Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS)
draft-huang-behave-rfc2767bis-00

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79. This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified 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.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on April 21, 2010.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.
Abstract

This document describes the "Bump-In-the-Stack" (BIS) host based protocol translation mechanism that allows applications supporting only one IP address family to communicate with peers that are reachable or supporting only the other address family. Furthermore, this technology avoids need for unnecessary double protocol translation in the case where destination is dual-stack enabled.

This specification addresses scenarios where a host is provided dual stack, IPv6 only or IPv4 only network connectivity. In the dual stack network case, single address family applications in the host will communicate directly with other hosts reachable with the same address family. In the case of IPv6 only network or IPv6 only destination, IPv4-originated communications have to be translated into IPv6. IPv6 communications may have to be translated similarly to IPv4. In the scenario of single address family access network, but dual-stack destination, network based translation is always avoided. Technically, the BIS-enabled host resolves both IPv4 and IPv6 addresses of the destination and behaves according to received responses.

Acknowledgement of previous work

This document is an update to and directly derivative from Kazuaki TSUCHIYA, Hidemitsu HIGUCHI, and Yoshifumi ATARASHI’s [RFC2767], which similarly provides a dual stack host means to communicate with other IPv6 host using existing IPv4 applications. The original document was a product of the NGTRANS working group.

The changes in this document reflect three components:

1. Supporting IPv6 only network connections
2. Supporting IPv4 only network connections
3. Supporting Well Known Prefix
1. Introduction

RFC2767 [RFC2767] stated that there are few applications for IPv6 [IPV6] as compared with IPv4 [IPV4] in which a great number of applications are available. In order to advance the transition smoothly, it is highly desirable to make the availability of IPv6 applications increase to the same level as IPv4. Unfortunately, however, this is expected to take a long time. Meanwhile, there are scenarios where a dual stack host is connected to IPv6-only network but it is running IPv4-only applications, or a host is running IPv6-only applications while connected to IPv4-only network.

RFC2767 proposed a mechanism of dual stack hosts using the technique called "Bump-in-the-Stack" [BUMP] in the IP security area. The technique inserts modules, which snoop data flowing between a TCP/IPv4 module and network card driver modules and translate IPv4 into IPv6 and vice versa, into the hosts, and makes them self-
translators. When they communicate with the other IPv6 hosts, pooled IPv4 addresses are assigned to the IPv6 hosts internally, but the IPv4 addresses never flow out from them.

The network scenario specified in RFC2767 is a dual stack network, where IPv4 communication can be transported independently of IPv6. However, if the network provides only IPv6 transport, applications’s IPv4 packets have to be translated into IPv6. The opposite happens when the network is IPv4-only and application is IPv6-only capable.

This specification assumes that host knows it is connected with a dual stack network, IPv6-only network or IPv4-only network. The host learns that from layer 2 or from results of layer 3 IP address configuration mechanisms.

If the network which host is connecting with is IPv4 only network, then host’s IPv4 application will behave regularly, and it’s IPv6 application’s packets have to be translated into IPv4 packets.

If the network which host is connecting with is IPv6 only network, then host’s IPv6 application will behave regularly, and it’s IPv4 application’s packets have to be translated into IPv6 in order to communicate with IPv6 peers.

If the network which host is connecting with is dual stack network, then host will behave as what RFC 2767 originally described. However, even in the dual stack access network case it can be that the destination peer is only reachable via single address family. In case there is a conflict between the address family supported by an application and the peer, BIS is needed.

The obvious scenario where a destination peer is not reachable with the address family a host is provisioned with, but supports same address family application is using, is not covered by this document, as that requires additional network based protocol translation solution - i.e. double translation. However, the BIS technology can complement network based protocol translation such as [NAT64] and [PNAT].

Moreover, since the translation is automatically carried out with help of DNS protocol, most applications do not need to know whether target hosts are IPv6 or IPv4 ones. That is, this allows hosts to communicate with other IPv6 hosts using existing IPv4 applications and other IPv4 hosts using existing IPv6 applications; thus it seems as if peers are always dual stack hosts with applications for both IPv4 and IPv6.

This memo uses the words defined in [IPV4], [IPV6], and [TRANS-MECH].
2. Components

Dual stack hosts defined in RFC4213 [TRANS-MECH] need applications, TCP/IP modules and addresses for both IPv4 and IPv6. The proposed hosts in this memo have 3 newly defined modules: a translator, an extension name resolver (ENR) and an address mapper. These hosts communicate with other IPv6 hosts using IPv4 application and IPv4 hosts using IPv6 application.

Figure 1 illustrates the structure of the host in which new modules are installed.

```
+----------------------------------------------------------+
|  +---------------------------+  +---------------------------+  |
|  | IPv4 applications         |  | IPv6 applications         |  |
|  +---------------------------+  +---------------------------+  |
|  +---------------------------+  +---------------------------+  |
|  | TCP/IPv4                  |  |                  TCP/IPv6 |  |
|  |   +-----------------------+  +----------------------+    |  |
|  |   |  +-----------+  +---------------+  +---------+  |    |  |
|  |   |  | Extension |  |  translator   |  | address |  |    |  |
|  |   |  | Name      |  +---------------+  | mapper  |  |    |  |
|  |   |  | Resolver  |  +------+ +------+  |         |  |    |  |
|  |   |  |           |  | IPv6 | | IPv4 |  |         |  |    |  |
|  +---+  +-----------+  +------+ +------+  +---------+  +----+  |
|  +----------------------------------------------------------+  |
|  |                  Network card drivers                    |  |
|  +----------------------------------------------------------+  |
|  +----------------------------------------------------------+  |
|                        Network cards                           |
```

Figure. 1 Structure of the proposed dual stack host
2.1 Translator

It translates IPv4 into IPv6 and vice versa using the IP conversion mechanism defined in [SIIT] and [SIIT-NEW].

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.

When receiving IPv6 packets from IPv6 applications, translator converts IPv6 packet headers into IPv4 packet headers, but doesn’t do fragmentation as packet size is decreased, and sends packets to IPv4 networks. When receiving IPv4 packets from the IPv4 networks, it works symmetrically to the previous case. (NOTE: for packet received from network, if there is unsolicited IPv4 1500 byte packet coming from the network, which needs to be translated into IPv6 inside a host, then there is need to fragment the packets)

2.2 Extension Name Resolver (ENR)

ENR returns always a "proper" answer in response to the IPv4 and IPv6 application’s name resolution requests. In the case network does not return the IP address family application requested, the ENR will requests the address mapper to assign a local IP address corresponding to received IP address, and then synthesize ‘A’ or ‘AAAA’ record for the assigned IP address. E.g. in case of AAAA response is received while application asked for A, the address mapper will select a local IPv4 address, and ENR will synthesize ‘A’ record based on it.

The application typically sends a query to a name server to resolve ‘A’, ‘AAAA’, or both, records for the target host name. ENR snoops the query, then, if required, creates another query to ensure both ‘A’ and ‘AAAA’ records are requested for the host name, and sends the queries to the DNS server.

The following table illustrates ENR behaviour. The address application receives, and whether synthesis happens, is independent of the address families a host is actually provisioned with.
NOTE: This action is similar to that of the DNS64 in the network side, here it happens on the host.

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 in BIS-module, same limitations arise as when DNS record synthesis is done on network. Anyway, it depends on the host to implement recursive DNS server by itself.

2.3 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 can consists of private IPv4 addresses [RFC1918]. Also, mapper maintains a table consisting of pairs of these locally selected IPv4 addresses and a destinations’ IPv6 addresses.

In the case of dual-stack networks and IPv4 only networks, mapper creates locally used IPv6 addresses by concatenation of well known prefix (WKP) and destination’s IPv4 address. The mapper maintains a table consisting of pairs of local IPv6 addresseses and destinations’ IPv4 addresses.

When the resolver or the translator requests mapper to assign an IPv4 address corresponding to an IPv6 address or assign an IPv6 address corresponding to an IPv4 address, mapper, if required, selects and returns an IPv4 address out of the pool, or concatenated IPv6 address, and registers a new entry into the table dynamically. The following table describes how mappings are created into the table in each scenario (note that the scenario of destination not supporting the same address family host is provisioned with, and where network based translation assistance would be needed, is shown in the table for the sake of completeness only (7 & 8)):  

<table>
<thead>
<tr>
<th>Application</th>
<th>Network</th>
<th>ENR behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>response</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>&lt;return as is&gt;</td>
</tr>
<tr>
<td>A</td>
<td>AAAA</td>
<td>&lt;synthesize A record&gt;</td>
</tr>
<tr>
<td>AAAA</td>
<td>AAAA</td>
<td>&lt;return as is&gt;</td>
</tr>
<tr>
<td>AAAA</td>
<td>A</td>
<td>&lt;synthesize AAAA record&gt;</td>
</tr>
</tbody>
</table>

Table 1 ENR behaviour illustration
Mapping table | Access link | Peer | Created
---|---|---|---
entry for | type | support | 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 | real IPv6 -> real IPv4
(5) local IPv4 | IPv6 or DS | v6 | local IPv4 -> real IPv6
(6) local IPv6 | IPv4 or DS | v4 | local IPv6 -> real IPv4
(7) real IPv4 | IPv6 | v4 | real IPv4 -> synthetic IPv6
(8) real IPv6 | IPv4 | v6 | real IPv6 -> synthetic IPv6

Table 2. Address Mapper’s mapping table illustration

Below are examples for all six 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) When the resolver gets both ‘A’ and ‘AAAA’ replies for application’s ‘AAAA’ query on IPv4-only access, there shall be mapping for real IPv6 to real IPv4.

(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) When the resolver gets only an ‘A’ record for the target host name for application’s ‘AAAA’ request on IPv4 only or DS access network, a local IPv6 address will be given to application and mapping for local IPv6 address to real IPv4 address is created.

(7) When the resolver gets only an ‘A’ record for the target host name for application’s ‘A’ request on IPv6 only access network, a double translation would be required and thus is out of the scope of this document.

(8) When the resolver gets only an ‘AAAA’ record for the target host name for application’s ‘AAAA’ request on IPv4 only access
network, a double translation would be required and thus is out of the scope of this document.

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. Action Examples -- dual stack network and IPv6 only peer

This section describes action of the proposed dual stack host called "dual stack," which communicates with an IPv6 peer called "host6" using an IPv4 application on dual stack network.

3.1 Originator behavior

This subsection describes the originator behavior of "dual stack." The communication is triggered by "dual stack."

The application sends a query to its name server to resolve 'A' records for "host6."

The resolver snoops the query, then creates another query for 'AAAA' to resolve both 'A' and 'AAAA' records for the host name, and sends it to the server. In this case, only the 'AAAA' record is resolved, so the resolver requests the mapper to assign an IPv4 address corresponding to the IPv6 address.

NOTE: In the case of communication with an IPv4 host, the 'A' record is resolved and then the resolver returns it to the application as is. There is no need for the IP conversion as shown later.

The mapper selects an IPv4 address out of the pool and returns it to the ENR.

The ENR creates the 'A' record for the assigned IPv4 address and returns it to the application.

NOTE: See subsection 6.3 about the influence on other hosts caused by an IPv4 address assigned here.

The application sends an IPv4 packet to "host6."

The IPv4 packet reaches the translator. The translator tries to translate the IPv4 packet into an IPv6 packet but does not know how to translate the IPv4 destination address and the IPv4 source address. So the translator requests the mapper to provide mapping entries for them.
The mapper checks its mapping table and finds entries for each of them, and then returns the IPv6 destination address and the IPv6 source address to the translator.

NOTE: The mapper will register its own IPv4 address and IPv6 address into the table beforehand. See subsection 2.3.

The translator translates the IPv4 packet into an IPv6 packet and then fragments the IPv6 packet if necessary and sends it to an IPv6 network.

The IPv6 packet reaches "host6." Then "host6" sends a new IPv6 packet to "dual stack."

The IPv6 packet reaches the translator in "dual stack."

The translator gets mapping entries for the IPv6 destination address and the IPv6 source address from the mapper in the same way as before.

Then the translator translates the IPv6 packet into an IPv4 packet and tosses it up to the application.
The following diagram illustrates the action described above:

```
"dual stack"                                            "host6"
IPV4    TCP/  extension address translator IPv6
appli- IPv4 name mapper
    |       |           |         |
<<Resolve an IPv4 address for "host6".>>       |         |
|------|------|         |-------|-----------|---------|-->
|      |       |  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.            |
```

Figure 2 Action of the originator (1/2)
3.2 Recipient behavior

This subsection describes the recipient behavior of "dual stack." The communication is triggered by "host6."

"host6" resolves the 'AAAA' record for "dual stack" through its name server, and then sends an IPv6 packet to the IPv6 address.

The IPv6 packet reaches the translator in "dual stack."

The translator tries to translate the IPv6 packet into an IPv4 packet but does not know how to translate the IPv6 destination address and the IPv6 source address. So the translator requests the mapper to provide mapping entries for them.
The mapper checks its mapping table with each of them and finds a
mapping entry for the IPv6 destination address.

NOTE: The mapper will register its own IPv4 address and IPv6 address
into the table beforehand. See subsection 2.3.

But there is not a mapping entry for the IPv6 source address, so the
mapper selects an IPv4 address out of the pool for it, and then
returns the IPv4 destination address and the IPv4 source address to
the translator.

NOTE: See subsection 6.3 about the influence on other hosts caused by
an IPv4 address assigned here.

The translator translates the IPv6 packet into an IPv4 packet and
tosses it up to the application.

The application sends a new IPv4 packet to "host6."

The following behavior is the same as that described in subsection
3.1.
The following diagram illustrates the action described above:

![Diagram](https://example.com/diagram.png)

Figure 3 Action of the recipient

4. Action Examples -- IPv6 only network and dual-stack peer

This section describes action of the proposed dual stack host called "dual stack," which communicates with a dual stack peer called "host46" using an IPv4 only application while provisioned only with IPv6 network connectivity.

4.1 Originator behavior

This subsection describes the originator behavior of "dual stack." The communication is triggered by "dual stack."

The application sends a query to its name server to resolve 'A' records for "host46."
The resolver snoops the query, then creates another query for 'AAAA' to resolve both 'A' and 'AAAA' records for the host name, and sends it to the server. In this case, both the 'A' and 'AAAA' records are resolved, so the resolver does not need to request the mapper to allocate any IPv4 addresses from its pool, but only to store mapping between received destination’s IPv4 and IPv6 addresses.

In this case of communication with an dual-stack host, the 'A' record is also resolved and the resolver can return it to the application as is.

The application sends an IPv4 packet to "host46."

The IPv4 packet reaches the translator. The translator tries to translate the IPv4 packet into an IPv6 packet but does not know how to translate the IPv4 destination address and the IPv4 source address. So the translator requests the mapper to provide mapping entries for them.
The mapper checks its mapping table and finds entries for each of them, and then returns the IPv6 destination address and the IPv6 source address to the translator.

NOTE: The mapper will register its own IPv4 address and IPv6 address into the table beforehand. See subsection 2.3.

The translator translates the IPv4 packet into an IPv6 packet then fragments the IPv6 packet if necessary and sends it to an IPv6 network.

The IPv6 packet reaches "host46." Then "host46" sends a new IPv6 packet to "dual stack."

The IPv6 packet reaches the translator in "dual stack."

The translator gets mapping entries for the IPv6 destination address and the IPv6 source address from the mapper in the same way as before.

Then the translator translates the IPv6 packet into an IPv4 packet and tosses it up to the application.
The following diagram illustrates the action described above:

```
"dual stack"                          "host46"
IPv4    TCP/extension address translator IPv6
appli- IPV4 name mapper
cation resolver

<<Resolve an IPv4 address for "host46".>>

----- ------  Query of 'A' records for "host46". | Name
     |       |-----------------------------|--->
|     |       | Query of 'A' records and 'AAAA' for "host46"
|     |       |<--------|-------|-----------|---------|----|
|     |       |  Reply with 'A' and 'AAAA' records.   |
|     |       |<<Both 'AAAA' and 'A' record is resolved.>>
|     |----------|-----------------------------|--->
|     | Request mapping of received IPv4 address | corresponding to the received IPv6 address.
|     |<-----|-------  Reply with the 'A' record. |
```

Figure 4 Action of the originator (1/2)
4.2 Recipient behavior

The recipient behaviour is exactly the same as in 3.2.

5. Action Examples -- IPv4 only network and IPv4 only peer

This section describes action of the proposed dual stack host called "dual stack," which communicates with an IPv4 peer called "host4" using an IPv6 application.

5.1 Originator behavior

This subsection describes the originator behavior of "dual stack."
The communication is triggered by "dual stack."

Figure 4 Action of the originator (2/2)
The application sends a query to its name server to resolve ’AAAA’ records for "host4."

The resolver snoops the query, then creates another query for ’A’ to resolve both ’A’ and ’AAAA’ records for the host name, and sends it to the server. In this case, only the ’A’ record is resolved, so the resolver requests the mapper to assign an IPv6 address corresponding to the IPv4 address.

NOTE: In the case of communication with a dual-stack host, the ’AAAA’ record is also resolved and then the resolver returns it to the application as is. The mapper will create mapping between the IPv4 and IPv6 addresses similarly as in 4.1.

The mapper concatenates IPv6 WKP with the resolved IPv4 address and returns it to the resolver.

The resolver creates the ’AAAA’ record for the assigned IPv6 address and returns it to the application.

NOTE: See subsection 6.3 about the influence on other hosts caused by an IPv6 address assigned here.

The application sends an IPv6 packet to "host4."

The IPv6 packet reaches the translator. The translator tries to translate the IPv6 packet into an IPv4 packet but does not know how to translate the IPv6 destination address and the IPv6 source address. So the translator requests the mapper to provide mapping entries for them.
The mapper checks its mapping table and finds entries for each of them, and then returns the IPv4 destination address and the IPv4 source address to the translator.

NOTE: The mapper will register its own IPv4 address and IPv6 address into the table beforehand. See subsection 2.3.

The translator translates the IPv6 packet into an IPv4 packet and sends it to an IPv4 network.

The IPv4 packet reaches "host4." Then "host4" sends a new IPv4 packet to "dual stack."

The IPv4 packet reaches the translator in "dual stack."

The translator gets mapping entries for the IPv4 destination address and the IPv4 source address from the mapper in the same way as before.

Then the translator translates the IPv4 packet into an IPv6 packet and tosses it up to the application.
The following diagram illustrates the action described above:

```
"dual stack"                                            "host4"
IPv6    TCP/  extension address translator IPv4
appli-  IPv6  name       mapper
ation    resolver

<<Resolve an IPv6 address for "host4".>>

----- ------
Query of 'AAAA' records for "host4". Name

--- --------
Query of 'A' records and 'AAAA' for "host4"

---------- ------- ------
Reply only with 'A' record.

<<Only 'A' record is resolved.>>

----- ------
Request one IPv6 address
corresponding to the IPv4 address.

<<Assign one IPv6 address.>>

------ -------
Reply with the IPv6 address.

<<Create 'AAAA' record for the IPv6 address.>>

------ ------
Reply with the 'AAAAA' record
```

Figure 6 Action of the originator (1/2)
5.2 Recipient behavior

This subsection describes the recipient behavior of "dual stack." The communication is triggered by "host4."

"host4" resolves the 'A' record for "dual stack" through its name server, and then sends an IPv4 packet to the IPv4 address.

The IPv4 packet reaches the translator in "dual stack."

The translator tries to translate the IPv4 packet into an IPv6 packet but does not know how to translate the IPv4 destination address and the IPv4 source address. So the translator requests the mapper to provide mapping entries for them.
The mapper checks its mapping table with each of them and finds a mapping entry for the IPv4 destination address.

NOTE: The mapper will register its own IPv4 address and IPv6 address into the table beforehand. See subsection 2.3.

But there is not a mapping entry for the IPv4 source address, so the mapper concatenates IPv6 WKP with the IPv4 source address and then returns the IPv6 destination address and the IPv6 source address to the translator.

NOTE: See subsection 6.3 about the influence on other hosts caused by an IPv6 address assigned here.

The translator translates the IPv4 packet into an IPv6 packet and tosses it up to the application.

The application sends a new IPv6 packet to "host4."

The following behavior is similar as described in subsection 3.1.
The following diagram illustrates the action described above:

```
"dual stack"        "host4"
IPv6    TCP/  extension address translator IPv4
appli- IPv6 name mapper
ation   resolver

<<Receive data from "host4".>>
        |       |         |       |           |         |
        |       |         |       |           |         |
        |       |         |An IPv4 packet.  |<==========|=========|
        |       |         |       |           |         |
        |       |         |------>|  Request IPv6 addresses
        |       |         |       |  corresponding to the IPv4
        |       |         |       |  addresses.         |
        |       |         |------>|  Reply with the IPv6
        |       |         |       |  addresses.         |
        |       |         |<------|  Request IPv6 addresses
        |       |         |       |  corresponding to the IPv4
        |       |         |       |  addresses.         |
        |       |         |<------|  Reply with the IPv6
        |       |         |       |  addresses.         |
        |       |         |------>|  Translate IPv4 into IPv6.
        |       |         |       |           |         |
        |       |         |<====|  Translate IPv4 into IPv6.
        |       |         |       |           |         |
        |       |         |       |<<Translate IPv4 into IPv6.>>
        |       |         |       |           |         |
        |       |         |       |<<Translate IPv6 into IPv4.>>
        |       |         |       |           |         |
        |       |         |An IPv4 packet.  |<==========|=========|
        |       |         |       |           |         |
        |       |         |<<Reply an IPv6 packet to "host4".>>
```

Figure 7 Action of the recipient

6. Considerations

This section considers some issues of the proposed dual stack hosts.

6.1 IP conversion

In common with NAT [RFC3022], IP protocol translation has to translate IP addresses embedded in application layer protocols, such as FTP [RFC959]. Translation of all such applications is a difficult problem.

6.2 IPv4 address pool and mapping table
The pool, for example, consists of private addresses [RFC1918]. So a large address space can be used for the spool. Nonetheless, IPv4 addresses in the spool will be exhausted and cannot be assigned to IPv6 target hosts, if the host communicates with a great number of other IPv6 hosts and the mapper never frees entries registered into the mapping table once. To solve the problem, for example, it is desirable for the mapper to free the oldest entry in the mapping table and re-use the IPv4 address for creating a new entry.

6.3 Internally assigned 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.

6.4 Well Known Prefix Support

The address mapper shall use the same WKP as will be allocated by IETF/IANA for [ADDRFORMAT].

7. Applicability and Limitations

This section considers applicability and limitations of the proposed dual stack hosts.

7.1 Applicability

The mechanism can be useful for people in the initial stages of IPv6 transition when significant percentage of applications are not yet modified into IPv6 realm. BIS can also help users who cannot upgrade their important legacy applications for any reason, such as due lack of maintenance support. The reason is that BIS allows hosts to communicate with IPv6 hosts using existing IPv4 applications, and that people can get connectivity for both IPv4 and IPv6 even if they do not have IPv6 applications.

Note that BIS can also work in conjunction with a complete IPv6 stack. People can communicate with both IPv4 hosts and IPv6 hosts using IPv4 applications via the mechanism, and can also communicate with IPv6 hosts using IPv6 applications via the complete IPv6 stack.

Furthermore, as protocol translation is supported also from IPv6 to IPv4, application developers can focus on implementing only IPv6 support.

7.2 Limitations
The mechanism is valid only for unicast communication, but invalid for multicast communication. Multicast communication needs another mechanism.

BIS allows hosts to communicate with IPv6 enabled hosts using existing IPv4 applications, but this cannot be applied to IPv4 applications which use special IPv4 options since it is impossible to translate IPv4 options into IPv6. Similarly it is impossible to translate any IPv6 option headers into IPv4, except for fragment headers and routing headers. So IPv6 inbound communication having the option headers may be rejected.

The BIS does not support the scenario where access network supports only the different address family to what both an application and related server support. In such a case double translation is required. The BIS can be used as component in such as setup.
In common with NAT [RFC3022], IP conversion needs to translate IP addresses embedded in application layer protocols, which are typically found in FTP [RFC959]. So it is hard to translate all such applications completely. However, this limitation is common for all address and protocol translators.

It may be impossible, without transport layer encapsulation, for the hosts using BIS to utilize the security above network layer, since the data may carry IP addresses.

Finally, BIS does not work well with secure DNS if only the extension name resolver is used. If the host’s DNS resolver is updated, then also DNSSEC can work.

8. Security Considerations

This section considers security of the proposed dual stack hosts.

The hosts can utilize the security of all layers like ordinary IPv4 communication when they communicate with IPv4 hosts using IPv4 applications via the mechanism. Likewise they can utilize the security of all layers like ordinary IPv6 communication when they communicate with IPv6 hosts using IPv6 applications via the complete IPv6 stack. However, unfortunately, they can not utilize the security above network layer when they communicate with IPv6 hosts using IPv4 applications via the mechanism. The reason is that when the protocol data with which IP addresses are embedded is encrypted, or when the protocol data is encrypted using IP addresses as keys, 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.

9. References


[DNS64] Bagnulo, M., Sullivan, A., Matthews, P., van Beijnum, I., "DNS64: DNS extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", draft-ietf-behave-dns64-00, July 2009, work-in-progress


10. Acknowledgements

The authors gratefully acknowledge the many helpful advice from Dan Wing and Dave Thaler for initiating this work, thanks mailing list discussion from Mohamed Boucadair, Yiu L. Lee, James Woodyatt, Lorenzo Colitti, Qibo Niu, Lin Xiao, and Pierrick Seite. Contributions from Gang Chen, Bo Zhou, Dapeng Liu, Hong Liu, Tao Sun et al. in the development of this document.

11. 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
Hermianskatu 12 D
FI-33720 TAMPERE
Finland

Email: teemu.savolainen@nokia.com