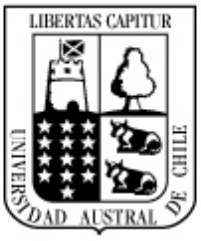


Universidad Austral de Chile

Escuela Ingeniería en Informática

DISEÑO E IMPLEMENTACIÓN DE SOFTWARE PARA PROTOCOLO IPV6.

Tesis de Grado para optar al Titulo
de Ingeniero Civil en Informática**.**

Profesor Patrocinante:
Luis Vidal
Ingeniero Civil en Informatica

MOISÉS EDUARDO CORONADO DELGADO
Valdivia – Chile
2004

Universidad Austral de Chile
Instituto de Informática

Valdivia, 29 de septiembre de 2004.

De : Luis Hernán Vidal Vidal.
A : Sra. Miguelina Vega R.
Directora de Escuela de Ingeniería Civil en Informática.
Ref. : Informa Calificación Trabajo de Titulación.

MOTIVO: Informar revisión y calificación del Proyecto de Título "Diseño e implementación de software para protocolo IPV6", presentado por el alumno Moisés Coronado Delgado, que refleja lo siguiente:

Se logró el objetivo planteado que permitió presentar las metodologías disponibles para el diseño de software para IPV6.

La revisión hecha sobre los estándares y tecnologías, especialmente en los métodos empleados para la migración de software que es soportado por IPV4 a IPV6 fue muy bien descrita y se presenta como un valioso aporte en el área de redes.

Una de las características más sobresalientes presentes en el trabajo de titulación tiene relación con las pruebas que se tuvieron que realizar, las cuales constituyen un muy buen referente no solo para el diseño de software para IPV6, si que además para el estudio del protocolo IPV6.

Por todo lo anterior expuesto califico el trabajo de titulación del Sr. Moisés Coronado Delgado con nota 7,0 (siete coma cero).

Sin otro particular, se despide atentamente.

UNIVERSIDAD AUSTRAL DE CHILE
INSTITUTO DE INFORMATICA
FACULTAD DE CIENCIAS DE LA INGENIERIA

Ing. Luis Hernán Vidal Vidal.
Profesor Instituto de Informática.
Facultad de Ciencias de la Ingeniería.
Universidad Austral de Chile.

Valdivia, 10 de Septiembre de 2004

De : Christian Lazo Ramírez

A : Directora Escuela Ingeniería Civil en Informática

Ref. : Informe Calificación Trabajo de Titulación

Nombre Trabajo de Titulación:

"DISEÑO E IMPLEMENTACIÓN DE SOFTWARE PARA PROTOCOLO IPV6"

Nombre Alumno:

SR MOISÉS EDUARDO CORONADO DELGADO

Nota: **6,1** **seis coma uno**

Evaluación:

Cumplimiento del objetivo propuesto	7.0
Satisfacción de alguna necesidad	6.0
Aplicación del método científico	5.5
Interpretación de los datos y obtención de conclusiones	6.0
Originalidad	7.0
Aplicación de criterios de análisis y diseño	6.0
Perspectivas del trabajo	6.5
Coherencia y rigurosidad lógica	6.0
Precisión del lenguaje técnico en la exposición, composición, redacción e ilustración	5.0
Nota Final	**6.1**

Sin otro particular, atte.:

Valdivia, 20 de Septiembre de 2004

De : Eugenio Ponisio Fernández

A : Directora Escuela Ingeniería Civil en Informática

Ref. : Informe Calificación Trabajo de Titulación

Nombre Trabajo de Titulación:

"DISEÑO E IMPLEMENTACIÓN DE SOFTWARE PARA PROTOCOLO !PV6"

Nombre Alumno:

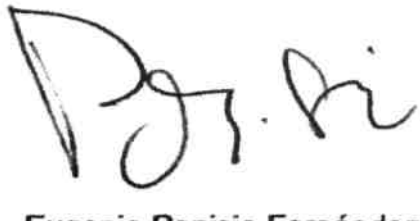
SR. MOISÉS EDUARDO CORONADO DELGADO

Nota: **5,0** **cinco coma cero**

Evaluación:

Cumplimiento del objetivo propuesto	5,5
Satisfacción de alguna necesidad	5.0
Aplicación del método científico	5.0
Interpretación de los datos y obtención de conclusiones	5,0
Originalidad	6,0
Aplicación de criterios de análisis y diseño	5,0
Perspectivas del trabajo	5,0
Coherencia y rigurosidad lógica	4,5
Precisión del lenguaje técnico en la exposición, composición, redacción e ilustración	4,0
Nota Final	5,0

Sin otro particular,



Eugenio Ponisio Fernández

ÍNDICE

Resumen

La versión actual del Protocolo Internet (denominada IP versión 4 o IPv4) no ha cambiado de forma significativa desde la publicación del documento RFC 791 en 1981. IPv4 ha demostrado ser un protocolo robusto, de fácil implementación e ínter operable, y ha superado la prueba de ampliar un conjunto de redes interconectadas para un uso global del tamaño que Internet tiene en la actualidad, siendo estos los objetivos del diseño inicial. Sin embargo, el diseño inicial no previó algunas circunstancias que en la actualidad hacen imprescindible la existencia de una mejora a este protocolo. Para solucionar estos problemas, el Grupo de trabajo de ingeniería de Internet (IETF) ha desarrollado un conjunto de protocolos y estándares denominados IP versión 6 (IPv6) [RFC 246].

“El valor de IPv6 sólo puede aprovecharse si el esfuerzo en su implantación se realiza de una forma amplia, a escala global” fueron las palabras de Dr. Vint Cerf Presidente Honorario del IPv6 Forum y un fundador de Internet. En el caso específico sobre el cual se aborda este proyecto de tesis, si bien el protocolo en general está en su mayor parte implementado, hace falta la creación de aplicaciones compatibles para apoyar los esfuerzos de crear un entorno receptivo para el desarrollo, concepción y uso de IPv6 en la Internet global. El enfoque específico que tendrá este proyecto de tesis es el análisis, diseño e implementación de una aplicación Compatible con el nuevo protocolo teniendo como punto de partida el desarrollo actual que tiene la Tecnología Multicast IPv6.

Abstract

The actual version of internet protocol (named IP version 4 or IPv4) has not changed in a significative way from publishment of file RFC 791 in 1981. IPv4 has shown to be a strong protocol being of easy implementation and interoperable, and it has surpassed the test of expanding a group of nets interconnected for a global use of the size that internet has currently being these the objectives of the initial design. Nevertheless the initial design didn't predict some circumstances that currently do (indispensable) the existence of an improvement to this protocol. To solve these problems the work group of internet engineers (IETF) has developed a group of protocols and standards named IP version 6 (IPv6) [FFC 246].

"The value of IPv6 only it can be taken advantage of if the effort in its introduction is carried out of an extensive form, to global scale." they were the words of Dr. Vint Cerf, honorary president of IPv6 Forum and founder of Internet. In the specific case on this project of thesis is undertaken though the protocol in general is in its greater part implemented, does lack the creation of compatible applications to support the efforts to create a receptive environment for the development, conception and use of IPv6 in the global Internet. The focus specify that it will have this project of thesis is the analysis, design and implementation of a compatible application with the new protocol having like point of departure the present development that has the technology multicast IPv6.

CAPITULO 1

1. Introducción

A la investigación y defensa, aplicaciones que originaron la aparición de Internet, se han sumado desde entonces una larga lista de usos y servicios: distribución de música, comercio electrónico, videoconferencia, telefonía y una larga cantidad de actividades cuyo denominador común es la utilización de Internet.

Miles de dispositivos móviles surgen día tras día con un único objetivo; que podamos estar conectados a la Red en cualquier momento y desde cualquier lugar.

Así la compañía de encuestas e investigación internacional (IDC) señala que más de 260 millones de personas tienen actualmente acceso a Internet, cifra que se elevará, puesto que cada segundo ingresan siete usuarios nuevos a Internet. A las puertas de tamaña revolución, la pregunta obligada es: ¿está preparado Internet para todo esto?, dar cabida a más usuarios y dispositivos. Ese es el reto al que se enfrenta actualmente Internet y que ha dado lugar a su revisión y renovación.

Los organismos encargados de velar por el correcto funcionamiento de la Red impulsaron en 1994 un debate conocido como IP Next Generation. El objetivo no era otro que encontrar una nueva arquitectura, un nueva Internet que hiciese frente a las necesidades de la Sociedad de la Información.

El resultado de estas investigaciones y estudios sería IPv6, el nuevo protocolo IP al que ya se ha bautizado como "la Internet del nuevo milenio".

El nuevo protocolo de comunicación representa un paso más allá, una clara y necesaria evolución de IPv4, el estándar que se utiliza actualmente y que tras 20 años de vida, se ha visto desbordado por el crecimiento de la Red.

IPv4 tiene una historia de gran éxito, con cerca de 200 millones de usuarios alrededor del mundo. Los cimientos de tan exitosa industria no pueden ser cambiados de la noche a la mañana. El cambio solo puede hacerse gradualmente, y con extremo cuidado para evitar cualquier impacto del tráfico vía IPv4.

Otro obstáculo para el despliegue masivo del IPv6 es la falta de estructura para un despliegue real en el ámbito de producción. Soporte de hardware, algunos sistemas operativos, aplicaciones, herramientas de manejo, y personal técnico entrenado es necesario para completar el cuadro. Cada uno de estos elementos faltantes está siendo tratado por diferentes entidades a nivel mundial y con gran éxito.

1.1. Importancia del proyecto:

Con referencia en la sección anterior con el nivel actual de avance en el desarrollo del protocolo una de las áreas que mayor preocupación está teniendo es el creación de aplicaciones IPv6 y también la migración de aplicaciones desde IPv4 a IPv6, es por esto que este proyecto de tesis busca ser un aporte a los esfuerzos globales por hacer de la implementación definitiva de IPv6 una realidad.

1.2. Objetivos

1.2.1. Objetivos Generales.

1. Conocer las principales características de la nueva versión del protocolo de red de la arquitectura TCP/IP: IP versión 6.
2. Comprender su funcionamiento y aplicar los conocimientos adquiridos efectuando así un aporte al desarrollo e implantación de IPv6 en Chile.
3. Probar las prestaciones de las futuras redes IPv6, y el uso no comercial nativo de los servicios y aplicaciones avanzadas IPv6.
4. El proyecto investigará, diseñará, e implantará una metodología para el desarrollo de aplicaciones basadas en el nuevo protocolo IPv6.

1.2.2. *Objetivos Específicos*.

1. Realizar una investigación respecto al desarrollo actual del Protocolo IPv6.
2. Efectuar un estudio sobre lenguajes y Sistemas Operativos adecuados para el desarrollo de aplicaciones IPv6.
3. Efectuar un estudio sobre las nuevas características de IPv6 para el desarrollo de aplicaciones Internet.
4. Crear un método de desarrollo de Software Compatible con IPv6.

.

CAPITULO 2- IP Versión 6

2. Introducción

IPv4 ha demostrado ser un protocolo robusto, de fácil implementación e ínter operable, y ha superado la prueba de ampliar un conjunto de redes interconectadas para un uso global del tamaño que Internet tiene en la actualidad. Éstas son las virtudes de su diseño inicial.

Sin embargo, el diseño inicial no previó las siguientes circunstancias:

- El reciente crecimiento exponencial de Internet y el agotamiento inminente del espacio de direcciones IPv4.

  Las direcciones IPv4 han empezado a escasear relativamente, lo que ha obligado a algunas organizaciones a utilizar un traductor de direcciones de red (NAT, *Network Address Translator*) para asignar múltiples direcciones privadas a una única dirección IP pública. Si bien los NAT fomentan la reutilización del espacio de direcciones privadas, no admiten la seguridad de nivel de red basada en estándares o la asignación correcta de todos los protocolos de nivel superior y pueden crear problemas al conectar dos organizaciones que utilizan el espacio de direcciones privadas.

  Además, la importancia cada vez mayor de los dispositivos conectados a Internet garantiza que acabará por agotarse el espacio de direcciones IPv4 públicas.

- El crecimiento de Internet y la capacidad de los routers de la red troncal de Internet para mantener tablas de enrutamiento grandes.

Debido a la forma en que los Id. de red de IPv4 se han asignado y se siguen asignando, lo normal es que existan más de 70.000 rutas en las tablas de enrutamiento de los routers de red troncal de Internet. La infraestructura actual de enrutamiento de la red Internet IPv4 es una combinación de enrutamiento plano y jerárquico.

- La necesidad de una configuración más sencilla.

La mayoría de las implementaciones actuales de IPv4 se deben configurar manualmente o mediante un protocolo de configuración de direcciones con estado como el Protocolo de configuración dinámica de host (DHCP, *Dynamic Host Configuration Protocol*). Al existir más equipos y dispositivos que utilizan IP, surge la necesidad de una configuración de direcciones más sencillas y automática y otras opciones de configuración que no dependan de la administración de una infraestructura DHCP.

- El requisito de seguridad en el nivel de IP.

  La comunicación privada a través de un medio público como Internet requiere servicios de cifrado que impidan que los datos enviados se puedan ver o modificar durante el tránsito. Aunque en la actualidad existe un estándar que proporciona seguridad para los paquetes IPv4 (denominado seguridad de Protocolo Internet o IPSec), este estándar es opcional y prevalecen las soluciones propietarias.

- La necesidad de mayor compatibilidad con la entrega de datos en tiempo real (denominado también calidad de servicio).

  Aunque existen estándares de calidad de servicio (QoS, *Quality of Service*) para IPv4, la compatibilidad con el tráfico en tiempo real depende del campo Tipo de Servicio (TOS, *Type of Service*) de IPv4 y la identificación de la carga, que suele utilizar un puerto UDP o TCP. Por desgracia, el campo TOS de IPv4 tiene una funcionalidad limitada y diferentes interpretaciones. Además, la identificación de la carga que utiliza un puerto TCP o UDP no es posible cuando la carga del paquete IPv4 está cifrada.

  Para solucionar estos problemas, el Grupo de trabajo de ingeniería de Internet (IETF, *Internet Engineering Task Force*) ha desarrollado un conjunto de protocolos y estándares denominados IP versión 6 (IPv6). Esta nueva versión, anteriormente llamada IP, La siguiente generación (IPng, *IP-The Next Generation*), incorpora los conceptos de muchos métodos propuestos para la actualización del protocolo IPv4. IPv6 está diseñado con la intención de reducir al mínimo el impacto en los protocolos de nivel superior e inferior al evitar la adición arbitraria de nuevas características.

3. Protocolo Internet versión 6

IPv6 está definido en el documento RFC 2460, "Internet Protocol, Version 6 (IPv6) Specification" [Especificación del Protocolo Internet, versión 6 (IPv6)]. IPv6 es un protocolo de datagramas sin conexión no confiable, que se utiliza principalmente para el direccionamiento y enrutamiento de paquetes entre hosts. Sin conexión significa que no se establece una sesión antes de intercambiar datos. No confiable significa que la entrega no está garantizada. IPv6 siempre intenta por todos los medios entregar los paquetes. Un paquete IPv6 se puede perder, entregar fuera de secuencia, duplicar o retrasar. IPv6 no intenta recuperarse de estos tipos de errores. La confirmación de paquetes entregados y la recuperación de paquetes perdidos se efectúan mediante un protocolo de nivel superior, como TCP. En estos aspectos IPv6 no se diferencia enormemente de IPv4.

En las siguientes secciones se detallaran algunas de las diferencias más notables de ipv6 frente a ipv4

3.1. La Cabecera IPv6.

La cabecera de un paquete IPv6 es, sorprendentemente, mas sencilla que la del paquete IPv4. Y recordemos que además la funcionalidad del protocolo IPv6 es mucho mayor.

La cabecera de un paquete IPv4 es variable, por lo que necesita un campo de tamaño o lenght. Sin embargo, para simplificar la vida de los routers, IPv6 utiliza un tamaño de cabecera fijo de 40 bytes, que componen un total de ocho campos:

- *Versión (4 bits*), sirve para que el router se entere de que es un paquete IPv6.
- *Dirección origen y de destino* (128 bits cada una), son las direcciones de los nodos IPv6 que realizan la comunicación.
- *Clase de tráfico* (8 bits), para poder diferenciar entre servicios sensibles a la latencia, como VoIP, de otros que no necesitan prioridad, como trafico http.
- *Etiqueta de flujo* (20 bits), permite la diferenciación de flujos de tráfico. Esto tiene importancia a la hora de manejar la calidad de servicio (QoS)
- *Siguiente cabecera* (8 bits), este campo permite a routers y hosts examinar con más detalle el paquete. A pesar de que el paquete básico IPv6 tiene cabecera de tamaño fijo, el protocolo puede añadir más para utilizar otras características como encriptación y autenticación.
- *Tamaño de payload* (16 bits), describe el tamaño en octetos de la sección de datos del paquete. Al ser este campo de 16 bits, podremos usar paquetes de hasta más de 64000 bytes.
- *Límite de saltos* (8 bits), especifica el número de saltos de router que puede hacer el paquete antes de ser desechado. Con 8 bits podremos tener un máximo de 255 saltos.

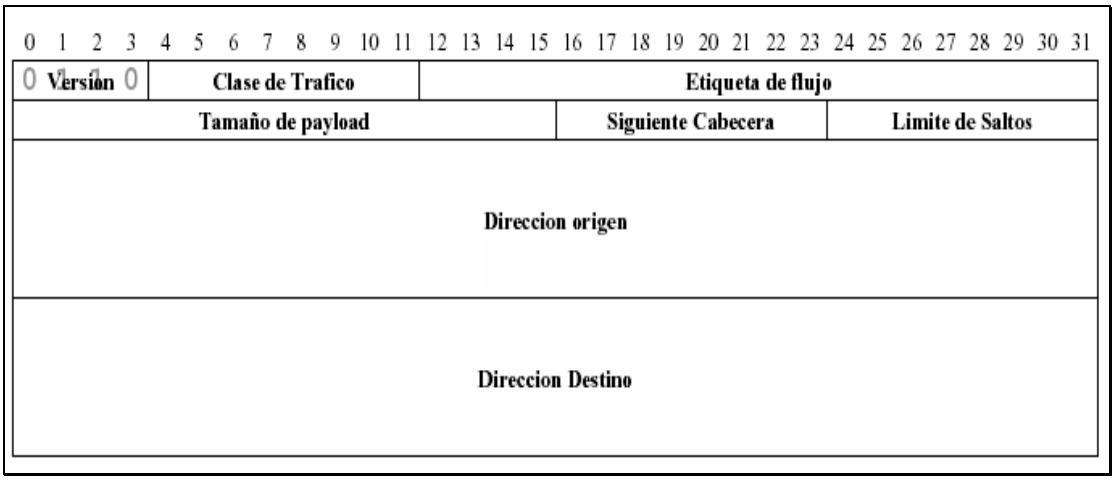


Figura 1

3.2. El Campo de Siguiente Cabecera (Next Header field)

Como se mencionó antes, el tamaño de la cabecera IPv6 básica es fijo. Dentro de esta cabecera existe un campo llamado de Siguiente Cabecera que permite describir con más detalle las opciones del paquete. Esto quiere decir que en realidad tendremos una cabecera de tamaño fijo por norma general y otra cabecera de tamaño variable en caso de que utilicemos alguna de las características avanzadas.

En el campo de *Siguiente Cabecera* se codificarán las siguientes opciones:

Siguiente Cabecera	Valor del Campo
Hop-by-Hop Options Header	0
Opciones de destino	60
Encaminamiento	43
Fragmento	44
Autentificación	51
Encapsulación	50
Ninguna	59

Tabla 1

3.2.1. Hop-by-Hop Options Header

Usado para transportar información opcional que debe ser examinada por todos los nodos del camino que recorre el paquete. Se identifica con el campo Next Header en el Header del paquete igual a cero.

3.2.2. Routing Header

Es usado por el origen para listar uno o más nodos intermedios para alcanzar el destino. Es muy similar a las opciones de ruteo de IPv4. Este Header se configura con el número 43 en campo Next Header del Header anterior.

3.2.3. Destination Options Header

El Destination Options Header es usado para mover información opcional para ser analizada en el destino final. Se identifica por el valor de Next Header 60 (igual que IPv4).

3.2.4. Fragment Header

El Fragment Header es usado por un origen IPv6 para mandar paquetes más grandes que el MTU del camino.

A diferencia de IPv4, en IPv6 la fragmentación es realizada por el nodo origen. Este Header es identificado con el valor 44 en el campo Header anterior.

3.2.5. No Next Header

El valor 59 en el campo Next Header de un Header IPv6 o en cualquier extensión header indica que no sigue nada. Si el campo longitud del Header IPv6 indica presencia de octetos después del header que tiene el Next Header en 59, estos octetos son ignorados.

4. Internet Control Message Protocol Para IPv6 (ICMPv6)

El Protocolo de control de mensajes Internet para IPv6 (ICMPv6, Internet Control Message Protocol for IPv6) es un estándar de IPv6 necesario que está definido en el documento RFC 2463, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Versión 6 (IPv6) Specification" [Especificación del Protocolo de control de mensajes de Internet (ICMP) para el Protocolo Internet versión 6 (IPv6)]. Con ICMPv6, los hosts y los routers que se comunican mediante IPv6 pueden informar de errores y enviar mensajes de eco simples.

4.1. Tipos de ICMPv6

Los mensajes de ICMP, se han dividido en 2 clases, los que comunican errores, y los que solicitan o entregan información sobre un nodo. Para diferenciarlos, se han adjudicado una numeración del 0 al 127 a los mensajes que contienen información y del 128 al 255, sobre los que informan de algún tipo de error de una petición.

Un paquete ICMPv6, está formado por una cabecera IPv6, y es precedido inmediatamente por una cabecera con valor 58 en el campo next header, Nótese que este procedimiento es diferente al de IPv4, y que un ICMP puede ser insertado en cualquier tipo de paquetes.

4.1.1. Tipos de ICMPv6 de Información

Los mensajes de información, pueden ser del tipo:

- **Echo Request (Type 128):** Un nodo, puede enviar un ICMP Echo Request (Mas conocidos como pings), para saber el tiempo de respuesta de otro host.
- **Echo Reply (Type 129):** El ICMP Echo Reply, es enviado como respuesta a un ICMP Echo Request. El ICMP Echo Reply debe ser trasportado al proceso que origino el ICMP Echo Request.

4.1.2. Tipos de ICMPv6 de error:

- **Destination Unreachable:** es mandado por un router, o por cualquier nodo, para informar de la imposibilidad de que un paquete llegue a su destino. NO se deberían mandar ICMPv6, si son ocasionados por problemas de congestión de la red.

  Estos ICMP se dividen en subclases, según el tipo de problema que haya ocasionado su emisión:

  -Si el error es ocasionado por un envío de un paquete al nodo erróneo, este enviará un ICMPv6 con código 0

  -Si el error es ocasionado por un envío hacia un destino cerrado por causas administrativas (un firewall por ejemplo), se debe enviar un ICMP de código 1.

  -Si el error es ocasionado por la imposibilidad de resolver la dirección IP de un link, se enviará un ICMPv6 con código 3.

-Si el error es ocasionado por un fallo en la capa de trasporte y si el puerto esta disponible para la misma, se enviará un ICMP con código 4. Por ejemplo, un paquete TCP enviado a un puerto UDP.

Packet Too Big (Type 2): es enviado cuando el tamaño máximo de un paquete es superior a la MTU del interfaz de red al que se ha enviado. También es enviado por un router, si el siguiente salto tiene un MTU inferior al tamaño del paquete. Este ICMP, puede ser usado para saber el MTU de un path.

Time Exceeded (Type 3): Si un router recibe un paquete con el Hop limit a 0, o si es El quien lo tiene que poner a 0, el paquete es descartado y se envía un ICMPv6 Time Exceeded. Si un host, no puede ensamblar un paquete en un tiempo x, descartará todos los fragmentos recibidos y también enviará un ICMP de esta clase.

Parameter Problem (Type 4): Si un nodo IPv6, al procesar un paquete, encuentra un error en uno de los parámetros de sus campos, enviará un ICMP Parameter Problem informando al destino de la situación del error en el paquete.

El protocolo ICMPv6 proporciona también un marco de trabajo para los protocolos siguientes:

- Descubrimiento de vecinos (ND)
- Descubrimiento de escucha de multidifusión (MLD)

4.2. Descubrimiento de vecinos (ND)

El descubrimiento de vecinos (ND, Neighbor Discovery) de IPv6 es un conjunto de mensajes y procesos que determinan las relaciones entre nodos vecinos. El descubrimiento de vecinos reemplaza al Protocolo de resolución de direcciones (ARP, Address Resolution Protocol), el descubrimiento de routers del Protocolo de mensajes de control de Internet (ICMP, Internet Control Message Protocol) y la redirección ICMP, que se utilizan en IPv4, y proporciona funciones adicionales. Descubrimiento de vecinos está definido en el documento RFC 2461, "Neighbor Discovery for IP Version 6 (IPv6)" [Descubrimiento de vecinos para IP versión 6 (IPv6)].

Los hosts utilizan el descubrimiento de vecinos para efectuar las funciones siguientes:

- Descubrir routers vecinos.
- Descubrir direcciones, prefijos de direcciones y otros parámetros de configuración.

Los enrutadores utilizan el Descubrimiento de vecinos para efectuar las funciones siguientes:

- Anunciar su presencia, los parámetros de configuración de host y los prefijos en vínculo.
- Informar a los hosts acerca de la mejor dirección de salto siguiente para reenviar los paquetes dirigidos a un destino determinado.

Los nodos utilizan el descubrimiento de vecinos para efectuar las funciones siguientes:

- Resolver la dirección de nivel de vínculo de un nodo vecino al que se reenvía un paquete IPv6 y determinar cuando ha cambiado la dirección de nivel de vínculo de un nodo vecino.
- Determinar si los paquetes IPv6 se pueden enviar y recibir de un vecino.

A continuación se detallarán cada uno de los procesos que involucra el descubrimiento de vecinos:

Descubrimiento de Routers: Proceso mediante el cual un host descubre los routers locales de un vínculo conectado (equivalente al descubrimiento de routers ICMPv4) y configura automáticamente un router predeterminado (equivalente a una puerta de enlace predeterminada o *gateway* en IPv4).

Descubrimiento de prefijos: Proceso mediante el cual un host descubre los prefijos de red para los destinos locales***.***

Descubrimiento de parámetros: Proceso mediante el cual un host descubre parámetros operativos adicionales, que incluyen la unidad máxima de transmisión (MTU, *Maximum Transmission Unit*) del vínculo y el límite de saltos predeterminado para los paquetes salientes.

Configuración Automática de Direcciones: Proceso de configuración de direcciones IP para las interfaces en presencia o en ausencia de un servidor de configuración de direcciones con estado como el Protocolo de configuración dinámica de host versión 6 (DHCPv6, *Dynamic Host Configuration Protocol version 6*)

Resolución de direcciones: Proceso mediante el cual un nodo resuelve la dirección IPv6 de un nodo vecino en su dirección de nivel de vínculo (equivalente

a ARP en IPv4). La dirección de nivel de vínculo resuelta se almacena como entrada en la caché de vecinos (equivalente a la caché ARP en IPv4) del nodo.

Determinación de Salto Siguiente: Proceso mediante el cual un nodo determina la dirección IPv6 del vecino al que se reenvía un paquete en función de la dirección de destino. La dirección de reenvío o de salto siguiente es la dirección de destino del paquete que se envía o la dirección de un enrutador vecino. La dirección de salto siguiente resuelta para un destino se almacena como entrada en la caché de destinos de un nodo (denominada también caché de enrutamiento).

Detección de vecinos inaccesibles: Proceso mediante el cual un nodo determina que los paquetes IPv6 no se pueden enviar a y recibir de un nodo vecino. Después de determinar la dirección de nivel de vínculo para un vecino, se hace un seguimiento del estado de la entrada en la caché de vecinos. Si el vecino ya no recibe y devuelve paquetes, se quita la entrada de la caché de vecinos. La detección de vecinos inaccesibles proporciona un mecanismo para que IPv6 determine que los hosts o routers vecinos ya no están disponibles en el segmento de red local.

Detección de direcciones duplicadas: Proceso mediante el cual un nodo determina que un nodo vecino no está utilizando una dirección cuyo uso se ha considerado.

Función de redirección: Proceso mediante el cual un router informa a un host de una dirección IPv6 mejor de primer salto, para llegar a un destino (equivalente a la función del mensaje de redirección ICMP de IPv4).

4.3. Descubrimiento de escucha Multicast (MLD)

El uso de multicast en redes IP está definido como estándar TCP/IP en RFC 1112, "Internet Group Management Protocol (IGMP)" [Protocolo de administración del grupo Internet (IGMP)]. En este documento RFC se definen las extensiones de direcciones y hosts para la forma en que los hosts IP admiten la multidifusión. Los mismos conceptos que se desarrollaron originalmente para la versión actual de IP, IP versión 4 (IPv4), se aplican también a IPv6.

El tráfico de multicast se envía a una única dirección, pero se procesa en múltiples hosts. La multidifusión es similar a la suscripción a un boletín. Al igual que sólo los suscriptores reciben el boletín cuando se publica, sólo los equipos host que pertenecen al grupo de multidifusión reciben y procesan el tráfico enviado a la dirección reservada del grupo. El conjunto de hosts que atienden en una dirección Multicast específica se denomina grupo de multidifusión o grupo Multicast.

Otros aspectos importantes de la multidifusión son los siguientes:

- La pertenencia a grupos es dinámica, lo que permite a los hosts unirse al grupo o abandonarlo en cualquier momento.
- La unión a grupos de multidifusión se realiza mediante el envío de mensajes de pertenencia a grupos. En IPv6, los mensajes de Descubrimiento de Escucha de Multidifusión (MLD, *Multicast Listener Discovery*) se utilizan para determinar la pertenencia a grupos en un segmento de red, denominado también vínculo o subred.
- Los grupos no tienen límite de tamaño y los miembros pueden estar repartidos en diversos segmentos de red (si los enrutadores de conexión admiten el reenvío del tráfico de multidifusión y la información de pertenencia a grupos).

- Un host puede enviar tráfico a la dirección del grupo aunque no pertenezca al grupo correspondiente.

4.4. Arquitectura del Direccionamiento

La característica más evidente de IPv6 es la utilización de direcciones de mucho mayor tamaño. El tamaño de una dirección en IPv6 es de 128 bits, que es cuatro veces mayor que el de una dirección de IPv4. Un espacio de direcciones de 32 bits permite 2^{32} o 4.294.967.296 direcciones posibles. Un espacio de direcciones de 128 bits permite 2^{128} o 340.282.366.920.938.463.463.374.607.431.768.211.456 ($3{,}4 \times 10^{38}$) direcciones posibles.

Con IPv6, es aún más difícil concebir el agotamiento del espacio de direcciones IPv6. Para ver el número con perspectiva, un espacio de direcciones de 128 bits proporciona 655.570.793.348.866.943.898.599 ($6{,}5 \times 10^{23}$) direcciones por cada metro cuadrado de la superficie terrestre.

El tamaño relativamente grande de la dirección IPv6 está diseñado para subdividirse en dominios de enrutamiento jerárquicos que reflejen la topología de Internet en la actualidad. La utilización de 128 bits proporciona múltiples niveles de jerarquía y flexibilidad en el diseño del direccionamiento y enrutamiento jerárquicos, que son los elementos de los que carece actualmente la red Internet basada en IPv4.

La arquitectura del direccionamiento IPv6 se describe en el documento RFC 2373, "IP Version 6 Addressing Architecture" (Arquitectura de direccionamiento IP versión 6)

Las direcciones IPv6 son identificadores de 128 bits para interfaces o conjuntos de interfaces. Existen 3 tipos de direcciones:

- **Unicast:** es un identificador para sólo una interfase.
- **Anycast:** es un identificador para un conjunto de interfaces. Un paquete enviado a una dirección anycast es entregado a una de las interfaces identificadas por esa dirección.
- **Multicast:** un identificador para un conjunto de interfaces. Un paquete enviado a una dirección multicast es entregado a todas las interfaces identificadas por esa dirección.

Con IPv6 dejan de existir las direcciones broadcast, cuya funcionalidad es absorbida por las direcciones Multicast.

4.4.1. Ámbitos

El protocolo IPv6 añade soporte para direcciones de distintos ámbitos, lo que quiere decir que tendremos direcciones globales y no globales. Si bien, con IPv4 ya habíamos empleado direccionamiento no global con la ayuda de prefijos de red privados, con IPv6 esta noción forma parte de la propia arquitectura de direccionamiento. Cada dirección IPv6 tiene un ámbito, que es un área dentro de la cual esta puede ser utilizada como identificador único de una o varias interfaces. El ámbito de cada dirección forma parte de la misma dirección, con lo que vamos a poder diferenciarlos a simple vista.

Para las direcciones unicast distinguimos tres ámbitos los cuales serán tratados en mayor profundidad en la sección siguiente:

- **De enlace local (link-local):** para identificar interfaces en un mismo enlace.
- **De sitio local (site-local):** para identificar interfaces en un mismo 'sitio'. La definición de 'sitio' es un tanto genérica, pero en principio un 'sitio' es el área topológica de red perteneciente a un edificio o un campus, perteneciente a una misma organización.
- **Global:** para identificar interfaces en toda Internet.

4.4.2. Asignación actual

De forma similar a como está dividido el espacio de direcciones IPv4, el espacio de direcciones IPv6 está dividido en función del valor de los bits de orden superior de la dirección. Los bits de orden superior y sus valores fijos se denominan Prefijo de Formato (FP, *Format Prefix*).

En la tabla siguiente se muestra la asignación del espacio de direcciones IPv6 por FP.

Asignación	Prefijo de formato (FP)
Reservado	0000 0000
Reservado para asignación NSAP	0000 001
Direcciones globales agregables de unicast	001
Direcciones de unicast locales del vínculo (Link Local)	1111 1110 10
Direcciones de unicast locales del sitio (Site Local)	1111 1110 11
Direcciones de Multicast	1111 1111

Tabla 2

A continuación se describe cada uno de las asignaciones clasificadas por unidifución (Unicast) y multidifución (Multicast)

4.4.3. Direcciones IPv6 Unicast

Una dirección de unicast identifica una sola interfaz en el ámbito del tipo de dirección de unicast. Con la topología adecuada de enrutamiento de unicast, los paquetes dirigidos a una dirección de unicast se entregan en una única interfaz. Los siguientes tipos de direcciones son direcciones IPv6 de unicast:

- **Direcciones globales agregables de unicast:** Las direcciones globales agregables de unicast, que se identifican mediante el Formato de Prefijo (FP, Format Prefix) de 001, equivalen a las direcciones IPv4 públicas. Se pueden enrutar y tener acceso a ellas globalmente en la parte IPv6 de Internet conocida como 6bone (IPv6 backbone, red troncal IPv6). Las direcciones globales agregables de unidifusión también se denominan direcciones globales.

  Como su nombre indica, las direcciones globales agregables de unicast están diseñadas para agregarse o resumirse de forma que produzcan una infraestructura de enrutamiento eficaz. A diferencia de la red Internet actual basada en IPv4, que tiene una mezcla de enrutamiento plano y jerárquico, la red Internet basada en IPv6 se ha diseñado desde la base para admitir direccionamiento y enrutamiento jerárquico eficaz. El ámbito (que es la región del conjunto de redes IPv6 donde la dirección es única) de una dirección global agregable de unicast es la red Internet IPv6 completa.

  Los campos de una dirección global agregable de unidicast se describen de la manera siguiente:

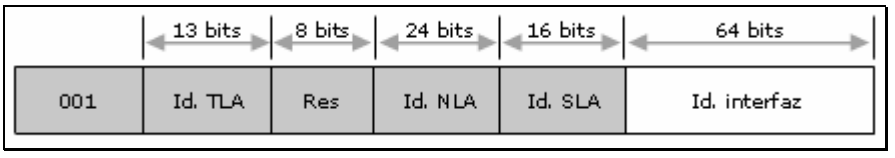


Figura 2

Id. de TLA

El campo Id. de TLA indica el Agregador de Nivel Superior (TLA, *Top Level Aggregator*) de la dirección. El tamaño de este campo es de 13 bits. TLA identifica el nivel superior en la jerarquía de enrutamiento. Los TLA son administrados por IANA y se asignan a los registros de Internet locales que, a su vez, asignan Id. de TLA individuales a grandes proveedores de servicios Internet (ISP) globales. Un campo de 13 bits permite hasta 8.192 Id. de TLA diferentes.

Res

El campo Res está reservado para su uso futuro en la ampliación del tamaño del Id. de TLA o el Id. de NLA. El tamaño de este campo es de 8 bits.

Id. de NLA

El campo Id. de NLA indica el Agregador de Siguiente Nivel (NLA, *Next Level Aggregator*) de la dirección. El campo Id. de NLA se utiliza para identificar un sitio cliente específico. El tamaño de este campo es de 24 bits. El campo Id. de NLA permite que un ISP cree múltiples niveles de jerarquía de direcciones para organizar el direccionamiento y enrutamiento, así como para identificar sitios.

Id. de SLA

El campo Id. de SLA indica el Agregador de Nivel de Sitio (SLA, *Site Level Aggregator*) de la dirección. El campo Id. de SLA sirve para que se identifiquen subredes en el sitio de una organización individual. El tamaño de este campo es de 16 bits. La organización puede utilizar los 16 bits correspondientes a su sitio para crear 65.536 subredes o múltiples niveles de jerarquía de direcciones y una infraestructura de enrutamiento eficaz. Con la flexibilidad de 16 bits para la creación de subredes, un prefijo global agregable de unicast asignado a una organización equivale a asignar a la organización un Id. de red IPv4 de Clase A (siempre y cuando el último octeto se utilice para identificar los nodos en las subredes). El ISP no puede ver la estructura de la red del cliente.

Id. de interfaz

El campo Id. de interfaz indica la interfaz de un nodo en una subred determinada. El tamaño de este campo es de 64 bits.

- **Direcciones de unicast de uso local**

Existen dos tipos de direcciones de unidifusión de uso local:

1. *Direcciones locales del vínculo*, que se utilizan entre vecinos en vínculo y en procesos de descubrimiento de vecinos.
2. *Direcciones locales del sitio*, que se utilizan entre nodos que se comunican con otros nodos del mismo sitio.

Direcciones locales del vínculo

Los nodos utilizan direcciones locales del vínculo, que se identifican mediante el prefijo de formato 1111 1110 10, para comunicarse con nodos vecinos que están en el mismo vínculo. Por ejemplo, en una red IPv6 con

un solo vínculo sin enrutador, las direcciones locales del vínculo se utilizan para la comunicación entre los hosts del vínculo. Las direcciones locales del vínculo equivalen a las direcciones IPv4 de Direccionamiento IP Privado Automático (APIPA, *Automatic Private IP Addressing*), que utilizan el prefijo 169.254.0.0/16. El ámbito de una dirección local del vínculo es el vínculo local. Es necesaria una dirección local del vínculo para los procesos de descubrimiento de vecinos y siempre se configura automáticamente, incluso si no hay ninguna otra dirección de unicast.

Las direcciones locales del vínculo siempre comienzan por FE80. Con el identificador de interfaz de 64 bits, el prefijo de las direcciones locales del vínculo siempre es FE80::/64. Un enrutador IPv6 nunca reenvía el tráfico local del vínculo fuera del vínculo.

Direcciones locales del sitio

Las direcciones locales del sitio, que se identifican mediante el prefijo de formato 1111 1110 11, equivalen al espacio de direcciones privadas de IPv4 (10.0.0.0/8, 172.16.0.0/12 y 192.168.0.0/16). Por ejemplo, las intranet privadas que no tienen una conexión directa enrutada a la red Internet IPv6 pueden utilizar direcciones locales del sitio sin entrar en conflicto con las direcciones globales agregables de unicast. Las direcciones locales del sitio no son accesibles desde otros sitios y los enrutadores no deben reenviar tráfico local del sitio fuera del sitio. Las direcciones locales del sitio se pueden utilizar al mismo tiempo que las direcciones globales agregables de unicast. El ámbito de una dirección local del sitio es el sitio (el conjunto de redes de la organización).

Los primeros 48 bits siempre son fijos en las direcciones locales del sitio y comienzan por FEC0::/48. A continuación de los 48 bits fijos hay un identificador de subred de 16 bits (campo Id. de subred) que proporciona 16 bits con los que se pueden crear subredes en la organización. Al disponer de 16 bits, puede haber hasta 65.536 subredes en una estructura de subredes plana o se pueden subdividir los bits de orden superior del campo Id. de subred para crear una infraestructura de enrutamiento jerárquica y agregable. Después del campo Id. de subred está el campo Id. de interfaz de 64 bits que identifica una interfaz específica de una subred.

- **Direcciones especiales**

Las siguientes son direcciones IPv6 especiales:

Dirección no especificada

La dirección no especificada (0:0:0:0:0:0:0:0 ó ::) sólo se utiliza para indicar la ausencia de dirección. Equivale a la dirección IPv4 no especificada de 0.0.0.0. La dirección no especificada se suele utilizar como dirección de origen en paquetes que intentan comprobar la exclusividad de una dirección tentativa. La dirección no especificada nunca se asigna a una interfaz ni se utiliza como dirección de destino.

Dirección de bucle de retroceso

La dirección de bucle de retroceso (0:0:0:0:0:0:0:1 ó ::1) sirve para identificar una interfaz de bucle de retroceso, lo que permite que un nodo se envíe paquetes a sí mismo. Equivale a la dirección IPv4 de bucle de retroceso de 127.0.0.1. Los paquetes dirigidos a la dirección de bucle de

retroceso nunca se envían en un vínculo ni se reenvían mediante un router IPv6.

Direcciones de compatibilidad

Para facilitar la migración de IPv4 a IPv6 y la coexistencia de ambos tipos de hosts, se han definido las direcciones siguientes:

- Dirección compatible con IPv4:

La dirección compatible con IPv4, 0:0:0:0:0:0:w.x.y.z o ::w.x.y.z (donde w.x.y.z es la representación decimal con puntos de una dirección IPv4), la utilizan los nodos de pila dual que se comunican con IPv6 a través de una infraestructura IPv4. Los nodos de pila dual son nodos con protocolos IPv4 e IPv6. Cuando la dirección compatible con IPv4 se utiliza como destino IPv6, el tráfico IPv6 se encapsula de forma automática con un encabezado IPv4 y se envía al destino mediante la infraestructura IPv4.

- Dirección asignada a IPv4

La dirección asignada a IPv4, 0:0:0:0:0:FFFF:w.x.y.z o ::FFFF:w.x.y.z, se utiliza para representar un nodo exclusivo de IPv4 ante un nodo IPv6. Sólo sirve para la representación interna. La dirección asignada a IPv4 nunca se utiliza como dirección de origen o destino de un paquete IPv6.

Dirección 6to4

La dirección 6to4 se utiliza para la comunicación entre dos nodos que ejecutan IPv4 e IPv6 sobre infraestructura de enrutamiento IPv4. La dirección 6to4 se crea mediante la combinación del prefijo 2002::/16 con los 32 bits de la dirección IPv4 pública del nodo, con lo que se forma un prefijo

de 48 bits. Por ejemplo, para la dirección IPv4 de 131.107.0.1, el prefijo de dirección 6to4 es 2002:836B:1::/48.

Direcciones NSAP

Para proporcionar un medio de asignar direcciones de Punto de Acceso al Servicio de Red (NSAP, *Network Service Access Point*) a direcciones IPv6, las direcciones NSAP utilizan el prefijo de formato 0000001 y asignan los últimos 121 bits de la dirección IPv6 a una dirección NSAP. Para obtener más información acerca de los cuatro tipos de asignación de direcciones NSAP, consulte el documento RFC 1888, "OSI NSAPs and IPv6" (NSAP de OSI e IPv6).

4.4.4. Direcciones IPv6 Multicast

En una dirección Multicast se identifican varias interfaces. Con la topología de enrutamiento de multidifusión adecuada, los paquetes dirigidos a una dirección Multicast se entregan en todas las interfaces identificadas en ella.

Las direcciones IPv6 de Multicast tienen el Prefijo de Formato (FP, *Format Prefix*) de 1111 1111. Se sabe fácilmente que una dirección IPv6 es Multicast porque siempre comienza por FF. Las direcciones Multicast no se pueden utilizar como direcciones de origen.

A continuación del prefijo de formato, las direcciones Multicast incluyen una estructura adicional que identifica su indicador, ámbito y grupo multicast, como se muestra en la figura

siguiente

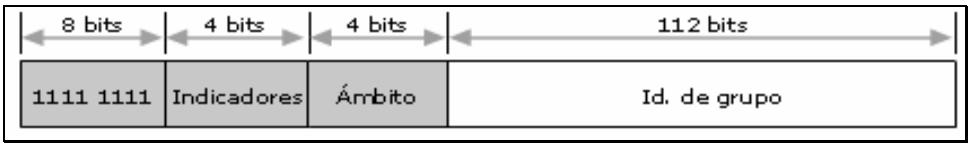


Figura 3

- *Indicadores*

El campo Indicadores corresponde a los indicadores establecidos en la dirección Multicast. El tamaño de este campo es de 4 bits. A partir del documento RFC 2373, el único indicador definido es Transitorio (T). El indicador T utiliza el bit de orden inferior del campo Indicadores. Cuando el indicador T está establecido en 0, especifica que se trata de una dirección de Multicast asignada de forma definitiva (conocida) por la Autoridad de números asignados de Internet (IANA, *Internet Assigned Numbers Authority*). Cuando el indicador T está establecido en 1, especifica que la dirección de Multicast es transitoria (no está asignada de forma definitiva).

- *Ámbito*

El campo Ámbito indica el ámbito del conjunto de redes IPv6 al que va dirigido el tráfico Multicast. El tamaño de este campo es de 4 bits. Además de la información proporcionada por los protocolos de enrutamiento Multicast, los routers utilizan el ámbito Multicast para determinar si se puede reenviar el tráfico de multidifusión.

Los ámbitos siguientes están definidos en el documento RFC 2373

Valor campo de ámbito	Ámbito
1	Local del nodo
2	Local del vínculo
5	Local del sitio
8	Local de la organización
E	Global

Tabla 3

Por ejemplo, el tráfico con la dirección Multicast de FF02::2 tiene ámbito local del vínculo. Un router IPv6 nunca reenvía este tráfico fuera del vínculo local.

- Id. de grupo

El campo Id. de grupo identifica el grupo Multicast y es único en el ámbito. El tamaño de este campo es de 112 bits. Los Id. de grupo asignados de forma definitiva son independientes del ámbito. Los Id. de grupo transitorios sólo son pertinentes para un ámbito específico. Las direcciones Multicast en el intervalo de FF01:: a FF0F:: son direcciones conocidas reservadas.

Para identificar todos los nodos de los ámbitos local del nodo y local del vínculo, se han definido las direcciones Multicast siguientes:

FF01::1 (dirección para todos los nodos de ámbito local del nodo)

FF02::1 (dirección para todos los nodos de ámbito local del vínculo)

Para identificar todos los routers de los ámbitos local del nodo, local del vínculo y local del sitio, se han definido las direcciones Multicast siguientes:

FF01::2 (dirección para todos los routers de ámbito local del nodo)

FF02::2 (dirección para todos los routers de ámbito local del vínculo)

FF05::2 (dirección para todos los routers de ámbito local del sitio)

Con 112 bits en el Id. de grupo, es posible tener 2^{112} identificadores de grupo. Sin embargo, debido a la forma en que las direcciones IPv6 Multicast se asignan a direcciones MAC de multidifusión de Ethernet, en el documento RFC 2373 se recomienda asignar el Id. de grupo de los 32 bits de orden inferior de la dirección IPv6 Multicast y establecer los restantes bits del Id. de grupo original en 0. Al utilizar sólo los 32 bits de orden inferior del Id. de grupo, cada Id. de grupo se asigna a una única dirección MAC de multidifusión de Ethernet.

4.4.5. Direcciones IPv6 Anycast

En una dirección Anycast se identifican varias interfaces. Con la topología de enrutamiento adecuada, los paquetes dirigidos a una dirección Anycast se entregan en una sola interfaz (la interfaz más próxima identificada en la dirección). Dicha interfaz se define como la más próxima en términos de distancia de enrutamiento. Una dirección de multicast se utiliza para la comunicación de uno a muchos y la entrega en varias interfaces. Una dirección de Anycast se utiliza para la comunicación de uno a uno de muchos y la entrega en una sola interfaz.

Para facilitar la entrega en el miembro más próximo del grupo Anycast, la infraestructura de enrutamiento debe conocer las interfaces que tienen asignadas direcciones Anycast y su distancia en términos de métrica de enrutamiento. Por el momento, las direcciones de Anycast sólo se utilizan como direcciones de destino y sólo se asignan a routers. Las direcciones

Anycast se asignan desde el espacio de direcciones de unicast. El ámbito de una dirección Anycast es el ámbito del tipo de dirección de unicast desde el que se asigna la dirección Anycast.

4.4.6. Identificadores de interfaz IPv6

Identificación de interfaz

El Institute of Electrical and Electronic Engineers (IEEE) define la dirección EUI-64 de 64 bits. Las direcciones EUI-64 se asignan a un adaptador de red o se derivan de las direcciones IEEE 802.

Direcciones IEEE 802

Los identificadores de interfaz tradicionales para los adaptadores de red utilizan una dirección de 48 bits que se llama dirección IEEE 802. Esta dirección consta de un Id. de compañía (también llamado Id. de fabricante) de 24 bits y un Id. de extensión (también llamado Id. de tarjeta) de 24 bits. La combinación del Id. de compañía, que se asigna de forma única a cada fabricante de adaptadores de red, y el Id. de tarjeta, que se asigna de forma única a cada adaptador de red en el momento del ensamblaje, genera una dirección única global de 48 bits. Esta dirección de 48 bits también se denomina dirección física, de hardware o de control de acceso a medios (MAC, *Media Access Control*).

- Universal o local (U/L)

  El bit U/L es el séptimo bit del primer byte y se utiliza para determinar si la dirección se administra de forma universal o local. Si el bit U/L está establecido en 0, la administración de la dirección corresponde a IEEE, mediante la designación de un Id. de compañía único. Si el bit U/L está

establecido en 1, la dirección se administra de forma local. El administrador de la red ha suplantado la dirección de fábrica y ha especificado una dirección distinta.

- Individual o grupo (I/G)

El bit I/G es el bit de orden inferior del primer byte y se utiliza para determinar si la dirección es individual (unicast) o de grupo (multicast). Si está establecido en 0, la dirección es de unicast. Si está establecido en 1, la dirección es de multicast.

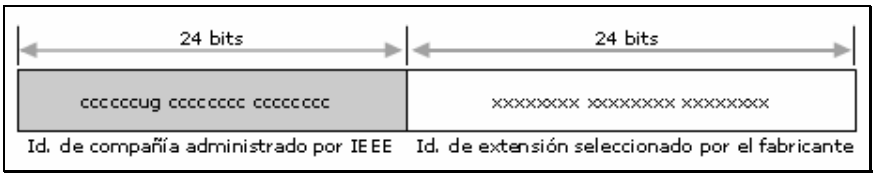


Figura 4

En una dirección típica de adaptador de red 802.x, los bits U/L e I/G están establecidos en 0, lo que corresponde a una dirección MAC de unicast administrada de forma universal.

Direcciones IEEE EUI-64

La dirección IEEE EUI-64 representa un nuevo estándar para el direccionamiento de interfaces de red. El Id. de compañía sigue teniendo 24 bits de longitud, pero el Id. de extensión tiene 40 bits, por lo que se crea un espacio de direcciones mucho mayor para los fabricantes de adaptadores de red. La dirección EUI-64 utiliza los bits U/L e I/G de la misma forma que la dirección IEEE 802.

Asignación de direcciones EUI-64 a identificadores de interfaz IPv6

Para obtener el identificador de interfaz de 64 bits para las direcciones IPv6 de unicast, se complementa el bit U/L de la dirección EUI-64 (si es 1, se establece en 0; y si es 0, se establece en 1). En la figura siguiente se muestra la conversión de una dirección EUI-64 de unicast administrada de forma universal.

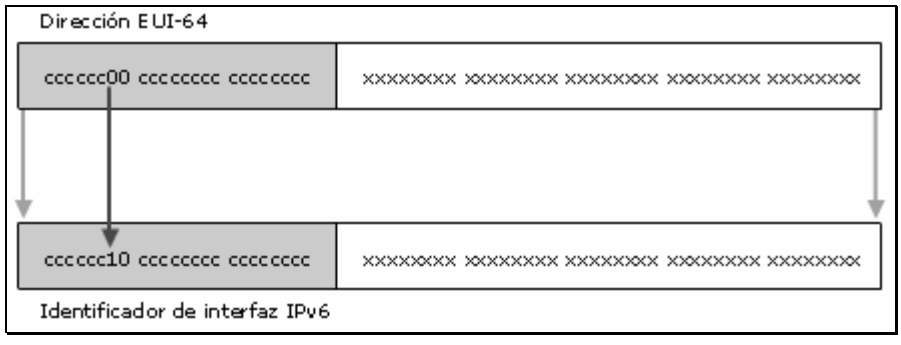


Figura 5

Para obtener un identificador de interfaz IPv6 a partir de una dirección IEEE 802, primero se debe asignar la dirección IEEE 802 a una dirección EUI-64 y, después, complementar el bit U/L. En la ilustración siguiente se muestra el proceso de conversión de una dirección IEEE 802 de unidifusión administrada de forma universal.

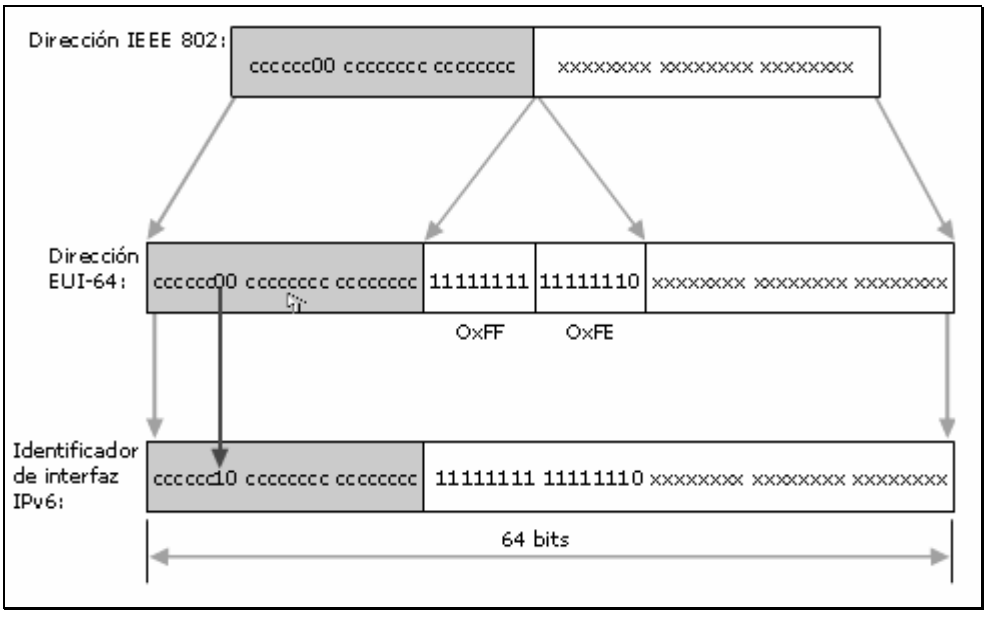


Figura 6

Ejemplo de conversión de una dirección IEEE 802

El host A tiene la dirección MAC de Ethernet de 00-AA-00-3F-2A-1C. Primero, se convierte al formato EUI-64 insertando FF-FE entre el tercer y cuarto bytes, con el resultado de 00-AA-00-FF-FE-3F-2A-1C. Después, se complementa el bit U/L, que es el séptimo bit del primer byte. El primer byte en formato binario es 00000000. Al complementar el séptimo bit, se convierte en 00000010 (0x02). El resultado final es 02-AA-00-FF-FE-3F-2A-1C que, cuando se convierte a notación hexadecimal con dos puntos, da como resultado el identificador de interfaz 2AA:FF:FE3F:2A1C. En consecuencia, la dirección local del vínculo correspondiente al adaptador de red que tiene la dirección MAC de 00-AA-00-3F-2A-1C es FE80::2AA:FF:FE3F:2A1C.

Configuración automática de direcciones IPv6

Un aspecto muy útil de IPv6 es su capacidad para configurarse automáticamente sin utilizar un protocolo de configuración con estado, como el Protocolo de configuración dinámica de host para IPv6 (DHCPv6, *Dynamic Host Configuration Protocol for IPv6*). De forma predeterminada, un host IPv6 puede configurar una dirección local del vínculo para cada interfaz. Mediante el descubrimiento de enrutadores, un host puede también determinar las direcciones de los enrutadores, direcciones adicionales y otros parámetros de configuración. El mensaje de anuncio de enrutador incluye una indicación de si se debe utilizar un protocolo de configuración de direcciones con estado.

La configuración automática de direcciones sólo se puede llevar a cabo en interfaces con capacidad para multidifusión. La configuración automática de direcciones se describe en el documento RFC 2462, "IPv6 Stateless Address Autoconfiguration" (Configuración automática de direcciones sin estado en IPv6).

Estados de direcciones configuradas automáticamente

Las direcciones configuradas automáticamente tienen uno o varios de los estados siguientes:

- Tentativo

  Estado de una dirección cuya exclusividad está en proceso de comprobación. La comprobación se produce mediante la detección de direcciones duplicadas.

- Preferido

  Estado de una dirección cuya exclusividad se ha comprobado. Un nodo puede enviar y recibir tráfico de unidifusión hacia y desde una dirección preferida. El período de tiempo que una dirección puede permanecer en estado tentativo o preferido se incluye en el mensaje de anuncio de router.

- Desaconsejado

  Estado de una dirección que sigue siendo válida, pero no se recomienda su uso para nuevas comunicaciones. Las sesiones de comunicación existentes pueden continuar utilizando una dirección desaconsejada. Un nodo puede enviar y recibir tráfico de unidifusión hacia y desde una dirección desaconsejada.

- Válido

  Estado de una dirección desde la que se puede enviar y recibir tráfico de unidifusión. El estado válido abarca los estados preferido y desaconsejado. La cantidad de tiempo que una dirección puede permanecer en estado tentativo o válido se incluye en el mensaje de anuncio de router. La duración válida debe ser mayor o igual que la duración preferida.

- No válido

  Estado de una dirección con la que un nodo ya no puede enviar o recibir tráfico de unidifusión. Una dirección pasa al estado no válido cuando caduca la duración válida.

  En la siguiente ilustración se muestra la relación entre los estados de una dirección configurada automáticamente, la duración preferida y la duración válida.

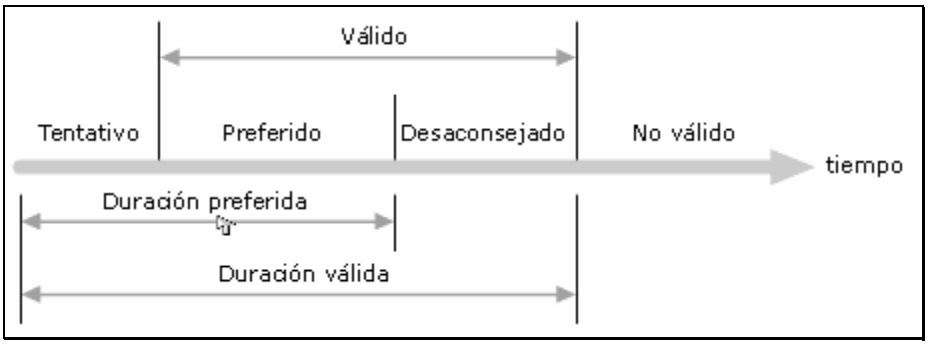


Figura 7

Tipos de configuración automática

Existen tres tipos de configuración automática:

1. Sin estado

   La configuración de direcciones se basa en la recepción de mensajes de anuncio de enrutador. Estos mensajes contienen prefijos de direcciones sin estado y requieren que los hosts no utilicen un protocolo de configuración de direcciones con estado.

2. Con estado

   La configuración se basa en el uso de un protocolo de configuración de direcciones con estado, como DHCPv6, para obtener las direcciones y otras opciones de configuración. Un host utiliza la configuración de direcciones con estado cuando recibe mensajes de anuncio de router que no incluyen prefijos de direcciones y requieren que el host utilice un protocolo de configuración de direcciones con estado. Un host también utilizará un protocolo de configuración de direcciones con estado cuando no haya routers presentes en el vínculo local.

3. Ambos

   La configuración se basa en la recepción de mensajes de anuncio de router. Estos mensajes contienen prefijos de direcciones sin estado y requieren que los hosts utilicen un protocolo de configuración de direcciones con estado.

   En todos los tipos de configuración automática siempre se configura una dirección local del vínculo.

   Proceso de configuración automática

   El proceso de configuración automática de direcciones en un nodo IPv6 se produce de la manera siguiente:

1. Se deriva una dirección local del vínculo tentativa, basada en el prefijo local del vínculo de FE80::/64 y el identificador de interfaz de 64 bits.
2. Se lleva a cabo la detección de direcciones duplicadas para comprobar la exclusividad de la dirección local del vínculo tentativa.
3. Si se produce un error en la detección de direcciones duplicadas, se debe realizar la configuración manual en el nodo.
4. Si la detección de direcciones duplicadas tiene éxito, la dirección local del vínculo tentativa se considera única y válida. La dirección local del vínculo se inicializa para la interfaz. La dirección correspondiente de nivel de vínculo Multicast para el nodo solicitado se registra en el adaptador de red.

   En un host IPv6, la configuración automática de direcciones continúa de la forma siguiente:

1. El host envía un mensaje de solicitud de router.

2. Si no se reciben mensajes de anuncio de router, el host utilizará un protocolo de configuración de direcciones con estado para obtener las direcciones y otros parámetros de configuración.
3. Si se recibe un mensaje de anuncio de router, se establece en el host la información de configuración incluida en el mensaje.
4. En cada uno de los prefijos de direcciones de configuración automática con estado que se incluyen:
    - El prefijo de dirección y el identificador de interfaz de 64 bits correspondiente, se utilizan para derivar una dirección tentativa.
    - Se utiliza la detección de direcciones duplicadas para comprobar la exclusividad de la dirección tentativa.
        - Si la dirección tentativa está en uso, no se configura para la interfaz.
        - Si la dirección tentativa no está en uso, se inicializa. Esto incluye la configuración de la duración válida y la duración preferida en función de la información contenida en el mensaje de anuncio de router.
5. Si se especifica en el mensaje de anuncio de router, el host utilizará un protocolo de configuración de direcciones con estado para obtener direcciones adicionales o parámetros de configuración.

Ver [EIR-01] - [RFC2460]

CAPITULO 3 – Soporte para Aplicaciones Ipv6

5. Introducción

La interfase de programación de aplicaciones (API) por defecto para las aplicaciones del protocolo TCP/IP es la interfase socket, esta API es usada hoy en día en la mayoría de los Sistemas Operativos, puesto que siempre ha gozado de una gran portabilidad por lo que es necesario que esa portabilidad se extienda al nuevo protocolo Internet (IPv6) de tal manera que las aplicaciones que hoy funcionan eficientemente en IPv4 también lo hagan en IPv6.

En el tiempo en que se desarrolló la interfase socket en la década de los ochenta no se pensó en la problemática que hoy en día hace necesario un nuevo protocolo, es por esto, que para el desarrollo de aplicaciones o migración de aplicaciones IPv6 se requiere tener en cuenta el cambio de una serie de elementos en el API, que son principalmente una nueva estructura para almacenar las direcciones IPv6, nuevas funciones de conversión y algunas nuevas opciones para el socket. Todos estos elementos proporcionan acceso a las características básicas de IPv6 necesarias para el desarrollo de aplicaciones UDP, TCP y Multicast siendo esta última de gran interés para el desarrollo de este proyecto de tesis. Hoy en día la mayoría de los lenguajes de programación y sistemas operativos están trabajando el soporte de las nuevas características del API de programación.

Es por lo anterior que el propósito de este capítulo es realizar un análisis de los sistemas operativos y lenguajes de programación de tal manera de determinar que combinación de ellos es la más eficiente para el desarrollo del presente proyecto.

6. Análisis de Sistemas Operativos

Como se dijo anteriormente son varios los Sistemas Operativos que están trabajando en la implementación definitiva del nuevo protocolo, aquí no se pretende abarcar todos pero se analizarán los más conocidos y los que sean técnicamente posibles de instalar y probar, puesto que serán necesarios para los capítulos prácticos de este proyecto.

El análisis se efectuará en base a pruebas prácticas, experiencias personales, además de documentación y experiencias presentes en Internet.

Tomaremos como referencia la investigación realizada en la url http://www.ipv6.org/impl, en la cual proporcionan una lista de todos los sistemas operativos que trabajan en el soporte para IPv6.**Microsoft Windows**

6.1.1. Soporte.

En 1998, Microsoft publicó una primera versión de su implementación de IPv6. Esta primera versión se podía ejecutar sobre Windows NT. El nombre que recibió este software fue MSRIPv6. Existen varias versiones de este software. Este es el producto del resultado de la primera fase.

En marzo del 2000, se entrega una versión preliminar para Windows 2000, la cual ya permite comenzar a familiarizarse con los conceptos y capacidades de la nueva tecnología.

En la actualidad, Microsoft dispone del software IPv6 Technology Preview, que funciona sobre Windows 2000 SP1 y puede ser descargado desde el sitio de MSDN

http://msdn.microsoft.com/downloads/sdks/
platform/tpipv6.asp

Este programa es una pila IPv6 para Windows 2000 para desarrolladores, y constituye el segundo paso de los cuatro que constaba la estrategia de Microsoft.

Actualmente soporta gran cantidad de propiedad IPv6 entre las cuales destacan:

- Basic IPv6 header processing
- Hop-By-Hop and Destination Options headers
- Fragmentation header
- Routing header
- Neighbor Discovery
- Stateless address autoconfiguration
- ICMPv6
- Multicast Listener Discovery
- Automatic and configured tunnels
- IPv6 over IPv4
- 6to4
- UDP and TCP over IPv6
- Router functionality (static routing tables)
- IPSec authentication

También hay otras características importantes que todavía no son implementadas y que desde el punto de vista de este proyecto de tesis son muy

relevantes al momento de la elección del sistema operativo, entre las cuales se pueden mencionar:

- Dynamic Routing Protocol
- DNS transport over IPv6

En octubre de 2001, Windows XP es lanzado con una versión preliminar del Stack IPv6 además de componentes claves de sistema operativo ya compatibles con IPv6, con esto Microsoft busca que desarrolladores y clientes comiencen a trabajar con IPv6 bajo condiciones controladas y de no producción.

Ver [MSTP-01]

En noviembre de 2001, Windows Server 2003 fue lanzado con la primera edición del stack de producción y mayor cantidad de aplicaciones compatibles con IPv6.

Independiente de Microsoft hay en estos momentos entidades trabajando en el soporte para IPv6 en Windows, una de estas es Hitachi la cual tiene un producto de software llamado **Hitachi Toolnet6,** el cual proporciona conectividad IPv6 para las plataformas Windows 95/98/NT, cabe señalar que no está disponible para Windows 2000. Este software proporciona acceso tanto a redes IPv4 e IPv6 remplazando el driver de la tarjeta de red que sólo puede ser de dos tipos; una NE2000 o de la familia EtherLink III, manteniendo inalterado el Winsock del sistema operativo, para hacer la transacción de direcciones realiza NAT de las direcciones apoyándose en un DNS compatible con IPv6. La forma de implementarlo esta documentado en el RFC 2767.

Ver [TOOL - 01]

Otra entidad que trabaja en el desarrollo de una aplicación que permita el soporte IPv6 para Windows 95/98/NT es Trumpet Software Internacional, la cual ha desarrollado la versión 5.0 de su aplicación Trumpet Winsock, la cual proporciona soporte IPv6 pero no en un 100%. Dentro de las funcionalidades más importantes con que cuenta la versión 5.0c se tiene:

- Trabaja con Socket TCP, UDP y Raw.
- Gateway discovery y address autoconfiguration estan operativos solamente para interfaces Ethernet.
- DNS lookups soporta AAAA y IP6.INT

Funcionalidades no implementadas:

- Routing
- Tunnel over IPv4
- Multicast
- Scoped Address
- IPsec

Ver [TRUM – 01]

6.1.2. Estado del arte en el desarrollo de Aplicaciones Windows compatible con IPv6.

En la actualidad, Microsoft incluye algunas aplicaciones que funcionan sobre IPv6.

- Utilidades de red: ping6, tracert6 (traceroute para IPv6) y ttcp (programa para probar la conexión TCP entre dos máquinas).
- Internet Explorer.
- Los clientes de FTP y Telnet (telnet.exe y ftp.exe) son capaces de conectarse a servidores IPv6.
- Servidor Telnet: el servidor de Telnet de Microsoft permite establecer sesiones telnet con clientes IPv4 y IPv6

- Programas que utilizan RPC (Remote Procedure Calls o Llamadas de Procedimiento Remoto) y pueden ejecutarse sobre IPv6

Si bien, Microsoft se encuentra en pleno desarrollo del soporte IPv6 en sus plataformas más modernas, en Windows 2003 y Windows XP es en este último donde hay mayores resultados puesto que ya es distribuido en las versiones finales del sistema operativo el stack IPv6 permitiendo ya desarrollar aplicaciones independientes, para ello se debe contar con las siguientes herramientas de desarrollo:

- ❖ Microsoft Platform Software Development Kit (SDK), (disponible en el sitio de MSDN).
- ❖ Microsoft Visual C++® versión 6.0 o posterior.
- ❖ Protocolo IPv6 para el SO correspondiente.

En función de esto el desarrollo de aplicaciones para Windows por parte de entidades o particulares externos también se encuentra avanzado, se puede obtener una lista actualizada de las más importantes en [IEAF- 01]

Como se dijo anteriormente Windows en ninguna de sus plataformas soporta ruteo avanzado, pero existe una aplicación llamada MRT (Multi- Threaded Routing Toolkit), la cual permite utilizar BGP4+/BGP/RIPng/RIP2, la cual repara una gran falencia que tiene Windows en estos momentos.

Ver [TMTR - 01]

Las Pruebas:

Las pruebas realizadas con Windows 2000 y Windows Xp, esta documentadas en el apéndice A.

6.2. Linux

6.2.1. Soporte

En septiembre del 2000, la versión de kernel 2.6.0 es lanzada con una serie de nuevas características, entre ellas la mejora del stack para IPv6, el cual tiene la gran mayoría de las características ipv6 implementadas o en vías de implementación, durante este periodo se han efectuado muchas pruebas con la finalidad de verificar si cumple eficientemente con las especificaciones del protocolo, es así como el académico Jun Tian and Zhongcheng Li publica un paper [TNGI-01] donde realiza pruebas de desempeño e implementación de las especificaciones generales y el protocolo ICMP6 en plataformas Linux, dichas pruebas son hechas sobre la base de las especificación del [RFC2460] para la implementación IPv6 y el [RFC2464] para el Internet Control Message Protocol. Se realizaron 55 test para el caso de la especificación IPv6 y 14 test para el caso ICMPv6, con los cuales se pudo concluir que, si bien se está trabajando en la implementación de IPv6, en el sistema operativo linux todavía se pueden apreciar algunas inconsistencias en la implementación.

Proyectos IPv6 Open-Source

La comunidad Open Souce hoy día se encuentra en pleno desarrollo de compatibilidad de linux para el protocolo de la nueva generación IPv6. Entre los cuales destaca The WIDE IPv6 Working Group (IPv6 WG), proyecto que partió en Japón a principios 1995 con el propósito de desarrollar e implementar IPv6, y a fines de 1995 el WG IPv6 contaba con varias puestas en práctica, además de pruebas de interoperabilidad. Más tarde y a raíz de algunos problemas de funcionalidad en los equipos de trabajo se crearía el proyecto KAME como un sub-proyecto con el único objetivo de la puesta en práctica en base a los trabajos realizados por WG IPv6.

Otro proyecto que también trabaja en los esfuerzos por hacer de IPv6 una realidad es el proyecto TAHI, el cual parte en Octubre del 1998 en Japón con el objetivo de desarrollar y proveer tecnologías de verificación para IPv6 a través de la investigación y desarrollo de pruebas de conformidad e interoperabilidad., este proyecto entrega sus resultados de las pruebas realizadas a las diferentes implementaciones por medio de publicaciones en [TAHP-01], resultados que serán utilizados en este proyecto de Tesis.

Existe otro grupo de trabajo llamado USAGI (UniverSAl playGround para IPv6), el cual tiene como objetivo el entregar un stack de producción IPv6 de calidad para Linux y se encuentra en estrecha colaboración con el proyecto WIDE, el proyecto KAME, el proyecto TAHI y el grupo de usuarios de Linux IPv6.

El proyecto de IPv6-DRET es una puesta en práctica pública de Linux de IPv6, financiada por el DGA/DRET (agencia militar francesa de la investigación) y co-desarrollada por INRIA Sophia-Antipolis y LIP6 París. IPv6-DRET estaba basado en el núcleo 2,1 de Linux, y la meta de esta puesta en práctica era probar la calidad en relación con ciertos algoritmos del servicio (QoS), pero los trabajos están detenidos y los trabajos ya pasaron al terreno de lo anticuado.

Entre todos los proyectos descritos anteriormente, sólo dos proporcionaron trabajos bastantes avanzados para IPv6 Linux: el grupo de desarrollo del núcleo de Linux y el proyecto USAGI.

Si bien aquellos que trabajan en el desarrollo del núcleo de linux tienen líneas de trabajo en el sentido de hacer linux compatible con IPv6, los resultados basados en el proyecto TAHI demuestran que no es la mejor implementación que existe, esto se debe a los atrasos que hay en los trabajos puesto que el desarrollador del código

para IPv6 linux, Pedro Roque a dejado la comunidad, siendo retomada por Alexey Kuznetsov y se espera que ya pronto hayan novedades.

En la siguiente figura se muestra un resumen del análisis comparativo entre la implementación USAGI y la implementación Linux

	Pass		Fail		Inconclusive	
	USAGI	Linux	USAGI	Linux	USAGI	Linux
Basic Specification	18	8	25	33	0	2
Address Autoconfiguration	5	5	2	2	0	0
Redirect	21	21	1	1	3	3
Neighbor Discovery	28	28	30	31	9	8
Total	72	62	58	67	12	13

Figura 8

En base a ello se concluye que USAGI Stack es la más madura implementación existente hasta el momento, además la que presenta mayor cantidad de funcionalidades y cumplimiento de las especificaciones del protocolo.

Ver [LIDP-01], KAME-01] y [TAHP-01]

CAPITULO 4 – Desarrollo de Aplicaciones IPv6

7. Introducción

La transición entre la Internet hoy IPv4 y la futura basada en IPv6, será una largo proceso durante el cual ambos protocolos deberán coexistir. El paso de IPv4 a IPv6 no es directo, es por esto que para simplificar dicho proceso se requiere desarrollar un método de transición de aplicaciones, hoy en día las formas de transición de la red están ampliamente difundidas, sin embargo las aplicaciones deben tener también métodos de transición para completar el proceso y poder hacer de IPv6 una realidad.

Las aplicaciones son escritas asumiendo que la red en que trabaja está basada en IPv4, sólo muy recientemente IPv6 ha sido tomado en cuenta, el objetivo de este capítulo es entregar una visión general de los distintos métodos que se pueden utilizar para el proceso de migración de aplicaciones al Protocolo IPv6 y el desarrollo de nuevas aplicaciones con soporte para dicho protocolo.

A continuación se analizará en que condiciones es posible la transición a IPv6 sin cambiar la aplicación, lo cual es válido en el caso de no contar con el código fuente de la aplicación.

7.1. Transición a IPv6 sin cambiar las aplicaciones.

Muchas metodologías se han estudiado para apoyar la transición a IPv6 en función de la arquitectura de red inicial, para hacer disponible la red IPv6 para los usuarios es necesario proporcionarles un gran número de aplicaciones, si bién las aplicaciones están desarrolladas para redes basadas en IPv4, no se puede esperar

que todas las aplicaciones estén operativas bajo IPv6 para poder iniciar el proceso de transición de la red.

El primer acercamiento durante las etapas iniciales de la transición es el uso de aplicaciones IPv4 sobre el nuevo ambiente IPv6, desafortunadamente esta no es una tarea fácil, puesto que un nodo IPv6 puro sólo entrega la interfase del socket IPv6 y las antiguas aplicaciones sólo tiene la interfase del socket IPv4. Si bien hay muchas similitudes entre ambas interfaces las diferencias que existen son de tal importancia que impiden el correcto funcionamiento de la aplicación:

El servidor DNS y el manejo de las direcciones IP,

Constantes de comunicación y estructuras de datos,

Nombres de las funciones y parámetros

Por tanto, en un nodo IPv6 puro debe estar disponible la interfase del socket IPv4 para poder utilizar las aplicaciones que no tengan soporte para esta última versión del protocolo. Puesto que la red con la cual funciona el nodo es IPv6. Para que ambos protocolos coexistan en una misma red se requiere proporcionar un sistema de traducción que permita la comunicación transparente.

El sistema de traducción se encarga de transformar la información que viajan desde la capa de aplicación a la capa de enlace, traduciendo todo lo relacionado con IPv4 en IPv6 y viceversa, cuando una aplicación IPv4 requiere comunicarse con otro nodo el traductor se encarga de hacer el cambio de la dirección de destino y origen. Para la recepción el proceso es similar todos los paquetes entrantes son examinados, traducidos y entregados a las capas superiores en formato IPv4. La traducción se puede realizar a nivel de enlace, a nivel de socket o a nivel de aplicación.

Traductor nivel de Enlace: la traducción tiene lugar entre la capa IP y los módulos de los driver de red (capa de enlace). En este nivel el traductor intercepta todos los paquetes de redes entrantes y salientes.

Traductor a nivel de socket: en este caso el traductor está a nivel de las interfaces de programación, así la traducción se simplifica puesto que no se traducen los encabezados como en el caso anterior. Aquí el traductor intercepta todas las llamadas al socket IPv4 y las traduce en función del socket IPv6.

Finalmente, algunas veces las aplicaciones utilizan librería de comunicación a nivel de usuarios, por ejemplo cuando se utiliza un prototipo o un lenguaje de programación interpretado. En tal caso, esto es posible introduciendo el traductor a nivel de aplicación. El resto de módulos del API de comunicación permanecen, pero este es reescrito en orden a trasformar las conexiones IPv4 en IPv6.

Desafortunadamente, hay aplicaciones que no tienen una modularización limpia o tienen dependencia de la dirección, en estos casos el código de la aplicación debe ser revisado para ser integrado al nuevo ambiente IPv6.

7.2. El nuevo escenario para las aplicaciones

El más importante requerimiento en la transición hacia IPv6 es que existan servicios que trabajen en el nuevo ambiente y continúen operando con nodos IPv4. El más simple acercamiento a introducir IPv6 sin cambiar las aplicaciones es usar el stack dual; soporte para IPv4 e IPv6 simultáneamente.

Para las redes que actualmente están operando la mejor solución es mantener el stack IPv4 e introducir el stack IPv6 en paralelo mientras dura el periodo de transición, una vez terminado el periodo de transición se removerán el stack IPv4 obteniendo así una red IPv6 pura.

Las organizaciones que realicen la transición desde IPv4 a IPv6 seguirán más o menos los mismos pasos. Cada etapa define un escenario de transición y para ello los administradores de sistemas deberán proveer aplicaciones que trabajen en todos ellos. Podemos considerar varios escenarios en el proceso de portar aplicaciones usando Stack dual:

Inicio: aplicaciones diseñadas para IPv4 y las redes solo trabajan con IPv4.

Paso I: las redes permanecen inalterables. Algunas aplicaciones IPv6 están disponibles pero la comunicación IPv6 es solamente posible sobre túneles IPv4.

Paso II: IPv6 está disponible a nivel de red a menos que algunas aplicaciones permanezcan inalterables y sólo soporten IPv4.

Paso III: las aplicaciones están listas para trabajar simultáneamente en IPv4 e IPv6

Paso IV: la red IPv4 es removida. La comunicación con IPv4 es solo posible por medio de túneles sobre la red IPv6.

Fin: todas las redes y aplicaciones IPv4 usan IPv6.

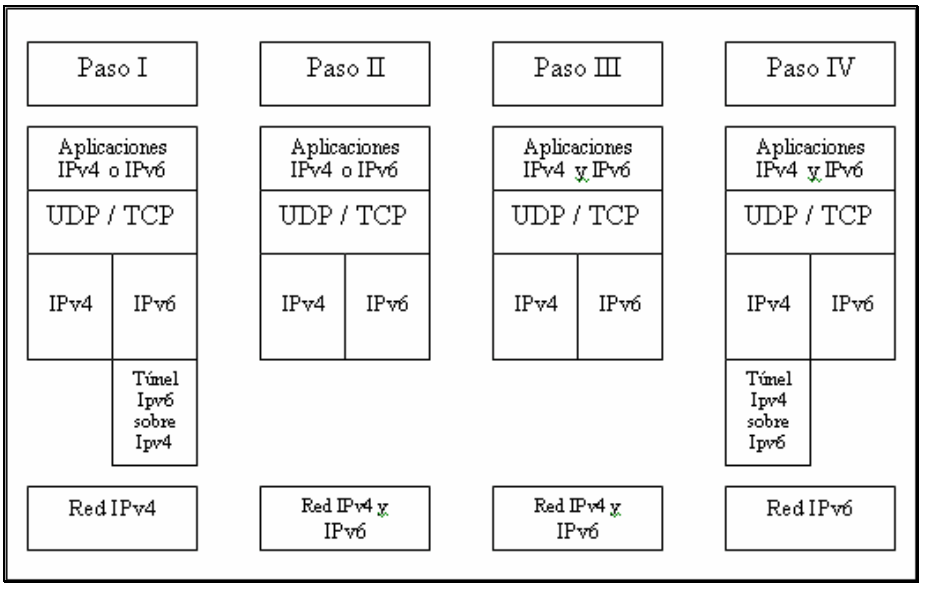


Figura 9

El proceso de modificación y desarrollo de aplicaciones para IPv6 se encuentra en el paso II y III, y este proyecto de tesis busca servir de apoyo en dichas etapas de la transición.

Existen tres posibles formas de efectuar la transición, la primera se basa en mantener dos versiones de las aplicaciones, la segunda consiste en contar con una sola versión dual y la tercera en desarrollar aplicaciones nuevas con soporte para ambos protocolos de tal manera de tener un completo set de aplicaciones que remplacen las aplicaciones IPv4.

8. Portando Aplicaciones

Muchas de las aplicaciones son escritas asumiendo IPv4. En general estas pueden ser convertidas sin demasiado esfuerzo, puesto que la mayoría de los cambios pueden ser hechos automáticamente y existen algunos script para hacerlo. Sin embargo, hay aplicaciones que hacen un uso especial de IPv4 o incluyen características avanzadas como Multicast, raw sockets o algunas otras opciones IP, en estos casos hay que hacer una exhaustiva revisión del código.

Además, hay aplicaciones que no solamente son afectadas por la modificación de las llamadas a sus funciones sino que también por la lógica que hay detrás de ello. Tales códigos son muy difíciles de modificarse y no puede ser portado automáticamente. Estas aplicaciones pueden ser dañadas si es que el desarrollador no considera la lógica de la aplicación durante el proceso.

En el caso más general, portar una aplicación al nuevo ambiente IPv6 requiere examinar algunos ítems en el código fuente de la aplicación:

Almacén de Información de red: Estructuras de datos.

Funciones de Resolución y conversión de direcciones.

Comunicación entre las funciones del API y las constantes predefinidas.

Muchas aplicaciones utilizan la dirección IP para referirse a un nodo remoto, esto no es recomendado puesto que las direcciones IPv6 cambian durante el tiempo (renumeración). Las aplicaciones deben usar un identificador estable para los demás nodos, por ejemplo el nombre de host. Si las aplicaciones utilizan la dirección IP deben hacer un mapeo entre el nombre del host y la dirección IP para resolver el problema.

8.1. Analizando Aplicaciones Existentes

Cuando se porten aplicaciones IPv4 a IPv6, hay diferentes opciones para realizar esta tarea. Si las aplicaciones utilizan sólo características básicas de comunicación, los desarrolladores sólo tienen que identificar los módulos de comunicación de la aplicación y cambiar algunas funciones para utilizar el nuevo API IPv6. Sin embargo, si la aplicación tiene características avanzadas, como lo son calidad de servicio, movilidad, multicast etc, un rediseño de algunas partes de las aplicaciones debe ser considerado. En este caso el proceso de portar las aplicaciones requiere de un mayor grado de conocimiento de la arquitectura de las aplicaciones y en muchos casos el proceso es similar al rediseño de la aplicación sobre IPv6.

Proveer características básicas de IPv6 a aplicaciones existentes es una tarea fácil si la arquitectura de la aplicación esta bien diseñada, con módulos aislados, y el módulo de transporte sea fácil de identificar. En el otro caso, si la aplicación no tiene módulos aislados, la adaptación a al nuevo API requiere muchos más cambios en diferentes segmentos del código fuente, haciendo más difícil la mantención y la reutilización del mismo.

El módulo de transporte es el encargado de implementar el paradigma de comunicación utilizado por la aplicación y debe ser adaptado al nuevo API de comunicación IPv6. Puede ocurrir que otros módulos o partes diferentes de la aplicación usen el modulo de trasporte, incluyendo dependencias de la dirección IP por tanto debe ser revisado:

Estructuras de la direcciones IP.

Uso de direcciones Especiales.

Selección de dirección IP Local.

Application Data Unit (ADU) fragmentation.

Registros de sistemas basados en la dirección IP.

En las siguientes secciones serán estudiadas algunas modificaciones que deben ser hechas a los módulos de comunicación y en otros módulos que incluyen dependencia en la dirección IP.

8.2. El módulo de Transporte

Cuando se portan aplicaciones a IPv6, el primer módulo a ser revisado es el relacionado con el manejo de los canales de comunicación remoto, llamado módulo de transporte, este módulo es dependiente de la versión de IP utilizada por la aplicación.

El modulo de transporte de la aplicación establece los canales de comunicación para que una aplicación específica pueda transmitir información a sitios remotos. Este modulo hace las aplicaciones de red independientes, entregando un modulo de comunicación genérico que permite intercambiar información entre usuarios remotos.

Durante la fase gradual de transición desde IPv4 a IPv6, algunas aplicaciones deberán trabajar sobre redes IPv4 e IPv6. por lo tanto la portabilidad es una de las principales características de la aplicación, las cuales deben trabajar en ambos ambientes, uno de los principales caminos para hacer aplicaciones independientes del protocolo usado, es desarrollar una librería genérica de comunicación, la cual oculta las dependencias del protocolo y hace que el modulo de transporte de la aplicación sea mas simple. El uso de módulo permite la reutilización y la fácil mantención.

Una librería genérica de comunicación permite a las aplicaciones ser independiente del nivel más bajo del protocolo, y entrega una interfase de comunicación común a cada aplicación, las cuales pueden tener acceso a diferentes protocolos desde la funciones de la librería, independientes del la versión del protocolo IP.

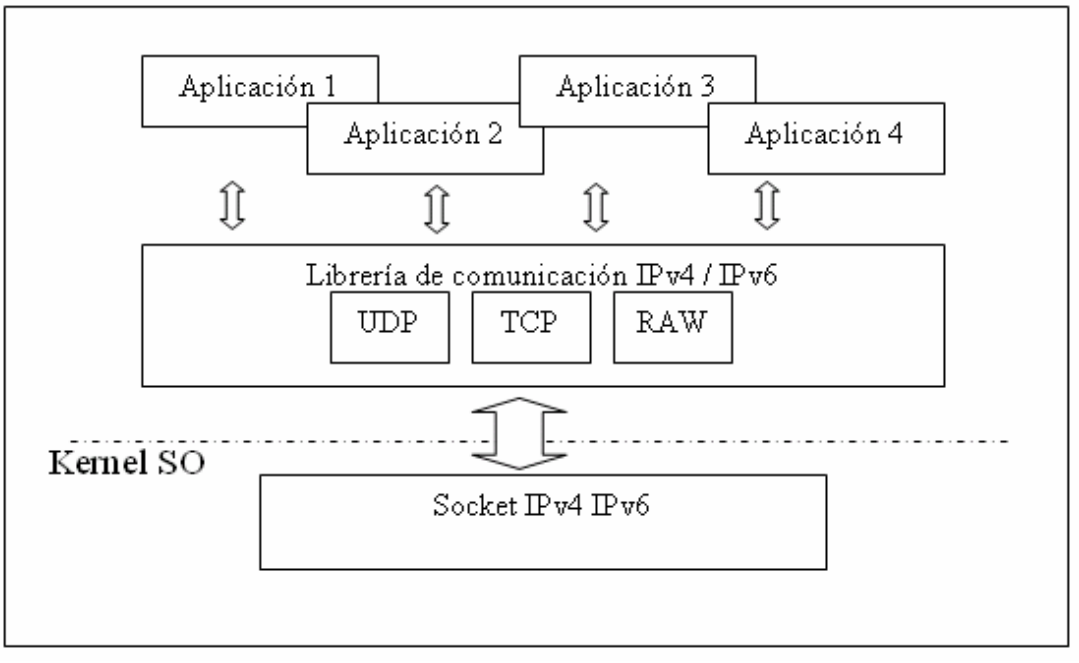


Figura 10

8.3. Otros módulos con dependencia de la dirección IP.

El módulo de transporte de una aplicación esta directamente relacionado con el establecimiento de comunicación y la transmisión de información por tanto depende de la dirección IP. Dentro de una aplicación pueden existir otros módulos que igualmente dependen de la dirección IP.

Muchas aplicaciones requieren una dirección IP como argumento para establecer una nueva conexión, por ejemplo cuando se usa el modelo cliente/servidor el cliente requiere conocer la dirección IP o el nombre del host donde se está ejecutando el servidor.

El uso de Fully Qualified Domain Name (FQDN) en lugar de la dirección IP es preferible puesto que la dirección del nodo puede cambiar en el tiempo y este proceso es deseable que sea transparente para la aplicación. Las aplicaciones pueden usar el FQDN, delegando la obtención de la dirección IP al sistema de resolución de nombres, el cual retorna la dirección IP asociada al nombre del nodo.

Desde el punto de vista de las aplicaciones, la resolución del nombre es un proceso independiente. Las aplicaciones llaman funciones desde la librería de sistema, conocidas como “resolver” el cual es enlazado a las aplicaciones cuando estas se construyen, los programadores deben cambiar el uso de funciones de resolución IPv4 por las correspondientes IPv6.

Comúnmente las aplicaciones no requieren conocer la versión IP que se esta utilizando, por lo tanto las aplicaciones sólo tratan de establecer comunicación usando las direcciones retornadas por el resolver, sin embargo pueden tener un diferente comportamiento dependiendo del tipo de dirección. Las que pueden ser IPv4, IPv6, IPv4 compatible IPv6.

Ver [DASI-01]

Otro problema que se debe tener en cuenta es el uso de la nomenclatura de las constantes o identificadores de direcciones especiales. A continuación se presenta una tabla con algunos ejemplos

Constante (IPv4 / IPv6)	Direccion IPv4	Direccion IPv6
INADDR_ANY / IN6ADDR_ANY_INIT	0.0.0.0	::
INADDR_LOOPBACK / IN6ADDR_LOOPBACK_INIT	127.0.0.1	::1
INADDR_BROADCAST	255.255.255.255	No existe

Tabla 4

8.4. Selección de la dirección IP

IPv6 permite muchas direcciones IP por cada interfase de red con diferente alcance (link-local, site-local o global). Por lo tanto, debe existir un mecanismo de selección de direcciones.

Normalmente, las funciones de resolución de nombres retornan una lista de direcciones validas para un FQDN específico. Las aplicaciones tienen que iterar sobre esta lista de direcciones para seleccionar aquella que será usada en la configuración de canal de comunicación. La selección de la dirección de origen es una de las operaciones más críticas, puesto que entrega información al receptor sobre la dirección donde tiene que enviar la respuesta. Si la selección no es apropiada el camino de regreso será distinto al camino de envio, con lo cual la respuesta puede perderse y la comunicación entre las aplicaciones puede fallar.

Cuando se elige la dirección de origen, algunas aplicaciones no especifican la dirección sino que dejan que el kernel del sistema operativo realice la selección, denominado selección de dirección por defecto. En el caso de la elección de dirección de destino, algunos criterios se podrían utilizar para preferir una dirección basada en los valores del par origen/destino. La dirección por defecto es relacionada por un algoritmo que retorna la dirección preferida a partir de un conjunto de candidatas,

basada en la política de la mejor elección, los administradores pueden configurar el mecanismo para sustituir el comportamiento por defecto. Existen varios trabajos en este sentido [SOD-01].

9. Interoperabilidad IPv4/IPv6

Cuando un nodo de la red desea comunicarse a otro, este consulta el sistema de nombres por la dirección IP, si la respuesta es una dirección IPv4, este asume que el path a través de Internet es también IPv4 y que este nodo es capas de recibir conexiones IPv4. Lo mismo sucede cuando la respuesta es una dirección IPv6. Si ambos nodos; origen y destino tienen stack dual, la comunicación usa el tipo de dirección retornada por el servicio de nombres.

A continuación se estudiara la interoperabilidad usando el modelo cliente-servidor entre los nodos con IPv4 puro, IPv6 puro y nodos con stack dual.

9.1. Cliente IPv6/IPv4 conectado a un servidor IPv4 Puro

En este caso existe una aplicación servidor IPv4 corriendo en un nodo que solamente entiende IPv4 y analizaremos todas las posibles conexiones desde un cliente.

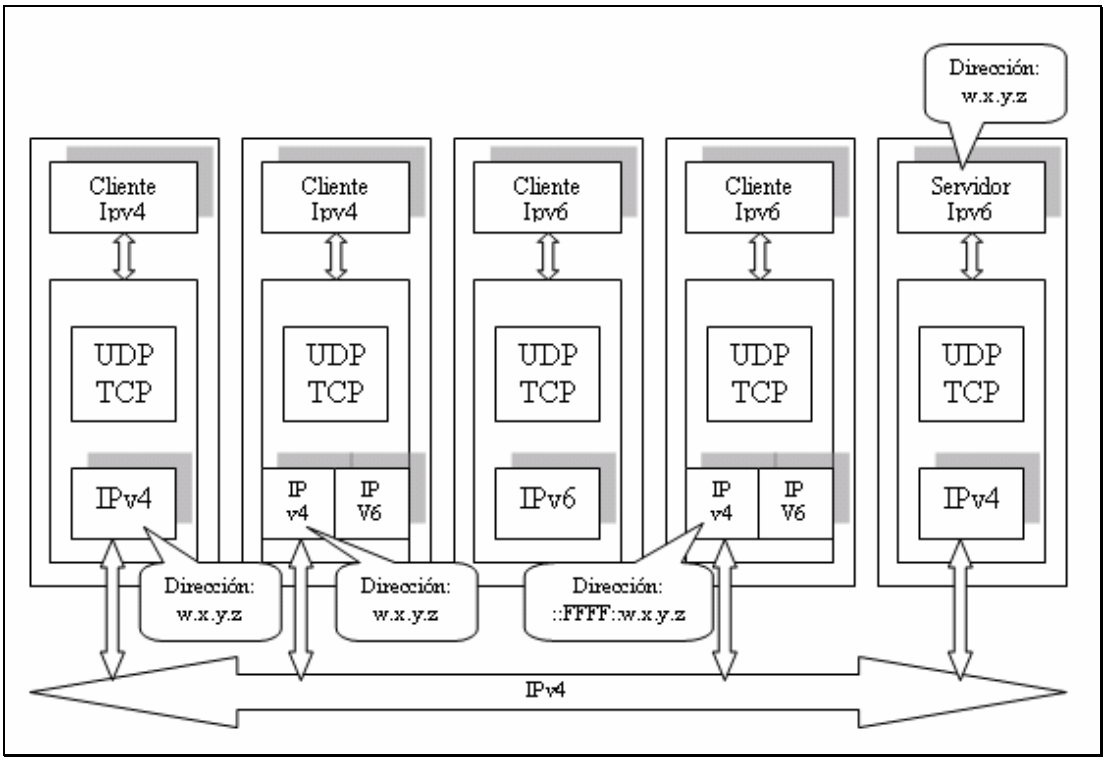


Figura 11

En el primer caso cuando cliente esta ejecutándose sobre un nodo que sólo se comunica por IPv4 la comunicación se efectúa normalmente.

El segundo caso, cuando se tiene el cliente IPv4 ejecutándose sobre un nodo con stack dual, el cual consulta al resolver la dirección IP del servidor, este al responder una dirección IPv4 hace que el cliente utilice el stack dual en la versión IPv4, y la comunicación se efectúa como si el cliente se ejecutara sobre un nodo IPv4 puro.

El tercer caso cuando se tiene un cliente IPv6 ejecutándose sobre un nodo con IPv6, la comunicación no se puede realizar pues el nodo cliente no puede intercambiar paquetes con el servidor, protocolos incompatibles.

En el cuarto caso tenemos un cliente IPv6 corriendo sobre un nodo con stack dual, cuando el cliente quiere comunicarse con el servidor le consulta al resolver y como el servidor está sobre un nodo IPv4, este le contesta con una dirección IPv4 mapeada como dirección IPv6, el cliente usará esta dirección para la conexión, entonces el cliente trabajará como si estuviera conectado a un nodo IPv6 y el servidor trabajará como si estuviera conectado con un cliente IPv4, en este caso el problema es resuelto por medio del stack dual.

9.2. Cliente IPv6/IPv4 conectado a un servidor IPv6 que se ejecuta en un nodo IPv6.

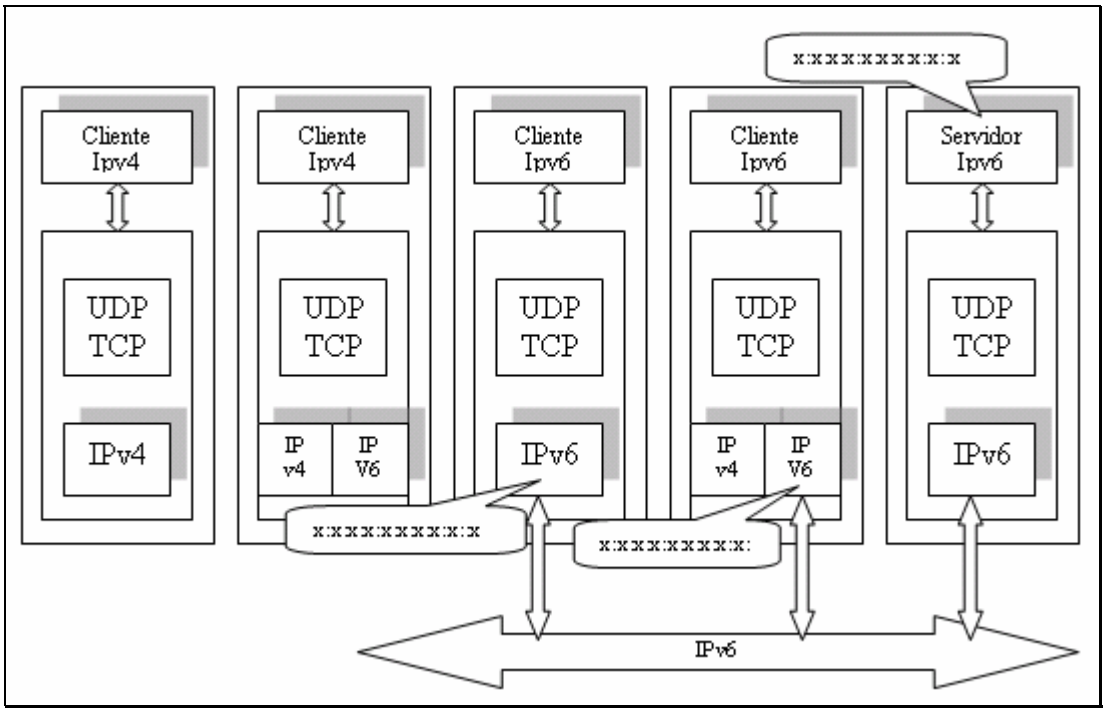


Figura 12

Con esta configuración sólo podemos comunicar aquellos clientes que corren ya sea sobre un nodo IPv6 puro o nodos que implementan stack dual y el programa cliente sea IPv6.

9.3. Clientes IPv6/IPv4 conectado a un servidor IPv4 montado en un nodo con stack dual.

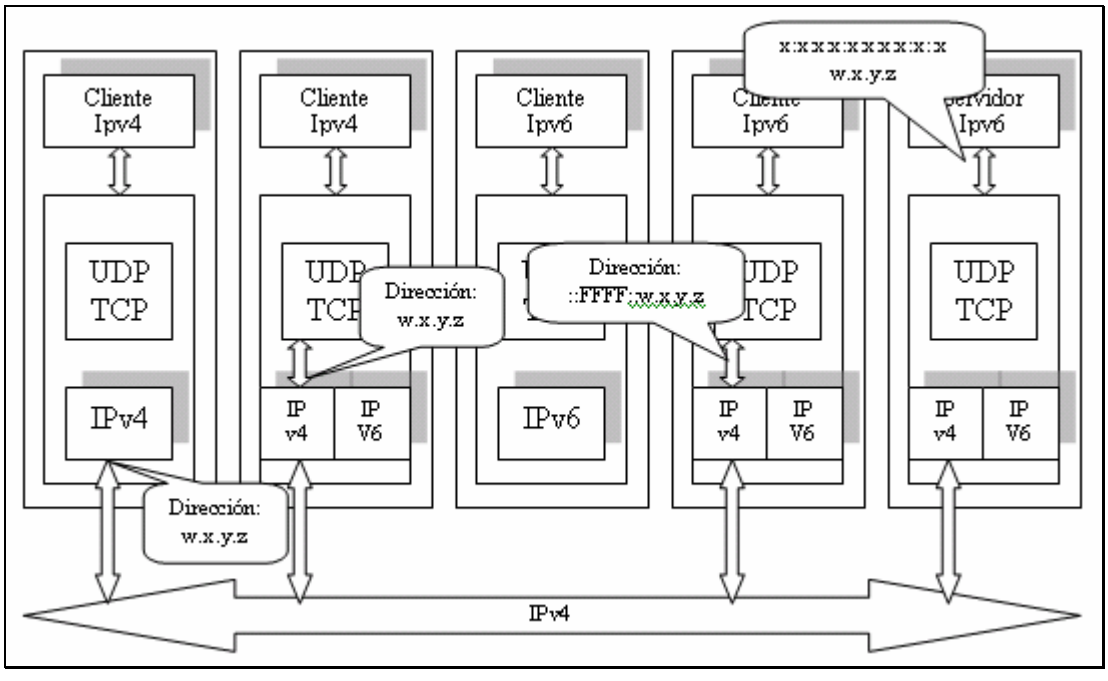


Figura 13

Cuando el cliente es IPv4, el mecanismo es similar al descrito en la sección 10.1 cuando el servidor es IPv4 se ejecuta sobre un nodo que sólo implementa IPv4.

Cuando se tiene un cliente IPv6 en un nodo que sólo tiene el protocolo IPv6 es imposible la conexión a un servidor IPv4, aunque los nodos puedan comunicarse entre ellos utilizando el protocolo IPv6.

En el caso en que el cliente sea IPv6 y se ejecute en un nodo que implemente stack dual puede conectarse a un servidor IPv4 usando las redes IPv4 el cliente IPv6 consulta al resolver por la dirección IP del servidor, el “resolver” retorna una dirección IPv6 mapeada como dirección IPv6 al cliente. El cliente trabaja como si estuviera

conectado a un servidor IPv6 y el servidor IPv4 trabaja como si estuviera conectado con un cliente IPv4.

9.4. Clientes IPv6/IPv4 conectados a un servidor IPv6 ejecutándose en un nodo con stack dual

Los clientes IPv4 se conectan al servidor mediante paquetes IPv4, en el nodo servidor las direcciones IPv4 son mapeadas de tal manera que el la aplicación servidor se comporta como si estuviera enlazada a un cliente IPv4.

Para los clientes IPv6 la comunicación entre los nodos se realiza por medio de IPv6.

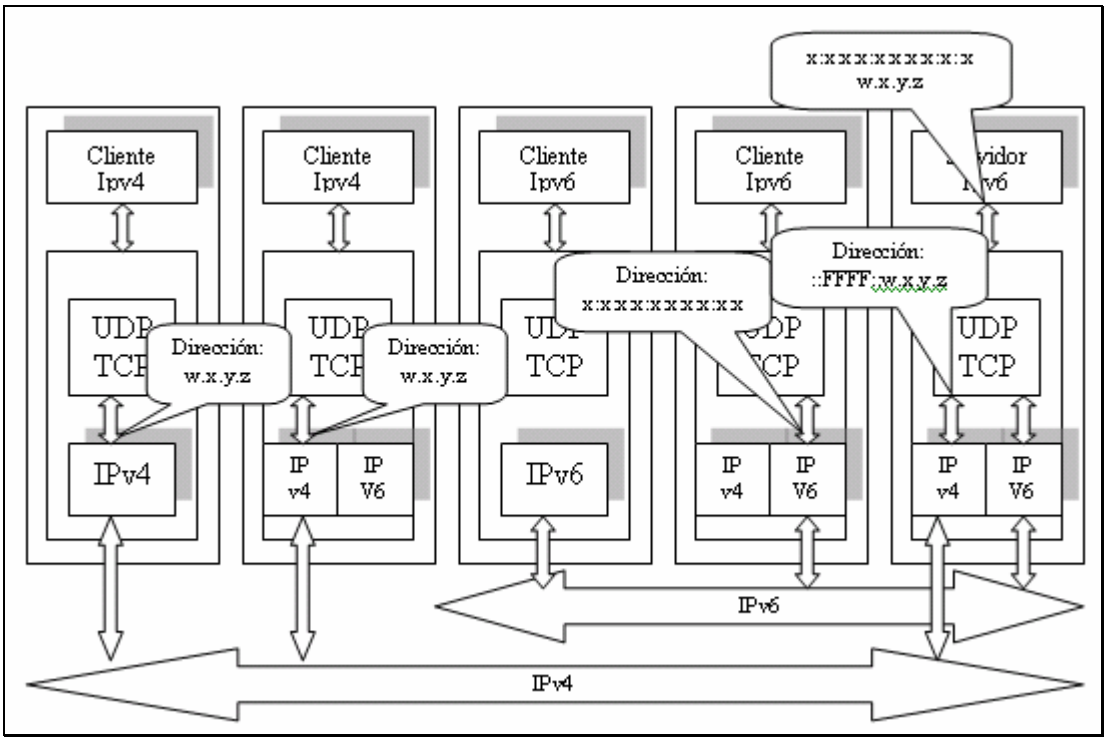


Figura 14

La siguiente tabla muestra un resumen de las combinaciones de conexión entre el modelo cliente-servidor montado sobre diferentes tipos de nodos. Las

combinaciones marcadas con una X denotan la imposibilidad de comunicación entre el tipo de nodos con el tipo de aplicaciones correspondientes.

		IPv4 server application		IPv6 server application	
		IPv4 node	Dual-stack	IPv6 node	Dual-stack
IPv4 client	IPv4 node	IPv4	IPv4	X	IPv4
	Dual-stack	IPv4	IPv4	X	IPv4
IPv6 client	IPv6 node	X	X	IPv6	IPv6
	Dual-stack	IPv4	IPv4 / X	IPv6	IPv6

Tabla 5

A raíz de esto podemos distinguir cuatro tipos de aplicaciones:

IPv4-only: Una aplicación que no puede manejar las direcciones IPv6, es decir, no puede comunicarse con los nodos que no tienen una dirección IPv4.

IPv6-aware: Una Aplicación que puede comunicarse con los nodos que no tienen direcciones IPv4, es decir, puede manejar las direcciones IPv6.

IPv6-enabled: Una aplicación que, además de ser IPv6-aware, se aprovecha de las características específicas del protocolo IPv6 tales como etiquetas del flujo. Las aplicaciones IPv6-enabled pueden a pesar de ello funcionar sobre el protocolo IPv4, en un modo más disgregado.

IPv6-required: son aplicaciones que requieren una característica específica de IPv6 y no pueden operar sobre las redes IPv4.

Ver [TAIP-01], [INDA-01]

Capitulo 5 – Desarrollo de aplicaciones Compatibles

10. Introducción

En la ardua tarea de implantar el nuevo protocolo IPv6, una de las tareas que hoy en día ha cobrando mayor relevancia, es la adaptación y creación de nuevas aplicaciones para las redes de la nueva generación. Existen métodos de transición de las aplicaciones como se vio en capítulos anteriores, además de métodos de desarrollo de aplicaciones. Siendo en este ultimo donde no existe mucha documentación formal, este déficit de información esta aun mas acrecentado para el desarrollo de aplicaciones multicast con soporte a IPv6, y si tomamos en cuanta el desarrollo de aplicaciones multicast genéricas es decir que soporten ambos protocolos el panorama se ve aun mas oscuro.

Es por esto que en este capitulo se analizara paso a paso el desarrollo de aplicaciones nuevas, genéricas y con soporte multicast.

11. El API del socket IPv6

En el API de programación varios cambios o extensiones son necesarios para el desarrollo de aplicaciones con soporte al nuevo protocolo, principalmente en las estructuras que comprenden el modulo de comunicaciones, las funciones y opciones que son utilizadas en la comunicación vía Socket, la documentación principal de este

tema se encuentra en el RFC-2553 [Basic Socket Interface Extensions for IPv6], además del RFC-2292 [Advanced Sockets API for IPv6].

Ver [RFC-2553], [RFC-2292]

Los cambios o extensiones a los que se hace referencia se basan principalmente en la adaptación al nuevo tamaño de las direcciones que pasa de 32 a 128 bit, puesto que las aplicaciones que están sobre IP virtualmente ellas conocen el tamaño de la dirección.

Los cambios también apuntan a nuevas características que se le agregan, como por ejemplo el control de flujo del trafico IP.

En estos cambios deben tomar en cuenta la compatibilidad hacia atrás como por ejemplo si se modifica o desarrolla una aplicación servidor debe ser capas de entregar servicio a aplicaciones tanto IPv4 como IPv6. Otro punto a tomar en cuenta es que los cambios que se deben realizar para la migración o soporte del nuevo protocolo deben ser en mínimo posible.

Tomando en cuenta los puntos anteriores, la pregunta es *¿Que es lo que se debe cambiar?*, para responder esta pregunta podemos dividir las componentes del modulo de comunicación en 5 componentes:

- Funciones principales del socket
- Estructuras de Información de Direcciones
- Funciones de traducción de nombre a dirección
- Funciones de conversión de direcciones

Las funciones principales del socket son las que manipulan las conexiones TCP y las que manipulan los paquetes UDP, es decir, son funciones independientes de estos protocolos de transporte, lo mismo sucede ahora con los protocolos de direccionamiento que debe ser administrado como parámetro a las funciones.

Como las funciones hacen un “cast” de los punteros que contiene las estructuras de direccionamiento, transformándolos a una estructura genérica "sockaddr", por lo tanto estas funciones no es necesario cambiarlas para dar soporte al nuevo protocolo, pero si, se requiere un cambio en las especificaciones de las estructuras que contienen las nuevas direcciones IPv6.

La estructura “sockaddr_in” es la que maneja los datos de las comunicaciones IPv4, esta estructura de datos tiene 8 octetos de espacio no utilizado, que se podría utilizar para dar almacenar las direcciones IPv6, desafortunadamente la estructura “sockaddr_in” no cuenta con el tamaño suficiente para dar cabida a las direcciones IPv6 y el resto de información que se requiere almacenar (familia de direcciones, puerto, etc), es por esto que una nueva estructura de direcciones debe ser definida para IPv6.

Como se vio en capítulos anteriores las direcciones IPv6 tienen ámbito o alcance el cual puede ser enlace-local, sitio, organización, global, o algún otro que aun no esta definido. Para aquellas aplicaciones que requieren poder identificar el alcance de las direcciones se requiere que la estructura IPv6 proporcione los medios para poder hacer esta identificación.

Las funciones de traducción de nombre a direcciones que actualmente existen son gethostbyname() y gethostbyaddr(), las que se mantienen. Se crean nuevas para dar soporte a las dos versiones del protocolo IP, las cuales son independiente de la versión del protocolo.

Las funciones de conversión de direcciones inet_ntoa() e inet_addr() convierten las direcciones IPv6 de su formato binario al formato imprimible, estas dos funciones son específicamente implementadas para direcciones de 32 bits, por tanto para IPv6 se diseñan dos funcione análogas que trabajan tanto con IPv4 como con IPv6.

Otro punto que se debe tener en cuenta es el soporte a algunas nuevas características con que cuanta IPv6. Se requieren diseñar nuevas interfaces para dar soporte a la clasificación del tráfico.

Las anteriormente descritas son las principales características que deben modificarse a nivel de programación de aplicaciones con soporte para IPv6, mas adelante se describirán otras de menor importancia.

A continuación se especifican los cambios efectuados sobre la interfaces IPv6.

11.1. Familia de Direcciones IPv6 y Familia de Protocolos IPv6

Se define un nuevo nombre de familia de direcciones para IPv6, definido en <bits/socket.h>. La definición AF_INET6 distingue entre el uso de las estructuras sockaddr_in para IPv4 y la nueva estructura sockaddr_in6

IPv6	IPv4
AF_INET6	AF_INET

Se define además un nuevo nombre para la familia de protocolos de IPv6, el cual es utilizado en el primer argumento de la función socket() para indicar que el nuevo socket que se esta creando es IPv6.

IPv6	IPv4
AF_INET6	AF_INET

Como la mayoría de los otros nombres de familia del protocolos, generalmente es definido en base a la familia de direcciones para tener el mismo valor

```
#define PF_INET6        AF_INET6
```

```
/* Protocol families.  */
#define PF_UNSPEC       0       /* Unspecified.  */
#define PF_LOCAL        1       /* Local to host (pipes and file-domain).  */
#define PF_UNIX         PF_LOCAL /* Old BSD name for PF_LOCAL.  */
#define PF_FILE         PF_LOCAL /* Another non-standard name for PF_LOCAL.  */
#define PF_INET         2       /* IP protocol family.  */
#define PF_AX25         3       /* Amateur Radio AX.25.  */
#define PF_IPX          4       /* Novell Internet Protocol.  */
#define PF_APPLETALK    5       /* Appletalk DDP.  */
#define PF_NETROM       6       /* Amateur radio NetROM.  */
#define PF_BRIDGE       7       /* Multiprotocol bridge.  */
#define PF_ATMPVC       8       /* ATM PVCs.  */
#define PF_X25          9       /* Reserved for X.25 project.  */
#define PF_INET6        10      /* IP version 6.  */
#define PF_ROSE         11      /* Amateur Radio X.25 PLP.  */

                        .
                        .
                        .
/* Address families.  */
#define AF_UNSPEC       PF_UNSPEC
#define AF_INET         PF_INET
#define AF_AX25         PF_AX25
#define AF_IPX          PF_IPX
#define AF_APPLETALK    PF_APPLETALK
#define AF_NETROM       PF_NETROM
#define AF_BRIDGE       PF_BRIDGE
#define AF_ATMPVC       PF_ATMPVC
#define AF_X25          PF_X25
#define AF_INET6        PF_INET6
```

Figura 15: Definición de familias y protocolos <bits/socket.h>

11.2. Estructuras para las direcciones IPv6

Las estructuras son usadas en la programación de sockets para almacenar información sobre direcciones, y comúnmente se utiliza una estructura genérica para hacer referencia en las funciones para cualquier familia de protocolos llamada sockaddr, que según el RFC 2553 se define de la siguiente manera:

```
struct sockaddr {
     sa_family_t sa_family;        /* Address family */
     char sa_data[14];             /* protocol-specific address */
};
```

Pero en Linux esta definido de forma distinta, con parámetros para hacerla más genérica como se muestra a continuación:

```
/* Get the definition of the macro to define the common sockaddr members.  */
#include <bits/sockaddr.h>

/* Structure describing a generic socket address.  */
struct sockaddr
  {
    __SOCKADDR_COMMON (sa_);    /* Common data: address family and length.  */
    char sa_data[14];           /* Address data.  */
  };
```

Figura 16 : Especificación estructura sockaddr en <bits/socket>

```
#define __SOCKADDR_COMMON(sa_prefix) \
  sa_family_t sa_prefix##family

#define __SOCKADDR_COMMON_SIZE  (sizeof (unsigned short int))
```

Figura 17 : Especificación macro __SOCKADDR_COMMON

Aunque las funciones del API utilicen la estructura genérica, los programadores deben usar las estructuras específicas para el protocolo.

Para IPv4 la estructura se llama sockaddr_in y las especificación es la siguiente:

```
struct sockaddr_in
  {
    __SOCKADDR_COMMON (sin_);
    in_port_t sin_port;                 /* Port number.  */
    struct in_addr sin_addr;            /* Internet address.  */

    /* Pad to size of `struct sockaddr'.  */
    unsigned char sin_zero[sizeof (struct sockaddr) -
                           __SOCKADDR_COMMON_SIZE -
                           sizeof (in_port_t) -
                           sizeof (struct in_addr)];
  };
```

Figura 18: Especificación de sockaddr_in <netinet/in.h>

```
/* Internet address.  */
typedef uint32_t in_addr_t;
struct in_addr
  {
    in_addr_t s_addr;
  };
```

Figura 19: Especificación estructura que almacena la direccion IPv4 <netinet/in.h>

En IPv6 la estructura se denomina sockaddr_in6 y la especificación difiera bastante con respecto a la estructura IPv4.

```
struct sockaddr_in6 {
    sa_family_t     sin6_family;    /* AF_INET6 */
    in_port_t       sin6_port;      /* transport layer port # */
    uint32_t        sin6_flowinfo;  /* IPv6 traffic class & flow info */
    struct in6_addr sin6_addr;      /* IPv6 address */
    uint32_t        sin6_scope_id;  /* set of interfaces for a scope */
};
```

Figura 20: Especificación del RFC 2553 para sockaddr_in6 para sistemas basados en BSD4.3

```
struct sockaddr_in6
  {
    __SOCKADDR_COMMON (sin6_);
    in_port_t sin6_port;        /* Transport layer port # */
    uint32_t sin6_flowinfo;     /* IPv6 flow information */
    struct in6_addr sin6_addr;  /* IPv6 address */
    uint32_t sin6_scope_id;     /* IPv6 scope-id */
  };
```

Figura 21: Especificación en Linux de la estructura sockaddr_in6 <netinet/in.h>

```
/* IPv6 address */
struct in6_addr
  {
    union
      {
        uint8_t u6_addr8[16];
        uint16_t u6_addr16[8];
        uint32_t u6_addr32[4];
      } in6_u;
#define s6_addr                 in6_u.u6_addr8
#define s6_addr16               in6_u.u6_addr16
#define s6_addr32               in6_u.u6_addr32
  };
```

Figura 22: Especificación de la estructura que almacena la dirección IPv6

En la figura 7 se muestra la definición de in6_addr la que tiene múltiples formas de almacenar la dirección IPv6, por lo general y para estar de acuerdo a la especificación del RFC 2553, se usa el arreglo de de 16 elementos de 8 bit con el nombre s6_addr para almacenar la dirección IPv6, dejándose la posibilidad de utilizar cualquiera de los otros dos arreglos. Debe tenerse en cuanta que el almacenamiento de la dirección de hace en *network byte order.*

Linux Actualmente implementa las especificaciones de los RFCs para sistemas basados en BSD 4.3.

El campo *sin6_family* identifica la familia de la dirección.

El campo *sin6_port* es un entero de 16 bit que identifica el numero de puerto UDP o TCP, por el cual se efectuara la comunicación. Los valores en este campo son almacenados en network byte order.

El campo *sin6_flowinfo* contiene 32 bit de información y contiene dos piezas de información: clase de tráfico y etiqueta de trafico. El contenido e interpretación de la información que contiene será tratado don posterioridad en este capitulo.

El campo *sin6_addr* es una estructura de tipo in6_addr definida en la figura 8. Amacena la dirección Ipv6 de 128 bits en network byte order.

El Campo sin6_scope_id en un entero de 32 bit que identifica un el alcance con la dirección que esta en el campo *sin6_addr.* La implementación de esta correlación entre el scopeID y la dirección de la interfase no esta completamente definido.

Notar que normalmente la estructura *sockaddr_in6* es mucho mas grande que la estructura genérica *sockaddr*, pero hoy en día hay aplicaciones que utilizan los valores entregados por sizeof(struct sockaddr_in) y sizeof(struct sockaddr), debe tenerse especial cuidado con esto al momento de la migración o desarrollo de aplicaciones IPv6.

12. Portabilidad del código

Se ha agregado una nueva estructura al API la cual fue creada con la finalidad de simplificar el desarrollo de código potable a través de las múltiples familias de direcciones y plataformas.

La estructura sockaddr_in y sockaddr_in6 son implementadas para el desarrollo de aplicaciones con soporte a IPv4 e IPv6 respectivamente. En el caso de aplicaciones que se desarrollen para IPv4 usando al estructura sockaddr_in, se

pueden hacer operar en IPv6 cambiando esta estructura por sockaddr_in6. Sin embargo este proceso significa tiempo, por lo cual es preferible eliminar la dependencia de la versión del protocolo, con este objetivo es creada la estructura sockaddr_storage con espacio suficiente para almacenar todos los tipos de direcciones y especialmente alineada para soportar los cast del las funciones.

```
/* Structure large enough to hold any socket address (with the historical
   exception of AF_UNIX).  We reserve 128 bytes.  */
#if ULONG_MAX > 0xffffffff
# define __ss_aligntype __uint64_t
#else
# define __ss_aligntype __uint32_t
#endif
#define _SS_SIZE        128
#define _SS_PADSIZE     (_SS_SIZE - (2 * sizeof (__ss_aligntype)))

struct sockaddr_storage
  {
    __SOCKADDR_COMMON (ss_);    /* Address family, etc.  */
    __ss_aligntype __ss_align;  /* Force desired alignment.  */
    char __ss_padding[_SS_PADSIZE];
  };
```

Figura 23: Especificación en Linux de la estructura Sockaddr_storage <bits/socket.h>

Con esta definición un ejemplo de utilización seria el siguiente:

```
struct sockaddr_storage EstructuraGeneral;
struct sockaddr_in6 *Estructura_v6;
Estructura_v6 = (struct sockaddr_in6 *) &EstructuraGeneral;
```

12.1. Funciones del API

Como se menciono anteriormente las funciones del socket hacen un cast de las estructuras que almacenan las direcciones (sockaddr_in, sockaddr_in6, sockaddr_storage) hacia una estructura genérica, es por esto que no requieren grandes cambio, sin embargo hay alteraciones en los argumentos en las llamadas a las funciones del socket.

Continuación se describen cada una de las funciones del API con sus respectivos parámetros según el protocolo.

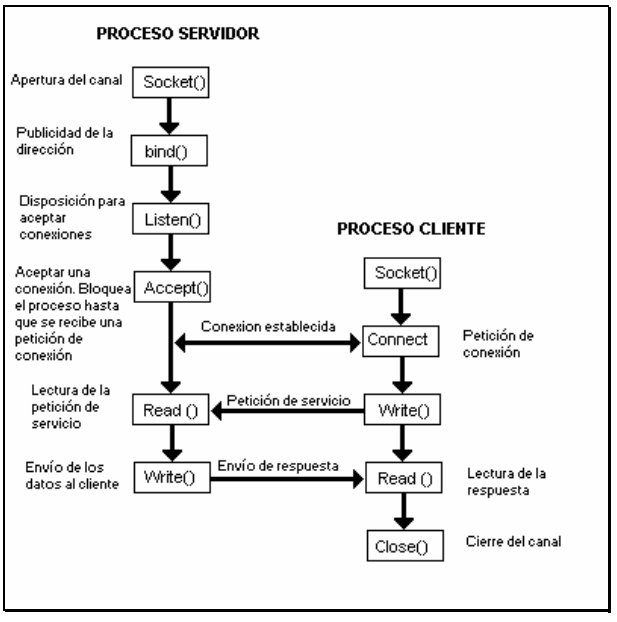


Figura 24: Procesos y orden de las funciones TCP

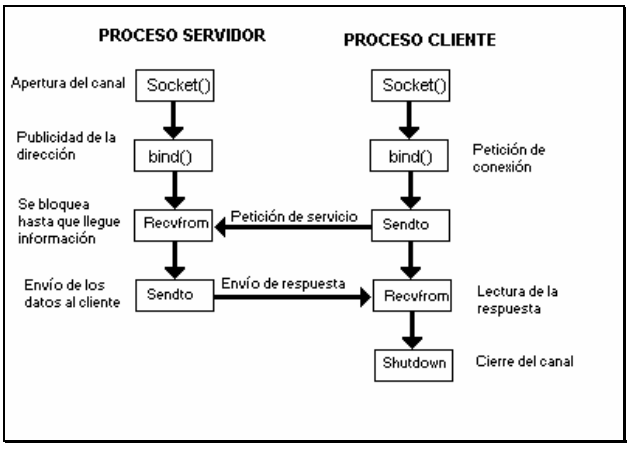


Figura 25: Procesos y orden de la funciones UDP

Antes de analizar las funciones del socket describiremos las funciones de conversión de dirección, para utilizarla en la ejemplificación de las demás funciones.

12.1.1. Funciones de Conversión

Estas funciones pasan las direcciones o puertos desde el formato binario (network byte ordered) al formato de texto (ascii).

a).- Función de conversión desde formato texto IPv4 a formato binario

inet_aton():

Sintaxis:

```
int inet_aton(const char *cp, struct in_addr *inp);
```

Descripción: Convierte la dirección de Internet cp desde la notación estándar de números y puntos (IPv4) a la representación binaria, y la guarda en la estructura a la que apunte inp.

b).- Función de conversión desde formato binario a texto IPv4

inet_ntoa()

Sintaxis:

```
char       *inet_ntoa(struct in_addr in);
```

Descripción: convierte la dirección de Internet “in” dada en orden de bytes de red a una cadena de caracteres en la notación estándar de números y puntos.

Ejemplo:

```
struct sockaddr_in Estructura_v4;
char *Direccion_str;

int puerto=13;
memset(&Estructura_v4, 0, sizeof(Estructura_v4));
Estructura_v4.sin_family = AF_INET;
Estructura_v4.sin_port = htons(puerto);

inet_aton("10.10.10.1", &(Estructura_v4.sin_addr));

Direccion_str = inet_ntoa(Estructura_v4.sin_addr);
```

Figura 26: Ejemplo transformación de direcciones IPv4

Las nuevas funciones desarrolladas para dar soporte a IPv6 fueron pensadas para efectuar las conversiones de todas las familias de direcciones, sean estas IPv4 o IPv6:

a).- Función de conversión desde formato texto IPv4/IPv6 a formato binario

inet_pton():

Sintaxis:

```
int inet_pton(int af, const char *src, void *dst);
```

Descripción: Esta función convierte el string “src” a una dirección de la familia de direcciones “af” en formato binario y la almacena en “dst”

b).- Función de conversión desde formato binario a texto IPv4/IPv6

Inet_ntop():

Sintaxis

```
const char *inet_ntop(int af, const void *src,char *dst, size_t cnt);
```

Descripción: Realiza el proceso inverso al funcion anterior, solo que ahora tiene tiene el parámetro cnt que indica el tamaño de "dst"

Ejemplo:

```
struct sockaddr_in6 Estructura_v6;
char direccion_str_v6[INET6_ADDRSTRLEN];
int puerto=13;
socklen_t addrlen = sizeof(Estructura_v6);
memset(&Estructura_v6, 0, addrlen);
Estructura_v6.sin6_family = AF_INET6;
Estructura_v6.sin6_port = htons(puerto);

inet_pton(AF_INET6, "3ffe:8040::1",&(Estructura_v6.sin6_addr));

inet_ntop(AF_INET6, &Estructura_v6.sin6_addr,
                        direccion_str_v6,addrlen);
```

Figura 27: Ejemplo transformación de direcciones IPv4/IPv6

Función Socket():

La diferencia de crear un socket para IPv4 o uno para IPv6 esta en el argumento que indica la familia del protocolo.

```
socket(PF_INET, SOCK_STREAM, 0);  /* TCP socket */
socket(PF_INET, SOCK_DGRAM,  0);  /* UDP socket */
```

Figura 28 : Creación de un socket IPv4

```
socket(PF_INET6, SOCK_STREAM, 0);  /* TCP socket */
socket(PF_INET6, SOCK_DGRAM,  0);  /* UDP socket */
```

Figura 29: Creación de un socket IPv6

```
socket(PF_UNSPEC, SOCK_STREAM, 0);  /* TCP socket */
socket(PF_UNSPEC, SOCK_DGRAM,  0);  /* UDP socket */
```

Figura 30: Creación de un socket sin especificar familia de protocolo

Función bind():

La llamada a esta función cambia en el tipo de estructura según el protocolo que se este usando

Ejemplos:

```
struct sockaddr_in Estructura_v4;
char *Direccion_str;
int sockfd
socklen_t addrlen = sizeof(Estructura_v4);
int puerto=13;
memset(&Estructura_v4, 0, addrlen);
Estructura_v4.sin_family = AF_INET;
Estructura_v4.sin_port = htons(puerto);
inet_aton("10.10.10.1", &(Estructura_v4.sin_addr));
Direccion_str = inet_ntoa(Estructura_v4.sin_addr);
sockfd=socket(PF_INET, SOCK_STREAM, 0);

bind(sockfd,(struct sockaddr *)&Estructura_v4, addrlen);
```

Figura 31: Ejemplo uso función bind() en IPv4

```
struct sockaddr_in6 Estructura_v6;
char direccion_str_v6[INET6_ADDRSTRLEN];
int puerto=13;
socklen_t addrlen = sizeof(Estructura_v6);
memset(&Estructura_v6, 0, addrlen);
Estructura_v6.sin6_family = AF_INET6;
Estructura_v6.sin6_port = htons(puerto);
inet_pton(AF_INET6, "3ffe:8040::1",&(Estructura_v6.sin6_addr));
inet_ntop(AF_INET6, &Estructura_v6.sin6_addr,
                        direccion_str_v6,addrlen);
sockfd=socket(PF_INET6, SOCK_STREAM, 0);

bind(sockfd,(struct sockaddr *)&Estructura_v6, addrlen);
```

Figura 32: Ejemplo uso función bind() en IPv6

```
struct sockaddr_storage Estructura_v4ov6;
int puerto=13;
socklen_t addrlen = sizeof(Estructura_v4ov6);
memset(&Estructura_v4ov6, 0, addrlen);
Estructura_v6.sin6_family = AF_INET6;
Estructura_v6.sin6_port = htons(puerto);
inet_pton(AF_INET6, "3ffe:8040::1",&(Estructura_v4ov6.sin6_addr));
sockfd=socket(PF_INET6, SOCK_STREAM, 0);

bind(sockfd,(struct sockaddr *)&Estructura_v4ov6, addrlen);
```

Figura 33: Ejemplo función bind() con estructura IPv4 e IPv6

Función accept():

Con la función accept() se ejemplifica el caso de aquellas funciones que reciben los datos del protocolo desde el kernel.

```
struct sockaddr_in Estructura_v4;
socklen_t tam = sizeof(Estructura_v4);

accept(sockfd,(struct sockaddr *)&Estructura_v4, &tam);
```

Figura 34: función accept() utilizando estructura IPv4

```
struct sockaddr_in6 Estructura_v6;
socklen_t tam = sizeof(Estructura_v6);

accept(sockfd,(struct sockaddr *)&Estructura_v6, &tam);
```

Figura 35: función accept() utilizando estructura para IPv6

```
struct sockaddr_storage Estructura_v4ov6;
socklen_t tam = sizeof(Estructura_v4ov6);

accept(sockfd,(struct sockaddr *)&Estructura_v4ov6, &tam);
```

Figura 36: función accept() utilizando estructura IPv4/IPv6

12.1.2. Funciones para resolución de direcciones

Para realizar las funciones de una variedad de operaciones con las direcciones IPv6 se requieren nuevas funciones. Como por ejemplo las funciones necesarias para el sistema de Nombres de Dominio ya sea para pasar de nombre a dirección como de nombre a dirección.

Para el caso de IPv4 existen dos funciones que realizan estas tareas gethostbyname() y gethostbyaddr() sin embargo no se pueden usar en IPv6 es por eso que se crean nuevas funciones que operan tanto en IPv4 como en IPv4, las que describiremos a continuación

- **Traducción desde Nombres a Direcciones**

La siguiente función esta desarrollada para hacer la traducción de nombre de domino a dirección IPv4 o IPv6

getipnodebyname():

Sintaxis:

```
struct hostent *getipnodebyname(const char *name, int af, int flags
                                int *error_num);
```

Figura 37: función getipnodebyname()

Descripción:

El argumento “name” puede ser un nombre de nodo o una dirección numérica en formato de string (decimal punteado para IPv4 y hexadecinal para IPv6).

El argumento “af” especifica la familia de direcciones que puede ser AF_INET o AF_INET6. en caso de error se retorna un puntero con el valor del tipo de error (HOST_NOT_FOUND, NO_ADDRESS, NO_RECOVERY, TRY_AGAIN).

El parámetro “flags” especifica el tipo de dirección que se esta buscando, y el tipo de dirección que se retorna:

Si el flags tiene el valor 0 y “af” es AF_INET, entonces el cliente busca direcciones IPv4, es decir se busca un registro tipo A. Si es satisfactoria la consulta, una dirección IPv4 es retornada y el campo h_length de la estructura hostent es asignado a 4, en caso contrario la funcion un puntero null.

Si el flags es 0 y “af” es AF_INET6 entonces la direcciones que se esta pidiendo es IPv6, es decir, se esta preguntando por un registro tipo AAAA o A6. En caso sue sea satisfactoria la consulta h_length tendrá el valor 16, en caso de algún error en la consulta se retorna un puntero a null.

Otro caso importantes es cuando el parámetro “flag” es especificado como AI_V4MAPPED y el “af” es establecido a AF_INET6, en este caso primero se busca una dirección IPv6 es decir un registro en el DNS del tipo A6 o AAAA en caso que no se encuentre ninguno, la consulta se cambia buscando una dirección IPv4 y se retorna una dirección IPv4 mapeada IPv6, h_length tendrá el valor 16. El flags es ignorado si “af” es distinto de AF_INET6.

En caso que el parámetro “flag” sea igual a AI_ADDRCONFIG, se obtendrá una dirección dependiendo del tipo de dirección que tenga configurada el nodo cliente, es decir si el nodo cliente no tiene direcciones IPv6 y el parámetro “af” tiene el valor AF_INET6, en este caso se retornara un puntero nulo.

Es recomienda usar el valor AI_DEFAULT para el parámetro “af”, su definición es:

#define AI_DEFAULT (AI_V4MAPPED | AI_ADDRCONFIG)

Como se menciona anteriormente esta función retorna una estructura que tiene el formato siguiente:

```
/* Description of data base entry for a single host.  */
struct hostent
{
  char *h_name;                 /* Official name of host.  */
  char **h_aliases;             /* Alias list.  */
  int h_addrtype;               /* Host address type.  */
  socklen_t h_length;           /* Length of address.  */
  char **h_addr_list;           /* List of addresses from name server.  */
#define h_addr  h_addr_list[0]  /* Address, for backward compatibility.  */
};
```

Figura 38: Estructura hostend

A continuación se describe como es llenada esta estructura dependiendo de los valores que tenga “name” y “af”

En el caso que “name” contenga una dirección IPv4 y “af” contenga el valor AF_INET, o “name” tenga una dirección IPv6 y “af” tenga el valor AF_INET6, los campos de estructura hostent serán:

h_name: puntero con el valor del parámetro “name”

h_aliases: puntero nulo

h_addrtype: una copia del valor del parámetro “af”

h_length: contiene 4 para AF_INET y 16 para AF_INET6

h_addr_list[0]: puntero a la dirección IPv6 o IPv4.

h_addr_list[1]: puntero nulo

El siguiente caso que es comúnmente manejado, cuando parámetro “name” es una dirección IPv4, el parámetro “af” tiene el valor AF_INET6 y por ultimo flag es AI_V4MAPPED, por tanto se entrega una dirección IPv4 capeada IPv6, en este caso los campos de la estructura hostent serán:

h_name: contiene un puntero a una dirección IPv6 (IPv4 capeada IPv6).

h_aliases: puntero nulo

h_addrtype: contiene AF_INET6

h_length: valor igual a 16

h_addr_list[0]: puntero a al dirección de 16 bytes

h_addr_list[1]: Puntero nulo.

- **Traducción de Nombre a Dirección**

La siguiente función tiene los mismos argumentos que la anterior, pero esta realiza el proceso inverso.

Getipnodebyaddr()

Sintaxis:

```
struct hostent *getipnodebyaddr(const void *src, size_t len,
                                int af, int *error_num);
```

Figura 39 función Getipnodebyaddr()

Descripción:

A continuación se trata el caso en que tenemos direcciones IPv4 mapeadas IPv6 e IPv4 compatibles IPv6, ya que son las que se prestan para mayor confusión.

En el caso que “af” tiene el valor AF_INET6, “len” es igual a 16 y la dirección IPv6 es IPv4 mapeadas IPv6 o IPv4 compatibles IPv6, en este caso se debe eliminar los primero 12 bytes de la dirección IPv6, poner el valor de “af” en AF_INET y “len” en 4.

Otro punto a tomar en cuenta son las direcciones especiales tales como ::1 y ::, para comprobar el tipo de dirección con el que se esta operando existen algunas macros:

```
#include <netinet/in.h>

int   IN6_IS_ADDR_UNSPECIFIED (const struct in6_addr *);
int   IN6_IS_ADDR_LOOPBACK    (const struct in6_addr *);
int   IN6_IS_ADDR_MULTICAST   (const struct in6_addr *);
int   IN6_IS_ADDR_LINKLOCAL   (const struct in6_addr *);
int   IN6_IS_ADDR_SITELOCAL   (const struct in6_addr *);
int   IN6_IS_ADDR_V4MAPPED    (const struct in6_addr *);
int   IN6_IS_ADDR_V4COMPAT    (const struct in6_addr *);
```

Figura 40: Macros que retornan verdadero en caso que la dirección sea del tipo especificado

- **Traducción de dirección de red y servicio**

Hay dos nuevas funciones para hacer Independiente las conversiones del nombre y de dirección, getaddrinfo() y el getnameinfo(). Además, el uso de estos nuevos en vez del gethostbyname y del gethostbyaddr se recomienda, pero se debe tener en cuanta la ventaja que tienen las dos ultimas funciones en la etapa de transición puesto que es mas fácil el manejo de las direcciones IPv4 mapedas IPv6 y las IPv4 compatibles que son descritas en las secciones anteriores.

A continuación se describen la función getaddrinfo() la cual retorna un lista enlazada de estructuras del tipo addrinfo los cuelas tiene la información solicitada referente a un hostname específico, ya sea direcciones, servicios, e información adicional que es almacenada en la lista.

Getaddrinfo()

Sintaxis:

```
int getaddrinfo(const char *node, const char *service,
                        const struct addrinfo *hints,
                        struct addrinfo **res);
```

Figura 41: getaddrinfo()

Descripción: retorna una lista enlazada de estructuras del tipo addrinfo ver siguiente figura

```
struct addrinfo {
            int     ai_flags;
            int     ai_family;
            int     ai_socktype;
            int     ai_protocol;
            size_t  ai_addrlen;
            struct sockaddr *ai_addr;
            char   *ai_canonname;
            struct addrinfo *ai_next;
        };
```

Figura 42: Estructura addrinfo

Esta función recibe como parámetros dos datos del nodo: nombre del nodo (**node), servicio (*service), de estos argumentos pueden ser ambos un puntero a nulo o solo uno dependiendo del tipo de nodo con que se este trabajando, por ejemplo si se trabaja como un cliente normal, se deben especificar ambos parámetros, la dirección o el nombre del nodo al que se quiere acceder además del nombre o numero de puerto del servicio que se quiera utilizar. En el caso que se trabaje solo como servidor solo es necesario especificar el número o nombre del servicio que se utilizara para dar servicio.

Al especificar el parámetro nodo (node), este puede ser un string que contenga el nombre del nodo o una dirección IPv4 (decimales y puntos) IPv6 (hexadecimal).

Opcionalmente se puede entregar una estructura de tipo addrinfo como parámetro (hints) para especificar aun más la información referente al tipo de socket que se utilizara en la comunicación, por ejemplo:

- Si solo se utiliza conexiones TCP y no UDP entonces en este caso la estructura (hints) en el campo ai_socktype tendrá el valor SOCK_STREAM

- Si solo se manejan direcciones IPv6 y no direcciones IPv4, entonces, en el campo ai_family de la estructura se establecerá con el valor PF_INET6. En el caso que se requiera utilizar cualquier protocolo de direcciones, el valor de ai_family será PF_UNSPEC.

Es importante mencionar que en caso de no especificar valores para la estructura hints, esta se debe inicializar completamente en cero.

Por medio del último argumento se retorna el puntero a la lista de una o mas estructuras, se encuentran enlazadas por medio del campo ai_next, cada estructura en el campo ai_addr es completada con una estructura de dirección, utilizada para la creación del socket la cual puede ser IPv4 o IPv6, el tamaño es especificado en el campo ai_addrlen, de tal manera de poder hacer la discriminación entre ellas.

En cada una de las estructuras los tres campos ai_family, ai_socktype, y ai_protocol se copian desde la estructura que es pasada como parámetro (hints).

Otro campo importante que tiene la estructura hints ai_flags el cual dependiendo se sus valores hace cambiar bastante la lista enlazada de retorno.

Si se quiere utilizar las estructuras de dirección del socket de la lista enlazada como parámetro del bind(), el valor del campo ai_flags debe ser AI_PASSIVE, en esta caso si el parámetro “node” es nulo, la sección donde se establece la dirección IP en la estructura de dirección se establece a INADDR_ANY en el caso de IPv4, y a IN6ADDR_ANY_INIT en el caso IPv6.

En caso contrario si no se establece el valor de AI_PASSIVE en ai_flags las estructuras de dirección de socket se pueden usar en las demás

funciones connect(), sendto(), o sendmsg(). Ya que si el parámetro “node” es nulo, entonces la sección de la estructura de dirección del socket donde se establece la dirección IP, se utiliza la dirección loopback.

Si se quiere tener el nombre canónico del nodo se debe poner el campo ai_flags el valor de AI_NUMERICHOST con esto la primera estructura de la lista contendrá en el campo ai_canonname el nombre DNS del nodo.

En el caso que no se quiera la resolución de nombre el valor que se debe usar en ai_flags es AI_NUMERICHOST.

Hasta este punto se tiene especificadas todas las funciones, estructuras, macros, etc que son necesarias para iniciar la comunicación vía socket. Otro punto importante y que es el objetivo de este tesis es el desarrollo de la implementación de Aplicaciones Multicast, para ello se requiere conocer algunas otras estructuras y opciones que se deben manejar al momento de la creación o migración de aplicaciones Multicast.

13. API para Multicast

En el desarrollo de aplicaciones multicast se utiliza el protocolo UDP y direcciones multicast IPv6, por tanto todas la funciones y estructuras que se definieron con anterioridad son utilizadas con dicho protocolo, a continuación se describen los cambios o consideraciones que se deben tener al momento de desarrollar o migrar una aplicación.

Se necesitan diversas extensiones al API de programación para poder soportar multicast. Todas ellas se manejan a través de dos llamadas al sistema: setsockopt() (usada para pasar información al kernel) y getsockopt() (para obtener información referente al comportamiento multicast). Para estas funciones existen tres opciones del

socket en la capa IPPROTO_IPV6 que controlan el envió de los paquetes multicast IPv6, las opciones se describen a continuación:

a).- IPV6_MULTICAST_IF

Generalmente, el administrador del sistema especifica el interfaz por el que se envían, por defecto, los datagramas multicast. El programador puede modificar esto y elegir un interfaz concreto para un socket determinado con esta opción.

Para elegir el interfaz de salida, las siguientes ioctls pueden ser útiles: SIOCGIFADDR (para obtener la dirección de un interfaz), SIOCGIFCONF (para obtener una lista con todos los interfaces) y SIOCGIFFLAGS (para obtener las flags de un interfaz, y, por tanto determinar si el interfaz tiene capacidades multicast).

```
struct in_addr interface_addr;
setsockopt (socket,IPPROTO_IPV6, IP_MULTICAST_IF, &interface_addr,
                                        sizeof(interface_addr));
```

Figura 43: Ejemplo IPV6_MULTICAST_IF

b).- IPV6_MULTICAST_HOPS

Especifica la cantidad de saltos de los paquetes multicast IPv6, si este valor es establecido en -1, entonces, se utilizara el valor que tenga configurado el kernel, pero si el valor esta entre 0 y 255 utilizara el valor establecido.

```
int hops=20;
setsockopt(descriptor,IPPROTO_IPV6,IPV6_MULTICAST_HOPS,
                                  &hops,sizeof(int));
```

Figura 44: Ejemplo de IPV6_MULTICAST_HOPS

c).- IPV6_MULTICAST_LOOP

Especifica si el datagrama es enviado a la interfase local, es decir, especifica si la información que envía por la interfase puede ser recibida en la misma interfase. Si se desea recibir una copia del datagrama que se envia por la interfase, este parámetro se debe establecer al valor 1, en caso contrario, es decir que no se requiere una copia de los datos el valor se establece a 0.

```
int loopback=1;
setsockopt(descriptor,IPPROTO_IPV6,IPV6_MULTICAST_LOOP,
                                &loopback,sizeof(int));
```

Figura 45: Ejemplo IPV6_MULTICAST_LOOP

La recepción de paquetes multicast IPv6 es controlada por dos opciones en el setsockopt(), que se agregan a las anteriores, estas opciones utilizan como parámetro un estructura denominada ipv6_mreq que es definida de la siguiente forma

```
struct ipv6_mreq {
       struct in6_addr ipv6mr_multiaddr; /* Direccion Multicast Ipv6  */
       unsigned int    ipv6mr_interface; /* interfase index */
   };
```

Figura 46: definición estructura ipv6_mreq <netinet/in.h>

d).- IPV6_ADD_MEMBERSHIP

El nodo se une al grupo multicast especificado por la estructura de dirección de socket ipv6mr_multiaddr, la interfase local especificada en ipv6mr_interface.

```
struct ipv6_mreq Struct_Multicast_v6;
Struct_Multicast_v6.ipv6mr_multiaddr=((struct sockaddr_in6 *)
                                      &Estruct_Storage_Servidor)->sin6_addr;
Struct_Multicast_v6.ipv6mr_interface=if_index;
setsockopt(descriptor,IPPROTO_IPV6,IPV6_ADD_MEMBERSHIP,&Struct_Multicast_v6,
                                      sizeof(Struct_Multicast_v6));
```

Figura 47: ejemplo IPV6_ADD_MEMBERSHIP

e). - IPV6_DROP_MEMBERSHIP

El nodo abandona el grupo multicast especificado por la estructura socket ipv6mr_multiaddr.

Capitulo 6 - Metodología para el Desarrollo de Aplicaciones Basadas en ipv6

14. Introducción

El importante cambio introducido por IPv6 en el tamaño de las direcciones, hace que los programas hechos para operar con la versión actual del protocolo IP no puedan funcionar con la versión 6. Sin embargo, la conversión de los programas a IPv6 no es muy complicada, por lo ya descrito anteriormente. Los cambios radican principalmente en las estructuras de datos utilizadas, las que se tuvieron que adaptar a las nuevas exigencias del protocolo.

Y es en este proceso de adaptación o de construcción de aplicaciones que se deben tener algunas consideraciones que permitiran tener una aplicación bien diseñada y apta para los nuevos tiempos, en que aún no se tiene una versión definitiva del nuevo protocolo que regirá Internet en los próximos años. Y es en este sentido que apunta este capitulo, entregar una visión clara de todo los elementos que se deben tener en cuenta al momento de desarrollar una aplicación.

14.1. La separación en módulos

Una de las principales recomendaciones al momento de diseñar o adaptar una aplicación es la separación de la sección correspondiente a la capa IP de las demás funcionalidades de la aplicación, haciendo que la esta sea independiente del sistema de red utilizado, esto quiere decir que si el protocolo de red cambia en algún momento, solo el modulo correspondiste a la capa IP será afectado y no se requerirán grandes esfuerzos para lograr la compatibilidad con el o los nuevos protocolos. Para lograr este objetivo juegan un papel importante las estructuras de datos, puesto que son estas la que proporcionan el canal de comunicación entre el modulo IP y los demás módulos de la aplicación, permitiendo que la aplicación haga uso del modulo de comunicación sin tener conocimiento de la versión del protocolo que está utilizando.

Otra recomendación importante que se debe tener en cuenta y que ya se menciono de cierta forma en capítulos anteriores es la portabilidad, es decir, hacer la aplicación independiente del protocolo de red que esté utilizando el nodo sobre el cual se esta ejecutando, como es el caso actual en que se requiere que la aplicación se ejecute sobre nodos con versión de IP mixto o versión de IP única cualquiera sea esta, lo que se logra utilizando en el medio de comunicación con el modulo IP estructuras genéricas que no discriminen entre las versiones de IP, esto es simple de realizar mientras se cuente con las funciones, estructuras y el soporte que permitan la portabilidad.

Continuación analizaremos la forma de implementar una aplicación utilizando estructuras y funciones genéricas que permitan la portabilidad y la modularidad, ejemplificado mediante el desarrollo de una aplicación Multicast de tal manera de abarcar un gran segmento de las características IPv6, quedando fuera de esto IPsec,

Control de flujo y Movilidad que son temas más amplios y que pueden dar motivo al desarrollo de otros proyectos de tesis.

El primer problema a resolver es la elección del la familia de direcciones, AF_INET o AF_INET6, sin embargo este es un problema que puede ser resuelto mediante la utilización de funciones que realicen la elección, para ello se debe llenar las estructuras de dirección con la familia de direcciones genéricas AF_UNSPEC, los mismo sucede para el caso del protocolo, lo especificamos en las estructuras de forma genérica PF_UNSPEC.

```
//**definicion de la estructura**//
struct addrinfo Entrada;

//**llenado de la estructura patron para la funcion addrinfo()**//
Entrada.ai_family=AF_UNSPEC;
Entrada.ai_socktype=SOCK_DGRAM;
Entrada.ai_protocol=PF_UNSPEC;
```

Figura 48: Elección de la familia y protocolo de red genéricos

El próximo paso el la obtención de la lista de direcciones con que cuenta el nodo, para ellos se utiliza la función getaddrinfo() que nos entrega una lista enlazada con todas las direcciones que cumplen con las características definidas en la estructura patrón [figura 48] y los parámetros que se entregan a la función.

```
//**definicion de la estructura**//
struct addrinfo Entrada;
struct addrinfo *Salida;

char *GrupoMulticast;
char *NumeroPuerto;

//**Asignacion de grupo multicast y puerto de comunicacion**//
GrupoMulticast=argv[1];
NumeroPuerto=argv[2];

getaddrinfo(GrupoMulticast,NumeroPuerto,&Entrada,&Salida);
```

Figura 49: obtención de lista de direcciones

Ahora al tener almacenado toda la información en la lista enlazada disponible en la estructura “Salida” debemos recorrer la lista y elegir aquella que nos sea útil para la comunicación.

Una vez que encontramos en uno de los nodos de la lista la dirección que nos asegura funcionamiento del socket, se copian los datos a una estructura de dirección genérica sockaddr_storage ya que la dirección elegida puede ser tanto IPv4 como IPv6.

```
struct addrinfo *Recorre;
 struct sockaddr_storage Estruct_Storage_Servidor;
Recorre=Salida;
i=0;
while (Recorre)
  {
    descriptor=socket(Recorre->ai_family,Recorre->ai_socktype,
                                        Recorre->ai_protocol);
    if (descriptor!=-1)
        {
         retorno_func=bind(descriptor,Recorre->ai_addr,
                                        Recorre->ai_addrlen);
         if (retorno_func!=-1)
            {
             close(descriptor);
             memcpy(&Estruct_Storage_Servidor,Recorre->ai_addr,
                                sizeof(Estruct_Storage_Servidor));
             printf("Direccion OK\n");
             direccionOK=1;
             break;
            };
          close(descriptor);
          descriptor=-1;
        }
    Recorre=Recorre->ai_next;
   };
freeaddrinfo(Recorre);
if (direccionOK==0)
  {
    printf("No se pudo conseguir una direccion valida\n");
    exit(1);
  }
```

Figura 50: obtención de una dirección valida

El paso siguiente es crear el socket en base a la dirección almacenada en la estructura genérica, y luego crear el enlace del socket con la función bind.

```
descriptor=socket(Estruct_Storage_Servidor.ss_family,SOCK_DGRAM,0);
if (descriptor==-1) printf("Error en el Socket\n");
    else printf("Socket Creado Con Exito\n");

retorno_func=bind(descriptor,(struct sockaddr *) &Estruct_Storage_Servidor,
                                    sizeof(Estruct_Storage_Servidor));
if (retorno_func==-1) perror("\nError en el Bind\n");
    else printf("\nBind Exitoso!!\n");
```

Figura 51: socket y bind con estructura genérica

Una vez que se a creado el socket se a enlazado el bind, a continuación se inicia la configuración de Multicast para nuestro servidor.

En primer lugar no existe una estructura genérica similar a sockaddr_storage que nos permita trabajar con grupos multicast IPv6 e IPv4, por lo tanto, tenemos que primero identificar que grupo de direcciones fue elegido y en base a ello utilizar una estructura de direcciones multicast ya sea para IPv4 (ip_mreq) en le caso que la familia de direcciones sea AF_INET

```
struct ip_mreq
  {
    struct in_addr imr_multiaddr; /* IP multicast address of group */
    struct in_addr imr_interface; /* local IP address of interface */
  };
```

Figura 52: Estructura multicast IPv4 <bits/in.h>

En el caso que la familia de direcciones elegida sea IF_INET6 entonces tendremos que usar es ip_mreq6

```
struct ipv6_mreq
 {
   /* IPv6 multicast address of group */
   struct in6_addr ipv6mr_multiaddr;

   /* local interface */
   unsigned int ipv6mr_interface;
 };
```

Figura 53: Estructura Multicast IPv6 <netinet/in.h>

Para poder elegir que estructura de direcciones multicast utilizaremos, la decisión es tomada en base a la familia de protocolo que contiene la estructura sockaddr_storage (Estruct_Storage_Servidor) el campo ss_family.

```
switch (Estruct_Storage_Servidor.ss_family)
   {
    case AF_INET:
      {
        Struct_Multicast_v4.imr_multiaddr.s_addr=
                ((struct sockaddr_in *)&Estruct_Storage_Servidor)->sin_addr.s_addr;
        strncpy(Struct_Interface.ifr_name,if_name,IFNAMSIZ);
        retorno_func=ioctl(descriptor,SIOCGIFADDR,&Struct_Interface);
        if (retorno_func==-1) perror("\nError en ioctl\n");
            else printf("\nioctl Exitoso!!\n");
        Struct_Multicast_v4.imr_interface=
                ((struct sockaddr_in *) &Struct_Interface.ifr_addr)->sin_addr;
        .
        .
        .
     case AF_INET6:
      {
        Struct_Multicast_v6.ipv6mr_multiaddr=
                ((struct sockaddr_in6 *) &Estruct_Storage_Servidor)->sin6_addr;
        Struct_Multicast_v6.ipv6mr_interface=if_index;
        .
        .
        .
```

Figura 54

Una vez completadas las estructuras que se utilizara, se deben establecer las opciones multicast, las cuales también deben hacerse por separado, ya que las funciones que se encargan pasar la información al kernel utilizan diferentes parámetros para cada uno de los protocolos, entre estos parámetros están las estructuras que como se dijo anteriormente son distintas para cada protocolo.

```
int loopback=1;
int hops=8;
const int on=1;
.
.
.
retorno_func=setsockopt(descriptor,SOL_SOCKET, SO_REUSEADDR,
                                        &on,sizeof(on));
if (retorno_func==-1) perror("\nError en SO_REUSEADDR\n");
    else printf("\nSO_REUSEADDR Exitoso!!\n");
retorno_func=setsockopt(descriptor,IPPROTO_IP, IP_MULTICAST_LOOP,
                                &loopback,sizeof(loopback));
if (retorno_func==-1) perror("\nError en IP_MULTICAST_LOOP\n");
    else printf("\nIP_MULTICAST_LOOP Exitoso!!\n");
retorno_func=setsockopt(descriptor,IPPROTO_IP, IP_MULTICAST_TTL,
                                        &hops,sizeof(hops));
if (retorno_func==-1) perror("\nError en IP_MULTICAST_TTL\n");
    else printf("\nIP_MULTICAST_TTL Exitoso!!\n");
retorno_func=setsockopt(descriptor,IPPROTO_IP, IP_ADD_MEMBERSHIP,
               &Struct_Multicast_v4,sizeof(Struct_Multicast_v4));
if (retorno_func==-1) perror("\nError en IP_ADD_MEMBERSHIP\n");
    else printf("\nIP_ADD_MEMBERSHIP Exitoso!!\n");
break;
```

Figura 55: Configuración opciones multicast para IPv4

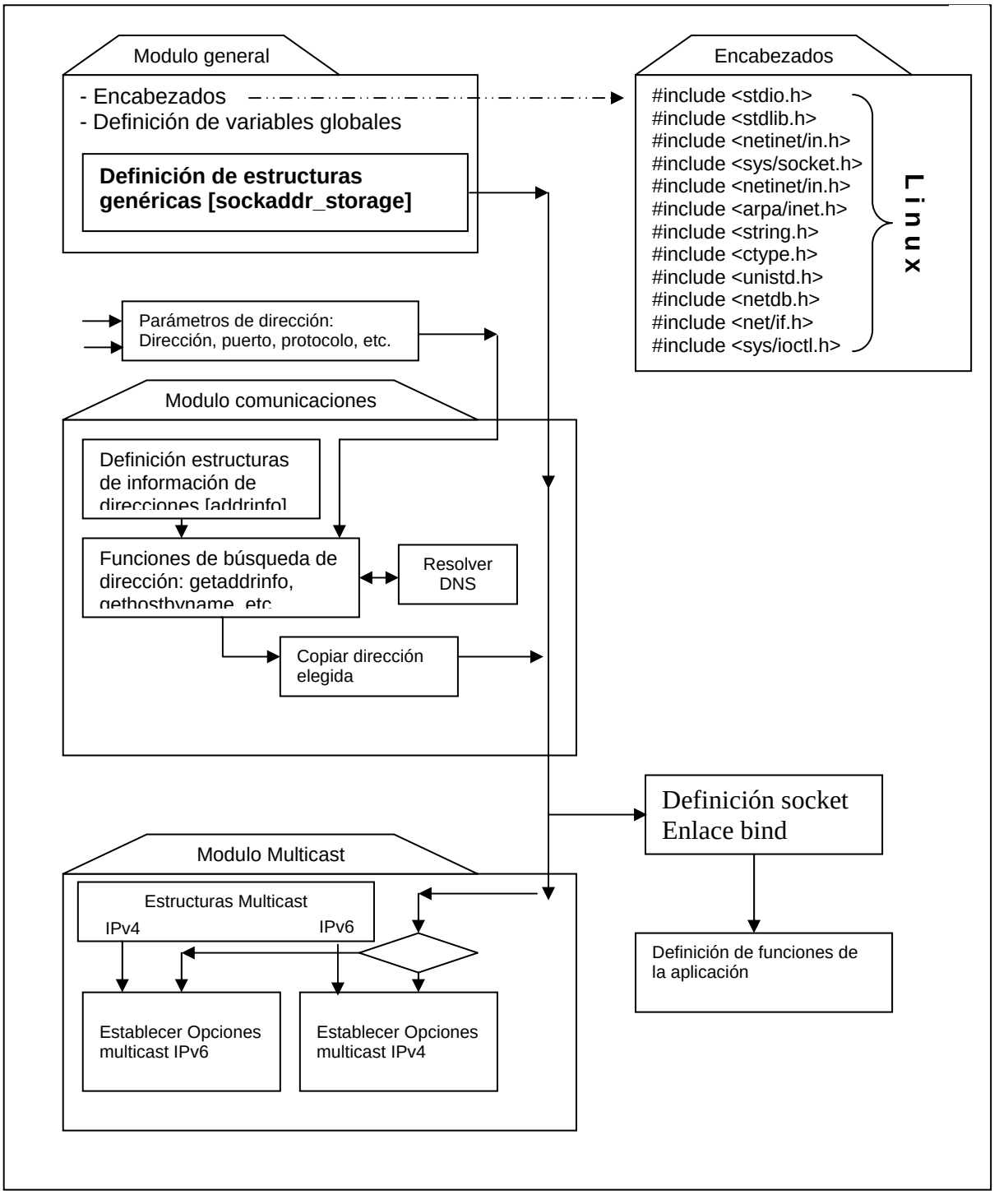
```
int loopback=1;
int hops=8;
const int on=1;
.
.
.
loopback=1;
retorno_func=setsockopt(descriptor,IPPROTO_IPV6,IPV6_MULTICAST_LOOP,
                                                &loopback,sizeof(int));
if (retorno_func==-1) perror("\nError en el IPV6_MULTICAST_LOOP\n");
  else printf("\nIPV6_MULTICAST_LOOP Exitoso!!\n");
retorno_func=setsockopt(descriptor,IPPROTO_IPV6,IPV6_MULTICAST_HOPS,
                                                &hops,sizeof(int));
if (retorno_func==-1) perror("\nError en el IPV6_MULTICAST_HOPS\n");
  else printf("\nIPV6_MULTICAST_HOPS Exitoso!!\n");
retorno_func=setsockopt(descriptor,IPPROTO_IPV6,IPV6_ADD_MEMBERSHIP,
               &Struct_Multicast_v6,sizeof(Struct_Multicast_v6));
if (retorno_func==-1) perror("\nError en el IPV6_ADD_MEMBERSHIP\n");
  else printf("\nIPV6_ADD_MEMBERSHIP Exitoso!!\n");
break;
```

Figura 56: Configuración de Opciones Multicast para IPv6

Una vez que se ha establecido las opciones multicast el sistema se encuentra listo ya sea para enviar o recibir Información vía multicast, entonces ahora la tarea es construir los algoritmos que creen, envíen y administren los paquetes multicast.

14.2. Resumen

El esquema siguiente muestra un diagrama resumen con el modelo de programación que permitiría tener una aplicación moderna y compatible con ambos protocolos de red y que puede ser aplicado a cualquier nuevo desarrollo de aplicación, con cualquier nuevo protocolo de red.



Otra forma de a hacer una aplicación moderna y que opere e ambos protocolos es manipular las opciones de compilación.

14.3. Las Opciones de Compilación

Otra forma de desarrollar las aplicaciones es recurrir a la opciones de compilación que tenga en lenguaje de programación por ejemplo en el caso de C se pueden utilizar la opciones de compilación #ifdefs

Ejemplo:

```
if SoporteIPv6()
 then
    {
    DefinirIPv6(); //poner #define IPV6 1
    Salida("soporte Ipv6 OK!");
    }
 else
    Salida("NO soporte para IPv6");
```

Figura 57: Opciones de Compilación

En el ejemplo la función SoporteIPv6() comprueba la existencia de algún elemento clave que solo esta disponible si el nodo en el que se esta compilando existe soporte para IPv6 como por ejemplo el archivo del modulo.

Por otra parte la función DefinirIPv6() consistes en poner en alguna de las líneas de los archivos de encabezado la instrucción #define IPv6 1 de tal manera que en el código se pueda utilizar la instrucción #ifdefine IPv6, de tal forma de poder discriminar si se pueden utilizar o no las características ipv6 en el nodo para el cual se esta compilando la aplicación.

Ejemplo:

```
if (ai->ai_family == AF_INET) {
   pr = &proto_v4;
#ifdef  IPV6
   } else if (ai->ai_family == AF_INET6) {
                pr = &proto_v6;
                if (IN6_IS_ADDR_V4MAPPED(&(((struct sockaddr_in6 *)
                                ai->ai_addr)->sin6_addr)))
                err_salida("No se pueden usar direcciones IPv4 mameadas IPv6");
#endif
        } else
                err_salida("Error en Familia de Direcciones %d", ai->ai_family);
```

Figura 58: Ejemplo opciones de compilación #ifdef

En el ejemplo se comprueba la posibilidad de utilizar direcciones IPv4 mapeadas como IPv6. En este caso si no hay soporte para IPv6 hay bastante código que se ahorra lo cual implica que las aplicaciones estarán optimizadas de acuerdo al protocolo.

Las opciones de compilación son posibles en aquellos casos en que las aplicaciones son distribuidas con sus códigos fuentes, lo que en la realidad se da con muy poca frecuencia solo en ambientes open sourse.

Cuando las aplicaciones son desarrolladas para su distribución en forma de binarios, la mejor opción para su desarrollo es implementar la portabilidad, para ellos se utilizan las opciones, estructuras y funciones que entrega el lenguaje de programación.

Conclusiones

Dentro de la comunidad Internet se ha acuñado la frase “IPv6 es el futuro. La pregunta no es si IPv6 sustituirá a IPv4 o no, sino que la verdadera pregunta que debemos hacernos es ¿cuando lo sustituirá?”. Después de la investigación, trabajos y pruebas sobre IPv6 se tiene una idea clara para poder responder esta y otras interrogantes que surgen en relación a las redes de nueva generación. Hoy se puede decir que la clave en la transición a IPv6 no esta en las redes sino que esta en la aplicaciones y servicios, es decir, que tan preparadas están la aplicaciones para trabajar en las redes de nueva generación, cuya transición a IPv6 ya es un echo puesto que en estos momentos se cuenta con una infraestructura tanto a nivel físico como administrativo que permite tener redes IPv6 de producción.

Los problemas que se tienen con respecto a las aplicaciones, es lograr que las antiguas aplicaciones soporten el nuevo protocolo y que la nuevas aplicaciones implemente el nuevo protocolo en toda su magnitud ya que es mas barato desarrollar aplicaciones que soporten desde un principio IPv6 que implementarlas primero para IPv4 y luego migrarlas a IPv6.

En el largo transcurso de este proyecto surgieron muchos problemas los que fueron superado uno a uno y en le caso de poderse solucionar se busco una alternativa.

Entre los grandes problemas que un desarrollador se enfrenta al momento de implementar una aplicación o servicio, es seleccionar el sistema operativo que utilizara. Como se grafico en la sección de configuración de laboratorio, las dos plataformas mas difundidas hoy en dia soportan o están en fase de desarrollo del nuevo protocolo, sin embargo esas tareas no están terminadas puesto que hay

muchas características que no están implementadas bajo condiciones normales, es decir, que solo están disponibles para usuarios avanzados como por ejemplo para tener IGMPv6 en Linux se requiere un kernel especial USAGI en otro caso para poder tener trafico Multicast v6 en cualquiera de sus modos no se puede lograr en los sistemas operativos tradicionales (Windows, Linux), para ello se requiere Freebsd. Y así se podrían enumerar varias características de IPv6 que no pueden ser logradas por usuarios tradicionales.

Otros problemas que nos encontramos es que características que están implementadas no tienen las interfaces de programación para poder sacarle provecho, un ejemplo de ellos es el tiempo de vida de las direcciones IPv6.

Otro punto importante sobre el cual se debe reflexionar y tomar muy en cuenta desde el punto de vista de programación, es la fuerte dependencia que existe entre las aplicaciones y el servicio DNS, lo que trae como consecuencias que la aplicaciones podrán ser adoptadas por el usuarios final siempre y cuando este no tenga que escribir una larga dirección IPv6 para poder hacer uso de la aplicación, es por esto que mientras no se tenga completamente defino el problema que existe entre autoconfiguracion y DNS los servicios migrados al nuevo protocolo no tendrán éxito frente al usuario final.

El desarrollo o migración de aplicaciones multicast esta bastante retrazado en comparación a las aplicaciones Unicast, y esto de debe principalmente a que el soporte para las características IPv6 no están soportadas por los lenguajes y sistemas operativos, pero si se nota que hay una preocupación por lograr dicho soporte y es uno de los objetivos que se buscaron en este proyecto de tesis, desarrollar un modelo de implementación de aplicaciones IPv6, recopilando toda la información que actualmente hay en cuanto al desarrollo de aplicaciones Multicast

IPv6, desarrollando una pequeña aplicación que ejemplifique y sirva de guía para los futuros programados de la red de nueva generación.

Por ultimo en base a la experiencia adquirida durante el desarrollo de este proyecto se puede decir que la migración de aplicaciones nos difícil en el caso que la aplicación este bien estructurada en la sección que corresponde a comunicación, pero si la aplicación no tiene una estructura definidita o no usa las herramientas de programación apropiadas (funciones) el proceso de migración puede ser muy costoso, y existen casos en que fue mas fácil y viable económicamente desarrollar la aplicación desde cero. En el caso de implementar una aplicación desde cero es muy recomendable implementar una separación clara entre la aplicación y el modulo de red de tal manera de poder lograr los cambios futuros que se requieran para tener un soporte completo para IPv6, ya que los cambios a futuro serán una realidad ya que todavía hay segmentos en desarrollo.

Bibliografía

[EIR-01] Encapsulamiento de IPv6 en Redes ATM
http://www.linti.unlp.edu.ar/trabajos/tesisDeGrado/Redes%20ATM/Tesis%20.doc

[IV6-01] IP version 6 (IPv6) Microsoft
http://www.microsoft.com/resources/documentation/WindowsServ/2003/standard/proddocs/en-us/Default.asp?url=/resources/documentation/WindowsServ/2003/standard/proddocs/en-us/sag_IP_v6_ovr_Topnode.asp

[MSTP-01] Microsoft IPv6 Technology Preview for Windows 2000
http://msdn.microsoft.com/downloads/sdks/platform/tpipv6.asp

[TOOL - 01] Toolnet6
http://www.hitachi.co.jp/Prod/comp/network/pexv6-e.htm

[TRUM - 01] Trumpet IPv6
http://www.trumpet.com.au/ipv6.htm

[IEAF- 01] IPv6 enabled applications for Windows
http://win6.jp/win2kxp-ipv6apps.html

[RFC2460] Internet Protocol, Version 6 (IPv6) Specification
S. Deering , R. Hinden
http://www.ietf.org/rfc/rfc2460.txt?number=2460

[TMTR - 01] The Multi-Threaded Routing Toolkit
http://www.merit.edu/~mrt/windows.html

[EHM-02] The Evaluation of High-Speed Multicast Data Transmission over IPv6 Networks
Kengo NAGAHASHI, Hiroshi ESAKI, Jun MURAI
http://www.isoc.org/isoc/conferences/inet/00/cdproceedings/1c/1c_3.htm

[TNGI-01] The Next Generation Internet Protocol and Its Test
http://www.ict.ac.cn/xueshu/2001/h035.doc

[TAHP-01] TAHI Project
http://www.tahi.org

[KAME-01] KAME Project
http://www.kame.net/

[LIDP-01] Linux IPv6 Development Project
http://www.linux-ipv6.org/

[IDM-03] Implementation and Deployment of IPv6 Multicasting
Tatuya JINMEI
http://www.isoc.org/inet2000/cdproceedings/1c/1c_2.htm

[MWI-04] Microsoft Windows IPv6
Microsoft Corporation
http://www.microsoft.com/windows.netserver/technologies/ipv6/default.mspx

[IGW-05] IPv6 Guide for Windows Sockets Applications
Microsoft Corporation
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/portguid_5ucy.asp

[PXI-06] Project: XCAST over IPv6
Takahiro Kurosawa, Yuji IMAI
http://sourceforge.net/projects/xcast6

[ERM-07] Explicit Route Multicast (ERM)
J. Bion, J. Bion, M. Shand, A. Tweedly
http://www.aist-nara.ac.jp/~visoo-va/mynotes/Multicast/draft-shand-erm-00.txt

[EMX-08] Explicit Multicast (Xcast) Basic Specification
R. Boivie, N. Feldman, W. Livens, D. Ooms, O. Paridaens
http://www.aist-nara.ac.jp/~visoo-va/mynotes/Multicast/draft-ooms-xcast-basic-spec-01.txt

[IDI-09] Implementation and Deployment of IPv6 Multicas
Tatuya JINMEI
http://www.isoc.org/isoc/conferences/inet/00/cdproceedings/1c/1c_2.htm

[SICA-10] Seminario de ingeniería civil acerca de ipv6
Pablo Carmona Amigo - Renato Ulloa Sepúlveda
http://ipv6.inele.ufro.cl

[IJW-01] IPv6 Java for Windows
https://doc.telin.nl/dscgi/ds.py/View/Collection-188

[6SOS] IPv6 Servicio de Información y Soporte
http://www.6sos.net

[RFC2464] Transmission of IPv6 Packets over Ethernet Networks
M. Crawford
http://www.ietf.org/rfc/rfc2464.txt?number=2464

[RFC2133] Basic Socket Interface Extensions for IPv6
R. Gilligan , S. Thomson, J. Bound, W. Stevens
http://www.ietf.org/rfc/rfc2133.txt?number=2133

[RFC2292] Advanced Sockets API for IPv6
W. Stevens, M. Thomas
http://www.ietf.org/rfc/rfc2292.txt?number=2292

[HTO01] Porting applications to IPv6
Eva M. Castro
http://jungla.dit.upm.es/~ecastro/IPv6-web/ipv6.html

[MULTv6] Multicast Multicast in IPv6
David Larrabeiti López
http://long.ccaba.upc.es/long/050Dissemination_Activies/david_larrabeiti.pdf

[RFC1886] DNS Extensions to support IP version 6
http://www.ietf.org/rfc/rfc1886.txt

[RFC2710] Multicast Listener Discovery (MLD) for IPv6
http://www.ietf.org/rfc/rfc2710.txt

[SAIF01] Socket API for IPv6 {traffic class,flow label} field
http://long.ccaba.upc.es/long/050DisseminationActivitis/david_larrabeiti01.pdf

[USAGI01] USAGI(UniverSAl playGround for Ipv6)
http://www.linux-ipv6.org

[CEIP01] Capítulo Español del IPv6 Task Force
http://www.spain.ipv6tf.org

[DASI-01] Desarrollo de Aplicaciones con soporte IPv6
http://www.cudi.edu.mx/otono_2003/presentaciones/DesarrolloAPI.PDF

[SOD-01] Selections of Directions
http://mark.doll.name/i-d/ipv6/draft-ietf-ipv6-default-addr-select.txt.gz

[TAIP-01] Taller IPv6
http://www.ipv6.unam.mx/documentos/Taller-IPv6.pdf

Apéndice A - Configuración del Laboratorio de Pruebas

15. Introducción

En esta etapa del proyecto se describen los componentes y configuraciones software y hardware que se utilizaron el las pruebas de sistemas operativos. Cada sistema operativo será probado en forma separada pasando por la fase de instalación, configuración, pruebas individuales y pruebas en red.

Comenzaremos con la configuración de sistemas operativos de la familia MSWindows desde sus versiones más modernas puesto que para las anteriores Microsoft no tiene planeado un desarrollo del protocolo IPv6.

Microsoft adopto el protocolo IPv6 a partir de la versión Windows 2000 publicando el “Microsoft IPv6 Technology Preview for Windows 2000” el cual se encuentra disponible para todo usuario en http://msdn.microsoft.com/downloads/sdks/platform/tpipv6.asp, actualmente esta en su versión 1.205, es una versión no definitiva por lo cual no cuenta con soporte y se recomienda su uso en ambientes no productivos.

Para el desarrollo de aplicaciones IPv6 sobre esta versión de prueba del protocolo se requiere de:

- Platform SDK 2000
- Visual C++ 6.0 o posterior
- Microsoft IPv6 Technology Preview

Para el caso de este proyecto de tesis se decidió no utilizar Windows 2000 puesto que por ser una versión de prueba no esta lo suficientemente madura, por lo anterior se comienza con la versión Windows XP, la cual también cuenta con un soporte IPv6 bastante mejorado y a partir de la cual Microsoft lanzo una versión definitiva en Windows 2003.

16. Configurar la infraestructura

La infraestructura de la red del laboratorio de pruebas esta equipado con cuatro equipos los cuales están configurados con la ultima versión en cada familia de sistema operativo.

- PC1: Sistema Operativo Windows XP SP1 con SDK 2004, visual studio 6.0 (Visual C++) y Advanced Networking Pack.
- PC2: Sistema Operativo Windows 2003 Server
- PC3: Sistema Operativo RedHat Kernel Usagi basado en 2.6, GNU project C and C++ Compiler con el IDE para KDE (Kdeveloper).
- PC4: Sistema Operativo RedHat Kernel 2.6, servidor Dns Bind 9 , anuncio de router por medio de Ravd version.

Redes Configuradas

Unicast

IPv4: 10.10.10.0/24

IPv6: 3ffe:8040::/48

Multicast

IPv4: 224.0.1.22

IPv6: FF01::1111

16.1. Implementación IPv6 Familia Microsoft Windows

16.1.1. Microsoft Windows XP

Microsoft Windows XP (antes del SP1) incluye una versión para desarrolladores del protocolo IPv6 que no contaba con todas las características y una documentación bastante escasa. En Windows XP SP1 se incluye la versión válida para entornos de producción del protocolo IPv6 con bastante documentación en línea, además Microsoft creo un centro de soporte para IPv6 por medio de news disponible en news:// microsoft.public.platformsdk.networking.ipv6.

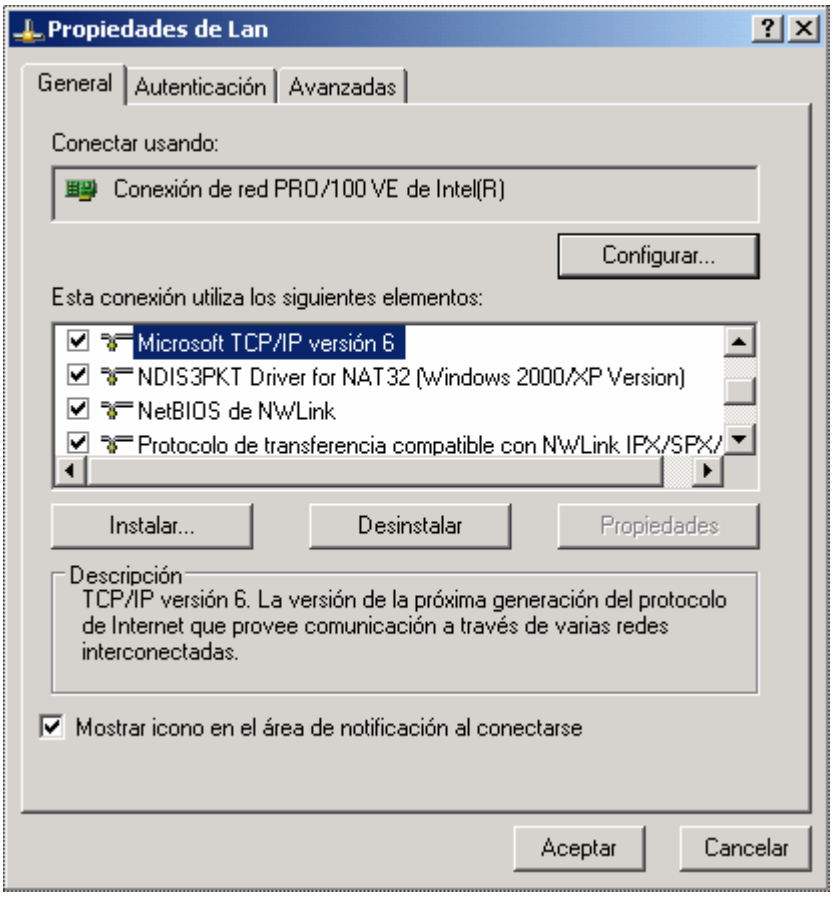
A continuación se describe el proceso de instalación y configuración del nodo con Windows XP en el laboratorio de pruebas de este proyecto.

Una vez instalado Windows XP SP1 se procede habilitar el soporte ipv6 ya que por defecto no lo está, en este punto se recomienda obtener desde el sitio http://download.microsoft.com el Advanced Networking Pack para Windows XP SP1 el cual actualmente esta en su versión 1.0 que cuenta con un conjunto de tecnologías diseñadas para ejecutarse en Windows XP SP1 con el fin de permitir el uso y la implementación de aplicaciones distribuidas de igual a igual (P2P) basadas en estándares de Internet. La actualización incluye una versión nueva de la pila de IPv6, incluida la compatibilidad para recorridos NAT para aplicaciones IPv6. Se incluye un servidor de seguridad IPv6 (ICF, *Internet Connection Firewall*) para proteger el equipo del usuario final de tráfico IPv6 no deseado, mientras que la plataforma de igual a igual facilita la escritura de soluciones distribuidas.

Una vez instalado el paquete se puede comprobar si fue satisfactorio o no chequeado que existe un valor DWORD **Installed** con el valor 1 en la siguiente clave del Registro:

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Hotfix\KB817778

El paso siguiente es habilitar el protocolo en las interfaces del equipo para ello se debe agregar el en el cuadro de dialogo que contiene los protocolo y servicios de red [Instalar → Agregar Protocolo]



Una vez habilitado el protocolo IPv6 se agrega automáticamente una dirección Ipv6 a la interfase y además se crean automáticamente dos Pseudo-Interfase por cada una de las interfaces que tenga el equipo:

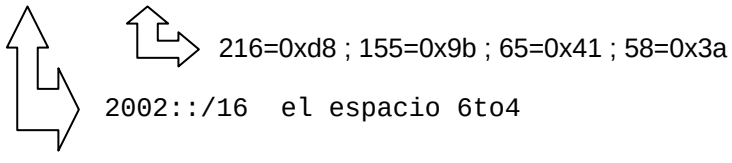
Pseudo-Interfase de Túnel 6to4: Esta interfase es creada para los túneles 6to4 [RFC 3056]. Un túnel 6to4 utiliza una máquina que se encarga de entender un tipo especial de paquetes IPv4 que son los paquetes 6to4, estos paquetes encapsulan paquetes IPv6 en paquetes IPv4 y crean un túnel entre nosotros y los sitios que entienden IPv6. Esto es necesario por que los ISP todavía no ofrecen conectividad IPv6 por lo que por ahora tenemos que utilizar túneles.

6to4 utiliza un tipo especial de direcciones, una dirección que traduce nuestra dirección pública única IPv4 en una dirección IPv6 única que nos permitirá conectarnos con el broker 6to4 para tener conectividad ipv6. La dirección 6to4 que nos corresponde se forma utilizando el prefijo 2002: y añadiéndole cada uno de los bytes de nuestra dirección ip ipv4 de dos en dos, en hexadecimal y separados por dos puntos.

Ejemplo:

Dirección IPv4: 216.155.65.58

Dirección 6to4: 2002:d89b:413a::d89b:413a



En el caso especifico de Microsoft existe una maquina que se encarga de interpretar los paquetes 6to4, la dirección IPv6 (6to4) de esta maquina se agrega automáticamente en la configuración como router por defecto.

En la figura siguiente vemos la configuración IPv4 del equipo conectado a Internet por medio de un modem

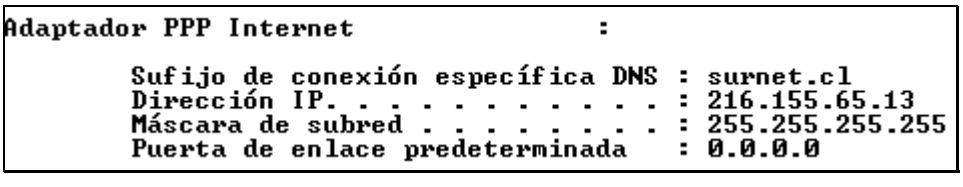


Figura 59

Ahora en la misma internase y su configuración 6to4:

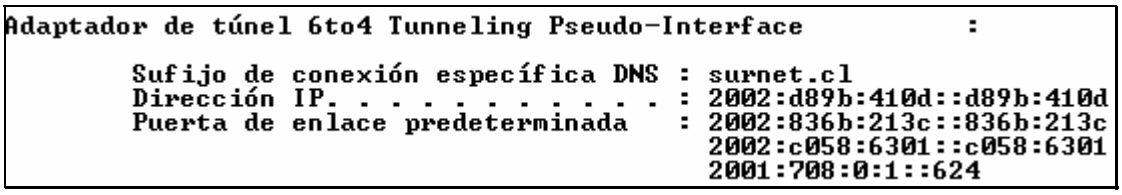


Figura 60

Se aprecia la configuración automática de las puertas de enlaces o router los cuales son encargados de manejar los paquetes 6to4. el paso siguiente es probar comunicación con los routers.

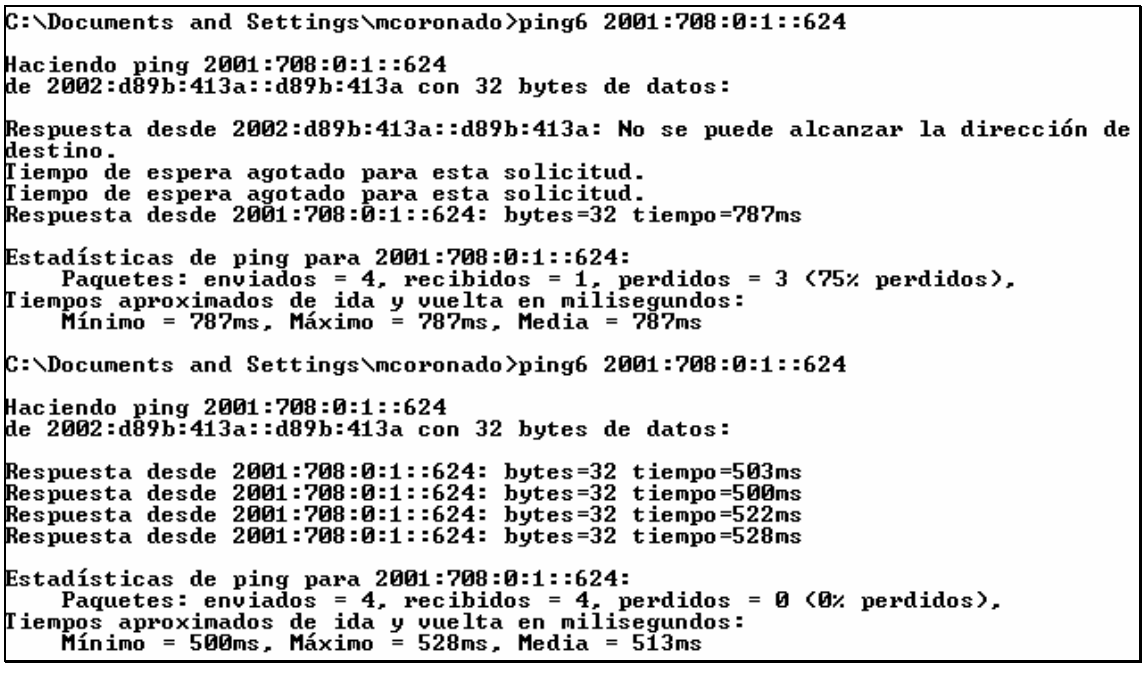


Figura 61

Se puede ver que en un primer intento se perdió un 75% de los paquetes IPv6 enviados, pero luego la comunicación es fluida.

Ahora probamos la comunicación con un sitio externo IPv6 para ver si el servidor 6to4 6to4.ipv6.funet.fi [2001:708:0:1::624] esta operando.

Probamos con el sitio www.kame.net es cual tiene habilitado el soporte para IPv6.

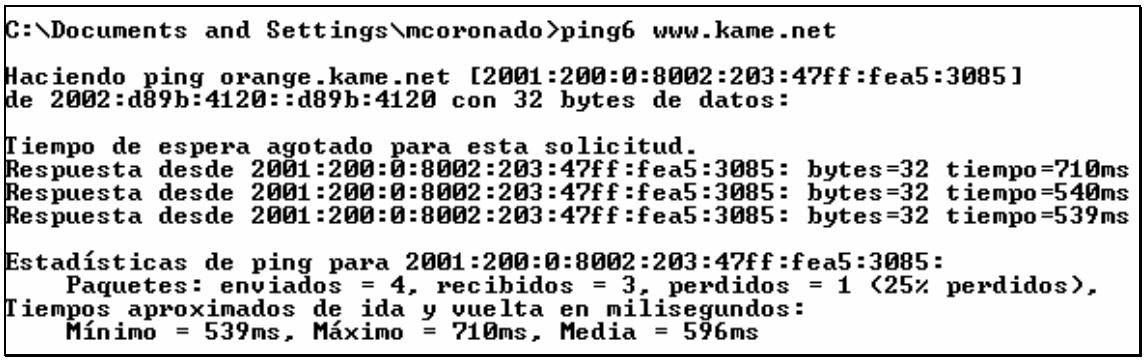


Figura 62

Con esta pequeña demostración, se ve lo fácil que es ya hoy en día obtener conectividad IPv6. Solo basta con habilitar el protocolo para que se este en condiciones de navegar sobre los sitios IPv6

Pseudo-Interface Automatic Tunneling: esta interfase utiliza una dirección compatible con IPv4, 0:0:0:0:0:0:w.x.y.z o ::w.x.y.z (donde w.x.y.z es la representación decimal con puntos de una dirección IPv4), la utilizan los nodos de pila dual que se comunican con Pv6 a través de una infraestructura IPv4. Cuando la dirección compatible con IPv4 se utiliza como destino IPv6, el tráfico IPv6 se encapsula de forma automática con un encabezado IPv4 y se envía al destino mediante la infraestructura IPv4.

Adicionalmente a estas interfases se instala el llamado **TEREDO IPv6**. Teredo IPv6 es una tecnología de transición que permite el establecimiento automático de túneles IPv6 entre hosts que se encuentran situados en diversos dispositivos NAT IPv4. La implementación de Microsoft de Teredo IPv6 se basa en la versión 8 del borrador del Grupo de trabajo de ingeniería de Internet (IETF, *Internet Engineer Task Force*), "Teredo: Tunneling IPv6 over UDP through NATs". Teredo posibilita que otros hosts de Internet se comuniquen directamente con un equipo y, en consecuencia, permite la comunicación directa.

Los métodos de transición no forman parte de los objetivos de este proyecto de tesis, por esto, no se profundizara más en esta línea.

Otra funcionalidad que entrega Windows XP y que es de gran importancia es ICF, *Internet Connection Firewall* el cual entrega las siguientes características para IPv6:

- Filtrado que conserva detalles de estado para el tráfico saliente
- El tráfico entrante no solicitado se descarta de forma automática y sin avisar
- Configuración de puertos: Cuando instala Advanced Networking Pack para Windows XP, ICF para IPv6 bloquea todos los puertos entrantes. Al habilitar el componente opcional Windows Peer-to-Peer Networking, los puertos 3540 (UDP) y 3587 (TCP) se abren para el tráfico entrante. Además, puede configurar manualmente los puertos de modo que acepten el tráfico no solicitado de la red. Por ejemplo, si aloja un servidor Web preparado para usar IPv6, puede configurar ICF para IPv6 de modo que permita el tráfico IPv6 no solicitado en el puerto TCP 80.
- Configuración ICMP: Puede configurar las opciones del Protocolo de mensajes de control de Internet (ICMP, *Internet Control Message Protocol*).
- Información de registro: Puede configurar el registro de los paquetes descartados, las conexiones que han podido establecerse o ambos. Puede usar los registros como ayuda en la solución de problemas de seguridad y rendimiento.

La configuración y administración del ICF es importante para el propósito de este proyecto de tesis por tanto se profundizara en ello.

Instalación de *Internet Connection Firewall*

ICF para IPv6 se habilita automáticamente cuando se habilita IPv6 en el equipo. No se precisa ninguna acción adicional. Sin embargo, si deshabilita ICF para IPv6, puede habilitarlo de forma manual. Para habilitar ICF para IPv6:

1. Inicio y, después, en Panel de control.
2. Agregar o quitar programas.
3. Agregar o quitar componentes de Windows. Se iniciará el Asistente para componentes de Windows.
4. Servicios de red (pero no desactive la casilla de verificación) y, después, haga clic en Detalles.
5. Active la casilla de verificación IPv6 Internet Connection Firewall y, después, haga clic en Aceptar.
6. Siga las instrucciones que aparecerán en la pantalla para instalar el componente en el equipo.

Una vez instalado se procede a la configuración, esto se realiza por medio de una herramienta de la línea de de comandos llamada **netsh,** la cual permiete la mostrar y modificar la configuración de red del de equipo:

1.- Abrir una ventana de comandos

2.- Ejecutar

netsh

3.- Ingresar a la configuración del firewall por medio del comando

firewall

4.- Listar la configuración de cada una de la interfaces por medio del comando

show adapter.

Ejemplo:

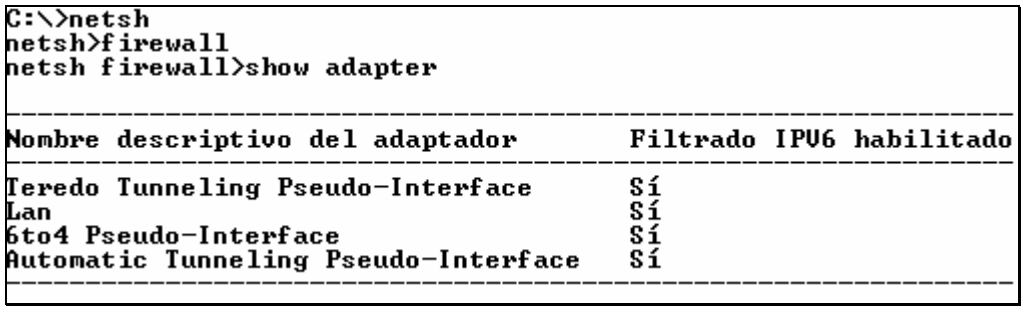


Figura 63

Existen otros comandos importantes como:

show adapter [Nombre_Apatador]: Muestra el estado de un adaptador especificado por *[Nombre_Apatador]*.

show globalport: despliega la configuración de los puertos globales del equipo también hay otras categorías como EffectivePort, OpenPort, IgnoredGlobalPort.

show logging: muestra la configuración de monitoreo de las interfaces.

Comandos importantes para establecer la configuración de los puertos que se dejaran abiertos y cuales se cerraran:

set : Permite establecer configuración.

set globalport : Establece la configuración de cualquier puerto bajo cualquier protocolo y su sintaxis es la siguiente

set globalport [port#=enable|disable] [name=name] [protocol=tcp|udp]

set adapter : permite habilitar una configuración en una interfase de red especifica, su sintaxis es la siguiente

set adapter [name][icmp type#=enable|disable][port port#=enable|disable[name=name][protocol=tcp|udp]]

[ignoreglobalport port#=enable|disable] [name=name] [protocol=tcp|udp]][filtering=enable|disable]

La configuración del sistema de seguridad de Windows XP es necesaria en la etapa de desarrollo de software, ya que por defecto cualquier puerto utilizado por una aplicación no conocida es filtrado, y puede causar problemas al momento de ejecutar aplicaciones IPv6.

Elementos de configuración de IPv6 en Windows

En el protocolo IPv6 para Windows XP se pueden configurar los elementos siguientes:

- Dirección IPv6
- Router predeterminado

Para resolver nombres de host en direcciones IPv6, es necesario agregar registros de recursos AAAA (cuádruple A) a la infraestructura DNS y configurar el Protocolo Internet (TCP/IP) con la dirección IP de al menos un servidor DNS. En posteriores secciones se hablara de la configuración de servidores DNS para IPv6.

Dirección IPv6

De forma predeterminada, las direcciones locales del vínculo se configuran automáticamente para cada interfaz de cada nodo IPv6 (host o router) con una dirección IPv6 local del vínculo. Si desea comunicarse con nodos IPv6 que no se encuentran en vínculos conectados, el host debe tener direcciones adicionales de unidifusión locales del sitio o globales. Las direcciones adicionales de los hosts se obtienen de anuncios de router

enviados por un router o se asignan manualmente. Las direcciones adicionales de los router se deben asignar manualmente.

Métodos de configuración de IPv6

El protocolo IPv6 para Windows XP puede utilizar los métodos de configuración siguientes:

I. Configuración manual
II. Configuración automática mediante direcciones sin estado

I. Configuración manual:

Puede configurar manualmente las direcciones y rutas IPv6 mediante la herramienta de la línea de comandos Ipv6.exe. La configuración manual puede ser necesaria en una red que tenga varios segmentos de red IPv6 en los que no haya routers IPv6 configurados para enviar anuncios de router. En una primera etapa configuraremos IPv6 manualmente para efectos de demostración.

Para asignar una dirección IPv6 a una interfase de red, el primer paso a realizar es obtener el índice de la interfase, esto se hace por medio del siguiente comando:

```
ipv6 if
```

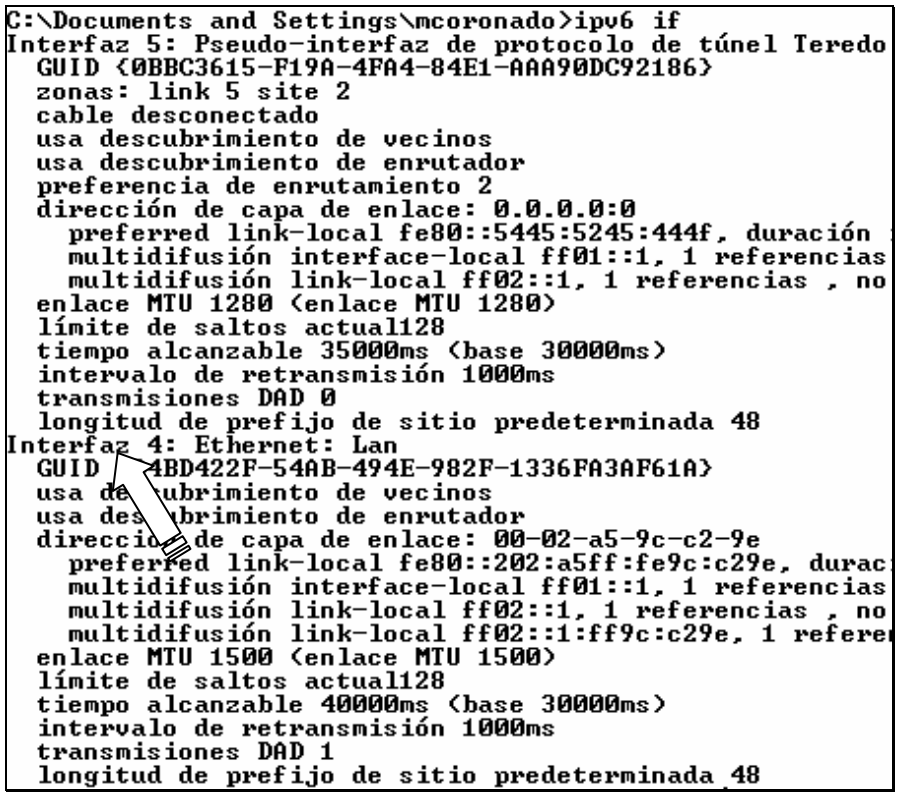


Figura 64

Luego se agrega la dirección IPv6 indicando la interfase a la cual se agrega por medio de índice

ipv6 adu [índiceDeInterfaz]/[dirección]

En el siguiente ejemplo agregaremos la dirección 3ffe:8040::1 a la interfase marcada por el Sistema Operativo con el numero 4.

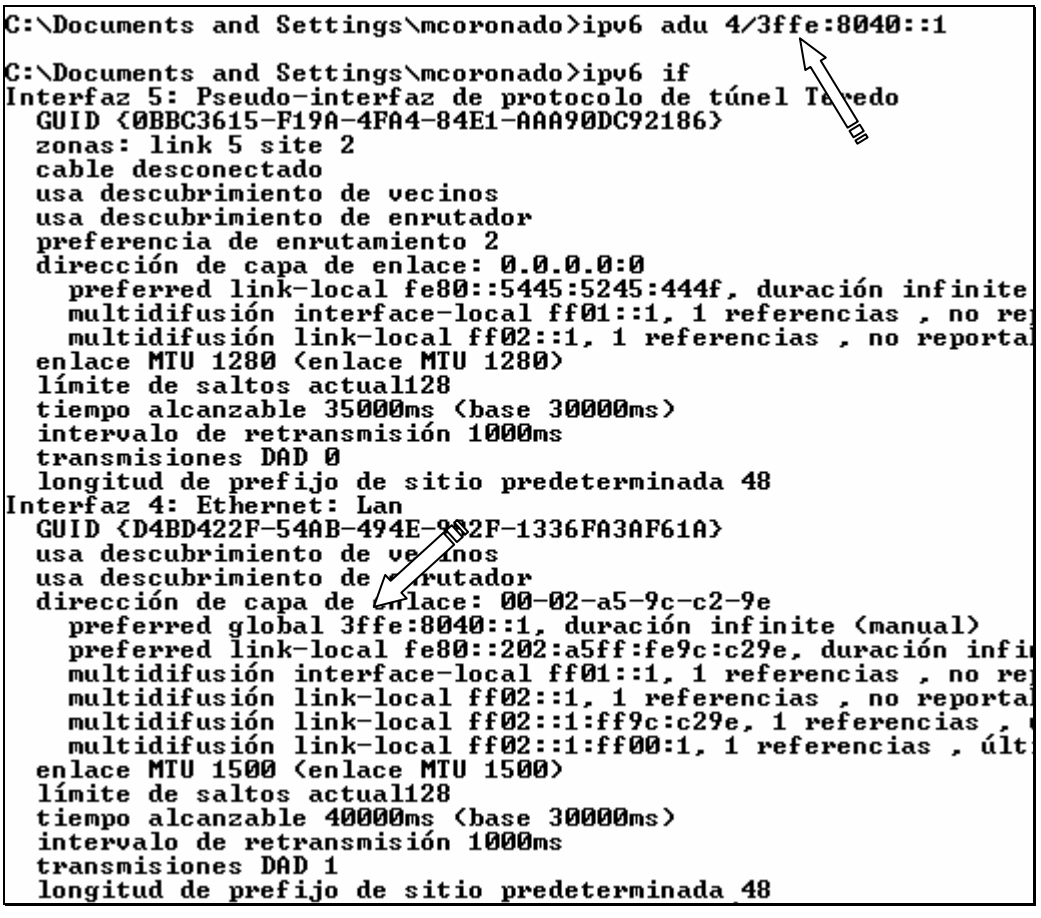


Figura 65

II. Configuración automática mediante direcciones sin estado

Existen tres tipos de configuración automática como se describió en el capitulo 1: con estado, sin estado, ambas. En Windows XP, el protocolo IPv6 no admite el uso de un protocolo de configuración de direcciones con estado.

El proceso de configuración automática de direcciones en un nodo IPv6 se produce de la manera siguiente:

a. Se deriva una dirección local del vínculo, basada en el prefijo local del vínculo de FE80::/64 y el identificador de interfaz de 64 bits.

b. Se lleva a cabo la detección de direcciones duplicadas para comprobar la exclusividad de la dirección local del vínculo

c. Si se produce un error en la detección de direcciones duplicadas, se debe realizar la configuración manual en el nodo.
d. Si la detección de direcciones duplicadas tiene éxito, la dirección local del vínculo se considera única y válida. La dirección local del vínculo se inicializa para la interfaz.
e. El host envía un mensaje de solicitud de router.
f. Si no se reciben mensajes de anuncio de enrutador, el host utilizará un protocolo de configuración de direcciones con estado para obtener las direcciones y otros parámetros de configuración. En esta paso windows XP no tiene soporte para el uso de direcciones automáticas con estado.
g. Si se recibe un mensaje de anuncio de router, se establece en el host la información de configuración incluida en el mensaje.

Para que Windows XP SP1 envíe los mensajes de solicitud de router se debe ejecutar el siguiente comando:

ipv6 ifc índiceDeInterfaz advertises

En caso contrario para que no se solicite el anuncio de router el comando es el siguiente

ipv6 ifc índiceDeInterfaz -advertises

Estos comando no los podemos ejemplificar todavía puesto que falta un router que proporcione el servicio de auto configuración de nodos, es por esto que la ejemplificación se dejara para secciones posteriores cuando se configure router Linux.

16.1.2. Microsoft Windows 2003 Server Enterprise Edition

La configuración de Windows 2003 server no dista demasiado con respecto a Windows XP SP1. En el caso de la instalación de la pila del protocolo, es exactamente igual a Windows XP

La gran diferencia se aprecia al momento de la configuración. Para la administración del protocolo IPv6 de las interfaces de red Windows 2003 ya no proporciona la utilidad IPv6.exe, es por esto que las configuraciones deben hacerse por medio del Network Shell (netsh), utilidad de línea de comandos que usa secuencias de comandos para componentes de red de Windows 2003 en equipos locales y remotos.

- Configuración de Direcciones Manuales:

Para poder ver el estado de la interfase de red se debe ingresar al network shell por medio del comando

```
Netsh
```

Para probar si el protocolo esta correctamente instalado, usar:

```
interface ipv6 show address
```

Se mostrará la configuración y las direcciones IPv6 adquiridas (auto-configuradas) para cada interfaz de red existente

“netsh interface ipv6” se puede utilizar para comprobar y configurar manualmente interfaces, direcciones y rutas (usuarios avanzados).

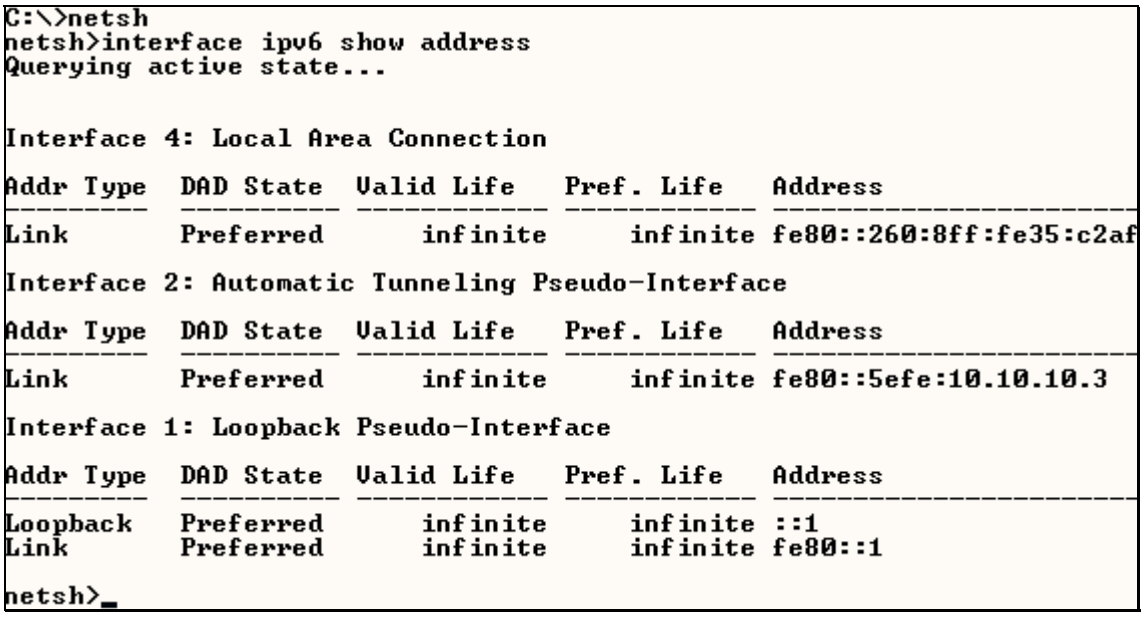


Figura 66

El comando *ipconfig* también funciona para el despliegue de información IPv6 pero no es tan detallada ver siguiente figura:

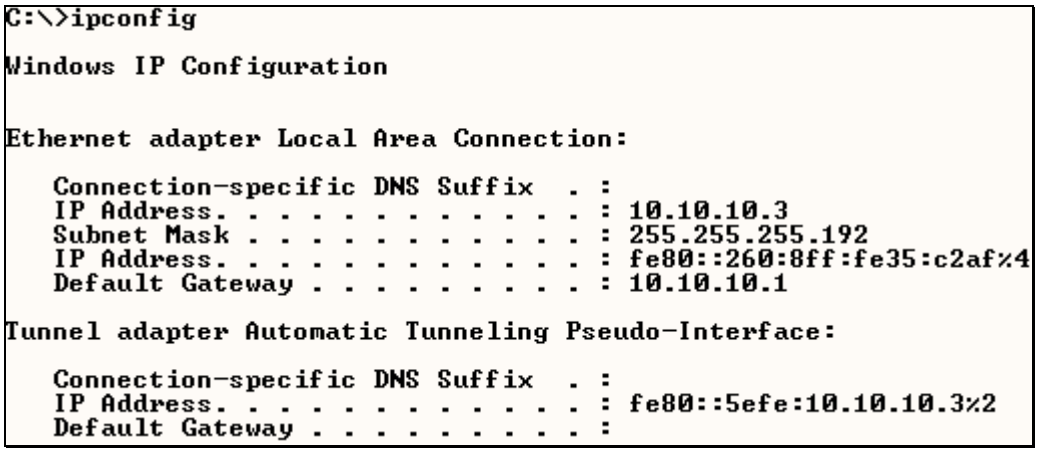


Figura 67

Se puede comprobar el correcto funcionamiento de la pila ipv6 con:

Ping ::1

Observar que no es necesario utilizar un comando ping diferente para direcciones ipv6

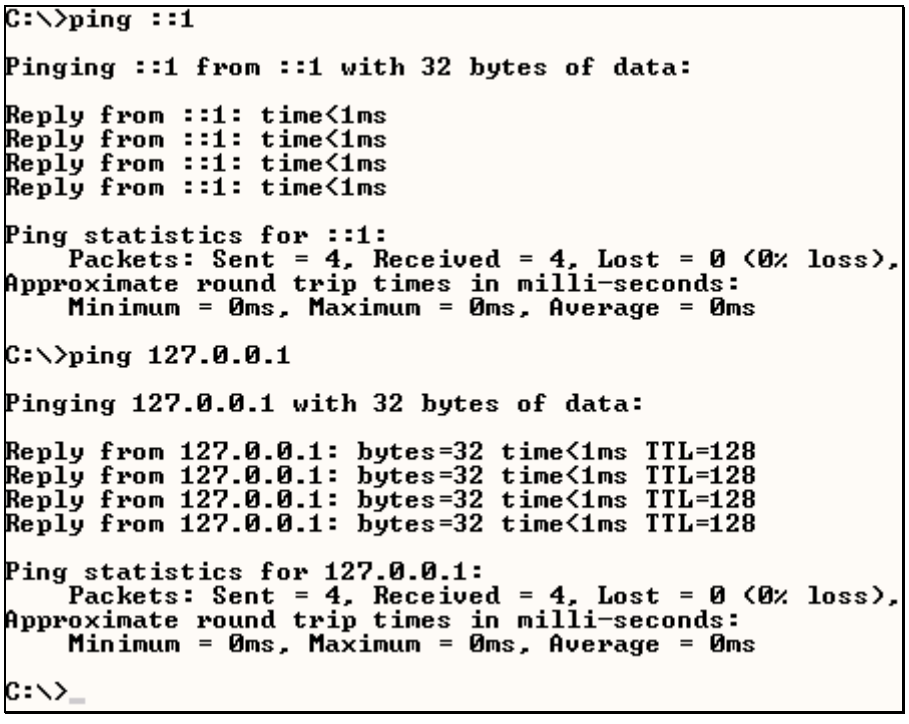


Figura 68

A continuación se agregara al PC2 la dirección 3ffe:8040::2 en forma manual, para efectuar pruebas de conectividad con el PC1 que fue configurado en la sección anterior.

Primero ingresar al network shell

Netsh

Luego,

interface ipv6

add address [interface =]String [address=] IPv6Address

Donde:

[interface=]String

Específica el nombre de la interfaz.

[address=]IPv6Address

Especifica la dirección IPv6.

En nuestro caso la configuración seria:

```
C:\>netsh
netsh>interface ipv6
netsh interface ipv6>show interface
Querying active state...

Idx  Met   MTU    State          Name
---  ----  -----  -------------  -----
  4    0   1500  Connected      Lan
  3    1   1280  Connected      6to4 Pseudo-Interface
  2    1   1280  Connected      Automatic Tunneling Pseudo-Interface
  1    0   1500  Connected      Loopback Pseudo-Interface

netsh interface ipv6>add address interface=Lan address=3ffe:8040::2
Ok.

netsh interface ipv6>_
```

Figura 69

Ahora se comprueba la configuración,

```
netsh interface ipv6>show address
Querying active state...


Interface 4: Lan

Addr Type  DAD State  Valid Life   Pref. Life   Address
---------  ---------- ------------ ------------ -------------------------
Manual     Preferred      infinite     infinite 3ffe:8040::2
Link       Preferred      infinite     infinite fe80::260:8ff:fe35:c2af

Interface 2: Automatic Tunneling Pseudo-Interface

Addr Type  DAD State  Valid Life   Pref. Life   Address
---------  ---------- ------------ ------------ -------------------------
Link       Preferred      infinite     infinite fe80::5efe:10.10.10.3

Interface 1: Loopback Pseudo-Interface

Addr Type  DAD State  Valid Life   Pref. Life   Address
---------  ---------- ------------ ------------ -------------------------
Loopback   Preferred      infinite     infinite ::1
Link       Preferred      infinite     infinite fe80::1

netsh interface ipv6>_
```

Figura 70

Probamos conectividad con PC2 [3ffe:8040::1]

```
C:\>ping 3ffe:8040::1

Pinging 3ffe:8040::1 from 3ffe:8040::2 with 32 bytes of data:

Reply from 3ffe:8040::1: time=3ms
Reply from 3ffe:8040::1: time<1ms
Reply from 3ffe:8040::1: time<1ms
Reply from 3ffe:8040::1: time<1ms

Ping statistics for 3ffe:8040::1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 3ms, Average = 0ms
```

Figura 71

- Configuración de Direcciones Automáticas

La configuración de direcciones automáticas para Windows 2003 se efectúa de la misma forma que en Windows XP SP1, con una pequeña diferencia en el comando utilizado:

```
netsh interface ipv6 set interface advertise={enabled | disabled}
```

Al igual que en Windows XP SP1 la ejemplificación se efectuara cuando este configurado el router linux puesto que ningún sistema operativo de la familia Windows proporciona el servicio de Router Advertisement para la configuración automática de nodos

16.2. Implementación IPv6 Linux

16.2.1. Linux Kernel 2.6

En la plataforma Linux el protocolo IPv6 esta bastante desarrollado, a partir del kernel 2.2 se ha hecho esfuerzos por brindar este soporte y hoy en día ya se cuanta con la versión 2.6 del kernel, proporcionándose al usuario una gran cantidad de opciones IPv6 para configurar y utilizar en medios productivos.

Además de los esfuerzos que realizan los encargados del desarrollo del kernel, existen paralelamente otros trabajos que están focalizados directamente en el desarrollo del protocolo IPv6, en este proyecto de tesis de trabajo con el kernel normal entregado en la distribución y uno especializado en IPv6 (kernel del proyecto USAGI).

A continuación se procede a la configuración del equipo con un kernel normal de tal manera de unirlo a la ya dos maquinas Windows configuradas con ipv6 en la sección anterior.

16.2.2. Configuración del Kernel

El primer paso para la configuración de una maquina Linux es chequear si el kernel soporta el nuevo protocolo. La manera formal de hace esto es examinando si es posible cargar el modulo IPv6 mediante el comando, si el modulo no puede ser cargado se debe compilar el kernel

modprobe ipv6

El cual carga de modo temporal el modulo de ipv6 que esta disponible en

/lib/modules/[version_del_kernel]/kernel/net/ipv6/

Al cargar el modulo diversos archivos son creados en el sistema de archivos /proc/net:

Archivo /proc/net/if_inet6: almacena las direcciones asignadas a cada dispositivo de red, al iniciar tiene la asignación de las direcciones asignadas en mase a la dirección MAC.

Archivo /proc/net/igmp6: contiene la información de los grupos multicast a los cuales pertenece el equipo

Archivo /proc/net/ip6_flowlabel: contiene la lista de todas las etiquetas de flujo asignadas al los tráficos IPv6

Archivo /proc/net/ipv6_route: contiene todas las entradas de la tabla de ruta IPv6 del equipo

Archivo /proc/net/raw6: Estadisticas del dispositivo en bruto(raw) (IPv6).

Archivo /proc/net/rt6_stats: Estadísticas globales de las tablas de ruta IPv6.

Archivo /proc/net/snmp6: Datos sobre SNMP (IPv6).

Archivo /proc/net/sockstat6: Estadisticas de sockets (IPv6).

Archivo /proc/net/tcp6: Sockets TCP (IPv6): Sockets TCP (IPv6).

Archivo /proc/net/udp6: Sockets UDP (IPv6).

Como el modulo del kernel fue cargado temporalmente estos archivos estarán disponibles solo hasta que la maquina sea reiniciada. Para cargar el modulo IPv6 de forma permanente en el equipo es necesario agregar la siguiente línea en /etc/sysconfig/network

NETWORKING_IPV6=yes

Generándose direcciones IPv6 link-local permanentes en base a las direcciones MAC, con esto el equipo PC3 estaría en condiciones de recibir trafico IPv6.

El pasó siguiente es asignar las direcciones IPv6 globales a las interfaces. El equipo PC3 con el kernel 2.6 será configurado como “router del sitio” para eso necesario asignarle una dirección fija, para ello se utiliza la primera dirección disponible [3FFE:8040::1].

Para configurar una dirección estática, es necesario agregar en el archivo /etc/sysconfig/network-script/ifcfg-eth0 una línea con la dirección, según el siguiente formato:

IPV6INIT=yes

IPV6ADDR=<IPv6 address>[/<prefix length>]

Entonces

IPV6ADDR=3FFE:8040::1/64 En el caso de nuestro router

Una vez configuradas las direcciones IPv6 el siguiente paso es el descubrimiento de vecinos [Neighbor Discovery] definido en el RFC 2461

Configuración de Neighbor Discovery

Para configurar nuestro equipo como router debemos informarle al sistema operativo que esta maquina funcionara como tal, para ello agregamos la siguiente línea en /etc/sysconfig/network.

IPV6FORWARDING=yes.

La configuración del demonio de aviso de router [radvd] permite la auto configuración de los equipos IPv6 de la red, habilitando nuestro equipo (PC3) para enviar los mensajes de aviso de router [RFC 2461] periódicamente y cuando es requerido por algún nodo de la red.

El software radvd necesario para instalar esta funcionalidad en equipos Linux esta disponible en [http://v6web.litech.org/radvd/].

Una vez instalado el RPM se requiere establecer las opciones del demonio en el archivo [/etc/radvd.conf], las opciones que podemos configurar son:

- El prefijo de la red
- Duración del prefijo de red
- Los intervalos de envió
- La interfase de red por la cual se envían los anuncios

En el caso de nuestro router el archivo de configuración es el siguiente:

```
interface eth1
{
        AdvSendAdvert on;
        MinRtrAdvInterval 3;
        MaxRtrAdvInterval 10;
        AdvHomeAgentFlag off;
        prefix 3ffe:8040::/64
        {
                AdvOnLink on;
                AdvAutonomous on;
        };
};
```

Interface: Especifica la interfase por la cual se publicarán los anuncios de router

AdvSendAdvert: Permite habilitar o deshabilitar el envío de los mensajes de anuncio de router

MaxRtrAdvInterval: especifica el máximo tiempo en segundos entre mensajes no solicitados, sus valores es recomendable que esten entre los 4 y 1800 segundos

MinRtrAdvInterval: especifica el mínimo tiempo en segundos entre mensajes de anuncio de router solicitados y no solicitados, los valores recomendables son entre 3 y 0,75* MaxRtrAdvInterval.

Prefix: especifica el prefijo a ser publicado por la interfase.

AdvAutonomous: el prefijo puede ser usado en la configuración de direcciones autónomas [RFC 2462]

Con esta configuración todos los clientes PC1, PC2 obtiene el prefijo [3FFE:8040::] y generan direcciones IPv6 en base al prefijo y su dirección MAC.

```
Adaptador Ethernet Lan          :

        Sufijo de conexión específica DNS : mcoronado.cl
        Dirección IP. . . . . . . . . . . : 10.10.10.2
        Máscara de subred . . . . . . . . : 255.255.255.0
        Dirección IP. . . . . . . . . . . : 3ffe:8040::4cc9:bd61:d647:45ab
        Dirección IP. . . . . . . . . . . : 3ffe:8040::202:a5ff:fe9c:c29e
        Dirección IP. . . . . . . . . . . : fe80::202:a5ff:fe9c:c29e%8
        Puerta de enlace predeterminada   : 10.10.10.1
                                            fe80::260:8ff:fe35:c2af%8
```

Ya tenemos la configuración automática de direcciones para todos los equipos, por tanto podemos establecer la configuración del DNS, el cual es fundamental para el desarrollo de las aplicaciones.

16.2.3. Configuración Kernel Usagi

Debido a las razones expuestas en capítulos anteriores donde se hace referencia ala mayor madures del kernel Usagi para Linux en materia de programación es que se ha decidido usar este kernel para el desarrollo de aplicaciones Multicast en uno de los equipos de nuestro laboratorio (PC4). A continuación veremos la instalación y compilación del mismo.

Instalación:

En primer lugar tenemos que descargar la última versión del kernel Usagi desde [www.linux-ipv6.org]. Luego los pasos son:

- Descargar y descomprimir el paquete con el nombre usagi-linux24-stable-[XXXXYYMM].tar.gz, automáticamente se descomprime en la carpeta usagi
- Dentro de la carpeta usagi ejecutar:

  make prepare TARGET=linux24

  Con esto se busca estructurar el kernel 2.4 para ser instalado puesto que también esta disponible para el kernel 2.2 (TARGET=linux24)
- Ingresar al directorio donde esta el nuevo kernel (usagi/kernel/linux24) y comenzar la compilación.
- Ejecutar

  make mrproper
- Ejecutar make menuconfig o make xconfig o make menuconfig
- Dentro del menú de configuración del kernel Usagi las opciones que se deben habilitar para que se tenga soporte ipv6 son: En el menú [*Code maturity level options*], seleccionar [*y*] en [*prompt for development and/or incomplete code/drivers*], al seleccionar esta

opción se habilitaran todos los menús correspondientes a módulos que actualmente están en desarrollo entre los que se encuentra el modulo IPv6

- En el menú [Networking options] (11º menu), seleccionar [y] en [The IPv6 protocol (EXPERIMENTAL)], al seleccionar esta opción se agrega al kernel el modulo IPv6, el cual es habilitado al inicio del sistema operativo con lo cual no es necesario cargar el modulo, otra opción que es de vital importancia para el desarrollo de aplicaciones es [*IPv6: Verbose debugging messages*], ya que permite disponer de los mensajes del kernel referentes al stack IPv6. Una opción que es muy importante desde el punto de vista de la seguridad pero que en ambientes de desarrollo puede no ser relevante es [*IPv6: drop packets with fake ipv4-mapped address(es)*], como su nombre lo indica permite eliminar falsos paquetes IPv4-Mapeados, lo que comúnmente no es filtrado por sistemas de seguridad.

Otra opción que también tiene relación con la seguridad es [IPv6: Restrict “double binding” only for same user], con esto se busca eliminar la posibilidad de que dos usuarios intenten utilizar un mismo puerto pero con distintos protocolos IP.

[IPv6: anycast suport] habilita al kernel para la utilización de direcciones anycast.

Luego tenemos 5 opciones de debugging que son importante de tener en cuenta ya que permiten contar con información adicional al momento del desarrollo, pudiendo así detectar errores. Estas son [IPv6: Neighbor Discovery

debugging], [IPv6:Address Autoconfiguration debugging], [IPv6: Debug on sourse address selection], [IPv6: Routing Information debugging], [IPv6, Multicast Listener Discovery debugging].

Estas son todas las opciones referentes a ipv6 que fueron analizadas y probadas en el desarrollo de este proyecto de tesis. Opciones que apuntan a simplificar y habilitar el desarrollo de aplicaciones IPv6.

- Una vez seleccionados los paquetes se recomienda hacer respaldo del archivo de opciones del kernel que se genero, con el objetivo de facilitar cambios en el futuro.
- Editar el archivo makefile, en la línea **EXTRAVERSION** cambie el valor para compilar el kernel en directorios distinto al actual, de tal manera de recuperar el anterior en caso de error, por ejemplo yo cambié EXTRAVERSION=-22, por EXTRAVERSION="22-USAGI".
- Ejecutar los siguientes comando para compilara el kernel y los módulos

```
make clean
make bzImage
make modules
```

- Ejecutar los siguientes comandos para instalar el nuevo kernel y sus módulos

```
make install
make modules_install
```

Si todo sale bien ya esta disponible el nuevo kernel en el menú de inicio del gestor de arranque.

Ahora debemos instalar las librerías de usagi para el desarrollo de aplicaciones con características avanzadas.

Instalación Librerías USAGI

Para el desarrollo de aplicaciones el proyecto USAGI a desarrollado sus propias librerías, con funciones especialmente diseñadas para el soporte IPv6, los que será detallado con mayor profundidad en el siguiente capitulo.

Estas librerías remplazan a las que trae Linux por defecto por lo que es recomendable hacer un respaldo de las librerías actuales.

tar -cvzf include.orig.tar.gz /usr/include

Dentro del paquete USAGI excite un script que hace el cambio automático de las librerías

En el directorio /usagi/usagi/libinet6

make install-includes

Al momento de de la compilación del la aplicación se debe utilizar la siguiente opción

-L /usr/local/v6/lib -linet6

Luego la configuración de las interfaces y la habilitación de IPv6 son idénticas a la configuración y habilitación de un equipo con Kernel 2.6 normal.

Con esto tenemos nuestro laboratorio configurado completamente en IPv6, listo para efectuar el desarrollo de aplicaciones en las diferentes plataformas de Sistemas Operativos.

Después del análisis y desarrollo de aplicaciones de prueba, se llega a la conclusión de la fuerte dependencia que existe entre el protocolo la aplicaciones y el servicio de DNS, por lo anterior se decide montar un servidor DNS en la red de prueba, en la siguiente sección se detalla el procedimiento a seguir.

16.2.4. Configuración Servicio DNS

Para la configuración del servicio de DNS dentro de la red de prueba, se debe elegir una de las dos plataformas, en este sentido la elección no es muy difícil puesto que la versión de servidor DNS de Windows es bastante limitado al momento de crear un DNS con soporte IPv6, puesto que solo permite agregar registros tipo AAAA, no permite la creación de zonas inversas o un servidor DNS con IPv6 nativo, lo que hace mucho mas viable la configuración de un equipo con BIND 9 corriendo sobre la plataforma Linux, el cual es independiente del kernel que se este utilizando.

Para configurar el servicio DNS con soporte para ipv6 con todas las características que tiene un servidor DNS (zonas IPv4/IPv6, zonas inversas IPv4/IPv6, transferencia de zonas, etc.), se recomienda el uso de Berkeley Internet Name Domain version 9 [B IND - http://www.isc.org/sw/bind], además de dos paquetes adicionales que facilitan la configuración:

dnsutils: pequeñas utilidades como "dig" y "nslookup" para hacer pruebas de DNS. Disponible dentro del paquete del Bind 9.

ipv6calc: Esto no es imprescindible, pero es de gran ayuda para convertir direcciones ipv6 a diferentes "formatos". Disponible en [http://www.deepspace6.net/projects/ipv6calc.html]

rpm –i bind-9.x.x-xx.i386.rpm

rpm –i binutils-2.xx.xx.x.xx-x.i386.rpm

rpm –i ipv6calc-x.xx-1.i386.rpm

La instalación de estas no representa mayor dificultad, y no requiere parámetros adicionales para dar soporte a IPv6.

En el laboratorio IPv6 el servicio DNS se ejecutara sobre la maquina con kernel Linux Standard 2.6

16.2.4.1. Configuración General

La configuración general del Bind 9 se hace en el archivo /etc/named.conf. Para creara zonas DNS IPv6 la forma de declaración es la misma que para zonas IPv4, distinto es el caso para zonas inversas IPv6, para ejemplificar de mejor forma la configuración se crearan 3 zonas.

a.- una zona para el dominio tesismcast.ipv6.cl para IPv4 e IPv6

b.- Una zona inversa para red 10.10.10.0/24

c.- Una zona Inversa para la red 3ffe:8040::/64

Definición de las zonas en /etc/named.conf

Primero se define las opciones globales, el puerto por el cual se efectuaran las consultas DNS por parte de los clientes, es importante especificar ambos puertos tanto para IPv4 e IPv6 ya que en el caso que se tenga IPv6 nativo en la red es necesario este en forma explicita el numero de puerto, en caso contrario las consultas DNS no funcionaran.

```
#Opciones Generales
options{
        listen-on port 53 { any; };
        listen-on-v6 port 53 {any;};
        directory "/var/named/";
};
```

Ahora la configuración de las tres zonas

```
#DNS de Solo Cache
zone  "." {
        type hint;
        file  "named.cache";
};

#Reverso Red Local 10.10.10.0
zone  "10.10.10.in-addr.arpa" {
        type master;
        file  "10.10.10.in-addr.arpa.zone";
};

#Nombres para el dominio tesismcast.ipv6.cl
zone "tesismcast.ipv6.cl" {
        type master;
        file "tesismcast.ipv6.cl.zone";
};

#Reverso Ipv6 arpa
zone "0.0.0.0.0.0.0.0.0.4.0.8.e.f.f.3.ip6.arpa" {
        type master;
        file "reversos-ipv6";
        allow-transfer { any;};
        allow-query { any; };
};

#Reverso Ipv6 int
zone "0.0.0.0.0.0.0.0.0.4.0.8.e.f.f.3.ip6.int" {
        type master;
        file "reversos-ipv6";
        allow-transfer { any;};
        allow-query { any; };
};
```

En Primer lugar se ingresa las entrada para la zona raiz (.), esta no sufre cambios en IPv6.

Luego tenemos la configuración de reverso para una zona IPv4

En tercer lugar tenemos la entrada para un dominio en cuyo archivo de zona contendrá direcciones IPv6, pero la declaración de la zona no presenta diferencias para soportar el nuevo tipo registros.

Para el DNS IPv6 inverso si hay cambios al momento de declarar la zona y existes dos formas de declarar la zonas, el formato INT es el más soportado. ARPA es el último formato pero no está tan difundido como INT. En ambos las diferencias son minimas, pero no menos importantes.

En el caso de [0.0.0.0.0.0.0.0.0.4.0.8.e.f.f.3.ip6.arpa] corresponde al prefijo que estamos configurando [3ffe:8040::/64] escrito en sentido contrario.

Esta nomenclatura es especialmente difícil de generar si es que el prefijo esta abreviado los que puede llevar a errores que impidan que el servidor DNS funcione correctamente, es por esto que se recomienda el uso de ipv6calc

Comando: ipv6calc -a 3ffe:8040::/64

Salida: 0.0.0.0.0.0.0.0.4.0.8.e.f.f.3.ip6.arpa

En cuanto a crear dos registros, el .int y el .arpa, tiene una explicación histórica: ip6.int se utilizaba al principio, al igual que A6, pero al día de hoy está cayendo en desuso y volviéndose obsoleto, dentro de BIND ambos pueden convivir sin problemas y pueden tener el mismo archivo definición de zona.

Para comprobar la correcta sintaxis del archivo de configuración named.conf se puede usar el comando

named-checkconf [-t directorio] named.conf

16.2.4.2. Definición de los archivos de zona

Para permitir que el servidor DNS entregue direcciones ipv6, se pueden utilizar dos tipos de registros de recursos (RR) en los archivos de zonas, en primer lugar se tiene el registro AAAA, en segundo lugar esta el A6, hoy en día solo se usa este ultimo, pero se debe tener cuidado ya que hay clientes DNS que no soportan aún este ultimo Registro de Recursos.

Registro de recursos AAAA: este registro es el análogo al registro tipo A en IPv4, y su formato es el siguiente

[nombre Host] ttl IN AAAA [direccion IPv6]

El uso de este tipo de Registro de recursos no es recomendable a menos que sea absolutamente necesario, en el caso que se tengan aplicaciones IPv6 antiguas

Registro de Recursos A6: el registro A6 es mucho más flexible que el registro AAAA ya que tiene múltiples opciones, la más importante es la posibilidad de formar cadenas de registros A6, cada una con uno especifica parte de la dirección IPv6.

En la configuración del DNS dentro del Laboratorio el servidor DNS tiene la siguiente configuración:

```
$TTL 86400
@               IN      SOA     negro.tesismcast.ipv6.cl.  mcoronado.uach.cl. (
                                                2003161033; seria (by mcoronado)
                                                28800 ; refresh
                                                7200 ; retry
                                                604800 ; expire
                                                86400) ; ttl
                IN      TXT     "Tesis IPv6 Multicast"
                IN      NS      negro.tesismcast.ipv6.cl.
negro           IN      A       10.10.10.3
negro6          IN      A6      0       3ffe:8040::2a0:24ff:fe54:2f00
notebook        IN      A       10.10.10.2
notebook6       IN      A6      0       3ffe:8040::1
clazo           IN      A6      3ffe:8040::2e0:4cff:fe9d:6d4a
```

Ahora se cuenta con un servidor DNS dentro del Laboratorio de pruebas IPv6, el siguiente paso es proporcionar resolución inversa

16.2.4.3. Definición de los archivos de zona inversa

Para proporcionar la resolución inversa de ipv6 existen dos formatos, la primera y que ya esta descontinuada solo se mantiene para la compatibilidad hacia atrás se denomina *formato Nibble.* El segundo y que en la actualidad se esta utilizando se denomina *formato Bitstring.*

La diferencia entre ellos radica en la representación de las direcciones IPv6 dentro del archivo de configuración, es recomendable tener los dos tipos de configuración dentro del servidor DNS porque hay aplicaciones que aún están basadas en el formato antiguo.

Formato Nibble:

Se anota la dirección ipv6 sin el prefijo y cada digito separado por un punto ejemplo:

$ORIGIN 0.6.8.1.1.0.2.0.0.5.0.8.e.f.f.3.ip6.int.

1.0.0.0.0.0.0.0.0.0.0.2.4.0.0 IN PTR host.example.com.

Formato Bitstring

En este caso la dirección se anota completa pero entre corchetes ejemplo:

$ORIGIN \[x3ffe805002011860/64].ip6.arpa.

\[x0042000000000001/64] IN PTR host.example.com.

Anexo 1 - IP Versión 4

17. El modelo OSI - TCP/IP

Aunque el modelo de referencia OSI sea universalmente reconocido, el estándar abierto de Internet desde el punto de vista histórico y técnico es el **Protocolo de control de transmisión/Protocolo Internet (TCP/IP)**. El modelo de referencia **TCP/IP y la pila de protocolo TCP/IP** hacen que sea posible la comunicación entre dos computadores, desde cualquier parte del mundo, a casi la velocidad de la luz. TCP/IP es compatible con cualquier sistema operativo y con cualquier tipo de hardware, proporcionando una abstracción total del medio.

El modelo TCP/IP está basado en el tipo de red packet-switched (de conmutación de paquetes), y tiene cuatro capas: la capa de aplicación, la capa de transporte, la capa de Internet y la capa de red. Es importante observar que algunas de las capas del modelo TCP/IP poseen el mismo nombre que las capas del modelo OSI, aunque no se corresponden exactamente unas con otras, por lo que no deben confundirse.

17.1. Visión General de la Arquitectura TCP/IP

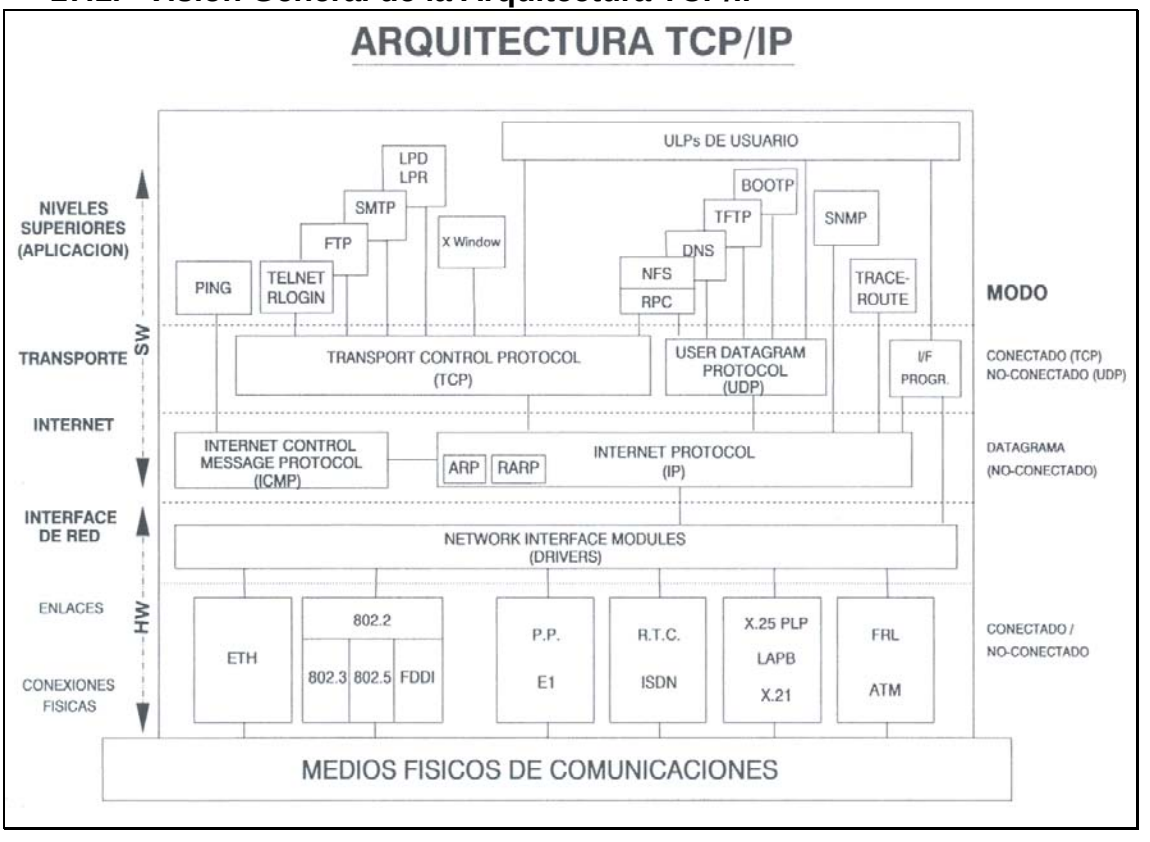


Figura 72

17.2. El Modelo de 4 Capas

17.2.1. Capa de aplicación

Los diseñadores de TCP/IP sintieron que los protocolos de nivel superior deberían incluir los detalles de las capas de sesión y presentación. Simplemente crearon una capa de aplicación que maneja protocolos de alto nivel, aspectos de representación, codificación y control de diálogo. El modelo TCP/IP combina todos los aspectos relacionados con las aplicaciones en una sola capa y da por sentado que estos datos están correctamente empaquetados para la siguiente capa.

17.2.2. Capa de transporte

La capa de transporte es la responsable del envío y la recepción de los segmentos de datos de la capa de aplicación. Esta capa ofrece a la capa de aplicación, dos servicios. Un servicio que consiste en el envío y recepción de datos

orientado a conexión y otro que consiste en el envío y recepción de datos no orientados a conexión.

El protocolo TCP "Transmition Control Protocol" ofrece un circuito virtual entre aplicaciones de usuario final. Sus características son las siguientes:

- Orientado a conexión
- Confiable
- Divide los mensajes salientes en segmentos
- Reensambla los mensajes en la estación destino
- Vuelve a enviar lo que no se ha recibido
- Reensambla los mensajes a partir de segmentos entrantes.

El protocolo UDP "User Datagram Protocol" transporta datos de manera no confiable entre hosts. Las siguientes son las características del UDP:

- No orientado la conexión
- Poco confiable
- Transmite mensajes (llamados datagramas del usuario)
- No ofrece verificación de software para la entrega de segmentos (poco confiable)
- No reensambla los mensajes entrantes
- No utiliza acuses de recibo
- No proporciona control de flujo

17.2.3. Capa de Internet o de red

El propósito de la capa de Internet es enviar paquetes desde cualquier red en Internetwork de redes y que estos paquetes lleguen a su destino independientemente de la ruta y de las redes que se utilizaron para llegar hasta ahí.

En esta capa se produce la determinación de la mejor ruta y la conmutación de paquetes. Durante su transmisión los paquetes pueden ser divididos en fragmentos, que se montan de nuevo en el destino.

Para poder enrutar los datagramas de la capa de transporte, éstos se encapsulan en unidades independientes, en las que se incorporan diferentes datos necesarios para el envío, como dirección de origen del datagrama, dirección de destino, longitud del mismo.

En una comunicación con arquitectura TCP/IP ambos host pueden introducir paquetes en la red, viajando estos independientemente de cual sea su destino. Por ello, no hay garantía ninguna de entrega de los paquetes ni de orden en los mismos.

17.2.4. Capa de acceso a la red

El nombre de esta capa es muy amplio y se presta a confusión. También se denomina capa de host a red. Es la capa que se ocupa de todos los aspectos que requiere un paquete IP para realizar realmente un enlace físico.

Uno de los principales elementos que maneja esta capa es el de las direcciones físicas, números únicos de 6 bytes asignados a cada tarjeta de red, y que son el medio principal de localización de un host dentro de una red. Cada tarjeta tiene un número identificador, cuyos 3 primeros bytes son asignados por el fabricante de la misma, mientras que los otros 3 se asignan de forma especial. Cuando un host debe enviar un paquete a otro de su red busca a éste mediante su número de tarjeta de red (dirección física).

17.3. Definición del Nivel IPv4

El protocolo IP surgió para interconectar redes. El principal trabajo de IP es buscar una ruta para que los datos lleguen al destino. Con respecto al modelo OSI (7 capas) podemos ubicar a este protocolo en el tercer nivel (capa de red).

IP es el protocolo primario de conexión responsable del envío y enrutamiento de paquetes entre maquinas (*hosts*).

IP no establece una sesión antes de intercambiar datos. IP no es fiable debido a que trabaja sin garantía de entrega. A lo largo del camino, un paquete puede perderse, cambiarse de secuencia, duplicarse, retrasarse, o incluso dividirse.

IP no requiere una comunicación ACK (*axknowledgment)* cuando se reciben los datos. El emisor o el receptor no es informado cuando un paquete se pierde o se envía fuera de secuencia. El ACK de los paquetes es responsabilidad de una capa de más alto nivel de transporte como por ejemplo el TCP.

Cada host se individualiza mediante una dirección de 32 bits, dividida en 4 octetos. En ella se especifica un identificador de red y un identificador de host.

Para administrar estas direcciones se definieron diferentes clases:

- Clase A: 8 bits para red, 24 bits para hosts.
- Clase B: 16 bits para red, 16 bits para hosts.
- Clase C: 24 bits para red, 8 bits para hosts.
- Clase D y E: reservadas (D para multicasting)

Las direcciones origen y destino a nivel IP nunca cambian en la vida de una trama.

Para permitir a los dispositivos intermedios transmitir datagramas, éste cuenta con un encabezado en el que se especifican todos los parámetros de control necesarios para que el datagrama llegue a destino (dirección origen, dirección destino, tipo de protocolo.). Además del encabezado, el datagrama contiene la porción de datos que se está queriendo transmitir. El encabezado tiene una parte fija de 20 octetos y una parte opcional de longitud variable.

Al datagrama se le añaden los campos descritos a continuación a su cabecera cuando se pasa un paquete a la capa de transporte.

- Dirección IP del origen
- Dirección IP del destino
- Protocolo (TCP o UDP)
- Cheksum (un numero formado por un sencillo algoritmo matemático que nos garantice la integridad de todo el paquete IP recibido).
- Time To Live (TTL) Tiempo de vida. Es el lapso de tiempo en el cual va a vivir el datagrama antes de que sea descartado.

17.3.1. Estructura del Datagrama

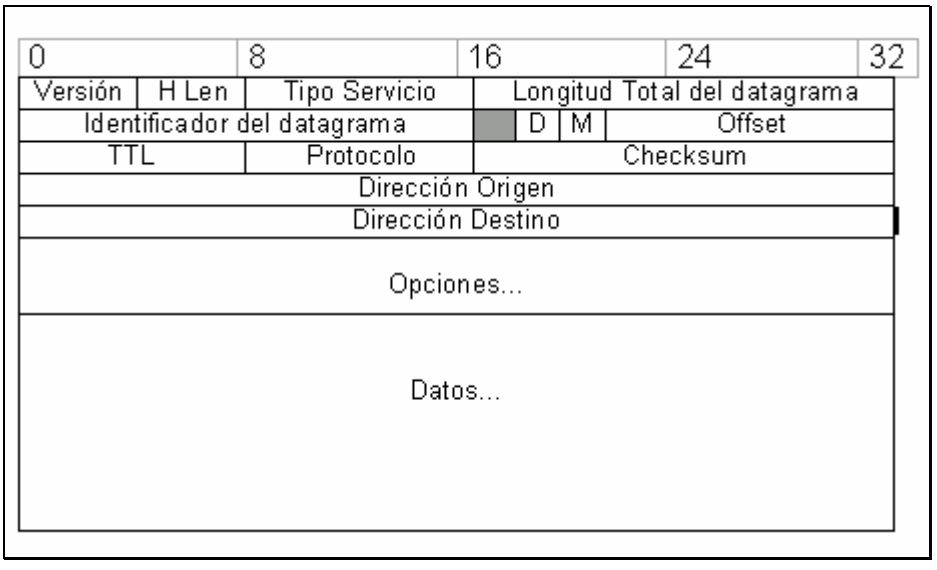


Figura 73

Versión: indica a qué versión del protocolo pertenece cada uno de los datagramas. Mediante la inclusión de la versión en cada datagrama, no se excluye la posibilidad de modificar los protocolos mientras la red se encuentra en operación.

H Len: especifica la longitud que tiene el encabezado en palabras de 32 bits, es necesario puesto que la longitud del encabezado es variable.

Tipo Servicio: indica el tipo de servicio, es posible tener varias combinaciones con respecto a la seguridad y a la velocidad.

Longitud Total del Datagrama: incluye todo el datagrama, tanto el encabezado como los datos, está expresado en bytes.

Identificador del Datagrama: se necesita para permitir al destino determinar a qué datagrama pertenece el fragmento recién llegado. Todos los fragmentos de un datagrama contienen el mismo identificador (el identificador se asigna aleatoriamente).

Bit D: si este campo esta seteado con 1 indica que el datagrama no se puede fragmentar.

Bit M: si este bit se encuentra en 1 significa que existen mas fragmentos, todos los fragmentos excepto el último deberán tener este bit seteado en 1.

Offset: es el desplazamiento del fragmento dentro del datagrama original. Se utiliza para regenerar el datagrama original.

TTL (Time To Live): es un contador que se utiliza para limitar el tiempo de vida de los paquetes. Cada vez que el datagrama pasa por un router el campo TTL se decrementa en 1, cuando llega a cero el datagrama se descarta.

Protocolo: indica el protocolo de nivel superior que el datagrama está transportando.

Checksum: es el campo que se utiliza para el reconocimiento de errores en IP.

Direcciones Origen y Destino: especifican las direcciones IP del host origen y del host destino respectivamente.

Opciones: este campo se utiliza con fines de seguridad, informe de errores, depuración, así como para otro tipo de información. Permite también, incluir a versiones de protocolos subsiguientes información que no está presente en el diseño original.

17.4. Routing de IPv4

Como se mencionó anteriormente IP es responsable de llevar un datagrama al destino indicado en el encabezado, pero poco se dijo como se hace. La tarea de encontrar la forma llevar un datagrama al destino es conocida como Routing.

Pero primero trataremos algunos temas importantes para la comprención del Routing

17.4.1. Mascara de Red y Dirección IPv4

Cada *host* en una red TCP/IP requiere una máscara de red (*subnet mask*). Vamos a ver el propósito de una máscara de red y como esta, forma parte del proceso que el IP usa para enviar paquetes.

Una máscara de red es una dirección de 32 bits usada para 'enmascarar' una parte de la dirección IP para distinguir el ID de red del ID de *host*. Esto es necesario para que se pueda determinar cuando una dirección IP pertenece a la red local o a una red remota.

Cada máquina en una red TCP/IP requiere una máscara de red, bien una máscara de red por defecto usada cuando una red no está dividida en subredes, o una máscara 'personalizada' cuando la red está dividida en segmentos.

Una máscara de red por defecto se usa en las redes cuando estas no están divididas en subredes. Todos los *hosts* IP requieren esta máscara aunque estén en un solo segmento de red. La máscara por defecto que podemos utilizar, depende de la 'clase' de dirección.

En la máscara de red, todos los bits que corresponden a un ID de red están colocados a 1. El valor decimal de un octeto con todos unos, es 255. Todos los bits que corresponden al ID *host* estarán colocados a cero.

Clase	Bits usados por la mascara de red	Valor decimal
Clase A	11111111 00000000 00000000 00000000	255.0.0.0
Clase B	11111111 11111111 00000000 00000000	255.255.0.0
Clase C	11111111 11111111 11111111 00000000	255.255.255.0

17.4.2. Determinando el destino de un paquete.

Una suma binaria (**AND**) es el proceso interno que el IP utiliza para determinar cuando un paquete está destinado para un *host* local (en la propia red local) o remoto (en una red remota). Debido a que el **AND** es usado internamente por el IP, normalmente no necesitaremos realizar esta tarea.

Cuando se inicializa el protocolo, se efectúa un and entre la dirección IP del *host* y la máscara de red. Antes de que un paquete sea enviado, se realiza un “and” entre la dirección IP del destino y la misma máscara. Si el resultado de ambas sumas es idéntico, el IP sabe que debe enviarlo a la red local. Si este resultado no coincide el paquete será enviado a la dirección de un *router* o *gateway* por defecto (*default gateway).*

Para efectuar un “and” entre una dirección IP con la máscara de red, el TCP/IP compara cada bit en la dirección IP con el correspondiente bit de la máscara de red.

17.4.3. Definiendo una Máscara de Subred

El definir una máscara de subred es un proceso de tres pasos. Vamos a ver esos tres pasos y a realizar unos ejemplos para definir las subredes.

Definir una máscara de subred es necesario si se está dividiendo la red en subredes. Para ello se deben efectuar los tres pasos siguientes:

1) Una vez determinado el número de segmentos en la red, se convierte dicho número a formato binario.

2) encontrar el número de bits necesarios para representar el número de segmentos físicos en binario. Por ejemplo, si se necesitan 6 subredes, el valor binario es 1 1 0. Para representar 6 en binario, se requieren **tres** bits.

3) Convertir ese número de bits a formato decimal de izquierda a derecha. Por ejemplo si son necesarios 3 bits, se utilizan los tres primeros bits del ID de *host* como el ID de subred. Es decir: 11100000. Su valor decimal es 224. La máscara de subred es por tanto: 255.255.224.0 en el ejemplo de clase B.

17.4.4. Implementando Routing de IPv4

Routing (encaminar) es el proceso de escoger el camino bajo el cual van a ser enviados los paquetes. El *routing* sucede cuando enviamos los paquetes a través de un *router* debido a que el *host* destino no está en nuestra red. Un *router* es una máquina o un dispositivo que reenvía los paquetes desde una red física a otra. A los *routers* muchas veces se les llama *gateways*.

La secuencia por la que el TCP/IP envía los paquetes:

1) Se hace un “and” entre la dirección IP origen (maquina local) y la máscara de IP.

2) Se hace un “and” entre la dirección IP destino y la máscara de IP.

3) Si coinciden, pertenece a la red local, por lo que se localizará la dirección física del destino, primero mirando en la caché ARP de nuestra máquina, y si no existe, se localizará la máquina destino mediante *broadcasting* de ARP. Una vez localizada se enviará el paquete IP al destino y se guardará la dirección física de ese destino en la caché ARP.

4) Si no coinciden, el paquete se envía al *router* o *gateway* por defecto que tengamos definido en nuestro *hosts*.

Pero antes de enviar el paquete, se debe tomar la decisión de por donde hay que enviarlo. Esta decisión deben tomarla todos los *hosts* bien sea el *localhost* o cualquier *router* por los que el paquete atraviese. Para tomar la decisión de

enrutamiento, la capa IP consulta una tabla de rutas que está almacenada en memoria. Una tabla de rutas, contiene entradas que relacionan la dirección IP buscada y la *interface* a utilizar.

Pensemos que nuestra maquina, puede tener más de una interfase de red. Este es el caso de los *routers* e incluso el caso de un PC doméstico con tarjeta de red y con módem (el cual es un adaptador más). Antes de enviar el paquete a la red, se debe tomar la decisión de 'por donde' enviarlo.

1) Cuando un *host* espera comunicar con otro *host*, el IP determina primero si el destino está en la red local o en otra red.

2) Si el destino es un *host* remoto (está en otra red), el IP busca en la tabla de rutas una posible ruta para localizar el *host* destino en la red remota.

3) Si no hay una ruta explicita, IP utiliza el *gateway* por defecto para enviar el paquete al *router*.

4) En el *router* otra vez, es consultada su tabla de rutas, para seguir buscando un camino del *host* remoto o de la red. Si no existe un camino explícito, el *router* reenviará otra vez el paquete a su propio *gateway* por defecto para continuar la cadena y que sea este siguiente *router* el encargado de repetir el ciclo.

Así como se encuentra un *router* el paquete se envía al siguiente *router*. Esto se le llama un "salto". Finalmente el paquete es entregado en el *host* destino. Si alguna ruta no se encuentra se envía un mensaje de error al *hosts* origen.

17.5. Multicast IPv4

El protocolo de Internet tiene tres formas de transmitir información. La forma más común es la llamada "Unicast", en esta forma existe una sola fuente y un solo destino, es decir una comunicación uno a uno.

La segunda es la llamada “broadcast”, en este caso existe una fuente y todos los nodos de la red son los receptores, este mecanismo es muy útil cuando se desea llegar a muchos receptores y se cuenta con recursos escasos como procesamiento o ancho de banda. La desventaja es que los nodos que no tienen interés en la información enviada se ven interrumpidos con cada mensaje de broadcast, además que el broadcast no es enviado por los routers de un segmento a otro.

La tercera forma es la llamada “multicast”. El multicast es similar al broadcast ya que con un solo paquete se intenta llegar a muchos nodos en la red, la diferencia radica en que sólo los nodos interesados reciben el tráfico y que el multicast puede ser enviado por los routers a través de varios segmentos. Esto se hace identificando a cada grupo de receptores y fuentes con un rango especial de direcciones de IP en el rango de clases “D”. Cuando un nodo desea recibir tráfico de cierto grupo, envía un mensaje al router de su segmento que desea ingresar a ese grupo. El router debe entonces crear un camino inverso para llegar a la fuente, se le llama inverso porque en “unicast” los routers envían paquetes a través de un camino formado tomando en cuenta el destino, en multicast se crea este camino tomando en cuenta a la fuente.

17.5.1. Generalidades Multicast

Como se ha podido ver, el multicast es muy útil para enviar los mismos datos para diferentes usuarios. Esto provee un mejor manejo del tráfico y un ahorro en el procesamiento de los servidores y equipos de redes como router, lográndose así un ahorro en el ancho de banda en comparación de lo que necesitaría por medio de unicast. Se podría decir que el multicast es un broadcast con “buenos modales”. Las figuras muestran las diferencias entre multicast (fig. 2) y unicast (fig. 1).

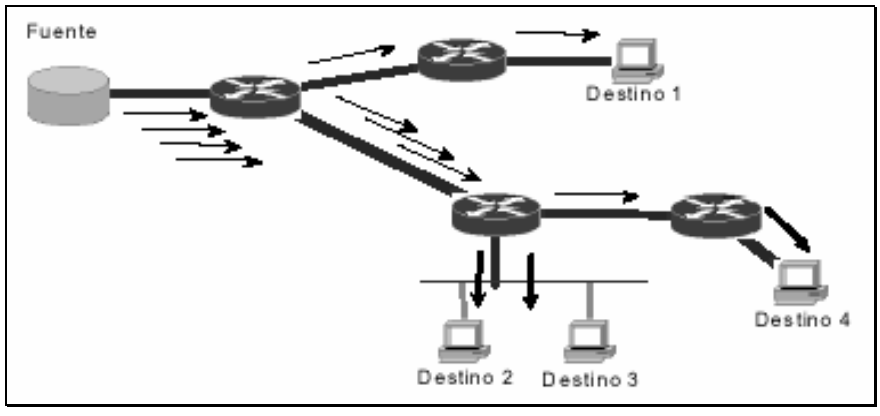


Figura 74

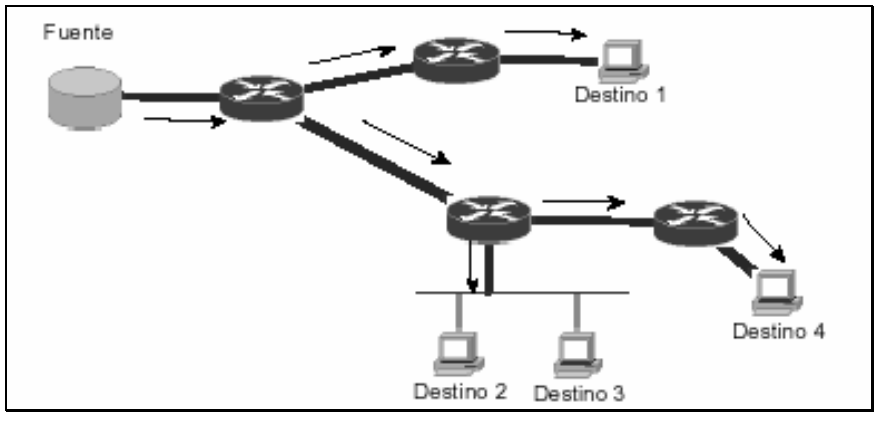


Figura 75

Entre las aplicaciones que pueden aprovechar las bondades de multicast están videoconferencias, ráfagas de video y audio, comunicación entre dispositivos de red (routers) con protocolos como OSPF y EIGRP, aprendizaje a distancia, transferencia de archivos, replicación de archivos y datos.

También a pesar de todas las ventajas, el multicast tiene algunas desventajas. Entre ellas es que su mecanismo de transmisión es UDP, no posee mecanismos de QoS, pueden esperarse pérdida de paquetes en la transmisión, no tiene control de congestión, pueden existir paquetes duplicados o fuera de orden y por lo tanto las aplicaciones deben saber como manejar esta situación.

17.5.2. Direcciones Multicast

Para identificar los grupos de multicast, el IANA asigno un set de direcciones específicas para este tipo de tráfico. Hay una cuarta clase, la clase D, reservada para las direcciones multicast. La clase D tiene reservado el rango de direcciones IPv4 entre la 224.0.0.0 y la 239.255.255.255. Los 4 bits de mayor peso de la dirección IP permiten direccionar entre el valor 224 y el 239. Los 28 bits restantes de menor peso, están reservados para el identificador del grupo multicast.

Las direcciones multicast IPv4 a nivel de red, deben mapearse sobre las direcciones físicas correspondientes al tipo de red con que se esté trabajando. Si se estuviese trabajando con direcciones a nivel de red unicast, se obtendría la dirección física asociada haciendo uso del protocolo ARP, en el caso de direcciones de red multicast, no se puede usar ARP y habrá que obtener la dirección física asociada mediante un procedimiento diferente. Se han definido varios documentos RFC que especifican la forma de realizar este mapeo, La correspondencia de direcciones multicast IPv4 a direcciones físicas ethernet esta especificado en RFC 1112.

Ethernet identifican un paquete de multicast como aquellos donde el primer bit de la dirección MAC está encendido. Para que se pueda mapear la dirección de clase D a dirección de hardware, no existe un mecanismo como el ARP, lo que se usa es un mapeo directo de los 23 primeros bits de derecha a izquierda de la dirección clase D al prefijo MAC 0100.5Exx.xxxx. El problema de esta solución es que debido a que las clases D cuentan con 28 bits y el prefijo sólo deja espacio para 23, esto deja 5 bits sin asignar y esto puede llegar a producir direcciones MAC iguales para diferentes clases D. Para FDDI con algunas diferencias en el manejo de los bits, es igual que en ethernet. En cambio token ring, debido a su implementación sufre más con el multicast, donde todos los nodos responden a este tipo de paquetes.

Hay algunas direcciones multicast IPv4 especiales:

- La dirección 224.0.0.1 identifica a todos los hosts de una subred. Cualquier host con capacidades multicast que se encuentre en una subred deberá unirse a este grupo.
- La dirección 224.0.0.2 identifica a todos los routers con capacidades multicast de una subred.
- El rango de direcciones 224.0.0.0 - 224.0.0.255 está reservado para protocolos de bajo nivel. Los datagramas destinados a direcciones dentro de este rango nunca serán encaminados por routers multicast.
- El rango de direcciones 239.0.0.0 - 239.255.255.255 está reservado para usos administrativos. Las direcciones en este rango se asignan de forma local por cada organización pero no se asegura que no existan otras direcciones como esas fuera de la red de la organización. Los routers de la organización no deberán encaminar los datagramas destinados a direcciones dentro de este rango fuera de la red corporativa.

Hay más direcciones multicast reservadas que las aquí mostradas, para una referencia completa consultar la última versión disponible del RFC "Assigned Numbers".

17.5.3. Funcionamiento Multicast

Para que los nodos puedan unirse a un grupo de multicast deben de informarlo a los routers para que estos envíen el tráfico de multicast a través de la interfaz donde el nodo se encuentra. Este anuncio se usa utilizando el Protocolo de Manejo de Grupos de Internet (IGMP). Hasta el momento existe la versión 1 (RFC1112 “d“) y la

versión 2 (RFC2236), actualmente se está desarrollando la versión 3. IGMP usa el modelo de pregunta-respuesta, en el cual los routers se dan cuenta si existen nodos que desean recibir tráfico de algún grupo de multicast. Inicialmente un router envía una solicitud para la posible recepción de tráfico de un grupo en específico mediante un “IGMP query”, esto se hace normalmente cada 60 segundos. Si algún nodo desea recibir tráfico de algún grupo, se informa al router mediante un “IGMP Membership Report”. Otros nodos en la subred escuchan la respuesta y si también desean recibir tráfico suprimen su respuesta para evitar el desperdicio de ancho de banda, este es llamado: mecanismo de supresión de reporte.

Cuando un nodo desea integrarse a un grupo, no es necesario que espere a que un router haga una solicitud, el nodo puede hacer un “IGMP membership report” sin solicitud. Esto ahorra tiempo a los nodos para unirse a grupos.

Desgraciadamente en IGMPv1 no existe un mecanismo de abandono de grupo, de esta forma los routers no pueden saber cuando el último miembro de un grupo ha dejado de pertenecer a él. La única forma de que los routers puedan saber que ya no existen miembros de un grupo con IGMPv1 es cuando dejan de recibir respuesta a los “IGMP query”. Esta situación puede causar un desperdicio de ancho de banda cuando el tráfico del grupo es una sesión de video de alta calidad. Con IGMPv1 existen aproximadamente 3 minutos de “time out” donde el router envía tráfico que ningún nodo escucha.

Esta fue una de las razones por las cuales el IETF generó una segunda versión del IGMP. Entre las mejoras que se añadieron al IGMPv2 se tiene:

- Proceso de Elección del “solicitador”: En IGMPv1, si existen varios routers en la misma subred, no existe un mecanismo que elija cual de ellos será el encargado de hacer las solicitudes de grupo. En IGMPv1, el proceso de decisión es dejado a

protocolos de nivel 3 (PIM, DVMRP, etc.). En la versión 2, los routers pueden hacer este proceso sin recurrir a protocolos de enrutamiento de multicast.

- Campo de Tiempo Máximo de Respuesta: Este nuevo campo permite a los routers definir el máximo tiempo de “Query-Response”. Permite afinar el temporizador para situaciones de tráfico excesivo.

- Mensajes de Solicitud de Grupos Específicos: En IGMPv1 el router hace la solicitud de grupo a todos los grupos en general. En IGMPv2, el router puede hacer la solicitud a un grupo en específico.

- Mensajes de Abandono de Grupo: En IGMPv1 los routers no tienen forma de saber cuando no existen más miembros de grupo. En IGMPv2 los nodos pueden (no obstante no es obligatorio) informar a los routers que abandonan el grupo.

Para asegurar la interoperabilidad entre IGMPv1 e IGMPv2 se definieron algunas reglas.

- Nodos V2/routers V1: Los routers V1 ignoran los mensajes de V2. Los nodos deben cambiar y enviar mensajes de versión 1.

- Nodos V1/routers V2: Los nodos ven los mensajes de solicitud de versión 2 como si fueran versión 1. Sólo el segundo octeto del mensaje es diferente y es ignorado. El mecanismo de salida de grupo es suprimido.

- Mezcla de routers de V1 y V2: No existe un mecanismo definido, de esta forma si existen enrutadores de V1 y V2 en la subred, los enrutadores V2 deben ser configurados administrativamente con la V1.

El que los routers conozcan si existen miembros de un grupo de multicast en una subred o no, es sólo una parte de todo el mecanismo del multicast. Como se mencionó anteriormente, al contrario del unicast donde los routers buscan el camino

en base a un destino, en multicast los routers buscan el camino en base a la fuente. Para poder lograr esto, deben de crearse los “Árboles de Distribución de Multicast”. Existen dos tipos básicos de árboles, los “árboles fuente” y los “árboles compartidos”. La forma más simple para crear un árbol es utilizando a la fuente como del tráfico de multicast como raíz. En este tipo de árboles las ramas forman el camino más corto desde los destinos a la fuente, por esta razón también son llamados “Árbol del camino más corto” (SPT).

Otro concepto importante para los enrutadores es el “Envío a través del camino inverso” (RPF). Cuando un paquete de multicast llega a un router, éste hace un chequeo del RPF. El chequeo consiste en verificar si la fuente del paquete está en el camino reverso a la fuente, es decir, si llega por la interfaz correcta. Si es exitoso el paquete, entonces es enviado a donde sea requerido, en el caso contrario el paquete es eliminado. La figura 3 ejemplifica esta situación, en donde se recibe un paquete de la fuente 200.23.40.55 por la interfaz s3, El router hace el chequeo del RPF y este falla debido a que la fuente tiene un camino inverso por la interfaz s0 y el paquete es descartado.

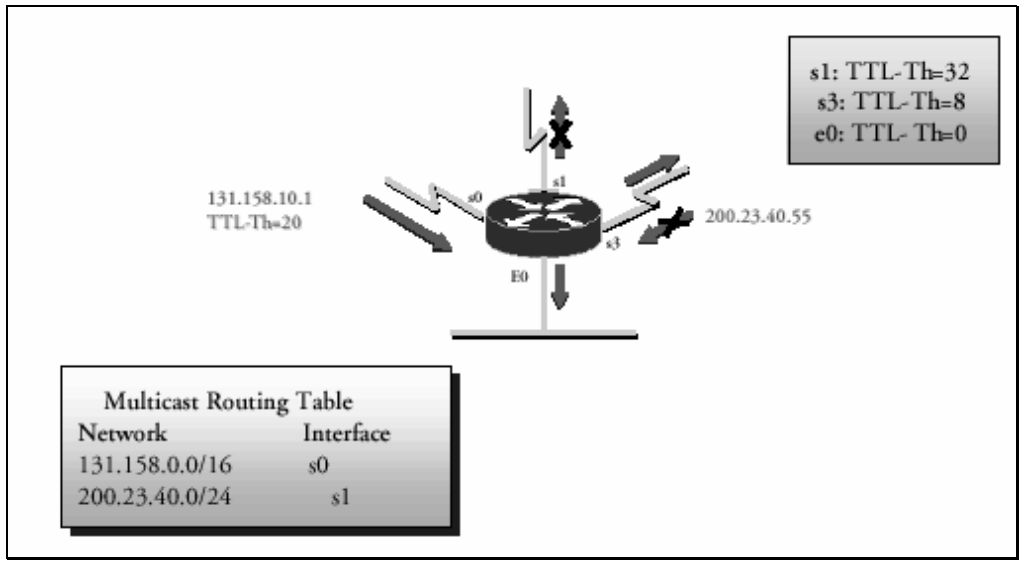


Figura 76

La figura anterior también muestra el chequeo del tiempo de vida (TTL) del paquete de multicast. Para que un router haga el envío de un paquete por una interfaz, el TTL del paquete debe ser mayor al TTL de la interfaz. En el caso del ejemplo, el paquete llega con TTL=20, al ser procesado por el router el TTL se ve disminuido en 1, para tener un nuevo valor igual a 19. Para la interfaz s1, el TTL=32 y por tanto la condición TTL(paquete)>TTL(interfaz) no se cumple y el paquete descartado. Para las demás interfaces si se cumple la condición y el paquete es enviado.

Para poder informar a otros routers sobre fuentes y destinos de multicast se deben emplear además protocolos de enrutamiento. Existen tres categorías básicas:

- Protocolos de Modo Denso (DVMRP y PIM-DM)

- Protocolos de Modo Sparse (PIM-SM y CBT)

- Protocolos de Estado de Enlace (MOSPF)

Los protocolos del tipo denso utilizan el árbol más corto junto con un mecanismo de empuje. Este mecanismo de empuje asume que en cada interfaz del enrutador existe al menos un receptor del grupo. El tráfico es enviado o “flooded” a través de todas las interfaces. Para evitar el desperdicio de recursos, si un router no desea recibir tráfico envía un mensaje de supresión (prune), como resultado es que sólo donde se tienen miembros de los grupos de multicast el tráfico es enviado. Este comportamiento de “Flood” y “Prune” se repite aproximadamente cada 2 o 3 minutos dependiendo del protocolo, por esta razón protocolos del tipo denso son mayormente empleados en ambientes LAN y donde el número de receptores usualmente es alto comparado con el de las fuentes.

Protocolos basados en modo denso son el Distance Vector Routing Protocol (DVMRP) y el Protocol Independent Multicast Dense Mode (PIM-DM). El DVMRP fue el primer protocolo de enrutamiento desarrollado para multicast, está basado en RIP y por lo tanto tiene muchas de las desventajas de los protocolos de Vector de Distancia.

La forma en que un grupo de multicast se identifica es mediante una dirección de IP de clase D. Para poder reconocer este tipo de información los equipos que procesen los paquetes de multicast deben de tener cierto entendimiento. Para redes basadas en routers esto no presenta ningún problema y son capaces de saber en que interfaz existen miembros de un grupo que desean recibir tráfico.

Las redes modernas son una mezcla de routers y conmutadores LAN. Si bien, los routers pueden eliminar el tráfico en las interfaces donde no existen miembros de grupos de multicast, también si por los menos existe un miembro, los routers harán el envío de tráfico. Esto trae como consecuencia que los conmutadores LAN al no tener capacidad de procesar información de capa 3, enviarán el tráfico multicast a través de todos sus puertos, existan o no, miembros de un grupo multicast. La figura 4 muestra el comportamiento de una red conmutada ante el tráfico multicast. En este caso el router tiene una troncal por donde se pasa tráfico de tres LANs, en este caso sólo los nodos A, B y C desean ser parte del grupo multicast. El router recibe los mensajes de “Join” de IGMP y hace el envío (flood) de paquetes según su tabla de enrutamiento a las LANs superior, derecha e inferior. El conmutador que no tiene capacidad de capa 3 hará el envío de paquetes hacia todos los puertos que pertenezcan a esa LAN, existan o no clientes del grupo multicast, esto se debe a que la topología lógica aparenta ser un router con tres segmentos ethernet y el multicast es tratado por los conmutadores LAN de la misma forma que el broadcast. Este comportamiento puede

llegar a causar congestiones en los puertos de los conmutadores LAN así como otros inconvenientes causados por tráfico excesivo.

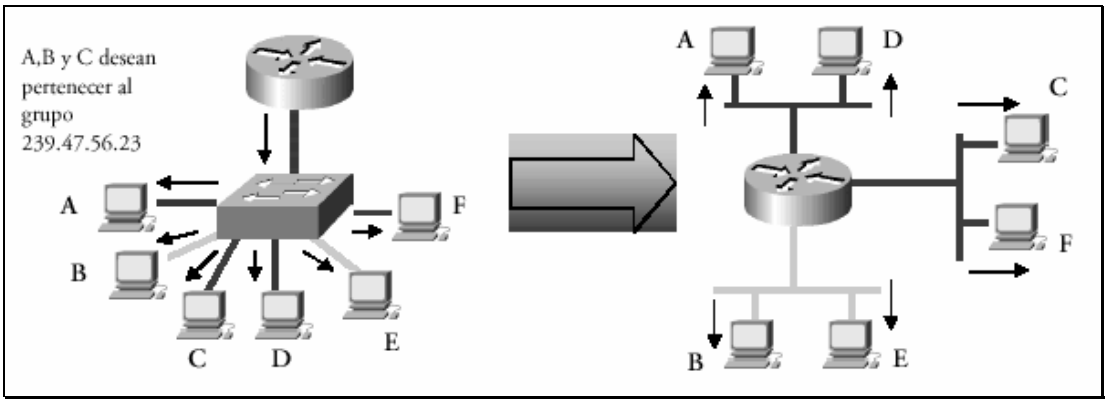


Figura 77

Para evitar este inútil envío de tráfico a donde no se requiere, se han creado mecanismos para hacer que los conmutadores LAN tengan conocimiento de grupos de multicast en sus puertos. Hasta el momento se han desarrollado los siguientes métodos:

- ***IGMP Snooping***: En este mecanismo el conmutador LAN debe "escuchar" las conversaciones de IGMP entre el router y el nodo. Aunque este método aparenta ser el más lógico, existen algunos problemas prácticos en las arquitecturas de los conmutadores que pueden llegar a hacer de esta solución no disponible. Una de las razones es que si el conmutador no posee algún tipo de capacidad de capa 3, como usualmente lo son los conmutadores de costo reducido, el procesamiento de IGMP se hace directamente en el

CPU del conmutador, haciendo que bajo altas cargas de tráfico de multicast, el conmutador pueda fallar.

- ***Cisco Group Management Protocol***: Este protocolo fue desarrollado por Cisco Systems. La ventaja de este mecanismo es que define un mensaje entre los routers. De tal forma los primeros pueden informar de una forma eficiente a los segundos ante la presencia de grupos de multicast.
- ***IEEE Generic Attribute Resolution Protocol (GARP):*** Este protocolo es definido por la IEEE bajo el IEEE 802.1p. Es aún muy nuevo y requiere de cambios de software y hardware en el nodo final.

Al utilizar alguno de los mecanismos anteriores, el ejemplo se transformará en el descrito por la figura

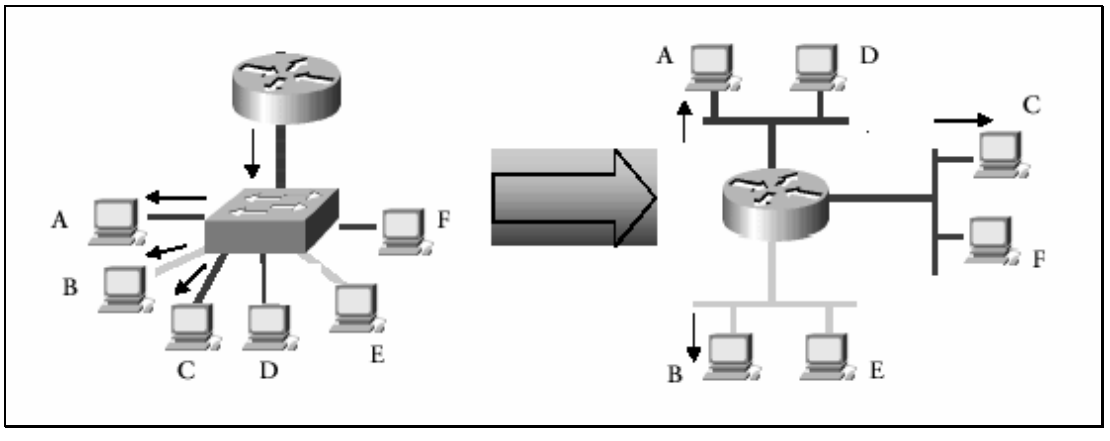


Figura 78

En este caso, el conmutador identifica que sólo los nodos A, B y C son miembros de grupos de multicast y suprime el tráfico de multicast en los puertos donde se encuentran los nodos D, E y F

ANEXO 2: Código Fuente

Este proyecto de tesis compre el desarrollo de una aplicación multicast de tal manera de ejemplificar todo lo descrito en los capítulos anteriores.

La aplicación que a continuación se describe en sus partes principales tiene la funcionalidad de distribución masiva de archivos por medio de multicast sobre el protocolo IPv6 como también IPv4 tal como se recomienda en el capitulo anterior.

18. Definición Codigo Fuente

18.1. Definición de Librerías

```
#include <stdio.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <netdb.h>
#include <net/if.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/time.h>
#include <time.h>
#include <string.h>
#include <signal.h>
```

Figura 79

18.2. Estructura de los segmentos de información

```
typedef struct
    {
     long long ID_paquete;      //Identificacion del paquete enviado
     long long Total_paquetes;  //Total de paquetes que componen el archivo
     long long Hora_envio;      //Tiempo de envio
     long long Tam_paquete;     //tamaño en bytes de los datos del paquete
     struct in_addr IP_origen;  //Estructura con la informacion IP del origen
     int ID_archivo;            //Identificacion de archivo
     int Codigo_OP;             //Tipo de Paquete
    } Cabecera;
```

Figura 80

18.3. Estructuras IP y variables Principales

```
FILE *Archivo;
int descriptor;
struct sockaddr_storage Estruct_Storage_Servidor;
struct ipv6_mreq Struct_Multicast_v6;
long long Total_de_paquetes;
```

Figura 81

18.4. Variables Principales

```
//Constantes para el manejo de Tamaños de paquetes
const long Tam_Paquete = 8800;
const long Tam_Cabecera = sizeof(Cabecera);
const long Tam_Datos = (Tam_Paquete-Tam_Cabecera);
Cabecera cabecera;
char Datos[Tam_Paquete-Tam_Cabecera];
char Paquete[Tam_Paquete];

struct stat Estado_archivo;

//variables par almacenar datos grupos e indices de interfaces de red
char *GrupoMulticast;
char *NumeroPuerto;
char *if_name;
int if_index;

//variables para el manejo y me dida de tiempos
struct timeval Estado_reloj_inicio,Estado_reloj_fin;
long long bytes_leidos;
long long Microtiempo;
double Tiempo;

//Estructuras para la busqueda de direcciones IP
struct addrinfo Entrada;
struct addrinfo *Salida;
struct addrinfo *Recorre;

struct ip_mreq Struct_Multicast_v4;
struct ifreq Struct_Interface;
```

Figura 82

18.5. Configuración de los parámetros de red para la búsqueda de una dirección valida

```
bzero(&Entrada,sizeof(Entrada));
Entrada.ai_flags=AI_CANONNAME;
Entrada.ai_family=AF_UNSPEC;
Entrada.ai_socktype=SOCK_DGRAM;
Entrada.ai_protocol=PF_UNSPEC;
```

Figura 83

18.6. Segmento de algoritmo que busca una dirección valida para la conexión.

```
while (Recorre)
  {
    fprintf(stderr,"\n **Direccion Numero: %d\n",i);
    i++;
    descriptor=socket(Recorre->ai_family,Recorre->ai_socktype,Recorre->ai_protocol);
    if (descriptor!=-1)
        {
         retorno_func=bind(descriptor,Recorre->ai_addr,Recorre->ai_addrlen);
         if (retorno_func!=-1)
            {
             close(descriptor);
             memcpy(&Estruct_Storage_Servidor,Recorre->ai_addr,sizeof(Estruct_Storage_Servidor));
             printf("Direccion OK\n");
             direccionOK=1;
             break;
            };
          close(descriptor);
          descriptor=-1;
        }
    Recorre=Recorre->ai_next;
   };
freeaddrinfo(Recorre);
if (direccionOK==0)
  {
    printf("Error!! : No se pudo conseguir una direccion valida");
    exit(1);
  }
```

Figura 84

18.7. Creación del Servidor

```
descriptor=socket(Estruct_Storage_Servidor.ss_family,SOCK_DGRAM,0);
if (descriptor==-1) printf("Error en el Socket\n");
  else printf("Socket Creado Con Exito\n");

retorno_func=bind(descriptor,(struct sockaddr *) &Estruct_Storage_Servidor,sizeof(Estruct_Storage_Servidor));
if (retorno_func==-1) perror("\nError en el Bind\n");
  else printf("\nBind Exitoso!!\n");
```

Figura 85

18.8. Ingreso a Grupo Multicast Caso IPv4

```
switch (Estruct_Storage_Servidor.ss_family)
 {
  case AF_INET:
    {

      //*****Seccion 6.1.- llenado de la estructura multicast para ipv4  *****//

      Struct_Multicast_v4.imr_multiaddr.s_addr=((struct sockaddr_in *)&Estruct_Storage_Servidor)
                                                                      ->sin_addr.s_addr;
      strncpy(Struct_Interface.ifr_name,if_name,IFNAMSIZ);
      retorno_func=ioctl(descriptor,SIOCGIFADDR,&Struct_Interface);
      if (retorno_func==-1) perror("\nError en ioctl\n");
          else printf("\nioctl Exitoso!!\n");
      Struct_Multicast_v4.imr_interface=((struct sockaddr_in *) &Struct_Interface.ifr_addr)
                                                                      ->sin_addr;

      //*****Seccion 6.2.- Fijar Opciones del socket ipv4 *****//

      retorno_func=setsockopt(descriptor,SOL_SOCKET, SO_REUSEADDR,&on,sizeof(on));
      if (retorno_func==-1) perror("\nError en SO_REUSEADDR\n");
          else printf("\nSO_REUSEADDR Exitoso!!\n");
      retorno_func=setsockopt(descriptor,IPPROTO_IP, IP_MULTICAST_LOOP,&loopback,sizeof(loopback));
      if (retorno_func==-1) perror("\nError en IP_MULTICAST_LOOP\n");
          else printf("\nIP_MULTICAST_LOOP Exitoso!!\n");
      retorno_func=setsockopt(descriptor,IPPROTO_IP, IP_MULTICAST_TTL,&hops,sizeof(hops));
      if (retorno_func==-1) perror("\nError en IP_MULTICAST_TTL\n");
          else printf("\nIP_MULTICAST_TTL Exitoso!!\n");
      retorno_func=setsockopt(descriptor,IPPROTO_IP, IP_ADD_MEMBERSHIP,&Struct_Multicast_v4,
                                                                      sizeof(Struct_Multicast_v4));
      if (retorno_func==-1) perror("\nError en IP_ADD_MEMBERSHIP\n");
          else printf("\nIP_ADD_MEMBERSHIP Exitoso!!\n");
      break;
    }
```

Figura 86

18.9. Rutina de ingreso a Grupos multicast IPv6

```
case AF_INET6:
  {
    //*****Seccion 6.3.- llenado de la estructura multicast para ipv6  *****//

    Struct_Multicast_v6.ipv6mr_multiaddr=((struct sockaddr_in6 *) &Estruct_Storage_Servidor)
                                                                    ->sin6_addr;
    Struct_Multicast_v6.ipv6mr_interface=if_index;

    //*****Seccion 6.4.- Fijar Opciones del socket ipv6  *****//

    loopback=1;
    retorno_func=setsockopt(descriptor,IPPROTO_IPV6,IPV6_MULTICAST_LOOP,&loopback,sizeof(int));
    if (retorno_func==-1) perror("\nError en el IPV6_MULTICAST_LOOP\n");
      else printf("\nIPV6_MULTICAST_LOOP Exitoso!!\n");

    hops=2;
    retorno_func=setsockopt(descriptor,IPPROTO_IPV6,IPV6_MULTICAST_HOPS,&hops,sizeof(int));
    if (retorno_func==-1) perror("\nError en el IPV6_MULTICAST_HOPS\n");
      else printf("\nIPV6_MULTICAST_HOPS Exitoso!!\n");

    retorno_func=setsockopt(descriptor,IPPROTO_IPV6,IPV6_ADD_MEMBERSHIP,&Struct_Multicast_v6,
                                                          sizeof(Struct_Multicast_v6));
    if (retorno_func==-1) perror("\nError en el IPV6_ADD_MEMBERSHIP\n");
      else printf("\nIPV6_ADD_MEMBERSHIP Exitoso!!\n");
    break;
  }
```

Figura 87

18.10. Algoritmo central de la aplicación Servidor

```
cabecera.ID_paquete=0;
cabecera.Total_paquetes=Total_de_paquetes;
cabecera.ID_archivo=0;
cabecera.Codigo_OP=0;
signal(2,&Control_C);
while (1)
{
  printf("\nInicio Envio archivo\n");
  gettimeofday(&Estado_reloj_inicio,NULL);
  for (i=0;i<Total_de_paquetes;i++)
  {
   bytes_leidos=fread(Datos,1,Tam_Datos,Archivo);
   cabecera.ID_paquete=i;
   cabecera.Codigo_OP=2;
   cabecera.Tam_paquete=(long long) bytes_leidos;
   Datos[bytes_leidos]=0;
   memmove(Paquete,&cabecera,Tam_Cabecera);
   memmove((Paquete+Tam_Cabecera),Datos,Tam_Datos);
   retorno_func=sendto(descriptor,Paquete,Tam_Paquete,0,
       (struct sockaddr *) &Estruct_Storage_Servidor,sizeof(Estruct_Storage_Servidor));
   if (retorno_func==-1) perror("\nError en Sendto :\n");
    else printf("\nPaquete N° %d enviado con exito!!\n",i);
   usleep(500000);
  }
  gettimeofday(&Estado_reloj_fin,NULL);
  Microtiempo = ((Estado_reloj_fin.tv_sec-Estado_reloj_inicio.tv_sec)*1000000)+
      (Estado_reloj_fin.tv_usec-Estado_reloj_inicio.tv_usec);
  Tiempo = ((double)(Microtiempo/1000000));
  printf("Se enviaron %d bytes en %f segundos: %f bytes/segundo\n", Estado_archivo.st_size,
      Tiempo, (Estado_archivo.st_size/Tiempo));
  rewind(Archivo);
}
return EXIT_SUCCESS;
```

Figura 88

18.11. Algoritmo central de la aplicación Cliente (parte I).

```
if(!(Archivo=fopen(argv[4], "w+"))) {
  fprintf(stderr, "Can not open %s to write!\n",argv[4]);
  exit(1);
}
gettimeofday(&Estado_reloj_inicio,NULL);
while (!fin)
{
  clientaddrlen=sizeof(Estruct_Cli);
  retorno_func=recvfrom(descriptor,Paquete,Tam_Paquete,0,
                        (struct sockaddr *)&Estruct_Cli,&clientaddrlen);
  if (retorno_func==-1) perror("\nError en Recvfrom\n");
    else printf("\nRecvfrom Exitoso!!\n");
  memmove(&cabecera, Paquete, Tam_Cabecera);
  memmove(Datos, (Paquete+Tam_Cabecera),Tam_Datos);
  Datos[cabecera.Tam_paquete]=0;
  printf("Numero de Paquete  : %Li\n",cabecera.ID_paquete);
  printf("Numero total de pauqtes del archivo: %Li\n",cabecera.Total_paquetes);
  printf("hora envio: %Li\n",cabecera.Hora_envio);
  printf("tamaño Paquete: %Li\n",cabecera.Tam_paquete);
  printf("ID Archivo: %d\n",cabecera.ID_archivo);
  printf("Codigo Operacion: %d\n",cabecera.Codigo_OP);

  //****Seccion 7.1.- Trabajo del Servidor (Operar Segun tipo de paquete) *****//

  switch (cabecera.Codigo_OP)
  {
      case (1):           //Caso en que el Servidor deja el Grupo
      {
        printf("\nServidor se Retiro del Grupo %s!!\n",GrupoMulticast);
        Hay_Servidor=0;
        break;
```

Figura 89

18.12. Algoritmo central de la aplicación Cliente (parte II).

```
case (2): case(4):  //caso en que llega un paquete desde el servidor
                    //o de otro cliente
{
  if (cabecera.Codigo_OP==2) Hay_Servidor=1;
    else  Hay_Servidor=0;
  retorno_func=fseek(Archivo, cabecera.ID_paquete*(Tam_Datos), SEEK_SET);
  if (retorno_func==-1) perror("\nError en fseek:\n");
    else printf("\nfseek Exitoso!!\n");
  retorno_func=fwrite(Datos, 1, cabecera.Tam_paquete, Archivo);
  if (retorno_func<cabecera.Tam_paquete) perror("\nError en fwrite:\n");
    else printf("\nfwrite Exitoso!!\n");
  if (Paquete_Inicial==-1)
  {
    Paquete_Inicial=cabecera.ID_paquete;
    Lista_paquetes_resividos=calloc(cabecera.Total_paquetes,1);
    Lista_paquetes_pedidos=calloc(cabecera.Total_paquetes,1);
  }
  Lista_paquetes_resividos[cabecera.ID_paquete]=1;
  Lista_paquetes_pedidos[cabecera.ID_paquete]=0;
  break;
}
```

Figura 90

18.13. Algoritmo central de la aplicación Cliente (parte III).

```
   case (3): //caso en que un cliente pide un Paquete y si se tiene se envia
   {
     Hay_Servidor=0;
     if (Lista_paquetes_resividos[cabecera.ID_paquete]==1)
       {
         retorno_func=fseek(Archivo, cabecera.ID_paquete*(Tam_Datos), SEEK_SET);
         if (retorno_func==-1) perror("\nError en fseek:\n");
           else printf("\nfseek Exitoso!!\n");
         bytes_leidos=fread(Datos,1,Tam_Datos,Archivo);
         cabecera.Codigo_OP=4;
         cabecera.Tam_paquete=(long long) bytes_leidos;
         Datos[bytes_leidos]=0;
         memmove(Paquete,&cabecera,Tam_Cabecera);
         memmove((Paquete+Tam_Cabecera),Datos,Tam_Datos);
         usleep(1+(int) (500000.0*rand()/(RAND_MAX+1.0)));
         retorno_func=sendto(descriptor,Paquete,Tam_Paquete,0,(struct sockaddr *)
                             &Estruct_Storage_Servidor,sizeof(Estruct_Storage_Servidor));
         if (retorno_func==-1) perror("\nError en Sendto :\n");
           else printf("\nPaquete N° %Li reenviado con exito!!\n",cabecera.ID_paquete);
       }
     break;
   }
}//end switsh
```

Figura 91

18.14. Algoritmo central de la aplicación Cliente (parte IV).

```
   //****Seccion 8.- Ver si esta Completo el Archivo*****//
   fin=1;
   for (i=0;i<cabecera.Total_paquetes;i++)
     if (Lista_paquetes_resividos[i]==0 && fin==1)
       {
        fin=0;
        pedir=i;
       }
   //****Seccion 9.- Pedir paquete en caso que el servidor ya no este *****//
   if ((!fin) && (!Hay_Servidor))
   {
    //pedir_paquete (pedir)
    cabecera.ID_paquete=pedir;
    cabecera.Codigo_OP=3;
    cabecera.Tam_paquete=1;
    Datos[0]=0;
    memmove(Paquete,&cabecera,Tam_Cabecera);
    memmove((Paquete+Tam_Cabecera),Datos,Tam_Datos);
    retorno_func=sendto(descriptor,Paquete,Tam_Paquete,0,
           (struct sockaddr *)&Estruct_Cli,sizeof(Estruct_Cli));
    if (retorno_func==-1) perror("\nError en Sendto de pedido de paquete :\n");
     else printf("\nPedido de Paquete N° %Li enviado con exito!!\n",pedir);
   } //end if
 }//end while gral
 printf("Archivo terminado con Exito");
 gettimeofday(&Estado_reloj_fin,NULL);

 fclose(Archivo);

 return EXIT_SUCCESS;
```