Dual Stack Implementation of Mobile IPv6
Software Architecture

Dr. K.L.Bansal
Associate Professor
Department of Computer Science
H.P.University, Shimla (H.P.) India-171005

Chaman Singh
Research Scholar
Department of Computer Science
H.P.University, Shimla (H.P.) India-171005

ABSTRACT
IPv6 is introduced mainly to resolve the address space issues and also provides several advanced features. IPv6 is estimated to replace IPv4 in a very near future. Dual Stack Mobile IPv6 (DSMIPv6) is an extension of Mobile IPv6 to support mobility of devices irrespective of IPv4 and IPv6 network. This paper provides an architectural overview of the existing DSMIPv6 implementation and software architecture to understand the significant modifications which have been made on DSMIPv6 basic implementation to achieve the requirements. The scope of the paper is to implement the Dual-stack Mobile IPv6 (DSMIPv6) protocol as per the IETF (Internet Engineering Task force) draft. The entities which have been implemented are ‘DSMIPv6 Home Agent’ and ‘DSMIPv6 Mobile Node’. The paper covers overview of NEPL (Network Mobility platform for Linux) and DSMIPv6 implementation and briefly describes the features supported by DSMIPv6 architecture. It also focuses on our Solution Approach and explains the high level view of modules used in DSMIPv6 using a block diagram schematic.

General Terms
Implementation MIPv6

Keywords
Dual Stack, IPv4, IPv6, MIPv6

1. INTRODUCTION
The application interface is required to exchange mobility information with Mobility Subsystem [1]. Mobile IPv6 (MIPv6) is a protocol developed as a subset of Internet Protocol version 6 (IPv6) [2] to support mobile connections. MIPv6 [3] allows a mobile node to transparently maintain connections while moving from one subnet to another. The Mobile IPv6 protocol takes care of binding addresses between Home Agent (HA) and Mobile Node (MN). It also ensures that the Mobile Node is always reachable through Home Agent. Each mobile node is always identified by its home address [4], regardless of its current point of attachment to the Internet. While situated away from its home, a mobile node is also associated with a care-of address, which provides information about its current point of attachment to the Internet. The protocol provides for registering the care-of address with a home agent. The home agent sends datagram’s destined for the mobile node through a tunnel to the care-of address. After arriving at the end of the tunnel, each datagram is then delivered to the mobile node. Currently, two mobility management protocols are defined for IPv4 and IPv6. Deploying both in a dual stack mobile node introduces a number of problems. This has been improved [5]. Mobile IPv6 uses IPSec (IP Security) to protect signaling between the home agent and the mobile node [6]. Generic Packet Tunneling [7] specifies a method and generic mechanisms by which a packet is encapsulated and carried as payload within an IPv6 packet. The resulting packet is called an IPv6 tunnel packet. The forwarding path between the source and destination of the tunnel packet is called an IPv6 tunnel. The technique is called IPv6 tunneling. A typical scenario for IPv6 tunneling is the case in which an intermediate node exerts explicit routing control by specifying particular forwarding paths for selected packets. This control is achieved by pre-pending IPv6 headers to each of the selected original packets. The current Mobile IPv6 [3] and Network Mobility [8] specifications support IPv6 only. These extend those standards to allow the registration of IPv4 addresses and prefixes, respectively, and the transport of both IPv4 and IPv6 packets over the tunnel to the Home Agent. [9] Allows the Mobile Node to roam over both IPv6 and IPv4, including the case where Network Address Translation is present on the path between the mobile node and its home agent.

2. ARCHITECTURAL REPRESENTATION OF DSIMIPv6
NEPL (NEMO Platform for Linux) [10] is a freely available implementation of DSMIPv6 for Linux platform. The original NEPL release was based on MIP6 (Mobile IPv6 for Linux) [11]. In Figure-1: Basic Operation of DSMIPv6, all Mobile Nodes (MN) has a fixed address, called a Home Address (HoA), assigned by Home Agent. When the Mobile Node moves to other networks, it gets Care-of Address (CoA) from foreign network. Mobile Node sends a Binding Update (BU) message to its Home Agent. Then Home Agent replies to the Mobile Node with a Binding Acknowledgement (BA) message to confirm the request. When Mobile Node is moved to any foreign network all packets sent to the Home Agent will be IPSec encrypted. A bi-directional tunnel is established between the Home Agent and the Care of address of the Mobile Node after the binding information has been successfully exchanged.
DSMIPv6 extends the MIPv6 and NEMO [12] Basic Support standards to allow mobile nodes to roam in both IPv6 and IPv4-only networks. The following features are supported by the DSMIPv6 Architecture.

1. The mobile node can register an IPv6/IPv4 Care of address to its Home Agent and thus roam in IPv6-only networks and IPv4-only networks by the use of IPv6 tunnels and IPv6-in-IPv4 tunnels between the Mobile Node and its Home Agent.

2. A Network Address Translation Detection and Traversal Mechanism allow the Mobile Node to communicate with its Home Agent even though it uses an IPv4 private address as a Care of address. The signaling messages are always UDP encapsulated in IPv4 network. However, when the Mobile Node is located behind a NAT, data traffic is also encapsulated in UDP.

3. Securing the signaling packets between Home Agent and Mobile Node when Mobile Node is moved to foreign network.

4. Session management on movement from one foreign link to another.

2.1 Solution Description

The solution is an extension to the existing NEPL solution provided by Nautilus [10]. We validated the DSMIPv6 functionality as per the requirements provided against the draft, along with other IETF standards. We took the baseline architecture implementation from the Nautilus6 which uses Linux platform. The below mentioned steps are taken by us to achieve the requirements:

1. Have setup DSMIPv6 Test Lab using Kernel 2.6.28.2 and UMIP veMyon 0.4. In order to test the basic functionality between Home Agent and Mobile Node.
2. Code changes have done in mip6d daemon and Linux kernel and also applied the open source patches/packages on Test Lab to meet the requirements.

3. The Routing Advertisement daemon (radvd), IPSec daemon (strongswan) and Web Server (httpd) daemon has been configured on Home Agent.

4. The Mobile Node is configured with IPSec daemon (strongswan). Mobile Node gets IPv6 address whenever it is moved to any IPv6 foreign network through the radvd server running on the router.

5. When Mobile Node is moved to IPv4 network, it gets configured with IPv4 Care of address from the DHCP server running on IPv4 Router.

6. In IPv4 network, DHCP is configured on the private network behind router. The network behind IPv4 router can be public or private.

2.2 Block Diagram of Module Representation in DSMIPv6

MIPL (Mobile IPv6 for Linux) is an open-source implementation of the Mobile IPv6 standard for the GNU/Linux operating system. MIPv6 is a user space for Mobile Node and Home Agent which aims at providing the necessary changes to MIPL in order to run on the latest kernels. Figure-2: Block Diagram of MIPL shows the internal data flow between two major components i.e. Home Agent and Mobile Node. Both of these two components consist of several helper modules which are also shown in this figure.

2.3 Module Description

2.3.1 DNA/DHCP Module

This section describes IPv4 address assignment mechanism used by DSMIPv6. DHCP DNA module is used to obtain IPv4 address from the DHCP server running on IPv4 network.
2.3.1.1 Process Description
When Mobile Node moves to IPv4 FL (Foreign Link) and its egress interface becomes enabled, Mip6d code in Mobile Node listens for Router Advertisement message, and since it does not receives Router Advertisement message in IPv4 FL, it gets timeout and sends Router Solicitation message (that will request the router to generate the Router Advertisement message immediately rather than at there next scheduled time), and Mobile Node wait for some time interval for Router Advertisement message before repeating the same procedure of sending Router Solicitation message. Meanwhile after sending Router Solicitation message, mipv6d daemon will check the presence of DHCP server on the Egress interface link of Mobile Node by sending the DHCP discover message and wait for DHCP offer packet. Since the DHCP server is running on the IPv4 FL, it gets the IPv4 address from DHCP server and then mipv6d code maps IPv4 address to IPv6 address, which is further used as Care of address. Mipv6 daemon sets the default route on Mobile Node fig3.

**Fig 3. Process flow in IPv4 Only Network.**

2.3.1.2 Data Structure
Struct dhcp_dna_control contains the DHCP client state machine, lease time and timeout information. File mipv6-daemon-unip-0.4/src/dhcp_dna.h is used.s and second Struct dhcp_message contains the information about DHCP messages send to and received from the server. File mipv6-daemon-unip-0.4/src/dhcp_dna.c is used. When Mobile Node moves to IPv4 only network or dual stack network then only this DHCP DNA module comes into the picture. Mobile Node first tries to acquire IPv6 Care of address and failure in IPv6 address configuration triggers the DHCP DNA code which sends dhcp discover messages on the network to acquire the IPv4 Care of address. The table-1 show the various methods used.

2.3.2 Movement Detection Module
This section describes Movement Detection module in DSMIPv6 implementation. The movement of a mobile node away from its home link is transparent to transport and higher-layer protocols and applications.
Table 1. DNA/DHCP Methods.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Input Parameter</th>
<th>Return Value</th>
<th>Caller</th>
<th>Define in file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send_discover</td>
<td>Broadcast a DHCP discover packet to the network with an optionally requested IP</td>
<td>dhcpCtrl: DHCP information bng; xid: Client ID</td>
<td>In case of error return value&lt;1</td>
<td>DHCP_listen</td>
<td>DHCP_DNA.C</td>
</tr>
<tr>
<td>Send_select</td>
<td>Broadcast a DHCP request packet to the network</td>
<td>Server: SERVER ID &amp; same as in Function 1</td>
<td>In case of error return value&lt;1</td>
<td>DHCP_listen</td>
<td>DHCP_DNA.C</td>
</tr>
<tr>
<td>Send_renew</td>
<td>Broadcast a DHCP renew request packet on the network</td>
<td>Server: SERVER ID &amp; same as in Function 1</td>
<td>In error return value&lt;1</td>
<td>DHCP_listener</td>
<td>DHCP_DNA.C</td>
</tr>
<tr>
<td>Send_release</td>
<td>Uncasts a DHCP release message</td>
<td>Server: SERVER ID &amp; same as in Function 1</td>
<td>In case of error return value&lt;1</td>
<td>DHCP_listen</td>
<td>DHCP_DNA.C</td>
</tr>
<tr>
<td>Dhcp_dns_init</td>
<td>Starts the dhcp client state machine</td>
<td>none</td>
<td>In case of error return value&lt;1</td>
<td>main</td>
<td>DHCP_DNA.C</td>
</tr>
<tr>
<td>Dhcp_listen</td>
<td>contains dhcp client state machine logic</td>
<td>args: argument passed to the function</td>
<td>In case of error return value&lt;1</td>
<td>Dhcp_dns_init</td>
<td>DHCP_DNA.C</td>
</tr>
<tr>
<td>Get_packet</td>
<td>Read a packet from socket fd</td>
<td>dhcp_message: dhcp_message received_fd: socket</td>
<td>In case of error return value&lt;1</td>
<td>Dhcp_dns_init</td>
<td>DHCP_DNA.C</td>
</tr>
</tbody>
</table>

The Movement detection uses Neighbor Unreachability Detection [13] to detect when the default router is no longer bi-directionally reachable, in which case the mobile node must discover a new default router (usually on a new link). However, this detection only occurs when the mobile node has packets to send. When the mobile node detects handover, it expire the previous routers and Care-of-Address (es) and selects a new default router as a consequence of Router Discovery, and then performs Prefix Discovery with that new router to form new care-of address (es). This is handled in 'movement.c'. It then registers its new primary care-of address with its home agent. After updating its home registration, the mobile node then updates associated mobility bindings in correspondent nodes. It triggers a movement event and then detects that Mobile Node is in foreign network. Route is modified and Home Address which was previously assigned to the physical interface is now moved to the tunnels. This is handled in 'main.c'.

2.3.3 Binding Management Module

When a Mobile Node moves between different networks, it is essential that binding update messages are sent to that node’s Home Agent and Correspondent Nodes as soon as possible, in order to facilitate a fast handoff. Mobile Nodes therefore cannot rely on the soft state timeout mechanism used in binding caches to refresh stale bindings maintained by Correspondent Nodes (typical binding lifetimes are of the order of minutes). An additional data structure, the binding update list, is therefore kept by Mobile Nodes, which maintains state on any Correspondent Nodes or Home Agents. And in Home Agent mobile IPv6 binding cache maps home addresses to the current care-of addresses for each mobile node. This allows the home agent to tunnel traffic to the mobile node at its current location, and allows a correspondent node to send packets directly to a mobile node at its current location. Binding Management Module is classified as:

- Binding Update List
- Binding Cache

**Binding Update**: When Mobile Node moves to any Foreign Link, Mip6 code in Mobile Node initiate binding update functionality after configuring Care of address either from Router Advertisement message received in IPv6 or Dual stack FL or from DHCP server in IPv4 FL on the egress interface of Mobile Node. After Mobile Node configures Care of address on its egress interface, it creates BUL which consists of information about Mobile Node's Home Address, Care of address, CN address, life time and delay time of this binding message in seconds and set of various flag.

**Binding Update List format is as follows:**

```plaintext
== BUL_ENTRY ==
Home address 2001: x: x: x: x: x: x: x
Care-of addresses 2001: x: x: x: x: x: x: x: x
CN address 2001: x: x: x: x: x: x: x
Lifetime = 32, delay = 1500
Flags: IP6_MH_BU_HOME, IP6_MH_BU_ACK, IP6_MH_BU_TLV
```
When Mip6d code in Mobile Node creates BUL successfully, it send Binding Update message to Home Agent to register the new Care of address of Mobile Node to Home Agent. So that Home Agent gets updated with current Care of address of Mobile Node.

**Binding Cache:** Mip6d code in Home Agent received Binding Update message sent by Mobile Node when it moves to any Foreign Link or when it directly boots in IPv6/IPv4 FL. Home Agent process BU by performing DAD (dynamic address discovery) [14] and later parse Binding Update message. After parsing the binding update message Home Agent creates or updates its Binding Cache entry. Binding Cache entry consists of Mobile Node's Home Address, Mobile Node's Care of address, its local address, lifetime and sequence no.

Binding Cache entry format is as follows:

```
== BC_ENTRY ==
HoA 2001:::1 status registered
CoA 2001:::1:80e:29ff:fea0:4026 flags AH- Local 2001:::1:80e:29ff:fea0:4026
Lifetime 23 /32 seq 3435
Unreach 0/95299 retry -2
```

After updating its Binding Cache entry, the mip6d code creates devices to tunnel traffic to Mobile Node. Mip6d code in Home Agent creates and send Binding Acknowledgment message to Mobile Node so that Mobile Node gets acknowledged that it's new Care of address gets successfully registers with Home Agent. When Mobile Node receives Binding Acknowledgment message, mip6d code in Mobile Node parses the BA packet and update the BUL entry. It checks for various options set in BA and proceed accordingly. If NAT is detected between Mobile Node and Home Agent, it set xfrm policies/states to UDP Encapsulate IPv6/IPv4 data traffic to bypass NAT. Mip6d daemon sets the callback function to resend the BA, once lifetime of BUL entry is expired. And finally set the binding update timer to decrease the lifetime of BUL entry. When Mobile Node moves back from any FL to Home Link, Mip6d code in Mobile Node sends Binding Update message with lifetime set as zero to Home Agent to indicate that it as returned to Home Link mip6d code in Mobile Node deletes corresponding BUL entry. On Home Agent side Mip6d code receives BU message with lifetime set as zero, it indicate that Mobile Node moved to Home Link, so it deletes corresponding Binding Cache entry and send Binding Acknowledgment back to Mobile Node.

### 2.3.4 Tunnel and Route Management Module

Tunnel and Route Management module is mainly responsible for tunneling when mobile node changes from IPv6 to IPv4, IPv4 to IPv6 and vice versa network. This module configures sit and ip6tnl interface via IODCT system call which in turns performs the task at kernel level. Route Management handles the return routability with CN. Some data structures are being used between some of the important functions in Tunnel Management module. Some user land data structures used in various routines in tunnel management module. Some data structures used in various routines in tunnel management module.

#### 2.3.5 XFRM and IPSec Module

XFRM [15] is a packet transformation framework residing in the Linux kernel. It performs operations on IP packets such as inserting, modifying headers, UDP encapsulation and de-capsulation. DSMIPv6 XFRM module will take the advantage of existing IPsec transformation and defines a simple UDP encapsulation scheme. IPsec module is responsible for interaction with IKE through MIGRATE messages. IPsec will be used to protect the following traffic between Home Agent and Mobile Node:

1. BU/BA messages.
2. Mobile prefix solicitation and advertisement messages.
3. Normal traffic between Mobile Node and Home Agent.
4. All tunnelled normal traffic between Mobile Node and correspondent Node.

#### 2.3.5.1 Module name and Functionality

In Mip6d, the Mobile Node (MN) and the Home Agent (HA) uses IPsec Security Associations (SAs) in transport mode to protect BU/BA messages, since the MN may change its attachment point to the Internet, it is necessary to update its endpoint address of the IPsec SAs. This indicates that corresponding entry in IPsec databases (Security Policy (SPD) and SA (SAD) databases) should be updated when Mobile Node performs movements. IPsec is used to protect the following traffic between Home Agent and Mobile Node:

1. **BU/BA messages**

**Process Description** (IPSec Protection for BU/BA) When Mobile Node move in FL a new Care of address is assigned to the Mobile Node by FL network. After detecting the movement following steps are taken to create IPsec tunnel:

1. Mip6d issues a PF_KEY MIGRATE message to the PF_KEY socket.
2. The operating system validates the message and checks if corresponding security policy entry exists in SPD.
3. When the message is confirmed to be valid, the target SPD entry is updated according to the MIGRATE message. If there is any target SA found that are also target of the update, those should also be updated.
4. After the MIGRATE message is successfully processed inside the kernel, it will be sent to all open PF_KEY sockets. The IKE daemon receives the MIGRATE message from its PF_KEY socket and updates its SPD.
2.3.6 NAT Detection and Traversal Module

NAT (Network Address Translation) is the translation of an Internet Protocol address (IP address) used within one network to a different IP address known within another network. One network is designated the inside network and the other is the outside. In DSMIPv6 the mip6 daemon should bypass NAT, when Mobile Node is behind NAT' ed device in IPv4 PL. NAT detection is done when the initial binding update message is sent from the mobile node to the home agent. When located in an IPv4-only foreign link, the mobile node sends the Binding Update message encapsulated in UDP (User Datagram Protocol) and IPv4; this is handled in xfrm.c file. The mip6 daemon adds xfrm policy/state for UDP encapsulation for BU packet. When the home agent receives the encapsulated Binding Update, it compares the IPv4 address of the source address field in the IPv4 header with the IPv4 address included in the IPv4 care-of address option. If the two addresses match, no NAT device is in the path. Otherwise, a NAT is detected in the path and the NAT detection option is included in the Binding Acknowledgment. The Binding Acknowledgment, and all future packets, is then encapsulated in UDP and IPv4. Note that the home agent also stores the port numbers and associates them with the mobile node's tunnel in order to forward future packets. This is handled in ha.c file. The mip6 daemon adds the xfrm policies/statess for UDP encapsulation of BA and IPv6/IPv4 data traffic. Upon receiving the Binding Acknowledgment with the NAT detection option, the mobile node sets the tunnel to the home agent for UDP encapsulation. Hence, all future packets to the home agent are tunneled in UDP and IPv4. If no NAT device is detected in the path between the mobile node and the home agent then IPv4/IPv6 data traffic is not UDP encapsulated. A mobile node will always tunnel the Binding Updates in UDP when located in an IPv4-only network. Essentially, this process allows for perpetual NAT detection. Similarly, the home agent will encapsulate Binding Acknowledgements in a UDP header whenever the Binding Update is encapsulated in UDP. This is handled in mn.c and xfrm.c file. The mip6 daemon adds xfrm policies/statess for UDP encapsulation of IPv6/IPv4 data traffic, when NAT is detected between Mobile Node and Home Agent.

2.3.7 Mobility Listener Module

The Mobility Header is an extension header used by mobile nodes, correspondent nodes, and home agents in all messaging related to the creation and management of bindings. The Mobility Header is identified by a Next Header value of 135 in the immediately preceding header. This Header is used to carry the following messages:

**Home Test Init**

A mobile node uses the Home Test Init (HoTI) message to initiate the return routability procedure and request a home keygen token from a correspondent node. The Home Test Init message uses the MH Type value 1.

**Home Test**

The Home Test (HoT) message is a response to the Home Test Init message, and is sent from the correspondent node to the mobile node. The Home Test message uses the MH Type value 3.

**Care-of Test Init**

A mobile node uses the Care-of Test Init (CoTI) message to initiate the return routability procedure and request a care-of
keygen token from a correspondent node. The Care-of Test Init message uses the MH Type value 2.

Care-of Test
The Care-of Test (CoT) message is a response to the Care-of Test Init message, and is sent from the correspondent node to the mobile node. The Care-of Test message uses the MH Type value 4.

Binding Update
The Binding Update (BU) message is used by a mobile node to notify other nodes of a new care-of address for itself. The Binding Update uses the MH Type value 5.

Binding Acknowledgement
The Binding Acknowledgement is used to acknowledge receipt of a Binding Update. The Binding Acknowledgement has the MH Type value 6.

Binding Refresh Request
The Binding Refresh Request (BRR) message requests a mobile node to update its mobility binding. This message is sent by correspondent nodes. The Binding Refresh Request message uses the MH Type value 0.

Binding Error
The Binding Error (BE) message is used by the correspondent node to signal an error related to mobility, such as an inappropriate attempt to use the Home Address destination option without an existing binding. The Binding Error message uses the MH Type value 7.

3. ASSUMPTION AND LIMITATIONS
1) Dynamic IPv4 Home Address assignment to Mobile Node using IKEv2.
2) Mobile Node with multiple tunnel interfaces.
3) The reqid (request ID's) defined in the IPsecPolicy lines of the mip6d.conf files currently must exactly match the reqid assigned by strongswan to the corresponding IPSec SA. Strongswan does the assignment using a linear counter starting with reqid 1. Otherwise the communication between the mip6d and strongswan daemons via MIGRATES and ACQUIRE kernel messages is simply not going to work. Thus make sure that Mobile Node-Home Agent connections are started in the correct order, i.e. in our example first the connection from Mobile Node carol ((request ID's) 1 and 2) and only after that the connection from Mobile Node dave ((request ID's) 3 and 4).
4) Home Agent behind NAT
5) DHAAD (Dynamic Home Agent Address Detection)

4. CONCLUSION
The paper represents the software architecture of dual stack implementation of mobile IPv6. This allows for IP address translation (NAT) and its detection and traversal on the co-stack implementation of Mobile IPv6. NAT (Network Address Translation) is the translation of an Internet Protocol address (IP address) used within one network to a different IP address known within another network. One network is designated the inside network and the other is the outside. In DSMIPv6 the mip6d daemon should bypass NAT, when Mobile Node is behind NAT device in IPv4 Foreign Link.

5. REFERENCES
ABSTRACT
IPv4 private networks are behind NAT devices. So, to bypass the Binding Update and Binding Acknowledgment by NAT, we need to encapsulate it in UDP packets. So, the dual stack mobile IPv6 should support NAT traversal and Detection. Dual Stack Mobile IPv6 (DSMIPv6) is an extension of MIPv6 to support mobility of devices irrespective of IPv4 and IPv6 network. Current IP networks are predominantly based on IPv4 technology, and hence various firewalls as well as Network Address Translators (NATs) have been originally designed for these networks. Deployment of IPv6 is currently work in progress. This research provides an overview of network address translation (NAT) and its detection and traversal on dual stack implementation on Mobile IPv6. In DSMIPv6 the MIP6D daemon should bypass NAT, when Mobile Node is behind NAT device in IPv4 Foreign Link.

General Terms
Networks

Keywords
NAT, Traversal, Detection, Dual Stack, MIPv6

2. NAT TRAVERSAL AND DETECTION MODULE

2.1 Module Name & Functionality
NAT (Network Address Translation) is the translation of an Internet Protocol address (IP address) used within one network to a different IP address known within another network. One network is designated the inside network and the other is the outside. In DSMIPv6 the mip6d daemon should bypass NAT, when Mobile Node is behind NAT device in IPv4 Foreign Link.

2.2 Files used
2.3 Process Description
NAT detection is done when the initial Binding Update message is sent from the mobile node to the home agent. When located in an IPv4-only foreign link, the mobile node sends the Binding Update message encapsulated in UDP and IPv4, this is handled in xfrm.c file. The mip6d daemon adds xfrm policy/state for UDP encapsulation for BU packet. When the home agent receives the encapsulated Binding Update, it compares the IPv4 address of the source address field in the IPv4 header with the IPv4 address included in the IPv4 care-of address option. If the two addresses match, no NAT device is in the path. Otherwise, a NAT is detected in the path and the NAT detection option is included in the Binding Acknowledgement. The Binding Acknowledgement, and all future packets, is then encapsulated in UDP and IPv4. Note that the home agent also stores the port numbers and associates them with the mobile node’s tunnel in order to forward future packets. This is handled in ha.c file. The mip6d daemon adds the xfrm policies/states for UDP encapsulation of BA and IPv6/IPv4 data traffic. Upon receiving the Binding Acknowledgement with the NAT detection option, the mobile node sets the tunnel to the home agent for UDP encapsulation. Hence, all future packets to the home agent are tunneled in UDP and IPv4. If no NAT device is detected in the path between the mobile node and the home agent then IPv4/IPv6 data traffic is not UDP encapsulated. A mobile node will always tunnel the Binding Updates in UDP when located in an IPv4-only network. Essentially, this process allows for perpetual NAT detection. Similarly, the home agent will encapsulate Binding Acknowledgements in a UDP header whenever the Binding Update is encapsulated in UDP. This is handled in in.c and xfrm.c file. The mip6d daemon adds xfrm policies/states for UDP encapsulation of IPv6/IPv4 data traffic, when NAT is detected between Mobile Node and Home Agent.

2.4 Flow Chart
NAT Detection and Traversal flow in Mobile Node given in figure 2, 3. And NAT Detection and Traversal flow in Home Agent is given in figure 4, 5.

2.5 Internal Data Structure
Following are the main structures that are being used between some of the important functions in NAT Traversal and Detection.

1. struct encap_info:
Description: Structure to store the source IP address and port information, once NAT is detected.
File: mipv6-daemon-umip-0.4/src/nat.h
Code Snippet:
```c
struct encap_info {
    struct inaddr src;
    uint16_t port;
};
```

2. Enum for NAT detection:
Description: Enumeration used for NAT detection.
File: mipv6-daemon-umip-0.4/src/nat.h
Code Snippet:
```c
typedef enum {
    MIP6_NAT_DISABLED,
    MIP6_NAT_ENABLED,
}...
```

3. struct xfrm_selector:
Description: Xfrm selectors for policy and state used for UDP Encapsulation.
File: linux-2.6.28.2/include/linux/xfrm.h
Code Snippet:
```c
struct xfrm_selector {...
    xfrm_address_t daddr;
    xfrm_address_t addr;
    bc16 dport;
    __bc16 dport_mask;
    __bc16 sport;
    __bc16 sport_mask;
    __u16 family;
};
```
Outbound Packet Flow

Start

Movement of MN to FL (movement.c)

IPv4 / FL

IPv4 or IPv6? (mn.c)

Check flags for sig and data traffic for UDP encapsulation (xfrm.c)

No

Do Nothing

Delete outbound and inbound xfrm policy/states for UDP Encapsulation of BU and IPv6/IPv4 data traffic (xfrm.c)

Yes

Delete the flags for sig and IPv6/IPv4 data traffic (mn.c)

IPv6 / HL

End

Fig. 2: Outbound Packet Flow in Mobile Node.

Code Snippet:

```
struct xfrm_user tmpl {
    struct xfrm_id id;
    _u16 family;
    _xfrm_address_t _saddr;
    _u32 reqid;
    _u8 mode;
    _u8 _share;
    _u8 _optional;
    _u32 aalgos;
    _u32 _calgos;
    _u32 _calgos;
}
```

4. struct xfrm_user tmpl:

Description: Used to create template for xfrm policy for UDP encapsulation.

File: linux-2.6.28.2/include/linux/xfiTn.h

Code Snippet:

```
struct xfrm_user tmpl {
    struct xfrm_id id;
    _u16 family;
    _xfrm_address_t _saddr;
    _u32 reqid;
    _u8 mode;
    _u8 _share;
    _u8 _optional;
    _u32 aalgos;
    _u32 _calgos;
    _u32 _calgos;
}
```

5. struct xfrm_encap tmpl:

Description: Used to create template for xfrm policy for UDP encapsulation.

File: linux-2.6.28.2/include/linux/xfiTn.h

```
struct xfrm_encap tmpl {
    _u16 encap_type;
    _bel6 encap_sport;
    _bel6 encap_dport;
    xfrm_address_t encap OA;
}
```

6. struct bulentry:

Description:

This structure stores information about Binding Update List. The members of this structure are used to set Xfim policy/states for UDP encapsulation in Mobile Node side.

File: mipv6-daemon-umip-0.4/src/bul.h

Code Snippet:

```
struct bulentry {
    struct home_addr_info *home; /* Pointer to home_address structure to which this entry belongs to */
    struct bul_elem telc;
    /* Timer queue entry */
    struct in6_addr peer_addr; /* CN / HA address */
    struct in6_addr home; struct in6_addr coa; /* care-of address of the sent BU */
    _int if Coa; _int if_tunnel; /* Tunnel iface for the BCE */
    _int _if tunnel4; /* 4/4 or 6/4 tunnel iface for the BCE */
    _int type; /* BUL / NON_MIP_CN / UNREACH */
    _uint16_t seq; /* sequence number of the latest BU */
    _uint16_t flags; /* BU send flags */
    struct in6_addr last Coa; /* Last good coa */
```
Inbound Packet Flow

Start

1. Wait for BA Packet from HA (mn.c)
2. UDP Encapsulation BA Packet received on MN side

MN behind NAT? (mn.c)

Yes

MN is behind NAT Device

Add outbound and inbound xfrm policies/states for UDP encapsulation of IPv6/IPv4 data Packet (mn.c)

Set flag for IPv6/IPv4 data traffic (mn.c)

Future IPv6/IPv4 data traffic will be UDP Encapsulated

No

MN is not behind NAT Device. So no need to add xfrm policy/state for UDP encapsulation of IPv4/IPv6 data traffic

Data Packets are encapsulated in IPv6 in IPv4 or IPv4 in IPv4 Tunneling

End

Fig 3. Inbound Packet in Mobile Node.
Information for NAT traversal

whether a nat was detected

sequence number of the latest BU

BU Receive Packet Flow

Start

Add xfrm policy/state to accept incoming BU Packet (xfrm.c)

UDP Encapsulation BU received on HA side

NAT Detection?

Yes

Store the Port Number in Order to forward future packets (ha.c)

Add the Inbound/Outbound xfrm policy/state for UDP encapsulation of BA and IPv6/IPv4 data traffic (ha.c)

Set flags for UDP encapsulation of BA and IPv6/IPv4 data traffic (ha.c)

Add NAT Detection Option in BA Packet

All future traffic will be UDP encapsulated

End

No

No need to store Port Numbers

Add xfrm policy/state to UDP encapsulation BA Traffic (ha.c)

Set flag for BA signaling (ha.c)

Only BA will be UDP encapsulated and no UDP encapsulation of data traffic

Fig 4. BU Receive Packet flow in Home Agent.
3. INTERNAL METHODS

Following table describe the main functions used in MIPv6d for implementation of NAT Traversal and Detection module. Various functions description, input parameter, returns value, which call them and the file in which they are stored are given in table 1.2.

4. EVENT TRIGGERING THE PROCESS

When MN moves in IPv4 FL, mip6d code adds xfrm policy/state for UDP encapsulation of BU and sends BU to Home Agent. This processing is handled by routine xfrm_pre_bu_add bule. When BU is received on Home Agent side, it triggers the NAT detection called. And if NAT is detected, it pushes xfrm policies/statess for UDP encapsulation of IPv6/IPv4 data traffic and BA in the kernel; otherwise if NAT is not detected it only pushes Policy/state for UDP encapsulation of BA. This processing is handled by routine ha_recv_bu_worker, which further calls routine udpencap_encap_out_traffic_start for adding UDP encapsulation of IPv6 data traffic and routine udpencap_encap_out_IPv4_traffic_start for UDP encapsulation of IPv4 data traffic. When UDP encapsulated BA is received on Mobile Node side. The mip6d checks the presence of NAT between Mobile Node and Home Agent. If NAT is detected, it pushes xfrm policies/statess in kernel for UDP encapsulation of future IPv4/IPv6 data traffic. This processing is handled by routine mn_recv_ba, which further call routine udpencap_encap_out_traffic_start for adding UDP encapsulation of IPv6 data traffic and routine udpencap_encap_out_IPv4_traffic_start for UDP encapsulation of IPv4 data traffic. On Mobile Node side to flush the xfrm policies and states for UDP encapsulation, the routine called is xfrm_del_bule dsmip, which further calls routine udpencap_encap_out_traffic_end to flush policies/statess for BU and IPv6 data traffic and routine udpencap_encap_out_IPv4_traffic_end to flush xfrm policies and states for UDP encapsulated of IPv4 data traffic. On Home Agent side to flush the xfrm policies and states for...
UDP encapsulation, the routine called is `ha_udpencap_encap_traffic_end`, which further calls routine `udpencap_encap_out_traffic_end` to flush policies/states for BU and IPv6 data traffic, and routine `udpencap_encap_out_IPv4_traffic_end` to flush xfrm policies and states for UDP encapsulated of IPv4 data traffic.

### Table 1. Internal Methods for NAT Detection and Traversal

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Input Parameter</th>
<th>Return Value</th>
<th>Caller</th>
<th>file</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xfrm_del_bule</code></td>
<td>Routine to delete states and policies related to UDP encapsulation for the BULE</td>
<td>bule: BUL structure</td>
<td></td>
<td>xfrm_del_bule</td>
<td>Xfrm.c</td>
</tr>
<tr>
<td><code>pre_bu_bul_update</code></td>
<td>This routine is called before sending BU; MN should insert UDP encapsulation policy/state only for BU/BA</td>
<td>bule: BUL structure</td>
<td></td>
<td>pre_bu_bul_update</td>
<td>Xfrm.c</td>
</tr>
<tr>
<td><code>ha_recv_bu_worker</code></td>
<td>This routine is handler of type <code>mh_handler</code>. It is called when BA is received on MN side by HA</td>
<td>mh: Header len: mh Length in:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>in6_addr_bundle</code> structure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>iif</code>: interface index</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>home_cleanup</code></td>
<td>to delete policy/state for udp encapsulation of BA, IPv6 and IPv4 data packets. When BU is UDP Encapsulated, BA is also UDP encapsulated. If NAT is detected, all future IPv6/IPv4 data traffic is also UDP encapsulated.</td>
<td>bce: binding cache</td>
<td>In case of error return integer value less than 0</td>
<td>home_cleanup</td>
<td>Ha.c</td>
</tr>
<tr>
<td><code>ha_receiv_bu_worker</code></td>
<td>to add policy/state for UDP encapsulation of BA, IPv6 and IPv4 data packets When BU is UDP Encapsulated, BA is also UDP encapsulated. If NAT is detected, all future IPv6/IPv4 data traffic is also UDP encapsulated.</td>
<td>bce: binding cache</td>
<td>In case of error return integer value less than 0</td>
<td>ha_receiv_bu_worker</td>
<td>Ha.c</td>
</tr>
<tr>
<td><code>udpencap_encap_traffic_start</code></td>
<td>to add UDP encapsulated xfrm policy/state for IPv4 data traffic, when NAT is detected</td>
<td>Same as that of Below function</td>
<td>In case of error return integer value less than 0</td>
<td>ha_udpencap_encap_traffic_start</td>
<td>Xfrm.c</td>
</tr>
</tbody>
</table>

### 5. CONCLUSION

This paper is one of the earliest attempts in the community to investigate the problems and impacts when middleboxes, especially NAT devices are placed in Dual stack Mobile IPv6are implemented in computer laboratory. With the support of NAT Detection and Traversal module in DSMIPv6, the mobile node is able to move freely from IPv6 network to IPv4 network or vice-versa. It accomplishes the main objective of not breaking the connectivity at the time of switching from one network to other. Now we are going to implement the following feature like Security considerations related to IPv6 with IPSEC and IKEv2.
Handover interactions for IPSec and IKE/IKE negotiations between Mobile Node and Home Agent and IKEv2 operation for securing DSMIPv6 signaling (BU & BA). The transition from IPv4 to IPv6 will be time consuming process, so there will be time, when both IPv4 and IPv6 networks will be there and there will be always being scope for further development.

Table 2. Internal Methods for NAT Detection and Traversal

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Input Parameter (in case of error return&lt;0)</th>
<th>Caller</th>
<th>file</th>
</tr>
</thead>
<tbody>
<tr>
<td>ha_udpencap_encap_traffic_end</td>
<td>to delete UDP encapsulated xfrm policy/state for IPv4 data traffic, when NAT is detected</td>
<td>local: IPv6 local address,lpreflen: prefix length of local address dest: IPv6 peer address,dpreflen: prefix length of peer address,proto: Protocol,type: MH header type,src: IPv4 local address,dst: IPv4 peer address,direction: Security parameter index</td>
<td>ha_udpencap_encap_traffic_end</td>
<td>xfmrmc</td>
</tr>
<tr>
<td>xfrmcninit</td>
<td>xfrmcninit</td>
<td>xfrmcninit</td>
<td>xfrmcninit</td>
<td>xfrmcninit</td>
</tr>
</tbody>
</table>

6. REFERENCES

5. E. Osterwell et al, “NAT traversal through tunneling” www.cs.arizona.edu/NAT.
6. G. Tsirtsis, Qualcomm; H. Saloman, Elevate Technologies; “Dual Stack Mobility”, [RFC 4977], August 2007
12. MIPL (Mobile IPv6 for Linux), how to, 2004-4-20.

7. AUTHOR PROFILE

Chaman Singh have received the Master of Computer Application Degree from H.P.University Shimla, India and also qualified UGC NET. Doctor of Philosophy in Computer Science is under Submission. Have more than 4 years of Working Experience in Teaching, Software Development (Programming) and Networks.
Design and Implementation of Mobile IPv6 Data Communication in Dual Networks

Chaman Singh¹, Sanjeev Kumar², Sanjeev Kumar³ and K.L. Bansal⁴

¹ Department of Computer Application, Govt. P.G. College, Chamba
Affiliated to H.P. University, Shimla, Himachal Pradesh 176310, India

² Department of Journalism and Mass Communication, Govt. P.G. College, Chamba
Affiliated to H.P. University, Shimla, Himachal Pradesh 176310, India

³ Department of Physics, Govt. P.G. College, Chamba
Affiliated to H.P. University, Shimla, Himachal Pradesh 176310, India

⁴ Department of Computer Science, H.P. University, Shimla
Himachal Pradesh 171005, India

Abstract
Dual Stack Mobile IPv6 is an extension of Mobile IPv6 to support mobility of devices irrespective of IPv4 and IPv6 Networks. This is implemented by combining different modules. IPv4 private networks are behind Network Address Translation (NAT) devices. To bypass the Binding Update and Binding Acknowledgment by NAT, we need to encapsulate it in User Datagram Protocol (UDP) Packets. So, the Dual Stack Mobile IPv6 should support NAT Traversal and Detection. With the support of NAT Detection and Traversal Module in Dual Stack Mobile IPv6, the mobile node is able to move freely from IPv6 Network to IPv4 Network or vice-versa. The main objective of not breaking the connectivity at the time of switching from one network to other is accomplished by NAT Module in Dual Stack Mobile IPv6.

Keywords: Dual Stack, Network Address Translation, Detection, Tunneling, Home Agent, Mobility.

1. Introduction
IPv6 is introduced generally to resolve the address space issues and also provides several advanced features. Dual Stack Mobile IPv6 is an extension of Mobile IPv6 to support mobility of devices irrespective of IPv4 and IPv6 network. The application interface is required to exchange mobility information with Mobility subsystem [1]. Mobile IPv6 (MIPv6) is a protocol developed as a subset of Internet Protocol version 6 (IPv6) [2] to support mobile connections. MIPv6 [3] allows a mobile node to transparently maintain connections while moving from one subnet to another. The Mobile IPv6 protocol takes care of binding addresses between Home Agent (HA) and Mobile Node (MN). It also ensures that the Mobile Node is always reachable through Home Agent. Each mobile node is always identified by its home address [4], regardless of its current point of attachment to the Internet. While situated away from its home, a mobile node is also associated with a care-of address, which provides information about its current point of attachment to the Internet. The protocol provides for registering the care-of address with a home agent. The home agent sends datagram’s destined for the mobile node through a tunnel to the care-of address. After arriving at the end of the tunnel, each datagram is then delivered to the mobile node. Currently, two mobility management protocols are defined for IPv4 and IPv6. Deploying both in a dual stack mobile node introduces a number of problems. This has been improved [5]. Mobile IPv6 uses IP Security (IPSec) to protect signaling between the home agent and the mobile node [6] [22]. Generic Packet Tunneling [7] specifies a method and generic mechanisms by which a packet is encapsulated and carried as payload within an IPv6 packet. The forwarding path between the source and destination of the tunnel packet is called an IPv6 tunnel. The technique is called IPv6 tunneling. The current Mobile IPv6 [3] and Network Mobility [8] [21] specifications support IPv6 only. These extend those standards to allow the registration of IPv4 addresses and prefixes, respectively, and the transport of both IPv4 and IPv6 packets over the tunnel to the Home Agent [9]. Allows the Mobile Node to roam over both IPv6 and IPv4, including the case where Network Address Translation is present on the path between the mobile node and its home agent. Middle Boxes [10], such as Firewalls and Network
Address Translators (NAT) [11], are an important component for a majority of Internet Protocol (IP) [5] networks today. Current IP networks are predominantly based on IPv4 [5] technology, and hence various firewalls as well as Network Address Translators have been originally designed for these networks. NAT's are necessary to overcome the IPv4 address space limitations of many network domains.

A network address translator a box that interconnects a local network to the public internet, where the local network runs on a block of private IPv4 addresses [12]. In the original design of the internet architecture, each IP address was defined to be globally unique and globally reachable. In contrast, a private IPv4 address is meaningful only within the scope of the local network behind a NAT and as such, the same private address block can be reused in multiple local networks [20], as long as those networks do not directly talk to each other. Instead, they communicate with each other and with the rest of internet through NAT boxes. Mobile IPv6 (MIPv6) [15] is a protocol developed as a subset of Internet Protocol version 6 (IPv6) to support mobile connections. The impact to IPv4, which changes IP address semantics, provides ample evidence, since now coming time MIPv6 are in progress so need of network address translation traversal and detection on Dual Stack implementation of mobile IPv6 [16]. NEPL (NEMO Platform for Linux) [17] is a freely available implementation of DSMIPv6 for Linux platform. The original NEPL release was based on MIPL (Mobile IPv6 for Linux) [18]. Without the support of NAT Detection and Traversal module in DSMIPv6, the mobile node will not be able to move freely from IPv6 network to IPv4 network or vice-versa. Connectivity also breaks at the time of switching from one network to other will be accomplished by this research.

2. Dual Stack Implementation of Mobile IPv6

First, NEPL (NEMO Platform for Linux) [17] is a freely available implementation of DSMIPv6 for Linux platform. The original NEPL release was based on MIPL (Mobile IPv6 for Linux) [18]. In Figure-1: Basic Operation of DSMIPv6, all Mobile Nodes (MN) has a fixed address, called a Home Address (HoA), assigned by Home Agent. When the Mobile Node moves to other networks, it gets Care-of Address (CoA) from foreign network. Mobile Node sends a Binding Update (BU) message to its Home Agent. Then Home Agent replies to the Mobile Node with a Binding Acknowledgement (BA) message to confirm the request. When Mobile Node is moved to any foreign network all packets sent to the Home Agent will be IPSec encrypted. A bi-directional tunnel is established between the Home Agent and the Care of address of the Mobile Node after the binding information has been successfully
exchanged. DSMIPv6 extends the MIPv6 and NEMO [6] Basic Support standards to allow mobile nodes to roam in both IPv6 [23] and IPv4-only networks. The following features are supported by the DSMIPv6 Architecture:-

1. The mobile node can register an IPv6/IPv4 Care of address to its Home Agent and so roam in IPv6-only networks and IPv4-only networks by the use of IPv6 and IPv6-in-IPv4 tunnels between the Node and its Home Agent.

2. A Network Address Translation Detection and Traversal Mechanism allow the Mobile Node to communicate with its Home Agent even though it uses an IPv4 private address as a Care of address. The signaling messages are always UDP encapsulated in IPv4 network. However, when the Mobile Node is located behind a NAT, data traffic is also encapsulated in UDP.

3. Securing the signaling packets between Home Agent and Mobile Node when Mobile Node is moved to foreign network and Session management on movement from one foreign link to another.

3. Solution Description

Solution is an extension to the existing NEPL solution provided by Nautilus [17]. We validated DSMIPv6 functionality as per the requirements provided against the draft, along with other IETF standards. We took the baseline architecture implementation from the Nautilus6 which uses Linux platform. The below mentioned steps are taken by us to achieve the requirements:-

1. Setup DSMIPv6 Test Lab using Kernel 2.6.28.2 and UMIP veMyon 0.4. In order to test the basic functionality between Home Agent and Mobile Node according to [3] the Test Bed has been setup.

2. Code change in MIP6 Daemon and Linux kernel and also applied the open source patches/packages. The Routing Advertisement daemon (radvd), IPSec daemon (strongswan) and Web Server (httpd) daemon has been configured on Home Agent.

3. The Mobile Node is configured with IPSec daemon (strongswan). Mobile Node gets IPv6 address whenever it is moved to any IPv6 foreign network through the radvd server running on the router.

4. When Mobile Node is moved to IPv4 network, it gets configured with IPv4 Care of address from the DHCP server running on IPv4 Router. In IPv4 network, DHCP is configured on the private network behind router. The network behind IPv4 router can be public or private.

4. Modules Representation in Dual Stack Mobile IPv6

MIPL (Mobile IPv6 for Linux) [24] is an open-source implementation of the Mobile IPv6 standard for the GNU/Linux operating system. MIPv6 is a user space for Mobile Node and Home Agent which aims at providing the necessary changes to MIPL in order to run on the latest kernels. Figure-2: Block Diagram of MIPv6 shows the internal data flow between two major components i.e. Home Agent and Mobile Node. Both of these two components consist of several helper modules which are also shown in this figure. This section describes IPv4 address assignment mechanism used by DSMIPv6. DHCP DNA module is used to obtain IPv4 address from the DHCP server running on IPv4 network.

4.1 DNA / DHCP Module

Module describes IPv4 address assignment mechanism used by DSMIPv6. When Mobile Node moves to IPv4 FL (Foreign Link) and its egress interface becomes enabled, Mip6d code in Mobile Node listens for Router Advertisement message, and since it does not receives Router Advertisement message in IPv4 FL, it gets timeout and sends Router Solicitation message (that will request the router to generate the Router Advertisement message immediately rather than at there next scheduled time), and Mobile Node wait for some time interval for Router Advertisement message before repeating the same procedure of sending Router Solicitation message. Meanwhile after sending Router Solicitation message, mip6d daemon will check the presence of DHCP server on the Egress interface link of Mobile Node by sending the DHCP discover message and wait for DHCP offer packet. Since the DHCP server is running on the IPv4 FL, it gets the IPv4 address from DHCP server and then mip6d code maps IPv4 address to IPv6 address, which is further used as Care of address. Mip6d daemon sets the default route on Mobile Node figure 3.

4.2 Movement Detection Modules

Describe Movement Detection in DSMIPv6 implementation. The movement of a mobile node away from its home link is transparent to transport and higher-layer protocols and applications. Describe Movement Detection in DSMIPv6 implementations. The Movement detection uses Neighbor Unreachablility Detection [13] to detect when the default router is no longer bi-directionally reachable, in which case the mobile node must discover a new default router (usually on a new link). However, this
detection only occurs when the mobile node has packets to send. When the mobile node detects handover, it expires the previous routers and Care-of-Address (es) and selects a new default router as a consequence of Router Discovery, and then performs Prefix Discovery with that new router to form new care-of address (es). It then registers its new primary care-of address with its home agent. After updating its home registration, the mobile node then updates associated mobility bindings in correspondent nodes. It triggers a movement event and then detects that Mobile Node is in foreign network. Route is modified and Home Address which was previously assigned to the physical interface is now moved to the tunnels. This is handled in 'mn.c'.

![Figure 2: MIPL for Home Agent and Mobile Node](image)

### 4.3 Binding Management Modules

Describe Binding Management in DSMIPv6 implementation. When a Mobile Node moves between different networks, it is essential that binding update messages are sent to that node's Home Agent and Correspondent Nodes as soon as possible, in order to facilitate a fast handoff. Mobile Nodes therefore cannot rely on the soft state timeout mechanism used in binding caches to refresh stale bindings maintained by Correspondent Nodes (typical binding lifetimes are of the order of minutes).
An additional data structure, the binding update list, is therefore kept by Mobile Nodes, which maintains state on any Correspondent Nodes or Home Agents. And in Home Agent mobile IPv6 binding cache maps home addresses to the current care-of addresses for each mobile node. This allows the home agent to tunnel traffic to the mobile node at its current location, and allows a correspondent node to send packets directly to a mobile node at its current location. Binding Management Module is classified as:

1. Binding Update List
2. Binding Cache

**Binding Update:** When Mobile Node moves to any Foreign Link, Mip6d code in Mobile Node initiate binding update functionality after configuring Care of address either from Router Advertisement message received in IPv6 or Dual stack FL or from DHCP server in IPv4 FL on the egress interface of Mobile Node. After Mobile Node configures Care of address on its egress interface, it creates BUL which consists of information about Mobile Node’s Home Address, Care of address, CN address, life time and delay time of this binding message in seconds and set of various flag.

![Process Flow in IPv4 Only Network](image)

Figure 3: Process Flow in IPv4 Only Networks

Binding Update List format is as follows:

```plaintext
== BUL_ENTRY ==
Home address 2001: x: x: x: x: x: x: x
Care-of addresses 2001: x: x: x: x: x: x: x
CN address 2001: x: x: x: x: x: x: x
Lifetime = 32, delay = 1500
Flags: IP6_MH_BU_HOMEx IP6_MH_BU_ACK IP6_MH_BU_TLV
```

When Mip6d code in Mobile Node creates BUL successfully, it send Binding Update message to Home Agent to register the new Care of address of Mobile Node to Home Agent. So that Home Agent gets updated with current Care of address of Mobile Node.

**Binding Cache:** Mip6d code in Home Agent received Binding Update message sent by Mobile Node when it moves to any Foreign Link or when it directly boots in IPv6/IPv4 FL. Home Agent process BU by performing DAD (dynamic address discover)[14] and latter parse Binding Update message. After pMyng the binding update message Home Agent creates or updates its Binding Cache entry. Binding Cache entry consists of
Mobile Node's Home Address, Mobile Node's Care of address, its local address, lifetime and sequence no. Binding Cache entry format is as follows:

\[ BC\_ENTRY = \]
\[ HoA\ 2001: a: b: 0:0:0:0:1\ status\ registered\ CoA\ 2001: a: d: 1:20c:29ff:fea0:4026 flags\ AH-- Local\ 2001: a: b: 0:0:0:0:1000\ Lifetime\ 23 / 32 sequence 3435\ Unreach 0 / 959299 retry -2\]

After updating its Binding Cache entry, the mip6d code creates devices to tunnel traffic to Mobile Node. Mip6d code in Home Agent creates and send Binding Acknowledgment message to Mobile Node so that Mobile Node gets acknowledged that it's new Care of address gets successfully registers with Home Agent. When Mobile Node receives Binding Acknowledgment message, mip6d code in Mobile Node parses the BA packet and update the BUL entry. It checks for various options set in BA and proceed accordingly. If NAT is detected between Mobile Node and Home Agent, it set xfrm policies/states to UDP Encapsulate IPv6/IPv4 data traffic to bypass NAT. Mip6d daemon sets the callback function to resend the BA, once lifetime of BUL entry is expired. And finally set the binding update timer to decrease the lifetime of BUL entry. When Mobile Node moves back from any FL to Home Link, Mip6d code in Mobile Node sends Binding Update message with lifetime set as zero to Home Agent to indicate that it as returned to Home Link mip6d code in Mobile Node deletes corresponding BUL entry. On Home Agent side Mip6d code receives BU message with lifetime set as zero, it indicate that Mobile Node moved to Home Link, so it deletes corresponding Binding Cache entry and send Binding Acknowledgment back to Mobile Node.

4.4 Route and Tunnel Management Modules

Tunnel and Route Management module figure 4 is mainly responsible for tunneling, when mobile node changes from IPv6 to IPv6, IPv6 to IPv4 and vice versa network. This module configures sit and ip6tnl interface via IOCTL system call which in turns performs the task at kernel level. Route Management handles the return rout-ability with CN. Some data structures are being used between some of the important functions in Tunnel Management module. Some user land data structures used in various routines in tunnel management module. Some data structures used in various routines in tunnel management module.
4.5 XFRM and IPSEC Management Modules

XFRM [15] is a packet transformation framework residing in the Linux kernel. It performs operations on IP packets such as inserting, modifying headers, UDP encapsulation and de-encapsulation. DSMIPv6 XFRM module will take the advantage of existing IPSEC transformation and defines a simple UDP encapsulation scheme. IPSEC module is responsible for interaction with IKE through MIGRATE messages. IPSEC will be used to protect the following traffic between Home Agent and Mobile Node.

1. BU/BA messages.
2. Mobile prefix solicitation and advertisement messages.
3. Normal traffic between Mobile Node and Home Agent.
4. All tunneled normal traffic between Mobile Node and correspondent Node.

In Mip6d, the Mobile Node and the Home Agent uses IPsec Security Associations in transport mode to protect BU/BA messages, since the MN may change its attachment point to the Internet, it is necessary to update its endpoint address of the IPsec SAs. This indicates that corresponding entry in IPsec databases (Security Policy and SA databases) should be updated when Mobile Node performs movements. IPsec is used to protect the following traffic between Home Agent and Mobile Node. When Mobile Node move in FL a new Care of address is assign to it by FL network. After detecting the movement following steps are taken to create IPsec tunnel.

1. Mip6d issues a PF_KEY MIGRATE message to the PF_KEY socket. The operating system validates the message and checks if corresponding security policy entry exists in SPD. When the message is confirmed to be valid, the target SPD entry is updated according to the MIGRATE message. If there is any target SA found that is also target of the update, those should also be updated.

2. After the MIGRATE message is successfully processed inside the kernel, it will be sent to all open PF_KEY sockets. The IKE daemon receives the MIGRATE message from its PF_KEY socket and updates its SPD and SAD images. The IKE daemon may also update its state to keep the IKE session alive. After that ESP protected BU is send with K-bit set.

   Mobile IPv6 specifies a flag named Key Management Mobility Capability bit (K-bit) in Binding Update (BU) and Binding Acknowledgement (BA) messages, which indicates the ability of IKE sessions to survive movement. When both the Mobile Node and Home Agent agree to use this functionality, the IKE daemon dynamically update the IKE session when the Mobile Node moves. The following methods are used.

4.6 NAT Detection and Traversal Modules

NAT (Network Address Translation (figure 5)) is the translation of an Internet Protocol address (IP address) used within one network to a different IP address known within another network. One network is designated the inside network and the other is the outside. In DSMIPv6 the mip6d daemon should bypass NAT, when Mobile Node is behind NAT ed device in IPv4 FL. NAT detection is done when the initial Binding Update message is sent from the mobile node to the home agent. When located in an IPv4-only foreign link, the mobile node sends the Binding Update message encapsulated in UDP (User Datagram Protocol) and IPv4; this is handled in xfrm.c file. The mip6d daemon adds xfrm policy/state for UDP encapsulation for BU packet. When the home agent receives the encapsulated Binding Update, it compares the IPv4 address of the source address field in the IPv4 header with the IPv4 address included in the IPv4 care-of address option. If the two addresses match, no NAT device is in the path. Otherwise, a NAT is detected in the path and the NAT detection option is included in the Binding Acknowledgement. The Binding Acknowledgement, and all future packets, is then encapsulated in UDP and IPv4. Note that the home agent also stores the port numbers and associates them with the mobile node's tunnel in order to forward future packets. This is handled in hac.c file. The mip6d daemon adds the xfrm polices/states for UDP encapsulation of BA and IPv6/IPv4 data traffic. Upon receiving the Binding Acknowledgement with the NAT detection option, the mobile node sets the tunnel to the home agent for UDP encapsulation. Hence, all future packets to the home agent are tunneled in UDP and IPv4. If no NAT device is detected in the path between the mobile node and the home agent then IPv4/IPv6 data traffic is not UDP encapsulated. A mobile node will always tunnel the Binding Updates in UDP when located in an IPv4-only network. Essentially, this process allows for perpetual NAT detection. Similarly, the home agent will encapsulate Binding Acknowledgements in a UDP header whenever the Binding Update is encapsulated in UDP. This is handled in mn.c and xfrm.c file. The mip6d daemon adds xfrm polices/states for UDP encapsulation of IPv6/IPv4 data traffic, when NAT is detected between Mobile Node and Home Agent.
4.7 Mobility Listener Modules

The Mobility Header is an extension header used by mobile nodes, correspondent nodes, and home agents in all messaging related to the creation and management of bindings. The Mobility Header is identified by a Next Header value of 135 in the immediately preceding header. Header is used to carry various messages. A mobile node uses the Home Test Init message to initiate the return rout-ability procedure and request a home keygen token from a correspondent node. The Home Test Init message uses the MH Type value 1. The Home Test message is a response to the Home Test Init message, and is sent from the correspondent node to the mobile node. The Home Test message uses the MH Type value 3. A mobile node uses the Care-of Test Init message to initiate the return rout-ability procedure and request a care-of keygen token from a correspondent node. The Care-of Test Init message uses the MH Type value 2. The Care-of Test message is a response to the Care-of Test Init message, and is sent from the correspondent node to the mobile node. The Care-of Test message uses the MH Type value 4. The Binding Update message is used by a mobile node to notify their nodes of a new care-of address for itself. The Binding Update uses the MH Type value 5. The Binding Acknowledgement is used to acknowledge receipt of a Binding Update. The Binding Acknowledgement has the MH Type value 6. The Binding Refresh Request message requests a mobile node to update its mobility binding. The Binding Refresh Request message uses the MH Type value 0. The Binding Error message is used by the correspondent node to signal an error related to mobility, such as an inappropriate attempt to use the Home Address destination option without an existing binding. The Binding Error message uses the MH Type value 7.

5. Conclusions

This paper is one of the earliest attempts in the community to investigate the problems and impacts when middle boxes, especially NAT devices are placed in Dual stack Mobile IPv6 are implemented in computer laboratory. With the support of modules in DSMIPv6, the mobile node is able to move freely from IPv6 network to IPv4 network or vice-versa. It accomplishes the main objective of not breaking the connectivity at the time of switching from one network to other. The transition from IPv4 to IPv6 will be time consuming process, so there will be time, when both IPv4 and IPv6 networks will be there and there will be always being scope for further development.

Acknowledgments
The authors would also like to express their sincere thanks to Mr. Manish Jamwal for his worthy Help and support for This Work.

References
[18]. MIPL (Mobile IPv6 for Linux), how to, 2004-4-20.
[20]. L. Cao, G J Qi, S F Tsai, M H Tsai, A D Pozo, T S Huang, et al. “Multimedia Information Networks in Social Media” in Social Network Data Analytics-2011.
NAT Traversal Capability and Keep-Alive Functionality with IPsec in IKEv2 Implementation

Chaman Singh¹

¹Research Scholar
chaman83mca@gmail.com
Department of Computer Science, Himachal Pradesh University Shimla, India

K.L. Bansal²

²Associate Professor
kishorilalbansal@yahoo.co.in
Department of Computer Science, Himachal Pradesh University Shimla, India

Abstract

Since IPv4 Private Networks are behind NAT (Network Address Translation) devices. So, to bypass the Binding Update and Binding Acknowledgment by NAT, we need to encapsulate it in UDP (User datagram Protocol) Packets. Hence, the Dual Stack Mobile IPv6 should support NAT Traversal and Detection. So for proper securing and fully functionality of NAT traversal, it should be IP Security Protected. Paper presents design and implementation of NAT traversal capability and keeps alive functionality with IP Security in IKEv2 (Internet Key Exchange version 2) implementation for proper Data Communication. It also implements how IPsec integrate with NAT.

Keywords—Network Address Translation, Traversal, Detection, IP Security, Home Link, Data Traffic, Linux Kernel, IKEv2.

1. Introduction

The Mobile IPv6 [1] is a protocol developed as a subset of Internet Protocol v6 (IPv6) [2] to support mobile connections. Mobile IPv6 allows a mobile node to transparently maintain connections while moving from one subnet to another. The Mobile IPv6 protocol takes care of binding addresses between Home Agent and Mobile Node. It also ensures that the Mobile Node is always reachable through Home Agent. Dual Stack Mobile IPv6 [5] is an extension of Mobile IPv6 to support mobility of devices irrespective of IPv4 and IPv6 network. NEPL (NEMO Platform for Linux) [6] is a freely available implementation of DSMIPv6 for Linux platform. The original NEPL release was based on MIPv (Mobile IPv6 for Linux) [7]. In DSMIPv6, all Mobile Nodes has a fixed address, called a Home Address assigned by Home Agent. When the MN moves to other networks, it gets Care-of Address from foreign network. MN sends a Binding Update message to its home agent. Then Home Agent replies to the Mobile Node with a Binding Acknowledgement message to confirm the request. When MN is moved to any foreign network all packets sent to the Home agent will be IPSec encrypted. A bi-directional tunnel [8] is established between the Home Agent and the care of address of the Mobile Node after the binding information has been successfully exchanged. DSMIPv6 [9] extends the Mobile IPv6 and NEMO Basic Support standards to allow MobileNodes to roam in both IPv6 and IPv4-only networks. [9] Solution is an extension to the existing NEPL solution provided by Nautilus [10]. Network Address Translation (NAT) [15] was meant to be temporary, but it's now in widespread use and it's actually holding back wider deployment of IPv6. Apart from the address shortage, Internet also has security related problems. There are different solutions for these problems currently in use, of which we are particularly interested in IPsec. IPsec is architecture [16], currently in a second generation that defines behaviour of compliant IPsec nodes. Those are Encapsulating Security Payload (ESP) [17] used for traffic encryption and integrity protection, and Authentication Header (AH) [18]. Of those two, ESP is mandatory to implement, while AH was mandatory but now is optional. The third protocol is Internet Key
Exchange version 2 (IKEv2) [19] and it is used for authentication, authorization and key exchange within IPsec (Internet Protocol Security) architecture. Widespread deployment of NAT based devices creates substantial problems to IPsec protocols. As we implemented NAT in IKEv2 protocol we had to do thorough analysis of possible problems and their solutions. This paper summarizes our design and code changes for proper communication in IKEv2 implementation. MY validated the DSMIPv6 functionality as per the requirements provided against the draft-ietf-mext-nemo-v4traversal-08.txt I-D, [5] along with other IETF standards. I have taken baseline architecture implementation from the Nautilus6 which uses Linux platform [11].

2. Network Address Translation

NAT (Network Address Translation) [13] is the translation of an Internet Protocol address (IP address) used within one network to a different IP address known within another network. One network is designated the inside network and the other is the outside. In DSMIPv6 the mip6d Daemon should bypass NAT, when MN is behind Nat"Ted device in IPv4 FL. NATs were introduced primarily because of the shortage of IPv4 addresses. IP nodes that are "behind" a NAT device have IP addresses that are not globally unique. They are more often assigned from some space that is unique within the network behind the NAT but which are likely to be reused by nodes behind other NATs. Node behind a NAT, which wants to communicate with other node on the Internet, is assigned a global IP address by NAT box which results with change of source IP address for outgoing packets. Similar situation is when destination node is behind a NAT, then for incoming packets NAT box changes destination IP address to the private IP address of node on the internal network. NAT box keeps the mapping for the duration of the communication. This duration is estimated by NAT box heuristically. Mapping is often achieved by additional translation based on UDP or TCP ports. In that case, NAT box is known as a NAPT box. There are many protocols having complications with NAT [20]. Applications such as FTP, H.323, SIP and RTSP use a control connection to establish a data flow and they are usually broken by NAT devices en-route. This is because these applications exchange address and port parameters within control session to establish data session and session orientations.

![Figure 1: NAT Detection and Traversal Module](image)

Most likely reasons for failures are that addressing information in payload could be realm specific and second, that control sessions permit data sessions to originate in a direction that NAT might not permit.
to peer applications also have problems with NAT. They can be originated by any of the peers and external peers will not be able to locate their peers in private realm unless they know the externally assigned IP address. Applications requiring retention of address mapping or requiring more public addresses than available are broken by NAT for obvious reasons. Namely, in the first case NAT cannot know this requirement and may assign external addresses between sessions to different hosts and in the second case NAT is limited by number of available public addresses.

3. NAT Traversal and Detection Design

NAT Detection [14] is done when the initial Binding Update message is sent from the mobile node to the home agent. When located in an IPv4-only foreign link, the mobile node sends the Binding Update message encapsulated in UDP and IPv4, this is handled in a particular file. When the home agent receives the encapsulated Binding Update, it compares the IPv4 address of the source address field in the IPv4 header with the IPv4 address included in the IPv4 care-of address option. Otherwise, a NAT is detected in the path and the NAT detection option is included in the Binding Acknowledgement. The Binding Acknowledgement, and all future packets, is then encapsulated in UDP and IPv4. Note that the home agent also stores the port numbers and associates them with the mobile node's tunnel in order to forward future packets. The mip6d Daemon adds the xfrm polices/states for UDP encapsulation of BA and IPv6/IPv4 data traffic [21]. Upon receiving the Binding Acknowledgement with the NAT detection option, the mobile node sets the tunnel to the home agent for UDP encapsulation. Hence, all future packets to the home agent are tunnelled in UDP and IPv4. If no NAT device was detected in the path between the mobile node and the home agent then IPv4/IPv6 data traffic is not UDP encapsulated. A mobile node will always tunnel the Binding Updates in UDP when located in an IPv4-only network. Essentially, this process allows for perpetual NAT detection. Similarly, the home agent will encapsulate Binding Acknowledgements in a UDP header whenever the Binding Update is encapsulated in UDP. The mip6d Daemon adds xfrm polices/states for UDP encapsulation of IPv6/IPv4 data traffic, when NAT was detected between MN and HA.

4. IPsec for Private Networks

IPsec keeps records about traffic which needs to be protected and how to protect it in two databases - SPD (Security Policy Database) and SAD (Security Association Database). SPD contains entries about security policy - which traffic to protect, which protocol to use, level of protection etc. Traffic selectors specify which packets to protect by specifying source and destination addresses, upper layer protocols and ports. IPsec is based on SA (Security association), which is a set of security parameters, for instance crypto algorithms used in communication. SA is uniquely defined by protocol (AH or ESP), destination IP address and SPI (Security Parameters Index). Two sides will establish connection if and only if they successfully negotiate security parameters for the connection. ESP and AH are two main security protocols in the IPsec architecture which assure traffic protection. AH is used for authentication and integrity check, while ESP is used primarily to enable confidentiality and optionally, authentication and integrity check. There are two IPsec modes: transport mode and tunnel mode. Transport mode is appropriate for usage when communication is end-to-end. In this mode, we have only one source and destination IPv4 address, which are in AH protected by ICV. This leads to problems with NAT, as described later. ESP doesn't have these problems because his integrity check doesn't cover IPv4 header where are these addresses situated. Tunnel mode is better where communication takes place between security gateways. In this case communication is maintained within the IPsec tunnel. This leads to another pair of IP addresses and therefore, to another header besides the original one: "outer" IP header. ESP encryption now covers whole IPv4 datagram, with inner header also. AH authentication checks integrity of the both inner and outer IPv4 header, and off course, IPv4 payload. Integrity check successfully reveals attempts of packet change by intruder on insecure network. AH and ESP require cryptographic keys to be in SA database. Though possible manual key management isn't particularly secure and doesn't scale well. These problems are solved by automatic key exchange, specifically by Internet Key Exchange version 2 (IKEv2) protocol. Daemon, which runs IKEv2 protocol, generates symmetric keys and does rekeying after some period. Authentication in IPsec is also performed by the IKEv2 protocol using pre-shared keys, digital certificates or EAP. IKEv2 messages are transferred via UDP protocol in pairs, requests and response. Each pair is known as exchange. Communication between two IKEv2 entities is established via two exchanges. Establishment of SA includes traffic selectors and cryptographic algorithms to use for data protection. There are two design possibilities with the respect to interrelation of IPsec and NAT devices. The first one is for IPsec protocols to completely ignore NAT, while the other one is to introduce mechanisms in the
protocol that will allow IPsec compliant devices to communicate in spite of NAT devices.

5. Interaction of IPSec and IKEv2

XFRM [11] is a packet transformation framework residing in the Linux kernel. It performs operations on IP packets such as inserting, modifying headers, UDP encapsulation and de-encapsulation. DSMIPv6 XFRM module will take the advantage of existing IPSEC transformation and defines a simple UDP encapsulation scheme. IPSEC module is responsible for interaction with IKEV2 through MIGRATE messages. IPSec will be used to protect the following traffic between Home Agent and Mobile Node:

1. BU/BA messages.
2. Mobile prefix solicitation and advertisement messages.
3. Normal traffic between Mobile Node and Home Agent.
4. All tunneled normal traffic between Mobile Node and correspondent Node.

In Mip6d, the Mobile Node (MN) and the Home Agent (HA) uses IPsec Security Associations (SAs) in transport mode to protect BU/BA messages, since the MN may change its attachment point to the Internet, it is necessary to update its endpoint address of the IPsec SAs. This indicates that corresponding entry in IPsec databases (Security Policy (SPD) and SA (SAD) databases) should be updated when Mobile Node performs movements. IPSec is used to protect the following traffic between Home Agent and Mobile Node:

**BU/BA messages:** IPSec Protection for BU/BA

When Mobile Node moves in FL a new Care of address is assigned to the Mobile Node by FL network. After detecting the movement following steps are taken to create IPSec tunnel.

1. Mip6d issues a PF_KEY MIGRATE message to the PF_KEY socket.
2. The operating system validates the message and checks if corresponding security policy entry exists in SPD.
3. When the message is confirmed to be valid, the target SPD entry is updated according to the MIGRATE message. If there is any target SA found that are also target of the update, those should also be updated.
4. After the MIGRATE message is successfully processed inside the kernel, it will be sent to all open PF_KEY sockets. The IKE daemon receives the MIGRATE message from its PF_KEY socket and updates its SPD and SAD images. The IKE daemon may also update its state to keep the IKE session alive.

5. After that ESP protected BU is send with K-bit set.

Mobile IPv6 specifies a flag named Key Management Mobility Capability bit (K-bit) in Binding Update (BU) and Binding Acknowledgement (BA) messages, which indicates the ability of IKE sessions to survive movement. When both the Mobile Node and Home Agent agree to use this functionality, the IKE daemons dynamically update the IKE session when the Mobile Node moves.

6. Description and Implementation

It contains the details about patches applied and code changes done by me in Linux kernel, mipv6 Daemon and strongSwan in different releases of DSMIPv6. The scope of this release is to demonstrate the following working scenarios:

**Scenario 1:** Movement of MN from HL to IPv4 network.

**Scenario 2:** Movement of MN from IPv4 to HL network.

**Scenario 3:** Movement of MN from IPv6 to IPv4 network.

**Scenario 4:** Movement of MN from IPv4 to IPv6 network.

It also captures the working of the below mentioned features to demonstrate the above mentioned scenarios:

2. Handover interactions for IPsec and IKE
3. IKE negotiations between MN and HA
4. IKEv2 operation for securing DSMIPv6 signalling (BU & BA).
5. NAT Detection in MN and HA.
6. NAT Traversal in MN and HA
8. UDP Encapsulation of IPv6 and IPv4 data traffic.

### 6.1 Linux Kernel

**Description:** Used patched Linux kernel 2.6.28.2 for this release.

**VeMyon:** 2.6.28.2

**Tar file:** linux-2.6.28.2.tgz

### 6.2 User land-DSMIP

**Description:** User land DSMIP Daemon used in mipv6 taken from nautilus site. All patches applied to support DSMIPv6.

**VeMyon:** 0.4

**Tar file:** mipv6-Daemon-umip-0.4.tgz

### 6.3 User land IKEv2 Daemon

**Description:** Used strongSwan package as user land IKEv2 Daemon

**VeMyon:** 4.2.9

**Tar file:** strongswan-4.2.9.tgz

### 6.4 Changes done in user land-DSMIP

**Description:** Code changes have been made in mip6d Daemon for successfully detecting NAT.

#### 6.4.1 Change list 1.

**Bug Description:** NAT Detection logic was failing, when we move from IPv6 FL to IPv4 FL.

**File Modified:** ha.c

**Function Modified:** ha_recv_bu_worker

**Bug Fix Description:** NAT detection logic was failing, when we move from IPv6 to IPv4 FL. In this scenario the CoA in bce (Binding Cache Entry) was old value that is IPv6 Address. Due to which IP addresses (source IP & CoA in bce) were getting mismatched and NAT was getting detected, which is wrong behavior. So at the time of comparing addresses, using CoA from out.bind_coa instead of CoA in BCE...

**Code Snippet for Minor Changes**

NAT detection logic was failing, when we move from IPv6 to IPv4. In this scenario the CoA in bce is old value that is IPv6 Address. Due to which IP addresses (src IP & CoA in bce) were getting mismatched and NAT was getting detected, which is wrong behaviour. So at the time of comparing addresses, using CoA from out.bind_coa instead of CoA in BCE

```c
if (! IN6_ARE_ADDR_EQUAL (&v4mapped_src, out.bind_coa))
```


6.4.2 Change list 2.

**Bug Description:** At the time of movement from IPv4 to IPv6 Network (or HL to IPv6). Sometime sit device was going in wrong state. And if after that MN moves from IPv6 to IPv4. Mip6d was not able to access the sit device and finally deleted the sit interface. As there was no sit device in MN, mip6d code throws assertion, when it tries to modify the sit tunnel endpoints.

**File Modified:** mn.c

**Function Modified:** mn_tnl_state_add

**New Functions Added:**
- mn_clr_interface_flag
- mn_set_interface_flag
- index2name

**Bug Fix Description:** To avoid the assertion currently added the hack to avoid this scenario. Doing down or up of sit device, only when MN moves from IPv4 to IPv6 FL or HL to IPv6 FL. Latter need some better fix.

**Comparison of Files for Major Changes:** Comparison of changes done to fix the issue in mn.c

6.4.3 Change list 3.

**Bug Description:** When MN moves second time from HL to IPv4, the IPv6 HoA is assigned to ip6tnl and latter it should move from ip6tnl to sit device, which was not happening. The correct behavior should be same as we do for IPv4 HoA. At the first movement from HL to IPv4, sit device is assigned IPv6 HoA correctly, because hai to if_tunnel stores the sit index value initially.

**File Modified:** mn.c

**Function Modified:** process_first_home_bu

**Bug Fix Description:** When Mobile Node moves second time from Home Link to IPv4 FL, moving IPv6 HoA from ip6tnl to sit device, by calling routine mv_hoa

**Code Snippet for Minor Changes:** When MN moves second time from HL to IPv4, the IPv6 HoA is assigned to ip6tnl and latter it should move from ip6tnl to sit device, which was not happening. The correct behavior should be same as we do for IPv4 HoA. At the first movement from HL to IPv4, sit device is assigned IPv6 HoA correctly, because hai to is_tunnel * stores the sit index value. Done changes to fix this issue....

```c
if (hai->if_tunnel != hai->if_tunnel64)
{
struct mv_hoa_args mha;
    mha.if_next = hai->if_tunnel64;
    mha.target = hai;
    addr_do (&hai->hoa.addr, 128, hai->if_tunnel, &mha, mv_hoa);
}
hai->if_tunnel = hai->if_tunnel64;
```

6.4.4 Change list 4.

**Bug Description:**

There was bug in mipv6 logic; it was not taking the prefix length of IPv4 HoA configured by user in configuration file. Due to which problem was coming in setting v4 route, when MN boots in IPv6 FL and moves to IPv4 FL.

**File Modified:** mn.c

**Function Modified:** flag_hoa4

**Bug Fix Description:** Done code changes to correct the logic, so that mipv6 Daemon should take the prefix length of IPv4 HoA, if configured by user in mip6d configuration file.

**Code Snippet for Minor Changes:** Changes have been made to accept the actual prefix Length from the configuration file. This case occurs when we boot the MN in IP6 LINK, here ifa_index and if_tunnel4 are different so take the prefix length from conf file. This also solves when MN moves IPv6 to IPv4 LINK because ifa_index and hai_index are not equal so it will take 32 as prefix length. This resolves the v4 route issue when MN boots up in IPv6 link.

```c
plen4 = (ifa->ifa_index! = hai->if_tunnel4 ? 32: hai->plen4);
```

6.4.5 Change list 5.
Bug Description: IPv4 traffic was not passing through tunnel device in IPv4 FL

File Modified: dhcp_dna.c, dhcp_dna.h, mn.c

Function Modified: dhcp_configuration

New Functions Added:

- mn_coa_route_add
- mn_route_coa_del

Bug Fix Description:
Added source based route to pass IPv4 traffic through tunnel device and not through egress interface of MN directly.

Code Snippet for Minor Changes:

dhcp_dna.h

Structure for storing the route in formation needed in IPV4 link

```c
struct dhcp_route {
    int if_index;
    unsigned long gateway;
};
```

dhcp_dna.c

Copying the value of Gateway and Ifindex value to dhcp_route structure Will needed in creating a source based route for BU.

```c
route_dhcp.if_index = if_index;
route_dhcp.gateway = dhcp_ctrl_gateway;
```

Comparison of files for Major Changes:
Comparison of changes done to fix the issue in mn.c

6.4.6 Change list 6.

Bug description: External IPv4 network of Home Agent was not reachable from MN in HL.

File Modified: mn.c

Function Modified: mn_move

Bug Fix Description: - Adding default route to Home Agent IPv4 address, when Mobile Node is in HL. So that MN can reach to other network than HL. Route is added on the physical interface where IPv4 HoA is configured.

Code Snippet for Minor Changes: Adding default route to HOMAGENTV4ADDRESS when MN is in HOME LINK So that Mobile Node can reach to other network than HL. Route is added on the physical interface where IPv4 Home Address is configured.

```c
MDBG("Default route is added in HL toward iface_index %d
", hai->hoa.iif);
if (route4_add(hai->hoa.iif, RT6_TABLE_MAIN, NULL, NULL, 0, &any4, 0, &ha4_addr) < 0)
    MDBG("Default route insertion failed for MN in HL.");
CHANGES: Deletion of default route to HOMAGENT V4ADDRESS, when MN is not in HOMELINK.
 MDBG("Default route is deleted in FL toward iface_index %d
", hai->hoa.iif);
if (route4_del(hai->hoa.iif, RT6_TABLE_MAIN, NULL, 0, &any4,0,&ha4_addr) < 0) MDBG("Default route deletion failed for MN in HL.");
```

6.4.7 Change list.

Bug Description: When MN was behind NAT in IPv4 FL, the large size IPv4/IPv6 data traffic was not getting exchanged. After initial handshake, client was not able to exchange data traffic and was in hang state. In nutshell everything that uses large packets was not working.

File Modified: - tunnelctl.c

Function modified: __tunnel44_add

```c
__tunnel44_add
```

Bug Fix description: Set the MTU size of tunnel device to 1472 instead of 1480, left 8 bytes for UDP Encapsulation header.

Code Snippet for Minor Changes: __tunnel44_add

```
[DSMIP BUG]: When MN was behind NAT in IPv4 FL, the large size IPv4/IPv6 data traffic was not getting exchanged. After initial handshake, client was not able to exchange data traffic and was in hang state. In nutshell everything that uses large packets was not working. Fix: set the MTU size of tunnel device to 1472 instead of 1480, left 8 bytes for UDP Encapsulation header...

```

```c
ifr.ifrfru.ifrumtu
```

105
if (ioctl(tnl4_fd, SIOCSIFMTU, &ifr) < 0) {
    TDBG ("SIOCSIFFLAGS failed MTU %d
    %s", errno, strerror(errno));
    goto err;
}

CHANGES: [DSMIP_BUG]: When MN was behind NAT in IPv4 FL, the large size IPv4/IPv6 data traffic was not getting exchanged. After initial handshake, client was not able to exchange data traffic and was in hang state. In nutshell everything that uses large packets was not working. Fix: set the MTU size of sit device to 1472 instead of 1480, left 8 bytes for UDP Encapsulation header.
ifr.ifr.ifru.ifru_mtu=MAX_MTU_SIZE_FOR_SIT_DEV; if (ioctl (tnl4_fd, SIOCSIFMTU, &ifr) < 0)
{
    TDBG ("SIOCSIFFLAGS failed MTU %d
    %s", errno, strerror(errno));
    goto err;
}

switch (this->mode) {
    case MODE_TRANSPORT:
        if (!this->config->use_proxy_mode(this->config) ||
            !ts_list_is_host(this->tsi, other) ||
            !ts_list_is_host(this->tsr, me))
        {
            this->mode = MODE_TUNNEL;
            DBG1(DBG_IKE, "not using transport mode, not host-to-host");
        }
        else if (this->ike_sa->has condition (this->ike_sa,
            COND_NAT_ANY))
        {
            /* Do not switch to tunnel mode when nat is detected. For securing signaling we need transport mode SA as per draft. So stopping switches to tunnel mode in case of NAT-T. DBG1 (DBG_IKE, "not using transport mode, connection NATed"); */
            //this->mode = MODE_TUNNEL;
            DBG1(DBG_IKE, "using transport mode, connection NATed");
        }
        break;
    case MODE_BEET:
        if (!ts_list_is_host(this->tsi, NULL) ||
            !ts_list_is_host(this->tsr, NULL))
        {
            this->mode = MODE_TUNNEL;
            DBG1(DBG_IKE, "not using BEET mode, not host-to-host");
        }
}

6.5 Changes done in user land-DSMIP
6.5.1 Changes done in user land IKEv2 Daemon.

Description: Integration of IPsec with NAT
Change list-1:
Bug Description: IPsec signalling when MN was behind NAT was failing.

File Modified:
strongswan-4.2.9/src/charon/sa/tasks/
    child_create.c

strongswan-4.2.9/src/charon/plugins/kernel_netlink/
    kernel_netlink_ipsec.c

Function Modified: select_and_install, status_t_add_sa

Code Snippet: child_create.c: This code is present in function select_and_install () of if (! this->initiator)
{
    /* check if requested mode is acceptable, down grade if required */

    Figure 3:- Implementation of Kernel Interface.
```c
kernel netlink ipsec
/*Implementation of kernel_interface_t.add_sa */
static status_t addsa (private_kernel_netlink_ipsec_t *this, host_t *src, host_t *dst, u_int32_t spi,
 protocol_id_t protocol, u_int32_t reqid, u_int64_t expire_soft, u_int64_t expire_hard, u_int16_t enc_alg, chunk_t
 enc_key, u_int16_t int_alg, chunk_t int_key, ipsec_mode_t mode, u_int16_t ipcomp, u_int16_t cpi, bool encap,
 bool inbound)
{
    netlink_buf_t request;
    char *alg_name;
    struct nlmsghdr *hdr;
    struct xfrm_usersa_info *sa;
    u_int16_t icv_size = 64;
    /* if IPComp is used, we install an additional IPComp SA. if the cpi is 0 we are in the recuMyve call below */
    if (ipcomp != IPCOMPNONE && cpi != 0)
    {
        add_sa(this, src, dst,
               0,0,ENCR_UNDEFINED,chunk_empty,
               FALSE, inbound);
        ipcomp = IPCOMP_NONE;
    }
    memset(&request, 0, sizeof(request));
    DBG2(DBG_KNL, "adding SAD entry with SPI %.8x and reqid{%u}", ntohl(spi), reqid);
    hdr = (struct nlmsghdr*)request;
    hdr->nlmsg_flags = NLMFREQUEST | NLMFACK;
    hdr->nlmsgjype = inbound ? XFRM_MSG_UPDSA

    /* we currently do not expire SAs by volume/packet count */
    sa->lft.soft_byte_limit = XFRM_INF;
    sa->lft.hard_byte_limit = XFRM_INF;
    sa->lft.soft_packet_limit = XFRM_INF;
    sa->lft.hard_packet_limit = XFRM_INF;
    /* we use lifetimes since added, not since used */
    sa->lft.soft_add_expires_seconds = expire_soft;
    sa->lft.hard_add_expires_seconds = expire_hard;
    sa->lft.soft_use_expires_seconds = 0;
    sa->lft.hard_use_expires_seconds = 0;
    struct rtattr *rhdr = XFRM_RTA(hdr, struct
    xfrm_usersa_info);
    switch (enc_alg)
    {
    case ENCR_UNDEFINED:
        /* no encryption */
        break;
    case ENCR_AES_CCM_ICV16:
        icv_size += 32;
        /* FALL */
    case ENCR_AES_CCM_ICV12:
        icv_size += 32;
        /* FALL */
    case ENCR_AES_CCM_ICV8:
        icv_size += 32;
        /* FALL */
    case ENCR_AES_GCM_ICV16:
        icv_size += 32;
        /* FALL */
    case ENCR_AES_GCM_ICV12:
        icv_size += 32;
        /* FALL */
    case ENCR_AES_GCM_ICV8:
        icv_size += 32;
        /* FALL */
    case ENCR_AES_GCM_ICV4:
        icv_size += 32;
        /* FALL */
    case ENCR_AES_CBC_ICV16:
        icv_size += 32;
        /* FALL */
    case ENCR_AES_CBC_ICV12:
        icv_size += 32;
        /* FALL */
    case ENCR_AES_CBC_ICV8:
        icv_size += 32;
        /* FALL */
    case ENCR_AES_CBC_ICV4:
        icv_size += 32;
        /* FALL */
    case ENCR_PSK_CBC_ICV16:
        icv_size += 32;
        /* FALL */
    case ENCR_PSK_CBC_ICV12:
        icv_size += 32;
        /* FALL */
    case ENCR_PSK_CBC_ICV8:
        icv_size += 32;
        /* FALL */
    case ENCR_PSK_CBC_ICV4:
        icv_size += 32;
        /* FALL */
    case ENCR_PSK_GCM_ICV16:
        icv_size += 32;
        /* FALL */
    case ENCR_PSK_GCM_ICV12:
        icv_size += 32;
        /* FALL */
    case ENCR_PSK_GCM_ICV8:
        icv_size += 32;
        /* FALL */
    case ENCR_PSK_GCM_ICV4:
        icv_size += 32;
        /* FALL */
    default:
        break;
    }
}
```

Figure 4: Implementation of Kernel Interface.
If (alg_name == NULL)
{
    DBG1(DBG_KNL, "algorithm %N not supported by kernel!", encryption_algorithm_names, enc_alg);
    return FAILED;
}

DBG2(DBG_KNL, " using encryption algorithm %N with key size %d", encryption_algorithm_names, enc_alg, enc_key.len * 8);
rthdr->rta_len = RTA_LENGTH(sizeof(struct xfrm_algo_aead) + enc_key.len);
hdr->nlmsg_len += rthdr->rta_len;
if (hdr->nlmsg_len > sizeof(request))
{
    return FAILED;
}

struct xfrm_algo_aead* algo = (struct xfrm_algo_aead*)RTA_DATA(rthdr);
algo->alg_key_len = enc_key.len * 8;
algo->alg_icv_len = icvsize;
strcpy(algo->alg_name, algname);
memcpy(algo->alg_key, enc_key.ptr, enc_key.len);
rthdr = XFRM_RTA_NEXT(rthdr);
break;
}
default:
{
    rthdr->rta_type = XFRMA_ALG_CRYPT;
    alg_name = lookup_algorithm( encryption_algs, enc_alg);
    if (alg_name == NULL)
    {
        DBG1(DBG_KNL, "algorithm %N not supported by kernel!", encryption_algorithm_names, enc_alg);
        return FAILED;
    }

    DBG2(DBG_KNL, " using encryption algorithm %N with key size %d", encryption_algorithm_names, enc_alg, enc_key.len * 8);
    rthdr->rta_len = RTA_LENGTH(sizeof(struct xfrmalgo) + enc_key.len);
    hdr->nlmsg_len += rthdr->rta_len;
    if (hdr->nlmsg_len > sizeof(request))
    {
        return FAILED;
    }

    struct xfrmalgo* algo = (struct xfrmalgo*)RTA_DATA(rthdr);
    algo->alg_key_len = enc_key.len * 8;
    strcpy(algo->alg_name, algname);
    memcpy(algo->alg_key, enc_key.ptr, enc_key.len);
    rthdr = XFRM_RTA_NEXT(rthdr);
    break;
}
}

if (int_alg != AUTH_UNDEFINED)
{
    rthdr->rta_type = XFRMA_ALG_AUTH;
    alg_name = lookup_algorithm(integrity_algs, int_alg);
    if (alg_name == NULL)
    {
        DBG1(DBG_KNL, "algorithm %N not supported by kernel!", integrity_algorithm_names, int_alg);
        return FAILED;
    }

Figure 5:- Implementation of Kernel Interface.
DBG2(DBG_KNL, " using integrity algorithm %N with key size %d",
   integrity_algorithm_names, int_alg, int_key.len * 8);
   rthdr->rta_len = RTA_LENGTH(sizeof(struct xfrmalgo) + int_key.len);
   hdr->nlmsg_len += rthdr->rta_len;
   if (hdr->nlmsg_len > sizeof(request))
     return FAILED;
   
   struct xfrmalgo* algo = (struct xfrmalgo*)RTA_DATA(rthdr);
   algo->alg_key_len = intkey.len * 8;
   strcpy (algo->alg_name, algname);
   memcpy (algo->alg_key, intkey.ptr, intkey.len);
   rthdr = XFRM_RTA_NEXT (rthdr);
}

if (ipcomp != IPCOMP_NONE)
{
   rthdr->rta_type = XFRMA_ALG_COMP;
   alg_name = lookup_algorithm (compression_algs, ipcomp);
   if (alg_name == NULL)
     {
       DBG1(DBG_KNL, "algorithm %N not supported by kernel!", algcomp_transform_names, algcomp);
       return FAILED;
     }

   DBG2(DBG_KNL, " using compression algorithm %N", algcomp_transform_names, algcomp);
   rthdr->rta_len = RTA_LENGTH(sizeof(struct xfrmalgo));
   hdr->nlmsg_len += rthdr->rta_len;
   if (hdr->nlmsg_len > sizeof(request))
     return FAILED;

   struct xfrmalgo* algo = (struct xfrmalgo*)RTA_DATA(rthdr);
   algo->alg_key_len = 0;
   strcpy(algo->alg_name, algname);
   rthdr = XFRM_RTA_NEXT(rthdr);
}

IPsec protection for data packets is not needed as of Now. This piece of code is conflicting with mip6d.

#if 0
if (encap)
{
   rthdr->rta_type = XFRMA_ENCAP;
   rthdr->rta_len = RTA_LENGTH(sizeof(struct xfrm_encap_tmpl));
   hdr->nlmsg_len += rthdr->rta_len;
   if (hdr->nlmsg_len > sizeof(request))
     return FAILED;

   struct xfrm_encap_tmpl* tmpl = (struct xfrm_encap_tmpl*) RTA_DATA (rthdr);
   tmpl->encap_type = UDP_ENCAP_ESPINUDP;
   tmpl->encap_sport = htons(src->get_port(src));
   tmpl->encap_dport = htons(dst->get_port(dst));
   memset (&tmpl->encap_oa, 0, sizeof (xfrmaddress_t));

   rthdr = XFRM_RTA_NEXT (rthdr);
}
#endif
if (this->socket_xfrm->send_ack (this->socket_xfrm, hdr) != SUCCESS)
  {
    DBG1 (DBG_KNL, "unable to add SAD entry with SPI %x", ntohl(spi));
    return FAILED;
  }
return SUCCESS;

Figure 6.- Implementation of Kernel Interface.
Function `encap_oa` could probably be derived from the traffic selectors [rfc4306]. In the net link kernel Implementation pluto does the same as we do here but it uses `encap_oa` in the pfkey implementation. BUT as `/usr/src/linux/net/key/af_key.c` indicates that the kernel ignores it anyway. Does that mean that NAT-Traversal encapsulation doesn't work in transport mode? No. The reason the kernel ignores NAT-OA is that it recomputed (or, rather, just ignores) the checksum. If packets pass the IPSec checks it marks them "checksum ok" so OA isn't needed.

8. Conclusion

NAT is a mechanism which brought momentary abandon to the problem of shortage of IPv4 addresses. Unfortunately, it also brought some problems which we solved as described. Security is an essence part of this protocol and therefore implementation procedure is used for NAT detection. After detection of NAT box, there are appropriate actions as described. Support of NAT traversal in IKEv2 implementation solved one of the important demands for IKEv2 implementations and made this implementation more general and therefore, more appropriate to use in the IPSec. We have also shows the integration of IPSec with NAT. So for proper securing and fully functionality of NAT traversal, it should be IP Security Protected. It contains the details about patches applied and code changes done by me in Linux kernel, mipv6 Daemon and strongSwan in different releases of DSMIPv6 to implement IKEv2. The implementation of this release was to demonstrate, Movement of MN from HL to IPv4 network and Movement of MN from IPv6 to IPv4 network and vice versa.

9. References


[7]. MIPL (Mobile IPv6 for Linux), how to, 2004-4-20.


<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AH</td>
<td>Authentication Header</td>
</tr>
<tr>
<td>CoT</td>
<td>Care-of Test</td>
</tr>
<tr>
<td>NATs</td>
<td>Network Address Translators</td>
</tr>
<tr>
<td>SCTP</td>
<td>Stream Control Transmission Protocol</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TURN</td>
<td>Traversal using Relay NAT</td>
</tr>
<tr>
<td>ALG</td>
<td>Application Layer Gateway</td>
</tr>
<tr>
<td>AR</td>
<td>Access Router</td>
</tr>
<tr>
<td>BA</td>
<td>Binding Acknowledgment</td>
</tr>
<tr>
<td>BU</td>
<td>Binding Update</td>
</tr>
<tr>
<td>BUL</td>
<td>Binding Update List</td>
</tr>
<tr>
<td>BUM</td>
<td>Binding Update Message</td>
</tr>
<tr>
<td>CN</td>
<td>Correspondent Node</td>
</tr>
<tr>
<td>CoA</td>
<td>Care-of-Address</td>
</tr>
<tr>
<td>CoTI</td>
<td>Care-of Test Init</td>
</tr>
<tr>
<td>CRM</td>
<td>Customer Relationship Management</td>
</tr>
<tr>
<td>DARPA</td>
<td>Defense Advanced Research Projects Agency</td>
</tr>
<tr>
<td>DHAAD</td>
<td>Dynamic home agent Address Discovery</td>
</tr>
<tr>
<td>DHCP</td>
<td>Dynamic Host Configuration Protocol</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DSMIPv6</td>
<td>Dual Stack Mobile Internet Protocol VeMyon 6</td>
</tr>
<tr>
<td>ERP</td>
<td>Enterprise Resource Planning</td>
</tr>
</tbody>
</table>
ESP  Encapsulating Security Payload
FL   Foreign Link
FW   Firewalls
GPL  General Public License
HA   Home Agent
HL   Home Link
HMAC Hash Message Authentication Code
HoA  Home Address
IETF Internet Engineering Task Force
IKE  Internet Key Exchange
IP   Internet Protocol
IPSec Internet Protocol Security
IPv4 Internet Protocol version 4.0
IPv6 Internet Protocol version 6.0
ISOC The Internet Society
ISP  Internet service Provider
MH   Mobile Header
MIDCOM Middlebox Communication
MIP6D Mobile IPv6 Daemon
MIPL Mobile IPv6 for Linux
MIPL Mobile Internet Protocol Version 6.0 for Linux
MIPv6 Mobile Internet Protocol version 6.0
MN   Mobile Node
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNN</td>
<td>Mobile Network Nodes</td>
</tr>
<tr>
<td>MPA</td>
<td>Mobile Prefix Advertisement</td>
</tr>
<tr>
<td>MPS</td>
<td>Mobile Prefix Solicitation</td>
</tr>
<tr>
<td>MR</td>
<td>Mobile Router</td>
</tr>
<tr>
<td>MY</td>
<td>My International Ltd.</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>NATTT</td>
<td>NAT Traversal Through Tunneling</td>
</tr>
<tr>
<td>NEMO</td>
<td>Network Mobility</td>
</tr>
<tr>
<td>NEPL</td>
<td>NEMO Platform For Linux</td>
</tr>
<tr>
<td>NEPL</td>
<td>NEMO platform for Linux</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>NM</td>
<td>Network Mobility</td>
</tr>
<tr>
<td>NSIS</td>
<td>NAT Signaling Layer Protocol</td>
</tr>
<tr>
<td>OSI</td>
<td>Open System Interconnection</td>
</tr>
<tr>
<td>PBN</td>
<td>Policy Based Network</td>
</tr>
<tr>
<td>PSK</td>
<td>Pre-shared keys</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>radvd</td>
<td>Routing Advertisement Daemon</td>
</tr>
<tr>
<td>RFC</td>
<td>Request for Comments</td>
</tr>
<tr>
<td>SA</td>
<td>Security Association</td>
</tr>
<tr>
<td>SIT</td>
<td>Simple Internet Transition</td>
</tr>
<tr>
<td>SP</td>
<td>Security Policy</td>
</tr>
<tr>
<td>SPD</td>
<td>Security Policy Database</td>
</tr>
</tbody>
</table>
DEFINITIONS

**Access Router (AR):** The AR is the Mobile Node's default router. The AR aggregates the outbound traffic of mobile nodes.

**Care-of address (CoA):** A unicast routable address associated with a mobile node while visiting a foreign network; the subnet prefix of this IP address is a foreign subnet prefix.

**Home address (HoA):** A unicast routable address assigned to a mobile node, used as the permanent address of the mobile node. This address is within the mobile node's home network.

**Security Association (SA):** An IPSec security association is a cooperative relationship formed by the sharing of cryptographic keying material and associated context. Security associations are simplex. That is, two security associations are needed to protect bidirectional traffic between two nodes, one for each direction.

**Security policy database (SPD):** A database that specifies what security services are to be offered to IP packets and in what fashion.

**UMIP:** UMIP is an open-source (GPLv2) Mobile IPv6 stack for the GNU/Linux Operating System.

**XFRM:** A packet transformation framework residing in the Linux Kernel.