in our own program.

9.0 Porting the Mobile-IP code

As part of our degree project we have ported the Mobile-IP code written by Anders Klemets for SunOS to two platforms; Solaris 2.4 and to the MINT (which runs a Unix server on top of MachOS 3). The reason for porting the code to Solaris is that the Department of Teleinformatics, where the work is conducted, will change their operating system from SunOS to Solaris in a near future.

The two ports have much in common. There are basically two parts of the code that have to change when moving from one UNIX to another, and those are the snmp daemon and the code handling the sending and receiving of ethernet frames. The rest of the code is standard C-code using standard UNIX commands.

9.1 Porting to Solaris 2.4

The Mobile-IP code for SunOS uses a device called the Network Interface Tap (`/dev/nit`) to get access to the ethernet packets sent on a network. This device is not supported in Solaris 2, which meant that we had to re-write the low-level routines that read and write ethernet frames. The means of getting direct access to the datalink-level frames in Solaris is by the Data Link Provider Interface (DLPI). This is a stream interface that provides the same functionality as the Network Interface Tap on SunOS and therefore the changes in the code are isolated to a few files. The files `bsdnit.h` and `bsdnit.c` are replaced by `dlpi.h` and `dlpi.c`. The file `dlpi.c` contains three functions; one for attaching a generic interface to a physical one, one for reading ethernet frames and one for writing ethernet frames, all with the same functionality as the ones in the SunOS code. In the file `dlcommon.c` there are several functions to help the programmer setting up the DLPI interface, heavily used by our code. For more information on how to use the DLPI read “How to Use DLPI” by Neal Nuckolls[28].

The only problem encountered during the porting was the filtering mechanism, pfmod. We did not get the filter to work properly so the filtering is done as following. The interface is configured to accept all ethernet frames, except the multicast addresses that we are not interested in. The multicast addresses of interest are added in the function *osJoinGroup* in the file `os.c`. When an ethernet frame is received, a test is made to check whether it has our ethernet address, or if this is a multicast or broadcast frame. All multicast frames received should be kicked upstairs since the interface only accepts multicast addresses that we are interested in. This is a minor improvement over the SunOS implementation where *all* multicast frames are read and filtering is done in user-space.

The snmp code was changed to work under Solaris by using the patch to cmu2.1.2 written by Yuri Rabover.

9.2 Porting to the MINT

The port of the Mobile-IP code to the MINT was one of the major parts of our degree project. Much time was spent not only with the port itself, but with understanding the MINT environment and the MINT operating system, and trying to get basic things to work (like booting a MINT).

9.2.1 Booting a MINT

At first we were not able to boot the MINT. This was because all the MINTs had been moved from one network to a new one, and all the network configuration files were wrong. For instance, the file `/etc/hosts` had to be updated with the new IP

addresses for the MINTs and the file servers, and the IP addresses of the name servers had to be corrected in the file `/etc/resolv.conf`.

9.2.2 RFS - Remote File Sharing

The MINTs have no hard disk or floppy disk drive, but only a small filesystem in the ram memory where a few important files are stored. Except from the Unix kernel itself and a few files that are used when booting the MINT, there are also a handful of useful user commands in the `/bin` directory. To be able to access files that are not in the ramdisk, the MINTs have support for something called Remote File Sharing (RFS). Remote File Sharing works like this. You create a small file which only contains an IP address. This file will function as link to a directory on a remote computer which is acting as an RFS server (see Figure 21). Now it is possible to access files on the other computer just as if they were on the local one. All disk operations that are issued on an RFS file are put in a packet and sent to the RFS server, which will process the command and send back the result, completely transparent to the user.

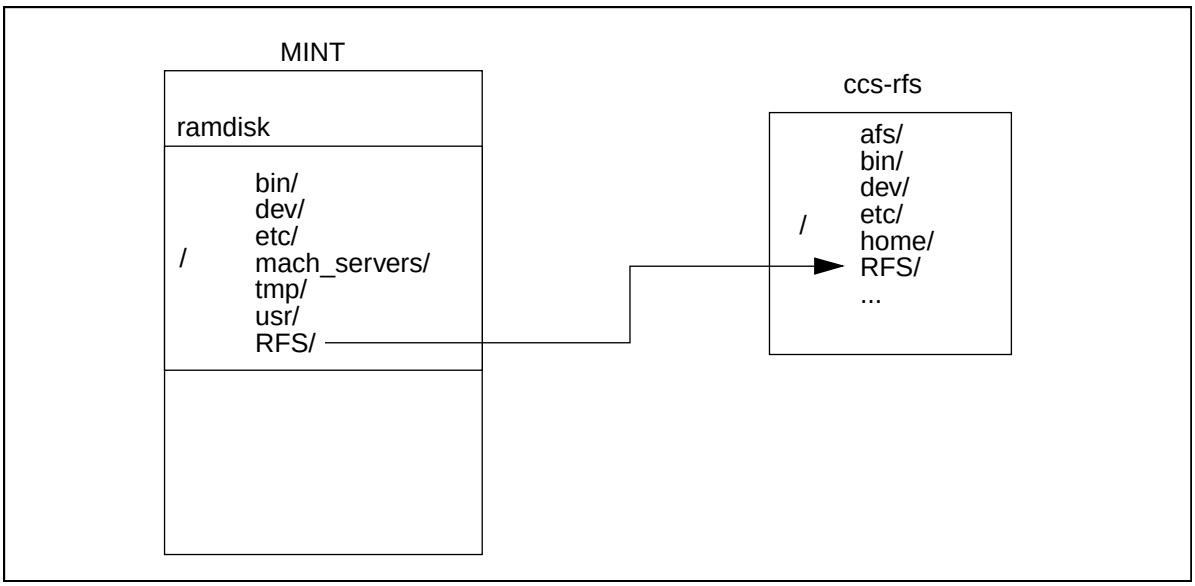

Figure 21. Remote File Sharing

In our lab the Toshiba computer (called `ccs-rfs`), which is located on the same subnet as the MINTs, was going to act as the RFS server. On this machine we intended to put our own programs (for example the Mobile-IP program), which the MINT then could get access to. Unfortunately this never worked, and we were never able to locate exactly where the problem was. We tried to debug the MINT kernel, and found the routines that should support the RFS functionality, but it seemed that the commands that were supposed to be sent to the RFS server were never sent out on the network.

After a few attempts to get the RFS to work, we started to think about other ways to make a MINT get access to external programs.

9.2.3 Running our program

The first and obvious solution was to put the program we wanted to run on the ramdisk. Then we should be able to start the program after we had booted the MINT, because the program would already be in the MINT's ram memory. The disadvantage of this method is that after every change in our program we have to

copy the program to the ramdisk and re-compile the whole MINT kernel, because the ramdisk is included in the kernel image. This procedure takes too much time to be useful.

Another problem was also that the ramdisk was almost full and our Mobile-IP program was too big to fit, even though its size was just around 100 kb. To overcome this problem we looked at the possibility of adding another ramdisk to the kernel, but after a few attempts we abandoned this idea because there was too much to change in the kernel code and we really would not gain that much of doing it.

Our next idea was to put a small file transfer program on the ramdisk, which could be used to fetch our Mobile-IP program after the MINT had booted. Then, of course, we again had the problem that the ramdisk did not have enough free space to store our program, but this could be circumvented by removing several programs from the ramdisk after the MINT had booted. The things that can be removed are for instance several programs and files in the `/etc` directory that are used only at start-up time.

9.2.4 A file transfer program

We compiled a version of the most commonly used file transfer program, ftp, for the MINT. First we had some problems to link the program, because it used floating point operations and the processor in the MINT does not have any floating point unit. This means that all floating point operations should be converted to software library routines, but according to a text written by Anders Klemets no such routines are currently available for the MINT. When we examined where the floating point variables were used in the ftp program, we discovered that they were used only in a small routine that displayed some information on the screen, and that these floating point variables easily could be substituted by integers. Now we were able to make an executable version of ftp for the MINT architecture, but unfortunately the resulting executable file was a little bit to large (around 76 Kb) to take up valuable space on the ramdisk. After trying the program on the MINT we also discovered that it was not very stable either, and crashed a bit to often.

Then we tried to compile a version of tftp (trivial file transfer protocol). This resulted in an executable file with a size of 37 Kb, which was better than the ftp, but not good enough. We tried the tftp program on the MINT, but were not able to contact the tftp server, so we gave that up and decided to write our own file transfer program.

We wrote a very simple file transfer program, which we called fssftp (Fredrik's super simple file transfer program). This includes a client part that is run on the MINT, and a server that runs on a workstation. The protocol is very simple and not very efficient, and the implementation is not optimized in any way, but it works and fulfils our purpose. The executable program for the MINT was only 9 Kb in size, which was quite alright. We put the program in the `/bin` directory on the MINT's ramdisk.

The syntax of the fssftp command is this:

```
fssftp ip-address src-filename [dest-filename]
```

When you for example want to fetch the Mobile-IP program (called xmipd) from a workstation to the MINT, you write like this on the MINT:

```
fssftp 130.237.215.110 xmipd
```

This assumes that an fssftp server has been started in the directory containing the file xmipd on the workstation with the IP address 130.237.215.110 (which is the computer anxiety). The fssftp server is started by typing the command `fssftpd`.

Now we had a way of transferring the Mobile-IP code to the MINT. The standard procedure when we developed the program was to first boot the MINT, then remove all the programs in the `/etc` directory on the ramdisk to make some space, and then download the Mobile-IP code. When we made any changes in the code it was quite easy to download the new version of the code (without having to re-boot the MINT).

9.2.5 Reading ethernet frames

The most difficult part of the port from SunOS to the MINT was how we should be able to read raw ethernet frames from the network. In the SunOS implementation this is done by opening a device called `/dev/nit`, the Network Interface Tap. That interface is SunOS specific and does not exist in the Unix version that is used on the MINT, which is a BSD Unix. Fortunately, BSD has a similar type of device called the BSD Packet Filter (BPF), which can be accessed by opening `/dev/bpf0`, `/dev/bpf1` and so on ([29], [30]). This lets you associate a device with a network interface (for example the ethernet interface) and install a filter to receive incoming packets selectively.

After some studying of the source code for the MINT mach kernel and the Unix server, we got the impression that there was support for BPF on the MINT. We modified the Mobile-IP code to open the bpf device instead of `/dev/nit`, and changed the routines for reading and writing ethernet frames. We also had to modify the code which defined the filter, because the filter code for nit and bpf are not compatible. The filter for BPF is written as a machine code program for a pseudo-machine. Here is an example of what a filter program could look like:

```
          lhd       [12]
          jeq       #ETHERTYPE_IP, L1, L3
L1:       ld        [26]
          and       #0xffffff00
          jeq       #0x82edd700, L2, L3
L2:       ret       #TRUE
L3:       ret       #0
```

This particular filter accepts all IP packets from the network 130.237.215.

Unfortunately we did not have any workstation running BSD Unix, so the first time we could try our code was when we downloaded the program to the MINT. This resultet in the error message “Can’t open /dev/bpf”. A quick look in the `/dev` directory on the ramdisk revealed that there was no file called `/dev/bpf0` or similar. This was a setback.

We added a bpf device in the `/dev` directory, but then we needed to add routines in the Unix server for opening and reading that special device, because this was not implemented. After a few attempts to use code from other BSD Unix implementations (for example netbsd), we realised that this problem must have been encountered before, and someone might already have solved it.

After some research on the Internet and a few postings in different newsgroups and mailing-lists, a solution began to appear. It seemed that BPF already was implemented, not in the Unix server, but in the underlying Mach kernel. No `/dev/bpf` was needed, instead we had to re-write the routines for opening and reading

ethernet frames so that they used system calls to the Mach operating system. This was not exactly what we had expected, but some example code showed us how to do it, and it was not that difficult to implement. This meant that the Mobile-IP program was no longer a pure Unix program, but a mix of Unix and Mach. Examples of Mach system calls that we are using are:

- device_open()
  To open the ethernet interface.
- device_set_status()
  To set the interface in promiscuous mode.
- device_set_filter()
  To configure the packet filter.
- mach_msg()
  To read an ethernet frame.
- device_write_request()
  To write an ethernet frame.

Though BPF was implemented in the kernal, there were some problems with the system calls. Two things could not be achived, and those were to put the interface into promiscuous mode and to set a non trivial filter on the interface. This means that the implementation will not work for all possible cases, but it should work for example when a MINT is acting as a Home Agent.

9.2.6 Unexpected problems

Once the MINT port could be tested an unexpected problem with the routine that calculates the checksum for the packets was discovered. This routine is used by several protocols, for example IP, UDP, ICMP and IGMP. We had earlier noticed that the checksum routine in the Mobile-IP code for SunOS did not calculate the correct checksum for UDP packets which had an odd number of bytes, but this was fixed by always setting the checksum to zero (which is allowed, but not very pretty). On the MINT, the checksum routine did not work at all, because it was optimized for the SPARC architecture and the program made several assumptions about the hardware which was not true when running on a Motorola 68030 in the MINT. We had to replace the ckecksum routine with a new one, but that was not optimized. The checksum routine is rather heavily used, and should actually be modified for each CPU to be as fast as possible. Even though this new routine was supposed to be platform independent it still does not work for UDP (but it does work for the other protocols). This probably has to do with the calculation of the checksum for the pseudo header which for some reason is not correct.

9.2.7 Summary of changes

The files `bsdnit.c` and `bsdnit.h` in the SunOS version of the Mobile-IP program has been replaced by the files `lowbpf.c` and `lowbpf.h`. In the file `lowether.c` the calls to bsdNITInput and bsdNITSendRaw have been replace by calls to bpfInput and bpfSendRaw, and in `xlowiface.c` the call to bsdNITattach has been replace by bpfAttach. Also, some minor changes have been made in the file `targetdefs.h`.

10.0 Analysis of the Mobile-IP protocol

To analyse the performance of the Mobile-IP protocol, or at least the particular implementation that we have, we have done a few experiments. These experiments and the conclusions we have drawn will be described in this chapter.

10.1 Delay

In our first experiment we wanted to measure the time it took for a packet to travel from a fixed host (the workstation artigonn) to a mobile node on a foreign network, compared with the time when the Mobile Node (MN) was at home. We used a modified version of the program `ping` to do the measurements. Normally ping just displays the time in milliseconds, but we wanted it to display microseconds too, so we had to modify the source and compile our own version.

Ping sends an ICMP ECHO_REQUEST packet to a host, and then waits for an ICMP ECHO_RESPONSE. We used a data size of 1000 bytes and set the count flag to 1000 packets.

10.1.1 Artigonn to explorer

In the first part of this experiment we measured the time to send a packet directly between two fixed hosts; artigonn and explorer (see Figure 22). This measurement will be used later when we calculate the time to reach the Mobile Node.

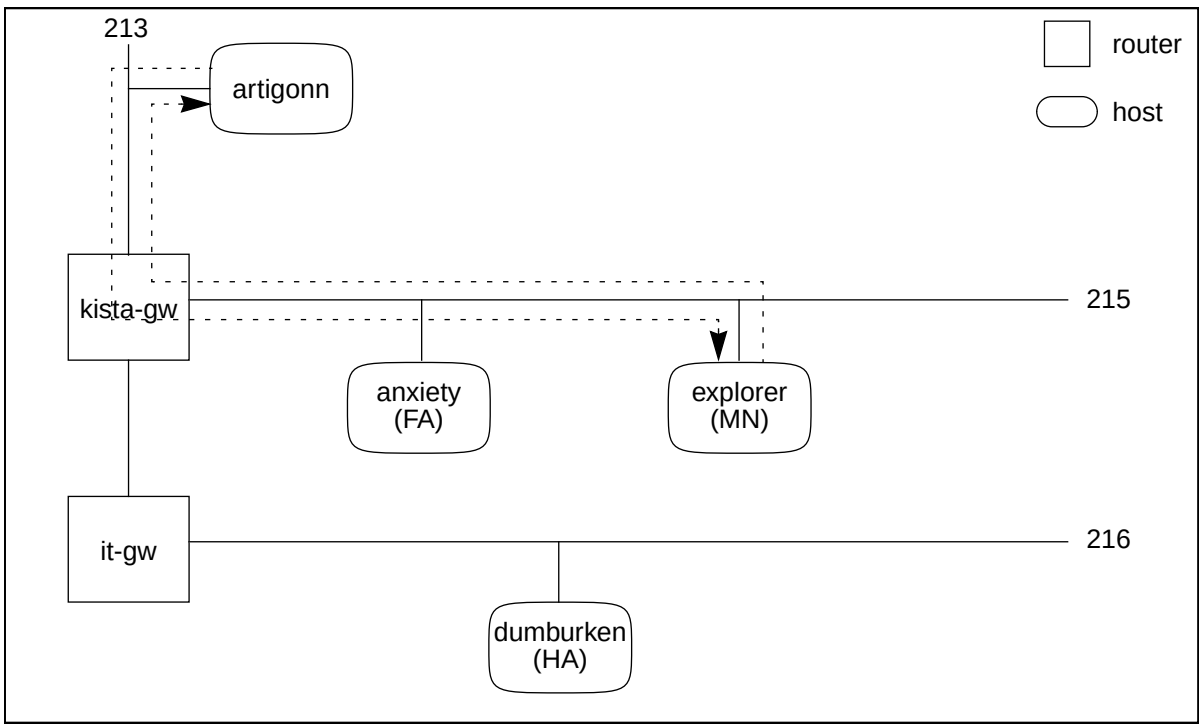

Figure 22. A ping from artigonn to anxiety

Results:
Minimum time = 6.8 ms
Maximum time = 25.2 ms

Average round trip time = 7.1 ms
Standard deviation = 1.0 ms

The interesting value here is of course the average round trip time, which is approximately 7 ms. This is the time it takes for a packet to travel from the sender (artigonn) to the receiver (explorer) and back.

10.1.2 Artigonn to dumburken

Here we wanted to measure the time for sending a packet to the Mobile Node when it is at home. The home network for our (virtual) Mobile Node is the 216 net, so we sent the packets to a host on that network; dumburken. See Figure 23.

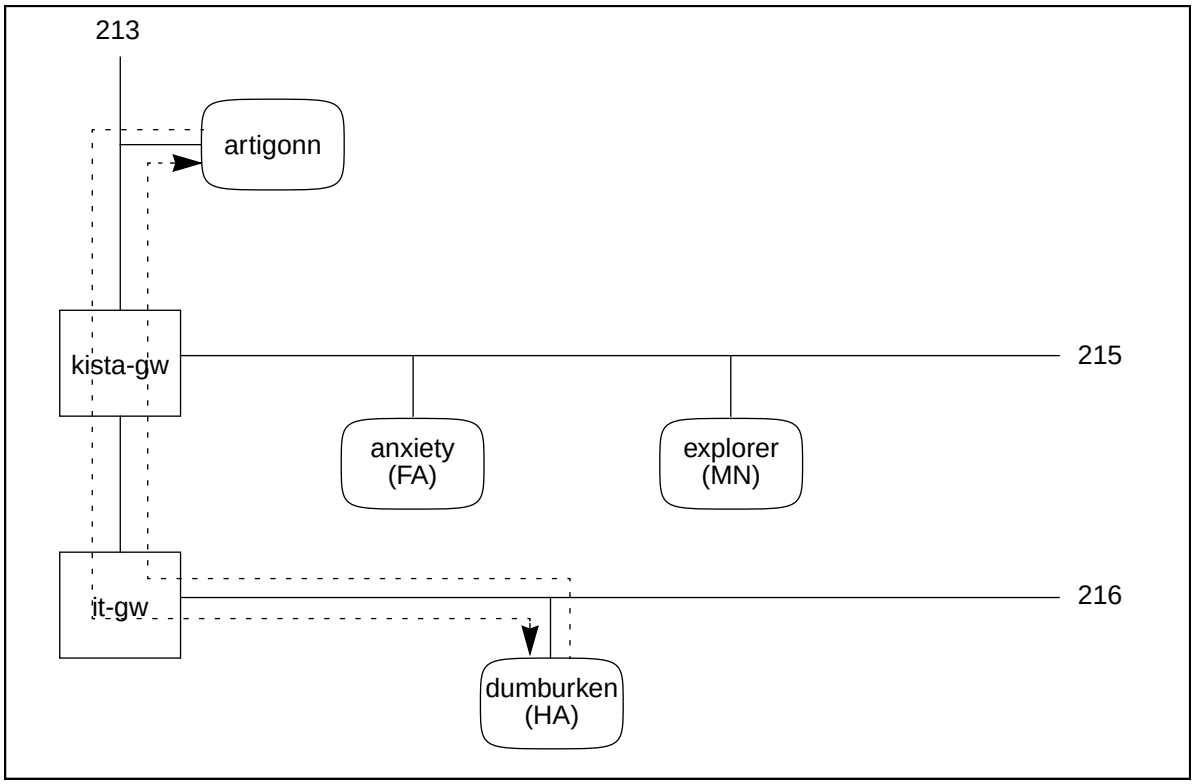

Figure 23. A ping from artigonn to dumburken

Results:
Minimum time = 7.0 ms
Maximum time = 15.4 ms
Average round trip time = 7.2 ms
Standard deviation = 0.4 ms

Here we can see that the round trip time is just a little longer than in the previous experiment, which could be expected, because we have one more router to pass on our way.

10.1.3 Dumburken to anxiety (HA -> FA)

We also wanted to know how long it takes to send a packet from the machine the Home Agent is running on (which is dumburken) to the machine on which the Foreign Agent is running (anxiety), and these are the results we got:

Results:
Minimum time = 5.3 ms
Maximum time = 9.6 ms
Average round trip time = 5.6 ms
Standard deviation = 0.4 ms

10.1.4 Anxiety to explorer (FA -> MN)

Also the time it takes for a packet to travel from anxiety (the Foreign Agent) to explorer (the Mobile Node) was of interest.

Results:
Minimum time = 3.0 ms
Maximum time = 8.6 ms
Average round trip time = 3.1 ms
Standard deviation = 0.3 ms

10.1.5 Artigonn to the Mobile Node

This experiment measured the time to ping the Mobile Node. The host explorer is acting as a Mobile Node and is assigned the IP address 130.237.216.146. We performed two versions of this experiment to see whether there were any differences between the two encapsulation methods minimal encapsulation and IP in IP.

This is what happens; Artigonn sends a packet to the Mobile Node's home network (which is the 216 net) where the packet is captured by the Home Agent (dumburken). The Home Agent encapsulates the packet and sends it to the Foreign Agent (anxiety) where it is decapsulated. Then the Foreign Agent forwards the packet to the Mobile Node (explorer), and finally the Mobile Node sends a reply to artigonn, which issued the ping command. See Figure 24.

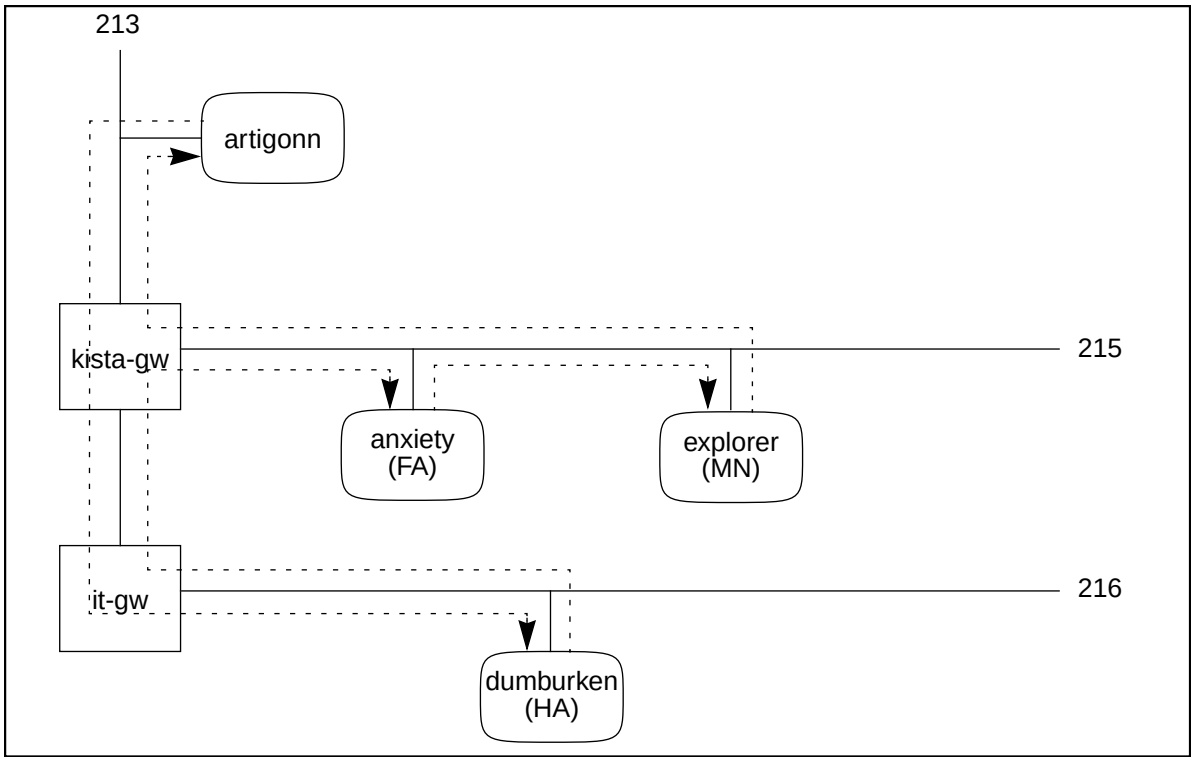

Figure 24. A ping from artigonn to the Mobile Node

Results (IP in IP encapsulation):
Minimum time = 14.7 ms
Maximum time = 27.7 ms
Average round trip time = 16.3 ms
Standard deviation = 1.4 ms

Results (Minimal encapsulation):
Minimum time = 15.6 ms
Maximum time = 26.3 ms
Average round trip time = 16.4 ms
Standard deviation = 1.4 ms

This experiment shows that there are almost no difference in performance between the two encapsulation methods.

10.1.6 Conclusions

The time to send a packet from the workstation artigonn on the 213 net to the Mobile Node when it is away from home can be calculated from the results in Section 10.1.1 and Section 10.1.5. Since there was no statistical determined difference between the encapsulation methods, the figures from IP in IP is used. The total round trip time was 16.3 ms, but this includes both the time to send the ICMP ECHO_REQUEST packet **to** the Mobile Node *and* to send the ICMP ECHO_REPLY packet **from** the Mobile Node back to artigonn. From part one of the experiment we see that the round trip time to ping anxiety from artigonn is 7.1 ms, which means that the time to send a packet in one of the directions is approximately half of that (which is 3.6 ms).

This implies that the time for a packet to travel from artigonn to the Mobile Node at explorer (via the HA and FA) is approximately (16.3 - 3.6) = 12.7 ms. This is roughly 3.5 times the time to send a packet to the Mobile Node when it is at home (Section 10.1.2) which is 7.2 / 2 = 3.6 ms.

An increase in time by a factor 3.5 is about what we had expected. When you send a packet directly to a host, the total time will be a sum of the transmission time and the propagation time (including the delay caused by the routers), but when a Home Agent is forwarding the packet to a Mobile Node on a foreign network (via a Foreign Agent), you will have three times the transmission time (at the sender, at the Home Agent and at the Foreign Agent), the propagation time will be longer and there will be some delay due to protocol handling (including packet encapsulation and decapsulation). The delay caused by the Mobile-IP protocol administration will be further examined in the following experiments.

10.2 Delay caused by encapsulation/decapsulation

The Mobile-IP protocol introduces a number of new places causing delay in the communication between two parties. In Section 10.1 we measured the total overhead time the protocol caused in the communication between a stationary host (artigonn) and a Mobile Node (explorer). In this section we will look more closely at the different entities in the process and how they contribute to the overhead.

10.2.1 The Home Agent

When a Home Agent receives a packet that is destined for one of the Mobile Nodes which it is serving, there are basically two operations it has to do. First, lookup the Mobile Node in the registration table to get its care-of address, and then encapsulate the packet and send it.

We have made two experiments to measure the time spent by the Home Agent from that it has received a packet until the packet is encapsulated but not yet sent. The difference between the two measurements were that in the first one IP in IP encapsulation was used and in the second one minimal encapsulation. The setup was the same as in Figure 24 and we sent 350 ECHO_REQUESTS of size 1000 bytes from artigonn to the Mobile Node.

Results (IP in IP encapsulation):
Minimum time = 2.8 ms
Maximum time = 6.0 ms
Average = 3.9 ms
Standard deviation = 0.4 ms

Results (Minimal encapsulation):
Minimum time = 2.5 ms
Maximum time = 5.1 ms
Average = 2.9 ms
Standard deviation = 0.3 ms

As can be seen above there is a difference between the two encapsulation method by 1 ms. This difference though did not show in the overall delay measured in Section 10.1.5.

10.2.2 The Foreign Agent

As with the Home Agent, the Foreign Agent has to do two things when receiving a packet. First, check if this packet should be forwarded, and if so, decapsulate and send it.

As in Section 10.2.1, two experiments were conducted. They both measured the time spent by the Foreign Agent handling the packet from the moment it receives the packet to when it has decapsulated the packet but not yet sent it. The two experiments differed as to which encapsulation method was used. Once again, the setup was the one in Figure 24 and the 350 ECHO_REQUESTS of size 1 000 bytes were sent from artigonn to the Mobile Node (via the Home Agent and the Foreign Agent).

Results (IP in IP encapsulation):
Minimum time = 2.3 ms
Maximum time = 4.6 ms
Average = 2.8 ms
Standard deviation = 0.5 ms

Results (Minimal encapsulation):
Minimum time = 2.1 ms
Maximum time = 6.1 ms
Average = 3.0 ms
Standard deviation = 0.7 ms

The difference between the two encapsulation methods was rather small, around 0.2 ms. However, the time it took to **de**capsulate a packet was about the same as the time to **en**capsulate it (Section 10.2.1). This seems reasonable, because the two operations are rather similar in complexity.

10.2.3 Conclusions regarding the delay

In the limited experiments that we have done, we could see that the time to administer a packet at the Home Agent and the foreign Agent constitute a large part of the total time to deliver the packet to the Mobile Node. This share of the total time will of course decrease when sending packets longer distances, but there are still time to save by optimizing this part of the code. It can be a good idea to incorporate support for tunnelling into the operating system kernel to speed things up, which has already been done in for instance Linux.

We also saw that there was not much difference between the two encapsulation methods that were used in our implementation.

10.3 Registration

Apart from the additional latency that the Mobile-IP protocol introduces, there are another time value of interest from the Mobile Node's point of view. That is the time to get a connection after it had booted, or after it had arrived at a foreign network. This time includes the time to get information about which Foreign Agents that are currently available, and the time to set up the connection by registering with the Home Agent (via the Foreign Agent).

10.3.1 The first registration

This experiment measured some time values of interest when a Mobile Node first gets started on a new, foreign subnet. The setup is described in Figure 25, and it is the same that we have used in the previous experiments.

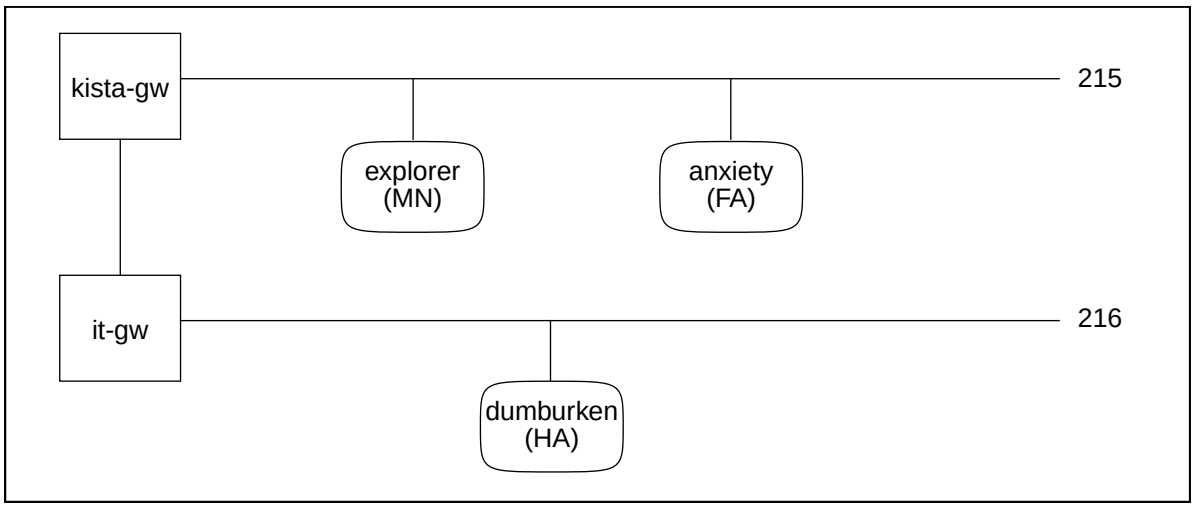

Figure 25. Registrations

First the Home Agent and the Foreign Agent were started. Then we started the Mobile Node, and the following data were collected (see Figure 26 for explanation):

- the time from the first solicitation sent by the Mobile Node until it got an agent advertisement (a)
- the number of solicitations sent by the Mobile Node before an agent advertisement was received
- the time from the first registration request was sent by the Mobile Node until a valid registration reply was received (b)

- the time between the first solicitation and the first valid registration reply. This indicates how long it takes for a Mobile Node to get a working connection after coming to a new network (c).
- the number of registration requests sent until the registration process was completed

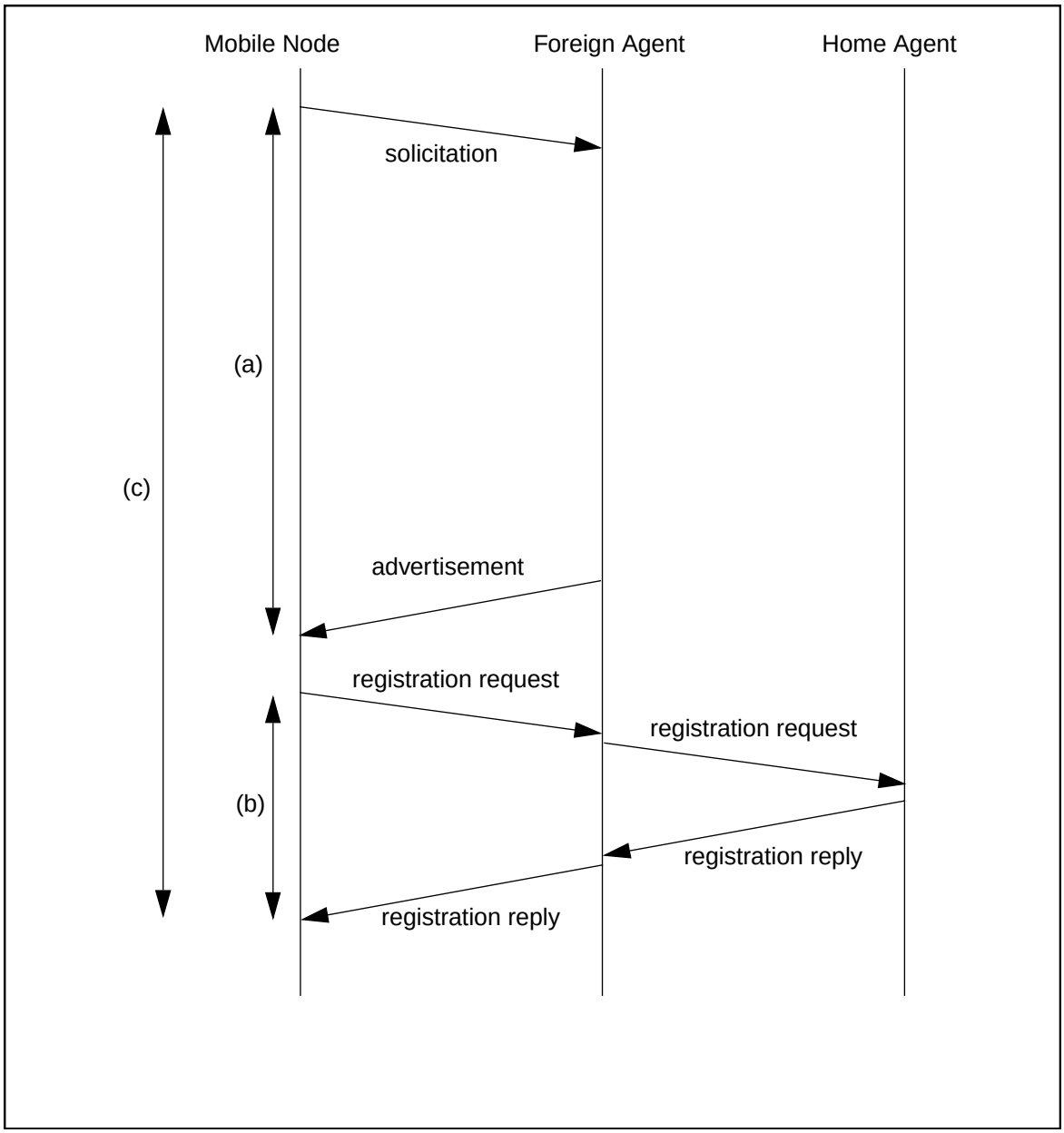

Figure 26. Sequence diagram for making a connection

(Solicitations are sent by the Mobile Node when it wants to know if there are any agents available, for instance when a mobile computer is connected to a new network. Agent advertisements are sent by the Home Agents and Foreign Agents to announce their presence.)

Each experiment was conducted 20 times, and the results are shown in Table 13.

	The time (ms) from first solicitation to first advertisement (a)	**The time (ms) from first request to first valid reply (b)**	**The time (ms) from first solicitation to first valid registration reply (c)**	**The number of solicitations sent before an advertisement**	**The number of registration requests sent before first valid reply.**
mean	*1 015*	*11*	*1 038*	*3.95*	*2*
std. dev.	396	2	396	1.54	0
min.	9	10	32	2	2
max.	1 996	16	2 017	6	2

Table 13: Registrations

10.3.2 Conclusions

The number of registration requests before a valid reply is received by the Mobile Node deserves a comment. Surprisingly this value is always 2. The reason for this is that when a Mobile Node first is started it has not yet agreed with the Home Agent on which nonce value to use. Therefore the first registration request is rejected because the Home Agent did not receive the nonce value it expected.

It should also be noted that over 98% of the time trying to get a connection is spent during the first phase where the Mobile Node is trying to contact a Foreign Agent. This is unexpected. Part of the explanation is that there are a number of packets that has to be sent before the first registration request can be transmitted. The order of events is like this; first the Mobile Node sends a Agent Solicitation (as a broadcast message). When this packet has been received and processed by the Foreign Agent, it sends an ARP request to get the Mobile Nodes ethernet address. The Mobile Node answers with an ARP reply. Now the Foreign Agent can send an Agent Advertisement to the correct ethernet address. When the Mobile Node has received the Agent Advertisement it knows the IP address of the Foreign Agent, but then it has to send an ARP request to get its ethernet address. Finally the Foreign Agent sends an ARP reply to the Mobile Node which now can send its first registration request. Another part of the explanation probably has to do with the large number of solicitations that has to be sent before an agent advertisement is received (a mean value of approximately 4). Why this happens has to be further examined, and it is important to find out whether this is a fundamental problem with the Mobile IP protocol itself or if it has to do with the particular implementation that we are using.

The extremely low minimal value measured between the first solicitation and the first advertisement was caused by the fact that during that particular test the Foreign Agent sent a multicast Agent Advertisement a very short time after that the Mobile Node had been started and thus there was no need for the Mobile Node to send any Agent Solicitations at all.

10.4 Throughput

Another aspect of the Mobile-IP implementation that is of interest is what the throughput is. The question is how many packets per second we can send to the Mobile Node when we are tunneling the packets, compared to the rate that a stationary system could handle. To get an idea of the efficiency of the code four different measurements were conducted.

10.4.1 Mobile-IP

In the first two experiments the setup was the same as in Figure 24. The goal was to see how many packets would get through from artigonn to the Mobile Node with different number of packets per second. On artigonn a small program was run that sent 10 000 UDP packets of size 1 000 bytes to the Mobile Node (via the Home Agent and the Foreign Agent).

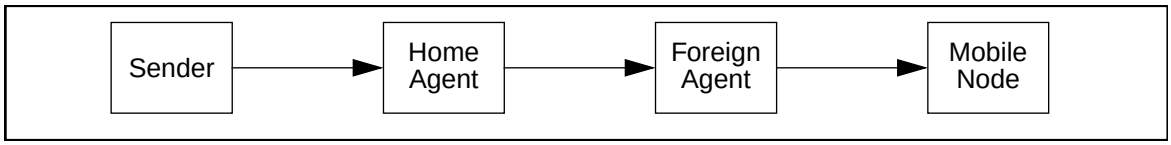

Figure 27. Throughput experiment

The experiments were performed five times each. The difference between the two experiments was the rate with which the packets were sent. In the first experiment the 10 000 packets were sent over a period of 200 seconds, and in the second experiment 13 seconds (as fast as possible). The result is displayed in Table 14 and in Table 15.

	# packets received by the HA	**# packets received by the FA**	**# packets received by the MN**
mean	10 000	9 997.8	9 996.6
std. dev.	0	2.3	2.6
min.	10 000	9 995	9 994
max.	10 000	10 000	10 000

Table 14: Throughput experiment 1

In the first experiment we see that all packets arrived at the Home Agent, and a very small number was lost between the Home Agent and the Foreign Agent.

	# packets received by the HA	**# packets received by the FA**	**# packets received by the MN**
mean	1 987.0	1 986.0	1 985.8
std. dev.	334.5	334.1	334.0
min.	1 397	1 397	1 397
max.	2 186	2 185	2 185

Table 15: Throughput experiment 2

From Table 15 we can see that a lot of packets were lost at the Home Agent. This is because the Home Agent did not manage to take care of all incoming packets. The packets that actually were taken care of were delivered almost without any loss to the Foreign Agent and then to the Mobile Node.

10.4.2 SunOS

In the next two experiments (experiment 3 and 4) the purpose was to see if the number of packets lost would be grater or smaller if we sent the UDP packets the same route using the network code in SunOS 4.1.4. The same computers were used as in Figure 24. On artigonn the same program as above was running which sent

10 000 UDP packets of size 1 000 bytes. On dumburken, a simple program received the packets from artigonn and sent them on to anxiety. On anxiety the same program sent the packets to explorer where the packets were collected. In experiment number three 10 000 packets were sent during 200 seconds and in experiment number four during 13 seconds. The result is in Table 16 and Table 17.

	# packets received by dumburken	# packets received by anxiety	# packets received by explorer
mean	10 000	9 998.4	9 998.4
std. dev.	0	0.5	0.5
min.	10 000	9 998	9 998
max.	10 000	9 999	9 999

Table 16: Throughput experiment 3

These results are almost identical to the results in experiment one (Table 14), which means that both the Mobile-IP code and the SunOS code does not have any problems with receiving the packets at the slow rate.

	# packets received by the HA	# packets received by the FA	# packets received by the MN
mean	2 333.4	2 288.4	2 285.8
std. dev.	37.1	50.8	52.5
min.	2 297	2 235	2 228
max.	2 396	2 360	2 360

Table 17: Throughput experiment 4

In this experiment we see quite a lot of packages are lost, but not as many as in experiment two (Table 15).

10.4.3 Conclusion

When comparing experiments one and three we see that there is no significant difference between them. However between experiment two and four there is a noticeable difference. SunOS can handle about 300 packets more than the Mobile-IP implementation. The reason for this is that the Home Agent running the Mobile-IP code must do some more processing of the incoming packets due to the Mobile-IP protocol administration (for example check the list of registered Mobile Nodes and encapsulate the packets), but also the fact that the Mobile-IP code runs in user-space and not in the operating system kernel contribute to the lower performance. Incorporation of the Mobile-IP code into the kernel would probably speed things up.

Another way of looking at the numbers is that when we are using the SunOS code about 23% of the 10 000 packets reach their destination, compared to 20% when running the Mobile-IP code. This difference is not that big.

10.5 Summary

The measurements that we have done on the Mobile-IP code have been accomplished by using both our own network management program (mipwatcher)

to get some values, and by modifying the code itself to print other values. The time intervals that we were interested in have been measured by registering the system clock at different points in time.

There are several factors that can have influenced the correctness of our measurements. For instance, when we are doing the ping experiment, the time to send a packet from one computer to another is of course dependent on how much traffic there is on the network at the same time, and what the queues at the routers look like. We tried to minimize the error by doing the same experiment many times and calculating the average, but there will still be differences in the time measurements depending on the time of the day when the experiment is done. This means that there is not much point in looking at the exact values, but more at the relations between the different time intervals.

Another problem is that the computers where we are running the Mobile-IP code are normal workstations, where a lot of different processes are running at the same time. This makes it hard to measure for example the time it takes to encapsulate or decapsulate a packet.

11.0 Conclusions

This master thesis project has included several areas around the not yet standardized Mobile-IP protocol, and has produced a few different kinds of results.

First of all, a Management Information Base (MIB) was written for the Mobile-IP protocol, something which had not been done before. Support for network management (SNMP) and this MIB was implemented using the CMU snmp2.1.2 package and incorporated into Anders Klemets Mobile-IP code. At the end of our thesis work four Internet drafts ([31, 32, 33, 34]) describing a Mobile-IP MIB were published by the Mobile-IP working group of the Internet Engineering Task Force. This official Mobile-IP MIB is based on our initial MIB, and we are mentioned in the acknowledgements.

The next step was to understand the Mobile-IP implementation made by Anders Klemets for the SunOS operating system. Once this was achieved, the work on the two ports of the implementation began. Porting the code from SunOS to Solaris was not that difficult. The only part to change was the low level routines that is used for reading ethernet frames from the network. In the SunOS implementation this is done by accessing something called the Network Interface Tap, but this interface is not supported in the Solaris operating system. Here we had to use another interface called the Data Link Provider Interface (DLPI) which of course is not compatible with the Network Interface Tap.

Porting the code to the MINT was a bit more difficult. Most of the time was spent trying to understand the MINT environment. There were also some initial problems when we first tried to boot a MINT, and later when we wanted to transfer the Mobile-IP program to the MINT. The lack of documentation about the programming environment and the operating system for the MINT has resulted in several documents written by us, which in detail describes what we did and how we did it. This should hopefully help a person that is interested in developing other applications for the MINT in the future. The port of the Mobile-IP code itself resulted in one major and one minor change from the SunOS version. The major change was the same as for the Solaris port; the part of the code that is reading and writing ether frames. This could be done neither as in the SunOS implementation, nor as in the Solaris port. Unfortunately it could not be done by using the Berkeley Packet Filter either, something that we had hoped for. Instead we had to abandon the pure Unix code and use some underlying MachOS system calls. Here we used something called the Mach Packet Filter, which is an extension of the Berkeley Packet Filter. At last we got the MINT implementation to run.

To summarise the porting part we can say that there really was not that much to change between the different versions that we made. When porting the SunOS code to MachOS, we made modifications of a few routines in five of the around 90 files included in the implementation. Almost all important changes was made in just one of the files; the file containing the routines for reading and writing ethernet frames. This should encourage people to port the implementation to other platforms as well.

Another part of our degree project was to test the Mobile-IP implementation and analyse the protocol. The testing of the Mobile-IP code was done by running the Mobile-IP code on workstations in our lab running SunOS or Solaris, and the tests revealed two interesting results. The first was that the handling of packets by the Home Agent and the Foreign Agent was a significant part of the overall delay caused by the Mobile-IP protocol. The second was, that of the time spent by a Mobile Node trying to establish a new connection with its Home Agent almost all

of it was spent getting a connection with the Foreign Agent. We do not know if this problem is due to the specific implementation of the Mobile-IP protocol that we are using, or if it is a fundamental problem with the protocol itself. To answer this question, further tests with other implementations should be made. Unfortunately there is hard to get another implementation to test at the moment, because the vendors that have made their own implementations of the protocol (for example IBM, Sun and Motorola) are not willing to release their code right now.

Finally we would like to thank all the people who has helped us during our degree project, especially Chip Maguire who has been of great support.

/ Fredrik & Fredrik, February 1996

Appendix A The CMU-SNMP package

A.1 Introduction

The CMU-SNMP library is a public domain, no guarantee library of SNMP functions from Carnegie Mellon University. It is available via anonymous ftp and written in the C language. Even though it is limited in its functionality it served our purposes well since all we needed was a tool to create an SNMP agent.

Except the library, the distribution also contains several applications which serve as excellent examples for using the library to create an SNMP manager. These include applications for getting and setting variables, walking through an agent's MIB, and a couple of applications for getting sets of variables. These applications, although somewhat useful in themselves, are very good sources and great starting points for writing your own applications. For further instructions on how to use the library to build a manager, read the document "Using the Carnegie Mellon University (CMU) SNMP Library To Build an SNMP Manager" [22].

The reason for including this appendix in our report is that we found that there was little or no information on how to use the library to generate an SNMP agent. Below, a few pointers and examples will be given to show how to use the package to create an SNMP agent.

A.2 How to obtain the library

The library is available from URL:
`ftp://lancaster.andrew.cmu.edu/pub/snmp-dist/`
Do not forget the README file.

A.3 Writing an agent

Before you write your snmp agent, you first have to define a MIB (Management Information Base). The following text assumes that this is already done. There are plenty of good books on how to write a MIB, for example "The Simple Book" by M. T. Rose [11].

There is mainly one file in the package that is of interest when writing an agent, and that is `agent/snmp_vars.c`. That file contains implementations of the standard components for an agent as given in RFC 1213 [24], which can serve as examples.

A.3.1 The data structures

Lets look at the data structures of interest. First there are several different structures called variableX, where X is an integer defining the length of the object identifier.

```
struct variable2 {
     u_char      magic;         /* passed to function as a hint */
     char        type;          /* type of variable */
     u_short     acl;           /* access control list for variable */
     u_char      *(*findVar)();/* function that finds variable */
     u_char      namelen;       /* length of name below */
     oid         name[2];       /* object identifier of variable */
};
```

The first field called *magic* is a number that can help the programmer to identify the requested variable. Its function will be clear as we go along.

The *type* field is one of the types that a MIB variable can have according to RFC 1212 [23]. Though we found that the type SEQUENCE, which is similar to a struct in C language, can not be used as an instantiated variable.

The *acl* field indicates the access rights of the variable. The different access rights that are allowed are RONLY, RWRITE and NOACCESS.

When the agent receives an snmp request it must have a way of getting or setting the value of the requested variable. This functionality is supplied by the function pointed to by the *findVar* variable. The implementation of findVar functions will be discussed below.

The *name* field is the only field that differ among the different variableX structures. An older version of the CMU package used the structure "variable" defined in `snmp_vars.h`, which reserves space for an object identifier (OID) with 32 sub-ids. When writing this newer version of CMU, the author of `snmp_vars.c` decided to create structures with less OID space, presumably in an effort to reduce the size of the compile-time data structures. Thus the "variable2" structure is identical to the "variable" structure, with the only difference that it has room for just 2 sub-ids. The system group variables will need only 2 sub-ids for instance names, e.g. ".3.0" for *sysUpTime.0*. The tcp group needs 13 sub-ids, since after the tcp group id (1.3.6.1.2.1.6) you can need up to 13 more sub-ids to specify a valid instance. E.g., you need 3 more sub-ids to define the *tcpConnState* object under the tcp group, .13.1.1, and then 10 more sub-ids to specify an instance of this object: A.B.C.D.X.E.F.G.H.Y where A.B.C.D is the local IP address, X is the local port, E.F.G.H is the remote IP address, and Y is the remote port. So, when defining variables, use a variableX structure big enough to hold all the instance sub-ids in a valid name.

All variables with a common OID prefix are put in a variable list. You'll find plenty of them in the `snmp_vars.c` file. For example the variable *at_variable* is a list of three variable2 that all have the prefix "1.3.6.1.2.1.3.1.1". The prefix itself is defined in the *subtree* structure discussed below, while the last part of the oid that identifies a variable is put in the name field.

The second structure of interest is the *subtree* structure. It contains an OID prefix which applies to all variables in the associated variable list. The way the subtree is used is rather self-evident if you look at how it is done in the `snmp_vars.c` file. There is one variable called *subtrees* which is a list where all the subtrees are defined. One thing to notice is that all the OID fields in the subtree list have to be unique and no OID can be a prefix to another OID, e.g. it is not valid to have one OID prefix called "1.2" and another called "1.2.1".

```
struct subtree {
     oid    name[16];             /* objid prefix of subtree */
     u_char namelen;              /* number of sub-ids in name above */
     struct variable *variables;  /* pointer to variables array */
     int    variables_len;        /* number of entries in above array */
     int    variables_width;      /* sizeof each variable entry */
};
```

A.3.2 The functions

There are two kinds of functions that are of interest; *findVar* and *writeVar* functions. For every variable added to a variable list there has to be a function that can correctly retrieve the value of that variable. The findVar field in the variable struct should point to the corresponding function.

As an example of a findVar function the *var_system()* function is shown below, with comments inserted between the C-code lines.

```
u_char *
var_system (vp, name, length, exact, var_len, write_method)
  register struct variable *vp; /* IN - pointer to variable entry that
                                   points here */
  register oid *name;           /* IN/OUT - input name requested,
                                   output name found */
  register int *length;         /* IN/OUT - length of input and output
                                   oid's */
  int exact;                    /* IN - TRUE if an exact match was
                                   requested */
  int *var_len;                 /* OUT - length of variable or 0 if
                                   function returned */
  int (**write_method)();       /* OUT - pointer to writeVar,
                                   otherwise 0 */
{
     extern int writeVersion(), writeSystem();
     oid newname[MAX_NAME_LEN];
     int result;
}
```

There are several input parameters to the function. The *vp* variable is a pointer to the variableX struct whose OID prefix in the subtree list plus the sub-id in the variableX struct matched the requested OID. The different fields in the vp variable is used to identify which variable is requested.

When the function is called, the *name* variable contains the OID that was ask for by the management program. When the function returns, the name variable should hold the OID of the exact variable found. It works like this; suppose the management program asks for the ipAdEntAddr (1.3.6.1.2.1.4.20.1.1) variable. Since this variable is a column in a table it is possible to answer with any one of the rows in that column. When the *var_ipAddrEntry()* function that handles the *ipAdEntAddr* variable has decided which row to return, the name variable will be set to 1.3.6.1.2.1.4.20.1.1.A.B.C.D, where A.B.C.D is an IP-address.

The *length* variable is the length of the name variable given as the number of sub-ids. When the *findVar* function is called, the *length* variable is the length of the requested OID and when it returns it should be the length of the OID found.

Exact is true when the manager requests an exact match between the OID of the requested variable and a variable in the agent. That is, if *exact* is true, the agent should only return an answer if it finds an exact match between a variable it knows of and the requested variable. For example if *name* is .1.3.6.1.2.1.4.2.0 which is the *ipDefaultTTL* variable, *var_ip()* should return the default TTL but if *name* looks like .1.3.6.1.2.1.4.2 it should not. On the other hand if *exact* is false then the *findvar* function should return the value whose OID is closest above the requested one, using the *compare* function. Using the same example as above this means that if *name* is .1.3.6.1.2.1.4.2, *var_ip()* should return the default TTL and set the *name* variable to .1.3.6.1.2.1.4.2.0, since this is the instance whose OID is closest above the requested one (in fact it is the only one). If *name* is .1.3.6.1.2.1.4.2.0 then *var_ip()* should return NULL since there is no instance of *ipDefaultTTL* that has an OID that is larger than the requested OID.

Var_len is set by the function to the length of the return value in bytes.

If the value can be written as well as read by a manager, the *findVar* function must supply a pointer to a *writeVar* function that correctly can write a new value to the variable. More about writeVar functions below.

The return value of the *findVar* function should either be NULL, if the request can not be satisfied, or a char pointer to the value found.

```
bcopy((char *)vp->name, (char *)newname, (int)vp->namelen * sizeof(oid));
newname[8] = 0;
result = compare(name, *length, newname, (int)vp->namelen + 1);
```

This piece of code does the following: First the *vp->name* field is copied to a new variable *newname*. The vp->name variable contains the prefix in the subtree list plus the sub-id in the variableX struct that was a prefix to the requested variable. For example, if the a manager tries to fetch the variable *sysContact*, the *vp->name* field will contain 1.3.6.1.2.1.1 concatenated with .4 for a total of 1.3.6.1.2.1.1.4. After the copy operation, *newname[8]* is given the value 0. Lets see why that is done. The *vp->name* variable will always have a length of 8 (since the prefix in the subtree list is 7 ids long and the sub-ids in the system_variables[] are all one id long), and if an exact match is requested the ninth position in the name variable will be a zero (there are no tables in the system group so there will never be any key values added at the end of the OID). The next line of code compares the name variable with the newname variable. If the name variable is larger, then result = 1. If the newname is larger, result = -1, and if they are equal, result = 0.

```
if ((exact && (result != 0)) || (!exact && (result >= 0)))
     return NULL;
```

Now, if *exact* is true and result is not equal to 0, which means that an exact match was requested but name and newname were not equal, or an exact match was not required but the requested OID is longer than any possible OID of a variable this function can handle, then return NULL.

```
bcopy((char *)newname, (char *)name, ((int)vp->namelen + 1) *
sizeof(oid));
*length = vp->namelen + 1;
*write_method = 0;
*var_len = sizeof(long);/* default length */
```

When this peace of code is reached it means that we will be able to satisfy the request. Firstly *newname*, which is the OID of the variable we will return, is copied to *name*. Then **length* is set to the length of the OID in the *name* field. Finally the **write_method* and the **var_len* variables are set to their respectively default values.

```
switch (vp->magic) {
     case VERSION_DESCR:
                *var_len = strlen(version_descr);
                *write_method = writeVersion;
                return (u_char *)version_descr;
     case VERSION_ID:
                *var_len = sizeof(version_id);
                return (u_char *)version_id;
     case UPTIME:
                (u_long)long_return = sysUpTime();
                return (u_char *)&long_return;
```

```
        case IFNUMBER:
                    long_return = Interface_Scan_Get_Count();
                    return (u_char *) &long_return;
        case SYSCONTACT:
                    *var_len = strlen(sysContact);
                    *write_method = writeSystem;
                    return (u_char *)sysContact;
        case SYSNAME:
                    *var_len = strlen(sysName);
                    *write_method = writeSystem;
                    return (u_char *)sysName;
        case SYSLOCATION:
                    *var_len = strlen(sysLocation);
                    *write_method = writeSystem;
                    return (u_char *)sysLocation;
        case SYSSERVICES:
                    long_return = 72;
                    return (u_char *)&long_return;
        default:
                    ERROR("");
    }
    return NULL;
```

To find the exact variable requested, a switch is made on the *vp->magic* number. The magic number is a user defined number given in the variableX structure for each variable. It is there for convenience. The same information could be extracted from the *name* variable. Lets say the *vp->magic* number equals SYSCONTACT. The following lines of code will then be executed. First **var_len*, which is the length of the returned data, is set to the length of the *sysContact* string, which contains the name of the system contact and how to get in touch with him. Secondly the **write_method* is set to the *writeSystem* function that can handle a change to the *sysContact* string. Lastly the *sysContact* string is returned.

Writing a *writeVar* function for a table is similar, but there are a few differences worth pointing out. The *var_atEntry* function will serve as an example.

```
u_char *
var_atEntry(vp, name, length, exact, var_len, write_method)
  register struct variable *vp; /* IN - pointer to variable entry that
                                   points here */
    register oid *name;         /* IN/OUT - input name requested, output
                                   name found */
    register int *length;       /* IN/OUT - length of input and output
                                   oid's */
    int exact;                  /* IN - TRUE if an exact match was
                                   requested. */
    int *var_len;              /* OUT - length of variable or 0 if function
                                   returned. */
    int (**write_method)();     /* OUT - pointer to function to set
                                   variable, otherwise 0 */
{
    /*
     * object identifier is of form:
     * 1.3.6.1.2.1.3.1.1.variable.interface.1.A.B.C.D, where A.B.C.D is IP
     * address.
```

```
     * Interface is at offset 10,
     * IPADDR starts at offset 12.
     */
    u_char *cp;
    oid *op;
    oid lowest[16];
    oid current[16];
    static char PhysAddr[6], LowPhysAddr[6];
    u_long Addr, LowAddr;
```

The arguments to the function are of course the same as above. The main difference between a function that handles a table and one that only takes care of a variable is the case when the *exact* variable is false. When an exact match is requested it is quite straight forward what to do; if there exists a key value in the table that gives an exact match, return the corresponding variable. But when *exact* if false the algorithm is slightly different. There are several cases to consider.

- The requested OID looks like 1.3.6.1.2.1.3.1.1.variable, that is a variable in the table is asked for but no instance is specified. The function should return the first instance of that variable in the table if there is one, else NULL. The first instance is the variable in the row whose OID is the smallest (compared using *compare*).
- The requested OID looks like 1.3.6.1.2.1.3.1.1.variable.interface.1.A.B.C.D, that is both a row and a column in the table are specified. The function should return the variable whose OID is closest above the requested OID. For example, if the requested OID has an exact match in the table, return the variable whose OID is the next OID, using *compare*. If there is no exact match just return the variable whose OID is closest above the requested one. If there is no next variable, return NULL.

The variables defined are the following:

- *lowest[16]*, contains the OID found that is closest above the requested OID so far
- *current[16],* holds the OID which we are working with and is compared to *lowest* to see if this one is closer to the requested OID or not, while traversing the table.
- *PhysAddr[6]* and *LowPhysAddr[6]* are the current physical address and the physical address whose OID is the lowest so far. Same for *LowAddr.*

```
/* fill in object part of name for current (less sizeof instance part) */
bcopy((char *)vp->name, (char *)current, (int)vp->namelen *sizeof(oid));
```

Copy *vp->name* to *current* to get the first part of our working OID.

```
LowAddr = -1;    /* Don't have one yet */
```

LowAddr is initiated to -1 to indicate that we do not have a lowest address yet. If *LowAddr* still is -1 when we have completed our traversal of the table it means that we got no hit.

```
ARP_Scan_Init();
```

ARP_Scan_Init initializes the table and places us at the first position in the table.

```
for (;;) {
     if (ARP_Scan_Next(&Addr, PhysAddr) == 0) break;
     current[10] = 1;/* IfIndex == 1 (ethernet???) XXX */
     current[11] = 1;
     cp = (u_char *)&Addr;
     op = current + 12;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
```

This is the main loop. First a call is made to the *ARP_Scan_Next* function to get the next entry in the table. Observe that this is not a sorted table so *ARP_Scan_Next* just returns the next entry until the whole table has been returned. Secondly the *current* variable is updated with the interfacenumber plus the address received from the *ARP_Scan_Next* call.

```
     if (exact) {
         if (compare(current, 16, name, *length) == 0) {
                 bcopy((char *)current, (char *)lowest, 16 * sizeof(oid));
                 LowAddr = Addr;
                 bcopy(PhysAddr, LowPhysAddr, sizeof(PhysAddr));
                 break; /* no need to search further */
         }
```

Next a test is made to see if this is an exact request. If it is, *current* and *name* are compared. If they are equal we have found a match and we can quit our search. The *LowAddr* is set to the address found in the *ARP_Scan_Next* call. The same with *LowPhysAddr.* If the *compare* fails, take another trip through the loop and get the next entry in the table.

```
         } else {
             if ((compare(current, 16, name, *length) > 0) && ((LowAddr ==
-1) || (compare(current, 16, lowest, 16) < 0))) {
                 bcopy((char *)current, (char *)lowest, 16 * sizeof(oid));
                 LowAddr = Addr;
                 bcopy(PhysAddr, LowPhysAddr, sizeof(PhysAddr));
             }
```

If exact is false we make a test to see if the *current* OID is larger than the requested OID and that it is smaller than the closest OID found so far. If so, *lowest* is set to *current* and the *LowAddr* and the *LowPhysAddr* are updated accordingly. Then we continue to go through the table to se if we can find an OID even closer to *name.*

```
         }
     }
     if (LowAddr == -1) return(NULL);
```

If this test is true it means that we did not find an exact match or that there is no entry in the table that is above the requested OID.

```
     bcopy((char *)lowest, (char *)name, 16 * sizeof(oid));
     *length = 16;
     *write_method = 0;
     switch(vp->magic) {
```

```
        case ATIFINDEX:
                  *var_len = sizeof long_return;
                  long_return = 1; /* XXX */
                  return (u_char *)&long_return;
        case ATPHYSADDRESS:
                  *var_len = sizeof(LowPhysAddr);
                  return (u_char *)LowPhysAddr;
        case ATNETADDRESS:
                  *var_len = sizeof long_return;
                  long_return = LowAddr;
                  return (u_char *)&long_return;
        default:
                  ERROR("");
     }
     return NULL;
}
```

This part is the same as for the non table case previously described.

If a variable is writeable as well as readable, a *writeVar()* function must be supplied for that variable. When a write request is received by the snmp agent, the *writeVar()* function will be called three times for each varbind in the packet. The first time *action* will have the value RESERVE1. During this pass the type and value of the variable should be checked to see if they are correct, i. e. they have the correct type, length etc. Also, if there are other variables in the MIB that depends on this variable, the new value should be stored in a place where other *writeVar()* functions can reach it. Note though, that during this first call, the new value should not be written to the variable. Next, the *writeVar()* is called with the action variable set to RESERVE2. If a variable depends on other variables, now is the time to check if any of the dependants has stored its new value and retrieve it so it can be used during the commit phase. If no errors have been returned so far from any of the *writeVar()* functions, the third call has the action variable set to COMMIT. The *writeVar* should write the new value to the variable and free any resources it has used during the two previous phases. If an error was detected in either of the two first *writeVar* calls, the action variable will have the value FREE. It gives the *writeVar* a chance deallocate any resources it has used in the RESERVE1 and RESERVE2 phases. Of course, in this case, no changes should be made to the variable.

As an example, the *writeVersion* function is shown.

```
int
writeVersion(action, var_val, var_val_type, var_val_len, statP, name,
name_len)
   int      action;      /*IN - RESERVE1, RESERVE2, COMMIT, or FREE*/
   u_char   *var_val;    /*IN - input or output buffer space*/
   u_char   var_val_type;/*IN - type of input buffer*/
   int      var_val_len; /*IN - input and output buffer len*/
   u_char   *statP;      /*IN - pointer to local statistic*/
   oid      *name;       /*IN - pointer to name requested
   int      name_len;    /*IN - number of sub-ids in the name*/
{
     int bigsize = 1000;
     u_char buf[sizeof(version_descr)], *cp;
     int count, size;
```

There are a total of seven parameters to a *writeVar()* function. The *action* variable is set during each call to the values described above. **var_val* is a char pointer to a buffer containing the new value. *var_val_type* is the type of the new value. It is there to make it possible for the function to check that the new value has a correct type. The type must be one of the types that a MIB variable can have according to RFC 1212 [21]. *var_val_len* is of course the length in bytes of the *var_val* variable. *name* is the OID of the requested variable and *name_len* is its length as the number of sub OID's.

```
if (var_val_type != STRING){
  printf("not string\n");
  return SNMP_ERR_WRONGTYPE;
}
```

Check the type of the new value.

```
if (var_val_len > sizeof(version_descr)-1){
  printf("bad length\n");
  return SNMP_ERR_WRONGLENGTH;
}
```

Check the length of the new value.

```
size = sizeof(buf);
asn_parse_string(var_val, &bigsize, &var_val_type, buf, &size);
for(cp = buf, count = 0; count < size; count++, cp++){
  if (!isprint(*cp)){
    printf("not print %x\n", *cp);
    return SNMP_ERR_WRONGVALUE;
  }
}
```

Check the value of the new value.

```
buf[size] = 0;
if (action == COMMIT){
  strcpy(version_descr, buf);
}
```

If *action* has the value COMMIT, write the new value to the *version_descr* variable.

```
  return SNMP_ERR_NOERROR;
}
```

Appendix B The SNMP code

This appendix includes the code added to the Mobile-IP deamon, the changes made to the internal structures of the Mobile-IP implementation, marked with sidebars, as well as the code written on the snmp Agent side.

B.1 Changes to internal structures

The changes are marked with bars.

B.1.1 struct mobileip_host

```
struct mobileip_host {
  /* The IP address of the MH in question */
  uint32 addr;
  /* The number of currently registered COA's is kept here.
   * It is zero if the host has deregisterred
   */
  int iCOACnt;
  /* The Care-Of Addresses of this host are in the following field.
   * An unused entry is set to INADDR_ANY.
   */
  uint32 coa[MOBILEIP_COA_MAX];
  /* The flags received in the registration */
  uint8 flags[MOBILEIP_COA_MAX];
  /* The registration timeout timer, one for each COA registration */
  generic_timer_t timer[MOBILEIP_COA_MAX];
  /* The ID to expect in the next registration request */
  uint64 nxtid;
  mh_t next;
};
```

B.1.2 struct fa_deencap_entry

```
struct fa_deencap_entry {
  uint32 addr;           /* The address of the Mobile Host */
  uint32 haaddr;         /* The address of the MN's HA */
  uint32 reqTS;          /* The time when the regreq was received */
  uint32 replTS;         /* The time when the regrepl was received */
  generic_iface_t gifp;  /* The interface to forward packets to */
  generic_timer_t timer; /* The registration timeout timer */
  deencap_t next;        /* Pointer to next entry in chain */
};
```

B.1.3 struct fa_saved_regstate

```
struct fa_saved_regstate {
  uint64 id;             /* ID that was used in registration request */
  uint32 mh;             /* IP address of MH that sent request */
  uint32 coa;            /* Care-Of Address in registration request */
  uint32 ha;             /* Home Agent in registration request */
  uint32 source;         /* IP source address of request message */
  uint16 port;           /* UDP source port of request message */
  uint32 reqTS;          /* The time when we got our first regreq */
  generic_iface_t gifp;  /* Interface request was received on */
  generic_timer_t timer; /* Expiry timer for this state entry */
  struct udp_cb *udp_cb; /* UDP control block, (transmit socket) */
  regstate_t next;       /* pointer to next in chain */
```

```
};
```

B.1.4 struct mobileip_agent

```
struct mobileip_agent {
  /* The IP address of the agent in question */
  uint32 addr;
  /* The number of bits in the IP address that are used in forming the
   * network number of the agent.  A zero value means that it is unknown.
   */
  unsigned char prefix_length;
  /* A generic interface pointer to the interface on which we learned
   * about this agent.  Typically the interface on which the the
   * ICMP Router Advertisement for this agent was received.
   */
  generic_iface_t gifp;
  /* The IP address of the home agent. */
  uint32 haaddr;
  /* The time when the first registration request was sent. */
  uint32 reqTS;
  /* The time when the first registration reply was received. */
  uint32 replTS;
  /* The sequence number of the last router advertisement
   * that was received
   */
  uint16 seq;
  /* The Care-Of Address to use with this agent */
  uint32 coa;
  /* The lifetime field in the latest received agent advertisement */
  uint16 lifetime;
  /* A copy of the flags bits in the latest agent advertisement */
  uint16 flags;
  /* The IP address of the preferred router, according to this agent */
  uint32 router;
  /* The registration or registration request retransmit timer */
  generic_timer_t timer;
  agent_t next;
};
```

B.1.5 struct pending_request

```
struct pending_request {
  /* A pointer to the foreign agent that the request was sent to */
  agent_t fa;
  /* A pointer to the home agent that the request was sent to */
  homeagent_t ha;
  /* The Care-Of Address to use with this registration */
  uint32 coa;
  /* The time when the first registration request was sent. */
  uint32 reqTS;
  /* The number of times the registration request has been retransmitted */
  int iRetransmitCnt;
  /* The current time when the latest agent advertisement was received */
  uint32 time_heard;
  /* A copy of the ID that was used in the request */
  uint64 id;
  /* The flags field that was used in the request */
```

```
  unsigned char byFlags;
  pending_req_t next;/* Next pending request in list */
};
```

B.2 Mobile-IP code

B.2.1 snmp_init.c

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include “targetdefs.h”
#include <stdio.h>
#include “statistics.h”

int SNMP_socket;
/*************************************************************
 * This is the socket on which we wait for requests form the*
 * snmpd.                                                    *
 *************************************************************/

boolean_t SNMP_inited = FALSE;
/*************************************************************
 * SNMP_inited will be true when the socket has been opened.*
 *************************************************************/

u_short SNMP_port = 0xffd3;
/*************************************************************
 * snmp_port is the port on which we are waiting for        *
 * requests from the snmpd. It can be changed by            *
 * starting mipd with option -p n, where n is a new port.   *
 *************************************************************/

/*************************************************************
 *                      SNMP_SOCKET_INIT                     *
 *************************************************************/
void
snmp_socket_init()
{
  struct sockaddr_in sin;

  SNMP_inited = TRUE;
  bzero((char *)&sin, sizeof(sin));
  sin.sin_family = AF_INET;
  sin.sin_addr.s_addr = INADDR_ANY;
  sin.sin_port = SNMP_port;

  debug_printf(“Open snmp socket\n”);
  if ((SNMP_socket = socket(PF_INET ,SOCK_DGRAM ,0))<0) {
     perror(“Snmp socket”);
     return;
  }
  if (bind(SNMP_socket, (struct sockaddr *)&sin, sizeof(sin))<0) {
     perror(“Snmp bind”);
     return;
  }
```

```
  debug_printf("Done!\n");

}

/************************************************************
 *                       MIPMIPSTAT_INIT                    *
 ************************************************************/

void
mipstat_init()
{
  bzero((char *)&mipstat,sizeof(mipstat));
}
```

B.2.2 snmp_magic.h

```
/************************************************************
 *                 MOBILE NODE MAGIC NUMBERS                *
 ************************************************************/

#define MNHOMEAGENTLIST 1
#define MNADVADDR 2
#define MNADVSEQNO 3
#define MNADVFLAGS 4
#define MNADVTS 5
#define MNADVCOUNT 6
#define MNERRADDR 7
#define MNERRCODE 8
#define MNERRTS 9
#define MNERRCOUNT 10
#define MNAUTHCOUNT 11
#define MNINVREPLCOUNT 12
#define MNSOLTS 13
#define MNSOLCOUNT 14
#define MNDECAPS 15
#define MNDISCARDS 16


/************************************************************
 *        MOBILE NODE REGISTRATION TABLE MAGIC NUMBERS      *
 ************************************************************/

#define MNREGHA 17
#define MNREGFA 18
#define MNREGREQTS 19
#define MNREGREPLTS 20
#define MNREGFLAGS 21
#define MNREGLIFETIME 22


/************************************************************
 *   MOBILE NODE PENDING REGISTRATION TABLE MAGIC NUMBERS   *
 ************************************************************/
```

```
#define MNPENDREGHA 23
#define MNPENDREGFA 24
#define MNPENDREGREQTS 25
#define MNPENDREGREQS 26
#define MNPENDREGFLAGS 27


/*************************************************************
 *                FOREIGN AGENT MAGIC NUMBERS                *
 *************************************************************/

#define FACOALIST 30
#define FAADVSEQNO 31
#define FAADVFLAGS 32
#define FAADVTS 33
#define FAADVCOUNT 34
#define FASOLADDR 35
#define FASOLTS 36
#define FASOLCOUNT 37
#define FAERRRECADDR 38
#define FAERRRECCODE 39
#define FAERRRECTS 40
#define FAERRRECCOUNT 41
#define FAERRSENTADDR 42
#define FAERRSENTCODE 43
#define FAERRSENTTS 44
#define FAERRSENTCOUNT 45
#define FAAUTHCOUNT 46
#define FAREGREQSREC 47
#define FADECAPS 48
#define FADISCARDS 49

/*************************************************************
 *      FOREIGN AGENT REGISTRATION TABLE MAGIC NUMBERS       *
 *************************************************************/

#define FAREGMN 50
#define FAREGHA 51
#define FAREGREQTS 52
#define FAREGREPLTS 53
#define FAREGLIFETIME 54

/*************************************************************
 *  FOREIGN AGENT PENDING REGISTRATION TABLE MAGIC NUMBERS   *
 *************************************************************/

#define FAPENDREGMN 55
#define FAPENDREGHA 56

/*************************************************************
 *                HOME AGENT MAGIC NUMBERS                   *
 *************************************************************/

#define HAAUTHNODELIST 60
#define HAADVSEQNO 61
```

```
#define HAADVFLAGS 62
#define HAADVTS 63
#define HAADVCOUNT 64
#define HASOLADDR 65
#define HASOLTS 66
#define HASOLCOUNT 67
#define HAERRADDR 68
#define HAERRCODE 69
#define HAERRTS 70
#define HAERRCOUNT 71
#define HAAUTHCOUNT 72
#define HAREGREQSREC 73
#define HAENCAPS 74
#define HABROADSCASTSREC 75
#define HABROADCASTSSENT 76

/*************************************************************
 *           HOME AGENT BINDING TABLE MAGIC NUMBERS          *
 *************************************************************/

#define HABINDINGMN 77
#define HABINDINGCOA 78
#define HABINDINGLIFETIME 79
#define HABINDINGFLAGS 80
/*************************************************************
 *                  MIP TYPE MAGIC NUMBERS                   *
 *************************************************************/

#define MIPTYPE 81
```

B.2.3 statistics.h

```
#include “targetdefs.h”

typedef uint32 TimeStamp;
typedef uint32 IPaddr;

struct MNstat {
  IPaddr    mnAdvAddr;
  uint32    mnAdvSeqNo;
  uint16    mnAdvFlags;
  TimeStamp mnAdvTS;
  uint32    mnAdvCount;
  TimeStamp mnAdvFirst;
  IPaddr    mnErrAddr;
  uint32    mnErrCode;
  TimeStamp mnErrTS;
  uint32    mnErrCount;
  uint32    mnAuthCount;
  uint32    mnInvReplCount;
  TimeStamp mnSolTS;
  uint32    mnSolCount;
  TimeStamp mnSolFirst;
  uint32    mnDecaps;
  uint32    mnDiscards;
};
```

```
struct FAstat {
  uint32    faAdvSeqNo;
  uint16    faAdvFlags;
  TimeStamp faAdvTS;
  uint32    faAdvCount;
  IPaddr    faSolAddr;
  TimeStamp faSolTS;
  uint32    faSolCount;
  IPaddr    faErrRecAddr;
  uint32    faErrRecCode;
  TimeStamp faErrRecTS;
  uint32    faErrRecCount;
  IPaddr    faErrSentAddr;
  uint32    faErrSentCode;
  TimeStamp faErrSentTS;
  uint32    faErrSentCount;
  uint32    faAuthCount;
  uint32    faRegReqsRec;
  uint32    faDecaps;
  uint32    faDiscards;
};


struct HAstat {
  uint32    haAdvSeqNo;
  uint16    haAdvFlags;
  TimeStamp haAdvTS;
  uint32    haAdvCount;
  IPaddr    haSolAddr;
  TimeStamp haSolTS;
  uint32    haSolCount;
  IPaddr    haErrAddr;
  uint32    haErrCode;
  TimeStamp haErrTS;
  uint32    haErrCount;
  uint32    haAuthCount;
  uint32    haRegReqsRec;
  uint32    haEncaps;
  uint32    haBroadcastsRec;
  uint32    haBroadcastsSent;
};

struct mipstatstruct {
  struct MNstat mn;
  struct FAstat fa;
  struct HAstat ha;
#define MN 128
#define FA 64
#define HA 32
  unsigned char miptype;
};

extern struct mipstatstruct mipstat;
```

B.2.4 statistics.c

```
#include “statistics.h”
struct mipstatstruct mipstat;
```

B.2.5 snmp.h

```
#include <sys/types.h>

typedef u_long oid;

void mipstat_init();
void snmp_socket_init();
u_long get_timem();
```

B.2.6 snmp.c

```
#include “statistics.h”
#include “snmp_magic.h”

#include <sys/types.h>
#include <netinet/in.h>
#include “targetdefs.h”
#include “absiface.h”
#include “abstimer.h”
#include “mipiface.h”
#include “mbuf.h”
#include “io.h”
#include “packet.h”
#include “enet.h”
#include “internet.h”
#include “ip.h”
#include “arp.h”
#include “os.h”
#include “icmp.h”
#include “auth.h”
#include “mip.h”
#include “mh.h”
#include “mhagent.h”
#include “agent.h”
#include “rdiscovery.h”
#include “rdrouter.h”
#include “lowmisc.h”
#include “lowether.h”
#include “udp.h”
#include “ha.h”
#include “fa.h”
#include “snmp.h”

#include <stdio.h>
#include <sys/uio.h>
#include <sys/time.h>
#include <netinet/in.h>

int snmpHandleReadReq(struct sockaddr_in *sin, char *bp);
int snmpHandleWriteReq(struct sockaddr_in *sin, char *bp);
void snmpMakeReply(char error, uint32 vlength, void *value, u_char
olength, oid *oidfound, int *size, char *buf);
```

```
void snmpHandleReq();
int snmpSend(struct sockaddr_in *sin, uint32 vlength, void *value, u_char
olength, oid *oidfound);
int snmpERROR(struct sockaddr_in *sin, char error);

extern mh_t MHlist;
extern deencap_t Deencap_list;
extern regstate_t Savedstate;
extern uint32 myIPaddr;
extern homeagent_t HomeAgents;
extern agent_t Registrations;
extern pending_req_t PendingRegistrations;

extern int SNMP_socket;

/*************************************************************
 *                        GET_TIMEM                          *
 *************************************************************/
/*************************************************************
 * This function returns the number of microseconds elapsed *
 * since January 1, 1970 (zero hour). 4 bytes are of cause  *
 * not enougth for this purpose but we don't care since all *
 * we want is a way of measuring the time between two       *
 * events.                                                   *
 *************************************************************/
u_long get_timem()
{
  struct timeval tim;
  gettimeofday(&tim, NULL); /* Ask the OS what the time is */
  return (tim.tv_sec*1000000 + tim.tv_usec);
}

/*************************************************************
 *                           COMPARE                         *
 *************************************************************/
/*************************************************************
 * Compare() compares two OID's. If the first OID is bigger *
 * than the second one, 1 is return. If the first OID is    *
 * smaller than the second one, -1 is returned. If they are *
 * equal, 0 is returned. An OID is bigger either if it is   *
 * longer or if it is lexicographicly larger.               *
 *************************************************************/
int
compare(name1, len1, name2, len2)
    register oid    *name1, *name2;
    register int    len1, len2;
{
    register int    len;

    /* len = minimum of len1 and len2 */
    if (len1 < len2)
       len = len1;
    else
       len = len2;
    /* find first non-matching byte */
```

```
    while(len-- > 0){
       if (*name1 < *name2)
         return -1;
       if (*name2++ < *name1++)
         return 1;
    }
    /* bytes match up to length of shorter string */
    if (len1 < len2)
       return -1;  /* name1 shorter, so it is “less” */
    if (len2 < len1)
       return 1;
    return 0;/* both strings are equal */
}

/*************************************************************
 *                    SNMPMAKEREPLY                          *
 *************************************************************/
/*************************************************************
 * Ones we have decided which packet type and contents we    *
 * shall reply with snmpMakeReply is called to put the       *
 * different components into the right place in the packet. *
 *************************************************************/
void
snmpMakeReply(error ,vlength ,value ,olength , oidfound, size, buf)
char error;     /* IN - The error code */
uint32 vlength; /* IN - Length of value in bytes */
void *value;    /* IN - The value found */
u_char olength; /* IN - The lenght of the OID found */
oid *oidfound;  /* IN - The OID of the variable found, if of interest */
int *size;      /* OUT - The size of the packet */
char *buf;      /* OUT - The packet to be sent */
{
  char *cp;
  int i;

  cp = buf;
  if (error == 0) {
     *size = 3 + vlength + olength; /* error, vlength, value, olength,
oidfound */
     /* Put data into packet */
     cp = buf;
     *cp++ = error; /* The first byte is the error code. */
     *cp++ = vlength; /* The second byte is the length of the value feild.
*/
     for (i=0;i<vlength;i++) /* Put vlength bytes of data in the value
feild. */
       *cp++ = ((char *)value)[i];
     *cp++ = olength; /* Then we add the length of the found OID. */
     if (olength !=0)
       /*
        * If the found OID matters olength is not zero, and we put the OID
        * at the end of the packet.
        */
       for (i=0;i<olength / 4;i++)
                /* Use put32 to get the OID’s in network byte order. */
```

```
                cp = put32(cp, oidfound[i]);

  }
  else {
     /* This is an error packet. It only contains the error code */
     *size = 1;
     /* Put the error code into the packet */
     *cp = error;
  }
  return;
}

/*************************************************************
 *                        SNMPSEND                           *
 *************************************************************/
/*************************************************************
 * If we find a variable to return snmpSend is called.      *
 * First snmpMakeReply is called to create the return packet*
 * and then the packet is sent to the port and address from *
 * which we got the request.                                *
 *************************************************************/
int
snmpSend(sin,vlength, value, olength, oidfound)
struct sockaddr_in *sin; /* IN - The address and port we are to send the
reply to. */
uint32 vlength;          /* IN - Length of value in bytes */
void *value;             /* IN - The value found */
u_char olength;          /* IN - The lenght of the OID found */
oid *oidfound;           /* IN - The OID of the variable returned */
{
  int size;
  char buf[200];

  /* Fill the return buffer */
  snmpMakeReply(0, vlength, value, olength, oidfound, &size, buf);
  /* Send the reply */
  if (sendto(SNMP_socket,buf,size,0,(struct sockaddr *)sin, sizeof(struct
sockaddr_in))<=0) {
     fprintf(stderr,"Failed to write to SNMP_socket.\n");
     return -1;
  }
  return 0;
}


/*************************************************************
 *                        SNMPERROR                          *
 *************************************************************/
/*************************************************************
 * If we do not find a value to return or some other error  *
 * occurs, snmpERROR sends an error message to the snmpd.   *
 *************************************************************/
int
snmpERROR(sin, error)
```

```
struct sockaddr_in *sin; /* IN - The address and port we are to send the
error to. */
char error;              /* IN - The error code. */
{
  int size;
  char buf[200];

  /* Fill the return packet */
  snmpMakeReply(error, 0, NULL, 0, NULL, &size, buf);
  /* Send the error*/
  if (sendto(SNMP_socket,buf,size,0,(struct sockaddr *)sin, sizeof(struct
sockaddr_in))<=0) {
     fprintf(stderr,"Failed to write to SNMP_socket.\n");
     return -1;
  }
  return 0;
}

/*************************************************************
 *                     MNHOMEAGENTLIST                       *
 *************************************************************/
int
mnHomeAgentList(sin, exact, current, rlength, request)
struct sockaddr_in *sin; /* IN - The address and port we are to send the
reply to. */
u_char exact;            /* IN - Is this an exact request or not? */
oid *current;            /* IN - The hint from snmpd. */
u_char rlength;          /* IN - The length of the request in OIDs. */
oid *request;            /* IN - The requested OID. */
{
  homeagent_t ha;
  homeagent_t lowestha;
  oid lowest[16];
  boolean_t found;
  u_char *cp;
  oid *op;

  found = FALSE;
  ha = HomeAgents;

  while (ha != (homeagent_t) NULL) {
     cp = (u_char *)&ha->addr;
     op = current + 12;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
     /*
      * Uppdate the current variable vith the found ha address.
      */

     if (exact) {
       if (compare(current, 16, request, rlength) == 0) {
                /* We got a hit */
                lowestha = ha;
```

```
                bcopy((char *)current, (char *)lowest, 16 * sizeof(oid));
                found = TRUE;
                /* We found a hit and do not need to search any more */
                break;
          }
        } else {
          if ((compare(current, 16, request, rlength)>0) &&
                     ((!found) || (compare(current, 16, lowest, 16)<0))){
                /*
                 * We have found an entry in the table whose OID is larger
                 * than the requested OID and is eigther the first entry
                 * that is larger or it is smaller than any other OID that
                 * is larger than the requested OID found so far.
                 */
                lowestha = ha;
                bcopy((char *)current, (char *)lowest, 16 * sizeof(oid));
                found = TRUE;
          }
        }
        ha=ha->next;
    }

    if (!found) {
        snmpERROR(sin, -1);
        return -1;
    }
    snmpSend(sin, sizeof(lowestha->addr), (void *)&lowestha->addr, 16 *
sizeof(oid), lowest);
    return 0;
}

/*************************************************************
 *                        MNREGTABLE                         *
 *************************************************************/
int
mnRegTable(sin, magic, exact, current, rlength, request)
struct sockaddr_in *sin; /* IN - The address and port we are to send the
reply to. */
u_char magic;            /* IN - The requested variable. */
u_char exact;            /* IN - Is this an exact request or not? */
oid *current;            /* IN - The hint from snmpd. */
u_char rlength;          /* IN - The length of the request in OIDs. */
oid *request;            /* IN - The requested OID. */
{
    agent_t reg;
    agent_t lowestreg;
    oid lowest[20];
    boolean_t found;
    u_char *cp;
    oid *op;
    uint32 lowestIP;
    int timeleft;

    found = FALSE;
    reg = Registrations;
```

```
  while (reg != (agent_t) NULL) {
     op = current + 12;
     cp = (u_char *)&reg->haaddr;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
     cp = (u_char *)&reg->addr;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
     /*
      * Uppdate the current variable vith the found agent address
      * and the home agent address.
      */

     if (exact) {
       if (compare(current, 20, request, rlength) == 0) {
                 /* We got a hit */
                 lowestreg = reg;
                 bcopy((char *)current, (char *)lowest, 20 * sizeof(oid));
                 found = TRUE;
                 /* We found a hit and do not need to search any more */
                 break;
       }
     } else {
       if ((compare(current, 20, request, rlength)>0) &&
                   ((!found) || (compare(current, 20, lowest, 20)<0))){
                 /*
                  * We have found an entry in the table whose OID is larger
                  * than the requested OID and is eigther the first entry
                  * that is larger or it is smaller than any other OID that
                  * is larger than the requested OID found so far.
                  */
                 lowestreg = reg;
                 bcopy((char *)current, (char *)lowest, 20 * sizeof(oid));
                 found = TRUE;
       }
     }
     reg=reg->next;
  }
  if (!found) {
     snmpERROR(sin, -1);
     return -1;
  }
  switch (magic) {
  case MNREGHA:
     snmpSend(sin, sizeof(lowestreg->haaddr), (void *)&lowestreg->haaddr,
20 * sizeof(oid), lowest);
     return 0;
  case MNREGFA:
     snmpSend(sin, sizeof(lowestreg->addr), (void *)&lowestreg->addr, 20 *
sizeof(oid), lowest);
```

```
     return 0;
  case MNREGREQTS:
     snmpSend(sin, sizeof(lowestreg->reqTS), (void *)&lowestreg->reqTS, 20
* sizeof(oid), lowest);
     return 0;
  case MNREGREPLTS:
     snmpSend(sin, sizeof(lowestreg->replTS), (void *)&lowestreg->replTS,
20 * sizeof(oid), lowest);
     return 0;
  case MNREGFLAGS:
     snmpSend(sin, sizeof(lowestreg->flags), (void *)&lowestreg->flags, 20
* sizeof(oid), lowest);
     return 0;
  case MNREGLIFETIME:
     TIMER_DURATION(lowestreg->timer,timeleft)
       snmpSend(sin, sizeof(timeleft),
                           (void *)&timeleft, 20 * sizeof(oid), lowest);
     return 0;
  }
}

/*************************************************************
 *                      MNPENDREGTABLE                       *
 *************************************************************/
int
mnPendRegTable(sin, magic, exact, current, rlength, request)
struct sockaddr_in *sin; /* IN - The address and port we are to send the
reply to. */
u_char magic;            /* IN - The requested variable. */
u_char exact;            /* IN - Is this an exact request or not? */
oid *current;            /* IN - The hint from snmpd. */
u_char rlength;          /* IN - The length of the request in OIDs. */
oid *request;            /* IN - The requested OID. */
{
  pending_req_t preg;
  pending_req_t lowestpreg;
  oid lowest[20];
  boolean_t found;
  u_char *cp;
  oid *op;
  int valid;
  uint32 lowestIP;
  int timeleft;

  found = FALSE;
  preg = PendingRegistrations;
  while (preg != (pending_req_t) NULL) {
     cp = (u_char *)&preg->ha->addr;
     op = current + 12;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
     cp = (u_char *)&preg->fa->addr;
     *op++ = *cp++;
```

```
      *op++ = *cp++;
      *op++ = *cp++;
      *op++ = *cp++;
      /*
       * Uppdate the current variable vith the found fa address
       * and the ha address.
       */

      if (exact) {
        if (compare(current, 20, request, rlength) == 0) {
                  /* We got a hit */
                  lowestpreg = preg;
                  bcopy((char *)current, (char *)lowest, 20 * sizeof(oid));
                  found = TRUE;
                  goto end_of_search6;
                  /* We found a hit and do not need to search any more */
                  /*
                   * I can not believe it! After years of programing
experience
                   * and a Masters in computer sience I still am useing
goto!
                   * What will become of me .....
                   */
        }
      } else {
        if ((compare(current, 20, request, rlength)>0) &&
                    ((!found) || (compare(current, 20, lowest, 20)<0))){
                  /*
                   * We have found an entry in the table whose OID is larger
                   * than the requested OID and is eigther the first entry
                   * that is larger or it is smaller than any other OID that
                   * is larger than the requested OID found so far.
                   */
                  lowestpreg = preg;
                  bcopy((char *)current, (char *)lowest, 20 * sizeof(oid));
                  found = TRUE;
        }
      }
      preg=preg->next;
    }
end_of_search6:
    if (!found) {
       snmpERROR(sin, -1);
       return -1;
    }
    switch (magic) {
    case MNPENDREGHA:
       snmpSend(sin, sizeof(lowestpreg->ha->addr), (void *)&lowestpreg->ha-
>addr, 20 * sizeof(oid), lowest);
       return 0;
    case MNPENDREGFA:
       snmpSend(sin, sizeof(lowestpreg->fa->addr), (void *)&lowestpreg->fa-
>addr, 20 * sizeof(oid), lowest);
       return 0;
    case MNPENDREGREQTS:
```

```
     snmpSend(sin, sizeof(lowestpreg->reqTS), (void *)&lowestpreg->reqTS,
20 * sizeof(oid), lowest);
     return 0;
  case MNPENDREGREQS:
     snmpSend(sin, sizeof(lowestpreg->iRetransmitCnt), (void
*)&lowestpreg->iRetransmitCnt,
                        20 * sizeof(oid), lowest);
     return 0;
  case MNPENDREGFLAGS:
     snmpSend(sin, sizeof(lowestpreg->byFlags), (void *)&lowestpreg-
>byFlags, 20 * sizeof(oid), lowest);
     return 0;
  }
}

/*************************************************************
 *                         FACOALIST                         *
 *************************************************************/
int
faCOAList(sin, exact, current, rlength, request)
struct sockaddr_in *sin; /* IN - The address and port we are to send the
reply to. */
u_char exact;            /* IN - Is this an exact request or not? */
oid *current;            /* IN - The hint from snmpd. */
u_char rlength;          /* IN - The length of the request in OIDs. */
oid *request;            /* IN - The requested OID. */
{
  uint32 lowestcoa;
  oid lowest[16];
  boolean_t found;
  u_char *cp;
  oid *op;

  found = FALSE;

  cp = (u_char *)&myIPaddr;
  op = current + 12;
  *op++ = *cp++;
  *op++ = *cp++;
  *op++ = *cp++;
  *op++ = *cp++;
  /*
   * Uppdate the current variable vith the found mh address
   */

  if (exact) {
     if (compare(current, 16, request, rlength) == 0) {
       /* We got a hit */
       lowestcoa = myIPaddr;
       bcopy((char *)current, (char *)lowest, 16 * sizeof(oid));
       found = TRUE;
     }
  } else {
     if (compare(current, 16, request, rlength)>0) {
       /*
```

```
         * Our COA is larger than the requested one.
         */
        lowestcoa = myIPaddr;
        bcopy((char *)current, (char *)lowest, 16 * sizeof(oid));
        found = TRUE;
      }
    }

    if (!found) {
      snmpERROR(sin, -1);
      return -1;
    }
    snmpSend(sin, sizeof(lowestcoa), (void *)&lowestcoa, 16 * sizeof(oid),
lowest);
    return 0;
}

/*************************************************************
 *                        FAREGTABLE                         *
 *************************************************************/
int
faRegTalbe(sin, magic, exact, current, rlength, request)
struct sockaddr_in *sin; /* IN - The address and port we are to send the
reply to. */
u_char magic;            /* IN - The requested variable. */
u_char exact;            /* IN - Is this an exact request or not? */
oid *current;            /* IN - The hint from snmpd. */
u_char rlength;          /* IN - The length of the request in OIDs. */
oid *request;            /* IN - The requested OID. */
{
    deencap_t dl;
    deencap_t lowestdl;
    oid lowest[20];
    boolean_t found;
    int j;
    u_char *cp;
    oid *op;
    int timeleft;

    found = FALSE;
    dl = Deencap_list;

    while (dl != (deencap_t) NULL) {
      cp = (u_char *)&dl->addr;
      op = current + 12;
      *op++ = *cp++;
      *op++ = *cp++;
      *op++ = *cp++;
      *op++ = *cp++;
      cp = (u_char *)&dl->haaddr;
      *op++ = *cp++;
      *op++ = *cp++;
      *op++ = *cp++;
      *op++ = *cp++;
      /*
```

```
       * Uppdate the current variable vith the found ha and
       *  mn addresses.
       */

      if (exact) {
        if (compare(current, 20, request, rlength) == 0) {
                  /* We got a hit */
                  lowestdl = dl;
                  bcopy((char *)current, (char *)lowest, 20 * sizeof(oid));
                  found = TRUE;
                  /* We found a hit and do not need to search any more */
                  break;
        }
      } else {
        if ((compare(current, 20, request, rlength)>0) &&
                    ((!found) || (compare(current, 20, lowest, 20)<0))){
                  /*
                   * We have found an entry in the table whose OID is larger
                   * than the requested OID and is eigther the first entry
                   * that is larger or it is smaller than any other OID that
                   * is larger than the requested OID found so far.
                   */
                  lowestdl = dl;
                  bcopy((char *)current, (char *)lowest, 20 * sizeof(oid));
                  found = TRUE;
        }
      }
      dl=dl->next;
    }
    if (!found) {
       snmpERROR(sin, -1);
       return -1;
    }
    switch (magic) {
    case FAREGMN:
       snmpSend(sin, sizeof(lowestdl->addr), (void *)&lowestdl->addr, 20 *
sizeof(oid), lowest);
       return 0;
    case FAREGHA:
       snmpSend(sin, sizeof(lowestdl->haaddr), (void *)&lowestdl->haaddr, 20
* sizeof(oid), lowest);
       return 0;
    case FAREGREQTS:
       snmpSend(sin, sizeof(lowestdl->reqTS), (void *)&lowestdl->reqTS, 20 *
sizeof(oid), lowest);
       return 0;
    case FAREGREPLTS:
       snmpSend(sin, sizeof(lowestdl->replTS), (void *)&lowestdl->replTS, 20
* sizeof(oid), lowest);
       return 0;
    case FAREGLIFETIME:
       TIMER_DURATION(lowestdl->timer,timeleft)
         snmpSend(sin, sizeof(timeleft),
                              (void *)&timeleft, 20 * sizeof(oid), lowest);
       return 0;
```

```
  }
}

/*************************************************************
 *                      FAPENDREGTABLE                       *
 *************************************************************/
int
faRegPendTalbe(sin, magic, exact, current, rlength, request)
struct sockaddr_in *sin; /* IN - The address and port we are to send the
reply to. */
u_char magic;            /* IN - The requested variable. */
u_char exact;            /* IN - Is this an exact request or not? */
oid *current;            /* IN - The hint from snmpd. */
u_char rlength;          /* IN - The length of the request in OIDs. */
oid *request;            /* IN - The requested OID. */
{
  regstate_t state;
  regstate_t loweststate;
  oid lowest[20];
  boolean_t found;
  int j;
  u_char *cp;
  oid *op;
  int timeleft;

  found = FALSE;
  state = Savedstate;

  while (state != (regstate_t) NULL) {
     cp = (u_char *)&state->mh;
     op = current + 12;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
     cp = (u_char *)&state->ha;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
     /*
      * Uppdate the current variable vith the found ha and
      *  mn addresses.
      */

     if (exact) {
       if (compare(current, 20, request, rlength) == 0) {
                /* We got a hit */
                loweststate = state;
                bcopy((char *)current, (char *)lowest, 20 * sizeof(oid));
                found = TRUE;
                /* We found a hit and do not need to search any more */
                break;
       }
     } else {
```

```
        if ((compare(current, 20, request, rlength)>0) &&
                    ((!found) || (compare(current, 20, lowest, 20)<0))){
                  /*
                   * We have found an entry in the table whose OID is larger
                   * than the requested OID and is eigther the first entry
                   * that is larger or it is smaller than any other OID that
                   * is larger than the requested OID found so far.
                   */
                  loweststate = state;
                  bcopy((char *)current, (char *)lowest, 20 * sizeof(oid));
                  found = TRUE;
        }
      }
      state=state->next;
    }
    if (!found) {
      snmpERROR(sin, -1);
      return -1;
    }
    switch (magic) {
    case FAPENDREGMN:
      snmpSend(sin, sizeof(loweststate->mh), (void *)&loweststate->mh, 20 *
sizeof(oid), lowest);
      return 0;
    case FAPENDREGHA:
      snmpSend(sin, sizeof(loweststate->ha), (void *)&loweststate->ha, 20 *
sizeof(oid), lowest);
      return 0;
    }
}

/*************************************************************
 *                      HABINDINGTABLE                       *
 *************************************************************/
int
haBindingTable(sin, magic, exact, current, rlength, request)
struct sockaddr_in *sin; /* IN - The address and port we are to send the
reply to. */
u_char magic;            /* IN - The requested variable. */
u_char exact;            /* IN - Is this an exact request or not? */
oid *current;            /* IN - The hint from snmpd. */
u_char rlength;          /* IN - The length of the request in OIDs. */
oid *request;            /* IN - The requested OID. */
{
  mh_t mh;
  mh_t lowestmh;
  oid lowest[20];
  int coaddrindex;
  boolean_t found;
  int j;
  u_char *cp;
  oid *op;
  int timeleft;

  found = FALSE;
```

```
  mh = MHlist;

  while (mh != (mh_t) NULL) {
     for (j=0; j<MOBILEIP_COA_MAX; j++) {
       if (mh->coa[j] != INADDR_ANY) {
                 cp = (u_char *)&mh->addr;
                 op = current + 12;
                 *op++ = *cp++;
                 *op++ = *cp++;
                 *op++ = *cp++;
                 *op++ = *cp++;
                 cp = (u_char *)&mh->coa[j];
                 *op++ = *cp++;
                 *op++ = *cp++;
                 *op++ = *cp++;
                 *op++ = *cp++;
                 /*
                  * Uppdate the current variable vith the found mh address
and
                  * the care of address.
                  */

                 if (exact) {
                   if (compare(current, 20, request, rlength) == 0) {
                        /* We got a hit */
                        lowestmh = mh;
                        coaddrindex = j;
                        bcopy((char *)current, (char *)lowest, 20 *
sizeof(oid));
                        found = TRUE;
                        goto end_of_search;
                        /* We found a hit and do not need to search any
more */
                        /*
                         * I can not believe it! After years of programing
experience
                         * and a Masters in computer sience I still am
useing goto!
                         * What will become of me .....
                         */
                   }
                 } else {
                   if ((compare(current, 20, request, rlength)>0) &&
                          ((!found) || (compare(current, 20, lowest,
20)<0))){
                        /*
                         * We have found an entry in the table whose OID is
larger
                         * than the requested OID and is eigther the first
entry
                         * that is larger or it is smaller than any other
OID that
                         * is larger than the requested OID found so far.
                         */
                        lowestmh = mh;
```

```
                        coaddrindex =j;
                        bcopy((char *)current, (char *)lowest, 20 *
sizeof(oid));
                        found = TRUE;
                    }
                  }
        }
      }
      mh=mh->next;
    }
end_of_search:
    if (!found) {
      snmpERROR(sin, -1);
      return -1;
    }
    switch (magic) {
    case HABINDINGMN:
      snmpSend(sin, sizeof(lowestmh->addr), (void *)&lowestmh->addr, 20 *
sizeof(oid), lowest);
      return 0;
    case HABINDINGCOA:
      snmpSend(sin, sizeof(lowestmh->coa[coaddrindex]), (void *)&lowestmh-
>coa[coaddrindex],
                        20 * sizeof(oid), lowest);
      return 0;
    case HABINDINGLIFETIME:
      TIMER_DURATION(lowestmh->timer[coaddrindex],timeleft);
      snmpSend(sin, sizeof(timeleft),
                        (void *)&timeleft, 20 * sizeof(oid), lowest);
      return 0;
    case HABINDINGFLAGS:
      snmpSend(sin, sizeof(lowestmh->flags[coaddrindex]), (void
*)&lowestmh->flags[coaddrindex], 20 * sizeof(oid), lowest);
      return 0;
    }
}

/*************************************************************
 *                      HAAUTHNODELIST                       *
 *************************************************************/
int
haAuthNodeList(sin, magic, exact, current, rlength, request)
struct sockaddr_in *sin; /* IN - The address and port we are to send the
reply to. */
u_char magic;            /* IN - The requested variable. */
u_char exact;            /* IN - Is this an exact request or not? */
oid *current;            /* IN - The hint from snmpd. */
u_char rlength;          /* IN - The length of the request in OIDs. */
oid *request;            /* IN - The requested OID. */
{
  mh_t mh;
  mh_t lowestmh;
  oid lowest[20];
  boolean_t found;
  u_char *cp;
```

```
  oid *op;

  found = FALSE;
  mh = MHlist;

  while (mh != (mh_t) NULL) {
     cp = (u_char *)&mh->addr;
     op = current + 12;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
     *op++ = *cp++;
     /*
      * Uppdate the current variable vith the found mh address.
      */

     if (exact) {
       if (compare(current, 16, request, rlength) == 0) {
                /* We got a hit */
                lowestmh = mh;
                bcopy((char *)current, (char *)lowest, 16 * sizeof(oid));
                found = TRUE;
                /* We found a hit and do not need to search any more */
                break;
       }
     } else {
       if ((compare(current, 16, request, rlength)>0) &&
                   ((!found) || (compare(current, 16, lowest, 16)<0))){
                /*
                 * We have found an entry in the table whose OID is larger
                 * than the requested OID and is eigther the first entry
                 * that is larger or it is smaller than any other OID that
                 * is larger than the requested OID found so far.
                 */
                lowestmh = mh;
                bcopy((char *)current, (char *)lowest, 16 * sizeof(oid));
                found = TRUE;
       }
     }
     mh=mh->next;
  }
  if (!found) {
     snmpERROR(sin, -1);
     return -1;
  }
  snmpSend(sin, sizeof(lowestmh->addr), (void *)&lowestmh->addr, 16 *
sizeof(oid), lowest);
  return 0;
}

/*************************************************************
 *                     SNMPHANDLEREQ                         *
 *************************************************************/
/*************************************************************
 * When we receive something on the SNMP_socket              *
```

```
 * snmpHandleReq is called to take care of the incoming     *
 * request. First it reads the packet from the SNMP_socket. *
 * Then it determines whether this is a read or a write     *
 * request and calls the corresponding routine.             *
 ************************************************************/
void
snmpHandleReq()
{
  u_char rd;
  char buf[200];
  struct sockaddr_in sin;
  int sinlen = sizeof(sin);

  /*
   * Read the request from SNMP_socket. The address and the port of the
   * sender is put in the sin struct.
   */
  if (recvfrom(SNMP_socket,buf,sizeof(buf),0,&sin,&sinlen)<0) {
     fprintf(stderr,"Failed to read from SNMP_socket.\n");
     return;
  }

  rd = buf[0];
  if (rd == 1) /* Read request */
     snmpHandleReadReq(&sin,&buf[1]);
  else if (rd == 2) /* Write request */
     snmpHandleWriteReq(&sin,&buf[1]);
  else {
     snmpERROR(&sin, -1);
     return;
  }
}

/************************************************************
 *                  SNMPHANDLEWRITEREQ                      *
 ************************************************************/
/************************************************************
 * So far no write request are handled.                     *
 ************************************************************/
int
snmpHandleWriteReq(sin, bp)
struct sockaddr_in *sin;
char *bp;
{
}
/************************************************************
 *                  SNMPHANDLEREADREQ                       *
 ************************************************************/
/************************************************************
 * We have received a read request. First, extract the magic*
 * value (the request) and the exact value from the packet. *
 * If the OID supplied by the vp->name variable, in the
 * snmpd, is contained in the packet then clength is larger *
 * than zero and the OID is put in current. Likewise if the *
 * requested OID is in the packet then rlength is larger    *
```

```
 * than zero and the OID is put in request.
 ************************************************************/
int
snmpHandleReadReq(sin, bp)
struct sockaddr_in *sin; /* IN - The address and port we are to send the
reply to. */
char *bp;               /* IN - A pointer to the request packet. */
{

  int i,j;
  u_char magic;
  u_char exact;
  u_char clength;
  oid current[20];
  u_char rlength;
  oid request[40]; /* Lets hope that no snmp manager sends a larger
request. */
  oid test; /* Used to convert oid from network byte order to host byte
order */
  char *testp;

  magic = *bp++; /* Get magic from the request. */
  exact = *bp++; /* Get exact from the request. */
  if ((clength = *bp++) != 0) { /* If clength != 0, get the current hint.
*/
     clength /= sizeof(oid); /* Convert clength from bytes to the number of
oids */
     for (i=0; i<clength; i++) {
       testp = (char *)&test;
       for (j=0; j<4; j++)
                 *testp++ = *bp++;
       current[i] = ntohl(test);
     }
     if ((rlength = *bp++) != 0) { /* If rlength != 0, get the requested
OID. */
       rlength /= sizeof(oid); /* Convert rlength from bytes to the number
of oids */
       for (i=0; i<rlength; i++) {
                 testp = (char *)&test;
                 for (j=0; j<4; j++)
                   *testp++ = *bp++;
                 request[i] = ntohl(test);
       }
     }
  }
  /* This is a horrid case grid. */
  switch ((int)magic) {
/************************************************************
 *                        MOBILE NODE                       *
 ************************************************************/
  case MNHOMEAGENTLIST:
     return mnHomeAgentList(sin, exact, current, rlength, request);
     break;
  case MNREGHA:
  case MNREGFA:
```

```
  case MNREGREQTS:
  case MNREGREPLTS:
  case MNREGFLAGS:
  case MNREGLIFETIME:
     return mnRegTable(sin, magic, exact, current, rlength, request);
     break;
  case MNPENDREGHA:
  case MNPENDREGFA:
  case MNPENDREGREQTS:
  case MNPENDREGREQS:
  case MNPENDREGFLAGS:
     return mnPendRegTable(sin, magic, exact, current, rlength, request);
     break;
  case MNADVADDR:
     snmpSend(sin, sizeof(mipstat.mn.mnAdvAddr), (void
*)&mipstat.mn.mnAdvAddr, 0, NULL);
     break;
  case MNADVSEQNO:
     snmpSend(sin, sizeof(mipstat.mn.mnAdvSeqNo), (void
*)&mipstat.mn.mnAdvSeqNo, 0, NULL);
     break;
  case MNADVFLAGS:
     snmpSend(sin, sizeof(mipstat.mn.mnAdvFlags), (void
*)&mipstat.mn.mnAdvFlags, 0, NULL);
     break;
  case MNADVTS:
     snmpSend(sin, sizeof(mipstat.mn.mnAdvTS), (void
*)&mipstat.mn.mnAdvTS, 0, NULL);
     break;
  case MNADVCOUNT:
     snmpSend(sin, sizeof(mipstat.mn.mnAdvCount), (void
*)&mipstat.mn.mnAdvCount, 0, NULL);
     break;
  case MNERRADDR:
     snmpSend(sin, sizeof(mipstat.mn.mnErrAddr), (void
*)&mipstat.mn.mnErrAddr, 0, NULL);
     break;
  case MNERRCODE:
     snmpSend(sin, sizeof(mipstat.mn.mnErrCode), (void
*)&mipstat.mn.mnErrCode, 0, NULL);
     break;
  case MNERRTS:
     snmpSend(sin, sizeof(mipstat.mn.mnErrTS), (void
*)&mipstat.mn.mnErrTS, 0, NULL);
     break;
  case MNERRCOUNT:
     snmpSend(sin, sizeof(mipstat.mn.mnErrCount), (void
*)&mipstat.mn.mnErrCount, 0, NULL);
     break;
  case MNAUTHCOUNT:
     snmpSend(sin, sizeof(mipstat.mn.mnAuthCount), (void
*)&mipstat.mn.mnAuthCount, 0, NULL);
     break;
  case MNINVREPLCOUNT:
```

```
     snmpSend(sin, sizeof(mipstat.mn.mnInvReplCount), (void
*)&mipstat.mn.mnInvReplCount, 0, NULL);
     break;
  case MNSOLTS:
     snmpSend(sin, sizeof(mipstat.mn.mnSolTS), (void
*)&mipstat.mn.mnSolTS, 0, NULL);
     break;
  case MNSOLCOUNT:
     snmpSend(sin, sizeof(mipstat.mn.mnSolCount), (void
*)&mipstat.mn.mnSolCount, 0, NULL);
     break;
  case MNDECAPS:
     snmpSend(sin, sizeof(mipstat.mn.mnDecaps), (void
*)&mipstat.mn.mnDecaps, 0, NULL);
     break;
  case MNDISCARDS:
     snmpSend(sin, sizeof(mipstat.mn.mnDiscards), (void
*)&mipstat.mn.mnDiscards, 0, NULL);
     break;
/*************************************************************
 *                        FOREIGN AGENT                      *
 *************************************************************/
  case FACOALIST:
     return faCOAList(sin, exact, current, rlength, request);
     break;
  case FAREGMN:
  case FAREGHA:
  case FAREGREQTS:
  case FAREGREPLTS:
  case FAREGLIFETIME:
     return faRegTalbe(sin, magic, exact, current, rlength, request);
     break;
  case FAPENDREGMN:
  case FAPENDREGHA:
     return faRegPendTalbe(sin, magic, exact, current, rlength, request);
     break;
  case FAADVSEQNO:
     snmpSend(sin, sizeof(mipstat.fa.faAdvSeqNo), (void
*)&mipstat.fa.faAdvSeqNo, 0, NULL);
     break;
  case FAADVFLAGS:
     snmpSend(sin, sizeof(mipstat.fa.faAdvFlags), (void
*)&mipstat.fa.faAdvFlags, 0, NULL);
     break;
  case FAADVTS:
     snmpSend(sin, sizeof(mipstat.fa.faAdvTS), (void
*)&mipstat.fa.faAdvTS, 0, NULL);
     break;
  case FAADVCOUNT:
     snmpSend(sin, sizeof(mipstat.fa.faAdvCount), (void
*)&mipstat.fa.faAdvCount, 0, NULL);
     break;
  case FASOLADDR:
     snmpSend(sin, sizeof(mipstat.fa.faSolAddr), (void
*)&mipstat.fa.faSolAddr, 0, NULL);
```

```
      break;
    case FASOLTS:
      snmpSend(sin, sizeof(mipstat.fa.faSolTS), (void
*)&mipstat.fa.faSolTS, 0, NULL);
      break;
    case FASOLCOUNT:
      snmpSend(sin, sizeof(mipstat.fa.faSolCount), (void
*)&mipstat.fa.faSolCount, 0, NULL);
      break;
    case FAERRRECADDR:
      snmpSend(sin, sizeof(mipstat.fa.faErrRecAddr), (void
*)&mipstat.fa.faErrRecAddr, 0, NULL);
      break;
    case FAERRRECCODE:
      snmpSend(sin, sizeof(mipstat.fa.faErrRecCode), (void
*)&mipstat.fa.faErrRecCode, 0, NULL);
      break;
    case FAERRRECTS:
      snmpSend(sin, sizeof(mipstat.fa.faErrRecTS), (void
*)&mipstat.fa.faErrRecTS, 0, NULL);
      break;
    case FAERRRECCOUNT:
      snmpSend(sin, sizeof(mipstat.fa.faErrRecCount), (void
*)&mipstat.fa.faErrRecCount, 0, NULL);
      break;
    case FAERRSENTADDR:
      snmpSend(sin, sizeof(mipstat.fa.faErrSentAddr), (void
*)&mipstat.fa.faErrSentAddr, 0, NULL);
      break;
    case FAERRSENTCODE:
      snmpSend(sin, sizeof(mipstat.fa.faErrSentCode), (void
*)&mipstat.fa.faErrSentCode, 0, NULL);
      break;
    case FAERRSENTTS:
      snmpSend(sin, sizeof(mipstat.fa.faErrSentTS), (void
*)&mipstat.fa.faErrSentTS, 0, NULL);
      break;
    case FAERRSENTCOUNT:
      snmpSend(sin, sizeof(mipstat.fa.faErrSentCount), (void
*)&mipstat.fa.faErrSentCount, 0, NULL);
      break;
    case FAAUTHCOUNT:
      snmpSend(sin, sizeof(mipstat.fa.faAuthCount), (void
*)&mipstat.fa.faAuthCount, 0, NULL);
      break;
    case FAREGREQSREC:
      snmpSend(sin, sizeof(mipstat.fa.faRegReqsRec), (void
*)&mipstat.fa.faRegReqsRec, 0, NULL);
      break;
    case FADECAPS:
      snmpSend(sin, sizeof(mipstat.fa.faDecaps), (void
*)&mipstat.fa.faDecaps, 0, NULL);
      break;
    case FADISCARDS:
```

```
     snmpSend(sin, sizeof(mipstat.fa.faDiscards), (void
*)&mipstat.fa.faDiscards, 0, NULL);
     break;
/*********************************************************
 *                          HOME AGENT                   *
 *********************************************************/
  case HABINDINGMN:
  case HABINDINGCOA:
  case HABINDINGLIFETIME:
  case HABINDINGFLAGS:
     return haBindingTable(sin, magic, exact, current, rlength, request);
     break;
  case HAAUTHNODELIST:
     return haAuthNodeList(sin, magic, exact, current, rlength, request);
     break;
  case HAADVSEQNO:
     snmpSend(sin, sizeof(mipstat.ha.haAdvSeqNo), (void
*)&mipstat.ha.haAdvSeqNo, 0, NULL);
     break;
  case HAADVFLAGS:
     snmpSend(sin, sizeof(mipstat.ha.haAdvFlags), (void
*)&mipstat.ha.haAdvFlags, 0, NULL);
     break;
  case HAADVTS:
     snmpSend(sin, sizeof(mipstat.ha.haAdvTS), (void
*)&mipstat.ha.haAdvTS, 0, NULL);
     break;
  case HAADVCOUNT:
     snmpSend(sin, sizeof(mipstat.ha.haAdvCount), (void
*)&mipstat.ha.haAdvCount, 0, NULL);
     break;
  case HASOLADDR:
     snmpSend(sin, sizeof(mipstat.ha.haSolAddr), (void
*)&mipstat.ha.haSolAddr, 0, NULL);
     break;
  case HASOLTS:
     snmpSend(sin, sizeof(mipstat.ha.haSolTS), (void
*)&mipstat.ha.haSolTS, 0, NULL);
     break;
  case HASOLCOUNT:
     snmpSend(sin, sizeof(mipstat.ha.haSolCount), (void
*)&mipstat.ha.haSolCount, 0, NULL);
     break;
  case HAERRADDR:
     snmpSend(sin, sizeof(mipstat.ha.haErrAddr), (void
*)&mipstat.ha.haErrAddr, 0, NULL);
     break;
  case HAERRCODE:
     snmpSend(sin, sizeof(mipstat.ha.haErrCode), (void
*)&mipstat.ha.haErrCode, 0, NULL);
     break;
  case HAERRTS:
     snmpSend(sin, sizeof(mipstat.ha.haErrTS), (void
*)&mipstat.ha.haErrTS, 0, NULL);
     break;
```

```
  case HAERRCOUNT:
     snmpSend(sin, sizeof(mipstat.ha.haErrCount), (void
*)&mipstat.ha.haErrCount, 0, NULL);
     break;
  case HAAUTHCOUNT:
     snmpSend(sin, sizeof(mipstat.ha.haAuthCount), (void
*)&mipstat.ha.haAuthCount, 0, NULL);
     break;
  case HAREGREQSREC:
     snmpSend(sin, sizeof(mipstat.ha.haRegReqsRec), (void
*)&mipstat.ha.haRegReqsRec, 0, NULL);
     break;
  case HAENCAPS:
     snmpSend(sin, sizeof(mipstat.ha.haEncaps), (void
*)&mipstat.ha.haEncaps, 0, NULL);
     break;
  case HABROADSCASTSREC:
     snmpSend(sin, sizeof(mipstat.ha.haBroadcastsRec), (void
*)&mipstat.ha.haBroadcastsRec, 0, NULL);
     break;
  case HABROADCASTSSENT:
     snmpSend(sin, sizeof(mipstat.ha.haBroadcastsSent), (void
*)&mipstat.ha.haBroadcastsSent, 0, NULL);
     break;
/*********************************************************
 *                          MIPTYPE                      *
 *********************************************************/
  case MIPTYPE:
     snmpSend(sin, sizeof(mipstat.miptype), (void *)&mipstat.miptype, 0,
NULL);
     break;
  default:
     fprintf(stderr,"Ush\n");
     snmpERROR(sin, -1);
  }
}
```

B.3 SNMP Agent code

Not all of the code in *snmp_vars.c* was included but only the part that is of interest to the Mobile-IP MIB.

B.3.1 snmp_vars.c

```
/*********************************************************
 *                  MOBILE NODE VARIABLE                 *
 *********************************************************/

struct variable20 mn_variables[] = {
  {MNHOMEAGENTLIST, IPADDRESS, RONLY, var_mipEntry, 3, {1, 1, 1}},
  {MNREGHA, IPADDRESS, RONLY, var_mipEntry, 3, {2, 1, 1}},
  {MNREGFA, IPADDRESS, RONLY, var_mipEntry, 3, {2, 1, 2}},
  {MNREGREQTS, INTEGER, RONLY, var_mipEntry, 3, {2, 1, 3}},
  {MNREGREPLTS, INTEGER, RONLY, var_mipEntry, 3, {2, 1, 4}},
  {MNREGFLAGS, BITSTRING, RONLY, var_mipEntry, 3, {2, 1, 5}},
  {MNREGLIFETIME, INTEGER, RONLY, var_mipEntry, 3, {2, 1, 6}},
  {MNPENDREGHA, IPADDRESS, RONLY, var_mipEntry, 3, {3, 1, 1}},
```

```
  {MNPENDREGFA, IPADDRESS, RONLY, var_mipEntry, 3, {3, 1, 2}},
  {MNPENDREGREQTS, INTEGER, RONLY, var_mipEntry, 3, {3, 1, 3}},
  {MNPENDREGREQS, INTEGER, RONLY, var_mipEntry, 3, {3, 1, 4}},
  {MNPENDREGFLAGS, BITSTRING, RONLY, var_mipEntry, 3, {3, 1, 5}},
  {MNADVADDR, IPADDRESS, RONLY, var_mip, 1, {4}},
  {MNADVSEQNO, INTEGER, RONLY, var_mip, 1, {5}},
  {MNADVFLAGS, BITSTRING, RONLY, var_mip, 1, {6}},
  {MNADVTS, INTEGER, RONLY, var_mip, 1, {7}},
  {MNADVCOUNT, COUNTER, RONLY, var_mip, 1, {8}},
  {MNERRADDR, IPADDRESS, RONLY, var_mip, 1, {9}},
  {MNERRCODE, INTEGER, RONLY, var_mip, 1, {10}},
  {MNERRTS, INTEGER, RONLY, var_mip, 1, {11}},
  {MNERRCOUNT, COUNTER, RONLY, var_mip, 1, {12}},
  {MNAUTHCOUNT, COUNTER, RONLY, var_mip, 1, {13}},
  {MNINVREPLCOUNT, COUNTER, RONLY, var_mip, 1, {14}},
  {MNSOLTS, INTEGER, RONLY, var_mip, 1, {15}},
  {MNSOLCOUNT, COUNTER, RONLY, var_mip, 1, {16}},
  {MNDECAPS, COUNTER, RONLY, var_mip, 1, {17}},
  {MNDISCARDS, COUNTER, RONLY, var_mip, 1, {18}}
};

/*********************************************************
 *                FOREIGN AGENT VARIABLE                 *
 *********************************************************/

struct variable20 fa_variables[] = {
  {FACOALIST,IPADDRESS, RONLY, var_mipEntry, 3, {1, 1, 1}},
  {FAREGMN, IPADDRESS, RONLY, var_mipEntry, 3, {2, 1, 1}},
  {FAREGHA, IPADDRESS , RONLY, var_mipEntry, 3, {2, 1, 2}},
  {FAREGREQTS, INTEGER, RONLY, var_mipEntry, 3, {2, 1, 3}},
  {FAREGREPLTS, INTEGER, RONLY, var_mipEntry, 3, {2, 1, 4}},
  {FAREGLIFETIME, INTEGER, RONLY, var_mipEntry, 3, {2, 1, 5}},
  {FAPENDREGMN,  IPADDRESS, RONLY, var_mipEntry, 3, {3, 1, 1}},
  {FAPENDREGHA,  IPADDRESS , RONLY, var_mipEntry, 3, {3, 1, 2}},
  {FAADVSEQNO, INTEGER, RONLY, var_mip, 1, {4}},
  {FAADVFLAGS, BITSTRING, RONLY, var_mip, 1, {5}},
  {FAADVTS, INTEGER, RONLY, var_mip, 1, {6}},
  {FAADVCOUNT, COUNTER, RONLY, var_mip, 1, {7}},
  {FASOLADDR, IPADDRESS, RONLY, var_mip, 1, {8}},
  {FASOLTS, INTEGER, RONLY, var_mip, 1, {9}},
  {FASOLCOUNT, COUNTER, RONLY, var_mip, 1, {10}},
  {FAERRRECADDR, IPADDRESS, RONLY, var_mip, 1, {11}},
  {FAERRRECCODE, INTEGER, RONLY, var_mip, 1, {12}},
  {FAERRRECTS, INTEGER, RONLY, var_mip, 1, {13}},
  {FAERRRECCOUNT, COUNTER, RONLY, var_mip, 1, {14}},
  {FAERRSENTADDR, IPADDRESS, RONLY, var_mip, 1, {15}},
  {FAERRSENTCODE, INTEGER, RONLY, var_mip, 1, {16}},
  {FAERRSENTTS, INTEGER, RONLY, var_mip, 1, {17}},
  {FAERRSENTCOUNT, COUNTER, RONLY, var_mip, 1, {18}},
  {FAAUTHCOUNT, COUNTER, RONLY, var_mip, 1, {19}},
  {FAREGREQSREC, COUNTER, RONLY, var_mip, 1, {20}},
  {FADECAPS, COUNTER, RONLY, var_mip, 1, {21}},
  {FADISCARDS, COUNTER, RONLY, var_mip, 1, {22}}
};
```

```
/*************************************************************
 *                    HOME AGENT VARIABLE                    *
 *************************************************************/

struct variable20 ha_variables[] = {
  {HABINDINGMN, IPADDRESS, RONLY, var_mipEntry, 3, {1, 1, 1}},
  {HABINDINGCOA, IPADDRESS, RONLY, var_mipEntry, 3, {1, 1, 2}},
  {HABINDINGLIFETIME, INTEGER, RONLY, var_mipEntry, 3, {1, 1, 3}},
  {HABINDINGFLAGS, BITSTRING, RONLY, var_mipEntry, 3, {1, 1, 4}},
  {HAAUTHNODELIST, IPADDRESS, RONLY, var_mipEntry, 3, {2, 1, 1}},
  {HAADVSEQNO, INTEGER, RONLY, var_mip, 1, {3}},
  {HAADVFLAGS, BITSTRING, RONLY, var_mip, 1, {4}},
  {HAADVTS, INTEGER, RONLY, var_mip, 1, {5}},
  {HAADVCOUNT, COUNTER, RONLY, var_mip, 1, {6}},
  {HASOLADDR, IPADDRESS, RONLY, var_mip, 1, {7}},
  {HASOLTS, INTEGER, RONLY, var_mip, 1, {8}},
  {HASOLCOUNT, COUNTER, RONLY, var_mip, 1, {9}},
  {HAERRADDR, IPADDRESS, RONLY, var_mip, 1, {10}},
  {HAERRCODE, INTEGER, RONLY, var_mip, 1, {11}},
  {HAERRTS, INTEGER, RONLY, var_mip, 1, {12}},
  {HAERRCOUNT, COUNTER, RONLY, var_mip, 1, {13}},
  {HAAUTHCOUNT, COUNTER, RONLY, var_mip, 1, {14}},
  {HAREGREQSREC, COUNTER, RONLY, var_mip, 1, {15}},
  {HAENCAPS, COUNTER, RONLY, var_mip, 1, {16}},
  {HABROADSCASTSREC, COUNTER, RONLY, var_mip, 1, {17}},
  {HABROADCASTSSENT, COUNTER, RONLY, var_mip, 1, {18}}
};

/*************************************************************
 *                     MIP TYPE VARIABLE                     *
 *************************************************************/
struct variable11 miptype_variables[] = {
  {MIPTYPE, BITSTRING, RONLY, var_miptype, 0, {1}}

  /*
   * Sorry about the bad code but I hade to set a 0 length
   * feild to be able to construct the subtree in a correct
   * manner
   */
};


struct subtree subtrees[] = {
     {{MOBILENODE}, 9, (struct variable *)mn_variables,
      sizeof(mn_variables)/sizeof(*mn_variables),
      sizeof(*mn_variables)},
     {{FOREIGNAGENT}, 9, (struct variable *)fa_variables,
      sizeof(fa_variables)/sizeof(*fa_variables),
      sizeof(*fa_variables)},
     {{HOMEAGENT}, 9, (struct variable *)ha_variables,
      sizeof(ha_variables)/sizeof(*ha_variables),
      sizeof(*ha_variables)},
     {{MIP_MIB, 4}, 9, (struct variable *)miptype_variables,
      sizeof(miptype_variables)/sizeof(*miptype_variables),
      sizeof(*miptype_variables)}
```

```
};
```

B.3.2 mipmib.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <errno.h>
#include <sys/uio.h>

#include “asn1.h”
#include “mipmib.h”
#include “snmp_impl.h”
#include “snmp_vars.h”

#define TIMEOUT -133
/*************************************************************
 *                      CONNECT_MIPD                         *
 *************************************************************/
int connect_mipd();

/*************************************************************
 * snmp_addr is the IP address to which snmpd should send    *
 * it’s requests to the mipd. The default value is loopback.*
 * It can be changed by running snmpd with the switch        *
 * -ma A.B.C.D.                                              *
 *************************************************************/
u_long snmp_addr = htonl(INADDR_LOOPBACK);

/*************************************************************
 * snmp_port is the port on which the mipd is waiting for a *
 * connection request from us. It can be changed by          *
 * starting snmpd with option -mp n, where n is a new port. *
 *************************************************************/
u_short snmp_port =  htons(0xffd3); /* 65491 */

/*************************************************************
 * The socket to the mipd.                                   *
 *************************************************************/
static int MIPsock = -1;

/*************************************************************
 * The number of seconds between retransmits. The default    *
 * value is 1.                                               *
 *************************************************************/
int TimeOutTime = 1;

/*************************************************************
 * The number of times a request is retransmitted to the     *
 * mipd. The default value is 1. Note that the manager       *
 * might retransmitting it’s requests to us which will make *
 * us send another packet to the mipd.                       *
 *************************************************************/
int NumOfRetrans = 1;
```

```
static u_long longreturn;

/*********************************************************
 *                      LOOKUP FUNCTIONS                 *
 *********************************************************/

/*********************************************************
 *                         VAR_MIP                       *
 *********************************************************/
u_char *var_mip(vp, name, length, exact, var_len, write_method)
    register struct variable *vp;   /* IN - pointer to variable entry that
points here */
    register oid*name;    /* IN/OUT - input name requested, output name
found */
    register int*length;    /* IN/OUT - length of input and output oid's */
    int                 exact;     /* IN - TRUE if an exact match was
requested. */
    int                 *var_len;   /* OUT - length of variable or 0 if
function returned. */
    int                 (**write_method)(); /* OUT - pointer to function
to set variable, otherwise 0 */
{
  oid newname[MAX_NAME_LEN];
  int result;
  int ret,retransmits;
  unsigned char flagsret[3];

  retransmits = 0;

  /*
   * Copy the name from the variable (vp->name) to newname.
   * Set the last id to zero (get the instance).
   * If an exact match is required and not found or if the requested
   * oid id longer than any we no of return NULL.
   * Else copy newname (including the trailing zero) to name,
   * set the right length and switch on the magic number to fetch
   * the correct value for the requested variable
   */

  bcopy((char *)vp->name, (char *)newname, (int)vp->namelen *
sizeof(oid));
  newname[10] = 0;
  result = compare(name, *length, newname, (int)vp->namelen + 1);
  if ((exact && (result != 0)) || (!exact && (result >= 0)))
     return NULL;
  bcopy((char *)newname, (char *)name, ((int)vp->namelen + 1) *
sizeof(oid));
  *length = vp->namelen + 1;
  *write_method = 0; /* There is no write_method() for the mip_vars */

  /*
   * Send an request to the mipd and await a response. If GetResp timesout
   * retransmit NumOfRetrans times and then give up.
   */
```

```
  do {
     if (SendReq(1, vp->magic, exact, 0, NULL, NULL, NULL)<0)
       /* Failed to send a read request to the mipd */
       return NULL;
     switch (vp->magic) {
     case MNADVFLAGS:
     case FAADVFLAGS:
     case HAADVFLAGS:
       if ((ret = GetResp(NULL, &flagsret[1], 0, NULL)) == 0){
                 *var_len = sizeof(flagsret);
                 /* Return the received value */
                 return flagsret;
       }
       else
                 retransmits++;
       break;
     default:
       if ((ret = GetResp(NULL, &longreturn, 0, NULL)) == 0) {
                 *var_len = sizeof(longreturn);
                 longreturn = ntohl(longreturn);
                 /* Return the received value */
                 return (u_char *)&longreturn;
       }
       else
                 retransmits++;
     }
  } while ((retransmits < NumOfRetrans) && (ret == TIMEOUT));
  if (ret < 0)
     /* We didn't get a response from mipd. */
     return NULL;
};

/*************************************************************
 *                        VAR_MIPTYPE                        *
 *************************************************************/
u_char *var_miptype(vp, name, length, exact, var_len, write_method)
    register struct variable *vp;   /* IN - pointer to variable entry that
points here */
    register oid*name;    /* IN/OUT - input name requested, output name
found */
    register int*length;    /* IN/OUT - length of input and output oid's */
    int                 exact;     /* IN - TRUE if an exact match was
requested. */
    int                 *var_len;   /* OUT - length of variable or 0 if
function returned. */
    int                 (**write_method)(); /* OUT - pointer to function
to set variable, otherwise 0 */
{
  oid newname[MAX_NAME_LEN];
  int result;
  u_char typeret[2];
  int ret,retransmits;
  retransmits = 0;
```

```
  bcopy((char *)vp->name, (char *)newname, (int)vp->namelen *
sizeof(oid));
  newname[9] = 0;
  result = compare(name, *length, newname, (int)vp->namelen + 1);
  if ((exact && (result != 0)) || (!exact && (result >= 0)))
     return NULL;
  bcopy((char *)newname, (char *)name, ((int)vp->namelen + 1) *
sizeof(oid));
  *length = vp->namelen + 1;
  *write_method = 0;

  if (SendReq(1, vp->magic, exact, 0, NULL, NULL, NULL)<0)
     /* Failed to send data to mipd. */
     return NULL;
  if (GetResp(NULL, &typeret[1], NULL, NULL)<0)
     /* mipd sent an error */
     return NULL;
  *var_len = sizeof(typeret);
  return typeret;
};

/*************************************************************
 *                      VAR_MIPENTRY                         *
 *************************************************************/
u_char *var_mipEntry(vp, name, length, exact, var_len, write_method)
    register struct variable *vp;   /* IN - pointer to variable entry that
points here */
    register oid*name;    /* IN/OUT - input name requested, output name
found */
    register int*length;    /* IN/OUT - length of input and output oid's */
    int                 exact;     /* IN - TRUE if an exact match was
requested. */
    int                 *var_len;   /* OUT - length of variable or 0 if
function returned. */
    int                 (**write_method)(); /* OUT - pointer to function
to set variable, otherwise 0 */
{
  int ret,retransmits;
  unsigned char flagsret[2];

  retransmits = 0;
  write_method = 0;

  do {
     if (SendReq(1, vp->magic, exact, vp->namelen, vp->name, *length,
name)<0)
       /* Failed to send a read request to the mipd */
       return NULL;
     switch (vp->magic) {
     case MNREGFLAGS:
     case MNPENDREGFLAGS:
     case HABINDINGFLAGS:
       if ((ret = GetResp(NULL, &flagsret[1], length, name)) == 0){
                *var_len = sizeof(flagsret);
                /* Return the received value */
```

```
                return flagsret;
        }
        else
                retransmits++;
        break;
      default:
        if ((ret = GetResp(NULL, &longreturn, length, name)) == 0) {
                *var_len = sizeof(longreturn);
                longreturn = ntohl(longreturn);
                /* Return the received value */
                return (u_char *)&longreturn;
        }
        else
                retransmits++;
      }
    } while ((retransmits < NumOfRetrans) && (ret == TIMEOUT));
    if (ret < 0)
      /* We didn't get a response from mipd. */
      return NULL;
};


/************************************************************
 *                      CONNECT_MIPD                        *
 ************************************************************/
/************************************************************
 * This function is called in snmpd.c and opens a UDP socket*
 * for the communikation with the mipd. The address to the  *
 * mipd is given by snmp_addr, and the post number by       *
 * snmp_port.                                               *
 ************************************************************/
int
connect_mipd()
{
  int test;
  struct sockaddr_in sin;

  fprintf(stderr,"\nOpening mipd socket ...\n");
  if ((MIPsock = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
     fprintf(stderr,"Can't create socket\n");
     return -1;
  }
  bzero((char *)&sin, sizeof(sin));
  sin.sin_addr.s_addr = snmp_addr;
  sin.sin_family = AF_INET;
  sin.sin_port = snmp_port;
  if (connect(MIPsock,(struct sockaddr *)&sin, sizeof(sin))<0) {
     fprintf(stderr,"Couldn't connect to mipd\n");
     return -1;
  }
  fprintf(stderr,"Done!\n");
  return 0;
};

/************************************************************
```

```
 *                          SENDREQ                                *
 ***************************************************************/
/***************************************************************
 * First, build a readpacket containing (in order) the R/W  *
 * value, the magic value, the exact value and the clength  *
 * value. If current != NULL include it next in the packet. *
 * If current is not NULL also put in the rlength and the   *
 * requested OID (request).                                 *
 * When the packet is done, send it to MIPsock.             *
 ***************************************************************/
int
SendReq(rw, magic, exact, clength, current, rlength, request)
u_char rw;      /* Read = 1, Write = 2 */
u_char magic;
u_char exact;
u_char clength; /* Lenght of current */
oid *current;   /* The vp->name value */
u_char rlength; /* Length of the request OID */
oid *request;   /* The requested oid */
{
  char buf[150];
  char *bufp,*cp;
  struct iovec iov[6];
  int i,j,len;
  oid temp;

  bufp = buf;
  bufp[0] = rw;
  bufp[1] = magic;
  bufp[2] = exact;
  bufp[3] = clength * sizeof(oid);
  bufp += 4;
  len = 4;
  if (current != NULL) {
     for (i=0;i<clength;i++) {
       (long)temp = htonl((long)current[i]);
       cp = (char *)&temp;
       for (j=0;j<4;j++) {
                 *bufp++ = *cp++;
       }
     }
     len += (clength * sizeof(oid));
     *bufp++ = rlength * sizeof(oid);
     len++;
     for (i=0;i<rlength;i++) {
       temp = htonl(request[i]);
       cp = (char *)&temp;
       for (j=0;j<4;j++) {
                 *bufp++ = *cp++;
       }
     }
     len += (rlength * sizeof(oid));
  }
  if (write(MIPsock,buf,len)<=0) {
     /* Couldn't write to MIPsock. */
```

```
    return -1;
  }
   /* Successful write */
  return 0;
}

/*************************************************************
 *                          GETRESP                          *
 *************************************************************/
/*************************************************************
 * Wait TimeOutTime seconds for an answer from mipd. If non *
 * is received, return TIMEOUT. If a packet is returned form*
 * mipd, first check the error value. If error is not zero  *
 * the packet will only contain the error value, there fore *
 * just return the error. If on the other hand error is     *
 * zero, get the length feild of the returned value is read.*
 * If the caller of GetResp is interested in the length of  *
 * the return value uppdate *vlength. Then read valuelength *
 * bytes from the packet into *value. Get the found OID     *
 * length. If it is non zero get the OID and  uppdate       *
 * olength and oidfound.                                    *
 *************************************************************/
int
GetResp(vlength, value, olength, oidfound)
u_char *vlength; /* OUT--The lenght of the value feild in bytes */
void *value; /* OUT--The variable requested */
int *olength; /* OUT--The length of the returned oid */
oid *oidfound; /* OUT--The oid returned */
{
  u_char buf[150]; /* This means that value can not be bigger than 64 bytes
*/
  char error;
  u_char valuelength;
  u_char oidlength;
  u_char *bufp;
  int len,i; /* Debug */
  fd_set readfds;
  struct timeval timeout;

  FD_ZERO(&readfds);
  FD_SET(MIPsock,&readfds);
  timeout.tv_sec = TimeOutTime;
  timeout.tv_usec = 0;

  if (select(FD_SETSIZE, &readfds, NULL, NULL, &timeout)>0) {
     if ((len =read(MIPsock, buf, sizeof(buf)))<=0) {
       /* Couldn't read from MIPsock. */
       perror("GetResp");
       return -1;
     }
     bufp = buf;
     if ((error = (int)bufp[0])<0) {
       return (int)error;
     }
     else {
```

```
        valuelength = bufp[1]
        if (vlength != NULL)
                *vlength = valuelength; /* If the length of the return
value is of interest */
        bufp += 2;
        bcopy((char *)bufp, (char *)value, valuelength);

        bufp += valuelength;
        oidlength = bufp[0];
        if (oidlength != 0) {
                *olength = oidlength/4;
                bufp += 1;
                bcopy((char *)bufp, (char *)oidfound, oidlength);
        }
      }
      /* Successful read */
      return 0;
  }
  /* Eighter an error on the socker or a timeout */
  return TIMEOUT;
}
```

Appendix C The Solaris Code

In this appendix the important changes to the Mobile-IP code done for the Solaris port is included.

C.1 dlpi.c

```
#include “targetdefs.h”
#include “absiface.h”
#include “absmem.h”
#include “abstimer.h”
#include “dlpi.h”
#include “mbuf.h”
#include “packet.h”
#include “enet.h”
#include <stdio.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/stropts.h>
#include <sys/file.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>
#include <sys/dlpi.h>
#include <sys/pfmod.h>
#include “dltest.h”
#include “internet.h”
#include “ip.h”
#include “arp.h”
#include “os.h”
#include <sys/fcntl.h>

#defineEADDR_LEN 6
#defineETHERLEN  14
#define DLPI_DEVDIR     “/dev/”

extern int SNMP_socket;
extern void snmpHandleReq();

static char Ether_bdcst[] = { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff };

dlpiAttach(unsigned char *hwaddr, generic_iface_t gifp)
{
  struct strioctl si;
  struct ifreq ifr;
  int fd,i;
  struct dlpi_if *dlpip;
  struct iface_os *os;

  longbuf[MAXDLBUF];/* aligned on long */
  intppa;
  intsap;
  union DL_primitives *dlp;
  struct packetfilt pf;
```

```
  register u_short *fwp = pf.Pf_Filter;
  struct strbuf data;
  int flags;
  char *p, devname[512], *device;

  sap = 0;
  device = gifp->szName;

  /*
   * Split the device name into type and unit number.
   * Won`t work for devicenumbers larger than 9.
   */
  if ((p = (char *)strpbrk(device, "0123456789")) == NULL) {
    fprintf(stderr,"No such device: %s\n",device);
    exit(1);
  }

  strcpy(devname, DLPI_DEVDIR);
  strncat(devname, device, p - device);
  ppa = atoi(p);


  if((fd = open(devname, O_RDWR)) == -1) {
    fprintf(stderr,"Can't open %s\n",devname);
    exit(1);
  }
  /* Init all datastructures. */
  os = (struct iface_os *) gifp->os;
  os->link = LINK_ETHER;

  dlpip = (struct dlpi_if *) DALLOC(sizeof(struct dlpi_if));

  bcopy(hwaddr, os->hwaddr, EADDR_LEN);
  dlpip->sock = fd;
  os->low_os = (void *) dlpip;

  /* Attach. */
  dlattachreq(fd, ppa);
  dlokack(fd, buf);

  /* Receive all SAP`s */
  dlpromisconreq(fd, DL_PROMISC_SAP);
  dlokack(fd, buf);
  /* Receive all ether-addresses */
  dlpromisconreq(fd, DL_PROMISC_PHYS);
  dlokack(fd, buf);

  /* Bind. */
  dlbindreq(fd, sap, 0, DL_CLDLS, 0, 0);
  dlbindack(fd, buf);


#ifdef 0
  /* Couldn`t get the filter working. */
```

```
  /* Build filter. */
  for (i=0; i<EADDR_LEN; i += 2) { /* EADDR_LEN == 6 */
    /* Check for own address */
    *fwp++ = ENF_PUSHWORD + i/2;
    *fwp++ = ENF_PUSHLIT | ENF_CAND ;
    *fwp++ = *((u_short *) &hwaddr[i]);
  }
  pf.Pf_FilterLen = fwp - &pf.Pf_Filter[0];
  if (strioctl(fd, PFIOCSETF, -1, sizeof(struct packetfilt), (char
*)&pf)<0){
    perror("packetfilter");
    exit(1);
  }
#endif
  /* Raw mode. */
  if (strioctl(fd, DLIOCRAW, -1, 0, NULL)<0)
    syserr("DLIOCRAW");


  /* Join all-hosts multicast group */
  osJoinGroup(gifp, INADDR_ALLHOSTS);

  /* Flush the read side of the Stream. */
  if (ioctl(fd, I_FLUSH, FLUSHR) < 0)
    syserr("I_FLUSH");
  return fd;
}

int
dlpiInput(generic_iface_t *gifpp, unsigned char *buf, int len, int
timeout)
{
  struct strbuf data;
  struct dlpi_if *dlpip;
  struct iface_os *os;
  mcast_t mcast;
  fd_set readfds;
  generic_iface_t gifp;
  int res, flags,i;
  struct timeval tim, *tp;

  if(timeout > 0) {
    tim.tv_sec = timeout;
    tim.tv_usec = 0;
    tp = &tim;
  } else
    tp = (struct timeval *) NULL;

  FD_ZERO (&readfds);

  /* Add all DLPI descriptors to the file descriptor set */
  gifp = get_first_iface();
  while(gifp != (generic_iface_t) NULL) {
    os = (struct iface_os *) gifp->os;
    if(os->low_os != NULL) {
```

```
      dlpip = (struct dlpi_if *) os->low_os;
      FD_SET(dlpip->sock, &readfds);
    }
    gifp = get_next_iface(gifp);
  }

  /* Add the SNMP_socket to the file descriptor set */
  FD_SET(SNMP_socket, &readfds);

  /* wait for an incoming packet */
  if(select(FD_SETSIZE, &readfds, NULL, NULL, tp) > 0) {
    /* find which on which file descriptor there is an available packet */
    gifp = get_first_iface();
    while(gifp != (generic_iface_t) NULL) {
      os = (struct iface_os *) gifp->os;
      if(os->low_os != NULL) {
     dlpip = (struct dlpi_if *) os->low_os;
     if(FD_ISSET(dlpip->sock, &readfds)) {
       data.buf = buf;
       data.maxlen = len;
       data.len = 0;
       flags = 0;
       res = getmsg(dlpip->sock, NULL, &data, &flags);
       if (res == -1)
         perror("getmsg");

       /* Process the packet. */
       /* Since I didn't manage to get the filter working,
        * we must check the destination address.
        */
       if ((bcmp(data.buf,os->hwaddr, EADDR_LEN)==0) || (data.buf[0] & 1))
{
         *gifpp = gifp;
         return data.len;
       }
     }
      }
      gifp = get_next_iface(gifp);
    }
    if(FD_ISSET(SNMP_socket, &readfds))
      /* Received something on the SNMP_socket */
      snmpHandleReq();

    /* Received something else? */
    return 0;
  }
  /* We were interrupted, probably by SIGALRM */
  return -1;
}

/* Send raw packet (caller provides header) */
void
dlpiSendRaw(generic_iface_t gifp, unsigned char *pbuf, int len)
{
  struct strbuf proto, data;
```

```
  struct sockaddr sock;
  struct dlpi_if *dlpip;
  struct iface_os *os;

  os = (struct iface_os *) gifp->os;
  dlpip = (struct dlpi_if *) os->low_os;

  if (write(dlpip->sock, pbuf, len)<0)
    perror("write");
}
```

Appendix D The MINT Code

In this appendix the important changes to the Mobile-IP code done for the MINT port is included.

D.1 lowbpf.c

```
#include <stdio.h>
#include <mach.h>
#include <cthreads.h>
#include <errno.h>
#include <mach_init.h>
#include <mach_error.h>
#include <device/device.h>
#include <device/bpf.h>
#include <sys/file.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/param.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/udp.h>
#include <netinet/if_ether.h>
#include <arpa/inet.h>
#include <netdb.h>
#include “absiface.h”
#include “absmem.h”
#include “abstimer.h”
#include “lowbpf.h”
#include “targetdefs.h”
#include “mbuf.h”
#include “enet.h”
#include “internet.h”
#include “ip.h”
#include “arp.h”
#include “os.h”

/* Link level protocol identifiers for use in the packet header */
#define LINK_NONE       0       /* No link level header, assuming IP */
#define LINK_ETHER      1       /* 14 byte Ethernet header */
#define LINK_INTERNAL   2       /* Internal Ethernet link to upper engine
*/

#ifndefBUFSIZE
#defineBUFSIZE 1024
#endif

#defineMSG_TIMEOUT 500

#defineMAX_NUM_IFACES 16

#define TURN_PROMISC_OFF 0
#define TURN_PROMISC_ON 1
```

```
extern int SNMP_socket;

struct iface
{
  char name[IFNAMSIZ];
  struct in_addr ip_addr;
  char *mac_addr;
};

struct iface_pipe
{
  mach_port_tinput_port;
  mach_port_toutput_port;
};

struct iface *get_iface_info();
void hex_dump();
/*
 * bpfAttach
 */
bpfAttach(unsigned char *hwaddr, generic_iface_t gifp)
{
  mach_port_t filter_port;
  mach_port_t if_port = MACH_PORT_NULL;
  struct iface_pipe*if_pipe;
  char *ip_addr_str;
  kern_return_t rc;

  char *if_name;
  struct iface_os *os;

  if_name = gifp->szName;
  if (open_interface(if_name, &if_port))
     {
       fprintf(stderr, “Unable to open interface <%s>, aborting...\n”,
if_name);
       exit(1);
     }

  if (config_interface(if_name, if_port, TURN_PROMISC_ON))
     {
       fprintf(stderr, “Unable to configure interface <%s>.\n”,
                        if_name);
       exit(1);
     }

  if (config_filter(if_port, &filter_port, hwaddr))
     {
       fprintf(stderr, “Unable to configure filter for port 1 (%s).\n”,
                        if_name);
       exit(1);
     }

  os = (struct iface_os *)gifp->os;
  os->link = LINK_ETHER;
```

```
  bcopy(hwaddr, os->hwaddr, EADDR_LEN);
  if_pipe = (struct iface_pipe *)DALLOC(sizeof(struct iface_pipe));
  if_pipe->input_port = filter_port;
  if_pipe->output_port = if_port;
  os->low_os = (void *) if_pipe;

  osJoinGroup(gifp, INADDR_ALLHOSTS);
}
/*
 * open_interface
 */
int open_interface(char *if_name, mach_port_t *if_port)
{
  struct iface *if_1;
  struct net_status if_stat;
  natural_t if_stat_count;
  mach_port_t device_port;
  kern_return_t rc;

  if ((if_1 = get_iface_info((struct in_addr *)NULL, if_name, 1)) ==
(struct iface *)NULL)
     {
       fprintf(stderr, "open_interfaces(): Unable to get interface info
for <%s>\n", if_name);
       return(1);
     }

  device_port = mach_master_device_port();

  rc = device_open(device_port, D_READ, if_1->name, if_port);
  fprintf(stderr,"Opened interface if_name = %s, if_1->name =
%s\n",if_name, if_1->name);

  if (rc != D_SUCCESS)
     {
       fprintf(stderr, "open_interfaces(): device_open(%s) returned <0x%x,
%d>\n",
                         if_1->name, rc, rc);
       return(1);
     }

  /*
   * Get status and address from interface.
   */

  if_stat_count = NET_STATUS_COUNT;
  rc = device_get_status(*if_port,
                                              NET_STATUS,
                                              (dev_status_t)&if_stat,
                                              &if_stat_count);
  if (rc != D_SUCCESS)
     {
       fprintf(stderr, "open_interfaces(): device_get_status(%s) failed,
rc is <0x%x, %d>\n",
                         if_1->name, rc, rc);
```

```
        (void) device_close(*if_port);
        (void) mach_port_deallocate(mach_task_self(), *if_port);
        return (1);
      }

  return(0);
}
/*
 * config_filter
 */
int config_filter(mach_port_t if_port, mach_port_t *filter_port, unsigned
char *hwaddr)
{
  kern_return_t rc;
  struct bpf_insn bpfilter[NET_MAX_BPF];
  bpf_insn_t bpfp = bpfilter;
  int idx = 0;
  static priority = 1;
  /*
   * The filter
   */

  /*
   * Allocate a new port with a send right and receive right.
   * Keep the send right and send the receive right to the proxy task.
   */
  rc = mach_port_allocate(mach_task_self(),
                                                MACH_PORT_RIGHT_RECEIVE,
                                                filter_port);

  if (rc != KERN_SUCCESS)
     {
       fprintf(stderr, “config_null_filter(): mach_port_allocate():”, rc);
       (void)device_close(if_port);
       return(1);
     }

  rc = mach_port_insert_right(mach_task_self(),
                                                        *filter_port,
*filter_port,

MACH_MSG_TYPE_MAKE_SEND);
  if (rc != KERN_SUCCESS)
     {
       fprintf(“config_null_filter(): mach_port_insert_right():”, rc);
       (void) mach_port_destroy(mach_task_self(), *filter_port);
       (void) device_close(if_port);
       return(1);
     }

  /*
   * Build the filter
   */
  idx = 0;
```

```
  /*
   * XXX BPF type tag for kernel.
   */
  /* Since the interface fore som reason does not enter
   * promiscius mode, the below filter has not been tested
   * properly.
   */
  idx++;
  BPF_INSN_STMT(bpfp, BPF_BEGIN, 0);
  /*
  idx++;
  BPF_INSN_STMT(bpfp, BPF_LD+BPF_B+BPF_ABS, 0);
  idx++;
  BPF_INSN_JUMP(bpfp, BPF_JMP+BPF_JSET+BPF_K, 0x1, 4, 0);
  idx++;
  BPF_INSN_STMT(bpfp, BPF_LD+BPF_W+BPF_ABS, 2);
  idx++;
  BPF_INSN_JUMP(bpfp, BPF_JMP+BPF_JEQ+BPF_K, 0x0, 0, 3);
  idx++;
  BPF_INSN_STMT(bpfp, BPF_LD+BPF_H+BPF_ABS, 0);
  idx++;
  BPF_INSN_JUMP(bpfp, BPF_JMP+BPF_JEQ+BPF_K, 0x0, 0, 1);
  */
  idx++;
  BPF_INSN_STMT(bpfp, BPF_RET+BPF_K, -1);
  /*
  idx++;
  BPF_INSN_STMT(bpfp, BPF_RET+BPF_K, 0);
  */
  /* Patch */
  /* Insert our ethernet address into the right place
   * in the filter.
   */
  /*
  bcopy((char *)&hwaddr[2],(char *)&bpfilter[4].k,4);
  bcopy((char *)hwaddr,(char *)&bpfilter[6].k,2);
  */
  rc = device_set_filter(if_port,
                                                *filter_port,
                                                MACH_MSG_TYPE_MAKE_SEND,
                                                (priority++ %
NET_HI_PRI),     /* priority */
                                                (filter_array_t)bpfilter,
                                                (idx*sizeof(struct
bpf_insn))/sizeof(filter_t));
  if (rc != D_SUCCESS)
     {
       fprintf(stderr, “config_filter(): device_set_filter failed, rc is
<0x%x, %d>.\n”, rc, rc);
       return(rc);
     }

  return(0);
}
/*
```

```
 * get_interface_info
 */
struct iface *get_iface_info(iface_ip_addr, iface_name_str, how)
struct in_addr *iface_ip_addr;
char *iface_name_str;
int how;
{
  structifconf         ifc;
  structifreq          *ifr;
  structiface          *ifp;
  structsockaddr_innetaddr;
  structsockaddr_in*saddr;
  int           s, i;

  if ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
     {
       fprintf(stderr, “get_iface_info(): Error, socket() returned
<%d>.\n”, s);
       return((struct iface *)NULL);
     }

  if ((ifc.ifc_buf = (caddr_t)malloc(MAX_NUM_IFACES * sizeof(struct
ifreq))) == (caddr_t)NULL)
     {
       fprintf(stderr, “get_iface_info(): Error, malloc(%d) failed!\n”,
(MAX_NUM_IFACES * sizeof(struct ifreq)));
       return((struct iface *)NULL);
     }

  ifc.ifc_len = MAX_NUM_IFACES * sizeof(struct ifreq);

  if (ioctl(s, SIOCGIFCONF, (caddr_t)&ifc) < 0)
     {
       fprintf(stderr, “get_iface_info(): ioctl(%d, SIOCGIFCONF) failed,
errno is <%d>\n”,
                        s, errno);
       return((struct iface *)NULL);
     }

  ifr = ifc.ifc_req;

  for (i=0; i < MAX_NUM_IFACES; i++)
     if (!strncmp(iface_name_str, ifr[i].ifr_name, IFNAMSIZ))
       break;

  if (i == MAX_NUM_IFACES)
     {
       fprintf(stderr, “get_iface_info(): ran out of interfaces...\n”);
       return((struct iface *)NULL);
     }

  if ((ifp = (struct iface *)malloc(sizeof(struct iface))) == (struct
iface *)NULL)
     {
```

```
       fprintf(stderr, “get_iface_info(): Error, malloc(%d) failed!\n”,
sizeof(struct iface));
       return((struct iface *)NULL);
     }

  strncpy(ifp->name, ifr[i].ifr_name, IFNAMSIZ);

  saddr = (struct sockaddr_in *)&(ifr[i].ifr_addr);
  bcopy((char *)&(saddr->sin_addr), (char *)&(ifp->ip_addr), sizeof(struct
in_addr));

  return(ifp);
}
/*
 * config_interface
 */
int config_interface(char *iface, mach_port_t iport, int promisc_flag)
{
  int s;
  struct ifreq ifr;

  /*
   * turn on promiscuous mode...
   */
  {
     struct net_status if_stat;
     natural_t if_stat_count;
     kern_return_t rc;

     if_stat_count = NET_STATUS_COUNT;
     rc = device_get_status(iport,
                                                NET_STATUS,
                                                (dev_status_t)&if_stat,
                                                &if_stat_count);
     if (rc != D_SUCCESS)
       {
                fprintf(stderr, “config_interface(): Oops,
device_get_status(%s) failed, rc is <0x%x, %d>\n”,
                          iface, rc, rc);
                return (1);
       }

     if (!(if_stat.flags & IFF_UP))
       {
                fprintf(stderr, “config_interface(): Interface <%s> isn’t
up, aborting.\n”,
                          iface);
                return(1);
       }

     printf(“Interface status = <0x%x>\n”, if_stat.flags);
     printf(“Setting promiscuous mode...\n”);

     if (promisc_flag)
       if_stat.flags |= IFF_PROMISC;
```

```
      else
        if_stat.flags &= ~IFF_PROMISC;

      printf("Setting Interface status = <0x%x>\n",if_stat.flags);

      if_stat_count = NET_STATUS_COUNT;
      rc = device_set_status(iport,
                                                NET_STATUS,
                                                (dev_status_t)&if_stat,
                                                if_stat_count);
      if (rc != D_SUCCESS)
        {
                  fprintf(stderr, "config_interface():
device_get_status(%s) failed, rc is <0x%x, %d>\n",
                            iface, rc, rc);
                  return (1);
        }

      if_stat_count = NET_STATUS_COUNT;
      rc = device_get_status(iport,
                                                NET_STATUS,
                                                (dev_status_t)&if_stat,
                                                &if_stat_count);
      if (rc != D_SUCCESS)
        {
                  fprintf(stderr, "device_get_status(%s) failed, rc is
<0x%x, %d>\n",
                            iface, rc, rc);
                  return (1);
        }
      printf("Interface status = <0x%x>\n",if_stat.flags);
    }

    return(0);
}
/*
 * bpfInput
 */
int bpfInput(generic_iface_t *gifpp, unsigned char *buf, int timeout)
{
    struct bpf_if *bpfp;
    struct iface_os *os;
    generic_iface_t gifp;
    int res;
    struct timeval tim, *tp;
    mcast_t mcast;
    struct iface_pipe *if_pipe;
    static struct net_rcv_msg msg_buf;
    register net_rcv_msg_t msg;
    mach_msg_size_t max_msg_size = NET_RCV_MAX;
    kern_return_t kr;
    int len, data_len;
    u_char *header;

    msg = &msg_buf;
```

```
  gifp = get_first_iface();
  while (gifp != (generic_iface_t)NULL) {
     os = (struct iface_os *)gifp->os;
     if (os->low_os != NULL) {
       if_pipe = (struct iface_pipe *)os->low_os;
       kr = mach_msg(&msg->msg_hdr, (MACH_RCV_MSG | MACH_RCV_TIMEOUT),
                                       0, max_msg_size,
                                       if_pipe->input_port,
                                       timeout, MACH_PORT_NULL);
       if (kr == MACH_MSG_SUCCESS) {
                 /* copy packet header to buf */
                 bcopy((char *)msg->header, buf, ETHERLEN);

                 /* copy packet data to buf */
                 data_len = msg->packet_type.msgt_number - sizeof(struct
packet_header);
                 bcopy((char *)msg->packet+sizeof(struct packet_header),
buf+ETHERLEN, data_len);

                 len = ETHERLEN+data_len;

                 if (*(char *)buf & 0x1) {
                   /* broadcast or multicast */
                   if(!bcmp(Ether_bdcst, buf, EADDR_LEN)) {
                        /* accept broaddcast packets */
                        *gifpp = gifp;
                        fprintf(stderr,"Broadcast\n");
                        return len;
                   }
                   mcast = os->groups;
                   while(mcast != (mcast_t) NULL) {
                        /* accept multicast packet, if we are member */
                        if(!bcmp(mcast->hwaddr, buf, EADDR_LEN)) {
                          *gifpp = gifp;
                          return len;
                        }
                        mcast = mcast->next;
                   }
                 } else {
                   /* Accept packet addressed to unicast address */
                   *gifpp = gifp;
                   fprintf(stderr,"Unicast\n");
                   return len;
                 }
       } else
                 return 0;
     }
     gifp = get_next_iface(gifp);
  }
}
/*
 * bpfSendRaw
 */
void bpfSendRaw(generic_iface_t gifp, unsigned char *pbuf, int len)
{
```

```
  struct iface_os *os;
  struct iface_pipe *if_pipe;
  kern_return_t kr;

  fprintf(stderr,"bpfSendRaw() on iface %s\n",gifp->szName);
  hex_dump(pbuf,len);
  os = (struct iface_os *)gifp->os;
  if_pipe = (struct iface_pipe *)os->low_os;
  kr = device_write_request(if_pipe->output_port, MACH_PORT_NULL,
                                                        0, 0,
                                                        (char *)pbuf,
                                                        len);
  if (kr != KERN_SUCCESS)
     fprintf(stderr, "bpfSendRaw(): Warning, device_write_request failed,
kr is <0x%x>\n", kr);
}

int
check_snmp(int timeout)
{
  fd_set readfds;
  int res, flags;
  struct timeval tim, *tp;

  if(timeout > 0) {
    tim.tv_sec = 0;
    tim.tv_usec = timeout;
    tp = &tim;
  } else
    tp = (struct timeval *) NULL;
  FD_ZERO (&readfds);
  /* Add the SNMP_socket to the file descriptor set */
  FD_SET(SNMP_socket, &readfds);
  /* wait for an incoming packet */
  if(select(FD_SETSIZE, &readfds, NULL, NULL, tp) > 0) {
     if(FD_ISSET(SNMP_socket, &readfds)) /* Not needed but... */
       /* Received something on the SNMP_socket */
       snmpHandleReq();
     return 0;
  }
  /* We were interrupted, probably by SIGALRM */
  return -1;
}
```

Appendix E State diagrams

This appendix includes the state machines of the different entities in the Mobile-IP specification giving a more complete view of how the protocol works. In the diagrams only messages that results in a state change are present. Other legal transitions are possible, but they would not effect the state of the system.

The following definitions are used: rec() means that the entity receives a message, send() that it sends one and forw() that it forwards the message to another entity. State transitions with other types of labels are transitions where no messages are sent. The types of messages are:

- adv - agent advertisement
- sol - agent solicitation
- req - registration request
- rep - reply (positive or negative)
- PosRep - positive reply
- NegRep - negative reply
- DeReg - deregistration request
- pckt - any packet that is not directly part of the protocol
- Breq - broadcast registration request
- Bcast - broadcast/multicast packet

E.1 The Home Agent

This is the state diagram describing the behaviour of the Home Agent.

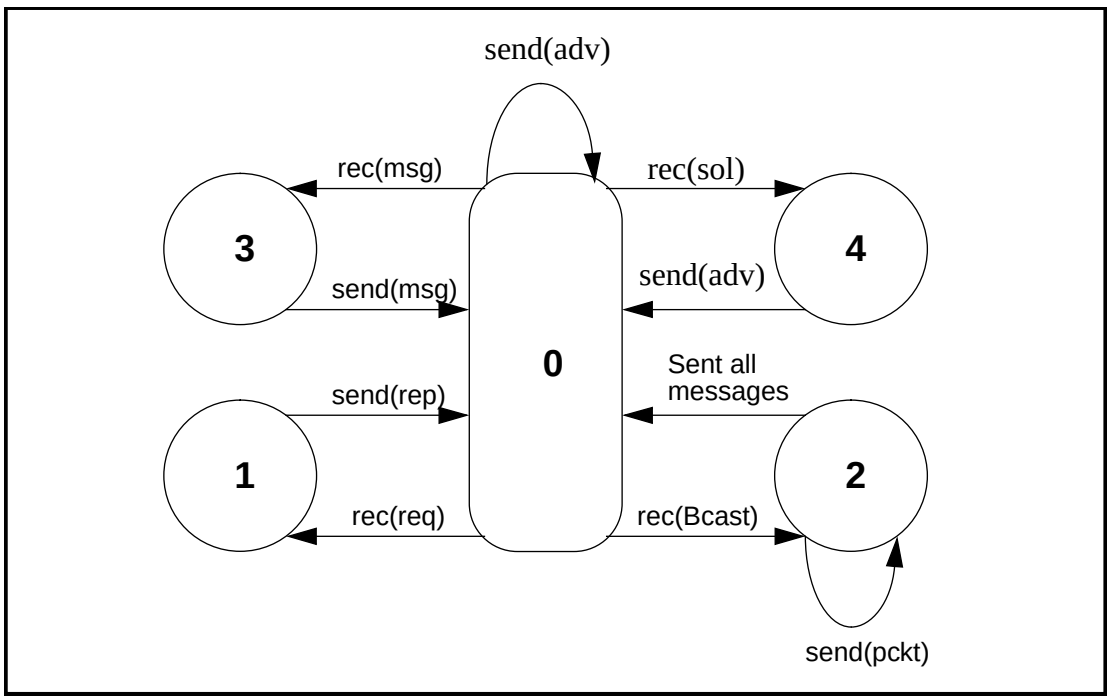

Figure 28. State diagram describing the Home Agent

State	State description
0	The Home Agent is waiting for messages to respond to, or it can send agent advertisements.
1	The HA received a registration request, decides whether to accept the request or not and sends the corresponding reply to the FA or MN.
2	A broadcast packet is received on the home network. The HA encapsulates the packet and sends it to all MNs that has requested that service.
3	A packet is received that has to be encapsulated and sent to one of the Home Agent's Mobile Nodes.
4	An agent solicitation is received, and an agent advertisement is sent.

Table 18: State description of the Home Agent

The transition between 2 and 0 called *Sent all messages* indicates that the Home Agent has sent the broadcast message to all Mobile Nodes that requested broadcast messages in their registration as described in section 8.6 of the Mobile-IP specification [5].

E.2 The Foreign Agent

Figure 29 shows the state diagram for the Foreign Agent.

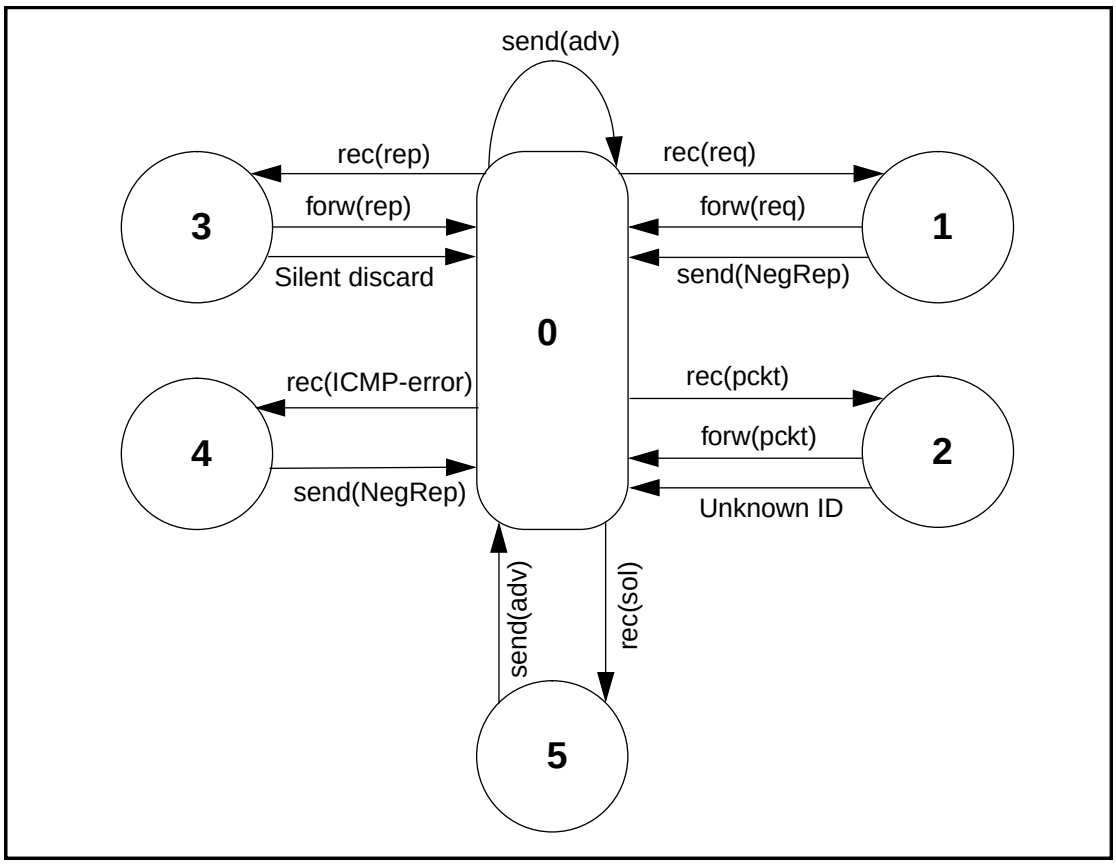

Figure 29. Statediagram describing the Foreign Agent

State	State description
0	The Foreign Agent is waiting for messages to respond to, or it can send agent advertisements.
1	The FA has received a registration request from a MN and has to decide whether to accept the request or not. In the former case it forwards the request to the HA and in the latter case it sends a negative reply to the MN.
2	The FA has received a packet that has to be encapsulated and forwarded to a MN. If the FA knows of the MN it sends the packet there, otherwise it silently discards the packet.
3	A registration reply is received. If the reply does not match the registration identification of its most recent registration request to the sender the message is silently discarded. Otherwise the reply is forwarded to the correct MN.
4	As a result of a registration request sent to a HA an ICMP-error is received. A reply with code 40 (unknown home agent address) is sent to the corresponding MN.
5	The FA receives an agent solicitation and responds with an agent advertisement.

Table 19: State description of the Foreign Agent

E.3 The Mobile Node

Here is the state diagram for the Mobile Node.

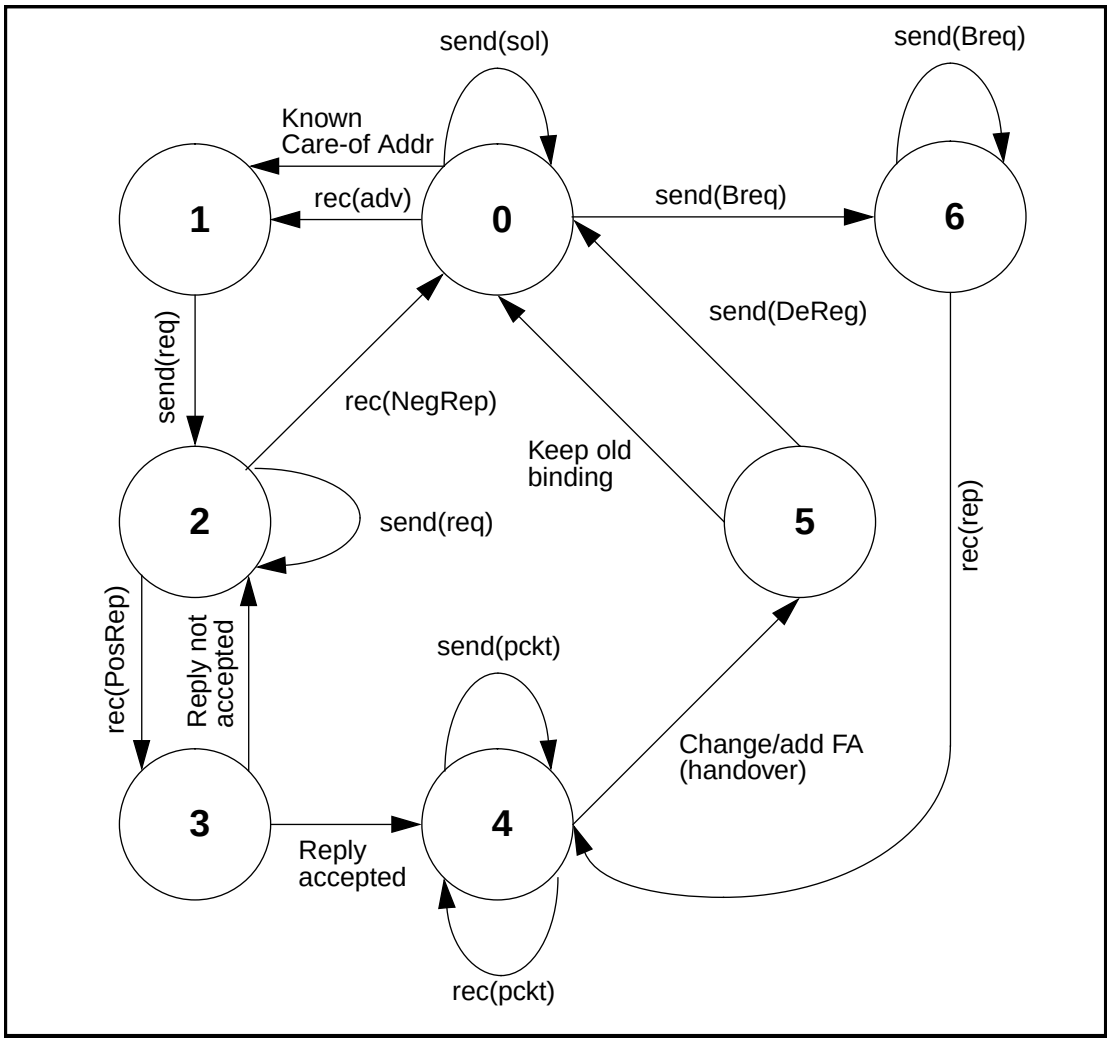

Figure 30. Statediagram describing the Mobile Node

State	State description
0	In this state the MN begins the process of registration. It can do one of several things. If the MN is at its home network and doesn't know the address of any of its Home Agents it can send a registration request to the directed broadcast address (and go to state 6). If the MN is visiting a foreign network and already has obtained a care-of address it can proceed directly to state 2. Else it waits for an agent advertisement (from a foreign agent) or tries to get one by sending agent solicitations.
1	The MN now knows its care-of address and sends a registration request to one of its Home Agents.

Table 20: State description of the Mobile Node

State	State description
2	The MN is now waiting for a registration reply. If it receives a negative reply the registration process starts all over. Else it keeps sending requests until it gets a positive reply.
3	When a positive reply is received, the MN determines whether the reply matches the registration identification of its most recently sent registration request to that HA. If not, the reply is silently discarded.
4	In this state the MN has got a valid registration with a HA and can send and receive packets as usual. The send(pckt) and rec(pckt) are not part of the Mobile-IP specification but rather there to indicate that the MN is working.
5	For some reason the MN wants to register with a new FA. It can either deregister with its old FA or add the new FA.
6	When the MN receives a reply from an Agent that is prepared to act as a HA for the MN, then, since the MN must be on its home network, it can go directly to state 4.

Table 20: State description of the Mobile Node

Appendix F Mobile-IP Watcher

```
#!/usr/local/bin/wish -f

# Check that the environment variable MIBFILE is set
if {[regexp MIBFILE [array names env]] == 0} {
     puts “Error: You must set the environment variable MIBFILE”
     exit
}

#*********************************************************************
# Globala variabler
#*********************************************************************
set g(snmpwalk) “./snmpwalk”
#set g(snmpwalk) /afs/it.kth.se/home/d91/d91-fbr/mobile-ip/mib/cmu-snmp2/apps/
snmpwalk
# The delay in milliseconds
set g(delay) 5000
set g(ipaddress) dumburken
set g(mnID) “.1.3.6.1.4.1.933.3.1”
set g(faID) “.1.3.6.1.4.1.933.3.2”
set g(haID) “.1.3.6.1.4.1.933.3.3”


#*********************************************************************
# textFrame
# Creates a labelled textwidget with scrollbar
# Input:  win - path name to a frame
#         title - the label above the text widget
#         ncols - the width of the text widget
#         nrows - the height of the text widget
# Output: a pointer to the text widget
#*********************************************************************
proc textFrame {win title ncols nrows} {
     label $win.l -text $title
     pack $win.l -side top -fill x
     frame $win.f
     text $win.f.t -height $nrows -width $ncols -relief sunken -bd 2 -
yscrollcommand “$win.f.sb set”
     scrollbar $win.f.sb -orient vertical -relief sunken -command “$win.f.t
yview”
     pack $win.f.t -side left -fill y
     pack $win.f.sb -side right -fill y
     pack $win.f -side left -fill x -fill y
     return $win.f.t
}


#*********************************************************************
# Mobile Node
#*********************************************************************
proc mobileNode {} {
     global g
     wm title . “Mobile Node: $g(ipaddress)”
```

```
     frame .f
     frame .f.f1
     frame .f.f1.a
     set varT [textFrame .f.f1.a “Variables” 40 15]
     pack .f.f1.a -side top -fill y

     frame .f.f1.b
     set halT [textFrame .f.f1.b “Home Agent List” 40 5]
     pack .f.f1.b -side top -fill y

     pack .f.f1 -side left -fill y

     frame .f.f2
     set regT [textFrame .f.f2 “Registration Table” 40 20]
     pack .f.f2 -side left -fill y

     frame .f.f3
     set pendRegT [textFrame .f.f3 “Pending Registration Table” 40 12]
     pack .f.f3 -side left -fill y

     pack .f

     button .b -text Quit -pady 5 -command exit
     pack .b -side bottom -fill x

     update

     # Parsa inmatningen och skriv till de olika fonstren
     while 1 {
          # Sudda fonster
          $varT delete 0.0 end
          $halT delete 0.0 end
          $regT delete 0.0 end
          $pendRegT delete 0.0 end

          # Anropa snmpwalk
          catch {set vars [exec $g(snmpwalk) $g(ipaddress) noAuth $g(mnID)]}
vars

          # Dela upp resultatet i rader
          set rowlist [split $vars \n]
          foreach row $rowlist {
               if {[regexp mipMN.* $row tmp]} {
                    set tmp [string range $tmp 6 end]
                    set wordlist [split $tmp .]
                    set key [lindex $wordlist 0]
                    if {$key == “mnHomeAgentList”} {
                         $halT insert end “[lindex $tmp 3]\n”
                    } elseif {$key == “mnRegTable”} {
                         regexp =.* $tmp tmp
                         $regT insert end “[lindex $wordlist 2] $tmp\n”
                    } elseif {$key == “mnPendRegTable”} {
                         regexp =.* $tmp tmp
                         $pendRegT insert end “[lindex $wordlist 2] $tmp\n”
                    } else {
```

```
                    $varT insert end “$tmp\n”
                }
            } else {
                $varT insert end “$row\n”
            }
        }
        update
        after $g(delay)
    }
}


#*********************************************************************
# Foreign Agent
#*********************************************************************
proc foreignAgent {} {
    global g
    wm title . “Foreign Agent: $g(ipaddress)”

    # Make windows
    frame .f
    frame .f.f1
    frame .f.f1.a
    set varT [textFrame .f.f1.a “Variables” 44 19]
    pack .f.f1.a -side top -fill y

    frame .f.f1.b
    set coaT [textFrame .f.f1.b “Care-Of Address List” 44 5]
    pack .f.f1.b -side top -fill y
    pack .f.f1 -side left -fill y

    frame .f.f2
    set regT [textFrame .f.f2 “Registration Table” 40 0]
    pack .f.f2 -side left -fill y

    frame .f.f3
    set pendRegT [textFrame .f.f3 “Pending Registration Table” 40 0]
    pack .f.f3 -side left -fill y

    pack .f

    # Quit-button
    button .b -text Quit -command exit
    pack .b -side bottom -fill x

    update

    while 1 {
        # Sudda fonster
        $varT delete 0.0 end
        $coaT delete 0.0 end
        $regT delete 0.0 end
        $pendRegT delete 0.0 end

        # Anropa snmpwalk
```

```
          catch {set vars [exec $g(snmpwalk) $g(ipaddress) noAuth $g(faID)]}
vars

          # Parsa inmatningen och skriv till de olika fonstren
          set rowlist [split $vars \n]
          foreach row $rowlist {
               if {[regexp mipFA.* $row tmp]} {
                    set tmp [string range $tmp 6 end]
                    set wordlist [split $tmp .]
                    set key [lindex $wordlist 0]
                    if {$key == “faCOAList”} {
                         $coaT insert end “[lindex $tmp 3]\n”
                    } elseif {$key == “faRegTable”} {
                         regexp =.* $tmp tmp
                         $regT insert end “[lindex $wordlist 2] $tmp\n”
                    } elseif {$key == “faPendRegTable”} {
                         regexp =.* $tmp tmp
                         $pendRegT insert end “[lindex $wordlist 2] $tmp\n”
                    } else {
                         $varT insert end “$tmp\n”
                    }
               } else {
                    $varT insert end “$row\n”
               }
          }
          update
          after $g(delay)
     }
}

#*********************************************************************
# Home Agent
#*********************************************************************
proc homeAgent {} {
     global g
     wm title . “Home Agent: $g(ipaddress)”

     frame .f
     frame .f.f1
     set varT [textFrame .f.f1 “Variables” 40 0]
     pack .f.f1 -side left -fill y

     frame .f.f2
     frame .f.f2.a
     set bindT [textFrame .f.f2.a “Mobility Binding Table” 44 16]
     pack .f.f2.a -side top
     frame .f.f2.b
     set authT [textFrame .f.f2.b “Authorized Node List” 44 4]
     pack .f.f2.b -side bottom

     pack .f.f2 -side right -fill y

     pack .f

     button .b -text Quit -command exit
```

```
     pack .b -side bottom -fill x

     update

     set a [exec ls]

     while 1 {
          $varT delete 0.0 end
          $bindT delete 0.0 end
          $authT delete 0.0 end

          # Anropa snmpwalk
          catch {set vars [exec $g(snmpwalk) $g(ipaddress) noAuth $g(haID)]}
vars

          # Parsa inmatningen och skriv till de olika fonstren
          set rowlist [split $vars \n]
          foreach row $rowlist {
               if {[regexp mipHA.* $row tmp]} {
                    set tmp [string range $tmp 6 end]
                    set wordlist [split $tmp .]
                    if {[lindex $wordlist 0] == “haBindingTable”} {
                         regexp =.* $tmp tmp
                         $bindT insert end “[lindex $wordlist 2] $tmp\n”
                    } elseif {[lindex $wordlist 0] == “haAuthNodeList”} {
                         $authT insert end “[lindex $tmp 3]\n”
                    } else {
                         $varT insert end “$tmp\n”
                    }
               } else {
                    $varT insert end “$row\n”
               }
          }
          update
          after $g(delay)
     }
}


#*********************************************************************
# inputAddress
# Asks for the IP-address where the snmp daemon is running, and sets
# the global variable g(ipaddress) to that value
#*********************************************************************
proc inputAddress {} {
     global g

     frame .t
     label .t.l -text “IP Address:”
     entry .t.e -width 15 -relief sunken -textvariable g(ipaddress)
     bind .t.e <Return> {
          destroy .t
     }

     pack .t.l .t.e -side left
```

```
     pack .t -padx 10 -pady 10

     focus .t.e
     tkwait window .t
}


#*********************************************************************
# First menu
# From here you choose witch entity you will monitor
#*********************************************************************
wm title . “Mobile-IP Watcher”
. configure -bd 2

frame .m1
label .m1.l1 -text “Mobile-IP Watcher” -font -*-Helvetica-Bold-R-Normal-*-180-*
label .m1.l2 -text “Choose an entity to monitor”
pack .m1.l1 .m1.l2 -side top -fill x
pack .m1

frame .m2
button .m2.b1 -text “Mobile Node” -padx 5 -pady 5 -command {
     destroy .m1 .m2
     set g(ipaddress) explorer
     inputAddress
     mobileNode
}
button .m2.b2 -text “Home Agent” -padx 5 -pady 5 -command {
     destroy .m1 .m2
     set g(ipaddress) dumburken
     inputAddress
     homeAgent
}
button .m2.b3 -text “Foreign Agent” -padx 5 -pady 5 -command {
     destroy .m1 .m2
     set g(ipaddress) anxiety
     inputAddress
     foreignAgent
}

button .m2.b4 -text “Quit” -padx 5 -pady 5 -command exit
pack .m2.b1 .m2.b3 .m2.b2 .m2.b4 -side left
pack .m2
```

Appendix G The Mobile-IP MIB

```
-- Mobile IP MIB
-- version 0.79
-- by Fredrik Tarberg and Fredrik Broman, KTH
-- 1995-11-17

-- HISTORY
-- v 0.79 Changed all timestamps from INTEGER to COUNTER
-- v 0.78 Corrected the flags field for advertisements
-- v 0.77 Changed the datatype for mnRegFlags and mnPendRegFlags from
--        INTEGER to BIT STRING.
--        Added mnAdvFlags, faAdvFlags, haAdvFlags and haBindingFlags.
-- v 0.75 Replaced all TimeStamp with INTEGER
-- v 0.71 Introduced version number

RFC1155-SMI DEFINITIONS ::= BEGIN;
      nullOID       OBJECT IDENTIFIER ::= { ccitt 0 }
      internet      OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
      directory     OBJECT IDENTIFIER ::= { internet 1 }
      mgmt          OBJECT IDENTIFIER ::= { internet 2 }
      experimental  OBJECT IDENTIFIER ::= { internet 3 }
      private       OBJECT IDENTIFIER ::= { internet 4 }
      enterprises   OBJECT IDENTIFIER ::= { private 1 }
END

CMU-MIB DEFINITIONS ::= BEGIN;
      Proteon OBJECT IDENTIFIER ::= { enterprises 1 }
      IBM OBJECT IDENTIFIER ::= { enterprises 2 }
      cmu OBJECT IDENTIFIER ::= { enterprises 3 }
      Unix OBJECT IDENTIFIER ::= { enterprises 4 }
      ACC OBJECT IDENTIFIER ::= { enterprises 5 }
      TWG OBJECT IDENTIFIER ::= { enterprises 6 }
      Cayman OBJECT IDENTIFIER ::= { enterprises 7 }
      PSI OBJECT IDENTIFIER ::= { enterprises 8 }
      Cisco OBJECT IDENTIFIER ::= { enterprises 9 }
      NSC OBJECT IDENTIFIER ::= { enterprises 10 }
      HP OBJECT IDENTIFIER ::= { enterprises 11 }
      Epilogue OBJECT IDENTIFIER ::= { enterprises 12 }
      UTK OBJECT IDENTIFIER ::= { enterprises 13 }
      BBN OBJECT IDENTIFIER ::= { enterprises 14 }
      Xylogics OBJECT IDENTIFIER ::= { enterprises 15 }
      Timeplex OBJECT IDENTIFIER ::= { enterprises 16 }
      Canstar OBJECT IDENTIFIER ::= { enterprises 17 }
      Wellfleet OBJECT IDENTIFIER ::= { enterprises 18 }
      TRW OBJECT IDENTIFIER ::= { enterprises 19 }
      MIT OBJECT IDENTIFIER ::= { enterprises 20 }
      EON OBJECT IDENTIFIER ::= { enterprises 21 }
      Spartacus OBJECT IDENTIFIER ::= { enterprises 22 }
      Excelan OBJECT IDENTIFIER ::= { enterprises 23 }
      Spider OBJECT IDENTIFIER ::= { enterprises 24 }
      NSFNET OBJECT IDENTIFIER ::= { enterprises 25 }
      HLS OBJECT IDENTIFIER ::= { enterprises 26 }
      Xyplex OBJECT IDENTIFIER ::= { enterprises 33 }
      Cray OBJECT IDENTIFIER ::= { enterprises 34 }
```

```
      Sun OBJECT IDENTIFIER ::= { enterprises 42 }
      Synoptics OBJECT IDENTIFIER ::= { enterprises 45 }
      DEC OBJECT IDENTIFIER ::= { enterprises 36 }
      TGV OBJECT IDENTIFIER ::= { enterprises 58 }
      Apple OBJECT IDENTIFIER ::= { enterprises 63 }
      NAT OBJECT IDENTIFIER ::= { enterprises 86 }
      SNMP-Research OBJECT IDENTIFIER ::= { enterprises 99 }
      FTP OBJECT IDENTIFIER ::= { enterprises 121 }
      Shiva OBJECT IDENTIFIER ::= { enterprises 166 }
      Transarc OBJECT IDENTIFIER ::= { enterprises 257 }
      Lexcel OBJECT IDENTIFIER ::= { enterprises 379 }
      Teleinformatics_Lab OBJECT IDENTIFIER ::= {enterprises 933}
END


-------------------------------------------------------------------
--                          MOBILE-IP MIB                        --
-------------------------------------------------------------------

MOBILE-IP-MIB DEFINITIONS ::= BEGIN

   IMPORTS
      mgmt, NetworkAddress, IpAddress, Counter, Gauge, TimeTicks
             FROM RFC1155-SMI
      OBJECT-TYPE
             FROM RFC-1212
      PhysAddress
             FROM RFC1213-MIB;

   mip OBJECT IDENTIFIER ::= {Teleinformatics_Lab 3}

   mipMN OBJECT IDENTIFIER ::= { mip 1 }
   mipFA OBJECT IDENTIFIER ::= { mip 2 }
   mipHA OBJECT IDENTIFIER ::= { mip 3 }

   mipType OBJECT-TYPE
      SYNTAX  BIT STRING {
                 mobileNode(0),   -- acting as a Mobile Node
                 foreignAgent(1),-- acting as a Foreign Agent
                 homeAgent(2)     -- acting as a Home Agent
                }
      ACCESS  read-write
      STATUS  mandatory
      DESCRIPTION
               "The indication of whether this entity is acting as a
               Mobile node, a Home Agent and/or a Foreign Agent."
      ::= { mip 4 }



   ---------------------------------------------------------------
   -- The Mobile Node
   ---------------------------------------------------------------

   --
```

```
-- Mobile Node Home Agent List
--

mnHomeAgentList OBJECT-TYPE
   SYNTAX  SEQUENCE OF MnHAEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
            "The Mobile Node's list of Home Agents"
   ::= { mipMN 1 }

mnHAEntry OBJECT-TYPE
   SYNTAX  MnHAEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
            "Information about one of the Home Agents"
   INDEX   { mnHALAddr }
   ::= { mnHomeAgentList 1 }

MnHAEntry ::=
   SEQUENCE {
          mnHALAddr
            IpAddress
   }

mnHALAddr OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The IP address of a Home Agent"
   ::= { mnHAEntry 1 }

--
-- Mobile Node Registration Table
--

mnRegTable OBJECT-TYPE
   SYNTAX  SEQUENCE OF MnRegEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
            "The Mobile Node's registration table"
   ::= { mipMN 2 }

mnRegEntry OBJECT-TYPE
   SYNTAX  MnRegEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
            "Information about a registration"
   INDEX   { mnRegHA, mnRegFA }
   ::= { mnRegTable 1 }
```

```
MnRegEntry ::=
   SEQUENCE {
           mnRegHA
             IpAddress,
           mnRegFA
             IpAddress,
           mnRegReqTS
             TimeTicks,
           mnRegReplTS
             COUNTER,
           mnRegFlags
             BIT STRING,
           mnRegLifetime
             INTEGER
   }

mnRegHA OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The IP-address of the Home Agent"
   ::= { mnRegEntry 1 }

mnRegFA OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The IP-address of the Foreign Agent"
   ::= { mnRegEntry 2 }

mnRegReqTS OBJECT-TYPE
   SYNTAX  COUNTER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The time when the first registration request was
            sent"
   ::= { mnRegEntry 3 }

mnRegReplTS OBJECT-TYPE
   SYNTAX  COUNTER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The time when the registration reply was received"
   ::= { mnRegEntry 4 }

mnRegFlags OBJECT-TYPE
   SYNTAX  BIT STRING {
             S(0),       -- retain prior mobility bindings
             B(1),       -- forward broadcasts
             D(2),       -- mn decapsulates itself
             M(3),       -- minimal encapsulation
```

```
            G(4)        -- GRE encapsulation
           }
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The flags field that was used in the request"
   ::= { mnRegEntry 5 }

mnRegLifetime OBJECT-TYPE
   SYNTAX  INTEGER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The remaining lifetime for this registration"
   ::= { mnRegEntry 6 }

--
-- Mobile Node Pending Registration Table
--

mnPendRegTable OBJECT-TYPE
   SYNTAX  SEQUENCE OF MnPendRegEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
            "The Mobile Node's pending registration table"
   ::= { mipMN 3 }

mnPendRegEntry OBJECT-TYPE
   SYNTAX  MnPendRegEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
            "Information about a pending registration"
   INDEX   { mnPendRegHA, mnPendRegFA }
   ::= { mnPendRegTable 1 }

MnPendRegEntry ::=
   SEQUENCE {
          mnPendRegHA
            IpAddress,
          mnPendRegFA
            IpAddress,
          mnPendRegReqTS
            COUNTER,
          mnPendRegReqs
            INTEGER,
          mnPendRegFlags
            BIT STRING
   }

mnPendRegHA OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
```

```
   DESCRIPTION
            "The IP-address of the Home Agent"
   ::= { mnPendRegEntry 1 }

mnPendRegFA OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The IP-address of the Foreign Agent"
   ::= { mnPendRegEntry 2 }

mnPendRegReqTS OBJECT-TYPE
   SYNTAX  COUNTER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The time when the first registration request was
            sent"
   ::= { mnPendRegEntry 3 }

mnPendRegReqs OBJECT-TYPE
   SYNTAX  INTEGER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The total number of registration requests sent"
   ::= { mnPendRegEntry 4 }

mnPendRegFlags OBJECT-TYPE
   SYNTAX  BIT STRING {
              S(0),       -- retain prior mobility bindings
              B(1),       -- forward broadcasts
              D(2),       -- mn decapsulates itself
              M(3),       -- minimal encapsulation
              G(4)        -- GRE encapsulation
             }
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The flags field that was used in the request"
   ::= { mnPendRegEntry 5 }

--
-- MN Advertisement
--

mnAdvAddr OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The IP address in the last received agent
            advertisement"
   ::= { mipMN 4 }
```

```
mnAdvSeqNo OBJECT-TYPE
   SYNTAX  INTEGER (0..65535)
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            “The sequence number in the last received agent
            advertisement”
   ::= { mipMN 5 }

mnAdvFlags OBJECT-TYPE
   SYNTAX  BIT STRING {
             R(0),       -- Foreign agent registration required
             B(1),       -- Busy bit
             H(2),       -- Offers service as Home Agent
             F(3),       -- Offers service as Foreign Agent
             M(4),       -- Offers minimal encapsulation
             G(5)        -- Offers GRE encapsulation
            }
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            “The flags in the last received agent advertisement”
   ::= { mipMN 6 }

mnAdvTS OBJECT-TYPE
   SYNTAX  COUNTER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            “The time when the last agent advertisement was
            received”
   ::= { mipMN 7 }

mnAdvCount OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            “The total number of agent advertisements received”
   ::= { mipMN 8 }

--
-- MN Error
--

mnErrAddr OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            “The IP address from which the last error message was
            received”
   ::= { mipMN 9 }
```

```
mnErrCode OBJECT-TYPE
   SYNTAX  INTEGER (0..255)
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The error code in the last received error message"
   ::= { mipMN 10 }

mnErrTS OBJECT-TYPE
   SYNTAX  COUNTER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The time when the last error message was received"
   ::= { mipMN 11 }

mnErrCount OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The total number of error messages received"
   ::= { mipMN 12 }

--

mnAuthCount OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The total number of authentication exceptions"
   ::= { mipMN 13 }

mnInvReplCount OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The total number of invalid replies"
   ::= { mipMN 14 }

--
-- MN Solicitation
--

mnSolTS OBJECT-TYPE
   SYNTAX  COUNTER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The time when the last agent solicitation message was
            sent"
   ::= { mipMN 15 }
```

```
mnSolCount OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The total number of agent solicitations sent"
   ::= { mipMN 16 }

--

mnDecaps OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The number of IP-packets decapsulated at the Mobile
            Node"
   ::= { mipMN 17 }

mnDiscards OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The number of IP-packets discarded at the Mobile Node"
   ::= { mipMN 18 }



-----------------------------------------------------------------
-- The Foreign Agent
-----------------------------------------------------------------

--
-- Foreign Agent Care-Of-Address List
--

faCOAList OBJECT-TYPE
   SYNTAX  SEQUENCE OF FaCOAEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
            "The Foreign Agents's list of care-of address, if any"
   ::= { mipFA 1 }

faCOAEntry OBJECT-TYPE
   SYNTAX  FaCOAEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
            "Information about a COA Address"
   INDEX   { faCOAddr }
   ::= { faCOAList 1 }

FaCOAEntry ::=
```

```
   SEQUENCE {
           faCOAddr
             IpAddress
   }

faCOAddr OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
             "The Care-of address"
   ::= { faCOAEntry 1 }

---
-- Foreign Agent Registration Table
---

faRegTable OBJECT-TYPE
   SYNTAX  SEQUENCE OF FaRegEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
             "The Foreign Agent's registration table"
   ::= { mipFA 2 }

faRegEntry OBJECT-TYPE
   SYNTAX  FaRegEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
             "Information about a visiting Mobile Node"
   INDEX   { faRegMN, faRegHA }
   ::= { faRegTable 1 }

FaRegEntry ::=
   SEQUENCE {
           faRegMN
             IpAddress
           faRegHA
             IpAddress,
           faRegReqTS
             COUNTER,
           faRegReplTS
             COUNTER,
           faRegLifetime
             INTEGER (0..65535)
   }

faRegMN OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
             "The home IP-address of the visiting Mobile Node"
   ::= { faRegEntry 1 }
```

```
faRegHA OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The IP-address of the Home Agent"
   ::= { faRegEntry 2 }

faRegReqTS OBJECT-TYPE
   SYNTAX  COUNTER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The time when the first registration request was
            sent"
   ::= { faRegEntry 3 }

faRegReplTS OBJECT-TYPE
   SYNTAX  COUNTER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The time when the registration reply was received"
   ::= { faRegEntry 4 }

faRegLifetime OBJECT-TYPE
   SYNTAX  INTEGER (0..65535)
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The remaining lifetime for this registration"
   ::= { faRegEntry 5 }

--
-- Foreign Agent Pending Registration Table
--

faPendRegTable OBJECT-TYPE
   SYNTAX  SEQUENCE OF FaPendRegEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
            "The Foreign Agent's pending registration table"
   ::= { mipFA 3 }

faPendRegEntry OBJECT-TYPE
   SYNTAX  FaPendRegEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
            "Information about a pending registration"
   INDEX   { faPendRegMN, faPendRegHA }
   ::= { faPendRegTable 1 }
```

```
FaPendRegEntry ::=
   SEQUENCE {
           faPendRegMN
             IpAddress,
           faPendRegHA
             IpAddress
   }

faPendRegMN OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The home IP-address of a visiting Mobile Node"
   ::= { faPendRegEntry 1 }

faPendRegHA OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The IP-address of the Home Agent"
   ::= { faPendRegEntry 2 }

--
-- FA Advertisement
--

faAdvSeqNo OBJECT-TYPE
   SYNTAX  INTEGER (0..65535)
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The sequence number in the last sent agent
            advertisement"
   ::= { mipFA 4 }

faAdvFlags OBJECT-TYPE
   SYNTAX  BIT STRING {
             R(0),       -- Foreign agent registration required
             B(1),       -- Busy bit
             H(2),       -- Offers service as Home Agent
             F(3),       -- Offers service as Foreign Agent
             M(4),       -- Offers minimal encapsulation
             G(5)        -- Offers GRE encapsulation
            }
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The sequence number in the last sent agent
            advertisement"
   ::= { mipFA 5 }

faAdvTS OBJECT-TYPE
   SYNTAX  COUNTER
```

```
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The sequence number in the last sent agent
            advertisement"
   ::= { mipFA 6 }

faAdvCount OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The sequence number in the last sent agent
            advertisement"
   ::= { mipFA 7 }

--
-- FA Solicitation
--

faSolAddr OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The IP address in the last received agent
            solicitation message"
   ::= { mipFA 8 }

faSolTS OBJECT-TYPE
   SYNTAX  COUNTER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The time when the last agent solicitation message was
            received"
   ::= { mipFA 9 }

faSolCount OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The total number of agent solicitation messages
            received"
   ::= { mipFA 10 }

--
-- FA Error messages received
--

faErrRecAddr OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
```

```
   DESCRIPTION
            "The IP address from which the last error message was
            received"
   ::= { mipFA 11 }

faErrRecCode OBJECT-TYPE
   SYNTAX  INTEGER (0..255)
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The error code in the last received error message"
   ::= { mipFA 12 }

faErrRecTS OBJECT-TYPE
   SYNTAX  COUNTER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The time when the last error message was received"
   ::= { mipFA 13 }

faErrRecCount OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The total number of error messages received"
   ::= { mipFA 14 }

--
-- FA Error messages sent
--

faErrSentAddr OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The IP address to which the last error message was
            sent"
   ::= { mipFA 15 }

faErrSentCode OBJECT-TYPE
   SYNTAX  INTEGER (0..255)
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The error code in the last sent error message"
   ::= { mipFA 16 }

faErrSentTS OBJECT-TYPE
   SYNTAX  COUNTER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
```

```
                “The time when the last error message was sent”
      ::= { mipFA 17 }

   faErrSentCount OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
                “The total number of error messages sent”
      ::= { mipFA 18 }

   --

   faAuthCount OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
                “The total number of authentication exceptions”
      ::= { mipFA 19 }


   faRegReqsRec OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
                “The number of registration requests received at the
                Foreign Agent”
      ::= { mipFA 20 }

   faDecaps OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
                “The number of IP-packets decapsulated at the Foreign
Agent”
      ::= { mipFA 21 }

   faDiscards OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
                “The number of IP-packets discarded”
      ::= { mipFA 22 }



   ----------------------------------------------------------------
   -- The Home Agent
   ----------------------------------------------------------------

   --
```

```
-- Home Agent Mobility Binding Table
--

haBindingTable OBJECT-TYPE
   SYNTAX  SEQUENCE OF HaBindingEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
            "The Home Agent's mobility binding table"
   ::= { mipHA 1 }

haBindingEntry OBJECT-TYPE
   SYNTAX  HaBindingEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
            "Information about a mobility binding"
   INDEX   { haBindingMN, haBindingCOA }
   ::= { haBindingTable 1 }

HaBindingEntry ::=
   SEQUENCE {
           haBindingMN
             IpAddress,
           haBindingCOA
             IpAddress,
           haBindingLifetime
             INTEGER (0..65535),
           haBindingFlags
             BIT STRING
}

haBindingMN OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The Mobile Node's home IP-address"
   ::= { haBindingEntry 1 }

haBindingCOA OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The Mobile Node's care-of address"
   ::= { haBindingEntry 2 }

haBindingLifetime OBJECT-TYPE
   SYNTAX  INTEGER (0..65535)
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The lifetime for the registration"
   ::= { haBindingEntry 3 }
```

```
haBindingFlags OBJECT-TYPE
   SYNTAX  BIT STRING {
              S(0),        -- retain prior mobility bindings
              B(1),        -- forward broadcasts
              D(2),        -- mn decapsulates itself
              M(3),        -- minimal encapsulation
              G(4)         -- GRE encapsulation
             }
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The flags field for this registration"
   ::= { haBindingEntry 4 }

--
-- Authorized Node List
--

haAuthNodeList OBJECT-TYPE
   SYNTAX  SEQUENCE OF HaANEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
            "The Home Agent's list of authorized mobile nodes"
   ::= { mipHA 2 }

haANEntry OBJECT-TYPE
   SYNTAX  HaANEntry
   ACCESS  not-accessible
   STATUS  mandatory
   DESCRIPTION
           "Information about an authorized mobile node"
   INDEX   { haANAddr }
   ::= { haAuthNodeList 1 }

HaANEntry ::=
   SEQUENCE {
           haANAddr
             IpAddress
   }

haANAddr OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
           "The IP address of an authorized mobile node"
   ::= { haANEntry 1 }

--
-- HA Advertisement
--

haAdvSeqNo OBJECT-TYPE
```

```
   SYNTAX  INTEGER (0..65535)
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The sequence number in the last sent agent
            advertisement"
   ::= { mipHA 3 }

haAdvFlags OBJECT-TYPE
   SYNTAX  BIT STRING {
             R(0),       -- Foreign agent registration required
             B(1),       -- Busy bit
             H(2),       -- Offers service as Home Agent
             F(3),       -- Offers service as Foreign Agent
             M(4),       -- Offers minimal encapsulation
             G(5)        -- Offers GRE encapsulation
            }
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The flags in the last sent agent advertisement"
   ::= { mipHA 4 }

haAdvTS OBJECT-TYPE
   SYNTAX  COUNTER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The time when the last agent advertisement was sent"
   ::= { mipHA 5 }

haAdvCount OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The total number of agent advertisements sent"
   ::= { mipHA 6 }

--
-- HA Solicitation
--

haSolAddr OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The IP address from which the last agent solicitation
            message was received"
   ::= { mipHA 7 }

haSolTS OBJECT-TYPE
   SYNTAX  COUNTER
   ACCESS  read-only
```

```
   STATUS  mandatory
   DESCRIPTION
            "The time when the last agent solicitation message was
            received"
   ::= { mipHA 8 }

haSolCount OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The total number of agent solicitation messages
            received"
   ::= { mipHA 9 }

--
-- HA Error
--

haErrAddr OBJECT-TYPE
   SYNTAX  IpAddress
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The IP address from which the last error message was
            received"
   ::= { mipHA 10 }

haErrCode OBJECT-TYPE
   SYNTAX  INTEGER (0..255)
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The error code in the last received error message"
   ::= { mipHA 11 }

haErrTS OBJECT-TYPE
   SYNTAX  COUNTER
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The time when the last error message was received"
   ::= { mipHA 12 }

haErrCount OBJECT-TYPE
   SYNTAX  Counter
   ACCESS  read-only
   STATUS  mandatory
   DESCRIPTION
            "The total number of error messages received"
   ::= { mipHA 13 }

--

haAuthCount OBJECT-TYPE
```

```
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
               “The total number of authentication exceptions”
      ::= { mipHA 14 }

   haRegReqsRec OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
               “The number of registration requests received”
      ::= { mipHA 15 }

   haEncaps OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
               “The number of IP-packets encapsulated”
      ::= { mipHA 16 }

   haBroadcastsRec OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
               “The number of broadcast packets received”
      ::= { mipHA 17 }

   haBroadcastsSent OBJECT-TYPE
      SYNTAX  Counter
      ACCESS  read-only
      STATUS  mandatory
      DESCRIPTION
               “The number of broadcast packets forwarded to Mobile Nodes”
      ::= { mipHA 18 }
END
```

References

[3] A. Klemets' latest version of the Mobile-IP implementation available from `ftp://ftp.it.kth.se/pub/klemets/`

[4] G. Q. Maguire Jr., M. T. Smith, T. Ohsawa, "Walkstation II project", 2nd International Workshop on Mobile Multi-Media Communications Workshop, Bristol, England, April 1-13, 1995.

[5] C. Perkins, "IP Mobility Support, draft-ietf-mobileip-protocol-14.txt", Internet Draft, December 1995.

[6] G. Maguire, F. Reichert, M. Smith, "A Multiport Mobile Internet Router", Proceedings of the 44th IEEE Vehicular Technology Conference '94, Volume 3, pages 1435-1439, Stockholm, Sweden, June 1994.

[7] A. Klemets, G. Q. Maguire Jr., F. Reichert, M. T. Smith. "MINT - A Mobile Internet Router", First IEEE International Symposium on Global Data Networking, pages 70-74, Institute of Electrical and Electronics Engineers, Cairo, Egypt, December 13-15, 1993. Note: Although this paper has the same title as the following one, they are quite different.

[8] R. Hager, A. Klemets, G. Q. Maguire Jr., M. T. Smith, F. Reichert, "MINT - A Mobile Internet Router", Proceedings of the IEEE Vehicular Technology Conference '93, pages 318-321. Institute of Electrical and Electronics Engineers, May 18-20, 1993.

[9] J. Ioannidis and G. Q. Maguire Jr., "The Design and Implementation of a Mobile Internetworking Architecture". USENIX Winter 1993 Technical Conference, pages 491-502. USENIX Association, January 1993.

[10] J. Ioannidis, D. Duchamp, G. Q. Maguire Jr., S. Deering, "Protocols for supporting Mobile IP hosts", Internet Draft, June 1992. Available from: `ftp://parcftp.xerox.com/pub/mobile-ip/columbia-draft-june-92`

[11] M. T. Rose, "The Simple Book: An Introduction to Internet Management", 1994.

[12] D. Duchamp, S. Feiner, G. Q. Maguire Jr., "Software Technology for Wireless Mobile Computing", IEEE Network 5(6):12-18, November 1991.

[13] R. Droms, "Dynamic Host Configuration Protocol", RFC 1541, October 1993.

[14] J. Postel, "User Datagram Protocol", RFC 768, August 1980.

[15] C. Perkins, "IP Encapsulation within IP, draft-ietf-ip4inip4-01.txt", Internet Draft, October 1995.

[16] S. M. Bellovin, "Security Problems in the TCP/IP Protocol Suite", ACM Computer Communications Review, 19(2), March 1989.

[17] R. Rivest. "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[18] A. Klemets, "Mach 3.0 as an Operating System for the MINT", October 1994, TRITA-IT R 94:20.

[19] J. Case, M. Fedor, M. Schoffstall, J. Davin, "A Simple Network Management Protocol (SNMP)", RFC 1157, May 1990.

[20] W. R. Stevens, "TCP/IP Illustrated, Volume 1", Addison-Wesley, 1994.

[21] M. Oelhafen, "SNMP Application for the MINT Router", June 1994., URL: `ftp://ftp.it.kth.se/www/documents/labs/ccs/WS/papers/940630-Oelhafen-A4.ps`

[22] Thomas L. Georges, "Using the Carnegie Mellon University (CMU) SNMP Library

To Build an SNMP Manager", March 1993, URL:
http://neptune.corp.harris.com/uniforum.html

[23] M. Rose, K. McCloghrie, "Concise MIB Definitions", RFC 1212, March 1991.

[24] K. McCloghrie, M. Rose, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", RFC 1213, March 1991.

[25] K. Sollins, "THE TFTP PROTOCOL (REVISION 2)", RFC 1350, July 1992.

[26] B. Croft, J. Gilmore, "BOOTSTRAP PROTOCOL (BOOTP)", RFC 951, September 1985.

[27] P. Guerin, "RadioNet Driver Implementation for the Mobile INTernet Router", 1994, URL:
ftp://ftp.it.kth.se/www/documents/labs/ccs/WS/papers/940630-Guerin-A4.ps

[28] N. Nuckolls, "How to Use DLPI", June 1992, URL:
ftp://opcom.sun.ca:/pub/drivers/dltest.tar.gz

[29] S. McCanne, V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture", December 1992, URL:
ftp://ftp.ee.lbl.gov/papers/bpf-usenix93.ps.Z

[30] G. R. Wright, W. R. Stevens, "TCP/IP Illustrated, Volume 2", Addison-Wesley, 1995.

[31] D. Cong, M. Hamlen, C. Perkins, "The Definitions of Managed Objects for the Home Agent function of IP Mobililty Support", Internet Draft, December 1995.

[32] D. Cong, M. Hamlen, C. Perkins, "The Definitions of Managed Objects for the Foreign Agent function of IP Mobililty Support", Internet Draft, December 1995.

[33] D. Cong, M. Hamlen, C. Perkins, "The Definitions of Managed Objects for the Mobile Node function of IP Mobililty Support", Internet Draft, December 1995.

[34] D. Cong, M. Hamlen, C. Perkins, "The Definitions of Managed Objects for the Security function of IP Mobililty Support", Internet Draft, December 1995.