Interoperabilidad de aplicaciones IPv4 e IPv6

Eva M. Castro, Jesús González, Gregorio Robles, Tomás de Miguel[*]
Grupo de Sistemas y Comunicaciones, Dpto. Informática, Estadística y Telemática
Universidad Rey Juan Carlos de Madrid, C/ Tulipán s/n, 28933 Móstoles (Madrid)
[*] Dpto. de Ingeniería y Sistemas Telemáticos
Universidad Politécnica de Madrid, Avda. Complutense s/n, 28040 Madrid

Abstract

Los problemas aparecidos con la versión del protocolo estándar de Internet, IPv4, –escaso espacio de direccionamiento y gestión del encaminamiento- han provocado el desarrollo de una nueva versión, IPv6. La transición a la nueva versión del protocolo está siendo un proceso gradual en el que tienen que convivir ambos tipos de redes y aplicaciones. Actualmente, millones de aplicaciones IPv4 están repartidas por toda Internet y no pueden comunicarse directamente con aplicaciones desarrolladas para IPv6. En este artículo se presentarán los problemas de interoperabilidad de aplicaciones IPv4 e IPv6 y se estudiarán soluciones aplicables en los nodos extremos de la comunicación, donde normalmente tenemos acceso a la configuración de la máquina/nodo. Como caso de estudio se analizará la implantación de un mecanismo de transición que hemos desarrollado basándonos en una propuesta experimental de Internet Engineering Task Force (IETF).

1. Introducción

Durante los primeros 1990 se hicieron obvios varios problemas de la implementación de IP desplegada, IPv4 -como el escaso espacio de direccionamiento y la gestión del encaminamiento. Con la intención de resolverlos se desarrollaron algunos mecanismos específicos, como por ejemplo Network Address Translation (NAT) [1], que es en la actualidad uno de los más extendidos. Sin embargo, NAT elimina el servicio de comunicación entre entidades finales y rompe el modelo de comunicación extremo a extremo inicialmente planteado en IP. La demanda de comunicación entre sistemas finales de las nuevas aplicaciones Peer-To-Peer (P2P) y la necesidad de calidad de servicio en las comunicaciones, han impulsado el desarrollo de IPv6 [2].

Actualmente millones de máquinas están conectadas a Internet usando IPv4 y es imposible realizar el cambio a IPv6 de forma inmediata. La transición se está realizando de forma gradual [3], primero portando la infraestructura de red, junto con sus servicios básicos y en segundo lugar, las aplicaciones.

Una de las principales preocupaciones durante el período de transición es la interoperabilidad entre las aplicaciones ya existentes y las nuevas desarrolladas para IPv6. En la mayoría de los casos la interoperabilidad de aplicaciones se consigue utilizando nodos que implementen ambos protocolos, es decir, nodos duales que puedan comunicarse utilizando tanto IPv4 como IPv6. Los nodos con pila dual proporcionan un mecanismo para permitir a las aplicaciones IPv4 comunicarse con aplicaciones IPv6 utilizando el protocolo IPv4 para el intercambio de paquetes entre los nodos finales. Sin embargo, en los escenarios en los que la conectividad entre los nodos origen y destino sólo es posible utilizando IPv6, la pila dual no será suficiente para establecer la comunicación y se necesitará algún mecanismo de transición adicional.

El objetivo de este trabajo es permitir la interoperabilidad de aplicaciones IPv4 e IPv6 cuando sólo es posible utilizar una infraestructura de red IPv6, sin cambiar el código fuente de las mismas. En muchos casos no se dispone del código fuente, o existe un problema de licencias que impide modificarlo. Además, existen casos en los que, aunque se dispone del código, portarlo a la nueva interfaz de IPv6, distribuirlo e instalarlo puede resultar un trabajo costoso que requiera demasiado tiempo. Por estos motivos, el caso de estudio de este trabajo es de especial relevancia durante el período de transición, permitiendo convivencia de aplicaciones, o partes de una aplicación, ya portadas a IPv6 con aplicaciones IPv4 existentes.

En el presente artículo se proporcionará una visión general de los mecanismos de transición a IPv6 en el ámbito de nodo. En la sección 2 se analizarán los mecanismos experimentales propuestos por el IETF, Bump In the Stack (BIS) [4] y Bump In the API (BIA) [5]. En la sección 3, 4 y 5 se estudiará la implantación de un mecanismo que hemos desarrollado basándonos en BIA, considerando sus ventajas y sus limitaciones. Y, finalmente, en la sección 6 se expondrán las conclusiones y los posibles trabajos futuros.

2. Interoperabilidad de aplicaciones IPv4 IPv6

La mayoría de los mecanismos de transición de IPv6 [3] se han desarrollado para conectar redes IPv4 con redes IPv6 y basan su funcionamiento en entidades situadas en un punto intermedio de la red que realizan algún tipo de procesamiento, por ejemplo túneles o traducción de protocolos, para permitir su comunicación. El objetivo de este trabajo no es estudiar mecanismos de transición para la interoperabilidad de redes, sino la comunicación

entre aplicaciones heterogéneas IPv4 e IPv6 sobre una red que permite únicamente conectividad IPv6, sin modificar ningún elemento intermedio de la red, donde muchas veces no tenemos la posibilidad de cambiar la configuración.

Partiendo del escenario caso de estudio en el que existen aplicaciones IPv4 que necesitan comunicarse con aplicaciones IPv6 sobre una red IPv6, lo más sencillo es introducir una traducción IPv4/IPv6 en el nodo donde se ejecuta la aplicación IPv4. Parece requisito indispensable que el nodo que realiza la traducción tenga instalada la doble pila, IPv4 para permitir que la aplicación se ejecute correctamente e IPv6 para establecer las comunicaciones.

Es posible utilizar un mecanismo basado en túneles para permitir la ejecución de aplicaciones IPv4 en redes IPv6. El túnel se establecería entre dos puntos, el nodo donde se ejecuta la aplicación IPv4 y otro nodo cualquiera de la red. Nótese que el hecho de utilizar túneles implica que las aplicaciones que están comunicándose deben haber sido desarrolladas para la misma versión del protocolo, ya que en el extremo final del túnel se proporciona el paquete tal y como fue emitido. Por tanto, si la aplicación remota es IPv6, sería necesario otro mecanismo de transición adicional antes de poder entregar el paquete original IPv4 en destino.

El IETF ha definido 2 métodos experimentales de transición para realizar la traducción IPv4/IPv6 en un nodo que ejecuta una aplicación IPv4, dependiendo del nivel en el se produzca la traducción: en la interfaz de programación de aplicaciones (Bump In the API, BIA) y en la pila IP (Bump In the Stack, BIS).

El mecanismo BIS intercepta los paquetes IP al salir del nivel IP de la máquina, antes de enviarlos a la tarjeta de red, y realiza la traducción entre IPv4 e IPv6 según los paquetes sean entrantes o salientes.

BIA sigue un funcionamiento análogo a BIS, salvo que la traducción de paquetes se realiza antes de construir el paquete, en la propia interfaz de programación de aplicaciones. Por tanto, BIA es un mecanismo de transición más ligero ya que no necesita realizar una traducción del paquete, sino que construye el paquete IPv6 a partir de las funciones de la API IPv4, véase Fig.1.

Aunque BIA ha sido diseñado con el objetivo específico de permitir a las aplicaciones IPv4 funcionar sobre redes IPv6, este mecanismo no impone limitaciones para utilizarse conjuntamente con otros mecanismos de transición y así ofrecer la máxima interoperabilidad entre redes y aplicaciones heterogéneas.

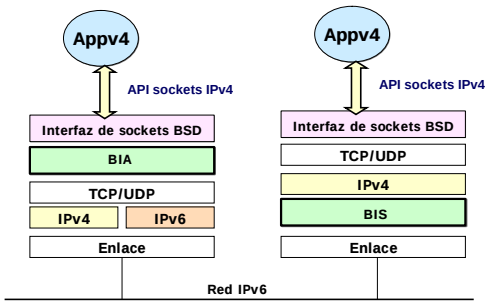


Fig 1. BIS y BIA, mecanismos de transición a IPv6 en el ámbito de un nodo.

3. Caso de estudio: Implementación de BIA

El mecanismo BIA permite que las aplicaciones IPv4 sigan funcionando normalmente sobre una red IPv6, sin que haya que modificar el código fuente, ni recompilarlo. Dado que una aplicación IPv4 sólo es capaz de manejar direcciones IPv4, el mecanismo BIA proporcionará direcciones ficticias IPv4 a la aplicación para que sea transparente el hecho de que los nodos remotos son sólo direccionables mediante IPv6. BIA se encarga de mantener la correspondencia entre las direcciones IPv4 ficticias y las direcciones IPv6 reales.

A continuación se describe la arquitectura e implementación que hemos realizado del mecanismo BIA, así como su uso en aplicaciones cliente/servidor.

3.1. Arquitectura

La arquitectura del mecanismo BIA que hemos desarrollado se divide en una serie de módulos encargados de realizar las siguientes funciones: extensión de la resolución de nombres, traducción de las funciones de la interfaz y traducción del mapa de direcciones.

El módulo de extensión de la resolución de nombres permite que una aplicación IPv4 reciba como respuesta a una consulta de resolución de un nombre de una máquina IPv6, una dirección ficticia IPv4.

El módulo traductor de funciones de la interfaz es el encargado de realizar la traducción entre la API de IPv4 e IPv6. Permite a la aplicación seguir usando la API de IPv4, aun cuando la aplicación acabe construyendo, enviando y recibiendo paquetes IPv6.

El módulo traductor del mapa de direcciones gestiona la asociación entre las direcciones IPv4 ficticias y las direcciones IPv6 reales.

3.2. Implementación

La implementación de BIA que hemos desarrollado se basa en una de las APIs más utilizadas en el mundo Internet, la interfaz de sockets BSD, y está construida para sistemas tipo Unix (probada concretamente sobre el kernel de Linux). Dicha implementación es una librería dinámica escrita en C, *libbia*, que mantiene la misma interfaz que la API de sockets y realiza la traducción IPv4/IPv6, si ésta es necesaria, para comunicar 2 aplicaciones heterogéneas. Haciendo uso de la variable *LD_PRELOAD*, la librería *libbia* se carga dinámicamente, interponiéndose entre la aplicación y la librería estándar del sistema, *libc*, suplantando las funciones de la API de sockets BSD de *libc*. Véase la Fig. 2.

Actualmente existe una implementación del mecanismo para el sistema operativo Windows 2000 [6] utilizando la interfaz de proveedores de servicio (Service Provider Interface, SPI).

En el módulo de extensión de la resolución de nombres, *libbia* realiza una nueva implementación de las dos funciones básicas la API estándar, *gethostbyname* y *gethostbyaddr*, invocando las funciones equivalentes de la nueva API: *getaddrinfo* y *getnodeinfo*. Si el resultado de una resolución son sólo direcciones IPv6, *libbia* devolverá como respuesta una dirección ficticia IPv4, válida para la aplicación, almacenando la información de dicha asociación en el módulo traductor del mapa de direcciones. Es posible que se solicite repetidas veces la resolución de un mismo nombre de nodo, por este motivo, la librería *libbia* realizará una consulta previa al módulo traductor del mapa de direcciones para comprobar si dicho nombre ya tiene asignada una dirección IPv4 ficticia, agilizando el proceso de resolución de nombres en BIA.

El módulo traductor de las funciones de la interfaz implementa todas las funciones de la API de sockets, realizando las llamadas correspondientes de la API de IPv6. Aquellas llamadas que requieran una estructura de dirección de socket no podrán tener una traducción directa, necesitarán conocer la asociación entre direcciones ficticias y reales.

El módulo traductor del mapa de direcciones asocia de forma biunívoca direcciones ficticias IPv4 y nombres de nodos IPv6. Las aplicaciones IPv4 utilizarán las direcciones IPv4 ficticias y *libbia* las convertirá a nombres de nodos IPv6 a través del módulo traductor del mapa de direcciones. Sin embargo, las funciones de la API de sockets no utilizan directamente los nombres de nodos para establecer las comunicaciones. Dichos nombres tienen que ser resueltos finalmente a sus correspondientes direcciones IP, en nuestro caso direcciones IPv6 dentro de *libbia*.

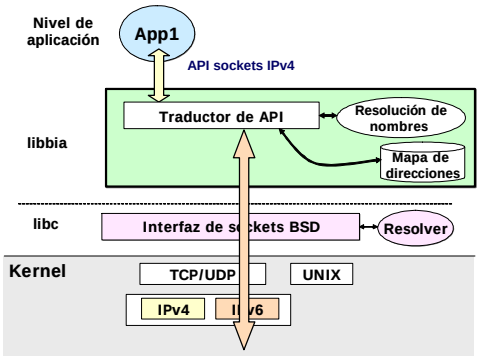
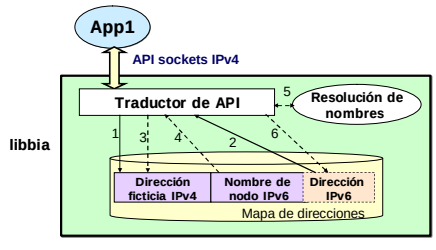


Fig. 2. Implementación de BIA.

Si sólo se almacenan la pareja de valores dirección ficticia IPv4 y nombre de nodo IPv6, por cada operación que requiera una dirección IP se necesitaría realizar una operación de resolución de nombre, provocando retardos innecesarios. Por este motivo, el módulo traductor del mapa de direcciones almacena junto al nombre del nodo IPv6 una de las posibles direcciones pertenecientes a dicho nodo, como caché, para acelerar el proceso de resolución.

La librería *libbia* utilizará la dirección IPv6 almacenada en el mapa de direcciones como una dirección candidata del nodo IPv6. Sin embargo, dada la naturaleza del sistema de direccionamiento IPv6, pensado para permitir la autoconfiguración y la asignación dinámica de direcciones, la dirección IPv6 almacenada puede dejar de ser válida en cualquier momento. Por tanto, si se produce algún error al utilizar dicha dirección, *libbia* solicitará de nuevo la resolución del nombre de nodo a una dirección IPv6 válida, actualizando la información en el mapa de direcciones, véase Fig. 3.

Es importante mantener como información permanente dentro del mapa de direcciones el nombre del nodo IPv6 ya que las direcciones pueden variar durante la comunicación, mientras que los nombres de los nodos normalmente permanecen invariables.



1. Petición dirección IPv6
2. Respuesta dirección IPv6

Si existe algún error utilizando dicha dirección:

3. Petición de nombre de nodo IPv6
4. Respuesta de nombre de nodo IPv6
5. Resolución de nombre de nodo IPv6
6. Modificación de la dirección IPv6

Fig. 3. Mapa de direcciones BIA.

Cada entrada almacenada en el mapa de direcciones debe tener un tiempo de vida máximo para poder liberar asociaciones antiguas. El tiempo de vida de una entrada se renueva cada vez que se accede a dicha asociación.

La asignación de direcciones ficticias se realiza de forma secuencial partiendo de un rango de direcciones IPv4 reservadas, definido por el administrador del nodo. Aunque el conjunto de direcciones reservadas sólo tendrá sentido dentro del nodo, es importante que el rango seleccionado no colisione con direcciones utilizadas en el ámbito de red en el que está instalado el nodo. Si así fuera, podría provocar confusión en la comunicación con nodos duales, ya que éstos pueden tener direcciones IPv4 registradas en el DNS. Si BIA utilizara direcciones ficticias en el mismo rango, no se podría diferenciar entre direcciones reales y ficticias.

3.3. Comunicación TCP/UDP

La comunicación TCP requiere el establecimiento de conexión. Una vez establecida, los extremos de la comunicación, direcciones origen y destino, permanecerán fijados. Por tanto, las funciones de la API para el intercambio de datos en una comunicación TCP utilizan únicamente el descriptor de socket, una vez establecida la conexión. Las operaciones de lectura/escritura que las aplicaciones IPv4 realicen sobre el descriptor de socket IPv4 serán realizadas análogamente sobre el descriptor de socket IPv6 en el módulo traductor de funciones de la interfaz dentro de *libbia*. Ambos sockets quedarán relacionados para poder aplicar las mismas operaciones en ambos.

La comunicación UDP no necesita el establecimiento de conexión para el intercambio de información. Con cada operación de lectura de datos del socket puede obtenerse una dirección de origen diferente y con cada operación de escritura puede especificarse un destino diferente. Esta versatilidad puede provocar la necesidad de utilizar el módulo traductor del mapa de direcciones con cada operación de lectura/escritura y en algunos casos incluso el módulo de resolución de direcciones.

3.4. Aplicaciones cliente

Las aplicaciones cliente normalmente realizan las siguientes funciones:

1) Resolución del nombre de máquina que ejecuta la aplicación servidor.
2) Creación del socket.
3) Conexión al servidor (en el caso de comunicaciones UDP, no es obligatorio).
4) Envío y recepción de datos.

Cuando una aplicación cliente necesita resolver el nombre de la máquina del nodo servidor, comprobará si ese nombre existe en el mapa de direcciones y tiene asociada una dirección IPv4 ficticia. Si no está registrado en el mapa de direcciones, utilizará el módulo de extensión de resolución de nombres para que éste obtenga la respuesta proporcionada por el *resolver* de la máquina y para que, en caso de que sea necesario, asocie una nueva dirección ficticia. Esta nueva asociación será almacenada en el mapa de direcciones para posteriores consultas.

La operación de creación de socket no se proporciona en *libbia*, ya que tanto si la aplicación requiere *libbia* como si no, la aplicación IPv4 necesita un punto de acceso al servicio a través de un socket IPv4.

Para establecer la conexión con una aplicación servidor, el cliente proporcionará la dirección IPv4 ficticia que le haya sido asignada anteriormente por el módulo de resolución de nombres de *libbia*. El módulo traductor de funciones de la interfaz utilizará el traductor del mapa de direcciones para comprobar si esta dirección del servidor se encuentra entre las direcciones ficticias asignadas por *libbia*, en cuyo caso deberá establecer la comunicación a través de IPv6. Es en este momento cuando será necesaria la creación de un socket IPv6 dentro de la librería *libbia* para que la comunicación pueda realizarse y asociar ambos sockets, el de la aplicación IPv4 y el de *libbia* IPv6.

Cualquier operación de lectura/escritura desde la aplicación realizada sobre el socket IPv4, será efectuada por el módulo de traductor de funciones de la interfaz en el socket IPv6.

En el caso de sockets TCP, una vez establecida la conexión las operaciones de lectura/escritura de la aplicación sobre el socket IPv4 serán efectuadas igualmente sobre el socket IPv6, prácticamente sin más cambios que el descriptor de socket utilizado. En cambio, en el caso de utilizar comunicación UDP, un mismo socket puede ser utilizado para el envío/recepción de datos con diferentes servidores, necesitando al menos el módulo traductor de mapa de direcciones con cada operación.

3.5. Aplicaciones servidor

Las aplicaciones servidor normalmente realizan las siguientes funciones:

1) Creación del socket.
2) Asociación del socket a una o todas las interfaces de red.
3) Escuchar y aceptar peticiones de clientes (en el caso de comunicaciones TCP).
4) Envío y recepción de datos.

La aplicación servidor IPv4 creará un socket y al igual que en el caso de los clientes, esta operación se realiza de forma transparente a libbia.

Una aplicación servidor puede esperar datos de los clientes, en una de sus interfaces concretas de red, o en cualquiera de ellas.

Si el servidor desea recibir la información de los clientes en una de sus interfaces de red concretas, asociará una de las direcciones IP que tiene configurada la máquina donde está ejecutándose al socket. Este caso puede provocar problemas en el funcionamiento de BIA, ya que una aplicación IPv4 no puede seleccionar una de las interfaces de red IPv6 (sólo hay conectividad IPv6) de la máquina en la que se está ejecutando. El mecanismo BIA sólo funcionará para este caso concreto, si la forma de seleccionar la interfaz de red se realiza a través del nombre de la máquina que tiene asociada la interfaz de red que la aplicación desea utilizar. Para ello, se usará el mismo mecanismo de resolución que utilizan los clientes cuando obtienen la dirección IP de los servidores.

Si el servidor desea recibir la información de los clientes en cualquiera de sus interfaces de red utilizará la dirección especial *INADDR_ANY* que será convertida por el módulo traductor de funciones de la interfaz en *IN6_ADDRANY*.

Cuando la aplicación IPv4 utilice la función para escuchar peticiones de clientes, *libbia* necesitará aplicar la misma función sobre el socket IPv6, sin necesidad de invocar ningún otro módulo ya que esa función es independiente de la versión de protocolo IP utilizada.

Las peticiones de conexión de los clientes TCP deben de ser aceptadas antes de comenzar el intercambio de información. Esta operación debe devolver a la aplicación IPv4 la dirección del cliente que está solicitando la conexión. En el caso de estudio, el cliente está utilizando IPv6 para comunicarse con el servidor, por tanto, la dirección origen de la comunicación es IPv6. Es necesario realizar una traducción de la dirección IPv6 de origen a una IPv4 que la aplicación servidor pueda gestionar. La librería *libbia* deberá almacenar esta nueva asociación en el módulo mapa de direcciones con el fin de poder enviar las respuestas del servidor.

Las funciones de envío/recepción de datos funcionan de la misma forma que en el caso cliente.

4. Limitaciones

Durante las pruebas de este mecanismo, se han detectado ciertas limitaciones en las que BIA no puede resolver el problema:

- Cuando las aplicaciones intercambian direcciones IP como datos de aplicación, caso típico de FTP. El nodo remoto recibirá en los datos del protocolo la dirección IP que la aplicación local maneja de sí misma, una dirección ficticia.

- Las aplicaciones que no utilizan la resolución de nombres/direcciones para averiguar las direcciones IP y rellenar las estructuras de direcciones (*sockaddr*, *sockaddr_in*) que las funciones de la API necesitan, no obtendrán direcciones ficticias proporcionadas por BIA y el mecanismo fallará.

- BIA asigna las direcciones de forma dinámica y temporal, sólo serán válidas durante el período de tiempo, mientras se mantengan en uso. Por tanto, si la aplicación almacena direcciones IP para utilizarlas nuevamente en otras sesiones, el mecanismo fallará.

- Si un nodo remoto tiene configuradas tanto direcciones IPv4 como IPv6 pero sólo es alcanzable a través de la red IPv6, la resolución de nombres proporcionará a la aplicación IPv4 una dirección IPv4 válida sin necesidad de utilizar la asociación de direcciones ficticias. Dado que el nodo es alcanzable utilizando IPv6, la aplicación fallará en cualquier intento de comunicación con dicho nodo. Es posible mejorar el mecanismo BIA para detectar este problema en el momento de la resolución de nombres, comprobando en ese instante si el nodo es alcanzable a través de la dirección IP obtenida. Si no lo fuera, BIA podría probar con el resto de las direcciones suministradas por el *resolver* hasta encontrar alguna alcanzable.

5. Seguridad

La implementación de *libbia* es un prototipo en fase experimental y no incluye suficientes mecanismos de seguridad que la conviertan en una forma segura de transición. Actualmente *libbia* presenta las siguientes vulnerabilidades:

- Rechazo de servicio (denial of service). Una entidad malintencionada podría abrir tantas conexiones como número de direcciones ficticias reservadas por BIA, impidiendo que otras aplicaciones pudieran utilizar el mecanismo.

- No soporta un sistema de autenticación que permita la utilización del mecanismo sólo a aquellas aplicaciones autorizadas.

6. Conclusiones y trabajo futuro

BIA permite a las aplicaciones IPv4 sencillas que siguen el clásico modelo cliente/servidor, operar con aplicaciones IPv6, sin necesidad de cambiarlas, ni siquiera recompilarlas. Para ello utiliza la traducción de protocolos en la API de comunicaciones, antes de

construir el paquete IP. BIA es un mecanismo más sencillo ya que no se requiere la traducción de cabeceras que realizan otros mecanismos en el nivel IP.

BIA es un mecanismo muy ligero que puede convertirse en uno de los procesos clave para la interoperabilidad de aplicaciones IPv4 e IPv6. Sin embargo, ciertos detalles de implementación de las aplicaciones IPv4 pueden provocar fallos en BIA. Por este motivo, se requiere un pan de pruebas lo más exhaustivo posible para detectar incompatibilidades y realizar una clasificación aún más detallada de las posibles fuentes de error en el código de las aplicaciones IPv4.

Aplicaciones que utilizan comunicación multicast o sockets en modo crudo no están contempladas en el mecanismo, aunque son caminos alternativos para trabajos futuros. En el caso de multicast es posible realizar una extensión asociando previamente los grupos de multicast IPv4 que utilizan las aplicaciones y los grupos de multicast IPv6 que se utilizarán en el intercambio de información. Los sockets en modo crudo pueden implementarse igualmente como una nueva extensión a *libbia*, tratando cada paquete IP y cada uno de sus campos de forma específica. Los sockets en modo crudo son una forma de capturar mensajes ICMP e informar a las aplicaciones.

Aunque BIA proporciona una solución a corto plazo para la continuidad de aplicaciones IPv4 sobre redes IPv6, es necesario tener presente que debería utilizarse como solución de transición. El objetivo final debe ser portar el código de las aplicaciones a IPv6 para su funcionamiento de forma nativa.

Agradecimientos

Este trabajo contó con la valiosa participación de Pedro de las Heras Quirós, colaborando con sus sugerencias y recomendaciones.

Referencias

[1] K. Egevang, P. Francis, *"The IP Network Address Translator (NAT)"*, IETF RFC 1631, mayo 1994.

[2] S. Deering, R. Hinden, *"Internet Protocol, Version 6 (IPv6) Specification"*, IETF RFC 2460, diciembre 1998.

[3] R. Gilligan, E. Nordmark, *"Transition Mechanisms for IPv6 Hosts and Routers"*, IETF RFC 2893, August 2000.

[4] K. Tsuchiya, H. Higuchi, Y. Atarashi, *"Dual Stack Hosts using the Bump-In-the-Stack Technique (BIS)"*, IETF RFC 2767, febrero 2000.

[5] S. Lee, M-K. Shin, Y-J. Kim, E. Nordmark, A. Durand, *"Dual Stack Hosts Using Bump-in-the-API (BIA)"*, IETF RFC 3338, octubre 2002.

[6] S. Lee "Bump in the API experience", Global IPv6 Summit Conference, Ontario, Canada, mayo 2001 (http://www.ipv6forum.com/navbar/events/ottawa01/presentations/pdf/lee.pdf).