# Filters for Mobile IP Implementation

# Overview

SunLabs Mobile IP software is a Mobile IP implementation for IPv4 networks, as specified in RFC 2002 (1). It consists of 2 separate software components, called the ***Agent software*** and the ***Mobile Node software***. It is a Linux based application and at present it is able to provide Mobile IP support on single network interface of a mobile node. Modern day mobile nodes have more that one network interface and requires that all these be usable when it is possible. These interfaces might be connected through different network technologies and can be up or down at different times. For example, a WLAN connection might not be active when the user of a mobile node is travelling on a Highway. The only connection that might be possible would be a GPRS connection.

Considering the same example, GPRS is usually expensive to use, compared to WLAN connection. Therefore the mobile node should be able to use these connections in a more inteligent maner, considering not only the cost, but many other criteria. ***Filters for Mobile IP*** (3) is a specification that specifies how a mobile node can be made to use it's network interfaces based on different criteria.

This document explains the implementation of the following capabilities on the SunLabs Mobile IP software.

- Multiple simultaneous (concurrent) bindings
- Filtering

It explains the current operation of the software, what was done to handle multiple simultaneous bindings, what was done to handle Filters for Mobile IP and finally, how the software can be operated.

# SunLabs Mobile IP

SunLabs Mobile IP enables Mobile IP support for environments that require mobility support. It consists of 2 separate software components. They are,

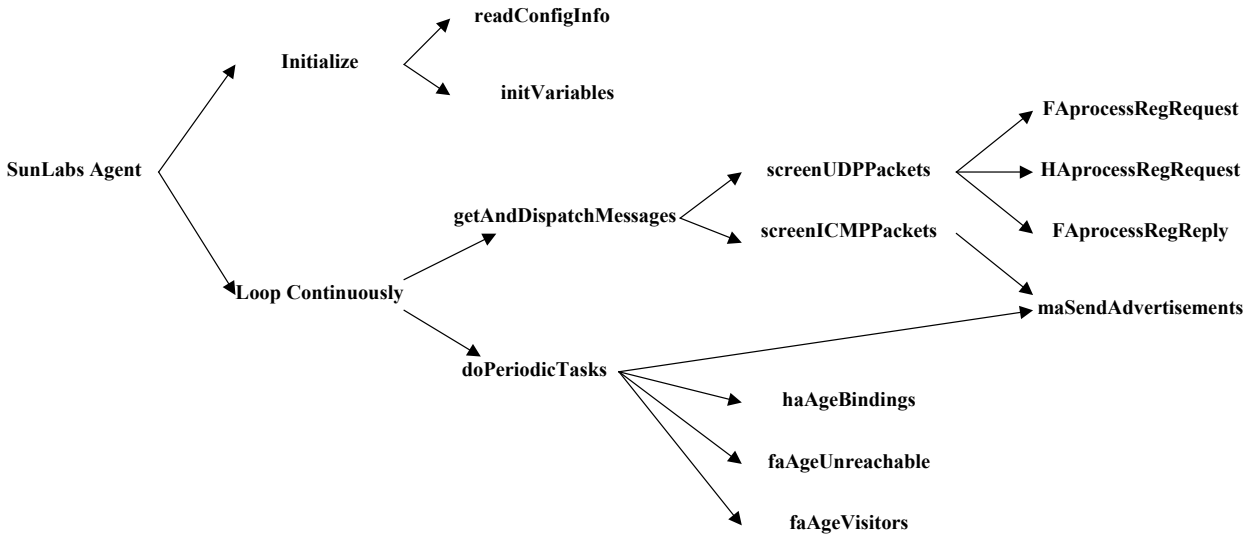
- Agent software
- Mobile Node software

The Agent is setup to run on a server of a network which requires Mobile IP support. The agent can act as a *Home Agent* or a *Foreign Agent* or as both (1,2). When the server is configured as a *Home Agent*, it should intercept and tunnel data to the mobile nodes that belong to it's network, when they are connected to the Internet through some other Mobile IP capable network. The data is orginally sent to the *Foreign Agent* in the network to which the mobile node is connected, who in turn forwards it to the mobile node. When the server in a network acts as a Foreign Agent, it should accept data sent on behalf of the mobile node and forward then to the mobile node.

The *Mobile Node software* is to be run on the mobile node. This software configurs the mobile node to be Mobile IP capable. When a mobile node is connected to a foreign network, it activates the Mobile IP operation to make connections with the Home Agent and when it is in the home network, reverts back to nomal IP behaviour. The Mobile Node software is able to handle the Foreign Agent Care-of-Address method as well as the Co-located Care-of-Address methods.

SunLabs Mobile IP is a user space application written to be executed in Unix based environments. This software was originally written for the Sun Solaris environment. Later it was ported to Linux 2.2. This software was changed by ComNets to handle Fast Handoff mechanisms (5) and the current ***Comnets version*** of this software is ***0.83***.

The Agent and the Mobile Node software have been developed in the C language and contains many IOCTL calls to manipulate the IP behaviour on the Linux kernel. The Agent software has the following program structure,




The basic structure of the Agent software is to loop continuously and process any messages if available. While iterating around this loop, it regularly performs a set of maintenance tasks. The time duration of performing these maintenance tasks is defined in the configuration file. The application has the following data structures to manage the operation.

- Table of interfaces on the server that are clssified as providing Mobile IP capabilities. These interfaces should be on the networks which host the different mobile nodes. This information is obtained from the configuration file.

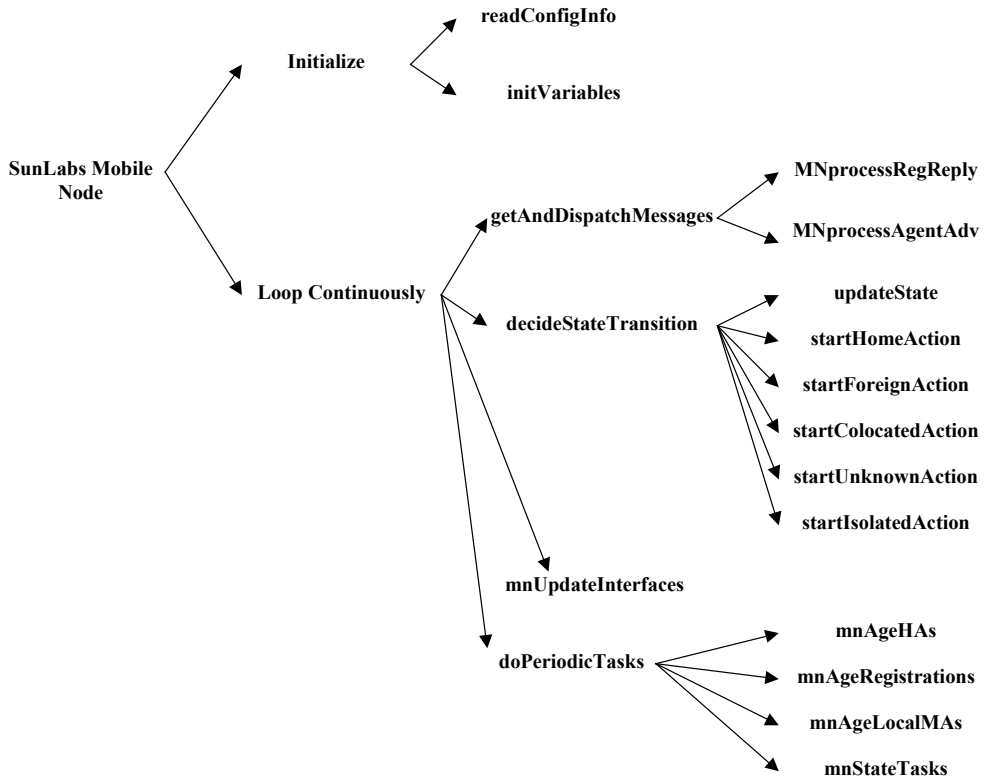
- Table to mainatain the visitors to the Mobile IP capable network. This table is valid only for Agents that act as Foreign Agent.

- Table of Home Agents from whom ICMP unreachable message was recived, recently.

- Table for each mobile node for which it provides Home Agent services. This information is read from the configuration file. Only valid for a Home Agent.

- Table for holding currently active bindings for each mobile node visiting some other Mobile IP capable network. Only valid for a Home Agent.

The Mobile Node software has the following program structure.




This too, loops indefinitely to perform message processing, state transition and update of the available network interfaces. While performing these, it also does some maintenance tasks repeatedly when a configuration file defined duration elapses. Following are the data structures it holds.

- Table to hold information related to the different network interfaces. They can be from wirelined (eth0, etc.) to wireless or logical (ppp0, etc.). Theses interfaces are monitored for any Agent Advertisement ICMP messages. If the IP address of the interface is the home address it starts sending Agent Solicitation ICMP messages.

- Table to hold the different Mobile IP supporting Agents from whom it recived advertisements. This table is updated when no Agent Advertisments are heard.

- Table of Home Agents valid for this mobile node. Some are defined in the configuration file, while it can also identify these dynamically.

- Table to hold the different registrations sent to the Home Agent.

- Structure to hold the current state of the mobile node. The sate can be,
    1. **Foreign** – If the mobile node is connected to a foreign network that provides Mobile IP support (MN uses a foreign agent's address).
    2. **Co-located** – If the mobile node is connected to a foreign network that does not provide Mobile IP support. The mobile node in this case should have obtained some address from the visiting network (thrugh DHCP, PPP).
    3. **Unknown** – If the mobile node has active network interfaces, but is not connected to any network.

4. **Isolated** – If the mobile node has no active interfaces.

In both applications (Mobile Node and Agent), all operating system calls are placed in 2 separate modules. Currently these modules are available for each platform (Solaris and Linux) that the software is capable of working. They contain IOCTL functions as well as direct exe commands that manipulate the IP environment. This software uses the following Kernel level operations,

- Configuring interfaces
- Creating IPIP tunnels
- Setting route entries
- Requesting Proxy ARPing

# Multiple Simultaneous Bindings for Mobile IP

A mobile node can have many network interfaces. These interfaces can be using different bearer technlogies. For example, one interface can be a wireless LAN connection while another can be a GPRS connection. These bearer technologies have differing characteristics that make them suitable at different times. Considering the same example, when a user is out on a moving train, the only connection possible might be GPRS. But at or close to the working place, a user can have access to Wireless LANs. Similarly, when all possible interfaces are active, the user might wish to use them for different activities, simultaneously, considering different factors such as bandwith requirement, cost, etc. This requires the mobile node to maintain a binding for each of the active interfaces.

The Mobile IP (ref nnn) specifies that a *Registration Request* of a mobile node should enable the *S bit* to indicate that it wants to hold multiple simultaneous bindings. This means, that when the Registration Request is received by the Agents (Home or Foreign), they do not purge any other bindings held for the same mobile node but, create a new binding. When there are multiple bindings of mobile node registered at the Home Agent, the RFC indicates that all packets destined to the mobile node should be multiplicated.

The SunLabs Mobile IP software does not allow to have multiple bindings for a mobile node. The Mobile Node software does not set the S bit in it's Registration Request and the Agent software also does not check for to see whether the S bit is set.

To make the SunLabs Mobile IP software handle multiple interfaces, the following changes were required. (in summary)

**Agent (Home and Foreign)**

- React to the S bit setting in an incomming registration request. For updating the binding list as well as removal of bindings

**Home Agent**

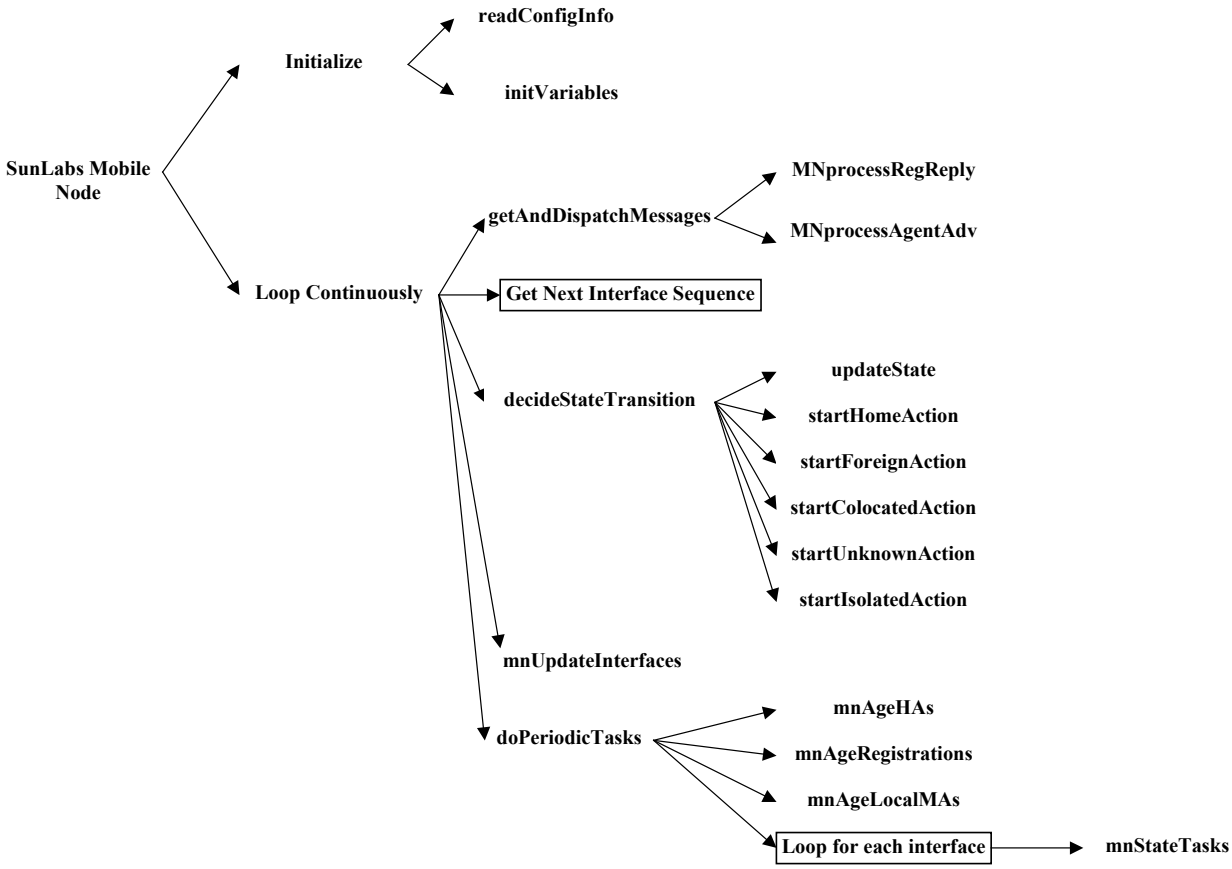
- Multiplicate all Mobile Node destined packets to all the active bindings

**Mobile Node**

- Change the data structures to allow the maintenance of multiple states, one for each active interface
- Relate interface and the data structures to assign the incomming or outgoing Mobile IP messages
- Enable S bit when sending Registration Requests

## Detailed Mobile Node Changes

Following is the changed program structure of the Mobile Node software of SunLabs Mobile IP. (i.e. functions with a rectangle around them)




The data structure meant to hold the state of the mobile node was changed to an array. Each of the elements represent one active interface. If no interfaces are active, then the 1st element of the array is taken to handle the state. Following is the new data structure introduced.

```
typedef struct is {
      char ifaceName[MAX_IFNAME_LEN];
      u32  COAddr;
      mnState currentState;
} InterfaceState;
```

When receiving Agent Advertisements or Registration Replys, identification was made to what interface through which they were received. This was required to update the data structures related to the binding specific to a interface. The following new functions were added.

**`int getIfcStateSeqFromName(char *ifaceName, InterfaceState ISTable[])`**

Return the sequence number of the element in the state data structure related to the given interface name.

**`int getIfcStateSeqFromCOA(u32 COAddr, InterfaceState ISTable[])`**

Return the sequence number of the element in the state data structure related to the given care-of-address of a binding.

**`int localMAAvailable(char *ifaceName, LocalMaEntry localMaTable[])`**

Checks if a local Foreign Agent is available for the given interface. This is reuired to determine if the interface is bound in Co-located mode or Foreign Agent mode.

**`int getNextIfcStateSeq(int lastISSeq, InterfaceState ISTable[])`**

Return the sequence number of the next intrface to consider for changing state. Finding the next sequence number is done in the main loop of the software.

**`int isValidInterface(char *ifaceName, IfaceEntry ifaceTable[], int ifaceTableSize)`**

Checks to see if an interface is still active in the interface table. This is required to update the state table.

**`int getIfaceAtHome(InterfaceState ISTable[])`**

Checks the state data structure to see if any interface is currently in the Home state. If an interface is in the Home state, all other active interfaces are not considered by the software.

**`int ifaceInColocated(InterfaceState ISTable[])`**

Checks the state data structure to see if any interface is currently in a Co-located mode. This is required to set the default route correctly when there are more that one active interface.

A Registration Request is sent to the Mobility Agents (Home agent, Foreign agent) when a interface becomes active. The formation of the Registration Request message was changed to include the S bit enabling so that the Mobility Agents will not remove any bindings related to the Mobile Node.

The software scans the active interface list to update the interface list data structures. This routine was changed to also update the state data structures to insert the newly available interfaces and to remove any dissapeared interfaces.

## Detailed Mobility Agent Changes

The major change in the Agent software was to allow the acceptance of multiple bindings per Mobile Node. This behaviour was only activated if an incomming Registration Request had the S bit set. If the S bit is not set, the Agent will remove all the bindings for the Mobile node before inserting the new binding.

The secondary change required was the implementation of packet multiplication action by the Home Agent. This activity was NOT programmed as it would require changing when implementing Filters for Mobile IP (next section)

# Filters for Mobile IP

A mobile node can have multiple network interfaces. As explained and implemented in the Multiple Simultaneous bindings for Mobile IP section, the standard Mobile IP indicates that all IP traffic to such mobile nodes should be multiplicated to all the interfaces. This means of operation is not very useful as this prevents the use of these network interfaces in a intelligent manner. For example, considering the same example in that section (WLAN & GPRS connections), when a WLAN connection is possible, it would be logical to stop any communication over the GPRS line, as this means is costly. But since Mobile IP specifies that all IP traffic should be multiplicated, this differentiation will not be able to be performed.

Similarly, there can be many such considerations as shown above, to differentiate IP traffic that is destined to the mobile node. They could be bandwidth, load balancing, special considerations such as video over a selected interface, etc.

Filters for Mobile IP is solution for this problem where the mobile node can instruct the Binding Agents to which it registers, to send incomming IP traffic though its active network interfaces, selectively. Filters for Mobile IP specifies the sending of a set of filtering instructions attached to the Registration Request as extensions. Once, the Binding Agent (Home Agent, Foreign Agent or other Agents related to HMIP (4)) receives these extensions, if it is a Filtering Agent, it sets its routing so that the traffic destined to the mobile node is differentiated as indicated in the filters.

The filters that are sent with a Registration Request consist of 3 distinct components. They are,

- Filter Management Data
- Filters
- Filter Modules

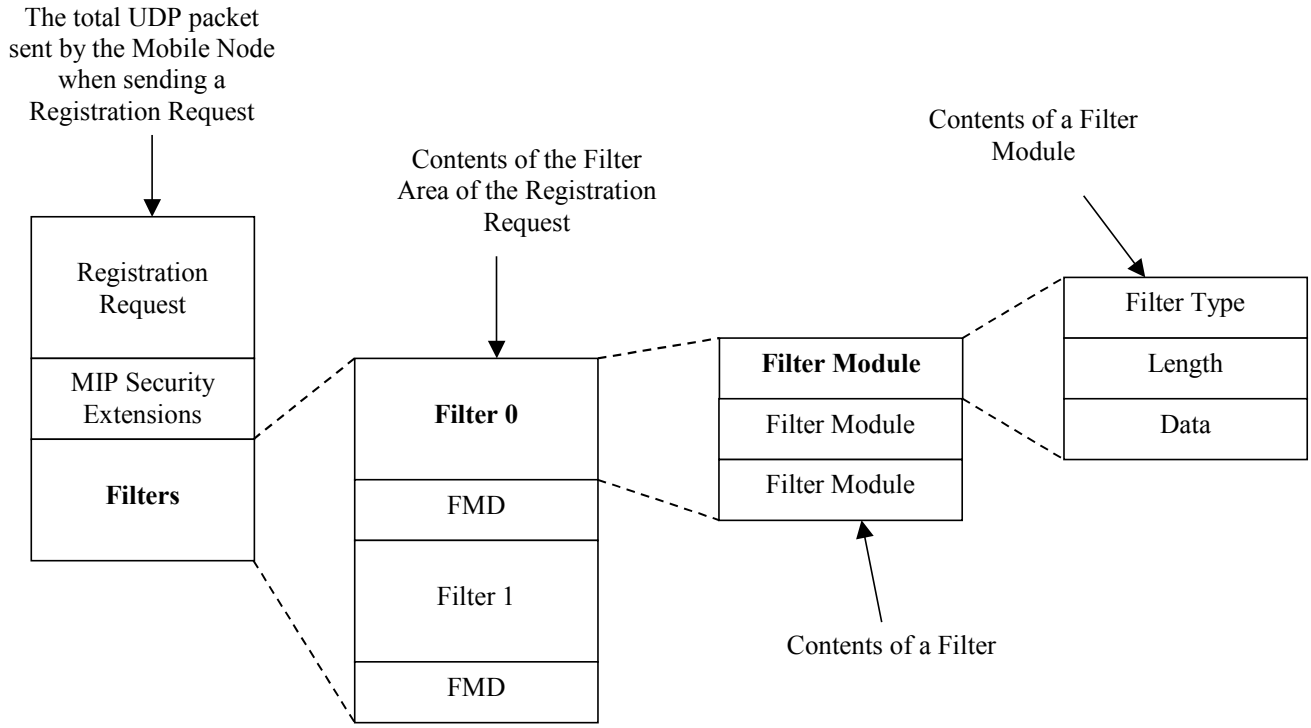
A Filter Module defines one criteria to check by the Filtering Agent when routing data to a specific interface of the mobile node. For example one such criteria would be the IP address of a correspondent node with which the mobile node is communicating.

A Filter is a collection of such Filtering Modules. I.e. a Filter can have multiple criteria. If a Filter has multiple criteria, the relationship between each criteria is considered as AND.

A Filter Management Data (FMD) is the instruction information about the Filters being sent with a Registration Request. Each Filter should be followed by a Filter Management Data component. A FMD contains the following information,

- **Target** – What needs to be done when a IP packet satisfies the condition specified in the Filter (route to an interface, drop packet silently, etc.)
- **Action** – What to do with the Filter being sent (can be added to the existing Filters, replace an existing filter, etc)
- **Index** – If the Action relates to a existing Filter (replace, delete, etc), then the sequence number of the Filter.

A single Registration Request can contain any number of Filters. The interface of the mobile node to which the packets should be routed (when criteria satisfies), is the Care-of-Address of the Registration Request through which the Filter was received by the Filtering Agent. When there are more than one Filter for a mobile node, the relationship between the Filters are OR. The Index value assigned to a Filter is also taken as the priotity of the Filter, i.e. it is the order in which the conditions (Filters) are evaluated. If one condition statisfies, the CoA associated with that filter is considered for routing the IP packet. The Index values are 0 based and the highest priority is recived by the lowest numbered Filter. The following diagram depicts the structure and the relationship of Filters.




Filters for Mobile IP specifies a group of filtering criteria through which different IP traffic flows can be identified and routed. Following are these filter criteria,

- **TOS Filter** – This filter allows the identification of the TOS value of an IP packet destined to the mobile node.
- **Protocol Filter** – Used to filter IP traffic based on the protocol field of an IP packet that is destined to the mobile node.
- **Source Address Filter** – Used to filter IP traffic based on the source IP address in an IP packet that is destined to the mobile node.
- **Source Network Filter** – Filters IP packets destined to the mobile node based on the network of the source address of a packet.
- **Source Port Filter** – Filters IP packets destined to a mobile node by using the source port number in the IP packet. This has to be used in conjunction with the Protocol Filter.
- **Source Port Range** – This filter checks for a conformance to a source port range in the IP packet. This too, is used in conjunction with the Protocol Filter.

- **Free-Form Filter** – This is a filter that allows any part of an IP packet to be used to filter. This filter allows the definition of an offset within the IP packet and the number of bytes there-of, to check.
- **Destination Port Filter** – Filters IP packets destined to a mobile node by using the destination port number in the IP packet. This has to be used in conjunction with the Protocol Filter.
- **Destination Port Range** – This filter checks for a conformance to a destination port range in the IP packet. This too, is used in conjunction with the Protocol Filter.

The Filtering Agent is the entity that identifies the traffic based on the above classified filter types and routes them to the mobile node. But there can be instances when an IP packet does not conform to any of the filtering criteria specified in the Filter list. In such a situation, all IP traffic should be routed through a **Default Filter**. The Default Filter is the filter created by Filtering Agent to pass IP traffic, when other filters do not satisfy. A default filter is found in the following manner.

1. If a binding exist without any filters, use that CoA for the Default Filter
2. Else use the the first binding's CoA for the Default Filter

SunLabs Mobile IP software required changes in the Agent component as well as the Mobile Node component to effect the Filters for Mobile IP behaviour. The changes are as follows (summary).

**Mobile Node**

- Read & display filter definition from a configuration file
- Attach the relevant filters in Registration Requests
- Re-attach the filters when a handoff occurs for any interface
- Apply reverse filters locally

**Agent**

- Extract, update & display filters from received Registration Requests
- Find the Default Filter
- Apply the filters as indicated

## Detailed Mobile Node Changes

The main configuration file of the Mobile Node software includes a flag called **FilterStatus** that indicates whether the software needs to consider about reading a filter definition file or not (1 or 0). The filter definition file contains the different filters (see Appendix for definition formats) associated with each of the network interfaces that is active or might come up in the future. Once read they are kept in an internal array through which they are printed out for the user to view. Following are the newly introduced functions to perform this activity.

```
int readFilterInfo(FILE *ffd)
```

Reads the filter information from a file named /etc/mipmn-filters.conf If the filter file contains any inconsistancies, the invokation of the software is totally terminated.

**void showFilterInfo()**

Displays the filter information read from the /etc/mipmn-filters.conf

The following data structures were introduced to hold filter information on the Mobile node software.

```
typedef struct filtermodule {
      int filterType;
      char *filterPtr;
} FilterModule;

typedef struct filter {
      int filterModuleCount;
      FilterModule filterModules[MAX_FILTER_MODULES_PER_FILTER];
      u32 boundCoA;
      int target;
      char boundIfaceName[MAX_IFNAME_LEN];
      int seq;
} Filter;
```

The above two structures are to hold general information about Filters and Filter Modules. The bottom structures define the data structures for each of the filter criteria types.

```
typedef struct bafextdatadata {
      int DSCP;
} BAFExtData;

typedef struct protoextdata {
      int protoNum;
} ProtoExtData;

typedef struct srcaddrextdata {
      u32 srcAddr;
} SrcAddrExtData;

typedef struct srcnetextdata {
      u32 srcAddr;
      u32 srcNetMask;
} SrcNetExtData;

typedef struct srcportextdata {
      short int portNum;
} SrcPortExtData;

typedef struct srcportrangeextdata {
      short int minPortNum;
      short int maxPortNum;
} SrcPortRangeExtData;

typedef struct freeformextdata {
      short int offset;
      int length;
      char value[MAX_FREEFORM_VAL_SIZE + 1];
      char mask[MAX_FREEFORM_MASK_SIZE + 1];
} FreeFormExtData;

typedef struct dstportextdata {
```

```
      short int portNum;
} DstPortExtData;

typedef struct dstportrangeextdata {
      short int minPortNum;
      short int maxPortNum;
} DstPortRangeExtData;
```

The following structure is used to hold what filters are required to be retransmitted. A filter is eligible for retransmission under the following two conditions.

- When an interface specified in one or set of filters become active
- When a handoff occurs for an interface.

```
typedef struct filterop {
      int action;
      int seq;
      u32 boundCoA;
      char boundIfaceName[MAX_IFNAME_LEN];
} FilterOp;
```

Once the filters are transmitted and locally applied, this structure is initialized.

Following fuctions where introduced in a new source file called **filters.c**, to effect the attaching and effecting of filters on the mobile node.

**`int appendFilterExt(unsigned char buffer[], int buflen, Filter fs[], int fsCnt, FilterOp fo[], int *foCnt, char *ifcname)`**

Appends the filters to a Registration Request if there are any filters available to transfer. Whether a filter needs to be transferred or not is dependent on the data in the *FilterOp* structure.

**`int setFilterForHomeBinding(Filter fs[], int fsCnt, FilterOp fo[], int *foCnt, char *ifcname)`**

When a mobile node is connected to the home network (i.e. when in HOME state), all actions of filtering should be terminated. This functions makes sets the *FilterOp* table to send a FLUSH all filters instruction.

**`int setFilterForNewBinding(Filter fs[], int fsCnt, FilterOp fo[], int *foCnt, char *ifcname)`**

When a network interface of a mobile node acquires a new state other that HOME (i.e. FOREIGN or COLOCATED), this function sets the *FilterOp* table to transmit all the filters associated with the handoff occuring network interface.

**`int applyReverseFilters(Filter fs[], InterfaceState ist[])`**

This function calls the routing functions of the mobile node to set the opposite of the Filters to effect **Revers Filtering** on the mobile node. Reverse Filters are the opposite of the defined filters, which are applied on the mobile node to make the outgoing IP traffic use the same network interfaces.

**`int isFilterAppliable(int elem, Filter fs[], InterfaceState ist[])`**

This function determines if a filter is appliable as a reverse filter. Certain filters such as user defined Default Filters are not appliable on the mobile node.

**u32 getValidRouterAddr(char *ifaceName, InterfaceState ist[])**

This function returns the Gateway used in the a Binding associated with the specified interface. This is required to set the routing entries of Reverse Filters.

**void deleteAllFilters(Filter fs[])**

This function calls the individual filter delete function to delete all the filters active at the mobile node. This deletion is required when the mobile node wants to re-apply filters or when the mobile node is connected to the home network (HOME status).

As indicated above, filters are attached to the registration request when sending them to the Filtering Agent. The following are the data structures used to place filter information to send with a Registration Request.

```
typedef struct controlext {
      unsigned char type;
      unsigned char length;
      unsigned char flags;
      unsigned char idx;
} controlExt;

typedef struct bafext {
      unsigned char type;
      unsigned char length;
      unsigned char DSCP;
} bafExt;

typedef struct protoext {
      unsigned char type;
      unsigned char length;
      unsigned char protoNum;
} protoExt;

typedef struct srcaddrext {
      unsigned char type;
      unsigned char length;
      unsigned char srcAddr[4];
} srcAddrExt;

struct srcNetEntry {
      unsigned char srcAddr[4];
      unsigned char srcNetMask[4];
};

typedef struct srcnetext {
      unsigned char type;
      unsigned char length;
      struct srcNetEntry srcNet;
} srcNetExt;

typedef struct srcportext {
      unsigned char type;
      unsigned char length;
      unsigned char portNum[2];
} srcPortExt;
```

```
struct portRangeEntry {
      unsigned char minPortNum[2];
      unsigned char maxPortNum[2];
};

typedef struct srcportrangeext {
      unsigned char type;
      unsigned char length;
      struct portRangeEntry portRange;
} srcPortRangeExt;

typedef struct freeformext {
      unsigned char type;
      unsigned char length;
      unsigned char offset[2];
      unsigned       char       valueNmask[(MAX_FREEFORM_VAL_SIZE/2)       +
(MAX_FREEFORM_MASK_SIZE/2)];
} freeFormExt

typedef struct dstportext {
      unsigned char type;
      unsigned char length;
      unsigned char portNum[2];
} dstPortExt;

typedef struct dstportrangeext {
      unsigned char type;
      unsigned char length;
      struct portRangeEntry portRange;
} dstPortRangeExt;
```

All operating system calls of SunLabs Mobile IP is included in a separate source module called **mipmodlinux.c** This file was changed to include the following functions,

**`int deleteFilters(int markValue, char *ifaceName)`**

This function deletes a reverse filter that has the given mark value. Interface name is also provided to assist the deletion.

**`int applyFilter(char *ifcName, char *fltStr, u32 routerAddr)`**

This function applies all the Reverse Filters related to all the active interfaces. Active interfaces are the interfaces which are currently bound through a Binding with the Home Agent.

**`int flushRouteCache()`**

This function flushes the route cache to make sure that the newly set Reverse Filters get effected immediately.

## Detailed Home Agent Changes

SunLabs Mobile IP Agent software can act as a Foreign Agent or as a Home Agent. Filters for Mobile IP has been implemented on the Home Agent part of the software. The Agent software used the same data structures as what the Mobile Node used. But, the Agent had to keep the Filters for not just one Mobile Node, but many. So the following data structure was introduced to hold Filters of each Mobile Node.

```
typedef struct boundmn {
      u32 mnAddr;
      int mnFilterCount;
      Filter mnFilters[MAX_FILTERS];
      int filtersChanged;
} BoundMN;
```

The **mnFilters**, in this structure was the link to the filters of each Mobile Node.

When filters of a mobile node is updated, the Filtering Agent will only apply filters that were changed. To identify what filters changed, the following operations structure is used.

```
typedef struct filteropentry {
      int opCode;          // 1 = APPLY filter, 2 = DELETE filter
      u32 boundCoA;
      int seq;
      int markValue;
} FltOpEntry;

typedef struct mnfltopentry {
      u32 mnaddr;
      FltOpEntry fltOpTable[MAX_FILTERS];
} MNFltOpEntry;
```

As indicated above, the first activity of the Agent software is to extract, update & display the filters. The following newly introduced functions in a new source file called **filters.c**, perform these activities.

**int extractFilters(int extCnt, unsigned char extType[], unsigned char *extIdx[], BoundMN bmn[], int *bts, u32 mnAddr, u32 coAddr)**

This function extracts the filters contained in a Registration Request and updates them in the data structures.

**void dispFilters(BoundMN bmn[], u32 mnAddr)**

Function to display the filters on the screen.

**int haveFiltersChanged(BoundMN bmn[], u32 mnAddr)**

This functions checks the modification flag in the filter data structures to determine if the filters belonging to a Mobile Node has been changed or not. This is required to display and apply the filters.

**int arrangeFiltersSequentially(BoundMN [], u32)**
**int sortFilters(BoundMN [], u32)**

The received filters within a Registration Request may have different Index numbers and might not be in an ascending order. Ordering and sorting of the filter list is required after a change in the filter list occurs. These 2 functions perform the activities of ordering and sorting.

**int addFilterTableEntry(BoundMN bmn[], u32 mnAddr)**

This is function adds a blank entry to hold filters of a bound Mobile Node. This function performs this insertion before updating the received filters.

**int removeSelectedFilters(BoundMN bmn[], int *bmnCnt, u32 mnAddr, u32 coAddr)**

This function performs the removal of the filters related to a specific binding. This is required when a binding of a Mobile Node goes down.

**void setFilterOp(int opCode, u32 mnAddr, u32 coAddr, int seq, int markValue, int pos)**

When filters are changed, only the changed filters need to be applied. This method updates an array with the changed filters. These are picked up by the filter application function.

Once the filters are received and updated or updated due to a invalidation of a binding, filters will be applied on the Home Agent. Application of filters refer to the setting of routing on the Home Agent to identify Mobile Node destined IP traffic based on the specified filters and to place them in the appropriate network interface (i.e. tunnels created for each binding). Following are the newly introduced functions that perform this task.

**int HAapplyFilters(BoundMN bmn[], HaBindingEntry habe[], u32 mnAddr)**

Function that calls the OS specific routines to set the routing entries to filter the different IP packets based on the specified filters. This function goes through the operations array, identifying the changed filters and creating a string with the filter information. That string is passed to the OS specific routines.

**u32 getDefaultFilterCoA(BoundMN bmn[], HaBindingEntry habe[], u32 mnAddr)**

When a IP packet does not conform to any filter, it has to be sent through the Default Filter. The Default Filter is the Filter with the lowest priority (i.e. the highest Index number). This funcion finds the Default Filter based on this criteria. This function is called by the filter application function.

**int getMarkValue(u32 mnAddr, int fltSeq)**

Filters for Mobile IP is implemented using a mechanism that identifies a flow using a mark. This function return the mark associated for filter belonging to a specific mobile node.

**void freeMarkValue(u32 mnAddr, int fltSeq)**

This function releases a mark used to identify a filter.

**int removeSelectedFilters(BoundMN bmn[], int *bmnCnt, u32 mnAddr, u32 coAddr)**

Function to remove filters related to a specific binding. This function places delete requests on the operations array, which are effected when application of filters occur.

Application of filters require the call of OS specific functions. These functions are kept in a source file called mipmodlinux.c This souce file was changed to include the following functions to perform the actual OS level calls to set the filters.

**int applyFilter(u32 mnaddr, u32 coaddr, char *fltStr)**

This function applies one filter for the given Mobile Node. This application can a delete or a creation. All details related to the filter are sent as a string.

**int flushRouteCache()**

Function to flush the routing cache after setting the filters. This is required to effect the operation of the filters immediately.

## Reverse Filters with Filters for Mobile IP

Filters for Mobile IP specifies the filtering of flows that is to be effected at a Binding Agent. Therefore, Filters for Mobile IP set filters only for IP traffic which are destined to the mobile node. It does not specify how the out going traffic from the mobile node should be sent. This is considered as being a vendor specific activity.

Logically, the mobile node should use the same outward path for any inward flow. This means that if a filter indicates that all packets from detination 134.102.158.1 are to be recived over the GPRS connection of a mobile node, then the same path should be taken when sending traffic to the given address. Setting such a filtering behaviour is termed as **Reverse Filtering**.

This implementation consideres the setting up of reverse filters and uses the **inverted condition** to effect a reverse of a filter. For example, if a filter is set to have the condition source network to be 134.102.158.0/24, then a reverse filter will be created at the mobile node that says to use the same path for traffic that have the destination network as 134.102.158.0/24.

## Effecting Filter Behaviour on Linux

This implementation of Filters for Mobile IP uses IPTABLES and IPROUTE2 functions to effect the filters on the Linux O/S. IPTABLES is used to mark packets based on the filters and IPROUTE2 is used to route packets to the respective CoA, once they are marked.

### IPTABLES

IPTABLES is a Linux based route manipulation tool that allows the checking of IP packets at a computer and then, to make changes to it's journey. This tool can manipulate IP packets that are arriving from outside the machine as well as any packet that is generated inside the computer. This facility uses a set of manipulation paths (called chains) to identify different traffic flows. These chains are as follows.

- As soon as it comes to a computer (INPUT)
- When a packet is generated by the computer (OUTPUT)
- Before checking how an IP packet needs to be routed (PREROUTING)
- After checking the routing table (POSTROUTING)
- Before forwarding an IP packet out of the computer (FORWARD)

It provides a set of tables to which these manipulation possibilities are associated. To manipulate an IP packet, it can check different information contained in the packet (such as destination IP address, TOS values, etc.) and, can make changes to some of this information in the packet (such as TOS, marking, etc). A user can avail the services of IPTABLES by running the function **iptables** at the command line. Filters for Mobile IP uses the marking

feature of IPTABLES to mark packets when it conforms to a specific criteria. The numbers used to mark a packet start from 1 and each flow identified by a filter is marked with these numbers.

Filters for Mobile IP settings at the Filtering Agent uses the mangle table and the PREROUTING capability of IPTABLES. The PREROUTING chain is used, as the Filtering Agent simply transfers the packet destined to a mobile node. Reverse filters (for outward IP traffic) use the mangle table and the OUTPUT chain as the IP packets are generated at the mobile node. The SunLabs Mobile IP software has been changed to use the following IPTABLES functionality.

- iptables -t mangle -A PREROUTING
- iptables -t mangle -A INPUT
- iptables -t mangle -D

**IPROUTE2**

IPROUTE2 is the second generation functionality made available on the Linux kernel to manipulate the IP environment. This functionality introduces new possibilities on the kernel including such things as rule based routing, multiple routing tables, etc. A user can avail these functions through the use of the **ip** command. This command has a number of switches that allows us to set the IP environment to cater to different requirements.

Filters for Mobile IP requires that an Agent be able to route data according to a specified set of conditions. The IPTABLES function marks a packet when it conforms to some condition and using IPROUTE2 functionality, these marked packets are sent to the mobile node base through the related CoA. SunLabs Mobile IP has been changed to use the following functions of the IPROUTE2,

- ip route
- ip rule

The **ip rule** function allows the setting of a rule to check whether a packet contains a specific mark. The **ip route** function allows the definition of new routing tables. Using these 2 commands we could define conditions, that when satisfied, could use different routing tables.

A routing table is a table that provides different routing paths based on the destination address of an IP packet. With IPROUTE2, these routing tables can be given names and these names should be placed in the file **/etc/iproute2/rt_tables**.

**Example of Calls to IPTABLES and IPROUTE2**

Considering an example, following is a filter and how it is effected using IPTABLES/IPROUTE2 commands.

Filter Information
Registration Request
Home IP Address 134.102.158.22

Care-of-Address 134.102.158.89
Filters
Seq 0
Target Accept
Action Append
Filter Modules
Module 0
Type Source Address
IP Address 134.102.186.10

Module 1
Type Behaviour Aggregate
TOS Value 04

Seq 1
Target Accet
Action Append
Filter Modules
Module 0
Type Source Network
Source Network 134.102.20.20/24

When these filter details are received at the Home Agent, the following ommand would be issued.

**`echo 167 mn134.102.158.22-134.102.158.89 > /etc/iproute2/rt_tables`**

This statement inserts the table name in the /etc/iproute2/rt_tables file. This table name, '**mn134.102.158.22-134.102.158.89**', will be used to associate all the routing entries related to one binding of a Mobile Node.

**`iptables -t mangle -A PREROUTING -v -d 134.102.158.22 -s 134.102.186.10 -m tos --tos 0x04 -j MARK --set-mark 1`**

This command adds a condition to check if an IP packet conforms to the condition specified in the first filter (source address and TOS value). If the packet conforms to this multi-criteria condition, it is marked with the number 1.

**`ip rule add fwmark 1 to 134.102.158.22 table mn134.102.158.22-134.102.158.89`**

This command sets the rule that allows the marked packet (by mark number 1) to find the correct routing table.

**`iptables -t mangle -A PREROUTING -v -d 134.102.158.22 -s 134.102.20.20/24 -j MARK --set-mark 2`**

This command sets the second filter (source network). If a packet conforms to this condition, it is marked with number 2

**`ip rule add fwmark 2 to 134.102.158.22 table mn134.102.158.22-134.102.158.89`**

This command sets the rule that allows the packets marked by the second filter, to find the correct routing table.

**`ip route add default dev tunl1 table mn134.102.158.22-134.102.158.89`**

This command creates the routing table associated with a specific binding of a Mobile Node.

The SunLabs Mobile IP software was changed to use the following versions of IPTABLES and IPROUTE2.

- IPROUTE2 - Version **iproute2-ss000305**
- IPTABLES - Version **iptables v1.2.7a**

Please refer to Linux documentation on how to install IPTABLES and IPROUTE2. Make sure, that the correct Kernel setings are done to activate the diferent filtering posibilities related to IPTABLES and IPROUTE2.

## Current Status

The IPTABLES together with IPROUTE2 facility on Linux allows the easy implementation of a considerable amount of the filters. Following are the filters that were implemented.

- Behaviour Aggregate (TOS)
- Protocol
- Source IP Address
- Source Network
- Source Port
- Source Port Range
- Destination Port
- Destination Port Range

The Free Form filter which was not implemented, require to check arbitrary locations in an IP packet. This means that a module should be attached to IPTABLES function to perform this checking. The next activity in accomodating Filters for Mobile IP in the SunLabs software is to write this module.

# Configuring Filters in SunLabs Mobile IP

SunLabs Mobile IP contains 2 configuration files, one each for the Agent and the Mobile Node software. They are

- /etc/mipmn.conf – for the Mobile Node
- /etc/mipagent.conf – for the Agent

To instruct the Mobile Node about the filters, a new configuration file has been introduced to the Mobile Node. This file is named **/etc/mipmn-filters.conf**. This file is read by the Mobile Node just after reading the standard configuration file (mipmn.conf). The decision to read the filter configuration file is done only after investigating the **FilterStatus** variable in mipmn.conf. Following are the list of information that needs to be placed in this file.

- Number of Filters
- Filter Sequence
- Interface Name
- Action
- Target
- Number of Filter Modules
- Type of Filter Extension
- Value(s) related to the Filter Type

**Filter Sequence**
This is the Index number assigned to the Filter. A Filter can have an Index number ranging between 0 – 255.

**Interface Name**
This is the name assigned to the mobility supporting network interface on the Mobile Node. This can have names such as eth0, eth1, ppp0, etc.

**Action**
When a Filter is received, the Home Agent must know what should be done with this filter, in respect of the Filter list it maintains for a Mobile Node. Following are the possible values that specifiy this action.

| | |
|---|---|
| 0 | Insert at the beginning of existing filters for the registered address |
| 1 | Append at the end of existing filters for the registered address |
| 2 | Delete from an exiting filter (requires the Seq i.e. Index) |
| 3 | Replace entry in an existing filter (requires the Seq i.e. Index) |
| 4 | Flush all filter entries for address |

**Target**
When an IP packet satisfies the conditions of a Filter, Filters for Mobile IP requires that we state what action we take. The possible Targets and their values are,

0 Drop incoming packets without notification
1 Reject incoming packets with notification
2 Accept incoming packets
3 Accept incoming packets but avoid route optimization
4 Masquerade

**Type of Filter Extensions**

Each Filter type is identified by an unique number. These numbers are a set of temporary numbers. The actual numbers would be assigned by IANA, later. Following are the current Filter type numbers.

67 Behavior Aggregate Filter Extension
68 Protocol Extension
69 Source Address Extension
70 Source Network Extension
71 Source Port Extension
72 Source Port Range Extension
73 Free-Form Extension
76 Destination Port Extension
77 Destination Port Range Extension

**Value(s) Related to the Filter Types**

Each Filter type has different valued parameters, which are specific to that each filter type. Following are the list of these Filters and some sample values.

| Type | DSCP Entry(6 bit value) |
|---|---|
| 67 | 22 |

| Filter Type | Protocol Number |
|---|---|
| 68 | 4 |

| Filter Type | Source IP Addresse |
|---|---|
| 69 | 134.102.158.1 |

| Filter Type | Source IP Addresse & Source IP Address Mask | |
|---|---|---|
| 70 | 134.102.158.65 | 255.255.255.248 |

| Filter Type | Port Number |
|---|---|
| 71 | 8819 |

| Filter Type | Source Port Number Range(Min - Max) | |
|---|---|---|
| 72 | 9002 | 9004 |

| Filter Type | Offset | Value & Mask (value & mask should be in hex) | |
|---|---|---|---|
| 73 | 8 | ABCDEF | FFFFFF |

| Filter Type | Port Number |
|---|---|
| 76 | 7200 |

| Filter Type | Source Port Number Range(Min - Max) | |
|---|---|---|
| 77 | 8002 | 8004 |

# References

[1] C. E. Perkins. IP Mobility Support. Request for Comment (Proposed Standard) 2002, Internet Engineering Task Force, Oct, 1996.

[2] C. E. Perkins. Mobile IP, Design Principles and Practices. Wireless Communications Series. Addison-Wesley, 1997.

[3] N.A. Fikouras, A.J. Koensgen, C. Goerg, W. Zirwas, M. Lott. Filters for Mobile IP Bindings (NOMAD).draft-nomad-mobileip-filters-02.txt, IETF, July 2002.

[4] E. Gustsfsson, A. Jonsson and C. Perkins. Mobile IP Regional Registration. (work in progress), draft-ietf-mobileip-reg-tunnel-06.txt, IETF, Oct 2002.

[5] N.A. Fikouras and C. Görg, "Performance comparison of hinted and advertisement based movement detection methods," paper presented at the 7th European conference on Fixed Radio Systems and Networks, Dresden, Germany, Sep. 2000.

[6] N. A. Fikouras, A. J. Könsgen, and C. Görg, "Accelerating mobile IP hand-offs through link-layer information," *Proceedings of the International Multi conference on Measurement, Modeling, and Evaluation of Computer-Communication Systems (MMB)*, Aachen, Germany, Sep. 2001.

# Appendix

## Sample Agent Configuration File

/etc/mipagent.conf

```
# Sample configuration file for mobility agents. Lines starting with the
hash
# character are treated as comments. Blank lines are ignored.
# It contains six main parts (the following ordering must be preserved):
#    1. version indicator
#    2. (optional) attribute value pairs
#    3. number of mobility supporting interfaces
#    4. configuration info for each mobility supporting interface
#    5. number of mobile nodes to which HA services are offered.
#    6. configuration information for each such mobile node.
#
# Note that part 2 is optional and if item 5 is zero, item 6 need
# not be present.

# version number for the configuration file. This line is required
# and must be the first non-comment/non-blank line.

version     1

# Other (optional) attribute-value pairs. Note that the attribute names are
# case insensitive. Currently the following attributes are allowed:
#
#  debuglevel       0-3 (controls verbosity of debug messages, 0=severe
#                        problems, 1=unexpected behavior, 2=important
events
#                        3=complete trace including messages)
#  IDfreshnessSLack  n  (When using timestamps for replay protection,
#                    this is the maximum skew tolerated in timestamp
#                      comparisons).
#  regLifetime       n  (Lifetime advertised in the mobility extension)
#  advLifetime       n  (Lifetime advertised in the RFC1256 portion)
#  periodicInterval  n  (Controls the frequency of advertisements and
#                        the granularity of other internal timers, e.g.
#                        aging of various bindings etc). This interval
#                        must be less than 1/3 advLifeTime.
#  advertiseOnBcast  1  (If 1, advertisements are sent on 255.255.255.255
#                        rather than 224.0.0.1)
#
# The agent program initializes appropriate default values for these
# parameters in agent.c (near the start).

debuglevel  3
IDfreshnessSlack 300
reglifetime  1000
advlifetime  3
periodicInterval 1
granularity 100000
advertiseOnBcast 0
```

```
# number of mobility supporting interfaces
1

# one line for each interface containing:
#   interface name, addr, netmask, advertised services flag, and prefix
flag
# The advertised services flag should be a hexadecimal number obtained
# by the logical ORing of some combination of the following.
#
#    ADV_IS_HOME_AGENT          0x20
#    ADV_IS_FOREIGN_AGENT       0x10
#    ADV_MIN_ENCAP              0x08
#    ADV_GRE_ENCAP              0x04
#    ADV_VJ_COMPRESSION         0x028
#
# It is invalid to set any service flag which is not currently implemented.
# The prefix flag is either 0 or 1 and controls whether prefix length
# extensions are included in agent advertisements (1 = include).
# In the following example, advertisements sent out on le0 offer
# both home agent and foreign agent services (0x20 | 0x10 = 0x30)
# and prefix length extensions are included.
eth1  134.102.158.17     255.255.255.248 00:04:76:11:DF:98     20     1

# number of supported mobile nodes
1

# one line for each supported mobile node containing:
#  addr, HA's addr on home network, SPI, Replay Prot code, key len, and key
# The replay protection code determines the replay protection
# algorithm used (0=NONE, 1=TIMESTAMPS, 2=NONCES). In the following
# example, SPI is 1, NONCES are used for replay protection and the
# key is 16 byte long with each byte being 0x11 (i.e. hexadecimal 11).
134.102.158.22    134.102.158.17  257  0  16
11111111111111111111111111111111
```

## Sample Mobile Node Configuration Files

/etc/mipmn.conf

```
# Sample configuration file for mobile nodes. Lines starting with the hash
# character are treated as comments. Blank lines are ignored.
# It contains six main parts (the following ordering must be preserved):
#    1. version indicator
#    2. (optional) attribute value pairs
#    3. Home Address, HomeNetmask  and default routers at home
#    4. number of home agents
#    5. configuration info for each home agent
#    6. Firewall traversal information

# version number for the configuration file. This line is required
# and must be the first non-comment/non-blank line.
version 1

# Other (optional) attribute-value pairs. Note that the attribute names are
# case insensitive and the program uses appropriate default values for
# attributes not specified in this configuration file. The unit of time is
# seconds. Currently the following attributes are allowed:
#
# debuglevel            0-3 (controls verbosity of debug messages, 0=severe
#                            problems, 1=unexpected behavior, 2=important
#                            events, 3=complete trace including messages)
# periodicInterval      n   (Controls the granularity of various internal
#                           internal timers, e.g. aging of local mobility
#                     agent table)
# agentSolicitThreshold t1  (If a mobile node hasn't heard from an agent in
#                           the last t1 seconds, it explicitly solicits an
#                           agent advertisement as a means ensuring that
#                           the agent is indeed unreachable)
# agentExpireThreshold  t2  (If a mobile node hasn't heard from an agent in
#                           the last t2 (> t1) seconds, it considers that
#                           agent unusable).
# ifacePollThreshold    n   (A mobile node checks its list of available
#                           network interfaces, once every n seconds.
# regLifetime           n   (Default registration lifetime requested in
#                           the absence of any agent advertisements).
# regRoundTrip          n   (Very rough round trip estimate for scheduling
#                           retransmissions)
# regflags        n   (Discretionary flag values to be used in a
#                     registration request)
#
#     The regflags value should be a hexadecimal number obtained
#     by the logical ORing of some combination of the following.
#           REQUEST_SIMULTANEOUS_BINDING  0x80
#           REQUEST_BROADCAST_DATAGRAMS   0x40
#           REQUEST_MINIMAL_ENCAPSULATION   0x10
#           REQUEST_GRE_ENCAPSULATION     0x08
#           REQUEST_VJ_COMPRESSION        0x40
#
#  haCooloffPeriond     n  (Time period for which to mark home agent
#                     unusable if a registration request is
#                     is rejected by home agent with a reason that
#                           can not be immediately fixed)
#  faCooloffPeriond     n  (Time period for which to mark foreign agent
#                     unusable if a registration request is
```

```
#                    is rejected by foreign agent with a reason that
#                           can not be immediately fixed)
#  IDfreshnessSLack     n  (When using timestamps for replay protection,
#                         this is the maximum skew tolerated in timestamp
#                           comparisons).
#
#  ignoreIfaces          ifname1 ifname2
#                            (List of interfaces that should be ignored
#                            by the mobile node. These interfaces are not
#                            monitored for agent advertisements or
#                            for a potential co-located address).
#
#  retransmissionPolicy x1 x2 .... (Multiples of estimated roundtrip times
#                                  after which successive retransmissions of
#                                  a registration request are sent, e.g. the
#                                  first retransmission occurs at x1 times
#                                  regRoundTrip and if no reply arrives in
#                                  the intervening period, the second
#                                  occurs after another x2 times regRoundTrip
#                                  and so on upto a maximum.
#
#  renewalPolicy  x1 x2 ..... (Accepted, renewable registrations are
#                                     renewd when remaining time becomes x1
#                                     times the granted time. Another renewal
#                                     is sent, if necessary, when remaining
#                                     time drops to x2 times the originally
#                                     granted lifetime and so on.
#
#  FilterStatus          n (0 indicates no filters, 1 indicates filters
#                             available in the file /etc/mipmn-filters.conf"
#

debuglevel                    3
periodicInterval         1
agentSolicitThreshold    2
agentExpireThreshold     3
ifacePollThreshold            1
regLifetime                   1000
regRoundTrip                  5
regFlags                      128
haCooloffPeriod               30
faCooloffPeriod               30
IDfreshnessSlack         300
retransmissionPolicy     2     4     8
renewalPolicy     0.5    0.25 0.1
movementDetection        2
#HApriority                   0
EagerSoliciting               1
#LazySoliciting               0
FilterStatus                  1

# Information about the mobile node's home network
# <Home Address of Mobile node>     <Netmask> <default routers>
# <default routers> is either a list of IP addresses in dotted decimal or
# the keyword "discover" (without quotes).
# NOTE: Currently, only an explicit list of routers is supported. To be
#       safe, we recommend that you specify only one default router
#       instead of a list. Different operating systems deal differently
#       with multiple default routing entries.
# MODIFY THIS FOR YOUR LOCAL ENVIRONMENT
134.102.158.78          255.255.255.248    134.102.158.73
```

```
# Number of home agent entries (entries beginning with a dotted decimal
# IP address specify a unique home agent, entries beginning with the
# keyword "default", without quotes, correspond to dynamically discovered
# home agents.
# MODIFY THIS FOR YOUR LOCAL ENVIRONMENT
1

# One line for each home agent
# HA's addr on home network, SPI, Replay Prot code, key len, and key
# HA's addr is in decimal format or "default"
# The replay protection code determines the replay protection
# algorithm used (0=NONE, 1=TIMESTAMPS, 2=NONCES). In the following
# example, when talking to home agent 129.146.122.191, the mobile node
# uses SPI 257, NONCE-based replay protection and a 16 byte long key
# with each byte being 0x11 (i.e. hexadecimal 11). In addition, the mobile
# node can dynamically discover other home agents and for them it uses
# SPI 570, TIMESTAMP based replay protection, 16-byte key with each byte
# equal to 0x22 (i.e. decimal 34).
# 129.146.122.191  257  2  16    11111111111111111111111111111111
# default          570  1  16    22222222222222222222222222222222
# Make sure that the home agents have been appropriately configured
# to support this mobile node and both use the same set of security
# parameters for mutual communication, i.e. the home agent
# 129.146.122.191 should be configured to support 129.146.122.195 as
# a mobile node and use the same SPI, replay protection method and
# shared secret as specified for 129.146.122.191 in this file.
# MODIFY THIS FOR YOUR LOCAL ENVIRONMENT
134.102.158.73   257  0   16  11111111111111111111111111111111
```

/etc/mipmn-filter.conf

```
# This file specifies the different filters that are associated
# with each of the bindings (interfaces)
#
# Values :
#
# Filter types
# 67 = Behavior Aggregate Filter Extension
# 68 = Protocol Extension
# 69 = Source Address Extension
# 70 = Source Network Extension
# 71 = Source Port Extension
# 72 = Source Port Range Extension
# 73 = Free-Form Extension
# 76 = Destination Port Extension
# 77 = Destination Port Range Extension
#
#
#Admissible values for the Action(ACT) field are as follows:
#      Value      Filter Management
#      0          Insert at the beginning of existing
#                 filters for the registered address
#      1          append at the end of existing filters
#                 for the registered address
#      2          Delete from an exiting filter
#      3          Replace entry in an existing filter
#      4          Flush all filter entries for address
#
#Admissible values for the Target (TG) field are as follows:
#      Value      Filter Target
#      0          Drop incoming packets without notification
#      1          Reject incoming packets with notification
#      2          Accept incoming packets
#      3          Accept incoming packets but avoid route
#                 optimization
#      4          Masquerade
#

#Number of filters
#-----------------
 7

#     Seq   Interface   Action      Target      Filter Modules
#     ---   ---------   ------      ------      --------------
       0    eth0        1     2     2

#           Type  DSCP Entry(6 bit value)
#           ----  ------------------------
             67   22

#           Type  Protocol Number
#           ----  ---------------
             68   4

#     Seq   Interface   Action      Target      Filter Modules
#     ---   ---------   ------      ------      --------------
       1    eth0        1     2     1
```

```
#           Type  Source IP Addresse
#           ----  ------------------
             69   134.102.158.1


#     Seq   Interface   Action      Target      Filter Modules
#     ---   ---------   ------      ------      --------------
      2     eth0        1     2     1

#           Type  Port Number
#           ----  ------------
             71   8819

#
#     Seq   Interface   Action      Target      Filter Modules
#     ---   ---------   ------      ------      --------------
       3    eth1        1     2     1

#           Type  Source IP Addresse & Source IP Address Mask
#           ----  ------------------------------------------------
             70   134.102.158.65          255.255.255.248

#     Seq   Interface   Action      Target      Filter Modules
#     ---   ---------   ------      ------      --------------
       4    eth1        1     2     1

#           Type  Port Number
#           ----  -----------
             71   8081

#     Seq   Interface   Action      Target      Filter Modules
#     ---   ---------   ------      ------      --------------
       5    eth1        1     2     1

#           Type  Source Port Number Range(Min - Max)
#           ----  ----------------------------------
             72   9002  9004

#     Seq   Interface   Action      Target      Filter Modules
#     ---   ---------   ------      ------      --------------
       6    eth1        1     2     1

#           Type  Offset      Value & Mask (value & mask should be in hex)
#           ----  ------      ----------------------------------------
             73   8     ABCDEF      FFFFFF
#
#-------------------------------------------------------------------
```