Behave WG	B.Huang. Huang
Internet-Draft	H.Deng. Deng
Obsoletes: 3338, 2767 (if approved)	China Mobile
Intended status: Standards Track	T.Savolainen. Savolainen
Expires: September 12, 2011	Nokia
	March 11, 2011

Dual Stack Hosts Using "Bump-in-the-Host" (BIH)
draft-ietf-behave-v4v6-bih-03

Abstract

Bump-In-the-Host (BIH) is a host-based IPv4 to IPv6 protocol translation mechanism that allows a class of IPv4-only applications that work through NATs to communicate with IPv6-only peers. The host on which applications are running may be connected to IPv6-only or dual-stack access networks. BIH hides IPv6 and makes the IPv4-only applications think they are talking with IPv4 peers by local synthesis of IPv4 addresses.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes Bump-in-the-Host (BIH), a successor and combination of the Bump-in-the-Stack (BIS)[RFC2767] and Bump-in-the-API (BIA) [RFC3338] technologies, which enable IPv4-only legacy applications to communicate with IPv6-only servers by synthesizing IPv4 addresses from AAAA records.

The supported class of applications includes those that use DNS for IP address resolution and that do not embed IP address literals in protocol payloads. This essentially includes legacy client-server applications using the DNS that are agnostic to the IP address family used by the destination and that are able to do NAT traversal. The synthetic IPv4 addresses shown to applications are taken from the RFC1918 private address pool in order to ensure that possible NAT traversal techniques will be initiated.

IETF recommends using dual-stack or tunneling based solutions for IPv6 transition and specifically recommends against deployments utilizing double protocol translation. Use of BIH together with a NAT64 is NOT RECOMMENDED as a competing technology for tunneling based transition solutions.

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

When BIH is implemented at the socket API layer, the translator intercepts IPv4 socket API function calls and invokes corresponding IPv6 socket API function calls to communicate with IPv6 hosts.

When BIH is implemented at the networking layer the IPv4 packets are intercepted and converted to IPv6 using the IP conversion mechanism defined in Stateless IP/ICMP Translation Algorithm (SIIT) [I-D.ietf-behave-v6v4-xlate]. The protocol translation has the same benefits and drawbacks as SIIT.

The location of the BIH refers essentially to the location of the protocol translation function. The location of DNS synthesis is orthogonal to the location of protocol translation, and may or may not happen at the same level.

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

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] .

This document uses terms defined in [RFC2460] , [RFC2893] , [RFC2767] and [RFC3338].

1.1. Acknowledgement of previous work

This document is a direct update to and directly derivative from Kazuaki TSHUCHIYA, Hidemitsu HIGUCHI, and Yoshifumi ATARASHI's Bump-in-the-Stack [RFC2767] and from Seungyun Lee, Myung-Ki Shin, Yong-Jin Kim, Alain Durand, and Erik Nordmark's Bump-in-the-API [RFC3338], which similarly provide a dual stack host means to communicate with other IPv6 hosts using existing IPv4 applications.

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

Figure 1 shows the architecture of a host in which BIH is implemented as a socket API layer translator, i.e., as a "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 2 shows the architecture of a host in which BIH is implemented as a network layer translator, i.e., a "Bump-in-the-Stack".

```
   +------------------------------------------------------------+
   |  +------------------------------------------+              |
   |  |    IPv4 applications                     |              |
   |  |    Host's main DNS resolver              |              |
   |  +------------------------------------------+              |
   |  +------------------------------------------+              |
   |  |    TCP/UDP                               |              |
   |  +------------------------------------------+              |
   |  +------------------------------------------+ +---------+  |
   |  |    IPv4                                  | |         |  |
   |  +------------------------------------------+ | Address |  |
   |  +------------------+ +---------------------+ | Mapper  |  |
   |  |    Protocol      | |   Extension Name    | |         |  |
   |  |    Translator    | |   Resolver          | |         |  |
   |  +------------------+ +---------------------+ |         |  |
   |  +------------------------------------------+ |         |  |
   |  |    IPv4 / IPv6                           | |         |  |
   |  +------------------------------------------+ +---------+  |
   +------------------------------------------------------------+
```

Dual stack hosts defined in RFC 2893 [RFC2893] 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 allow existing IPv4 applications to communicate with IPv6-only peers. The BIH architecture consists of an Extension Name Resolver, an Address Mapper, and depending on implementation either a Function Mapper or a Protocol Translator. It is worth noting that Extension Name Resolver's placement is orthogonal decision to placement of protocol translation. For example, the Extension Name Resolver may reside in the socket API while protocol translation takes place at the networking layer.

2.1. Function Mapper

The function mapper translates an IPv4 socket API function into an IPv6 socket API function.
When detecting IPv4 socket API function calls from IPv4 applications, the function mapper intercepts the function calls and invokes IPv6 socket API functions that correspond to the IPv4 socket API functions. See Appendix B for a list of functions that MUST be intercepted by the function mapper.

2.2. Protocol translator

The protocol translator translates IPv4 into IPv6 and vice versa using the IP conversion mechanism defined in SIIT [I-D.ietf-behave-v6v4-

xlate]. To avoid unnecessary fragmentation, host's IPv4 module should be configured with small enough MTU (IPv6 link MTU - 20 bytes).

2.3. Extension Name Resolver

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

In the case of the socket API layer implementation option, when an IPv4 application tries to do a forward lookup to resolve names via the resolver library (e.g., gethostbyname()), BIH intercepts the function call and instead calls the IPv6 equivalent functions (e.g., getnameinfo()) that will resolve both A and AAAA records. This implementation option is name resolution protocol agnostic, and hence supports techniques such as "hosts-file", NetBIOS, mDNS, and essentially anything underlying operating system uses.

In the case of the network layer implementation option, the ENR intercepts the A query and creates an additional AAAA query with essentially the same content. The ENR will then collect replies to both A and AAAA queries and, depending on results, either return an A reply unmodified or synthesize a new A reply. The network layer implementation option will only be able to catch applications' name resolution requests that result in actual DNS queries, hence is more limited when compared to socket API layer implementation option.

In either implementation option, if only AAAA records are available for the queried name, the ENR asks the address mapper to assign a local IPv4 address corresponding to each IPv6 address. In the case of the API layer implementation option, the ENR will simply the make API (e.g. gethostbyname) return the synthetic address. In the case of the network-layer implementation option, the ENR synthesizes an A record for the assigned IPv4 address, and delivers it up the stack.

If there is a real A record available, the ENR SHOULD NOT synthesize IPv4 addresses. By default an ENR implementation MUST NOT synthesize IPv4 addresses when real A records exist.

If the response contains a CNAME or a DNAME record, then the CNAME or DNAME chain is followed until the first terminating A or AAAA record is reached.

Application query	Network response	ENR behavior
A	A	<return real A record>
A	AAAA	<synthesize A record>
A	A/AAAA	<return real A record>

2.3.1. Special exclusion sets for A and AAAA records

An ENR implementation MAY by default exclude certain IPv4 and IPv6 addresses seen on received A and AAAA records. The addresses to be excluded by default SHOULD include martian addresses such as those that should not appear in the DNS or on the wire. Additional addresses MAY be excluded based on possibly configurable local policies.

2.3.2. DNSSEC support

When the ENR is implemented at the network layer, the A record synthesis can cause essentially the same issues as are described in [I-D.ietf-behave-dns64] section 3. To avoid unwanted discarding of synthetic A records on the host's main resolver, the host's main resolver MUST send DNS questions with the CD "Checking Disabled" bit cleared. The ENR can support DNSSEC as any resolver on a host.

When the ENR is implemented at the socket API level, there are no problems with DNSSEC, as the ENR itself uses socket APIs for DNS resolution.

DNSSEC can also be supported by configuring the (stub) resolver on a host to trust validations done by the ENR located at network layer or alternatively the validating resolver can implement ENR on itself.

In order to properly support DNSSEC, the ENR SHOULD be implemented at the socket API level. If the socket API level implementation is not possible, DNSSEC support SHOULD be provided by other means.

2.3.3. Reverse DNS lookup

When an application initiates a reverse DNS query for a PTR record, 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 a corresponding reverse DNS query for the real IPv6 address. 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 MUST be forwarded unmodified to the network.

For example, when an application initiates a reverse DNS query for a synthesized locally valid IPv4 address, the ENR needs to intercept that query. The ENR asks the address mapper for the IPv6 address that corresponds to the IPv4 address. The ENR shall perform a 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

The address mapper maintains a local IPv4 address pool. The pool consists of private IPv4 addresses as per section 4.3. Also, the address mapper maintains a table consisting of pairs of locally selected IPv4 addresses and destinations' IPv6 addresses.

When the extension name resolver, translator, or the function mapper requests the address mapper to assign an IPv4 address corresponding to an IPv6 address, the 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 AAAA records for the target host name in the dual stack or IPv6-only network and there is no existing mapping entry for the IPv6 addresses. One or more local IPv4 addresses will be returned to application and mappings for local IPv4 addresses to real IPv6 addresses are created.

(2) When the extension name resolver gets both A records and AAAA records, but the A records contain only excluded IPv4 addresses. Behavior will follow the case (1).

(3) When the function mapper gets a socket API function call triggered by a received IPv6 packet and there is no existing mapping entry for the IPv6 source address (for example, client sent UDP request to anycast address but response was received from unicast address).

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

3. Behavior and network Examples

Figure 4 illustrates a very basic network scenario. An IPv4-only application is running on a host attached to the IPv6-only Internet and is talking to an IPv6-only server. Communication is made possible by Bump-In-the-Host.

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

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 the NAT44. The only means to contact the server is to use IPv6.

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

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 IPv4 address 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.   |
 |        |        |        |        |         |              |
 |<-------|<-------| Reply with the IPv4 address              |
 |        |        |        |        |         |              |
 |        |        |        |        |         |              |
<<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.|=======================>|
 |        |        |        |        |         |              |
 |        |        |        |        |<<IPv6 data received    |
 |        |        |        |        |  from network.>>       |
 |        |        |        |        |         |              |
 |  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.
 |        |        |        |        |         |              |
   "dual stack"                                         "host6"
IPv4 stub  TCP/    ENR     address  translator  IPv6
app  res.  IPv4            mapper
  |   |    |       |         |       |           |         |
<<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 each IPv6 address.
  |   |    |       |         |       |           |         |
  |   |    |       |         |<<Assign IPv4 addresses.>>   |
  |   |    |       |         |       |           |         |
  |   |    |       |<--------|  Reply with the IPv4 address.
  |   |    |       |         |       |           |         |
  |   |    |       |<<Create 'A' record for the IPv4 address.>>
  |   |    |       |         |       |           |         |
  |   |<-----------|  Reply with the 'A' record. |         |
  |   |    |       |         |       |           |         |
  |<--|<<Reply with the IPv4 address |           |         |
  |   |    |       |         |       |           |         |
  <<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 with an IPv6 packet to.>>
  |   |    |       |         |       |           |         |
  |   |    |       |An IPv6 packet.  |<==========|<========|
  |   |    |       |         |       |           |         |
  |   |    |       |         |       |<<Translate IPv6 into IPv4.>>
  |   |    |       |         |       |           |         |
  |<=======|=========================|  An IPv4 packet.    |
  |   |    |       |         |       |           |         |
```

4. Considerations

4.1. Socket API Conversion

IPv4 socket API functions are translated into IPv6 socket API functions that are semantically as identical 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 IPv4 supports features that are not present in IPv6, such as SO_BROADCAST.

4.2. ICMP Message Handling

When an application needs ICMP messages values (e.g., Type, Code, etc.) sent from the network layer, ICMPv4 message values MAY be translated into ICMPv6 message values based on SIIT [I-D.ietf-behave-v6v4-xlate], and vice versa.

4.3. IPv4 Address Pool and Mapping Table

The address pool consists of the private IPv4 addresses as per [RFC1918]. This pool can be implemented at different granularities in the node, e.g., a single pool per node, or at some finer granularity such as per-user or per-process. In the case of a large number of IPv4 applications communicating with a large number of IPv6 servers, the available address space may be exhausted if the granularity is not fine enough. This should be a rare event and chances will decrease as IPv6 support increases. The possible problem can also mitigated with smart management techniques of the address pool. For example, entries with the longest inactivity time can be cleared and IPv4 addresses reused for creating new entries.

The RFC1918 address space was chosen because generally legacy applications understand it as a private address space. A new dedicated address space would run a risk of not being understood by applications as private. 127/8 and 169.254/16 are rejected due to 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 a least commonly used portion of the RFC1918 address space. Addresses from 172.16/12 are thought to be less likely to conflict than addresses from 10/8 or 192.168/16 spaces, hence the RECOMMENDED IPv4 addresses are following (Editor's comment: this is first proposal, educated better guesses are welcome): Source addresses: 172.21.112.0/30. Source addresses have to be allocated because applications use getsockname() calls and in the BIS mode an IP address of the IPv4 interface has to be shown (e.g., by

'ifconfig'). 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 subnets than destination addresses to that ensure applications would not make on-link assumptions and would instead enable NAT traversal functions.
Primary destination addresses: 172.21.80.0/20. The address mapper will select destination addresses primarily out of this pool.
Secondary destination addresses: 10.170.160.0/20. The address mapper will select destination addresses out of this pool if the node has a dual-stack connection conflicting with primary destination addresses.

4.4. Multi-interface

In the case of dual-stack destinations BIH MUST NOT do protocol translation from IPv4 to IPv6 when the host has 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 an 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 implementations and it may depend on whether IPv4 address conflict occurs between addresses used by BIH and addresses used by the new IPv4 interface.

4.5. Multicast

Protocol translation for multicast is not supported.

4.6. DNS cache

When BIH shuts down, e.g., due to an IPv4 interface becoming available, BIH MUST flush the node's DNS cache of possible locally generated entries. This cache may be in the ENR itself, but also possibly host's caching stub resolver.

5. Considerations due ALG requirements

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

6. Security Considerations

The security considerations of BIH mostly relies on that of [I-D.ietf-behave-v6v4-xlate-stateful].
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 layer, 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 such, 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, IPv4-using IKE will not work. This means IPv4-based IPsec/IKE using VPN solutions cannot work through BIH. However, transport and application layer solutions such as TLS or SSL-VPN do work through BIH.
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] does.

7. Changes since RFC2767 and RFC3338

This document combines and obsoletes both [RFC2767] and [RFC3338].
The changes in this document mainly reflect the following components:

*1. Supporting IPv6-only network connections

*2. The IPv4 address pool uses private address instead of reserved 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. Standards track instead of experimental/informational

*6. Supporting reverse (PTR) queries

8. 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, Ed Jankiewizh, Marnix Goossens, Ala Hamarsheh, Dan Wing, Magnus Westerlun and Julien Laganier in reviewing this document are gratefully acknowledged.
Special acknowledgements go to Dave Thaler for his extensive review and support.
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 (BIW) (draft-ietf-biw-00.txt, October 2006), P. Moster, L. Chin, and D. Green, are acknowledged. Some ideas and clarifications from BIW have been adapted to this document.

9. References

9.1. Normative References

[RFC1918]	Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G. and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[RFC2460]	Deering, S.E. and R.M. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.
[RFC2767]	Tsuchiya, K., HIGUCHI, H. and Y. Atarashi, "Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS)", RFC 2767, February 2000.
[RFC2893]	Gilligan, R. and E. Nordmark, "Transition Mechanisms for IPv6 Hosts and Routers", RFC 2893, August 2000.
[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.
[I-D.ietf-behave-v6v4-xlate]	Li, X, Bao, C and F Baker, "IP/ICMP Translation Algorithm", Internet-Draft draft-ietf-behave-v6v4-xlate-23, September 2010.
[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", Internet-Draft draft-ietf-behave-v6v4-xlate-stateful-12, July 2010.
[I-D.ietf-behave-dns64]	Bagnulo, M, Sullivan, A, Matthews, P and I Beijnum, "DNS64: DNS extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", Internet-Draft draft-ietf-behave-dns64-11, October 2010.

9.2. Informative References

[RFC3493]	Gilligan, R., Thomson, S., Bound, J., McCann, J. and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.

Appendix A. Implementation option for the ENR

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 that 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

The following functions are the API list which SHOULD be intercepted by BIH module when implemented at socket layer. Please note that this list may not be fully exhaustive.
The functions that the application uses to pass addresses into the system are:

*bind()

*connect()

*sendmsg()

*sendto()

*gethostbyaddr()

*getnameinfo()

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

*accept()

*recvfrom()

*recvmsg()

*getpeername()

*getsockname()

*gethostbyname()

*getaddrinfo()

The functions that are related to socket options are:

*getsocketopt()

*setsocketopt()

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 [RFC3493], 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

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() and getaddrinfo() calls 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

Bill Huang Huang China Mobile 53A,Xibianmennei Ave., Xuanwu District, Beijing, 100053 China EMail: bill.huang@chinamobile.com

Hui Deng Deng China Mobile 53A,Xibianmennei Ave., Xuanwu District, Beijing, 100053 China EMail: denghui02@gmail.com

Teemu Savolainen Savolainen Nokia Hermiankatu 12 D FI-33720 TAMPERE Finland EMail: teemu.savolainen@nokia.com