Behave WG
Internet-Draft
Obsoletes: 3338, 2767
(if approved)
Intended status: Standards Track
Expires: July 27, 2011

B. Huang
H. Deng
China Mobile
T. Savolainen
Nokia

January 23, 2011

# Dual Stack Hosts Using "Bump-in-the-Host" (BIH)
# draft-ietf-behave-v4v6-bih-02

## Abstract

Bump-In-the-Host (BIH) is a host based IPv4 to IPv6 protocol translation mechanism that allows class of IPv4-only applications that work through NATs to communicate with IPv6-only peers. The host applications are running on 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 synthetization of A records.

## 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 July 27, 2011.

## Copyright Notice

Table of Contents

## 1. Introduction

This document describes a Bump-in-the-Host (BIH), successor and combination of Bump-in-the-Stack (BIS)[RFC2767] and Bump-in-the-API (BIA) [RFC3338] technologies, which enables IPv4-only legacy applications to communicate with IPv6-only servers by synthesizing A records 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 RFC1918 private address pool in order to ensure 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 network-side IP translation is NOT RECOMMENDED as a competing technology for tunneling based transition solutions.

BIH technique includes two major implementation options: a protocol translator between the IPv4 and the IPv6 stacks of a host or 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 the 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 the IPv6 hosts.

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]. The protocol layer translation has the same benefits and drawbacks as SIIT.

The 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 direct 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 mainly reflect following 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

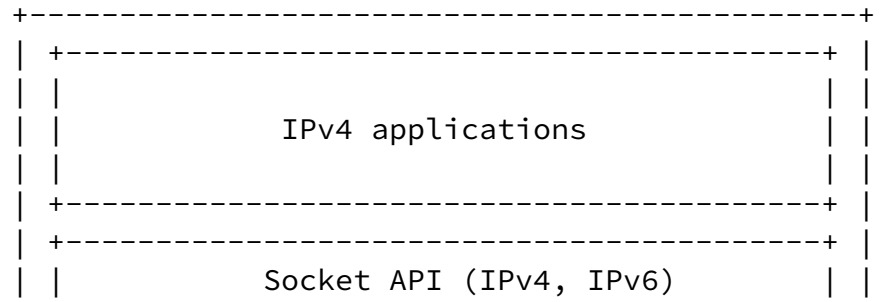
6. Supporting reverse (PTR) queries





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

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".



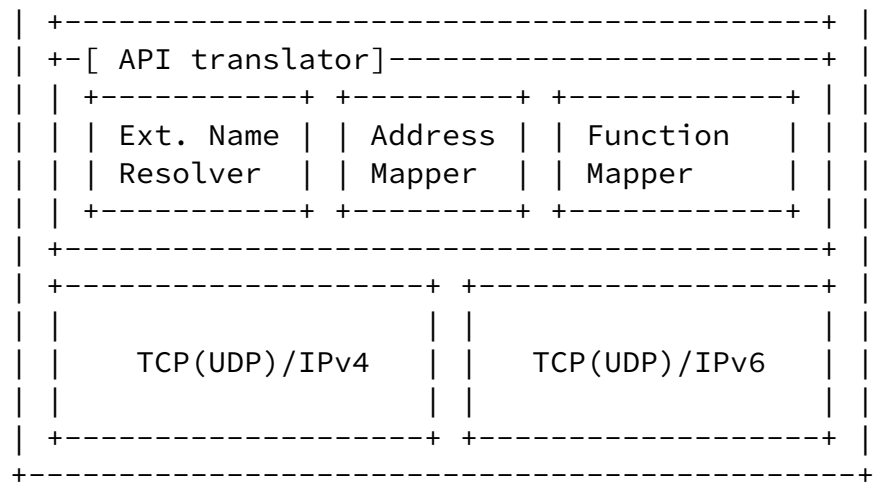
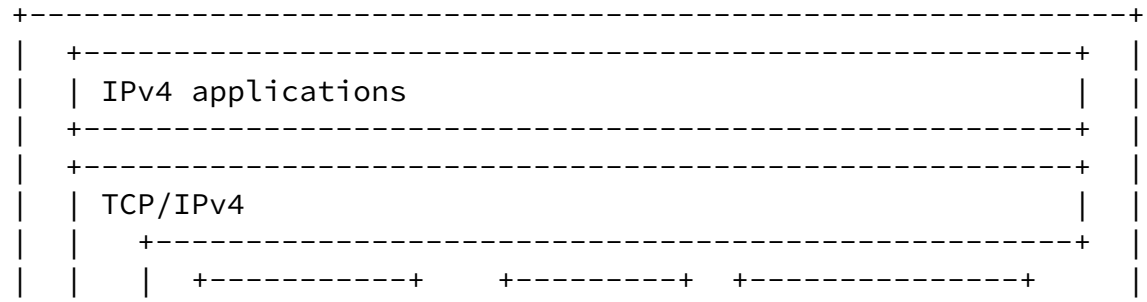



Figure 1: Architecture of the dual stack host using BIH at socket layer

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".







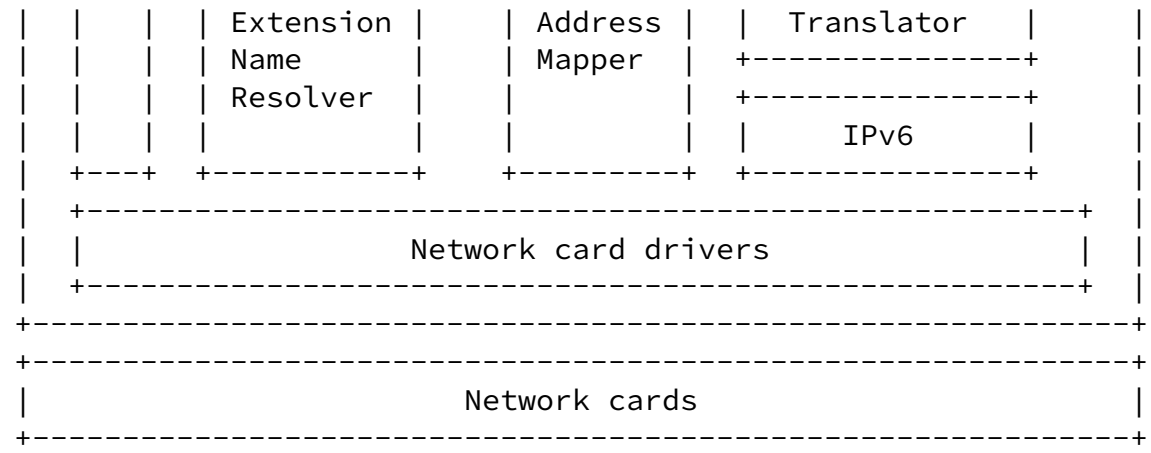



Figure 2: Architecture of the dual-stack host using BIH at network layer

Dual stack hosts defined in RFC2893 [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 communicate with IPv6-only peers 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

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 IPv6 socket API functions which correspond to the IPv4 socket API functions. The 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.

See Appendix B for list of functions that may be intercepted by the function mapper.





## 2.2. Translator

Translator translates IPv4 into IPv6 and vice versa using the IP conversion mechanism defined in SIIT [I-D.ietf-behave-v6v4-xlate].

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.

## 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 layer implementation option, when an IPv4 application 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.

In the case of stack layer implementation option the ENR intercepts the A query and creates 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 returns A reply unmodified or drops the real A reply and synthesizes a new A reply.

In either implementation options, if only non-excluded AAAA records are available for the queried name, ENR requests the address mapper to assign a local IPv4 address corresponding to the IPv6 address(es). In the case of API layer implementation option the ENR will simply make API (e.g. gethostbyname) to return the synthetic address. In the case of network layer implementation option ENR synthesizes an A record for the assigned IPv4 address, and returns the A record to the IPv4 application.

If there is real, non-excluded, A record available, ENR SHOULD NOT synthetize IPv4 addresses to be given to the application. By default 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 chains is followed until the first terminating A or AAAA record is reached.

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

Figure 3: ENR behaviour illustration

### 2.3.1. Special exclusion sets for A and AAAA records

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

The A record synthesis done by ENR in the network layer model can cause problems for DNSSEC validation possibly done by the host's resolver, as the synthetic responses cannot be succesfully validated. DNSSEC can be supported by configuring the (stub) resolver on a host to trust validations done by the local ENR or alternatively the validating resolver can implement ENR on itself and only SIIT takes place at network layer.

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

### 2.3.3. Reverse DNS lookup

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

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 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, but the A record contains only excluded IPv4 addresses. Behavior will follow the case (1).

(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

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-only server. A communication is made possible by Bump-In-the-Host.

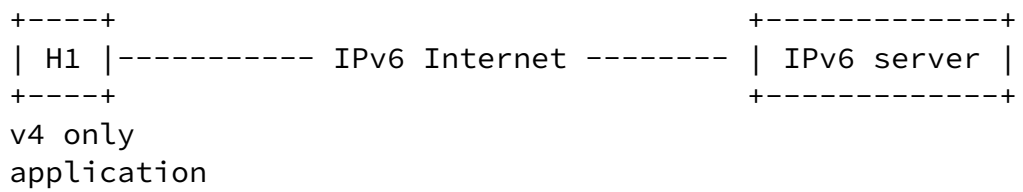
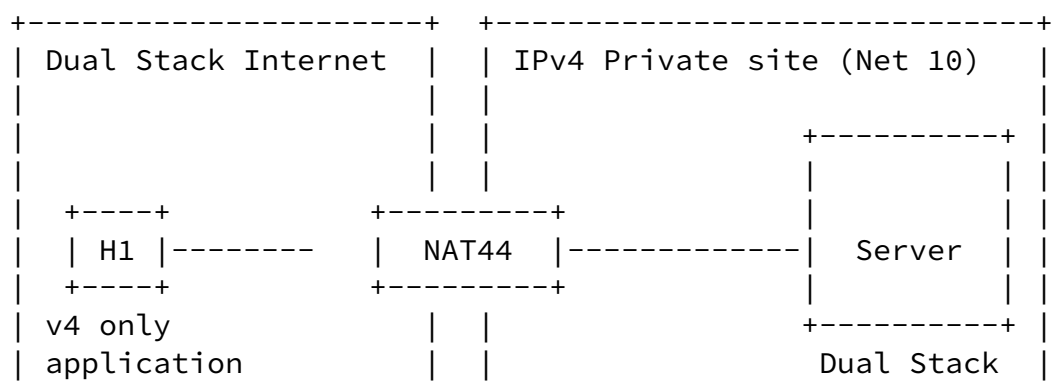



Figure 4: Network Scenario #1

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.



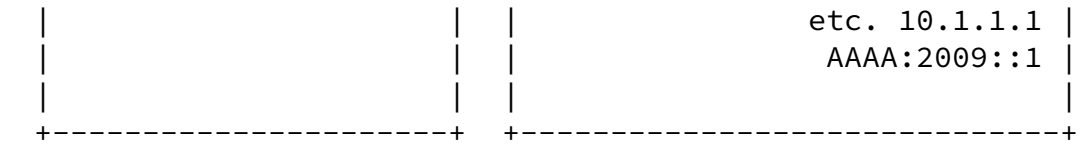
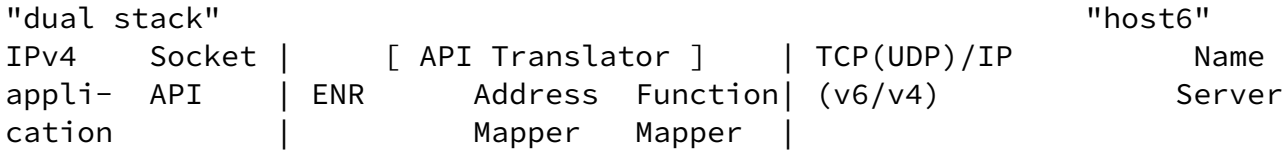


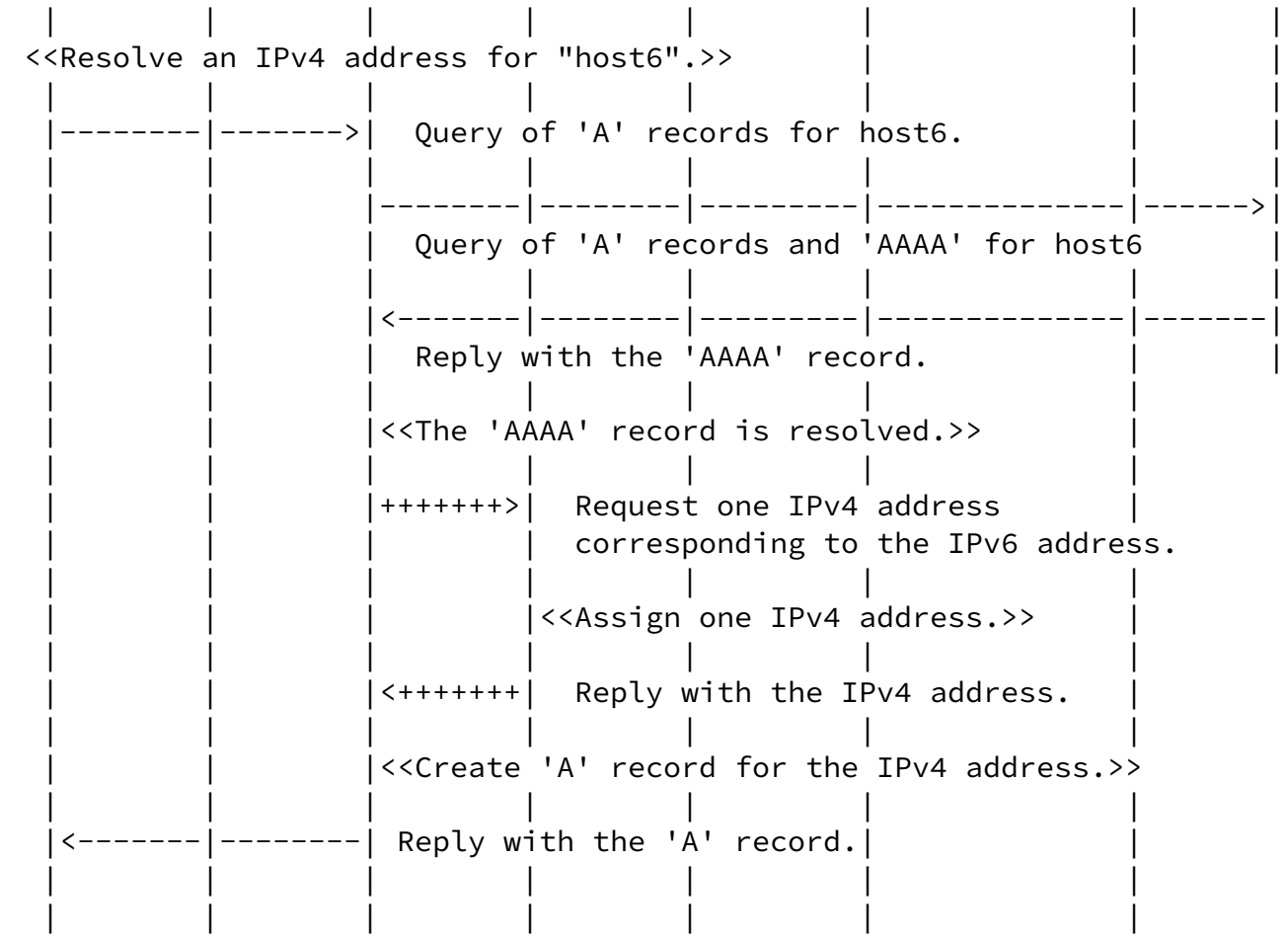

Figure 5: Network Scenario #2

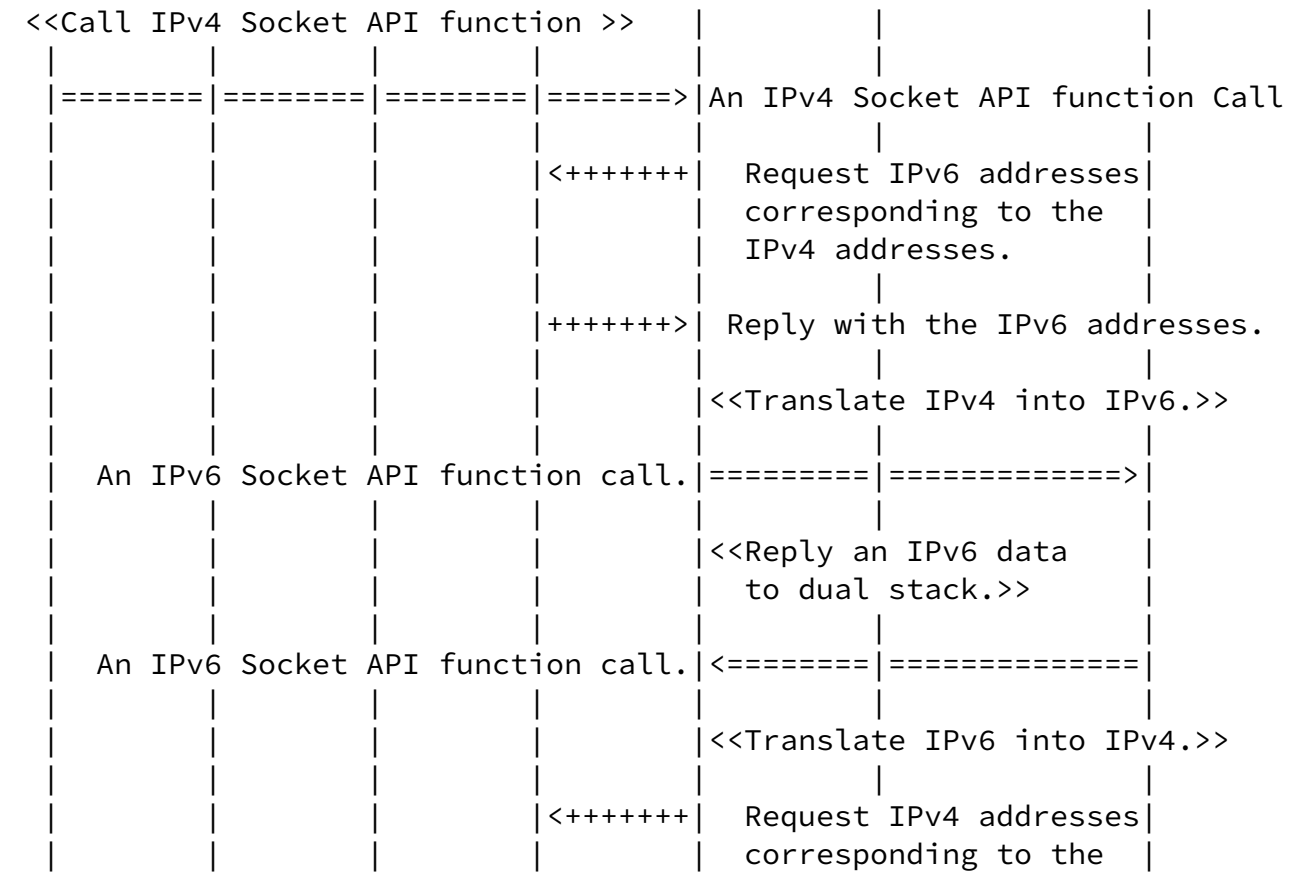
Illustrations of host behavior in both implementation options are given here. Figure 6 illustrates the setup where BIH is implemented as a bump in the API, and figure 7 illustrates the setup where BIH is implemented as a bump in the stack.













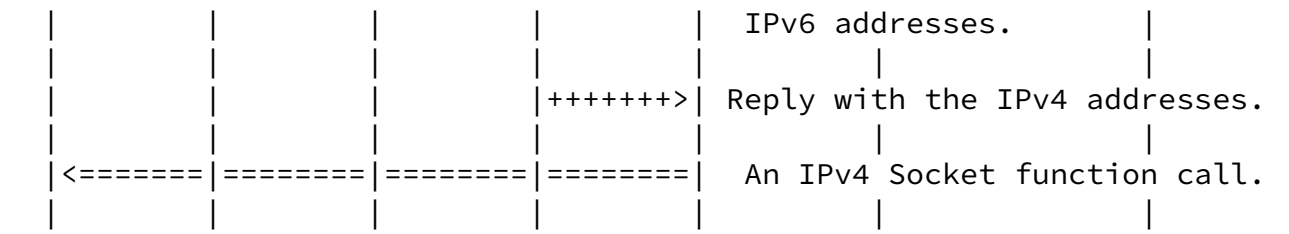
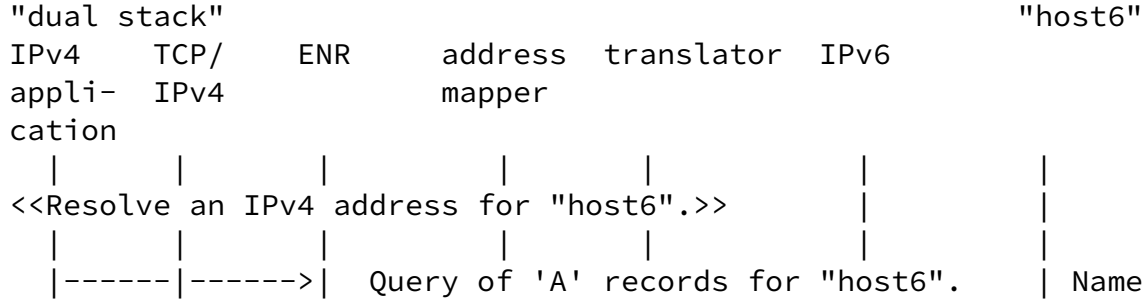
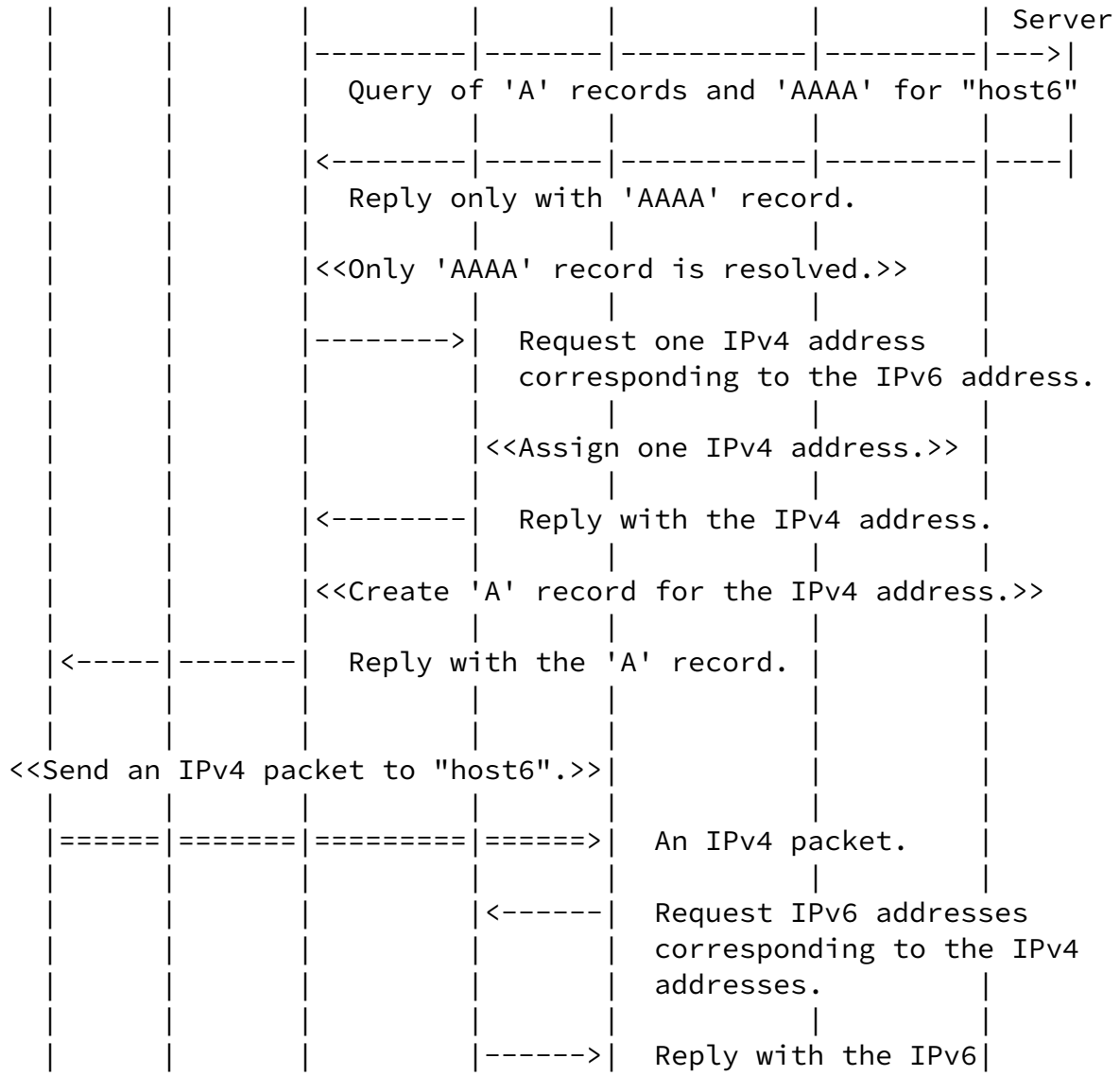



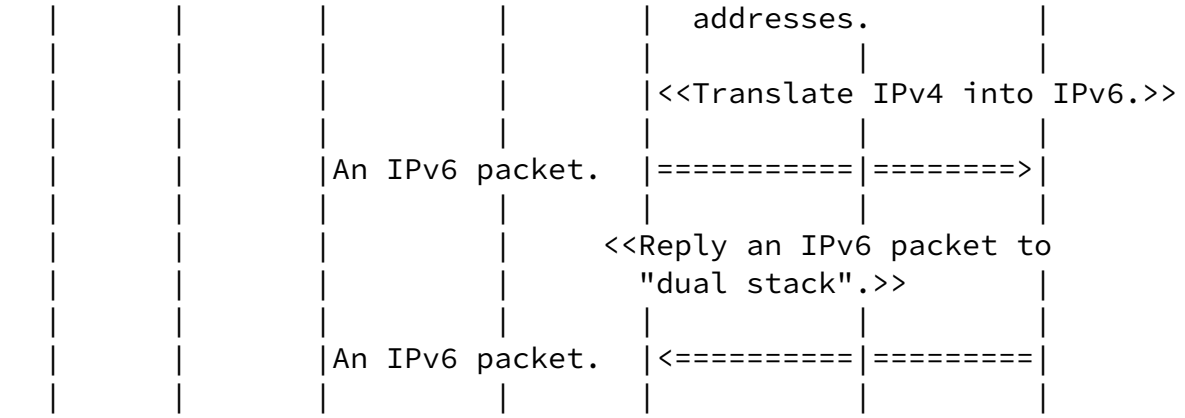

Figure 6: Example of BIH as API addition




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.

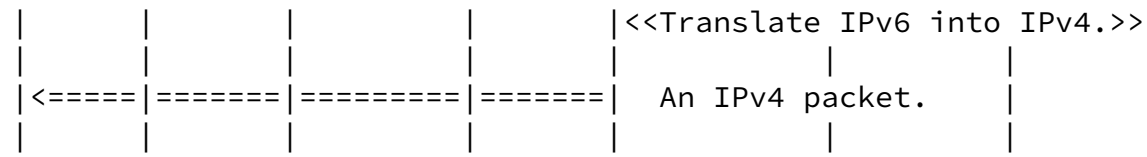



Figure 7: Example of BIH at network layer





## 4. Considerations

### 4.1. Socket API Conversion

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, but IPv4-only application are unlikely to need them.

## 4.2. ICMP Message Handling

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], 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 granularity in the node e.g., a single pool per node, or at some finer granularity such as per user or per process. In he case of large number of IPv4 applications would communicate with large number of IPv6 servers, the available address spaces may be exhausted. This should be quite rare event and changes will decrease as IPv6 support increases. The possible problem can also mitigated with smart management techniques of the address pool. For example, entries with longest inactivity time can be cleared and IPv4 addresess reused for creating new entries.

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 proposal, educated better quesses are welcome):

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 (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 subnet than destination addresses to ensure applications would not do on-link assumptions and would 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

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

Protocol translation for multicast is not supported.

## 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

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

The security consideration 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 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] does.

## 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, Ed Jankiewizh, Marnix Goossens, Ala Hamarsheh, and Julien Laganier 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

## 8.1. Normative References

[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.

[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.

[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. and R. 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.

## 8.2. Informative References

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





## 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 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.

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], 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

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