NAT Traversal and Detection on Dual Stack Implementation of Mobile IPv6

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

Chaman Singh
Assistant Professor
Department of Computer Applications
Govt.P.G.College, Chamba
(H.P) India-171005

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 networks 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

1. INTRODUCTION
Network Address Translators are an important component for a majority of Internet Protocol (IP) networks today. Current IP networks are predominantly based on IPv4 technology, and hence various NATs have been originally designed for these networks. 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 [1]. 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, 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. It is worth pointing out that in the recent years many efforts were devoted to the development and deployment of NAT traversal solution, such as simple traversal of UDP (User Datagram Protocol) through NAT (STUN) [2], traversal using relay NAT (TURN) [3], and Teredo [4], to name a few. Theses solution removes obstacles introduced by NATs to enable an increasing number of new application deployments. A new effort in this direction is NAT traversal through tunneling NATTT [5]. Mobile IPv6 (MIPv6) [15] is a protocol developed as a subset of Internet Protocol veMyon 6 (IPv6) to support mobile connections. MIPv6 [6] allows a mobile node to transparently maintain connections while moving from one subnet to another [9]. 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. Dual Stack Mobile IPv6 (DSMIPv6) is an extension of MIPv6 to support mobility of devices irrespective of IPv4 [7] and IPv6 [8] network. The impact to IPv4, which changes IP address semantics, provide 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 [10]. NEPL (NEMO Platform for Linux) [11] is a freely available implementation of DSMIPv6 for Linux platform. The original NEPL release was based on MIPL (Mobile IPv6 for Linux) [12]. 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 and how NAT behave in common protocol like TCP [13] and UDP [14].

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
mn.c, ha.c, xfrm.c, mn.h, ha.h, xfrm.h, nat.h, bcache.h, bul.h.

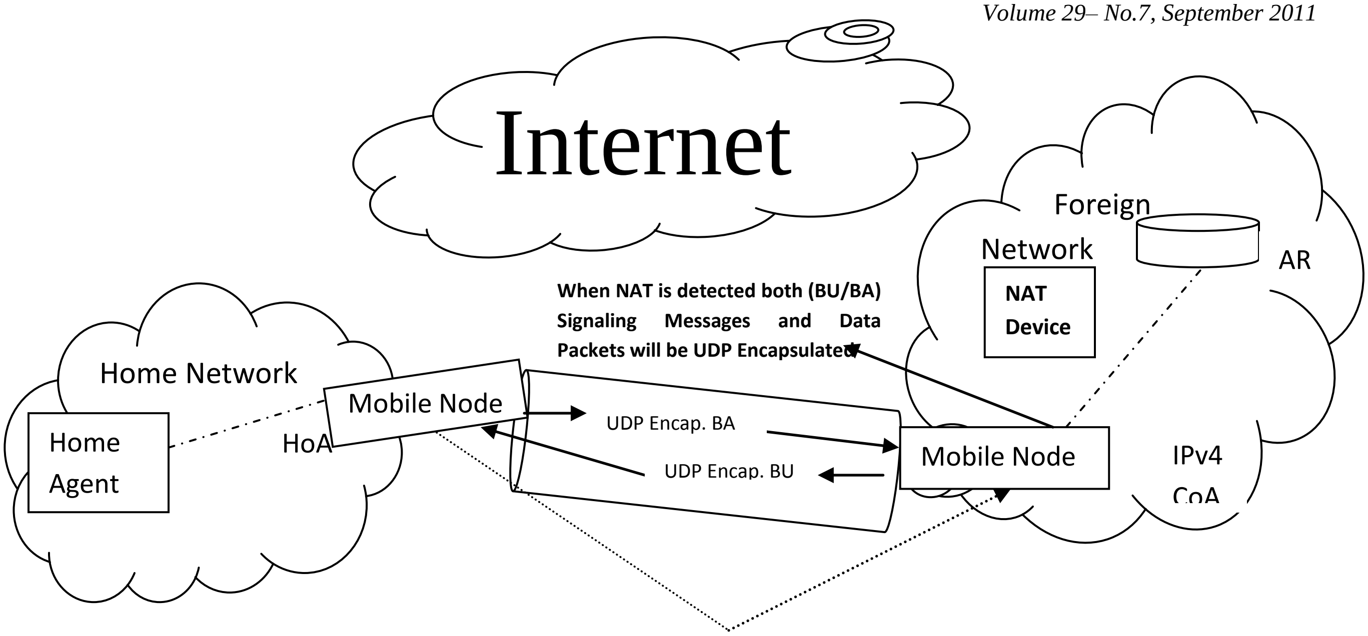

Fig 1: NAT Detection and Traversal Modules.

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 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.

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:

```
struct  encap_info {
        struct  in_addr 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:

```
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:

```
struct xfrm_selector{
        xfrm_address_t    daddr;
        xfrm_address_t    saddr;
        __be16   dport;
        __be16   dport_mask;
        __be16   sport;
        __be16   sport_mask;
        __u16    family;
```

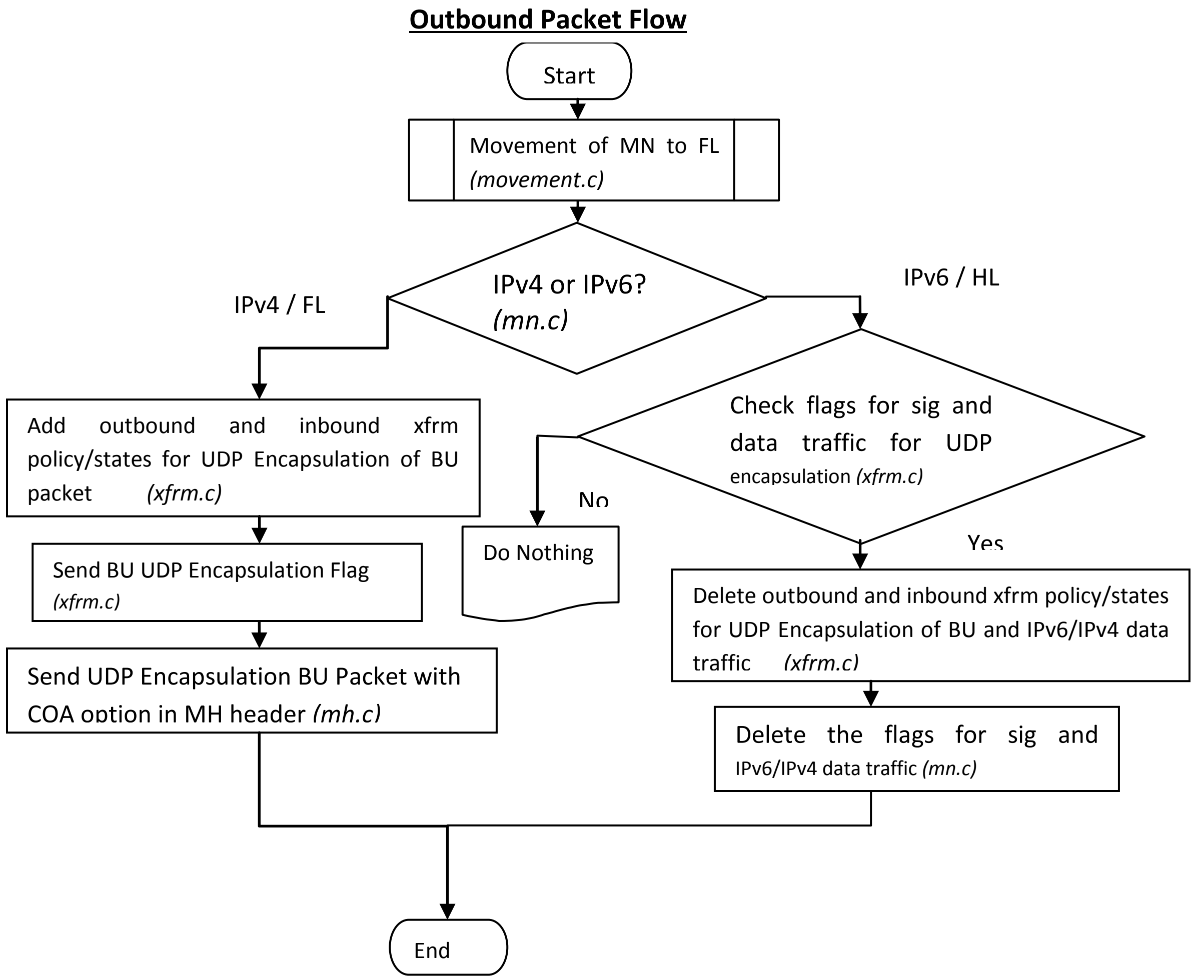

Fig. 2: Outbound Packet Flow in Mobile Node.

```
__u8      prefixlen_d;
          __u8      prefixlen_s;
          __u8      proto;
          int       ifindex;
          uid_t     user;};
```

4. struct xfrm_user_tmpl:

Description:

Used to create template for xfrm policy for UDP encapsulation.

File: linux-2.6.28.2/include/linux/xfrm.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                   ealgos;
          __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/xfrm.h

Code Snippet:

```
struct xfrm_encap_tmpl {
          __u16               encap_type;
          __be16              encap_sport;
          __be16              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 Xfrm 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 tq_elem tqe; /* Timer queue entry */
struct in6_addr peer_addr;          /* CN / HA address */
struct in6_addr hoa; 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 */
struct timespec lastsent;
```

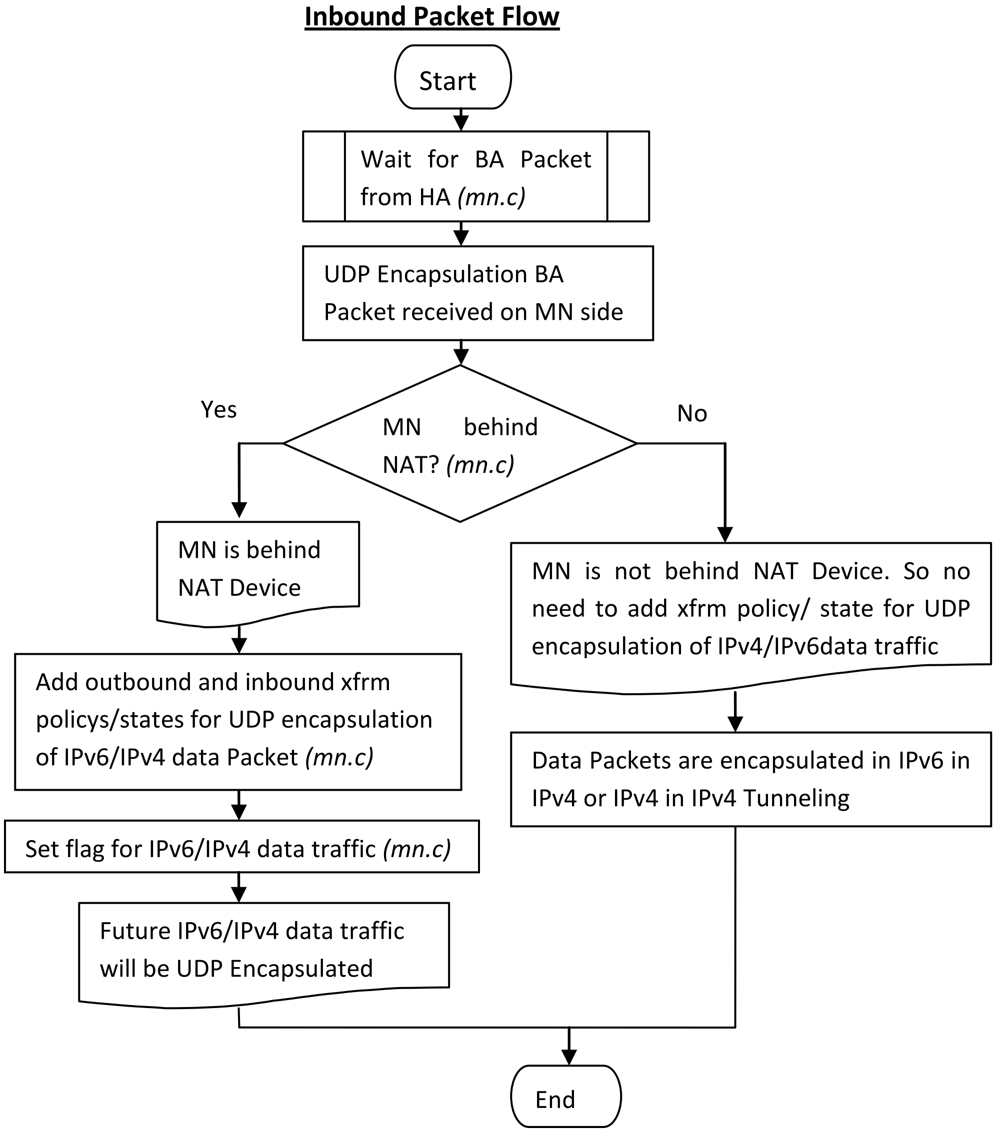

Fig 3. Inbound Packet in Mobile Node.

```
struct timespec lifetime;          \* lifetime sent in this BU */
struct timespec delay;             /* call back time in ms*/
struct timespec expires;           /* Absolute time for timer expire */
struct timespec hard_expire;  /* Absolute bulentry expiry time */
int consecutive_resends;      /* Number of consecutive BU's
resent*/  int8_t coa_changed;
uint8_t wait_ack                   ;/* WAIT / READY */
uint8_t xfrm_state;
uint8_t use_alt_coa;          /* Whether to use alt. CoA option */
uint8_t dereg;                     /* for calculating BSA key */
uint8_t do_send_bu;                /* send bu / not send bu */
uint8_t behind_nat;           /* whether a nat was detected */
uint8_t udp_encap;                 /* doing UDP encap */
                       /* Information for return routability */
struct retrout_info rr;
uint8_t Kbm[HMAC_SHA1_KEY_SIZE];
void (* callback)(struct tq_elem *);
void (*ext_cleanup)(struct bulentry *);};
```

7. struct bcentry:

Description:

This structure stores information about Binding Cache Entry. The members of this structure are used to set Xfrm policy/states for UDP encapsulation in Home Agent side.

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

Code Snippet:

```
struct bcentry{
Struct in6_addr our_addr; /* Address to which we got BU */
Struct in6_addr peer_addr;/* Mobile Node home address IPv6 */
Struct in_addr peer_addr4;           /* MN home address IPv4 */
Struct in6_addr old_coa;             /* Previous care-of address */
Struct in6_addr coa                  ;/* MN care-of address */
struct timespec add_time;   /* When was the binding added or
modified */ struct timespec lifetime;          /* lifetime sent
in this BU in seconds */ struct encap_info nat_info;     /*
Information for NAT traversal */      uint8_t      behind_nat/*
```

whether a nat was detected */uint16_t seqno;/* sequence number of the latest BU */
uint16_t flags; /* BU flags */
uint16_t nonce_coa;
uint16_t nonce_hoa;

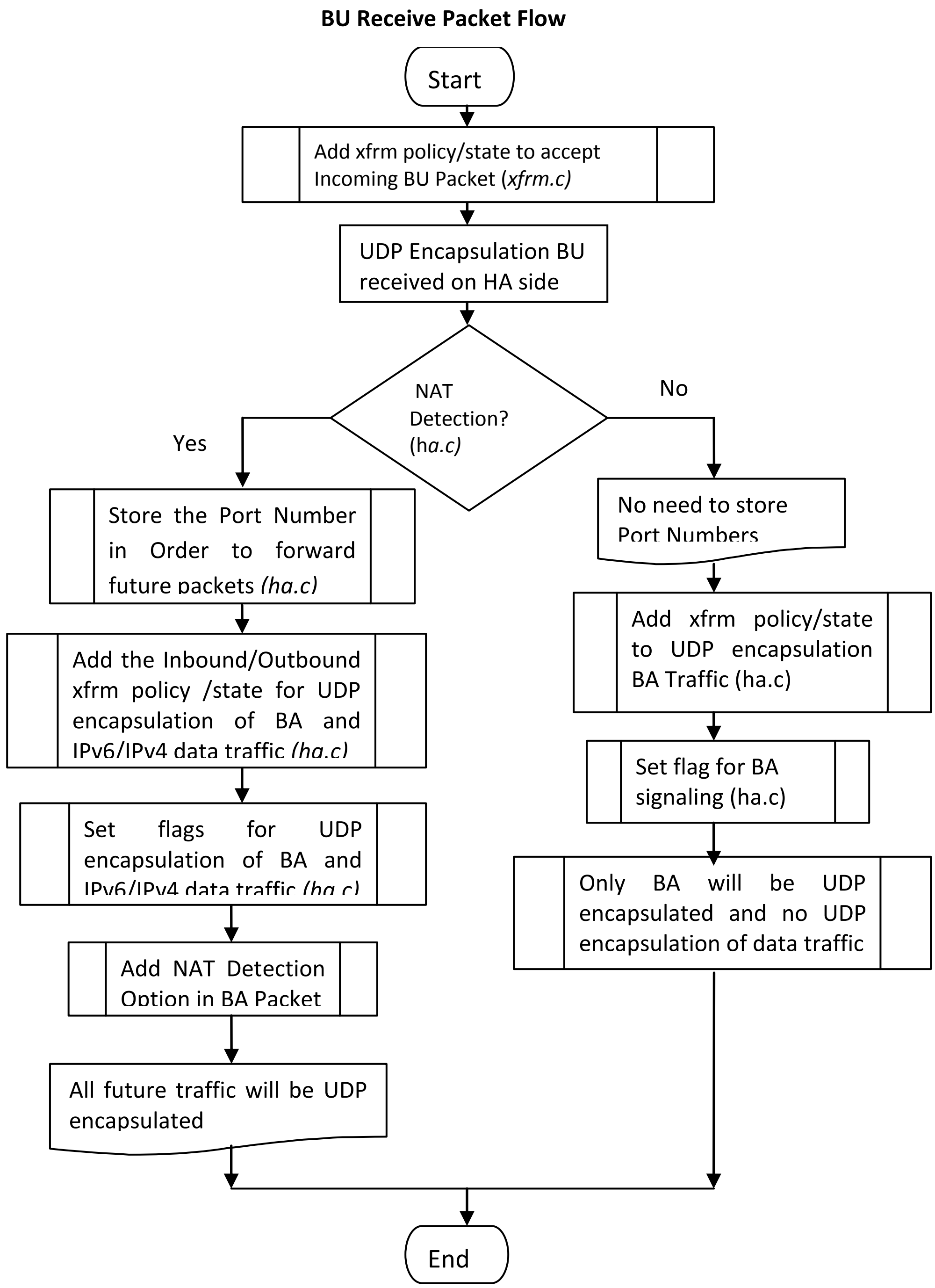

Fig 4. BU Receive Packet flow in Home Agent.

uint16_t type; /* Entry type */
uint16_t nemo_type; /* NEMO registration type */
int unreach; /* ICMP dest unreach count */
int tunnel; /* 6/6 or 6/4 tunnel interface index */
int tunnel4; /* 4/4 or 4/6 tunnel interface index */
int link; /* Home link interface index */

```
int id;                                            /* For testing */
                          /* Following fields are for internal use only */
struct timespec br_lastsent;                      /* BR ratelimit */
int br_count;                                     /* BR ratelimit */
pthread_rwlock_t lock;                        /* Protects the entry */
struct tq_elem tqe;                 /* Timer queue entry for expire */
```

```
 uint8_t xfrm_state; /* MY: status of xfrm state for UDP
encapsulation in HA */
void (*cleanup)(struct bcentry *bce);      /* Clean up bce data */
struct list_head mob_net_prefixes; Mobile network    prefixes
v6*/Struct net_prefix4 *mob_net_prefixes4;/*Mobile network
prefixes v4*/};
```

Cleanup Flow

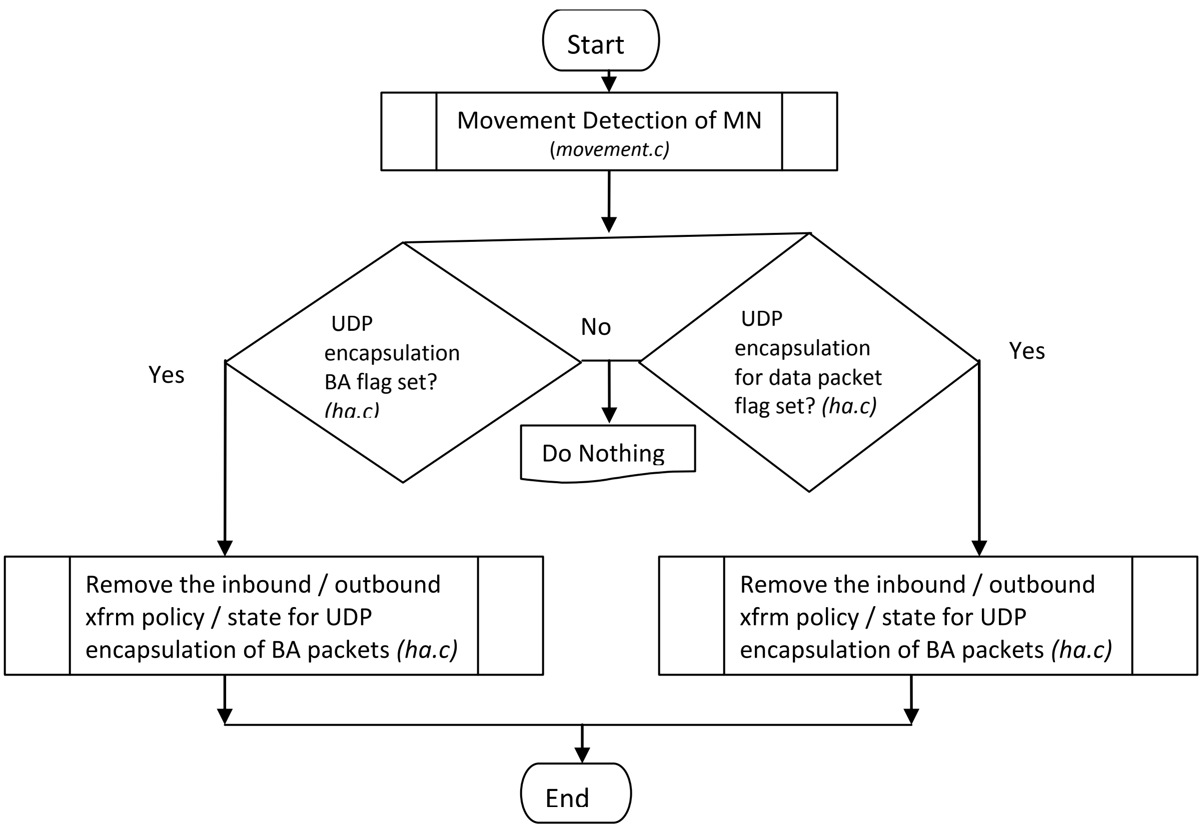

Fig 5. Cleanup flow in Home Agent.

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/states 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/states 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/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. 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

Function	Description	Input Parameter	Return Value 0	Caller	file
xfrm_state_encap_del	for deletion of UDP encapsulation xfrm state	proto: Protocol sel: IPv4 selectors spi: Security parameter index		xfrm_cn_cleanup xfrm_udp_encap_delete udpencap_receive_traffic_end udpencap_encap_out_traffic_end udpencap_encap_out_IPv4_traffic_end	Xfrm.c
xfrm_state_encap_del	for deletion of UDP encapsulation xfrm state	proto: Protocol sel: IPv4 selectors spi: Security parameter index		xfrm_cn_cleanup xfrm_udp_encap_delete udpencap_receive_traffic_end udpencap_encap_out_traffic_end udpencap_encap_out_IPv4_traffic_end	Xfrm.c
xfrm_del_bule_dsmip	Routine to delete states and policies related to UDP encapsulation for the BULE	bule: BUL structure		xfrm_del_bule	Xfrm.c
xfrm_pre_bu_add_bule	This routine is called before sending BU; MN should insert UDP encapsulation policy/state only for BU/BA	bule: BUL structure	No return type	pre_bu_bul_update	Xfrm.c
mn_recv_ba	is called, when MN receives BA from HA. From this routine the Xfrm policies/states for UDP encapsulation are added, when NAT is detected	mh: Header len: mh Length in: in6_addr_bundle structure iif: interface index	none	This routine is handler of type mh_handler. It is called when BA is received on MN side by HA	Xfrm.c
ha_udpencap_encap_traffic_end	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	bce: binding cache	In case of error return integer value less than 0	home_cleanup	Ha.c
ha_udpencap_encap_traffic_start	to add policy/state for UDP encapsulation of BA, IPv6 and IPv4 data packets When BU is UDP Encapsulated, BA is also UDP encapsulated. I NAT is detected, all future IPv6/IPv4 data traffic is also UDP encapsulated.	bce: binding cache	In case of error return integer value less than 0	ha_recv_bu_worker	Ha.c
udpencap_encap_out_IPv4_traffic_start	to add UDP encapsulated xfrm policy/state for IPv4 data traffic, when NAT is detected	Same as that of Below function	In case of error return integer value less than 0	ha_udpencap_encap_traffic_start mn_recv_ba	Xfrm.c

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

Function	Description	Input Parameter (in case of error return<0)	Caller	file
udpencap_encap_out_IPv4_traffic_end	to delete UDP encapsulated xfrm policy/state for IPv4 data traffic, when NAT is detected	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,dir: direction,spi: Security parameter index	ha_udpencap_encap_traffic_end xfrm_del_bule_dsmip	Xfrm.c
udpencap_encap_out_traffic_start	to install a state and policy to encapsulate some kind of traffic into IPv4/UDP.	local: local IPv6 IP,lpreflen: length of local IP,dest: destination IPv6 IP,dpreflen: length of destination IP,proto: protocol,type: MH header type,/* Outer ip and UDP */,src: Source IP,sport: Source IP,dst: destination IP,dport: destination port,/* Policy */,prio: priority,dir: direction,spi: Security parameter index	ha_udpencap_encap_traffic_start mn_recv_ba	Xfrm.c
udpencap_encap_out_traffic_end	to remove state and policy installed in previous function	local: local IPv6 IP,lpreflen: length of local IP,dest: destination IPv6 IP,dpreflen: length of destination IP,proto: Protocol,type: MH header type,/* Outer ip and UDP */,src: Source IP,dst: destination IP,dir: direction,spi: Security parameter index	ha_udpencap_encap_traffic_end xfrm_del_bule_dsmip	Xfrm.c
xfrm_state_encap_add	to add the XFRM states for IPv4/UDP encapsulation	sel: IPv6 selectors,proto: protocol tmpl: template,update: add new SA or update old one,flags: flags set v4: IPv4 selectors,spi: Security parameter index	xfrm_cn_init xfrm_pre_bu_add_bule udpencap_encap_out_traffic_start udpencap_receive_traffic_start udpencap_encap_out_IPv4_traffic_start	Xfrm.c

6. REFERENCES

1. Y.Rekhter et al, "Address allocation for private Internets" RFC 1918, 1996.
2. J. Rosenberg et al., "STUN: Simple Traversal of User Datagram Protocol (UDP) through Network Address Translators (NATs)," RFC 3489, 2003.
3. J. Rosenberg, R. Mahy, and P. Matthews, "Traversal Using Relays around NAT (TURN)," draft-ietf-behave-turn-08, 2008.
4. C. Huitema, "Teredo: Tunnelling IPv6 over UDP through Network Address Translations (NATs)," RFC 4380, 2006.
5. E. Osterwell et. Al. "NAT traversal through tunnelling" www.cs.arizona.edu/NAT.
6. G. Tsirtsis, Qualcomm; H. Soliman, Elevate Technologies; "Dual Stack Mobility", [RFC 4977], August 2007
7. Perkins, C., "IP Mobility Support for IPv4", RFC 3344, August 2002.
8. Johnson, D., Perkins, C., and J. Arkko, "Mobility Support in IPv6", RFC 3775, June 2004.
9. H. Soliman, Ed., Elevate Technologies; Mobile IPv6 Support for Dual Stack Hosts and Routers draft-ietf-mext-nemo-v4traversal-06.txt, November 3, 2008.
10. K.L.Bansal, Chaman Singh, "Dual Stack Implementation of Mobile IPv6 Software Architecture", IJCA- Volume 25, No 9, July 2011.
11. NEPL (NEMO Platform for Linux) how to, June 24th, 2009.
12. MIPL (Mobile Ipv6 for Linux), how to, 2004-4-20.
13. S. Guha, K. Biswas, B. Ford, S. Sivakumar, and P. Srisuresh. NAT Behavioral Requirements for TCP. RFC 5382, Internet Engineering Task Force, October 2008.
14. F. Audet and C. Jennings. Network Address Translation (NAT) Behavioral Requirements for Unicast UDP. RFC 4787, Internet Engineering Task Force, January 2007.
15. Arkko, J., Devarapalli, V. and F. Dupont, RFC 3776, June 2004. "Using IPsec to Protect Mobile IPv6 Signaling Between Mobile Nodes and Home Agents".

7. AUTHORS 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.