Behave WG
Internet-Draft
Obsoletes: 3338, 2767
(if approved)
Intended status: Standards Track
Expires: April 14, 2011

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

October 11, 2010

Dual Stack Hosts Using "Bump-in-the-Host" (BIH)
draft-ietf-behave-v4v6-bih-00

Abstract

This document describes the "Bump-In-the-Host" (BIH), a host based protocol translation mechanism that allows a subset of applications supporting only one IP address family to communicate with peers that are reachable or supporting only the other address family.

This specification addresses scenarios where a host is provided dual stack or IPv6 only network connectivity. In the dual stack network case, single address family applications in the host sometime will communicate directly with other hosts using the different address family. In the case of IPv6 only network or IPv6 only destination, IPv4 originated communications have to be translated into IPv6. The BIH makes the IPv4 applications think they talk to IPv4 peers and hence hides the 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

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 April 14, 2011.

Copyright Notice

outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction

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] and Bump-in-the-API (BIA) [RFC3338] technologies, which enables accommodation of significant set of the legacy IPv4-only applications in the IPv6-world.

Bump-In-the-Host is not a general purpose translation solution. The class of IPv4-only applications the described host-based NAT46 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: inserts 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]. The translation has the same benefits and drawbacks as SIIT.

In order to support communication between IPv4 applications and the target IPv6 hosts, pooled IPv4 addresses will be assigned through the extension name resolver.

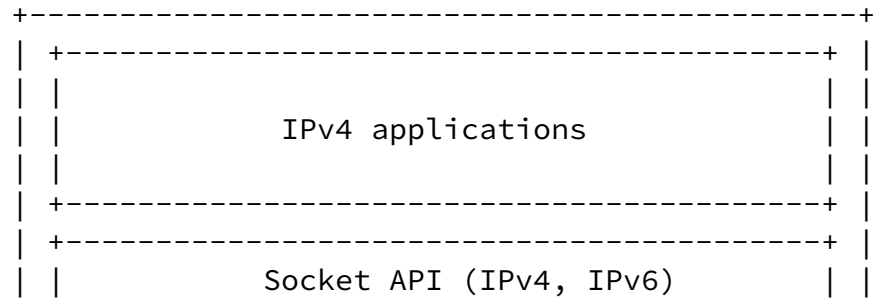
The BIH can be used whenever an IPv4-only application needs to communicate with an IPv6 peer, independently of the address families supported by the access network. Hence the access network can be IPv6-only or dual-stack capable.

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].

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".



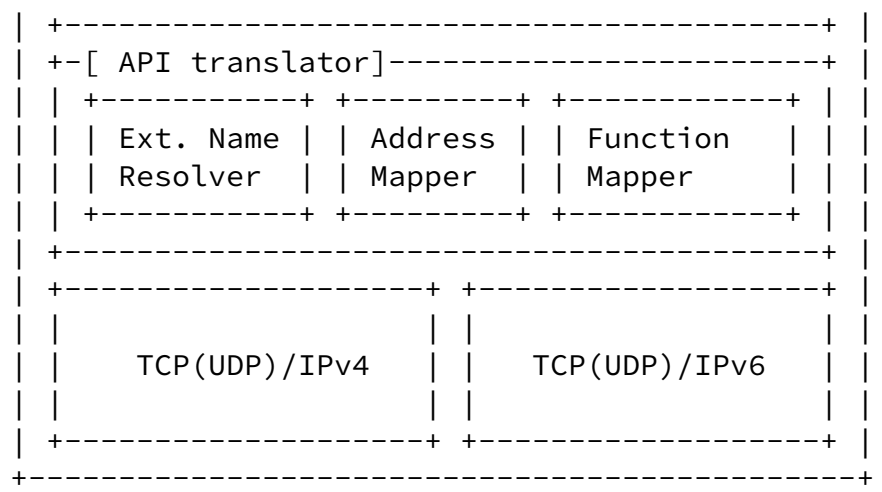
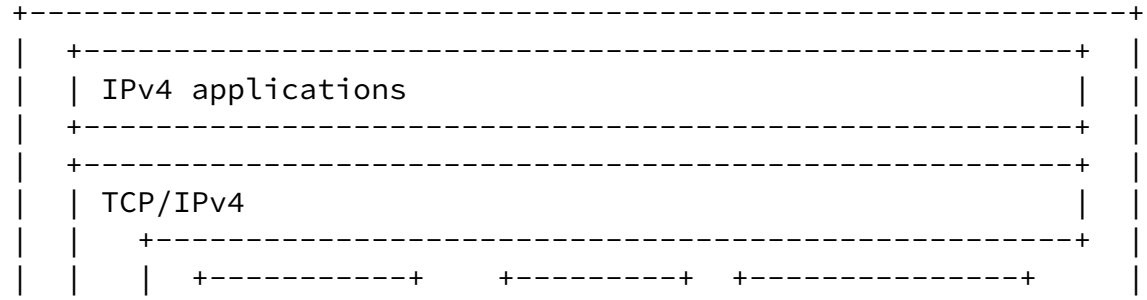


Figure 1: Architecture of the dual stack host using BIH at socket layer

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".



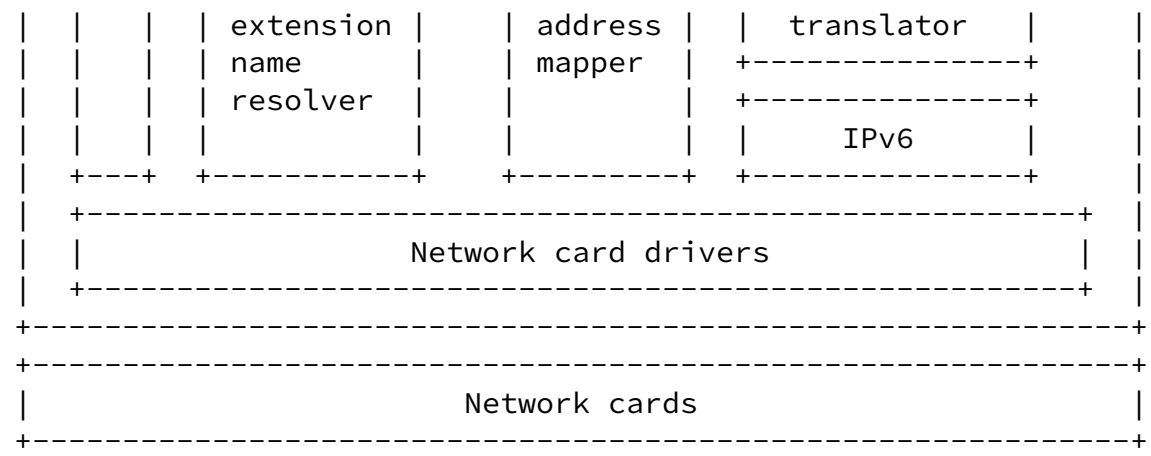


Figure 2: Architecture of the dual-stack host using BIH at network layer

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 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

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

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.

2.3. Extension Name Resolver

Extension Name Resolver returns a proper answer in response to the IPv4 or IPv6 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 an 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.

Application query	Network response	ENR behaviour
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

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 cache of the Address Mapper and is a local IP address. If an entry is found and queried address is locally generated, the ENR must initiate corresponding reverse DNS query for the real IP address.

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

2.4. Address Mapper

Address mapper ("the mapper" later on), maintains an IPv4 address pool in the case of dual stack network and IPv6 only network. The pool consists of private IPv4 addresses as per [RFC1918]. Also, mapper maintains a table consisting of pairs of these 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 pool, and registers a new entry into the table dynamically. 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 not a mapping entry for the IPv6 address.

(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 not a mapping entry for the IPv6 address. But ENR does not need an IPv4 address out of the pool, just to register both IPv4 and IPv6 addresses from 'A' and 'AAAA' records into a new entry into the table.

(3) When the function mapper gets a socket API function call from the

data received and there is not a mapping entry for the IPv6 source address.

When the resolver, translator, or the function mapper requests mapper to assign an IPv4 address corresponding to an IPv6 address, mapper, if required, selects and returns an IPv4 address out of the pool, and registers a new entry into the table dynamically. The following table describes how mappings are created into the table in each possible scenario:

Mapping table entry for	Access link type	Peer support	Created address mapping
(1) real IPv4	IPv4 or DS	v4	< no mapping needed >
(2) real IPv6	IPv6 or DS	v6	< no mapping needed >
(3) real IPv4	IPv6	v4 & v6	real IPv4 -> real IPv6
(4) real IPv6	IPv4	v4 & v6	< out of scope >
(5) local IPv4	IPv6 or DS	v6	local IPv4 -> real IPv6
(6) local IPv6	IPv4 or DS	v4	< out of scope >
(7) real IPv4	IPv6	v4	< out of scope >
(8) real IPv6	IPv4	v6	< out of scope >

Figure 4: Address Mapper's mapping table illustration

Below are examples for all eight scenarios:

(1) When the resolver gets an 'A' reply for application's 'A' query on access network supporting IPv4, there is no need to create mapping (or just stub mapping real IPv4 -> real IPv4).

(2) When the resolver gets an 'AAAA' reply for application's 'AAAA' query on access network supporting IPv6, there is no need to create mapping (or just stub mapping real IPv6 -> real IPv6).

(3) When the resolver gets both 'A' and 'AAAA' replies for application's 'A' query on IPv6-only access, there shall be mapping for real IPv4 to real IPv6.

(4) The scenario where the resolver gets both 'A' and 'AAAA' replies for application's 'AAAA' query on IPv4-only access is out of scope.

(5) When the resolver gets only an 'AAAA' record for the target host

name for application's 'A' request on IPv6 only or DS access network, a local IPv4 address will be given to application and mapping for local IPv4 address to real IPv6 address is created.

(6) The scenario where the resolver gets only an 'A' record for the target host name for application's 'AAAA' request on IPv4 only or DS access network is out of scope.

(7) The scenario where the resolver gets only an 'A' record for the

target host name for application's 'A' request on IPv6 only access network is out of scope.

(8) The scenario where the resolver gets only an 'AAAA' record for the target host name for application's 'AAAA' request on IPv4 only access network is out of scope.

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

3. Behavior and network Examples

Figure 5 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.

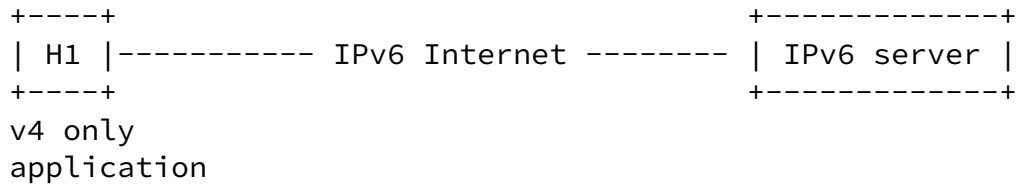
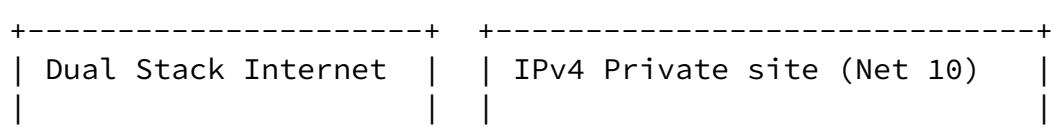


Figure 5: Network Scenario #1

Figure 6 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.



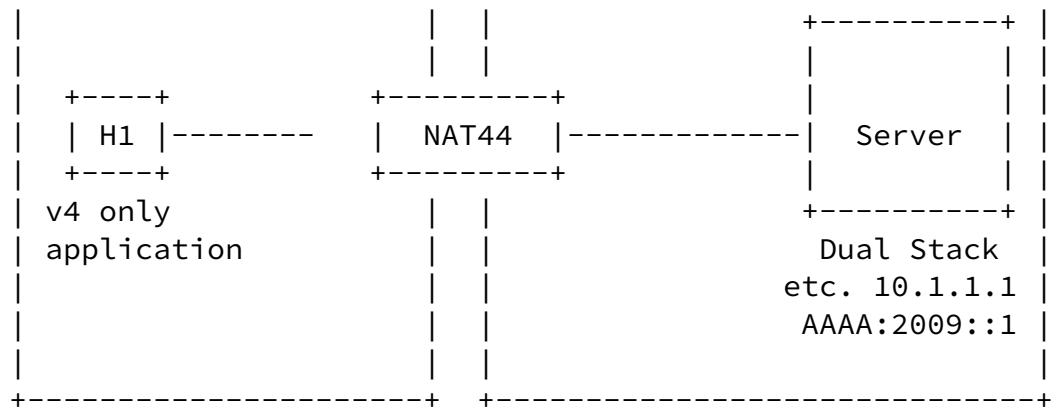
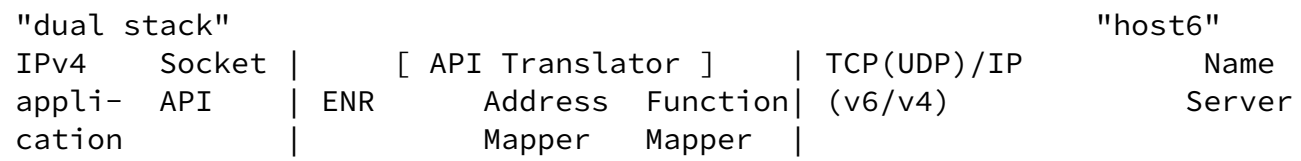
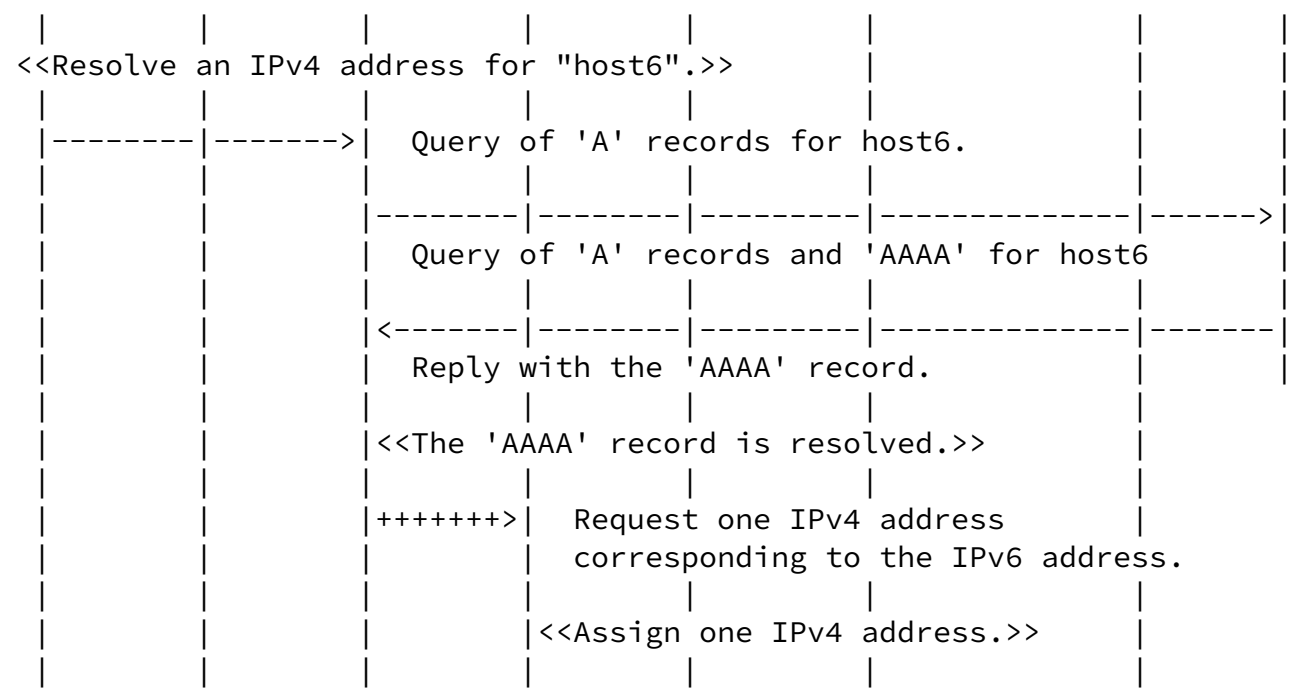
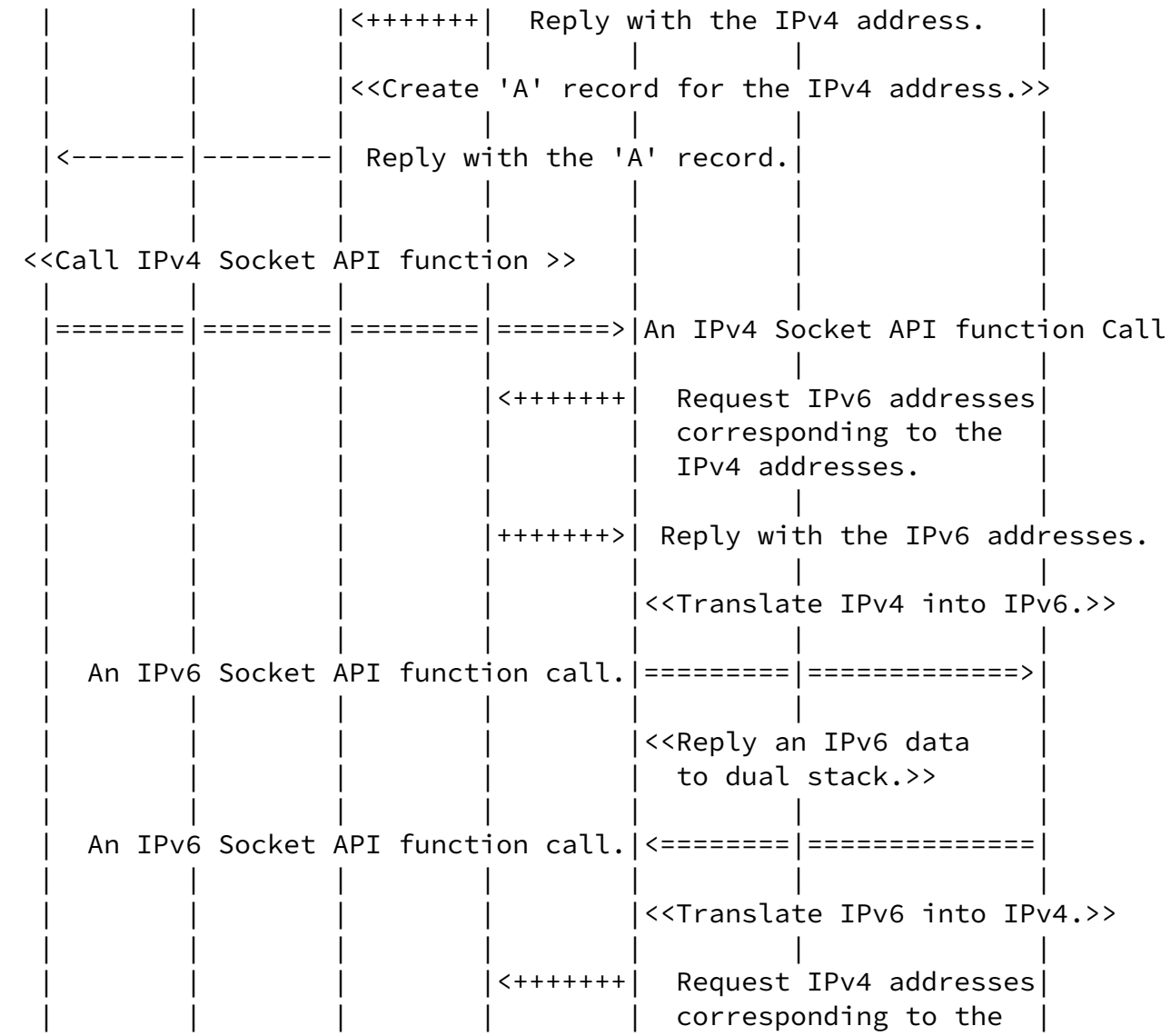


Figure 6: Network Scenario #2

Illustrations of host behavior in both implementation options are given here. Figure 7 illustrates the setup where BIH is implemented as a bump in the API, and figure 8 illustrates the setup where BIH is implemented as a bump in the stack.







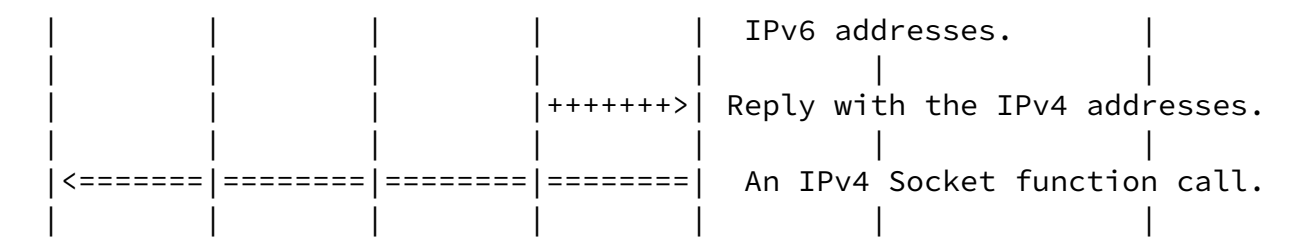
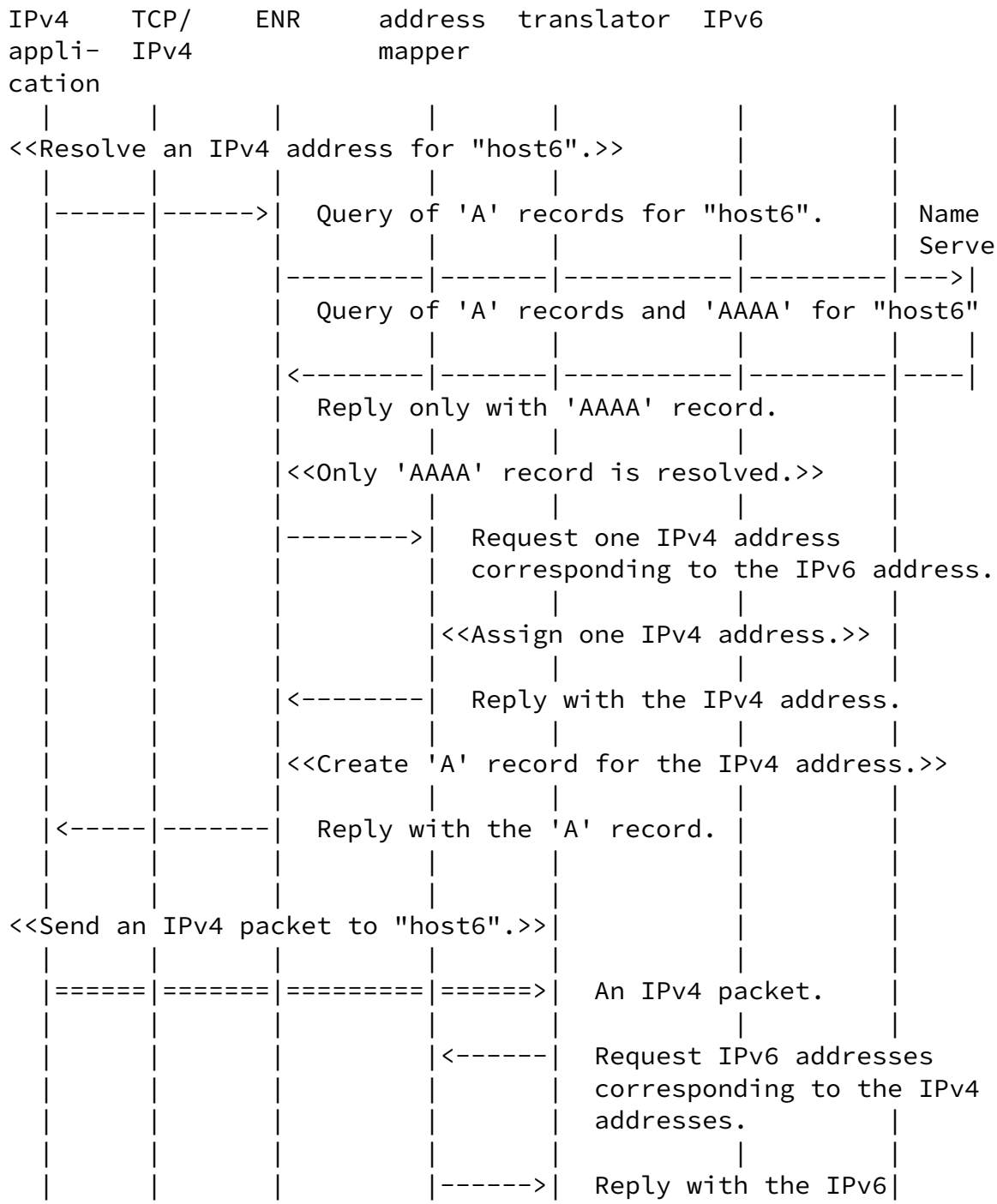


Figure 7: Example of BIH as API addition

"dual stack" "host6"


IPv4 appli- cation
TCP/ IPv4
ENR
address mapper
translator
IPv6
<<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.>>

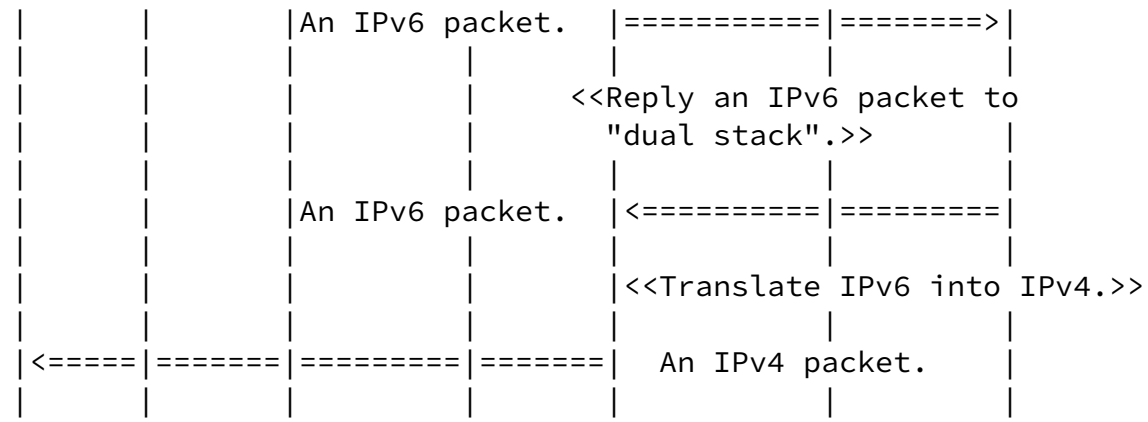


Figure 8: Example of BIH at network layer

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.

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. It can be implemented using raw socket.

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. 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. 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 (TO BE SELECTED) and, if possible, cease using BIH on IPv6-interface after an IPv4-enabled interface is activated.

4.4. Internally Assigned IPv4 Addresses

The IPv4 addresses, which are internally assigned to IPv6 target hosts out of the pool, are the private IPv4 addresses. IPv4 addresses, which are internally assigned to IPv6 target hosts out of the spool, never flow out from the host, and so do not negatively

affect other hosts.

The internally assigned IPv4 address, which applications see as the source address, MUST be from different subnet than the IPv4 addresses used by the address synthesis function. This approach ensures legacy applications realize they are not on the same link with their destination node and if needed, will trigger NAT traversal procedures such as keep-alive message sending.

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 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] 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, 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).

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 9

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