INTERNET DRAFT S. Bandyopadhyay
draft-shyam-site-multi-28.txt July 26, 2016
Intended status: Experimental
Expires: January 26, 2017

Solution for Site Multihoming in a Real IP Environment
draft-shyam-site-multi-28.txt

Abstract

This document provides a solution for Site Multihoming of stub networks in a real IP environment. Each user interface in a customer network may have as many global unicast addresses as many service providers it will be connected with. Users can establish multiple connections through different service providers simultaneously. Customer networks can maintain private address space to communicate within its users. Customer networks can provide IP mobility services as well.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 26, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction

Based on the definition of "multihoming" as stated in RFC3582[1],

"A "multihomed" site is one with more than one transit provider. "Site-multihoming" is the practice of arranging a site to be multihomed."

This is a general solution for site multihoming of stub networks in a real IP world irrespective of the framework supported by the service provider network. The solution is applicable to any customer network that receives globally unique IP addresses for all of its nodes and communicates with the rest of the world without the help of NAT[15]. It is applicable to any version of IP, i.e. IPv4, IPv6 or any new generation of IP that may emerge by removing the drawbacks associated with IPv6[7]. Within a provider assigned address space, each customer network will possess as many global unicast address space as many service providers it gets connected with. So, an user interface of a host may have as many global unicast addresses as many service providers it will be connected with.

Users can maintain multiple connections through multiple service providers simultaneously. A customer network can maintain private IP addresses to communicate within its users. Communication using private IP is restricted to private IP space for the sake of privacy. Customer networks can provide IP mobility support as well.

There are many variants of UNIX systems (as well as real time operating systems) which make use of BSD source code for their implementation of TCP/IP stack. The solution given below highlights the changes required with the BSD release 4.4 source code with the notations used by IPv4. It addresses issues relevant to IPv6 wherever applicable. All other implementations of TCP/IP have to be updated in the similar manner.

In this document the term "default router" will refer to the customer edge (CE) router that communicates with the provider network. Also the term "intermediate routers" will refer to all the routers apart from the CE routers.

2. Solution for site multihoming

RFC1122[2] made an extensive study related to different aspects of multihoming. Some of the requirements suggested in that document related to UDP and the application layer were avoided for multihomed hosts in a connected network with a single gateway to reach the outside world. This was achieved by the implementation of TCP/IP by making sure that the interface address of an outgoing packet gets selected based on the route to be followed by the destination address. This criterion holds good in a connected environment with a single gateway to reach the outside world. Once more than one gateway comes into play to reach the outside world, either routing table of the entire world has to be brought in or needs some enhancements within the existing system to make the things work.

Whenever a customer network gets service from more than one service provider, the customer network can be viewed as having multiple source-id (user-id) space. Each of these IP domain gets connected to different service providers through different routers. So each interface of customer network may have as many global unicast addresses as many service providers it is connected with. Number of routing entries in the routing table will (roughly) become a multiple of IP domains that it supports. Communication between any two hosts within the customer network will follow the traditional routing mechanism. In order to provide multihoming services it is needed that a host computer always forwards packets to the customer edge router associated to the same IP domain while communicating to someone in the outside world. i.e. if the interface of a host computer H receives an IP address 'addr1' and 'addr2' from two service providers P1 and P2 which are connected through routers R1 and R2 respectively, host H has to forward a packet to R1 while using its IP address as 'addr1' in order to send packets to the outside world. So, host computers as well as the intermediate routers have to use default routing based on the source domain of the source address in the IP header.

In order to achieve this, host computers as well as intermediate routers need to have information related to its IP domain (net address/net mask) and the associated default router for all of its IP domains. They need to have a route entry per IP domain for all of its default routers. These information should be uploaded at the system start up time.

Routing of IP packets (in the ip_output module of the hosts and in the ip_forwarding module of the intermediate routers) need to be modified in the following manner.

If destination address of a packet falls outside of its IP domains, it has to be forwarded to the default router based on the domain that the source address belongs to.

If destination address of the IP header falls within any one of its IP domains, usual routing mechanism has to be followed.

If customer network maintains private IP domain, communication using private IP has to be restricted within private IP space.

UDP (or RAW) based servers that need to support multiple clients simultaneously need to respond to a client's request with the same source address that the client had specified as the destination address. In order to satisfy this, system needs to introduce two system calls along with the existing system calls (i.e. read, write, send, sendto, recv, recvfrom)

```
ssize_t recvwithdstaddr (int sockfd, char *buf, size_t nbytes,
    int flags, struct sockaddr *from, socklen_t *fromlen,
    struct sockaddr *fromcladdr, socklen_t *fromcladdrlen,
    struct sockaddr *dst, socklen_t *dstlen,
    struct sockaddr *dstcladdr, socklen_t *dstcladdrlen);
```

'recvwithdstaddr' receives data with destination address as specified by the sender. It is similar to 'recvfrom' with the additional field 'dst' related to the address of the receiving interface of the host. 'fromcladdr' and 'dstcladdr' will hold the values of co-located care-of addresses (see section 2.2) of source and destination if they happen to be mobile.

```
ssize_t sendwithsrcaddr (int sockfd, char *buf, size_t nbytes,
    int flags, struct sockaddr *to, socklen_t tolen,
    struct sockaddr *dstcladdr, socklen_t dstcladdrlen,
    struct sockaddr *src, socklen_t srclen,
    struct sockaddr *srccladdr, socklen_t srccladdrlen);
```

'sendwithsrcaddr' sends data specifying the source address of the

outgoing interface of the host. It is similar to 'sendto' with additional parameters related to source address. It behaves like 'sendto' if no address is specified for 'src'. 'srccladdr' and 'dstcladdr' will hold the values of co-located care-of addresses of source and destination.

All the UDP based servers that need to support multiple clients simultaneously, need to replace 'sendto' with 'sendwithsrcaddr' and 'recvfrom' with 'recvwithdstaddr'.

It has been expressed in several documents including RFC4291[3], that a single interface will possess multiple IP addresses in a real IP environment. In these cases, all the UDP servers have to be updated with the system calls 'sendwithsrcaddr' and 'recvwithdstaddr' even if a customer site gets attached to a single gateway to reach the outside world.

The same logic will apply to server applications with RAW sockets. Server applications that are TCP based should work in the usual manner.

2.1. Selection of source and destination address

If a source network is connected with 'n' service providers and the destination network is connected with 'm' service providers, there will be a possible 'm*n' combination of source-destination pairs for connection between source and destination. So, application program needs to select a source and destination address before initiating communication with the destination.

A system call needs to be introduced to get the source address based on the destination address. If application program needs to use the destination address directly, it needs to use this system call.

```
int getsrcaddr(int sockfd, struct in_addr *dst, struct in_addr *src);
```

It returns the number of source addresses that can be used. The addresses will be available from 'src', which is an array of type struct in_addr. The 'src' addresses will be available in sorted manner. Application program needs to use these source addresses from the top (i.e. the 0th) to establish connection with the destination. 'sockfd' is used to get the 'type of service' assigned. So, an application program needs to set its type of service before using this call.

Client applications need to use 'getsrcaddr' and 'bind' the source address before communicating with their peer.

Users may use name instead of IP address to reach the destination. The usual procedure is to use the system call 'gethostbyname' to resolve the destination address and then to use the same for communication. The destination may also be multihomed. In order to find out the best possible choice to reach the destination, another system call needs to be introduced.

```
struct hostent *gethostbynamewithsrcaddr(int sockfd, const char *name,
               int *ndst, struct addr_pair *dst);
```

where 'addr_pair' is defined as

```
struct addr_pair {
    struct in_addr src;
    struct in_addr dst;
};
```

'gethostbynamewithsrcaddr' takes 'name' and 'sockfd' as input parameters and finds out the best possible route to reach the destination. It returns the pointer to the 'hostent' structure as returned by 'gethostbyname' system call. The parameter 'ndst' gets the number of possible routes to be used and the corresponding source and destination addresses gets assigned to 'dst' in sorted manner. 'sockfd' is used to get the 'type of service' assigned. So, an application program needs to set its type of service before using this call.

2.1.1. Path selection

Paths are selected by sending RSVP messages from user to the PE routers using MPLS UNI[12] with the following changes in respective modules.

In order to transport a packet from one network to another, provider network sets up a LSP. In RSVP[10,11], resource reservation is receiver-initiated. In the Path message, the sending application construct Path message using RSVP SENDER_TSPEC and ADSPEC objects. The path properties of ADSPEC object gets modified by the network elements as the Path message moves from sender to receiver. The receiver makes use of SENDER_TSPEC and ADSPEC objects and forms FLOWSPEC object and sends back to the network element towards the sender. In order to make decision which path an application should select from multiple possible paths due to multihoming, ADSPEC object that was received by the receiver has to be passed back to the sender by appending them with the Resv message.

For best effort service, path is selected based on widest-shortest path approach, i.e. the path having the maximum effective available bandwidth with minimum NUMBER_OF_IS_HOPS. Effective available

bandwidth is calculated as

```
  bandwidth allocated to the customer
---------------------------------------- * AVAILABLE_PATH_BANDWIDTH
gross effective bandwidth allocated to customers

If (Effective available bandwidth > unused bandwidth
                                    allocated to the customer)
    Effective available bandwidth = unused bandwidth
                                    allocated to the customer.
```

When a Path message is sent from a user to the ingress PE router, for best-effort service the PE router sets up a LSP with the egress PE router and stores the path attributes with the ADSPEC objects if no LSP has already been created. The ingress PE router sends the path attributes (with AVAILABLE_PATH_BANDWIDTH set as Effective available bandwidth) to the sender. If ingress PE router finds an existing LSP for the destination node, it sends the path attributes associated to the LSP.

PE routers need to maintain a list of customers that have accessed the LSP with the last time of access. At the end of each RSVP refresh time, it needs to check the list and delete those entries whose last time of access exceeds the time period of RSVP refresh time. Gross effective bandwidth is calculated as the sum of bandwidths allocated to all the customers available in the list.

The above equation is applicable when communication takes place between global unicast/multicast addresses. In case of VPN, service providers allocate fixed bandwidth path between two customer locations. So, when communication takes place between private addresses actual unused bandwidth of that path has to be returned.

For Guaranteed bandwidth[14] and Controlled-Load service[13] path is selected with MINIMUM_PATH_LATENCY with minimum NUMBER_OF_IS_HOPS, also sender applications need to send PathTear messages for all the paths that are not selected.

A PE router will be in a different address space than the address space of the customer network. As hosts need not be aware of the PE routers, hosts need to send queries to the CE router to get the address of the PE router and store the same in their cache, the way it works with DNS.

2.1.2. Link failure and switch over to an alternate route

As stated in section 2.1, there are possible "m*n" routes. Client applications select any one of them for communication. If

communication fails due to link failure, it may be desirable to switch over to an alternate route (application programs must ensure that it conforms to the requirement of the application).

In reality link failure is a rare phenomenon; so detection of link failure should not become an overhead for the network. Fault gets detected first at the local site where the fault is associated with. Say, if CE-PE link fails, it is the CE router that comes to know about it at the beginning. So, the local site needs to take initiative for the switchover operation. When failure happens, system generates trap which triggers the operation for switchover to an alternate route.

The steps can be summarized as follows:

o When application program calls 'getsrcaddr' or 'gethostbynamewithsrcaddr' system finds out a list of possible "source-destination" pairs for communication.

o Systems stores all the possible routes with the protocol control bloc(PCB).

o Application program calls a system call "regroutestopeer" to register the list of possible routes to its peer after the 5 unit tuple gets established (say for TCP, this has to be called after client application calls 'connect'). This is required because failure may happen in the remote site too; also it tells the system to activate switchover operation if there is a link failure.

o When CE router detects failure of CE-PE link, it broadcasts an ICMP message ICMP_LINKFAILURE_CE_PE_LINK to all the hosts.

o On receiving ICMP_LINKFAILURE_CE_PE_LINK, system goes through the list of PCB and gets the list of applications for which it needs to start the switchover operation. For any such particular application, it prepares the list of possible routes for communication through the active links. It tries to set alternate route to its peer by sending ICMP message ICMP_LINKFAILURE_SET_ALT_ROUTE. It tries all the possible routes one after the other till the operation becomes successful.

o On receiving ICMP_LINKFAILURE_SET_ALT_ROUTE, peer host checks whether there is any application in the list of PCB where the request can be applicable. On finding the right PCB, it sets the alternate route and sends a message ICMP_LINKFAILURE_ALT_ROUTE_ESTABLISHED to its peer.

o On receiving ICMP_LINKFAILURE_ALT_ROUTE_ESTABLISHED, system sets

the alternate route and completes the operation of switchover.

So, it introduces an ICMP message of type ICMP_LINKFAILURE (type 41) with the following codes:

```
   ICMP_LINKFAILURE_CE_PE_LINK              1
   ICMP_LINKFAILURE_CE_FAILURE              2
   ICMP_LINKFAILURE_REG_ROUTES              3
   ICMP_LINKFAILURE_ROUTES_REGISTERED       4
   ICMP_LINKFAILURE_SET_ALT_ROUTE           5
   ICMP_LINKFAILURE_ALT_ROUTE_ESTABLISHED   6
   ICMP_LINKFAILURE_NOT_APPLICABLE          7
```

It introduces an element 'struct id_lf' inside union 'icmp_dun' of 'struct icmp' as follows:

```
struct icmp {
   u_char   icmp_type;   /* type of message, see below */
   u_char   icmp_code;   /* type sub code */
   u_short  icmp_cksum;  /* ones complement cksum of struct */
   union {
      u_char ih_pptr;   /* ICMP_PARAMPROB */
      struct in_addr ih_gwaddr;    /* ICMP_REDIRECT */
      struct ih_idseq {
         uint16_t   icd_id;    /* network format */
         uint16_t   icd_seq; /* network format */
      } ih_idseq;
      int ih_void;

      /* ICMP_UNREACH_NEEDFRAG -- Path MTU Discovery (RFC1191) */
      struct ih_pmtu {
         uint16_t ipm_void;     /* network format */
         uint16_t ipm_nextmtu;     /* network format */
      } ih_pmtu;

      struct ih_rtradv {
         u_char irt_num_addrs;
         u_char irt_wpa;
         u_int16_t irt_lifetime;
      } ih_rtradv;
   } icmp_hun; #define   icmp_pptr icmp_hun.ih_pptr
#define   icmp_gwaddr    icmp_hun.ih_gwaddr
#define   icmp_id        icmp_hun.ih_idseq.icd_id
#define   icmp_seq  icmp_hun.ih_idseq.icd_seq
#define   icmp_void icmp_hun.ih_void
#define   icmp_pmvoid    icmp_hun.ih_pmtu.ipm_void
#define   icmp_nextmtu   icmp_hun.ih_pmtu.ipm_nextmtu
#define   icmp_num_addrs icmp_hun.ih_rtradv.irt_num_addrs
#define   icmp_wpa  icmp_hun.ih_rtradv.irt_wpa
```

```
   #define   icmp_lifetime  icmp_hun.ih_rtradv.irt_lifetime
      union {
         struct id_ts {           /* ICMP Timestamp */
            uint32_t its_otime;   /* Originate */
            uint32_t its_rtime;   /* Receive */
            uint32_t its_ttime;   /* Transmit */
         } id_ts;
         struct id_ip  {
            struct ip idi_ip;
                        /* options and then 64 bits of data */
         } id_ip;
         struct id_lf {
            u_char ip_proto;          /* protocol TCP/UDP    */
            u_char n_routes;          /* number of routes    */
            struct in_addr inp_laddr, /* source address      */
                           inp_faddr; /* destination address */
            u_short inp_lport,        /* source port         */
                    inp_fport;        /* destination port    */
         } id_lf;
         struct icmp_ra_addr id_radv;
         u_int32_t id_mask;
         char     id_data[1];
      } icmp_dun; #define   icmp_otime     icmp_dun.id_ts.its_otime
   #define   icmp_rtime     icmp_dun.id_ts.its_rtime
   #define   icmp_ttime     icmp_dun.id_ts.its_ttime
   #define   icmp_ip        icmp_dun.id_ip.idi_ip
   #define   icmp_radv icmp_dun.id_radv
   #define   icmp_mask icmp_dun.id_mask
   #define   icmp_data icmp_dun.id_data };
```

'struct inpcb' of protocol control block includes three new fields 'inp_lf_n_routes', 'inp_lf_stat' and 'inp_lf_routes' as follows:

```
   struct inpcb {
      struct inpcb *inp_next, *inp_prev;  /* doubly linked list  */
      struct inpcb *inp_head;   /* pointer back to chain of inpcb's for
                                   this protocol */
      struct in_addr inp_faddr; /* foreign IP address */
      u_short inp_fport;        /* foreign port# */
      struct in_addr inp_laddr; /* local IP address */
      u_short inp_lport;        /* local port# */
      struct socket *inp_socket;/* back pointer to socket */
      caddr_t inp_ppcb;         /* pointer to per-protocol pcb */
      struct route inp_route    /* placeholder for routing entry */
      int inp_flags;            /* generic IP/datagram flags */
      struct ip inp_ip;         /* header prototype; should have more */
      struct mbuf *inp_options; /* IP options */
      struct ip_moptions *inp_moptions; /* IP multicast options */
```

```
      u_char inp_lf_n_routes; /* total number of link failure routes */
      u_char inp_lf_stat;     /* link failure status; TRUE/FALSE whether
                                 link failure operation has to be
                                 started or not */
      struct addr_pair *inp_lf_routes;
                              /* pointer to the array of lf routes */

   };
```

Details of the above operations are described below:

ICMP_LINKFAILURE_CE_PE_LINK

CE router detects link failure and sends this message to all the users in the network; The field 'icmp_gwadd' of 'struct icmp' holds the IP address of the PE router.

ICMP_LINKFAILURE_CE_FAILURE

CE router itself may fail. It gets detected by alternate CE router. CE routers send keep-alive messages between themselves at regular interval to detect this failure. The field 'icmp_gwadd' of 'struct icmp' holds the IP address of the faulty CE router.

ICMP_LINKFAILURE_REG_ROUTES

This message is used by the system call 'regroutestopeer' to register all the routes to its peer. The fields 'icmp_id' and 'icmp_seq' of 'struct icmp' holds the value of process id and sequence number of the message. The fields 'ip_proto', 'n_routes', 'inp_laddr', 'inp_faddr', 'inp_lport' and 'inp_fport' of 'struct id_lf' of union 'icmp_dun' of 'struct icmp' holds the values of protocol(TCP/UDP), total number of routes, source address destination address, source port and destination port respectively. Source address and destination address of all the routes are appended at the end of 'struct icmp'.

ICMP_LINKFAILURE_ROUTES_REGISTERED

This message is sent by the system to its peer on successful completion of registering the alternate routes to the PCB of the application by receiving the message ICMP_LINKFAILURE_REG_ROUTES from its peer. The same message received with ICMP_LINKFAILURE_REG_ROUTES is returned by changing the field 'icmp_code' to ICMP_LINKFAILURE_ROUTES_REGISTERED. If no match is found in the list of PCB, it sets 'icmp_code' to ICMP_LINKFAILURE_NOT_APPLICABLE.

The prototype of the system call 'regroutestopeer' is as follows:

int regroutestopeer(int sockfd)

It returns TRUE on successful completion, i.e. after receiving ICMP_LINKFAILURE_ROUTES_REGISTERED from its peer.

ICMP_LINKFAILURE_SET_ALT_ROUTE

This message is sent by a host to its peer after receiving ICMP_LINKFAILURE_CE_PE_LINK or ICMP_LINKFAILURE_CE_FAILURE. Information of the current active link (i.e. the link that has been failed just now) i.e. protocol id, source address, destination address, source port and destination port are set with the fields of 'struct id_lf' of 'struct icmp'. Source address and destination address of the desired alternate route is appended at the end of 'struct icmp'. Host sends this message to its peer through the active link using the addresses of the desired route.

ICMP_LINKFAILURE_ALT_ROUTE_REGISTERED

This message is sent by a host to its peer on successful completion of changing 'source address' and 'destination address' with the desired value of the alternate route to the PCB. The same message that was received with ICMP_LINKFAILURE_SET_ALT_ROUTE is returned by changing the 'icmp_code' field to ICMP_LINKFAILURE_ALT_ROUTE_REGISTERED. If 'source port', 'destination port' and 'protocol id' of incoming ICMP message matches any entry in the list of PCB and the fields 'source address' and 'destination address of the entry of PCB matches that of the alternate route of the incoming ICMP message, system needs to send ICMP_LINKFAILURE_ALT_ROUTE_REGISTERED too. If no match is found in the list of PCB, it sets 'icmp_code' to ICMP_LINKFAILURE_NOT_APPLICABLE.

2.2. Multihoming, IP Mobility and Provider Independent addressing

For a mobile node, its co-located care-of IP address[4] has to be bound to one of the IP addresses supported by the service providers (if mobile node advertises more than one address, the home agent will get confused, also there are other implications). Transport layer must ensure that the 'home address' gets tightly coupled with that particular IP address.

A mobile node in a foreign site will have all the IP addresses supported by the foreign site as well as its "Home Address". As the mobile node will also communicate with the outside world with its "Home Address", user should get a provision to choose its "Home Address" while initiating communication. If mobile node makes use of the address of foreign site for applications that do not need its

"Home Address" (say, accessing a web site) cost of communication will get reduced. This feature is useful when a mobile user is in a foreign site but remains within the same sphere of influence (say an user lives in one city but works in a different city which is in a different sphere of influence and likes to access web during his working hours).

If "Home Address" is selected for communication, the transport layer of the mobile node should use its care-of address as the source address and pass its "Home Address" as an option field in the stack. This is because multihoming expects the source address as the deciding factor for packet forwarding.

The IP address of a node with a provider independent address have to be mapped with one of the global unicast addresses. So for the purpose of multihoming whatever will be applicable to a mobile node will also be applicable to a node with provider independent address.

All the issues that need to be handled for IP mobility, provider independent addressing related to multihoming have been thoroughly discussed in section 4 of the architectural specification[7]. Please go through that section first before going through the rest.

2.2.1. IP Address Stacking

IP address stacking in IPv6 is performed with the approach introduced in section 6.4 of RFC6275[8] with slight modification. RFC6275 describes how to pass "Home Address" as well as co-located care-of address of the destination address if it happen to be mobile. The same approach has been extended to support IP address stacking for the source address and to support IP address stacking for both source address as well as destination address. The "Reserved" space in the type 2 routing header has been split into two parts; an one octet field to address the "Stacking Type" and the rest 3 octets are left as Reserved.

Stacking Type is interpreted as follows:

Stacking Type=0
   Source Address: Address of the sender.
   Destination Address: co-located care-of address of the receiver.
   Address 1: Home Address/PI Address of the receiver.
   Hdr Ext Len=2.

So, type 2 routing header for stacking type 0 will be as follows:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header   | Hdr Ext Len=2 | Routing Type=2|Segments Left=1|
```

```
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |Stacking Type=0|                Reserved                       |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    +                                                               +
    |                                                               |
    +       Address 1:Home Address/PI Address of the receiver       +
    |                                                               |
    +                                                               +
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

    Stacking Type=1
       Source Address: co-located care-of address of the sender.
       Destination address: Address of the receiver.
       Address 1: Home Address/PI Address of the sender.
       Hdr Ext Len=2.

So, type 2 routing header for stacking type 1 will be as follows:

```
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |  Next Header  | Hdr Ext Len=2 | Routing Type=2|Segments Left=1|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |Stacking Type=1|                Reserved                       |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    +                                                               +
    |                                                               |
    +       Address 1:Home Address/PI Address of the sender         +
    |                                                               |
    +                                                               +
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

    Stacking Type 2
       Source Address: co-located care-of address of the sender.
       Destination Address: co-located care-of address of the receiver.
       Address 1: Home Address/PI Address of the sender.
       Address 2: Home Address/PI Address of the receiver.
       Hdr Ext Len=4.

So, type 2 routing header for stacking type 2 will be as follows:

```
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |  Next Header  | Hdr Ext Len=4 | Routing Type=2|Segments Left=1|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |Stacking Type=2|                Reserved                       |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
   |                                                               |
   +                                                               +
   |                                                               |
   +       Address 1:Home Address/PI Address of the sender         +
   |                                                               |
   +                                                               +
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   +                                                               +
   |                                                               |
   +       Address 2:Home Address/PI Address of the receiver       +
   |                                                               |
   +                                                               +
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Next Header
   8-bit selector. Identifies the type of header immediately
   following the routing header. Uses the same values as the IPv6
   Next Header field [9].

Hdr Ext Len
   4 (8-bit unsigned integer); length of the routing header in 8-
   octet units, not including the first 8 octets.

Routing Type
   2 (8-bit unsigned integer).

Segments Left
   1 (8-bit unsigned integer).

Stacking Type
   2 (8-bit unsigned integer).

Reserved
   24-bit reserved field. The value MUST be initialized to zero by
   the sender, and MUST be ignored by the receiver.

Address 1
   Home Address/PI Address of the sender.

Address 2
   Home Address/PI Address of the receiver.

IP address stacking in IPv4 is performed by introducing new IP option
under the option class "Datagram or Network Control", i.e. 0. The
option number is 16. The CODE(144) field is followed by one octet

field "Stacking Type" followed by two octet reserved space (NULL) as padding followed by the address fields based on the Stacking Type.

```
Stacking Type is interpreted as follows:
Stacking Type=0
   Source Address: Address of the sender.
   Destination Address: co-located care-of address of the receiver.
   Address 1: Home Address/PI Address of the receiver.
   Header Length:7
```

Format of IP address stacking option with stacking type 0 in the IP header will be as follows:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  CODE(144)    |Stacking Type=0| Reserved                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+       Address 1:Home Address/PI Address of the receiver       +
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
Stacking Type=1
   Source Address: co-located care-of address of the sender.
   Destination Address: Address of the receiver.
   Address 1: Home Address/PI Address of the sender.
   Header Length:7
```

Format of IP address stacking option with stacking type 1 in the IP header will be as follows:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  CODE(144)    |Stacking Type=1| Reserved                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+       Address 1:Home Address/PI Address of the sender         +
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
Stacking Type=2
   Source Address: co-located care-of address of the sender.
   Destination Address: co-located care-of address of the receiver.
   Address 1: Home Address/PI Address of the sender.
   Address 2: Home Address/PI Address of the receiver.
   Header Length:8
```

Format of IP address stacking option with stacking type 2 in the IP header will be as follows:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  CODE(144)    |Stacking Type=2| Reserved                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+       Address 1:Home Address/PI Address of the sender         +
```

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+       Address 2:Home Address/PI Address of the receiver       +
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

2.3. Implementation aspects

Following changes are expected with the source code of BSD.

Introduce ip_domain structure and some parameters as follows:

```
struct ip_domain {
    struct in_addr net_addr;
    struct in_addr net_mask;
    struct in_addr def_router;
};
#define MAX_IP_DOMAINS    16
short num_ipdomains;
struct ip_domain *ipdomain[MAX_IP_DOMAINS];
```

If customer network maintains private IP domain (along with the user-id space provided by the service providers) and expects its communication to be confined within its own space, 'def_router' has to be set as NULL.

Upload IP domain information for all of its IP domains during system start up. These domain information can be uploaded through router advertisement or through DHCP. The domain information should contain the next hop address to reach the corresponding default router as well.

There has to be a provision to upload these information through 'sysctl' to configure them manually.

Three new 'sysctl' routines have to be introduced under the 'ip' node of the MIB tree (i.e. under CTL_NET, PF_INET, IPPROTO_IP) IPCTL_NUM_DOMAINS, IPCTL_DOMAIN and IPCTL_USE_HOMEADDR (applicable for mobile node). Both IPCTL_NUM_DOMAINS and IPCTL_USE_HOMEADDR are of type CTLTYPE_INT and IPCTL_DOMAIN is of type CTLTYPE_NODE. Using 'sysctl' IPCTL_NUM_DOMAINS has to be configured first. Configuration of IPCTL_NUM_DOMAINS has to populate IPCTL_NUM_DOMAIN entries of nodes under IPCTL_DOMAIN and for each of these nodes three MIB attributes DOMAIN_NET_ADDR, DOMAIN_NET_MASK and DOMAIN_DEF_ROUTER (each of type CTLTYPE_NODE) has to be allocated.

All the routers as well as hosts that are having interfaces connecting to multiple subnets (see section 2.4) need to be configured through 'sysctl'.

Mobile users should get provision to change IPCTL_USE_HOMEADDR attribute dynamically.

Add a route entry for all the default routers during system start up.

2.3.1. Processing of system call 'getsrcaddr'

Introduce a routine (say 'getendpointaddr') that will find out a list of source-destination addressees sorted in order based on sending Path messages between a list of source addresses to a list of destination addresses. The routine should select the service type based on the type of service field (which can be obtained by calling 'getsockopt' with the socket id 'sockfd' passed as a parameter).

System call 'getsrcaddr' has to be processed in the following manner:

```
If destination address of the IP packet falls outside of its
IP domains {
   If user has selected its "Home Address" {
      /*Applicable to IP mobility/PI address*/
      return its "Home Address";
   }
   If destination address is from private address space {
      if the host is having only one interface {
         call 'getendpointaddr' for the destination address
         with all the private addresses assigned to it. get source
         address based on the output of 'getendpointaddr'.
      }
      else {
         for all the default routers {
            use 'rtalloc' to get the next hop address for the
            default router.

            select source address based on the outgoing interface
            'ia', and the private address associated with the default
            router.
         }
         call 'getendpointaddr' for the destination address with
         all the private addresses selected above. get source
         address based on the output of 'getendpointaddr'.
      }
   }
   else {
      if the host is having only one interface {
         call 'getendpointaddr' for the destination address
         with all the global unicast addresses assigned to it. get
         source addresses based on the output of 'getendpointaddr'.
      }
```

```
         else {
            for all the default routers {
               use 'rtalloc' to get the next hop address for the
               default router.

               select source address based on the outgoing interface
               'ia', and the global unicast address associated
               with the default router.
            }
            call 'getendpointaddr' for the destination address with
            all the global unicast addresses selected above. get source
            address based on the output of 'getendpointaddr'.
         }
      }
   }
   else { /* i.e. destination address is inside its IP domains */
      use 'rtalloc' to get the next hop address for the
      destination address.

      if destination address is a link local address {
         select source address based on the outgoing interface
         and the link local address assigned to it.
      }
      else {
         select source address based on the outgoing interface
         and the domain that the destination address belongs to.
      }
   }
```

2.3.2. Processing of 'gethostbynamewithsrcaddr'

System call 'gethostbynamewithsrcaddr' has to be processed in the following manner:

This is an enhancement of the system call 'gethostbyname'. 'gethostbyname' calls three routines that performs host table search, NIS search and DNS search. Once name is resolved, following additions are expected to resolve source-destination pair.

```
   If 'hostent' structure contains addresses which are inside its IP
   domains {
      if 'hostent' structure contains a private address {
         Assign destination address as a private address
         contained in 'hostent';

         use 'rtalloc' to get the next hop address for the
         destination address.
```

```
         select source address based on the outgoing interface
         and the domain that the destination address belongs to.
      }
      else {
         Select a global unicast address contained in 'hostent' for
         destination address.

         use 'rtalloc' to get the next hop address for the
         destination address.

         select source address based on the outgoing interface
         and the domain that the destination address belongs to.
      }
   }
   else {
      if 'hostent' structure contains private address {
         if host is having only one interface {
            call 'getendpointaddr' with all the private addresses
            returned by 'gethostbyname' as destination addresses
            with all the private addresses assigned to it as host
            addresses and return source and destination addresses
            based on its output.
         }
         else {
            for all the default routers {
               use 'rtalloc' to get the next hop address for the
               default router.

               select source address based on the outgoing interface
               'ia', and the private address associated with the default
               router.
            }
            call 'getendpointaddr' with all the private addresses
            returned by 'gethostbyname' as destination addresses
            with all the selected private addresses above as host
            addresses. get source and destination addresses
            based on the output of 'getendpointaddr'.
         }
      }
      else {
         if user has selected its "Home Address" {
         /*Applicable to IP mobility/PI address*/

            Call 'getendpointaddr' with all the addresses returned
            by 'gethostbyname' as destination addresses with the
            "Home Address" and return source and destination
            addresses based on its output.
         }
```

```
         if host is having only one interface {
            Call 'getendpointaddr' with all the addresses returned
            by 'gethostbyname' as destination addresses with all of its
            global unicast addresses as source addresses and return
            source and destination addresses based on its output.
         }
         else {
            for all the default routers {
               use 'rtalloc' to get the next hop address for the
               default router.

               select source address based on the outgoing interface
               'ia', and the global unicast address associated
               with the default router.
            }
            call 'getendpointaddr' with all the global unicast
            addresses returned by 'gethostbyname' as destination
            addresses with all the selected global unicast addresses
            as host addresses. get source and destination addresses
            based on the output of 'getendpointaddr'.
         }
      }
   }
```

2.3.3. Changes required in ip_output and ip_forwarding modules

Execute the following steps in the 'ip_output' routine of the IP stack before it calls 'rtalloc' for route look up.

```
   If destination address of the IP packet falls outside of its
   IP domains {
       get def router address based on the IP domain
       the source address belongs to.

       use 'rtalloc' to get the next hop address for the def router.

       Forward the packet to the next hop.
   }
   else { /* i.e. destination address is inside its IP domains */
      follow the usual procedure to forward packets
   }
```

In BSD, the 'ip_forwarding' routine calls 'ip_output'; so it should be left as it is.

2.3.4. Processing of protocol input routines and socket IO system calls

Protocol input routines need to locate the socket/process in the

usual manner with the 5 unit tuple (i.e. protocol, source address, source port, destination address, destination port).

When a packet is received by a mobile node (at a foreign site), it can be received in two modes. It can be received directly from the correspondent node with the 'destination address' as the co-located care-of address and its home address in the IP stack (see section 4.1 of RFC6275[8]). In the second mode the packet can be received via the home agent using IP over IP. Once the IP layer receives a packet with IP over IP, it is supposed to strip off the outer header before passing the packet to the protocol input routine. In this case packet will be received by the protocol input routine with destination address as the home address of the mobile node with no information related to its care-of address. So, protocol input routine needs to check whether the destination address of the received packet belongs to any one of its IP domains. If it does not, it needs to find out the co-located care-of address by going through the interface list if it is not already found in the packet received. This information is needed by the TCP input routine while processing a SYN message. It is also needed by the UDP/RAW modules while processing the system call 'recvwithdstaddr'.

While processing the output routines like 'sendwithsrcaddr', 'sendto', UDP/RAW modules needs to check the parameters related to source address, source port, destination address, destination port, care-of address of the source, care-of address of the destination in the protocol control block. Parameters in the PCB should prevail over parameters passed by the system call while forming the IP packet.

2.4. Multihoming and VPN

For a corporate, that maintains multiple offices and communicates within themselves through private address space using VPN needs to distribute its entire private address space to all its site in a suitable manner. Each one of its offices will get multiple private address space where each of them will be associated with a particular link. Let us consider one of its offices gets connected to two providers P1 and P2 and gets address space as 'unicastNetAddr1'/'unicastNetMask1' and 'unicastNetAddr2'/'unicastNetMask2' respectively. It also gets assigned private address space from its corporate as 'privateDomainNetAddr1'/'privateDomainNetMask1' and 'privateDomainNetAddr2'/'privateDomainNetMask2' which will be associated with the CE routers CE1 and CE2 respectively.

All hosts as well as the intermediate routers will have four entries of ip_domain:

```
   1: 'net_addr = 'unicastNetAddr1'
      'net_mask = 'unicastNetMask1'
      'def_router = CE1
   2: 'net_addr = 'unicastNetAddr2'
      'net_mask = 'unicastNetMask2'
      'def_router = CE2
   3: 'net_addr' = 'privateDomainNetAddr1'
      'net_mask' = 'privateDomainNetMask1'
      'def_router' = CE1
   4: 'net_addr' = 'privateDomainNetAddr2'
      'net_mask' = 'privateDomainNetMask2'
      'def_router' = CE2
```

3. Security Consideration

This document provides a solution for site multihoming of stub networks. Information exchange between two hosts that is required for switchover operation as described in section 2.1.2. has to be executed through secured IP. As there are 'm*n' possible 'source-destination' address pair between two hosts, same keys have to be used for all 'source-destination' address combination. All the issues related to separation of locator and identifier that were addressed in RFC4218[5] are not applicable here but for common security related issues that any site may experience, one needs to consult with the "Site Security Handbook", RFC2196[6]. For issues related to IP Mobility, section 5 of RFC5944[4] has to be consulted.

4. IANA Consideration

This draft does not request any action from IANA.

5. Normative References

[1] J. Abley, B. Black, V. Gill, "Goals for IPv6 Site-Multihoming Architectures", RFC3582, August 2003.

[2] R. Braden, "Requirements for Internet Hosts -- Communication Layers", RFC1122, October 1989.

[3] R. Hinden, S. Deering, "IP Version 6 Addressing Architecture.", RFC4291, February 2006.

[4] C. Perkins, "IP Mobility Support for IPv4, Revised", RFC5944, November 2010.

[5] E. Nordmark, T. Li, "E. Nordmark, "Threats Relating to IPv6 Multihoming Solutions", RFC4218, October 2005.

[6] B. Fraser, "Site Security Handbook", RFC2196, September 1997.

[7] S. Bandyopadhyay, "An Architectural Framework of the Internet for the Real IP World" <draft-shyam-real-ip-framework-24.txt> (work in progress).

[8] C. Perkins, Ed., D. Johnson, J. Arkko, "Mobility Support in IPv6" RFC 6275, July 2011.

[9] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[10] L. Zhang, S. Berson, S. Herzog, S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205, September 1997.

[11] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC 3209, December 2001.

[12] G. Swallow, J. Drake, H. Ishimatsu, Y. Rekhter, "Generalized Multiprotocol Label Switching (GMPLS) User-Network Interface (UNI): Resource ReserVation Protocol-Traffic Engineering (RSVP-TE) Support for the Overlay Model", RFC 4208, October 2005.

[13] J. Wroclawski, "Specification of the Controlled-Load Network Element Service", RFC 2211, September 1997.

[14] S. Shenker, C. Partridge, R. Guerin, "Specification of Guaranteed Quality of Service", RFC 2212, September 1997.

6. Informative References

[15] P. Srisuresh, K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC3022, January 2001.

7. Author's Address

Shyamaprasad Bandyopadhyay
HL No 205/157/7, Kharagpur 721305, India
Phone: +91 3222 225137
e-mail: shyamb66@gmail.com