| | | |
|---|---|---|
| Behave WG | B. Huang | |
| Internet-Draft | H. Deng | |
| Obsoletes: 3338, 2767 | China Mobile | |
| (if approved) | T. Savolainen | |
| Intended status: Standards Track | Nokia | |
| Expires: April 22, 2011 | October 19, 2010 | |

TOC

**Dual Stack Hosts Using "Bump-in-the-Host" (BIH)**
**draft-ietf-behave-v4v6-bih-01**

## Abstract

This document describes the "Bump-In-the-Host" (BIH), a host based IPv4 to IPv6 protocol translation mechanism that allows a subset of applications supporting only IPv4 to communicate with peers that are reachable only with IPv6. A host may be connected to IPv6-only or dual-stack access network. Essentially BIH makes the IPv4 applications think they talk to IPv4 peers and hence hides the existence of IPv6 from those applications.

Acknowledgement of previous work

This document is an update to and directly derivative from Kazuaki TSHUCHIYA, Hidemitsu HIGUCHI, and Yoshifumi ATARASHI [RFC2767] and from Seungyun Lee, Myung-Ki Shin, Yong-Jin Kim, Alain Durand, and Erik Nordmark's [RFC3338], which similarly provides a dual stack host means to communicate with other IPv6 host using existing IPv4 appliations. This document combines and updates both [RFC2767] and [RFC3338].

The changes in this document reflect five components

> 1. Supporting IPv6 only network connections
>
> 2. IPv4 address pool use private address instead of the unassigned IPv4 addresses (0.0.0.1 - 0.0.0.255)
>
> 3. Extending ENR and address mapper to operate differently
>
> 4. Adding an alternative way to implement the ENR
>
> 5. Going for standards track instead of experimental/ informational

## Status of this Memo

This Internet-Draft will expire on April 22, 2011.

## Copyright Notice



---

## Table of Contents

---

## 1. Introduction

TOC

While IPv6 support is being widely introduced throughout the Internet, classes of applications are going to remain IPv4-only. This document describes a Bump-in-the-Host (BIH), successor and combination of Bump-in-the-Stack (BIS) [RFC2767] (Tsuchiya, K., HIGUCHI, H., and Y. Atarashi, "Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS)," February 2000.) and Bump-in-the-API (BIA) [RFC3338] (Lee, S., Shin, M-K., Kim, Y-J., Nordmark, E., and A. Durand, "Dual Stack Hosts Using "Bump-in-the-API" (BIA)," October 2002.) technologies, which enables accommodation of significant set of the legacy IPv4-only applications in the IPv6-world.

Bump-In-the-Host is not recommended to be used in double translation scenarios if the server is dual-stack enabled. The class of IPv4-only applications the described host-based protocol translation solution provides Internet connectivity over IPv6-only network access includes those applications that use DNS for IP address resolution and that do not embed IP address literals in protocol payloads. This includes essentially all DNS using legacy client-server model applications, which are agnostic on IP address family used by the destination, but not other classes of applications. The transition towards IPv6-only Internet is made easier by decreasing number of key applications that must be updated to IPv6.

BIH technique includes two major implementation options: a protocol translator between the IPv4 and the IPv6 stacks of a host or between the socket API module and the TCP/IP module. Essentially, IPv4 is translated into IPv6 at the socket API level or at the IP level.

When the BIH is implemented at the socket API layer, and IPv4 applications communicate with IPv6 peers, the API translator intercepts the socket API functions from IPv4 applications and invokes the IPv6 socket API functions to communicate with the IPv6 hosts, and vice versa.

When the BIH is implemented at the networking layer, the IPv4 packets are intercepted and converted to IPv6 using the IP conversion mechanism defined in SIIT [I-D.ietf-behave-v6v4-xlate] (Li, X., Bao, C., and F. Baker, "IP/ICMP Translation Algorithm," September 2010.). The translation has the same benefits and drawbacks as SIIT.

In order to support communication between IPv4 applications and the target IPv6 peers, pooled IPv4 addresses as defined in section 4.3 will be assigned through the extension name resolver.

The BIH can be used whenever an IPv4-only application needs to communicate with a peer reachable only with IPv6, independently of the address families supported by the access network. Hence the access network can be IPv6-only or dual-stack capable.

In the case BIH enabled host has a possibility to choose between IPv4-only path or path including IPv4 to IPv6 protocol translation, the host MUST select IPv4-only path. However, lacking IPv4-only path and on request BIH will attempt protocol translation also in the case a destination has IPv4 addresses in addition to IPv6.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.) .

This document uses terms defined in [RFC2460] (Deering, S. and R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification,” December 1998.) , [RFC2893] (Gilligan, R. and E. Nordmark, “Transition Mechanisms for IPv6 Hosts and Routers,” August 2000.) , [RFC2767] (Tsuchiya, K., HIGUCHI, H., and Y. Atarashi, “Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS),” February 2000.) and [RFC3338] (Lee, S., Shin, M-K., Kim, Y-J., Nordmark, E., and A. Durand, “Dual Stack Hosts Using "Bump-in-the-API" (BIA),” October 2002.).

## 2. Components of the Bump-in-the-Host

TOC

Figure 1 shows the architecture of the host in which BIH is implemented as socket API layer translator, i.e. as the original "Bump-in-the-API".

```
+----------------------------------------------+
| +------------------------------------------+ |
| |                                          | |
| |            IPv4 applications             | |
| |                                          | |
| +------------------------------------------+ |
| +------------------------------------------+ |
| |           Socket API (IPv4, IPv6)        | |
| +------------------------------------------+ |
| +-[ API translator]------------------------+ |
| | +-----------+ +---------+ +------------+ | |
| | | Ext. Name | | Address | | Function   | | |
| | | Resolver  | | Mapper  | | Mapper     | | |
| | +-----------+ +---------+ +------------+ | |
| +------------------------------------------+ |
| +--------------------+ +-------------------+ |
| |                    | |                   | |
| |    TCP(UDP)/IPv4   | |   TCP(UDP)/IPv6   | |
| |                    | |                   | |
| +--------------------+ +-------------------+ |
+----------------------------------------------+
```

**Figure 1: Architecture of the dual stack host using BIH at socket layer**

Figure 2 shows the architecture of the host in which BIH is implemented as network layer translator, i.e. as the original "Bump-in-the-Stack".

```
+-------------------------------------------------------------+
|  +-----------------------------------------------------+    |
|  | IPv4 applications                                   |    |
|  +-----------------------------------------------------+    |
|  +-----------------------------------------------------+    |
|  | TCP/IPv4                                            |    |
|  |   +-------------------------------------------------+    |
|  |   |  +-----------+    +---------+  +---------------+     |
|  |   |  | Extension |    | Address |  |  Translator   |     |
|  |   |  | Name      |    | Mapper  |  +---------------+     |
|  |   |  | Resolver  |    |         |  +---------------+     |
|  |   |  |           |    |         |  |     IPv6      |     |
|  +---+  +-----------+    +---------+  +---------------+     |
|  +-----------------------------------------------------+    |
|  |                Network card drivers                 |    |
|  +-----------------------------------------------------+    |
+-------------------------------------------------------------+
+-------------------------------------------------------------+
|                       Network cards                         |
+-------------------------------------------------------------+
```

**Figure 2: Architecture of the dual-stack host using BIH at network layer**

Dual stack hosts defined in RFC2893 [RFC2893] (Gilligan, R. and E. Nordmark, “Transition Mechanisms for IPv6 Hosts and Routers,” August 2000.) need applications, TCP/IP modules and addresses for both IPv4 and IPv6. The proposed hosts in this document have an API or network layer translator to communicate with other IPv6 hosts using existing IPv4 applications. The BIH translator consists of an extension name resolver, an address mapper, and depending on implementation either a function mapper or a protocol translator.

### 2.1. Function Mapper

TOC

Function mapper translates an IPv4 socket API function into an IPv6 socket API function, and vice versa.

When detecting IPv4 socket API function calls from IPv4 applications, function mapper intercepts the function calls and invokes new IPv6 socket API functions which correspond to the IPv4 socket API functions. Those IPv6 API functions are used to communicate with the target IPv6 peers. When detecting IPv6 socket API function calls triggered by the data received from the IPv6 peers, function mapper works symmetrically in relation to the previous case.

### 2.2. Translator

TOC

Translator translates IPv4 into IPv6 and vice versa using the IP conversion mechanism defined in SIIT [I-D.ietf-behave-v6v4-xlate] (Li, X., Bao, C., and F. Baker, “IP/ICMP Translation Algorithm,” September 2010.).

When receiving IPv4 packets from IPv4 applications, translator converts IPv4 packet headers into IPv6 packet headers, then, if required, fragments the IPv6 packets (because header length of IPv6 is typically 20 bytes larger than that of IPv4), and sends them to IPv6 networks. When receiving IPv6 packets from the IPv6 networks, translator works symmetrically to the previous case, except that there is no need to fragment the packets.

The translator module has to adjust transport protocol checksums when translating between IPv4 and IPv6. In the IPv6 to IPv4 direction the translator also has to calculate IPv4 header checksum.

TOC

### 2.3. Extension Name Resolver

Extension Name Resolver returns a proper answer in response to the IPv4 application's name resolution request.

In the case of socket API implementation option, when an IPv4 application in an IPv6 only network tries to do forward lookup to resolve names via the resolver library (e.g. gethostbyname()), BIH intercept the function call and instead calls the IPv6 equivalent functions (e.g. getnameinfo()) that will resolve both A and AAAA records.

If only AAAA record is available for the name queried, ENR requests the address mapper to assign a local IPv4 address corresponding to the IPv6 address, creates an A record for the assigned IPv4 address, and returns the A record to the IPv4 application.

If both A and AAAA record are available in the IPv6 only network, ENR does not require address mapper to assign IPv4 address, but instead asks address mapper to store relationship between the A and AAAA records, and then directly passes the received A record to the IPv4 application. Note: this is a scenario where a host should use encapsulation instead to avoid protocol translation taking place at a host.

If only an A record is available it will be passed unmodified to the application so that the application learns a record exists for the destination. However, the application will not be able to use the address for communications if the host is in IPv6-only access network. If the application tries to send data to such an IPv4 address destination unreachable/host unreachable error will be returned, which allows application to behave accordingly.

```
Application | Network  | ENR behaviour
query       | response |
------------+----------+---------------------
   A        |   A      |  <return A record>
   A        |   AAAA   |  <synthesize A record>
   A        |  A/AAAA  |  <return A record>
```

**Figure 3: ENR behaviour illustration**

NOTE: An implementation option is to have ENR support in host's (stub) DNS resolver itself as described in [DNS64], in which case record synthesis is not needed and advanced functions such as DNSSEC are possible. If the ENR is implemented at the network layer, same limitations arise as when DNS record synthesis is done on the network. A host also has an option to implement recursive DNS server function.

### 2.3.1. Reverse DNS lookup

TOC

When an application initiates a reverse DNS query for a PTR record (in-addr.arpa), to find a name for an IP address, the ENR MUST check whether the queried IP address can be found in the Address Mapper's mapping table and is a local IP address. If an entry is found and the queried address is locally generated, the ENR must initiate corresponding reverse DNS query for the real IPv6 address (ip6.arpa). In the case application requested reverse lookup for an address not part of the local IPv4 address pool, e.g. a global address, the request shall be forwarded unmodified to the network.

For example, when an application initiates reverse DNS query for a synthesized locally valid IPv4 address, the ENR needs to intercept that query. The ENR will ask the address mapper for the IPv6 address that corresponds to the IPv4 address. The ENR shall perform reverse lookup procedure for the destination's IPv6 address and return the name received as a response to the application that initiated the IPv4 query.

---

## 2.4. Address Mapper

TOC

Address mapper ("the mapper" later on), maintains a local IPv4 address pool in the case of dual stack network and IPv6 only network. The pool consists of private IPv4 addresses as per section 4.3. Also, mapper maintains a table consisting of pairs of locally selected IPv4 addresses and destinationss' IPv6 addresses.

When the extension name resolver, translator, or the function mapper requests mapper to assign an IPv4 address corresponding to an IPv6 address, mapper selects and returns an IPv4 address out of the local pool, and registers a new entry into the table. The registration occurs in the following 3 cases:

(1) When the extension name resolver gets only an 'AAAA' record for the target host name in the dual stack or IPv6 only network and there is no existing mapping entry for the IPv6 address. A local IPv4 address will be returned to application and mapping for local IPv4 address to real IPv6 address is created.

(2) When the extension name resolver gets both an 'A' record and an 'AAAA' record for the target host name in the IPv6 only network and there is no existing apping entry for the IPv6 address. In this case local IPv4 address does not need to be selected, but mapping entry has to be created between IPv4 and IPv6 addresses from 'A' and 'AAAA' records. The IPv4 address will be returned to an application. Note: this is a scenario where IPv4 communications, native or encapsulated, are preferred over translation.

(3) When the function mapper gets a socket API function call triggered by received IPv6 packet and there is no existing mapping

entry for the IPv6 source address (Editor's note: can this ever happen in case of client-server nature of BIH?).

Other possible combinations are outside of BIH and BIH is not involved in those.

NOTE: There is one exception. When initializing the table the mapper registers a pair of its own IPv4 address and IPv6 address into the table.

## 3. Behavior and network Examples

TOC

Figure 4 illustrates the very basic network scenario. An IPv4-only application is running on a host attached to IPv6-only Internet and is talking to IPv6 enabled server. A communication is made possible by bump in the host.

It is worth noting that while the IPv6 server may additionally have IPv4 addresses, those are unreachable for the host not having any direct IPv4 connectivity, and hence can be considered irrelevant.

```
+----+                                   +-------------+
| H1 |----------- IPv6 Internet -------- | IPv6 server |
+----+                                   +-------------+
v4 only
application
```

**Figure 4: Network Scenario #1**

Figure 5 illustrates a possible network scenario where an IPv4-only application is running on a host attached to a dual-stack network, but the destination server is running on a private site that is numbered with public IPv6 addresses and private IPv4 addresses without port forwarding setup on NAT44. The only means to contact to server is to use IPv6.

```
+----------------------+  +-------------------------------+
| Dual Stack Internet  |  | IPv4 Private site (Net 10)    |
|                      |  |                               |
|                      |  |                 +-----------+ |
|                      |  |                 |           | |
|  +----+           +---------+            |           | |
|  | H1 |--------   |  NAT44  |------------|  Server   | |
|  +----+           +---------+            |           | |
| v4 only              |  |                 +-----------+ |
| application          |  |                  Dual Stack   |
|                      |  |                etc. 10.1.1.1 |
|                      |  |                 AAAA:2009::1 |
|                      |  |                               |
+----------------------+  +-------------------------------+
```

**Figure 5: Network Scenario #2**

Illustrations of host behavior in both implementation options are given here. Figure 6 illustrates the setup where BIH is implemented as a bump in the API, and figure 7 illustrates the setup where BIH is implemented as a bump in the stack.

```
"dual stack"                                                "host6"
IPv4    Socket |     [ API Translator ]    | TCP(UDP)/IP         Name
appli-  API    | ENR      Address  Function| (v6/v4)            Server
cation         |          Mapper   Mapper  |
 |        |        |        |        |         |              |       |
<<Resolve an IPv4 address for "host6".>>       |              |       |
 |        |        |        |        |         |              |       |
 |--------|------->|  Query of 'A' records for host6.         |       |
 |        |        |        |        |         |              |       |
 |        |        |--------|--------|---------|--------------|------>|
 |        |        |  Query of 'A' records and 'AAAA' for host6       |
 |        |        |        |        |         |              |       |
 |        |        |<-------|--------|---------|--------------|-------|
 |        |        |  Reply with the 'AAAA' record.           |       |
 |        |        |        |        |         |              |
 |        |        |<<The 'AAAA' record is resolved.>>        |
 |        |        |        |        |         |              |
 |        |        |+++++++>|  Request one IPv4 address       |
 |        |        |        |  corresponding to the IPv6 address.
 |        |        |        |        |         |              |
 |        |        |        |<<Assign one IPv4 address.>>     |
 |        |        |        |        |         |              |
 |        |        |<+++++++|  Reply with the IPv4 address.   |
 |        |        |        |        |         |              |
 |        |        |<<Create 'A' record for the IPv4 address.>>
 |        |        |        |        |         |              |
 |<-------|--------| Reply with the 'A' record.|              |
 |        |        |        |        |         |              |
 |        |        |        |        |         |              |
<<Call IPv4 Socket API function >>   |         |              |
 |        |        |        |        |         |              |
 |========|========|========|=======>|An IPv4 Socket API function Call
 |        |        |        |        |         |              |
 |        |        |        |<+++++++|  Request IPv6 addresses|
 |        |        |        |        |  corresponding to the  |
 |        |        |        |        |  IPv4 addresses.       |
 |        |        |        |        |         |              |
 |        |        |        |+++++++>| Reply with the IPv6 addresses.
 |        |        |        |        |         |              |
 |        |        |        |        |<<Translate IPv4 into IPv6.>>
 |        |        |        |        |         |              |
 |  An IPv6 Socket API function call.|=========|=============>|
 |        |        |        |        |         |              |
 |        |        |        |        |<<Reply an IPv6 data    |
 |        |        |        |        |  to dual stack.>>      |
 |        |        |        |        |         |              |
 |  An IPv6 Socket API function call.|<========|==============|
 |        |        |        |        |         |              |
 |        |        |        |        |<<Translate IPv6 into IPv4.>>
 |        |        |        |        |         |              |
 |        |        |        |<+++++++|  Request IPv4 addresses|
 |        |        |        |        |  corresponding to the  |
 |        |        |        |        |  IPv6 addresses.       |
 |        |        |        |        |         |              |
 |        |        |        |+++++++>| Reply with the IPv4 addresses.
 |        |        |        |        |         |              |
```

```
|<=======|========|========|========|  An IPv4 Socket function call.
|        |        |        |        |          |               |
```

**Figure 6: Example of BIH as API addition**

```
"dual stack"                                            "host6"
IPv4    TCP/    ENR     address  translator  IPv6
appli-  IPv4            mapper
cation
  |      |       |         |       |           |         |
<<Resolve an IPv4 address for "host6".>>       |         |
  |      |       |         |       |           |         |
  |------|------>|  Query of 'A' records for "host6".    | Name
  |      |       |         |       |           |         | Server
  |      |       |---------|-------|-----------|---------|--->|
  |      |       |  Query of 'A' records and 'AAAA' for "host6"
  |      |       |         |       |           |         |    |
  |      |       |<--------|-------|-----------|---------|----|
  |      |       |  Reply only with 'AAAA' record.       |
  |      |       |         |       |           |         |
  |      |       |<<Only 'AAAA' record is resolved.>>    |
  |      |       |         |       |           |         |
  |      |       |-------->|  Request one IPv4 address   |
  |      |       |         |  corresponding to the IPv6 address.
  |      |       |         |       |           |         |
  |      |       |         |<<Assign one IPv4 address.>> |
  |      |       |         |       |           |         |
  |      |       |<--------|  Reply with the IPv4 address.
  |      |       |         |       |           |         |
  |      |       |<<Create 'A' record for the IPv4 address.>>
  |      |       |         |       |           |         |
  |<-----|-------|  Reply with the 'A' record. |         |
  |      |       |         |       |           |         |
  |      |       |         |       |           |         |
<<Send an IPv4 packet to "host6".>>|           |         |
  |      |       |         |       |           |         |
  |======|=======|=========|======>|  An IPv4 packet.    |
  |      |       |         |       |           |         |
  |      |       |         |<------|  Request IPv6 addresses
  |      |       |         |       |  corresponding to the IPv4
  |      |       |         |       |  addresses.         |
  |      |       |         |       |           |         |
  |      |       |         |------>|  Reply with the IPv6|
  |      |       |         |       |  addresses.         |
  |      |       |         |       |           |         |
  |      |       |         |       |<<Translate IPv4 into IPv6.>>
  |      |       |         |       |           |         |
  |      |       |An IPv6 packet.  |===========|========>|
  |      |       |         |       |           |         |
  |      |       |         |     <<Reply an IPv6 packet to
  |      |       |         |       "dual stack".>>       |
  |      |       |         |       |           |         |
  |      |       |An IPv6 packet.  |<==========|=========|
  |      |       |         |       |           |         |
  |      |       |         |       |<<Translate IPv6 into IPv4.>>
  |      |       |         |       |           |         |
  |<=====|=======|=========|=======|  An IPv4 packet.    |
  |      |       |         |       |           |         |
```

**Figure 7: Example of BIH at network layer**

## 4. Considerations

TOC

### 4.1. Socket API Conversion

TOC

IPv4 socket API functions are translated into semantically as same IPv6 socket API functions as possible and vice versa. See Appendix B for the API list intercepted by BIH. However, IPv4 socket API functions are not fully compatible with IPv6 since the IPv6 has new advanced features.

### 4.2. ICMP Message Handling

TOC

When an application needs ICMP messages values (e.g., Type, Code, etc.) sent from a network layer, ICMPv4 message values MAY be translated into ICMPv6 message values based on SIIT [I-D.ietf-behave-v6v4-xlate] (Li, X., Bao, C., and F. Baker, “IP/ICMP Translation Algorithm,” September 2010.), and vice versa. It can be implemented using raw socket.

### 4.3. IPv4 Address Pool and Mapping Table

TOC

The address pool consists of the private IPv4 addresses as per [RFC1918] (Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G., and E. Lear, “Address Allocation for Private Internets,” February 1996.). This pool can be implemented at different granularity in the node e.g., a single pool per node, or at some finer granularity such as per user or per process. However, if a number of IPv4 applications communicate with IPv6 hosts, the available address spaces may be exhausted. As a result, it will be impossible for IPv4 applications to communicate with IPv6 nodes. It requires smart management techniques for address pool. For example, it is desirable for the mapper to free the oldest entry and reuse the IPv4 address or IPv6 address for creating a new entry. In case of a per-node address mapping table, it MAY cause a larger risk of running out of address.

The RFC1918 address space was chosen because generally legacy applications understand that as a private address space. A new dedicated address space would run a risk of not being understood by applications as private. 127/8 or 169.254/16 are rejected due possible assumptions applications may make when seeing those.

The RFC1918 addresses have a risk of conflicting with other interfaces. The conflicts can be mitigated by using least commonly used network number of the RFC1918 address space. Addresses from 172.16/12 prefix are thought to be less likely to conflict than addresses from 10/8 or 192.168/16 spaces, hence the used IPv4 addresses are following (Editor's comment: this is first and almost random proposals):

Source addresses: 172.21.112.0/30. Source address have to be allocated because applications use getsockname() calls and as in the BIS mode an IP address of the IPv4 interface has to be shown. More than one address is allocated to allow implementation flexibility, e.g. for cases where a host has multiple IPv6 interfaces. The source addresses are from different subnet than destination addresses to ensure applications do not do on-link assumptions and do enable NAT traversal functions.

Primary destination addresses: 172.21.80.0/20. Address mapper will select destination addresses primarily out of this pool.

Secondary destination addresses: 10.170.160.0/20. Address mapper will select destination addresses out of this pool if the node has dual-stack connection conflicting with primary destination addresses.

---

### 4.4. Multi-interface

TOC

In the case of dual-stack destinations BIH must do protocol translation from IPv4 to IPv6 only when the host does not have any IPv4 interfaces, native or tunneled, available for use.

It is possible that an IPv4 interface is activated during BIH operation, for example if a node moves to a coverage area of IPv4 enabled network. In such an event BIH must stop initiating protocol translation sessions for new connections and BIH may disconnect active sessions. The choice of disconnection is left for implementatations and it may depend on whether IPv4 address conflict situation occurs between addresses used by BIH and addresses used by new IPv4 interface.

---

### 4.5. Multicast

TOC

Protocol translation for multicast is not supported.

---

TOC

### 4.6. DNS cache

When BIH module shuts down, e.g. due IPv4 interface becoming available, BIH must flush node's DNS cache of possible locally generated entries.

## 5. Considerations due ALG requirements

TOC

No ALG functionality is specified herein as ALG design is generally not encouraged for host based translation and as BIH is intended for applications not including IP addresses in protocol payloads.

## 6. Security Considerations

TOC

The security consideration of BIH mostly relies on that of [I-D.ietf-behave-v6v4-xlate-stateful] (Bagnulo, M., Matthews, P., and I. Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers," July 2010.).

In the socket layer implementation approach the differences are due to the address translation occurring at the API and not in the network layer. That is, since the mechanism uses the API translator at the socket API level, hosts can utilize the security of the network layer (e.g., IPSec) when they communicate with IPv6 hosts using IPv4 applications via the mechanism. As well, there is no need for DNS ALG as in NAT-PT, so there is no interference with DNSSEC either.

In the network layer implementation approach hosts cannot utilize the security above network layer when they communicate with IPv6 hosts using IPv4 applications via BIH and encrypt embedded IP addresses, or when the protocol data is encrypted using IP addresses as keys. In these cases it is impossible for the mechanism to translate the IPv4 data into IPv6 and vice versa. Therefore it is highly desirable to upgrade to the applications modified into IPv6 for utilizing the security at communication with IPv6 hosts.

The use of address pooling may open a denial of service attack vulnerability. So BIH should employ the same sort of protection techniques as NAT64 [I-D.ietf-behave-v6v4-xlate-stateful] (Bagnulo, M., Matthews, P., and I. Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers," July 2010.) does.

TOC

## 7. Acknowledgments

The author thanks the discussion from Gang Chen, Dapeng Liu, Bo Zhou, Hong Liu, Tao Sun, Zhen Cao, Feng Cao et al. in the development of this document.

The efforts of Suresh Krishnan, Mohamed Boucadair, Yiu L. Lee, James Woodyatt, Lorenzo Colitti, Qibo Niu, Pierrick Seite, Dean Cheng, Christian Vogt, Jan M. Melen, and Ed Jankiewizh in reviewing this document are gratefully acknowledged.

Advice from Dan Wing, Dave Thaler and Magnus Westerlund are greatly appreciated

The authors of RFC2767 acknowledged WIDE Project, Kazuhiko YAMAMOTO, Jun MURAI, Munechika SUMIKAWA, Ken WATANABE, and Takahisa MIYAMOTO. The authors of RFC3338 acknowledged implementation contributions by Wanjik Lee (wjlee@arang.miryang.ac.kr) and i2soft Corporation (www.i2soft.net).

The authors of Bump-in-the-Wire (draft-ietf-biw-00.txt, October 2006), P. Moster, L. Chin, and D. Green, are acknowledged. Few ideas and clarifications from BIW have been adapted to this document.

## 8. References

TOC

### 8.1. Normative References

TOC


| | |
|---|---|
| [I-D.ietf-behave-v6v4-xlate] | Li, X., Bao, C., and F. Baker, "IP/ICMP Translation Algorithm," draft-ietf-behave-v6v4-xlate-23 (work in progress), September 2010 (TXT). |
| [I-D.ietf-behave-v6v4-xlate-stateful] | Bagnulo, M., Matthews, P., and I. Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers," draft-ietf-behave-v6v4-xlate-stateful-12 (work in progress), July 2010 (TXT). |
| [RFC1918] | Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G., and E. Lear, "Address Allocation for Private Internets," BCP 5, RFC 1918, February 1996 (TXT). |
| [RFC2119] | Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML). |
| [RFC2460] | Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460, December 1998 (TXT, HTML, XML). |
| [RFC2767] | Tsuchiya, K., HIGUCHI, H., and Y. Atarashi, "Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS)," RFC 2767, February 2000 (TXT). |
| [RFC2893] | |

| | |
|---|---|
| | Gilligan, R. and E. Nordmark, "Transition Mechanisms for IPv6 Hosts and Routers," RFC 2893, August 2000 (TXT). |
| [RFC3338] | Lee, S., Shin, M-K., Kim, Y-J., Nordmark, E., and A. Durand, "Dual Stack Hosts Using "Bump-in-the-API" (BIA)," RFC 3338, October 2002 (TXT). |


### 8.2. Informative References

TOC


| | |
|---|---|
| [RFC2553] | Gilligan, R., Thomson, S., Bound, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6," RFC 2553, March 1999 (TXT). |


## Appendix A. Implementation option for the ENR

TOC

It is not necessary to implement the ENR at the kernel level, but it can be implemented instead at the user space by setting the host's default DNS server to point to 127.0.0.1. DNS queries would then always be sent to the ENR, which furthermore ensures both A and AAAA queries are sent to the actual DNS server and A queries are always answered and required mappings created.

## Appendix B. API list intercepted by BIH

TOC

The following functions are the API list which SHOULD be intercepted by BIH module when implemented at socket layer.

The functions that the application uses to pass addresses into the system are:

> bind()
>
> connect()
>
> sendmsg()
>
> sendto()

The functions that return an address from the system to an application are:

> accept()
>
> recvfrom()
>
> recvmsg()
>
> getpeername()

> getsockname()

The functions that are related to socket options are:

> getsocketopt()
>
> setsocketopt()

The functions that are used for conversion of IP addresses embedded in application layer protocol (e.g., FTP, DNS, etc.) are:

> recv()
>
> send()
>
> read()
>
> write()

As well, raw sockets for IPv4 and IPv6 MAY be intercepted.

Most of the socket functions require a pointer to the socket address structure as an argument. Each IPv4 argument is mapped into corresponding an IPv6 argument, and vice versa.

According to [RFC2553] (Gilligan, R., Thomson, S., Bound, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6," March 1999.), the following new IPv6 basic APIs and structures are required.

```
IPv4                       new IPv6
------------------------------------------------
AF_INET                    AF_INET6
sockaddr_in                sockaddr_in6
gethostbyname()            getaddrinfo()
gethostbyaddr()            getnameinfo()
inet_ntoa()/inet_addr()    inet_pton()/inet_ntop()
INADDR_ANY                 in6addr_any
```

**Figure 8**

BIH MAY intercept inet_ntoa() and inet_addr() and use the address mapper for those. Doing that enables BIH to support literal IP addresses.

The gethostbyname() call return a list of addresses. When the name resolver function invokes getaddrinfo() and getaddrinfo() returns multiple IP addresses, whether IPv4 or IPv6, they SHOULD all be represented in the addresses returned by gethostbyname(). Thus if getaddrinfo() returns multiple IPv6 addresses, this implies that multiple address mappings will be created; one for each IPv6 address.

## Authors' Addresses

TOC


| | |
|---|---|
| | Bill Huang |
| | China Mobile |
| | 53A,Xibianmennei Ave., |
| | Xuanwu District, |
| | Beijing 100053 |
| | China |
| Email: | bill.huang@chinamobile.com |
| | |
| | Hui Deng |
| | China Mobile |
| | 53A,Xibianmennei Ave., |
| | Xuanwu District, |
| | Beijing 100053 |
| | China |
| Email: | denghui02@gmail.com |
| | |
| | Teemu Savolainen |
| | Nokia |
| | Hermiankatu 12 D |
| | FI-33720 TAMPERE |
| | Finland |
| Email: | teemu.savolainen@nokia.com |