Behave A. Hamarsheh
Internet-Draft ETRO/Vrije Universiteit Brussel
Obsoletes: 3338 (if approved) M. Goossens
Intended status: Experimental ETRO/Vrije Universiteit Brussel
Expires: March 13, 2011 September 09, 2010

Hosts with Any Network Connectivity Using "Bump-in-the-API"(BIA) draft-hamarsheh-behave-biav2-02

Abstract

This document specifies a mechanism for hosts with any network connectivity (IPv4 only, IPv6 only, or dual IPv4/IPv6 connectivity) to run applications of any capability (IPv4 only, IPv6 only, or dual IPv4/IPv6) without any modification to those applications. It is a generalisation of a previous experimental protocol called "Bump-in-the-API" (BIA) [RFC3338]. New mechanism of BIA allows a changeover between the application layer and the IP communication layers from IPv4 to IPv6 and vice versa or IPv6 to IPv4 and vice versa, without requiring those applications to be converted in addressing capabilities, effectively shielding the application layer from IPv4 or IPv6 connectivity. This is considered by the authors to be one of the essential conditions for the transition to IPv6 in the Internet to be successful.

Table of Contents:

1. Motivation And Introduction

1.1 Motivation

It is probably important to give a brief analysis first of one of the blocking factors withholding the wide-spread introduction of IPv6 in order to fully understand why the here proposed BIA is considered an essential component in the unlocking of IPv6.

At the inception of IPv6 it was - rather naively - presumed that all parties involved with the Internet would be eager to make the changeoverand that the transition would happen spontaneously.

It is now quite generally acknowledged that some human and commercial factors preventing a spontaneous transition have been largely underestimated. In thetransition to IPv6 there are essentially two parties involved: network providers and end-users.

The benefits of using IPv6 are almost entirely for the network providers, while the end-users have only potentially indirect benefits from better network operation. No drive to make the changeover should be expected from the majority of end-users, as they have probably little to gain. The network providers can expect benefits,but they are obviously dependent on the willingness of their end-users to make any changeover. The result is some kind of deadlock: no (commercial) network provider is going to force the customers to make the changeover against their will. So making the transition transparent to the end-user is the key in any transition to IPv6. The average end-users are not really aware about what goes on in the network layer, and even if they do, they usually could not care less. It does not matter much to them if their applications are communicating using IPv4 or IPv6. But, while there is no drive to be expected from the end-users for any transition to IPv6, the vast majority would not object to the transition on condition they can go on using their applications as before.

While the first impression is that applications are not affected by the changeover on the IP layer from IPv4 to IPv6, this is unfortunately not true. The applications are using IP addresses, and hence should be capable of dealing with the longer IPv6 addresses when having to communicate over IPv6.

Expecting all applications to be modified to be capable of dealing with the longer IPv6 addresses is rather naive. Apart from the "standard" Internet applications with rather good support such as web browsers, email programs, etc. that can be expected to be IPv6 enabled, there are thousands of other applications, some of them are written by small companies (of which some may be out of business) and others are even "home-made". For some applications, Internet communication is only a side-issue, for example for registering and/or checking for updates, and upgrading to become IPv6 compatible is probably not a high priority. It is to be expected that a large proportion of applications will only be modified to be IPv6 compatible when IPv6 usage gets into full swing. And even if the

IPv6 capable new versions of application software are made available, it is again rather naive to expect all end-users to do the required updating of all the software on their system.
The end-users MAY be willing to accept a changeover to IPv6, but will NOT accept that some of their applications will no longer work as before. From this observation it becomes obvious that it is absolutely essential that provisions are standard installed and enabled on any general purpose machine (the vast majority of systems connected to the Internet) that is provided for IPv6 communication and potentially has to run IPv4-only applications to continue communicating as before when communicating using IPv6. While the demand for mandatory provisions on every general purpose machine capable of communicating using IPv6 may seem a tall order, it should be realized that this approach is much more realistic then expecting all applications to be made IPv6 compatible: compared to thousands of applications that would need conversion requiring all application developers to follow suit, the number of communication stack implementations on general purpose machines is very small and is made by only a handful of developers.
While somewhat less of an urgent issue, the solution should be general enough to handle the reverse problem as well: an IPv6 only application should be able to communicate on a machine with IPv4 only connectivity, or dual IPv4/IPv6 connectivity when communicating using IPv4 with remote hosts that have IPv4-only connectivity.While this looks like a move in the wrong direction in the context of transition towards IPv6, this capability is also important to break the slowdown of the development of IPv6 compatible applications, as described in [RFC2460]. Little effort is being invested into making applications IPv6 capable, as almost no machines currently have IPv6 connectivity. BIA allows to using these IPv6 capable applications to run on the IPv4 infrastructure, removing the practical limitation that IPv6 applications cannot be used at this time.
Other practical issues are blocking the deployment of IPv6, such as the lack of IPv6 support in public access networks, the lack of real auto configuration between IPv4 and IPv6 connectivity, incompatibility in IPv4/IPv6 connectivity of hosts, etc. Solutions to these other practical issues are being investigated currently by the authors.

1.2 Introduction

The original BIA is an experimental function intended at allowing IPv4 only applications on dual stack (dual connectivity) hosts to communicate over IPv6 with remote IPv6 only applications. It was also only useable in the specific context described.
The proposed BIA is a generalisation of the original concept, allowing any mixture of IPv4/IPv6 type capable applications to communicate over

any IPv4/IPv6 connections with any IPv4/IPv6 type capable remote applications. BIA effectively decouples application IPv4/IPv6 capability from IPv4/IPv6 connectivity, and all allows IPv4/IPv6 incompatibility between two communicating applications.
The concept is quite simple: BIA essentially does internal address translation where necessary between IPv4 and IPv6 addresses in between the application and the communication stack; functionally, it can be compared to an internal NAT [RFC3022] between the communication stack and the application layer. Conceptually BIA is an adaptation layer that needs to be inserted between the application layer and the IP communication stack as an API layer on top of the native API functions, offering the same API functions as the native ones to the application layer. In an optimized implementation, it can probably better be implemented as an internal modification to the API itself.

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], [RFC4213], [RFC2767], [RFC3338], and [I-D.draft-huang-behave-rfc3338bis].

2. Applicability and Related techniques

The term IPv4/IPv6 connectivity will be used, rather than stack, since it is the connectivity that specifies whether the machine can communicate using IPv4, IPv6 or both, and not only the implementation of the IP stacks inside the machine; e.g. a dual stack machine has both IPv4 and IPv6 stacks implemented, but may have only IPv4 or IPv6 network connectivity due to the type of network it is connected to, or may have to use IPv6 because the remote has only IPv6 connectivity.

2.1 Applicability

The BIA is a mechanism that should be mandatory installed and enabled on hosts potentially having to run applications with incompatible IPv4/IPv6 addressing capability regarding to their IPv4/IPv6 network connectivity. For example, IPv4 only applications that have to communicate over IPv6; or IPv6 only applications having to communicate over IPv4 only connectivity. It allows an IPv4 only application which is running on the local host to communicate over IPv6 with another IPv4/IPv6 application on another host without any modification.
It is important to note that the mechanism assumes that the host knows whether it is connected via dual IPv4/IPv6 connectivity, IPv4 only connectivity, or IPv6 only connectivity (this is an implementation issue and will not be discussed here). Table 1 describes the scenarios of all IPv4/IPv6 capability types of applications running over all

possible types of host connectivities. Only the situations with incompatibility between application IPv4/IPv6 capability and IPv4/IPv6 connectivity are listed.

Source Host			Destination Host
Appl. Version	Host Connectivity	Network	Host Connectivity
IPv4	IPv6	<-IPv6->	IPv6
IPv4	IPv6	<-IPv6->	IPv4/IPv6
IPv6	IPv4	<-IPv4->	IPv4
IPv6	IPv4	<-IPv4->	IPv4/IPv6
IPv4	IPv4/IPv6	<-IPv6->	IPv6
IPv6	IPv4/IPv6	<-IPv4->	IPv4

Table 1: List all the scenarios treated by BIA mechanism

2.2 Related Techniques

The original BIA mechanism is customized for dual stack hosts. BIA is a mechanism that is inserted between the socket API module and the TCP/IP module. The main purpose of this mechanism is to make the IPv4 applications communicate with applications that can only communicate using IPv6 (IPv6 only connectivity and/or IPv6 only application) without any modification on those IPv4 applications. This would be achieved by translating the IPv4 socket API functions into IPv6 socket API functions and vice versa.
BIS mechanism [RFC2767] allows host to communicate with other IPv6 hosts using existing IPv4 applications. It is also customized for dual stack hosts. The technique uses SIIT [RFC2765] to translate the IPv4 traffic into IPv6 traffic and vice versa. However, this mechanism uses translator which is inserted between the TCP/IP module and the network card driver. The limitations of this mechanism are similar to the SIIT limitations concerning the IP header translation methods. Its implementation is also fully dependent on the network interface driver.

3. Host Configurations using BIA

BIA can be installed on three different host configurations regarding to IPv4/IPv6
connectivity:
1. IPv4 only host: only IPv4 connectivity.
2. IPv6 only host: only IPv6 connectivity.
3. IPv4/IPv6 host: both IPv4 and IPv6 connectivity.

The connectivity of the local host for communication with a remote host is actually decided by several factors:
- The implementation of stack(s) in the local (IPv4 only stack, IPv6 only stack, or dual stack).
- The network connectivity of the local host (IPv4 network connectivity only, IPv6 network connectivity only, both IPv4 and IPv6 network connectivity).
- The connectivity of the remote machine (IPv4 only connectivity, IPv6 only connectivity, both IPv4 and IPv6 connectivity).

This means that the connectivity of a host, even with dual stack implementation, is dynamic: it depends on the network connectivity, which may change (e.g. a laptop that may be regularly connected to different networks over time) and/or the connectivity of the remote host.
For example a local host may be limited to IPv6 communication with a remote host because it only has an IPv6 stack implemented, it may have a dual stack implementation but only IPv6 network connectivity, or the remote host may have only IPv6 connectivity.

The connectivity of the host will be combined with three possibilities of application addressing capability.
- IPv4 only application: only IPv4 addressing capability.
- IPv6 only application: only IPv6 addressing capability.
- IPv4/IPv6 application: both IPv4 and IPv6 addressing capability.

There will be different behavior for BIA depending on the local host IPv4/IPv6 connectivity as well as the application's IPv4/IPv6 addressing capability.
- IPv4 applications communicating over IPv4 or IPv6 applications communicating over IPv6 are the native situations and do not need consideration here; in this case BIA simply has to perform no action.
- For an IPv4 application that needs to communicate using IPv6, the IPv4 application's addressing needs to be converted to IPv6 in order to be transmitted to the remote host. The opposite conversion has to be applied when an IPv6 application needs to communicate over IPv4.

- For an IPv6 only application on an IPv4/IPv6 host communicating with an IPv4/IPv6 remote host, the mechanism provides an optional feature to make this application able to communicate over IPv4 as well as over IPv6. If this application is trying to communicate over IPv4, the application's addressing needs to be converted to IPv4 in order to be transmitted to the remote host.

3.1 IPv6 Only Host Architecture Using BIA

IPv4 applications need IPv4 connectivity for communication. BIA is a mechanism that enables hosts that have to communicate using IPv6 to run IPv4 applications. Such hosts MUST have BIA installed and enabled. Figure 1 shows the architecture of the IPv6 host in which BIA is installed.

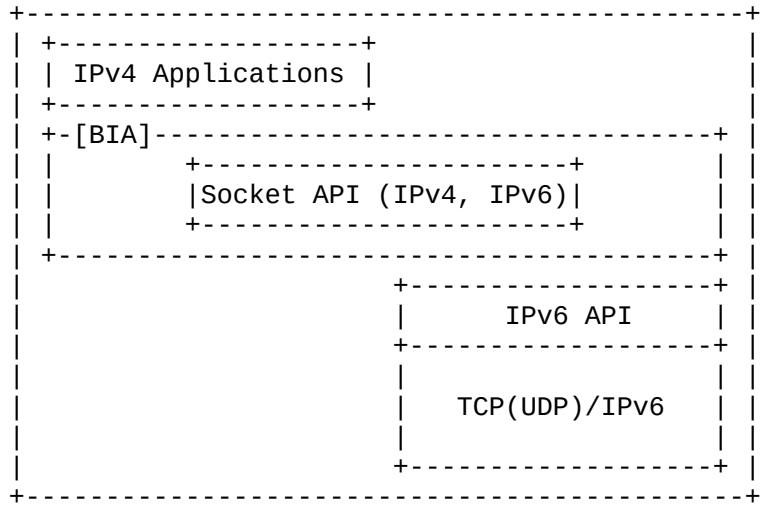

Figure 1: the architecture of IPv6 only host
In which BIA is installed.

3.2 IPv4 Only Host Architecture Using BIA

IPv4 only hosts are capable of running IPv4 applications only. BIA can be installed on such machines to allow these hosts to run IPv6 only applications as well. Figure 2 shows the host architecture of the IPv4 host in which BIA is installed.

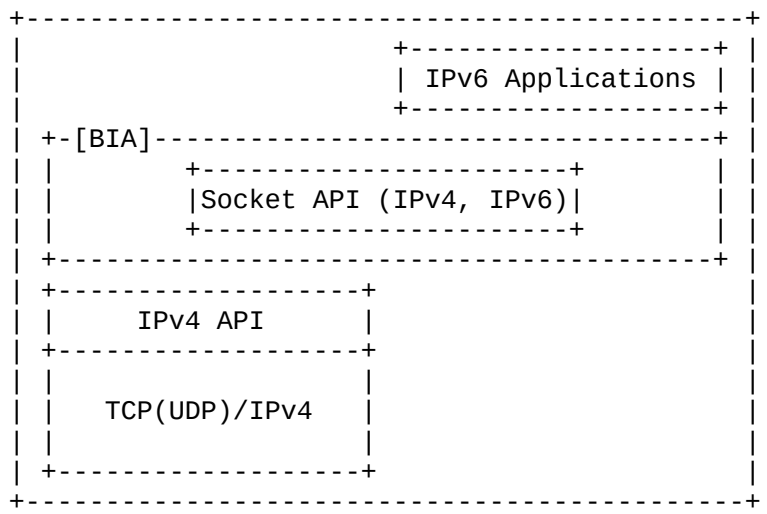

Figure 2: the architecture of IPv4 only host
In which BIA is installed.

3.3 Dual Connectivity Host Architecture Using BIA

[RFC4213] suggests that dual stack hosts need applications, dual TCP/IP modules and addresses for both IPv4 and IPv6. In such hosts, the BIA will be used only when the received DNS record(s) version is incompatible with the running of the application's IPv4/IPv6 capability. For example, if a dual connectivity host is running an IPv4 only application, and this application is going to communicate with an IPv6 only host, then the name resolver will receive the 'AAAA' record for the destination host so that the current connectivity will be IPv6, and BIA will translate the IPv4 socket API functions into IPv6 socket API functions and vice versa. BIA always will use the API functions that are compatible with the destination address. Figure 3 shows a dual connectivity host on which BIA is installed.

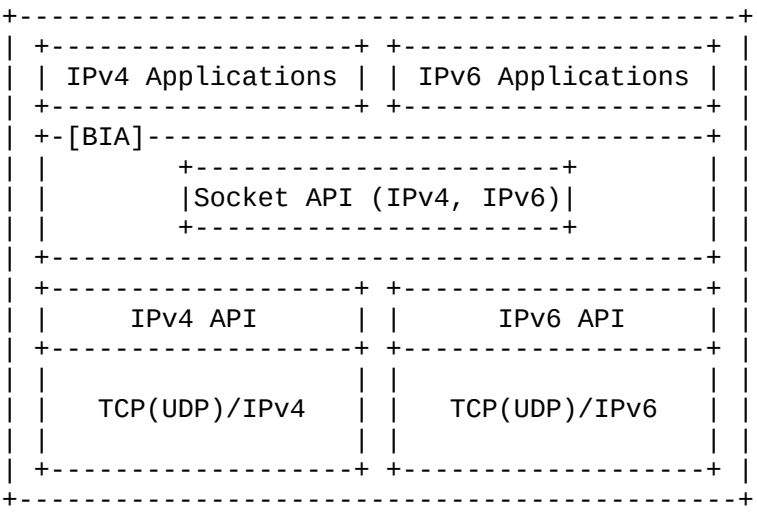

Figure 3: the architecture of dual stack host
In which BIA is installed.

4. BIA Modules

Like BIA, the API translator in BIA consists of three modules, name resolver, address resolver, and function mapper.

4.1 Name Resolver

In general the name resolver module returns a proper answer in response to the IPv4 or IPv6 application's DNS resolving request. The name resolver has different behaviors depending on the running application's IPv4/IPv6 capability and the IPv4/IPv6 connectivity.

4.1.1 IPv4 only application on the local host:

- IPv4 Only Host:
Since this is the native situation, BIA needs to perform no action.

- IPv6 Only Host
This behavior of the name resolver occurs when an IPv4 application needs to communicate using IPv6. The application will try to resolve names via the IPv4 resolver library (e.g. gethostbyname). BIA will call the IPv6 equivalent function (e.g. getnameinfo) that will resolve both 'A' and 'AAAA' records. If it got 'AAAA' record(s) only, it requests the address resolver (see below) to assign internal IPv4 address(es) corresponding to the IPv6 address(es) of the 'AAAA' record(s), then creates 'A' record(s) for the assigned IPv4 address(es), finally returns the created 'A' record(s) of the internal address(es) to the IPv4 application. Note that this behavior is similar to the name resolver behavior in the BIA mechanism, but

there are differences in the way internal addresses are assigned and managed (see address resolver further).
If both 'A' and 'AAAA' records are received, BIA will discard the 'A' record(s) as these cannot be used with IPv6, and select only the 'AAAA' record(s).

- IPv4/IPv6 Host:
The application will try to resolve names via the IPv4 resolver library (e.g. gethostbyname). BIA will call the IPv6 equivalent function (e.g. getnameinfo) that will resolve both 'A' and 'AAAA' records. If it got 'AAAA' record(s) only, it requests the address resolver (see below) to assign internal IPv4 address(es) corresponding to the IPv6 address(es) of the 'AAAA' record(s), then creates 'A' record(s) for the assigned IPv4 address(es), finally returns the created 'A' record(s) of the internal address(es) to the IPv4 application. If it got 'A' record(s) only, the communication will continue as native IPv4 communication, and BIA has to do no operation. If both 'A' and 'AAAA' records are returned, the communication can be effected natively using IPv4 using the 'A' record(s). But as an additional, optional feature, the IPv4 application can also be allowed to communicate over IPv6 with IPv6 peer(s). In that case, it requests the address resolver (see below) to assign internal IPv4 address(es) corresponding to the IPv6 address(es) of the 'AAAA' record(s), then creates 'A' record(s) for the assigned IPv4 address(es), and finally returns both the original 'A' for the remote AND the created 'A' record(s) of the internal address(es) to the IPv4 application. This extends the communication capabilities of the application to cover both IPv4 and IPv6 communication with the remote(s).

4.1.2 IPv6 Application on the local host:

- IPv6 Only Host:
Since this is the native situation, BIA needs to perform no action.

- IPv4 Only Host:
This situation occurs when an IPv6 application needs to communicate using IPv4, the application will try to resolve names via the IPv6 resolver library (e.g. getnameinfo). BIA will call the IPv4 equivalent function (e.g. gethostbyname). If it got 'A' record(s) only, it requests the address resolver to assign an internal IPv4-embedded IPv6 address(es) [I-D.draft-ietf-behave-address-format] corresponding to the IPv4 ddress(es), then creates 'AAAA' record(s) for the IPv4-embedded IPv6 address(es) and returns these 'AAAA' record(s) to the IPv6 application.

- IPv4/IPv6 Host:
The application will try to resolve names via the IPv6 resolver library (e.g. gethostinfo) that will resolve both 'A' and 'AAAA' records. If it

got 'A' record(s) only, the name resolver will request the address resolverto assign internal IPv4-embedded IPv6 address(es) corresponding to the IPv4 address(es), then creates 'AAAA' record(s) for the IPv4-embedded IPv6 address(es) and returns these 'AAAA' record(s) to the application. If it got 'AAAA' records only, the communication will continue as native IPv6 communication, and BIA has to do no operation. If both 'A' and 'AAAA' records are returned, the communication can be effected natively using IPv6 using the 'AAAA' record(s). But as an additional, optional feature, the IPv6 application can also be allowed to communicate over IPv4 with IPv4 peer(s). In that case, it requests the address resolver to assign internal IPv4-embedded IPv6 address(es), then creates 'AAAA' record(s), and finally returns both the original 'AAAA' for the remote AND the created 'AAAA' record(s) of the internal address(es) to the IPv6 application. This extends the communication capabilities of the application to cover both IPv4 and IPv6 communication with the remote(s).

4.1.3 Reverse DNS lookup

For various reasons, applications may do "pointer" lookups, i.e. the application passes the IP address and expects the host name in return. BIA should be able to handle these calls. When address translation (mapping or embedding) was performed on the host IP address, the application will call with the internal address generated by BIA. The DNS call to resolve the name should obviously be made with the external address that corresponds to the translated address, and the name returned for the external address needs to be returned to the application.

4.1.4 Originating without DNS lookup

Some applications bypass the DNS lookup, and use an IP address directly instead. While often this is bad practice, in some instances this how the software is being operated. For an IPv4 only application making such call, if an address mapping was stored for the supplied IPv4 address, that mapping can be used. Otherwise, as no DNS call is made, a correspondence between IPv4 and IPv6 addresses cannot be made by the name and address resolvers, and communication using incompatible application IPv4 capability and IPv6 connectivity is impossible. But for both for IPv4 and IPv6 applications, as a last resort, a "dirty trick" can be attempted however. Using the IP address from the application, a "pointer" DNS lookup can be made. If this succeeds, a forward DNS lookup can be made on the returned name, which may return one or more addresses of the other type required to establish the required IPv4/IPv6 address relationship. If this trick does not succeed, communication will be impossible, unless native communication is available (IPv4 over IPv4 connectivity or IPv6 over IPv6 connectivity).

It is recommended that address relationships can be manually entered in the mapping table for such occurrences. Such address mappings are in this case external IPv4-to-external IPv6 address mappings, and not relations between an internal and an external address.

4.2 Address Resolver

The address resolver is only involved with incompatibility between application IPv4/IPv6 capability and host IPv4/IPv6 connectivity. The address resolver has different behavior depending on the name resolver and function mapper requests. Like in the original BIA address mapper, the address resolver in BIA maintains a table of the pairs of an internal IPv4 address and an external IPv6 address in an IPv6 only host. These IPv4 addresses are assigned from an IPv4 address pool for internal addresses, but the mechanism for the pool is different here, as explained further.
The key difference between BIA and the BIA mechanism is the ability for the later to address all kinds of remote host connectivity (i.e. IPv4 only, IPv6 only and dual IPv4/IPv6 connectivity). Different addressing techniques are used depending on the remote host. The sending host has to take the decision either to map the destination IPv6 address into an internal IPv4 address assigned from the IPv4 address pool, or to embed the IPv4 address into an internal IPv4-embedded IPv6 address. The Address resolver in BIA can receive two possible address types-normally after calling the name resolver and querying the DNS- regarding the remote host(s) for that domain name:

- IPv6 Address: in this case it receives 'AAAA' record(s) and the address resolver has to map the IPv6 into an internal IPv4 address(es).
- IPv4 address: in this case it receives 'A' record(s) and the address resolver has to embed the IPv4 address into an internal IPv6 address(es).

4.2.1 Mapping

This technique is used when an IPv4 application needs to communicate using IPv6. It internally maintains a table of the pairs of IPv4 address(es) and IPv6 address(es). The IPv4 addresses are assigned from an IPv4 address pool. These addresses should be reserved from an unassigned class A domain reserved by IANA to be used by BIA for mapping purposes (see further). When the name resolver or the function mapper requests it to assign an internal IPv4 address corresponding to an IPv6 address, it selects and returns an IPv4 address out of the pool, and registers a new entry into the table dynamically. As in the original BIA, the registration occurs in the following two cases:

1. When the name resolver gets only an 'AAAA' record for the target host name and there is not a mapping entry for the IPv6 address.
2. 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. Address mappings are stored. When the address resolver is called to map an IPv6 external address into an IPv4 internal address, it will first look up the table to check whether there was a previous mapping for that address. If one is found, it will reuse and return that mapping. If not, it will create a new mapping, store that mapping and return the newly created mapping.

4.2.2 Embedding

Unlike the original BIA, BIA also allows IPv6 only applications to communicate over IPv4. Therefore a correspondence between external IPv4 addresses and internal IPv6 addresses need to be established. The proposed method is "IPv4-in-IPv6" address embedding [I-D.draft-ietf-behave-address-format]. The address resolver is configured to use one of the methodologies that are described in [I-D.draft-ietf-behave-address-format] to create an IPv4-embedded IPv6 address. The new address consists of: Network Specific Prefix (NSP) (32 bits), the IPv4 destination address (32 bits), and finally the suffix (64 bits). Figure 4 demonstrates the IPv4-embedded IPv6 address structure.
As the real external IPv4 address is embedded into the internal IPv6 address, no registering is required in this case, as there is always a unique correspondence.

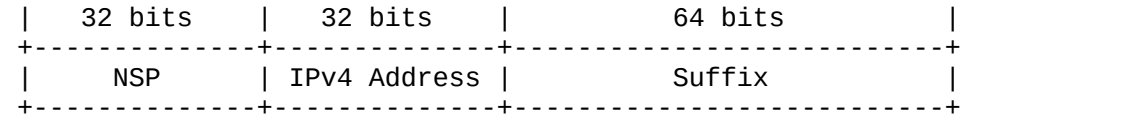

Figure 4: The structure of IPv4-embedded IPv6 address

4.3 Function Mapper

The function mapper has different behavior depending on the host connectivity. In dual connectivity hosts (IPv4 and IPv6 connectivity) the function mapper is used to decide which API functions to call in the current communication. It is important to note that the BIA modules will be invoked just if there is an incompatibility between the running application and the connectivity type. In case of IPv6 only or IPv4 only connectivity, the main goal of the function mapper is just like in original BIA mechanism. It is used when conversion is required from IPv4 to IPv6 or the other way around between application layer and communication stack. In that case it translates the API functions used by the application into the API functions needed for the communication, and vice versa.

In dual connectivity hosts, deciding which API functions to call depends on the address type of the remote. The following is the behavior of the function mapper running on dual connectivity hosts:

1. IPv4 only application communicating over IPv6: in this case it will call the equivalent IPv6 socket API functions. The application will use the IPv4 socket API to communicate with other hosts. Since the applicationneeds to communicate over IPv6, the function mapper intercepts the IPv4 socket API functions and calls the equivalent IPv6 socket API functions instead.

2. IPv6 only application communicating over IPv4: in this case it will call the equivalent IPv4 socket API functions. The application will use the IPv6 socket API to communicate with other hosts. Since the application needs to communicate over IPv4, the function mapper intercepts the IPv6 socket API functions and calls the equivalent IPv4 socket API functions instead.

5. Behavior Examples

The following sections will describe the behaviors of the hosts and applications that are listed in table 1.
In the following sections, the meanings of arrows are as follows:

- `--->` A DNS message for name resolving created by the Applications and the name resolver in the API translator.
- `+++>` An IPv4 or IPv4-embedded IPv6 address request to and reply from the address resolver for the name resolver and the function mapper.
- `===>` Data flow by API functions created by the applications and the function mapper in the API translator.

5.1 IPv4 Only Application, IPv6 Only Connectivity with an IPv6 Only Peer.

5.1.1 Behavior for IPv4 Only Originator Application on IPv6 Only Host Communicating to IPv6 Only Peer

When an IPv4 application sends a DNS query to its name server, the name resolver intercepts the query and then creates a new query to resolve both 'A' and 'AAAA' records. When only 'AAAA' record(s) is (are) resolved, the name resolver requests the address resolver to get IPv4 address(es) corresponding to the IPv6 address(es) for each IPv6 address from the 'AAAA' record. The address resolver first looks up the table of stored entries to check if the correspondence was made previously. If yes,the stored mapping is retrieved and passed to the name resolver. If not, the address resolver creates a new mapping for an internal IPv4 address corresponding to the IPv6 external address, stores the mapping, and returns the mapping to the name resolver. The name resolver, upon receiving the internal IPv4 address(es) creates 'A' record(s) for the

assigned IPv4 address(es) and returns these to the application. In order for the IPv4 application to send IPv4 packets over IPv6, it calls the IPv4 socket API function. The function mapper detects the API function call from the application. The IPv6 address is required to invoke the IPv6 socket API function, thus the function mapper requests the IPv6 address corresponding for the internal IPv4 address to the address resolver. The address resolver selects the external destination IPv6 address corresponding to the internal IPv4 address from the mapping table and returns it to the function mapper. Using this IPv6 address, the function mapper will invoke the IPv6 socket API function corresponding to the IPv4 socket API function received from the application.
When a reply is received, this will come in through the IPv6 socket API, and the function mapper requests the address resolver for the IPv4 address corresponding to the received IPv6 address. This IPv4 address will be used to translate the IPv6 socket API function call into the corresponding IPv4 socket API function call for the IPv4 application. Figure 5 illustrates the behavior described above.

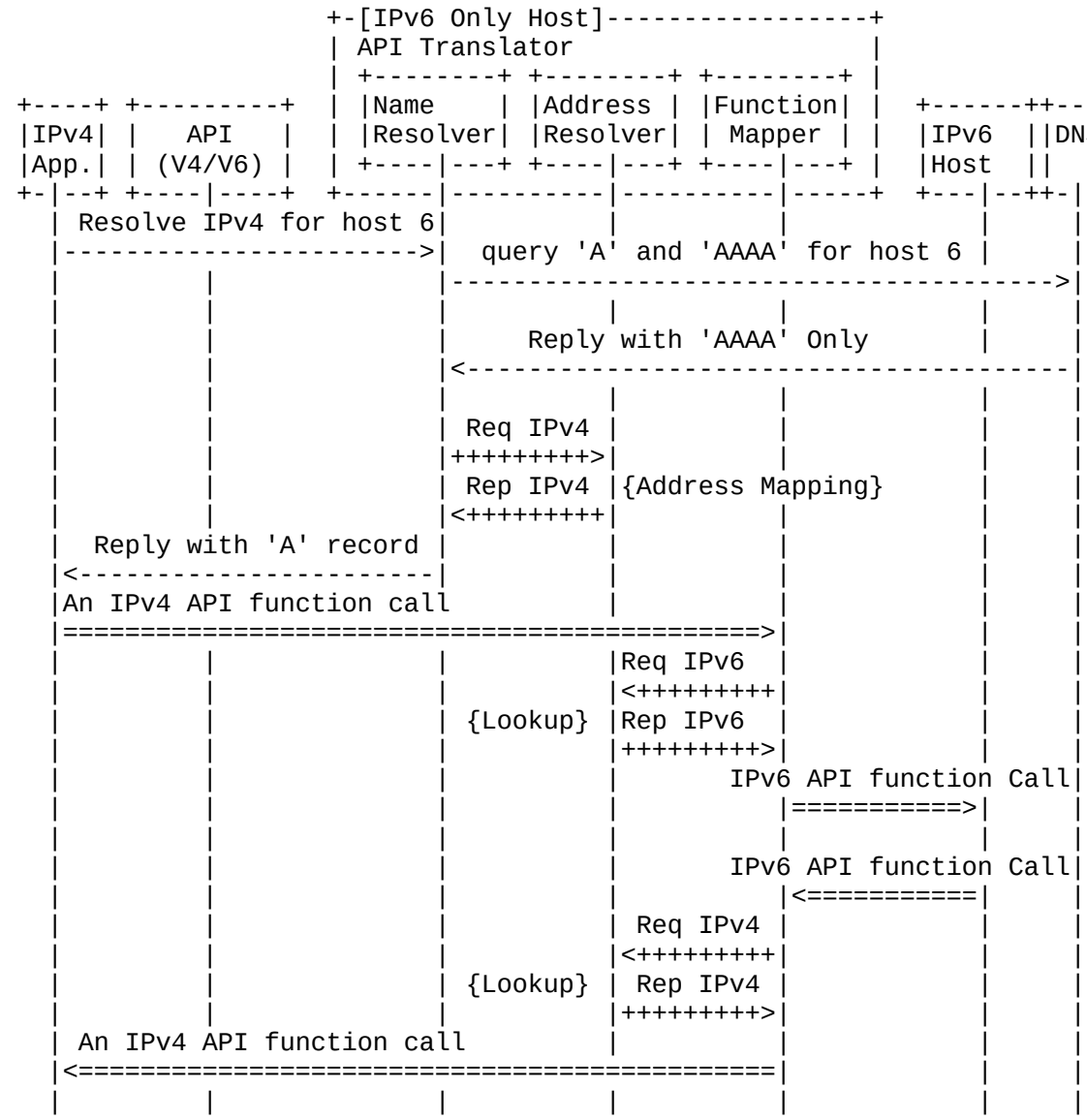

Figure 5: The behavior of the originator communicates with IPv6 Application

5.1.2 Behavior for IPv4 Only Recipient Application on IPv6 Only Host

The IPv6 originator host that started the communication to this host has resolved the address of this IPv6 host with 'AAAA' record(s) through its name server, and has sent an IPv6 packet to this IPv6 host. The function mapper requests the internal IPv4 address

corresponding to the originator's IPv6 address. The address resolver looks up the mapping table to check for an entry. If one is found, it returns the internal IPv4 address corresponding to the IPv6 address. Then the function mapper invokes the corresponding IPv4 socket API function for the IPv4 application corresponding to the IPv6 function. If not, the address resolver creates a new mapping for an internal IPv4 address corresponding to the IPv6 external address, stores the mapping, and returns the mapping to the function resolver. The remaining part of the handling is identical to what was described in 5.1.1. Figure 6 illustrates the behavior described above.

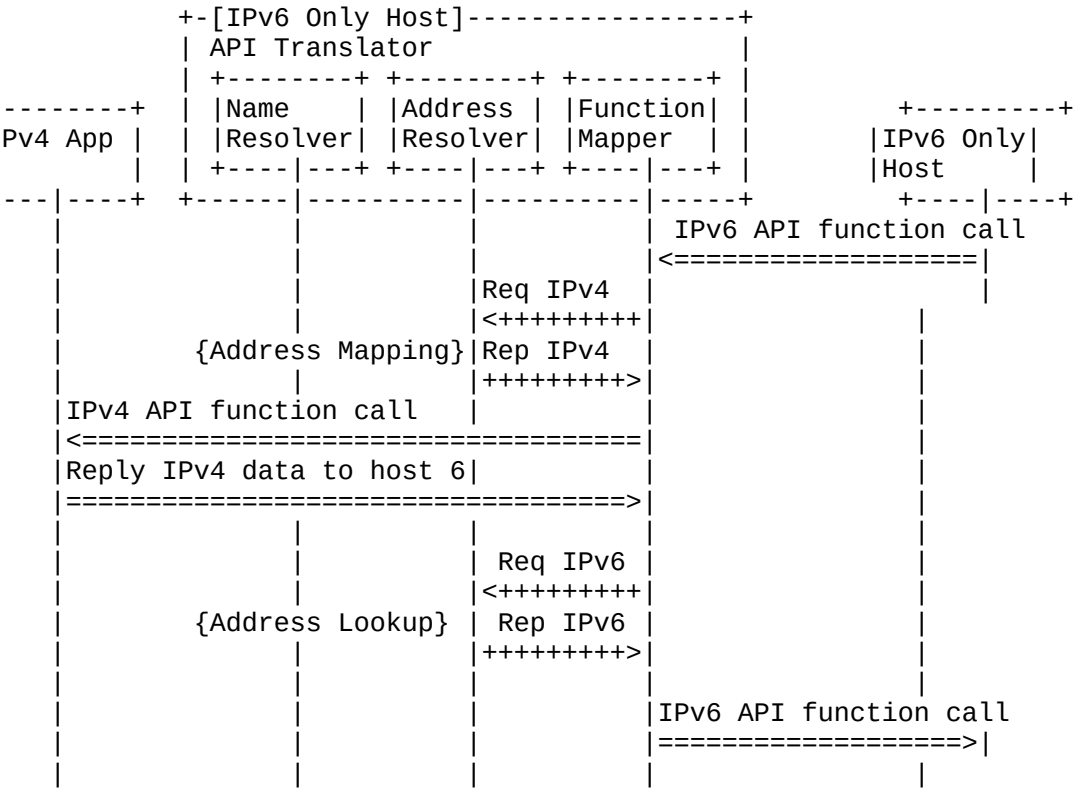

Figure 6: Behavior of receiving data from IPv6 host

5.2 IPv4 Only Application, IPv6 Only Network and Dual Connectivity Peer

5.2.1 Behavior for IPv4 Only Originator Application on IPv6 Only Host Communicating with Dual Connectivity Host

When an IPv4 application sends a DNS query to its name server, the name resolver intercepts the query and then creates a new query to resolve

both 'A' and 'AAAA' records. If both 'A' and 'AAAA' records are resolved, the name resolver will only select the 'AAAA' records and drop the drop the 'A' record(s) and requests the address resolver to assign internal IPv4 address(es) corresponding to the IPv6 address(es). The remaining behavior is exactly like described in 5.1.1.

5.2.2 Behavior for IPv4 Only Recipient Application on IPv6 Only Host Communicating with Dual Connectivity Host

Exactly the same as in section 5.1.2

5.3 IPv6 Only Application, IPv4 Only Connectivity with an IPv4 Only Peer

5.3.1 Behavior for IPv6 Only Originator Application on IPv4 Only Host Communicating with IPv4 Only Host

When an IPv6 application sends a DNS query to its name server to resolve both 'A' and 'AAAA' records, the name resolver intercepts the query and then creates a new query to resolve only 'A' record(s), since it is a IPv4 only host. With only 'A' record(s) resolved, the name resolver requests the address resolver to embed the IPv4 address(es) into IPv6 address(es) using the format described in section 4.2.2. The name resolver creates 'AAAA' record(s) for the IPv4 embedded IPv6 address(es) and returns it to the application. In order for the IPv6 application to send IPv6 packets to IPv4 only host, it calls the IPv6 socket API function. The function mapper detects the API function call from the application. The function mapper requires an IPv4 address to invoke the IPv4 socket API function, so it requests the corresponding IPv4 address to the address resolver. The address resolver extracts the destination IPv4 address from the IPv4-embedded IPv6 address and returns it to the function mapper. Using this IPv4 address, the function mapper will invoke the IPv4 socket API function corresponding to the IPv6 socket API function. We notice here the address resolver is not going to save any new records to the mapping table.
When a reply is received, this will come in through the IPv4 socket API, and the function mapper requests the address resolver for the IPv6 address corresponding to the received IPv4 address. This IPv6 address will be used to translate the IPv4 socket API function call into the corresponding IPv6 socket API function call for the IPv6 application. Figure 7 illustrates the behavior described above.

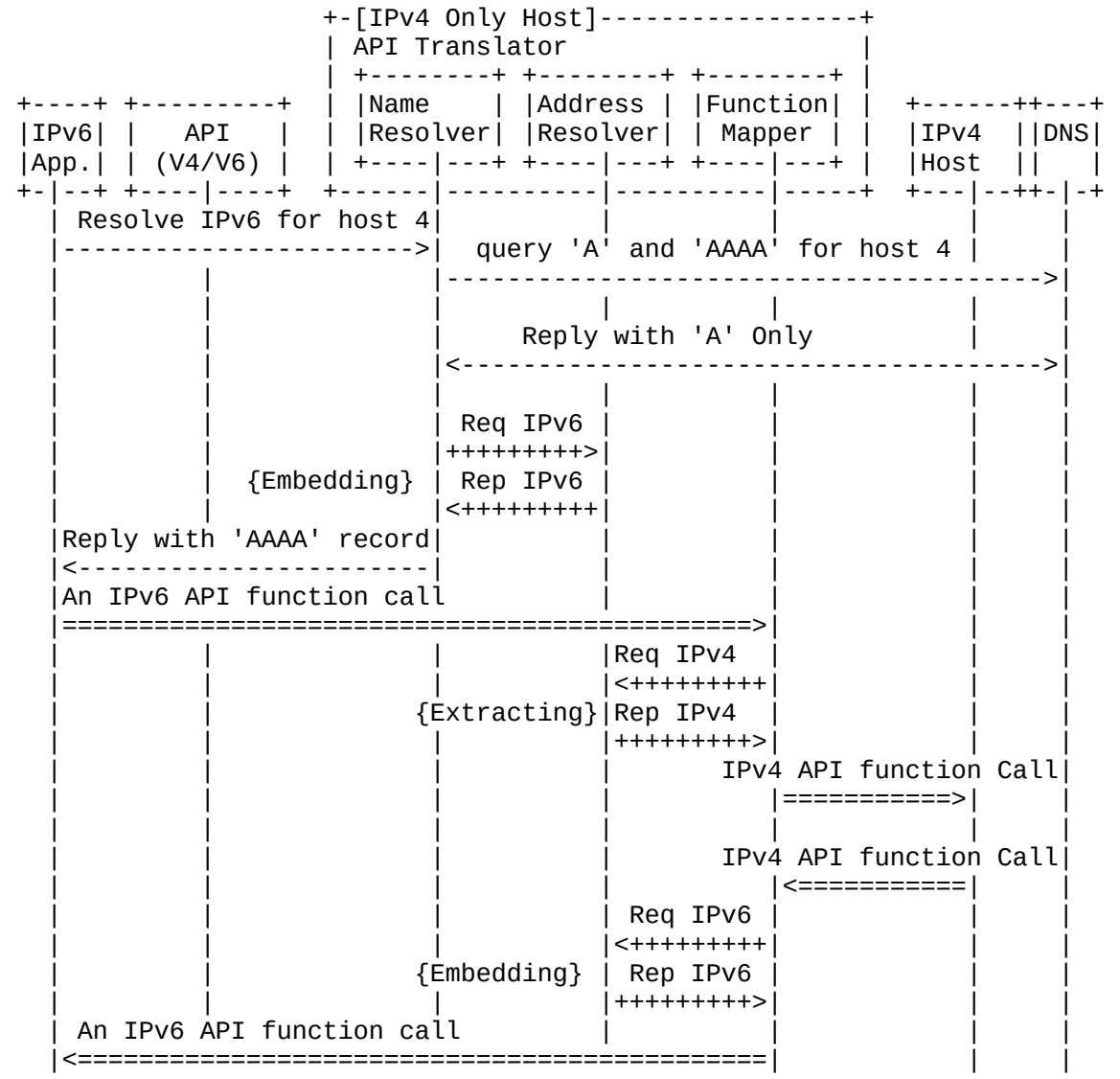

Figure 7: The behavior of the originator communicates with IPv4 Application

5.3.2 Behavior for IPv6 Only Recipient Application on IPv4 Only Host Communicating with IPv4 Only Host

The IPv4 originator host has resolved the address of this IPv4 host with 'A'records through its name server, and has sent an IPv4 packet to this IPv4 host. The function mapper requests the IPv6 address to the address resolver in order to invoke the IPv6 socket API function to communicate with the IPv6 application. The address resolver embeds

the IPv4 address(es) into IPv6 address(es) using the format described in section 4.2.2, and returns this address. Then the function mapper invokes the corresponding IPv6 socket API function for the IPv6 application corresponding to the IPv4 function.

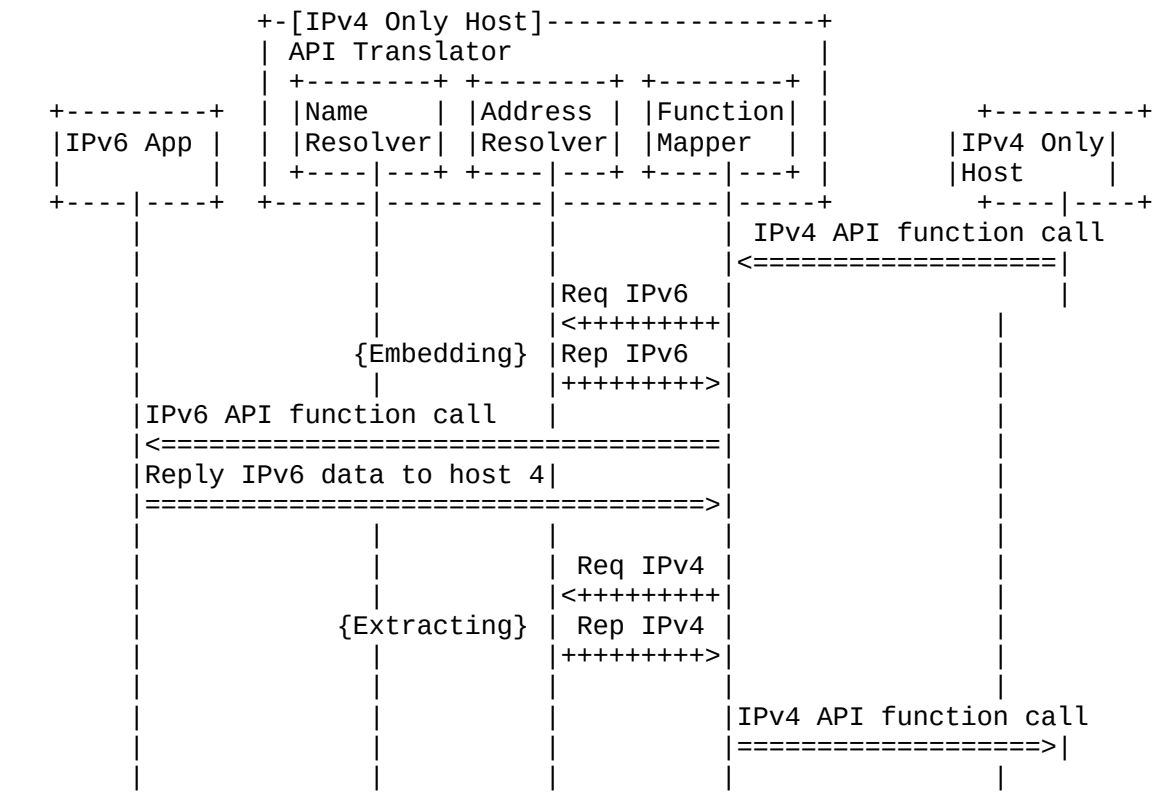

Figure 8: Behavior of receiving data from IPv4 host

5.4 IPv6 Only Application, IPv4 Only Network and Dual Connectivity Peer

5.4.1 Behavior for IPv6 Only Originator Application on IPv4 Only Host Communicating with Dual Connectivity Host

When an IPv6 application sends a DNS query to its name server, the name resolver intercepts the query and then creates a new query to resolve 'A' record(s); no 'AAAA' record(s) are returned, as it is an IPv4 only host. When 'A' record(s) is/are resolved, the name resolver will request the address resolver to embed the IPv4 address(es) into IPv6 address(es) using the format that is describes in section 4.2.2. The remaining processing is as in 5.3.1.

5.4.2 Behavior for IPv6 Only Recipient Application on IPv4 Only Host Communicating with Dual Connectivity Host

Exactly the same as in section 5.3.2

5.5 IPv4 Only Application on Dual Connectivity Host Communicating to IPv6 Only Peer

5.5.1 IPv4 Only Originator Application on Dual Connectivity Host Communicating to IPv6 Only Peer

Exactly the same as in section 5.1.1

5.5.2 IPv4 Only Recipient Application on Dual Connectivity Host Communicating to IPv6 Only Peer

Exactly the same as in section 5.1.2

5.6 IPv6 Only Application on Dual Connectivity Host Communicating to IPv4 Only Peer

5.6.1 IPv6 Only Originator Application on Dual Connectivity Host Communicating to IPv4 Only Peer

Exactly the same as in section 5.3.1

5.6.2 IPv4 Only Recipient Application on Dual Connectivity Host Communicating to IPv4 Only Peer

Exactly the same as in section 5.3.2

6. Considerations

6.1 Socket API Conversion

IPv4 socket API functions are translated into semantically the same IPv6 socket API functions and vice versa. See Appendix A for the API list intercepted by BIA. IP addresses embedded in application layer protocols (e.g., FTP) can be translated in API functions. Its implementation depends on operating systems.

NOTE: Basically, IPv4 socket API functions are not fully compatible with IPv6 since the IPv6 has new advanced features.

6.2 Address Mapping and Embedding

There are some considerations as to the choice and management of the internal addresses:
- For a diversity of reasons, several applications store the addresses of machines they have been communicating with, and check these

addresses on next contact. It is hence important that each mapping of external IPv6 to internal IPv4 addresses is being stored by BIA, so as to alwaysuse the same internal address for a particular external address at the next communication. This implies a very wide IPv4 address range available for mapping. The authors propose a Class A range, making approximately 16Mega addresses available. Even that can get exhausted in some cases, so this range should be supplemented by a round-robin scheme where,in case of exhaustion, the mappings that remain unused for the longest time can be reused for new mappings (not the creation time, but the last time that mapping was actively used is important). It is considered that this would be sufficient in about all operational situations. As this mapping list potentially can become very large, the store/retrieval mechanism implementation should be optimized for speed or it may introduce unacceptable long delays. This is an implementation issue however, and will not be dealt with here.
- It is obvious that an IPv4/IPv6 correspondence while be frequently required in the course of a communication; short time caching seems essential to avoid having to look up the correspondence again during the course of a communication.
- A machine having both IPv4 and IPv6 connectivity will be using both IPv4 and IPv6 addresses. To avoid conflicts, it is essential that the internal addresses used will never be used as an external address. Original BIA proposed the use of a limited (256 addresses) range as a "pool" in an "unassigned" IPv4 address range. The limited size (256) is much too small for operational purposes, even not considering the requirement for storing the mappings as described in the previous paragraph, as a machine may have more than this number of mappings active concurrently. Taking into account the requirement for storing the mappings, a very large range of unassigned addresses is required. Please refer to section 8 of this document.

6.3 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 [RFC2765], and vice versa. It can be implemented using raw socket.

6.4 Implementation Issues

Some operating systems support the preload library functions, so it is easy to implement the API translator by using it. For example, the user can replace all existing socket API functions with user-defined socket API functions which translate the socket API function.
In this case, every IPv4 application has its own translation library using a preloaded library which will be bound into the application before executing it dynamically.

Some other operating systems support the user-defined layered protocol allowing a user to develop some additional protocols and put them in the existing protocol stack. In this case, the API translator can be implemented as a layered protocol module.

In the above two approaches, it is assumed that there exists both TCP(UDP)/IPv4 and TCP(UDP)/IPv6 stacks and there is no need to modify or to add a new TCP-UDP/IPv6 stack.

7. Limitations

This mechanism supports unicast communications only. In order to support multicast functions, some other additional functionalities must be considered in the function mapper module.
Since the IPv6 socket API has new advanced features, it is difficult to translate such kinds of IPv6 socket APIs into IPv4 socket APIs. Thus, IPv6 inbound communication with advanced features may be discarded.

It should be noted that the original BIA assumes the hosts have compatible network connectivity. The new version of the BIA is developed to support the heterogeneity between connectivity and applications only, NOT incompatible network connectivity. Communication between hosts with incompatible connectivity (IPv4 only connectivity to IPv6 only connectivity, or the other way around) cannot be handled by BIA, and other solutions need to be applied, e.g. protocol translation mechanisms PNAT [I-D.draft-huang-behave-pnat], NAT64 [I-D.ietf-behave-v6v4-xlate-stateful], NAT-PT-HIST[RFC4966], or [I-D.draft-ietf-behave-v6v4-framework].

8. IANA Considerations

The authors propose that IANA reserves one of the few remaining reserved IPv4 Class A ranges specifically to be used in the internal mapping, and making sure this range will never be used for external addressing. While giving up one of the precious last remaining IPv4 Class A ranges for this purpose seems a big demand, the authors feel that unblocking one of the main obstacles in IPv6 deployment warrants this.
Similarly, a NSP for the embedding of IPv4 in internal IPv6 addresses should be reserved by IANA for BIA use, in order to avoid conflicts with other types of embedded IPv6 addresses being used as external addresses. This assignment should not be a big problem however.

9. Security Considerations

The security consideration of BIA mostly relies on that of NAT-PT-HIST [RFC4966]. 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 isn't a DNS ALG as in NAT-PT-HIST, so there is no interference with DNSSEC.

The use of address pooling may open a denial of service attack vulnerability. So BIA should employ the same sort of protection techniques as NAT-PT-HIST [RFC4966] does.

10. Normative References

[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.

[RFC2460] Deering, S., and R., Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC3022] Srisuresh, P. and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)", RFC 3022, January 2001.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4213] Nordmark, E. and R. Gilligan, "Basic Transition Mechanisms for IPv6 Hosts and Routers", RFC 4213, October 2005.

[RFC2767] Tsuchiya, K., HIGUCHI, H., and Y. Atarashi, "Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS)", RFC 2767, February 2000.

[I-D.draft-huang-behave-rfc3338bis] Huang, B., Deng, H., and T. Savolainen, "Dual Stack Hosts Using "Bump-in-the-API" (BIA)", draft-huang-behave-rfc3338bis-02 (work in progress), March 2010.

[RFC2765] Nordmark, E., "Stateless IP/ICMP Translation Algorithm (SIIT)", RFC 2765, February 2000.

[I-D.draft-ietf-behave-address-format] Huitema, C., "IPv6 Addressing of IPv4/IPv6 Translators", draft-ietf-behave-address-format-10 (work in progress), August 2010.

[I-D.draft-huang-behave-pnat] Huang, B., and H., Deng, "Prefix NAT: Host based IPv6 translation", draft-huang-behave-pnat-01 (work in progress), February 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.

[RFC4966] Aoun, C. and E. Davies, "Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status", RFC 4966, July 2007.

[I-D.draft-ietf-behave-v6v4-framework] Baker, F., Li, X., Bao, C., and K. Yin, "Framework for IPv4/IPv6 Translation", draft-ietf-behave-v6v4-framework-10(work in progress), August 17, 2010.

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

Appendix A : API list intercepted by BIA

The following functions are the API list which SHOULD be intercepted by BIA module.

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 [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

BIA MAY intercept inet_ntoa() and inet_addr() and use the address mapper for those. Doing that enables BIA 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

Ala Hamarsheh
Electronics and Informatics Department ETRO/Vrije Universiteit Brussel
Pleinlaan 2, 1050 Elsene, Brussels, Belgium
Tel: +32 2 629 2930
Fax: +32 2 629 2883
Email: ala.hamarsheh@vub.ac.be

Prof. Marnix Goossens
Electronics and Informatics Department ETRO/Vrije Universiteit Brussel
Pleinlaan 2, 1050 Elsene, Brussels, Belgium
Tel: +32 2 629 2987
Fax: +32 2 629 2883
Email: marnix.goossens@vub.ac.be