INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS ESTADO DE MÉXICO

MECANISMO DE CONTROL PARA UNA VIDEOCÁMARA SOBRE EL PROTOCOLO IPv6

TESIS QUE PARA OPTAR EL GRADO DE
MAESTRO EN CIENCIAS DE LA COMPUTACIÓN
PRESENTA

GABRIELA AZUCENA CAMPOS GARCÍA

Asesor:	**Dr. Rafael Fernández Flores**	
Comité de Tesis:	**Dr. José de Jesús Vázquez Gómez**	
	MCC. Ricardo López Castro	
Jurado:	**Dr. José de Jesús Vázquez Gómez**	**Presidente**
	MCC. Ricardo López Castro	**Secretario**
	Dr. Rafael Fernández Flores	**Vocal**

Atizapán de Zaragoza, Edo. Méx., Diciembre de 1999

RESUMEN

En la presente tesis se muestra el desarrollo de un mecanismo de control para una video-cámara utilizando el protocolo IPv6. Primeramente, se analizaron las características de IPv6, para posteriormente programar la interfaz socket necesaria para la comunicación cliente-servidor. En la programación se hace uso de la familia **PF_INET6**, las estructuras **in6_addr, sockaddr_in6**, **hostent *getipnodebyname,** la función **inet_ntop** y el protocolo TCP. Por otro lado, se encontró que para trabajar con el protocolo IPv6 es necesario la instalación del *Release* IPv6 para Solaris 7 FCS, y sus utilerías. Una vez analizados los requerimientos de software, se creó un protocolo para asegurar la transmisión de datos entre cliente-servidor. La video-cámara a controlar, utilizando una estrategia de control en lazo abierto, es una Sony EVI-D30. Por las características de la cámara, los comandos de control se envían a través del puerto RS-232. El control de la video-cámara se realizó en sus movimientos básicos: arriba, abajo, derecha, izquierda y "home". Para la varificación del mecanismo de control se estableció una comunicación remota entre dos máquinas SUN, localizadas en: ITESM - CEM (cliente, acorral), y DGSCA - UNAM (servidor, dogbert). Finalmente, se realizaron pruebas operativas en tiempo real, obteniéndose resultados exitosos.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

CAPÍTULO I

INTRODUCCIÓN

La necesidad de contar con un manejo cada vez más eficiente de la transmisión de información a través de Internet, ha motivado investigar el potencial que el protocolo IPv6 puede ofrecer para lograrlo. Más aún, en años recientes la transmisión de audio, video y datos, de manera simultánea, también ha generado un gran interés en IPv6.

En la presente tesis se propone como proyecto, investigar el potencial de IPv6 utilizándolo como protocolo de comunicación (transmisión de datos), para manipular o controlar en lazo abierto una video cámara. Cabe mencionar que únicamente se controlará la cámara en forma básica, es decir, la posición de la cámara (arriba, abajo, izquierda y derecha). El objetivo, luego entonces, es investigar y describir qué es IPv6, qué sistemas operativos y utilerías se requieren para que una red opere con IPv6, cómo se instala el protocolo IPv6 y cómo se programan sockets en IPv6; para posteriormente, crear un prototipo de red entre dos computadoras SUN, obtener los comandos de control de la cámara, escribir en el puerto serial de una máquina SUN, construir el cable para el acoplamiento del RS-232 con la máquina SUN y la cámara, para realizar pruebas generales del mecanismo de control. Finalmente, y con base en las pruebas operativas, se obtendrán algunas conclusiones con respecto al potencial que podría ofrecer el protocolo IPv6.

El transporte de información en Internet está basado en una arquitectura organizada en capas o niveles abstractos de funcionalidades llamadas protocolos que, esencialmente, añaden propiedades a la comunicación. Actualmente, Internet se basa en la versión 4 del protocolo IP (Internet Protocol), ya que las tres versiones anteriores fueron definidas y subsecuentemente sustituidas por la siguiente, hasta alcanzar la cuarta. La estrecha relación que une a IP con TCP (Transmission Control Protocol) ha conducido a la muy extendida nomenclatura de protocolos TCP/IP, constitutivos del corazón de la red Internet.

Hasta hace muy pocos años, las aplicaciones se limitaban al correo electrónico, la transferencia de archivos de datos y la conexión remota para los cuales IPv4 satisfacía plenamente a los usuarios. Sin embargo, la aparición del gopher como forma primitiva de navegación por la red en base exclusiva de textos y la "World Wide Web", han conducido a variaciones decisivas en el perfil del tráfico producido por los usuarios. Así mismo, la combinación de imágenes, texto y audio hizo que miles de usuarios comenzarán a emplear el "browser" o navegador desarrollado por Centro Nacional para Aplicaciones de Supercómputo (NCSA, por sus siglas en inglés) y bautizado como Mosaic. Bajo estas condiciones, las cifras de tráfico se dispararon motivando la necesidad de enlaces de red con mayor capacidad de transmisión, rebasando los límites de IPv4 [29].

En la actualidad se pide a Internet servicios en tiempo real, como las aplicaciones de telefonía a través de la red. Además, ha surgido un importante sector que emplea Internet como vehículo publicitario y comercial, reclamando mecanismos de seguridad que garanticen la confiabilidad, integridad y disponibilidad en las transacciones realizadas. Esta coyuntura ha originado la aparición de soluciones específicas basadas en los estándares existentes, tales como: el Protocolo en Tiempo Real (RTP, por sus siglas en inglés) [23], -empleado para aplicaciones de tiempo real, ya que ofrece calidad de servicio por medio del Protocolo de Control en Tiempo Real (RTCP, por sus siglas en inglés), y opciones de seguridad que proveen integridad, autentificación y confidencialidad de mensajes-, y el Protocolo de Reserva de Recursos (RSVP, por sus siglas en inglés) [24] -para la reserva de recursos o el empleo de algoritmos de seguridad específicos en las capas altas de los protocolos-. Debido a que el servicio portador común sobre Internet, es el protocolo IP, la nueva versión de IP, IPng (IP next generation) pretende componer y ofrecer un **estándar abierto** que auxilie a los protocolos antes mencionados (RTP y RSVP) e integre los nuevos requisitos necesarios para los servicios de red, como se menciona más adelante en las propiedades del IPv6. Con el fin de asignar una versión al nuevo IP, dado que la versión 5 ha sido empleada para el protocolo "Stream" (ST) que corre en paralelo con IPv4 en algunos ruteadores, se tomó la determinación de hacer corresponder IPv6 [1-7] al IPng.

Básicamente, la emergencia de un nuevo IP viene desencadenada por el mecanismo empleado en la asignación de direcciones a los sistemas conectados a Internet. IPv4 prevé el empleo de 32 bits para el direccionamiento de los equipos admitiendo un máximo de casi 4,300

millones de direcciones, que es totalmente insuficiente para responder a las demandas de los próximos años. Por está razón, el nuevo estándar IP ha previsto el empleo de 128 bits para el direccionamiento, alcanzando una cifra de 3.4×10^{38} números posibles. Otras mejoras que ofrece IPv6 son:

- **Formato de Encabezados:** IPv6 utiliza un formato de encabezado totalmente diferente al encabezado del IPv4. IPv6 simplifica el encabezado, utilizando sólo 7 campos, contra 13 campos que utiliza IPv4. Este cambio permite a los enruteadores procesar con mayor rapidez los paquetes y mejorar, por tanto, el rendimiento.

- **Encabezados de Extensión:** IPv6 codifica la información en cabeceras separadas. Cada datagrama consiste en la cabecera IPv6 base, seguida (opcionalmente) de una o más cabeceras de extensión y luego los datos. De esta forma se mejora el tiempo de procesamiento de los paquetes.

- **Manejo de Audio y Video:** IPv6 incluye un mecanismo que permite establecer una trayectoria de alta Calidad de Servicio (QoS) por la red, soportando así, aplicaciones como audio y video en tiempo real.

- **Autentificación y Privacidad:** IPv6 define extensiones que permiten la autenticidad de los usuarios, la integridad y confidencialidad de los datos mediante el uso de herramientas de criptografía.

- **Posibilidad de Autoconfiguración:** IPv6 contiene varias formas de autoconfiguración como la configuración "Plug and Play" de direcciones de nodos sobre una red aislada gracias a las características ofrecidas por DHCP (Dynamic Host Configuration Protocol).

Además de estas mejoras, IPv6 puede manejar hasta 16 niveles de prioridad de tráfico, encriptado y autentificación. Sin embargo, aunque IPv6 aporte estas soluciones necesita la colaboración de una nueva Internet que solucione el problema de velocidad. Así, en Octubre de 1996 se anunció la nueva Internet, llamada Internet2 (I2), como un esfuerzo conjunto de investigación y estudios avanzados que permite desarrollar, instalar y operar una nueva

generación de aplicaciones que aprovechen plenamente la capacidad de integración de redes de banda ancha, colaboración e interactividad en tiempo real; para que los ambientes de educación superior soporten objetivos de investigación nacionales, educación a distancia y enseñanza continua, entre otros. I2 [25] es un esfuerzo privado administrado por la Corporación Universitaria para el Desarrollo de Internet Avanzado (UCAID) que incluye más de 150 universidades, algunas oficinas del gobierno americano, entidades sin fines de lucro y 30 miembros de la industria de la computación y las telecomunicaciones como 3COM, Lucent, MCI, Nortel, etc. Otros esfuerzos realizados son la creación de "Very high speed Backbone Network Service" (VbNS), [30] y "Next Generation Internet" (NGI), [31] .

Por lo tanto, el protocolo IPv6 junto con Internet2 podrían tener diversas aplicaciones tales como:

- Manejo remoto de telescopios, en donde la actividad de un científico, en particular un astrónomo, puede potenciar su trabajo académico al disponer de contacto oportuno con los resultados de sus observaciones y adquisición de datos.

- Manipulación síncrona de modelos en la dinámica molecular, en donde las necesidades de cómputo y sincronía en aplicaciones requieren control y manipulación de modelos en puntos distantes.

- Transmisión de imágenes médicas, en donde la posibilidad de operar y compartir equipo médico costoso se puede hacer por medio de una red de cómputo. En este caso se requiere transmitir audio-video y señales de control a los equipos médicos en casos de intervenciones quirúrgicas remotas.

- Mecanismos de control en ambientes de realidad virtual en Ingeniería, estas aplicaciones basadas en realidad virtual requieren de cómputo distribuido y de mecanismos de control eficientes a través de la red para manipular máquinas o robots.

Aquí cabe mencionar que el control por computadora es una estrategia, que como su nombre lo indica, hace uso de una computadora para manipular procesos. Esto se puede lograr mediante 2

estrategias: lazo abierto (sin retroalimentación, Fig. 1), siendo en este caso la computadora un medio de comunicación entre el operador y el proceso; y lazo cerrado (con retroalimentación, Fig. 2). En este último caso la computadora hace las veces de controlador, substituyendo al operador, el cual sólo se encarga de dar las consignas a ser satisfechas por el proceso.

Los aspectos técnicos que tienen que ser tomados en cuenta son: el muestreo y conversión (analógico ↔ digital) de señales, períodos de muestreo, efectos de retención de señal y retardos.

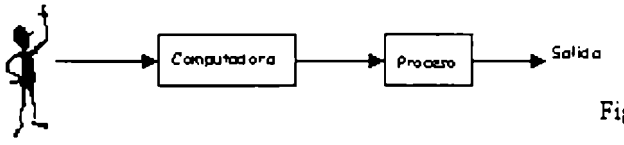

Fig. 1

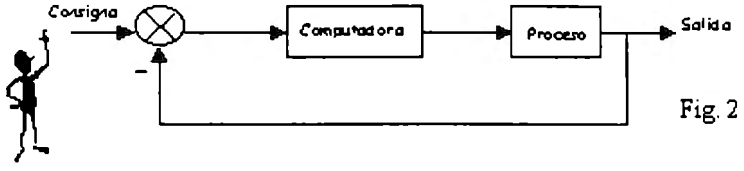

Fig. 2

Todos ellos se generan debido a que las computadoras trabajan con señales discretas (digitales), mientras que (en su mayoría) los procesos reciben y generan señales continuas (analógicas).

La tesis esta organizada como sigue: después de una descripción de las características generales del datagrama IPv6 en el capítulo II, en el capítulo III se mencionan los cambios principales que se deben realizar para crear un socket en IPv6; además, una vez realizada la aplicación cliente/servidor, se crea el protocolo de comunicación entre éstos, el cual se describe en el capítulo IV. En el capítulo V, se muestran los resultados obtenidos en las pruebas operativas y, finalmente, en el capítulo VI se muestran las conclusiones.

CAPÍTULO II

2. CARACTERÍSTICAS DEL IPV6

INTRODUCCIÓN

El presente capítulo muestra las principales características y propiedades del protocolo IPv6. En particular muestra los detalles del formato general del datagrama, del encabezado base y de los encabezados de extensión utilizados para especificar los recursos que el datagrama requiere; así también, se analiza el direccionamiento, enruteamiento y fragmentación necesarios para que los datagramas viajen por la red.

2.1 FORMATO GENERAL DE UN DATAGRAMA IPV6, [3].

Como se muestra en la figura 2.1, un datagrama IPv6 comienza con un encabezado base de tamaño fijo, seguido por ceros o más encabezados de extensión, seguidos a su vez por datos.

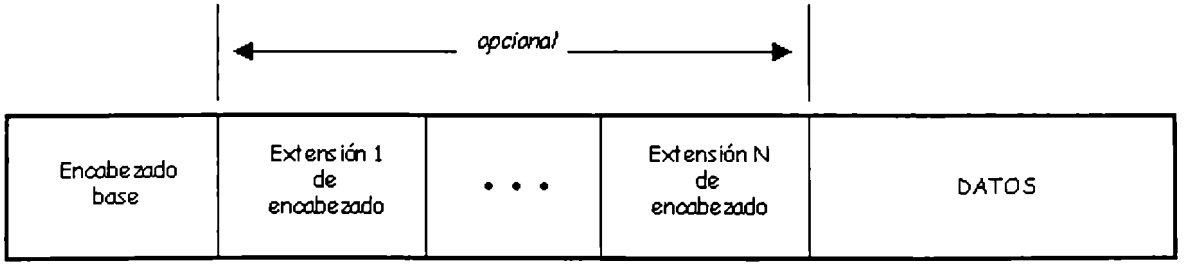

Figura 2.1 Formato general de un datagrama IPv6

2.2 FORMATO DEL ENCABEZADO BASE IPV6, [8].

Un encabezado base del IPv6 contiene menos información que un encabezado de datagrama IPv4. Las opciones y algunos de los campos fijos que aparecen en un encabezado de datagrama del IPv4 se han cambiado por encabezados de extensión en IPv6. La figura 2.2 muestra el contenido y el formato de un encabezado base IPv6.

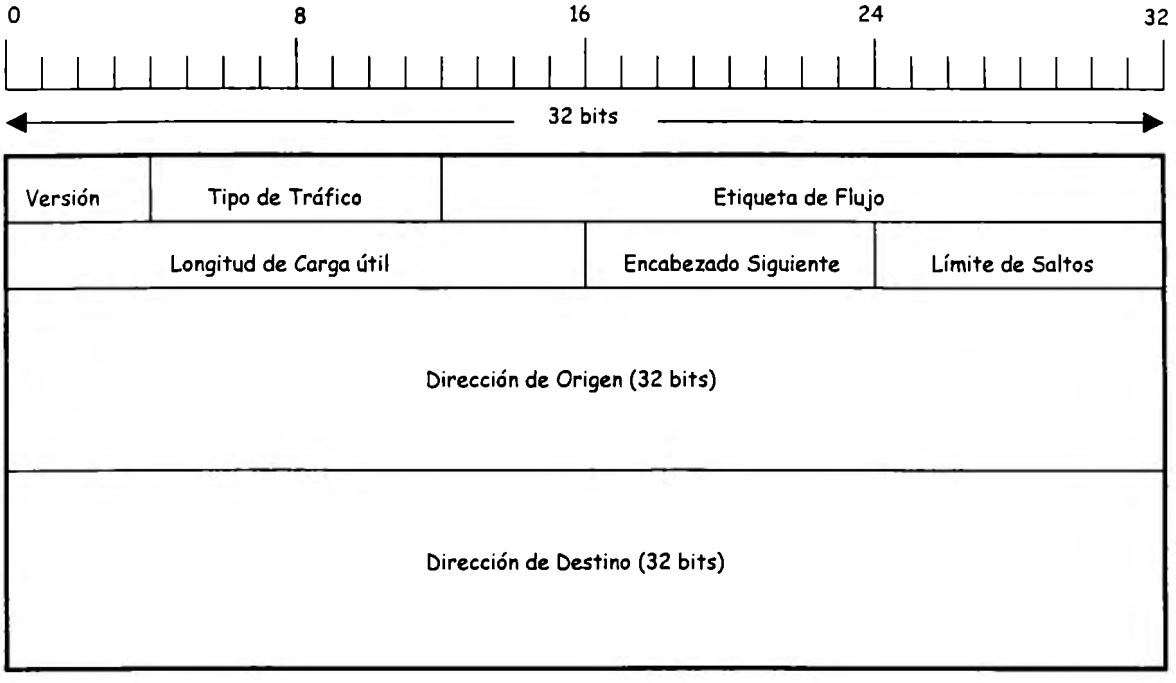

Figura 2.2 Formato del encabezado base del IPv6

Cada campo del datagrama se describe a continuación:

- **Versión**

  Campo de 4 bits. Contiene el número de versión del Protocolo Internet (IP). En este caso el valor de este campo es igual a 6.

- **Tipo de Tráfico**

Campo de 8 bits. Este campo es utilizado por los nodos fuentes y por los ruteadores para identificar y distinguir el tipo de tráfico o prioridad de los paquetes IPv6. Existen requerimientos generales que aplican a este campo, los cuales son:

- La interface para el servicio de IPv6 dentro de un nodo, debe proveer un medio para que el protocolo de capa superior suministre el valor en bits al campo de **Tipo de Tráfico,** en paquetes originados por el protocolo de la capa superior. El valor de *default* para los 8 bits del campo debe ser cero.

- Los nodos que soportan un uso específico, para algunos o todos los bits, del campo **Tipo de Tráfico** pueden cambiar el valor de aquellos bits en el paquete que ellos originan o reciben, conforme son requeridos para ese uso específico. Los nodos pueden ignorar o dejar sin cambio cualquiera de los bits del campo de **Tipo de Tráfico** para los cuales no soportan el uso específico.

- Un protocolo de capa superior no debe asumir que el valor del campo **Tipo de Tráfico** en un paquete recibido, son los mismos valores de un paquete enviado por la fuente.

- **Etiqueta de flujo**

Campo de 20 bits. Este campo contiene información que los ruteadores utilizan para asociar un datagrama con una prioridad y un flujo específico, para darles un trato especial; tal como los servicios de tiempo real. La etiqueta puede usarse también, para asociar los datagramas a trayectorias de redes particulares. El valor de esta etiqueta, se escoge con un valor (pseudo) aleatorio entre 1 y FFFFFF hex. El propósito de esta asignación aleatoria es para formar un conjunto de bits dentro del campo **Etiqueta de Flujo** adecuado para usarse como clave "hash" para los ruteadores, y para buscar el estado asociado al flujo, [9].

- **Longitud de Carga útil**

Campo de 16 bits. Campo que especifica el número de octetos transportados en un datagrama, excluyendo al encabezado mismo.

- **Encabezado Siguiente**

Campo de 8 bits. Este campo indica cuál de los seis encabezados de extensión, de haberlos, sigue a éste. Si este encabezado es el último encabezado de IP, el campo de **Encabezado Siguiente** indica el manejador de protocolo de transporte (por ejemplo TCP, UDP) al que se entregará el paquete. Utiliza los mismos valores que el campo **Protocol** de IPv4.

- **Límite de saltos**

Campo de 8 bits. Se usa para evitar que los paquetes circulen por la red indefinidamente. Se le asigna un valor distinto de 0 y es decrementado en una unidad cada vez que el paquete pasa por un nodo. En la práctica es igual que el campo de **Tiempo de Vida (TTL)** del IPv4.

- **Dirección de Origen**

Campo de 128 bits que contiene la dirección origen. Su longitud es cuatro veces mayor que en la versión 4 del protocolo IP.

- **Dirección de Destino**

Campo de 128 bits de la dirección destino. Su longitud es cuatro veces mayor que en la versión 4 del protocolo IP.

2.3 ENCABEZADOS DE EXTENSIÓN, [8].

Los encabezados de extensión IPv6 son similares a las opciones IPv4. Cada datagrama incluye encabezados de extensión sólo para los recursos que el datagrama utilice. Hay un pequeño número de encabezados de extensión, cada uno de ellos identificados por un valor de **Next Header (Encabezado Siguiente)** distinto.

Salvo excepciones, los encabezados de extensión apenas son examinados por los nodos alcanzados por el paquete a lo largo de su camino hasta que éste llega al nodo identificado por el

campo dirección de destino del encabezado IPv6. En este momento se trata el primer encabezado de extensión, o el encabezado de transporte en el caso de ausencia de encabezados de extensión. El contenido de cada encabezado determinará si es necesario tratar el **Encabezado Siguiente**.

La única excepción es el encabezado de opciones salto a salto, que lleva información que deberá ser examinada por los nodos de la red. Este encabezado, cuando está presente, tiene que seguir inmediatamente al encabezado base IPv6.

Cada encabezado de extensión es de una longitud de un múltiplo de 8 bits, para conservar una alineación de 8 bytes en los encabezados de extensión.

Cuando hay más de un encabezado de extensión en el mismo paquete, los encabezados deben de aparecer en el orden que la figura 2.3 muestra.

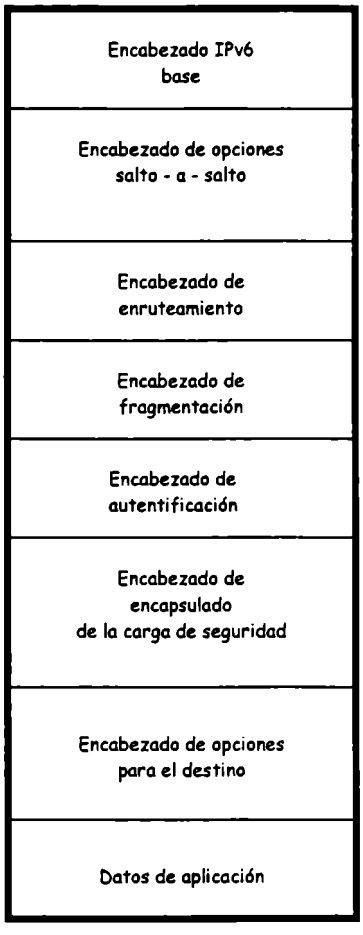

Figura 2.3 Paquete IPv6 con todos los encabezados de extensión

2.3.1 ENCABEZADO DE OPCIONES SALTO - A - SALTO, [8].

El encabezado de opciones salto - a - salto lleva información opcional que, si está presente, debe ser examinada por cada dispositivo de encaminamiento a lo largo de la ruta. Este encabezado tiene el formato que se muestra en la figura 2.4.

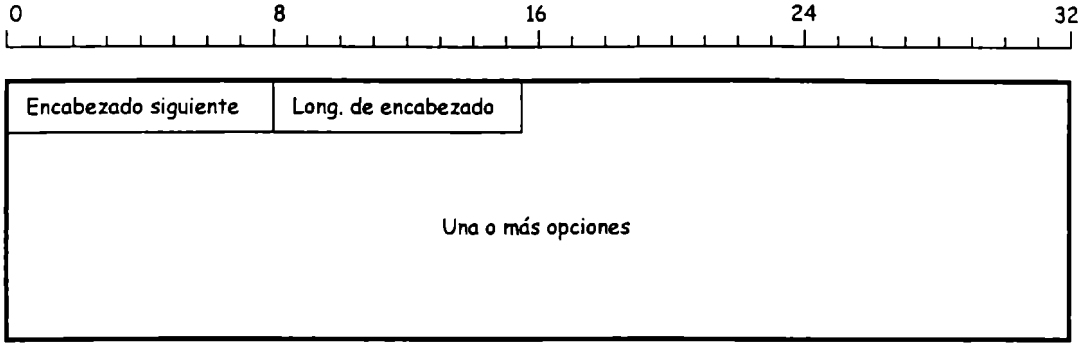

Figura 2.4 Encabezado de opciones salto - a - salto

Donde:

- **Encabezado Siguiente**

Campo de 8 bits (valor 0). Identifica el tipo de encabezado que sigue inmediatamente a este. Los valores son idénticos al campo **Protocol** de la verisón IPv4.

- **Longitud de encabezado**

Campo de 8 bits. Longitud del encabezado en unidades de 64 bits, sin incluir los primeros 64 bits.

- **Opciones**

Campo de longitud variable. Este campo contiene una o varias opciones codificadas en **TLV** (Type-Lenght-Value). Cada opción se expresa mediante tres subcampos: tipo de opción (8 bits), que identifica la opción; longitud (8bits), que especifica la longitud en octetos del campo de datos de la opción; y datos de opción, que es una especificación de longitud variable de la opción.

Se utilizan los cinco bits menos significativos del campo tipo de operación para especificar una opción particular. Los bits más significativos indican la acción que tiene que realizar un nodo que no reconoce el tipo de opción, de acuerdo a:

00 Ignorar esta opción y continuar procesando el encabezado.

01 Descartar el paquete.

10 Descartar el paquete y enviar un mensaje **ICMP,** [21], de problema de parámetro, código 2, a la dirección origen del paquete, indicando el tipo de opción no reconocida.

11 Descartar el paquete solamente si la dirección destino del paquete no es una dirección *multicast*, enviar un mensaje **ICMP** de problema de parámetro, código 2, a la dirección origen del paquete, indicando el tipo de opción no reconocida.

El tercer bit de mayor peso, indica si los datos específicos de la opción pueden cambiar durante el recorrido del paquete. Esto es útil cuando existe un encabezado de autentificación. Los valores son:

0 datos de la opción que NO pueden cambiar en ruta

1 datos de la opción que pueden cambiar en ruta

Además de las opciones con la estructura descrita, existe una opción especial, la *Carga Jumbo*. La estructura de esta opción se muestra en la figura 2.5, [1].

Identificador 194	Long. de opciones	Longitud de Carga Jumbo
8 bits	8 bits	32 bits

Figura 2.5 Estructura de la *Carga Jumbo*

La opción *Carga Jumbo*, es utilizada para enviar paquetes con cargas superiores a los 65,535 octetos. La longitud especificada por la *Carga Jumbo* es el tamaño total del paquete, excluyendo el encabezado base IPv6, e incluyendo el encabezado de opciones salto - a - salto.

La longitud determinada debe ser siempre superior a 65,535, si se recibe un paquete con una *Carga Jumbo* que indique un tamaño de paquete igual o menor a 65,535, **ICMP** se encargará de enviar un error.

Cada paquete cuya longitud esté especificada por una opción *Carga Jumbo*, debe tener un valor de 0 en el campo longitud de la carga en el encabezado base IPv6, además, la opción *Carga Jumbo* no puede ser usada en un paquete conteniendo un fragmento. El incumplimiento de cualquiera de estas restricciones provocará un error **ICMP**.

2.3.2 ENCABEZADO DE ENRUTEAMIENTO, [8].

El encabezado de enruteamiento es utilizado por un emisor para establecer una lista de uno o más nodos intermedios que debe seguir el paquete para llegar a su destino. Este encabezado está diseñado para soportar el Protocolo de Enruteamiento a Petición del Emisor (Source Demand Routing Protocol, SDRP, [11]). El formato de este encabezado se muestra en la figura 2.6.

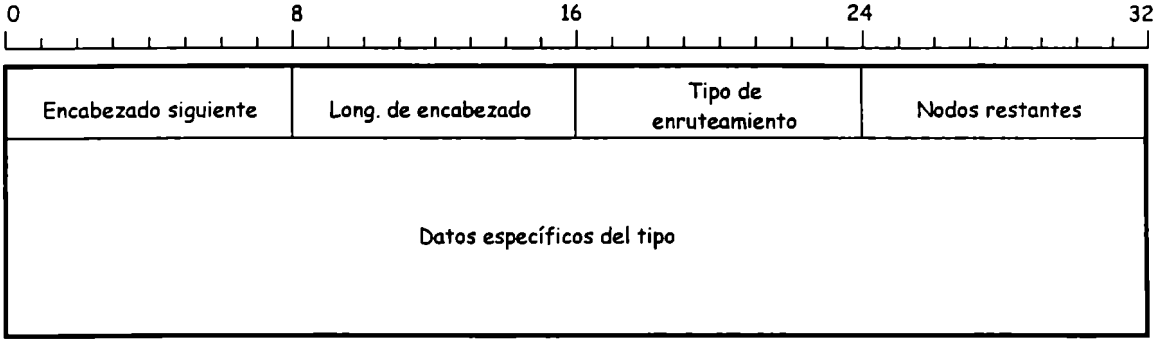

Figura 2.6 Encabezado de enruteamiento

Donde:

- **Encabezado Siguiente**

Campo de 8 bits (valor 43). Identifica el tipo de **Encabezado Siguiente**.

- **Longitud de encabezado**

Campo de 8 bits. Indica la longitud del encabezado en octetos, sin incluir los ocho primeros.

- **Tipo de enruteamiento**

Campo de 8 bits. Indica el tipo particular de encabezado de enruteamiento.

- **Nodos restantes**

Campo de 8 bits. Indica el número de nodos que restan por visitar, siempre sobre los nodos marcados explícitamente.

- **Datos**

Campo de longitud variable, siempre múltiplo de 8 (en octetos), y su formato viene determinado por el tipo de enruteamiento específico.

Si un nodo encuentra un paquete con tipo de enruteamiento desconocido, tomará alguna de estas dos medidas:

- Si el número de nodos restantes es cero, se ignora el encabezado y se pasa al **Encabezado Siguiente**.

- Si el número de nodos restantes NO es cero, se descarta el paquete y se enviará un error **ICMP**.

El formato del Tipo 0 (tipo de enruteamiento = 0) se muestra en la figura 2.7.

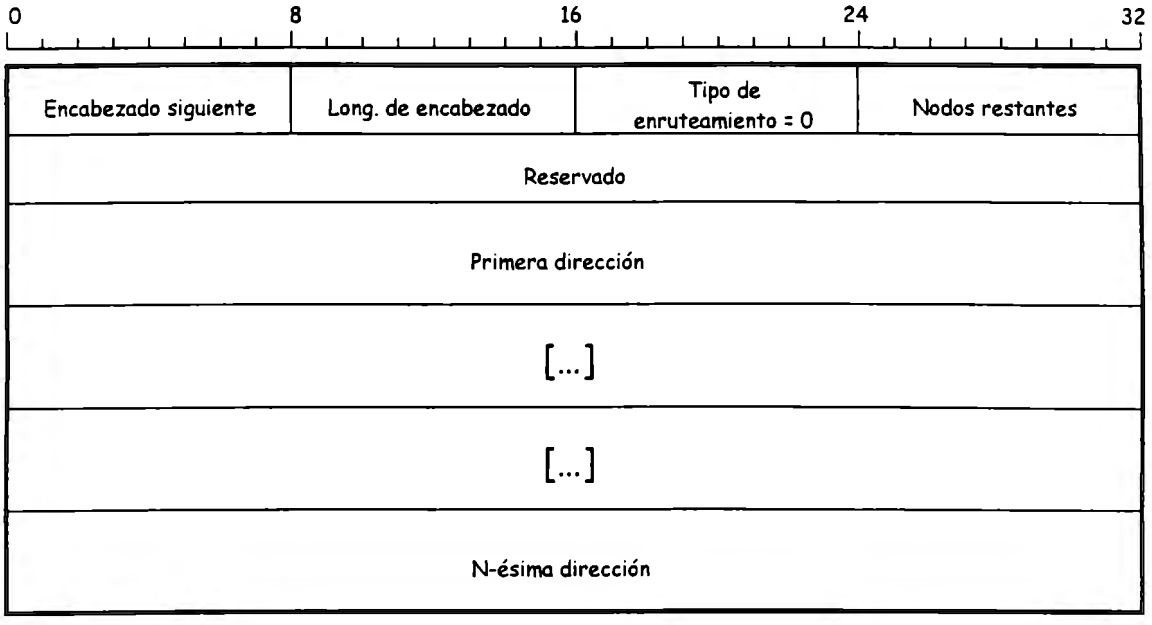

Figura 2.7 Encabezado de enruteamiento Tipo 0

Donde:

- **Encabezado Siguiente**

Campo de 8 bits. Identifica el tipo de **Encabezado Siguiente**.

- **Longitud de encabezado**

Campo de 8 bits. Indica la longitud del encabezado en octetos, sin incluir los 8 primeros octetos, para el Tipo 0, es igual al doble de direcciones especificadas, y debe ser un número par menor o igual a 46, de igual forma, el máximo número de nodos que puede especificarse es 23.

- **Tipo de enruteamiento**

Campo de 8 bits, en este caso el valor de este campo es igual a 0.

- **Nodos restantes**

Campo de 8 bits. Indica el número de nodos que restan por visitar, siempre sobre los nodos marcados explícitamente.

- **Reservado**

Campo de 32 bits. El origen inicializa este campo en 0, ignorado en el destino.

- **Dirección siguiente**

Campo de 32 bits, las direcciones a visitar se especifican una tras otra, se enumeran de 1 a *n*, y pueden aparecer como máximo 23.

En el Tipo 0 no pueden aparecer direcciones *multicast*. Si un paquete IPv6 tiene como destino una dirección *multicast*, no puede contener un encabezado de enruteamiento de Tipo 0.

2.3.3 ENCABEZADO DE FRAGMENTACIÓN, [8].

El encabezado de fragmentación es utilizado por el nodo origen, no por los ruteadores que intervienen a lo largo del camino del paquete. Para obtener las ventajas completas del entorno de interconexión, un nodo debe ejecutar un algoritmo de obtención de la ruta, lo que permite conocer la unidad máxima de transferencia (**MTU**, Maximun Transfer Unit) permitida por cada subred en la ruta. Con este conocimiento, el nodo origen fragmentará, según se requiera para cada dirección destino dada. Si no se ejecuta este algoritmo, el origen debe limitar todos los paquetes a 1,280 octetos, que es la mínima **MTU** que admiten las subredes, [12].

Este encabezado contiene los siguientes campos (figura 2.8).

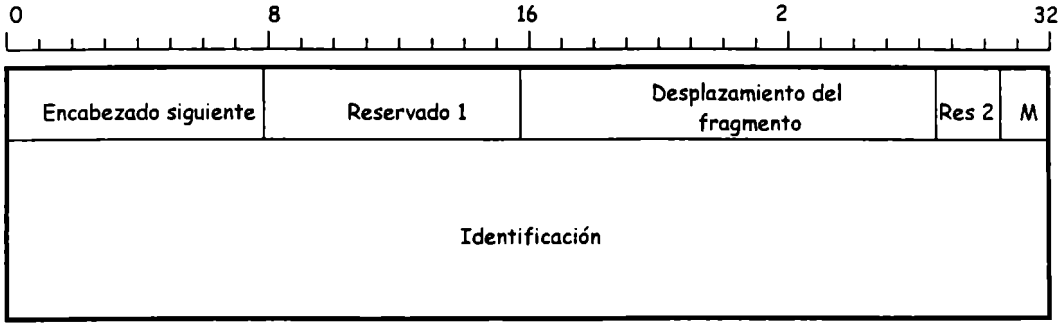

Figura 2.8 Encabezado de fragmentación

Donde:

- **Encabezado Siguiente**

Campo de 8 bits (valor 44). Indica el tipo de **Encabezado Siguiente**.

- **Reservado 1**

Campo de 8 bits, el origen inicializa este campo en 0, ignorado en el destino.

- **Desplazamiento del fragmento**

Campo de 13 bits. Indica donde se sitúa en el paquete original la carga útil de este fragmento. Se mide en unidades de 64 bits. Esto implica que los fragmentos (excepto el último) deben contener un campo de datos con una longitud múltiplo de 64 bits.

- **Reservado 2**

Campo de 2 bits, el origen inicializa este campo en 0, ignorado en el destino.

- **Indicador M**

Campo de 1 bit, indica si existen más fragmentos. El 1 indica más fragmentos y el 0 indica que es el último fragmento.

- **Identificación**

Campo de 32 bits. Es utilizado para identificar de forma única el paquete original. El identificador debe ser único para la dirección origen, dirección destino y para el valor del **Encabezado Siguiente** del paquete, durante el tiempo que el paquete permanece en Internet.

2.3.4 ENCABEZADO DE AUTENTIFICACIÓN, [13].

El encabezado de autentificación es utilizado para asegurar la integridad de los paquetes. Para no rechazar los paquetes es necesario ejecutar un algoritmo de autentificación sobre este encabezado. El encabezado de autentificación tiene el siguiente formato (figura 2.9).

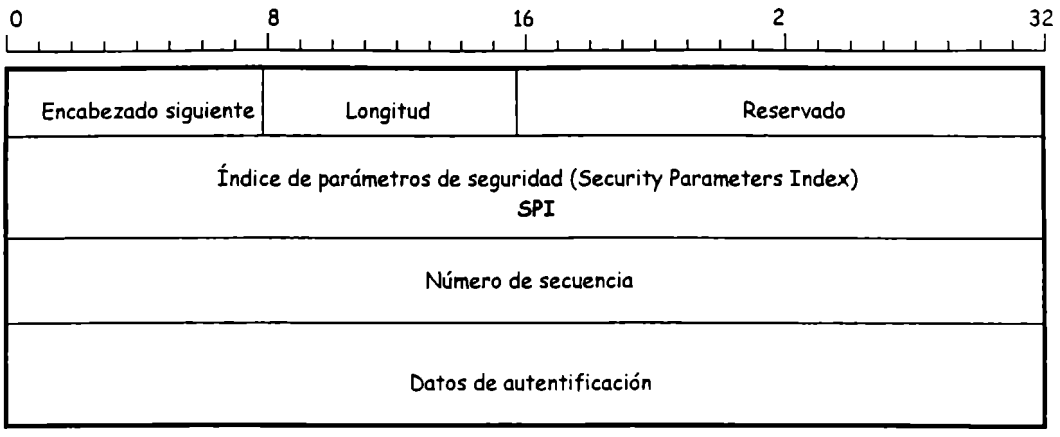

Figura 2.9 Encabezado de autentificación

Donde:

- **Encabezado Siguiente**

Campo de 8 bits (valor 51). Identifica el tipo de **Encabezado Siguiente**. Los valores son idénticos a los del campo **Protocol** de IPv4.

- **Longitud**

Campo de 8 bits, especifica la longitud del campo, debe ser múltiplo de 8 octetos.

- **Reservado**

Campo de 16 bits, el origen inicializa este campo en 0, ignorado en el destino.

- **Índice de Parámetros de Seguridad (Security Parameters Index, SPI)**

Campo de 32 bits. Este campo contiene un valor pseudoaleatorio, identifica la asociación de seguridad asignada a este paquete. Este valor es proporcionado por la IANA (Internet Assigned Numbers Authority).

- **Número de Secuencia**

Campo de 32 bits. Este campo contiene un contador de incremento monotónico (secuencia de números). Es obligatorio y está siempre presente aún cuando el receptor no es elegible para habilitar el servicio de *anti-replay* para un Security Architecture específico. El procesamiento del campo **Número de Secuencia** es a discreción del

receptor, esto es, el emisor siempre debe transmitir ese campo, pero el receptor no necesita hacer uso de él.

- **Datos de Autentificación**

Campo de longitud variable. Contiene información sobre el algoritmo específico necesario para autentificar el origen del paquete y para asegurar su integridad con respecto al tipo de seguridad asociado.

2.3.5 ENCABEZADO DE ENCAPSULADO DE LA CARGA ÚLTIL DE SEGURIDAD (ESP), [14].

La función de este campo es dar confidencialidad e integridad a los datos, esto se hará encapsulando los datos a proteger y colocándolos en la parte del encabezado de seguridad. Dependiendo de las exigencias de seguridad del usuario, se podrá encapsular la trama del nivel de transporte (**UDP** o **TCP**) o el datagrama entero. Este enfoque de encapsulación es necesario para asegurar una confidencialidad completa al datagrama original. Si está presente el encabezado de seguridad, este será el último campo no encapsulado de un paquete.

El formato de este encabezado lo muestra la figura 2.10.

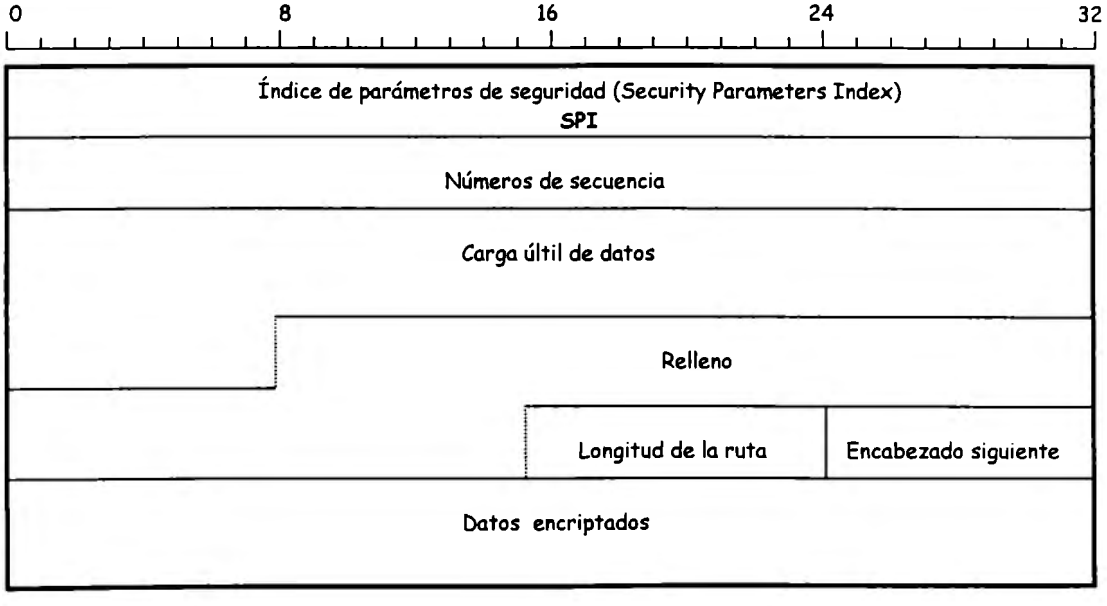

Figura 2.10 Encabezado de encapsulado de la carga de seguridad

Donde:

- **Índice de Parámetros de Seguridad (SPI)**

Campo de 32 bits, que en combinación con la dirección IP de destino y el protocolo de seguridad (ESP), identifica de manera única la *Asociación de Seguridad* para este datagrama. El conjunto de valores, de 1 a 255, del SPI está reservado para el IANA para uso futuro; un valor reservado del SPI no será normalmente asignado por el IANA a menos que, el uso del valor asignado del SPI esté especificado en un RFC. Este es normalmente seleccionado por el sistema de destino basado en el establecimiento de un SA (Security Architecture), [15]. El campo de SPI es obligatorio.

Un valor de cero del SPI se reserva para el uso local de una implementación específica, y no debe ser transmitido.

- **Número de Secuencia**

Campo de 32 bits. Este campo contiene un contador de incremento monotónico (secuencia de números). Es obligatorio y está siempre presente aún cuando el receptor no es elegible para habilitar el servicio de *anti-replay* para un SA específico. El procesamiento del campo **Número de Secuencia** es a discreción del receptor, esto es, el emisor siempre debe transmitir ese campo, pero el receptor no necesita hacer uso de él.

- **Dato de Carga útil**

Campo variable. Este campo contiene datos descritos por el campo de **Encabezado Siguiente**. El **Dato de Carga útil** es obligatorio y es un número entero. Si el algoritmo usado para encriptar la carga útil, requiere sincronización criptográfica de datos, entonces, este dato puede ser cargado explícitamente en el campo de carga útil.

- **Datos de relleno**

Algunos factores requieren o motivan el uso de un campo de relleno.

- Si se usa un algoritmo criptográfico que requiere de *plaintext* para hacer un múltiplo de algún número de bytes, el campo de relleno es utilizado para llenar el *plaintext* al tamaño requerido por el algoritmo.

- El relleno también puede ser requerido, independientemente de los requerimientos del algoritmo criptográfico, para asegurar que el *ciphertext* resultante termine en 4 bytes.

- **Longitud del campo de relleno**

El campo de longitud del campo de relleno, indica el número de bytes de relleno que lo preceden inmediatamente. El rango de valores válido es de 0-255, donde un valor de cero indica que no hay relleno. La longitud del campo de relleno es obligatoria.

- **Encabezado Siguiente**

Campo de 8 bits (valor 50). Identifica el tipo de dato contenido en el campo de **Dato de carga útil**. El valor de este campo es seleccionado del conjunto de números de protocolos IP definido en el más reciente IANA. Este campo es obligatorio.

- **Datos Encriptados**

Campo de longitud variable. Este campo contiene un valor de verificación de integridad (ICV) calculado sobre el paquete ESP menos el dato de autentificación. La longitud del campo es especificada por la función seleccionada de autentificación. El campo **de Datos Encriptados** es opcional y es incluido, únicamente si el servicio de autentificación ha sido seleccionado por el SA en cuestión. Las especificaciones del algoritmo de autentificación deben especificar la longitud del ICV y las reglas de comparación, y paso de procesamiento para la validación.

2.3.6 ENCABEZADO DE OPCIONES PARA EL DESTINO, [8].

El encabezado de opciones para el destino lleva información opcional que, si está presente, se examina por el nodo destino del paquete. El formato de esta encabezado tiene el siguiente formato (figura 2.11).

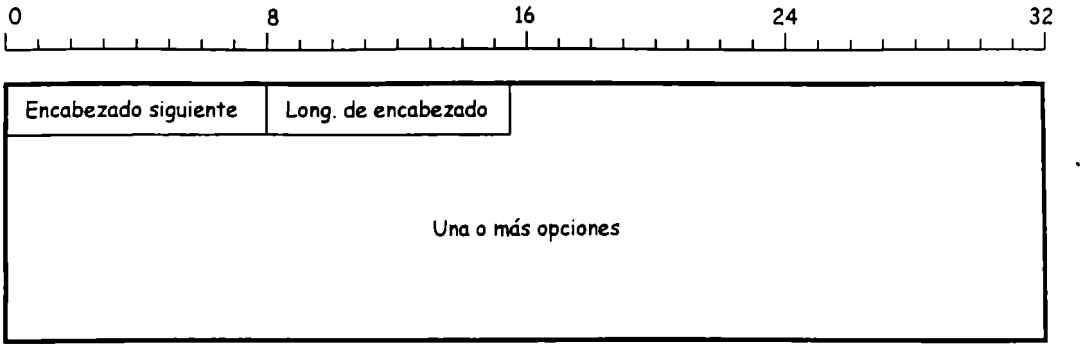

Figura 2.11 Encabezado de opciones para el destino

Donde:

- **Encabezado Siguiente**

  Campo de 8 bits (valor 60). Indica el tipo de **Encabezado Siguiente**.

- **Longitud de encabezado**

  Campo de 8 bits. Indica la longitud del encabezado en unidades de 8 octetos, sin incluir los 8 primeros.

- **Opciones**

  Campo de longitud variable, siempre alineada a 8 octetos. Contiene una o más opciones de la estructura descrita en el apartado 2.3.1.

Hay dos posibles formas de codificar opciones **TLV**, como una opción en el encabezado **Opciones en destino**, o como un encabezado extendido aparte. Elegir una forma u otra dependerá de cual sea la acción deseada del destino, si no entiende la información.

Si se quiere que el destino, en caso de no reconocer la opción, descarte el paquete y envíe un error **ICMP** (sólo si la dirección destino no es *multicast*), entonces la opción deberá ser codificada en un encabezado extendido aparte.

Si se quiere cualquier otra opción, entonces la opción se codificará dentro de un encabezado **Opciones en destino**, y la acción especificada vendrá dada por los dos bits de mayor peso del campo **Tipo de opción**, en la forma descrita en el apartado 2.3.1.

2.3.7 FIN DE ENCABEZADOS, [8].

Cuando no hay más encabezados de extensión, el último **Encabezado Siguiente** deberá contener el valor 59. Si el campo de longitud indica la presencia de más octetos después de un encabezado cuyo **Encabezado Siguiente** sea igual a 59, se ignorará el resto.

2.4 DIRECCIONAMIENTO IPV6, [16 Y 17].

Las direcciones de IPv6 tienen una longitud de 128 bits. Existen tres tipos básicos de direcciones para identificar a una interfaz en concreto o a un grupo de interfaces. Los bits de mayor peso de los que componen la dirección IPv6 son los que permiten distinguir el tipo de dirección, empleándose un número variable de bits para cada caso. Estos tres tipos de direcciones son:

- **Direcciones *unicast***

  Identificador para una sola interfaz de la red. Las direcciones *unicast* que se encuentran definidas actualmente están divididas en varios grupos. Dentro de este tipo de direcciones se encuentra también un formato especial que facilita la compatibilidad con las direcciones de la versión 4 del protocolo IP.

- **Direcciones *anycast***

  Identificador para un conjunto de interfaces. Un paquete IPv6 con una dirección destino *anycast* es enruteado a una y sólo una de las interfaces identificadas por la dirección. El paquete será enruteado a la interfaz más cercana, de acuerdo a la trayectoria más corta.

❖ **Direcciones *multicast***

Identificador para un conjunto de interfaces. Un paquete IPv6 con una dirección destino *multicast* es enruteado a todos y cada uno de las interfaces identificadas por la dirección.

Una dirección *multicast* consta de un prefijo de 8 bits, un campo de indicadores de 4 bits, un campo de ámbito de 4 bits y un identificador de grupo de 112 bits. Actualmente, el campo de indicadores está formado por 3 bits cero seguido del bit T con el siguiente significado:

- T = 0: indica una dirección *multicast* asignada permanentemente, o bien asignada por la autoridad de asignación de números Internet global.
- T = 1: indica que es una dirección *multicast* transitoria o no asignada permanentemente.

El valor del campo de ámbito se usa para limitar el ámbito del grupo de *multicast*. Los valores posibles son:

0	Reservado
1	Nodo local
2	Enlace local
3	No asignado
4	No asignado
5	Zona local
6	No asignado
7	No asignado
8	Organización local
9	No asignado
10	No asignado
11	No asignado
12	No asignado
13	No asignado
14	Global
15	Reservado

❖ **Direcciones *broadcast***

Las direcciones *broadcast* NO existen en IPv6. Sin embargo, funciones similares se pueden generar con direcciones *multicast*.

Ahora bien, el espacio de las direcciones del IPv6 se divide como se muestra a continuación:

PREFIJO (BINARIO)	USO	FRACCIÓN
0000 0000	Reservado (incluye IPv4)	1/256
0000 0001	No asignado	1/256
0000 001	Direcciones OSI NSAP	1/128
0000 010	Direcciones IPX de Novell Netware	1/128
0000 011	No asignado	1/128
0000 1	No asignado	1/32
0001	No asignado	1/16
001	No asignado	1/8
010	Direcciones basadas en proveedor	1/8
011	No asignado	1/8
100	Direcciones basadas en geografía	1/8
101	No asignado	1/8
110	No asignado	1/8
1110	No asignado	1/16
1111 0	No asignado	1/32
1111 10	No asignado	1/64
1111 110	No asignado	1/128
1111 110 0	No asignado	1/512
1111 1110 10	Direcciones de enlace de uso local	1/1024
1111 1110 11	Direcciones de instalación de uso local	1/1024
1111 1111	Multitransmisión	1/256

Las direcciones que comienzan con 80 ceros se reservan para direcciones IPv4. Se reconocen dos variantes distinguidas por los siguientes 16 bits. Estas variantes se relacionan con la manera en que se enviarán en túnel los paquetes IPv6 a través de la infraestructura IPv4

existente, [10]. A continuación se muestran algunos ejemplos de direccionamientos definidos anteriormente.

- Dirección *unicast*: 1080:0:0:0:8:800:200C:417A
- Dirección *multicast*: FF01:0:0:0:0:0:0:101
- Dirección *loopback*: 0:0:0:0:0:0:0:1
- Dirección no especificada: 0:0:0:0:0:0:0:0

2.5 REPRESENTACIÓN DE DIRECCIONES IPV6, [16 y 17].

Existen tres formas convencionales para representar las direcciones IPv6:

1.- La forma más indicada es mediante la estructura x:x:x:x:x:x:x:x, donde las x representan los valores hexadecimales de cada bloque de 16 bits de la dirección.

Ejemplos:

FEDC:BA98:7654:3210:FEDC:BS98:7654:3210

1080:0:0:8:800:200C:417A

Hay que destacar que no es necesario escribir todos los ceros que hay por delante de un número hexadecimal en un campo individual, pero se ha de tener por lo menos una cifra en cada campo.

2.- La segunda forma para representar las direcciones, es agrupando largas series de 0's, para hacer más legibles las direcciones, el uso de "**::**" indica uno o varios grupos de 16 bits iguales a 0.

Ejemplo:

La dirección *multicast* siguiente: FF01:0:0:0:0:0:43 se puede representar también de la siguiente manera: FF01::43.

3.- La tercera forma para representar las direcciones IPv6 que contengan direcciones IPv4, es colocando los 2 últimos bloques de 16 bits como 4 bloques de 8 bits mostrando sus valores en decimal.

Ejemplo:

0:0:0:0:0:FFFF:129.144.52.38 ó ::FFFF:129.144.52.38

CONCLUSIONES

Durante el desarrollo de este capítulo, se puede observar que el protocolo IPv6 conserva muchas de las características de diseño que hacen a IPv4 tan exitoso. Al igual que IPv4, IPv6 opera sin conexiones, es decir, cada datagrama tiene una dirección destino y se enruta independientemente; también permite al emisor seleccionar el tamaño de un datagrama y el máximo número de saltos que un datagrama puede realizar antes de ser eliminado. Además, el IPv6 utiliza direcciones más largas y añade algunas características nuevas, tales como: el formato de encabezado, encabezado de extensión, manejo de audio y video, autentificación y privacidad. Algo muy importante, el IPv6 revisa completamente el formato de los datagramas, remplazando el campo de opción de longitud variable del IPv4 por una serie de encabezados de formato fijo.

Por lo tanto, una vez analizados los formatos de los encabezados de IPv6, la forma de direccionamiento, enruteamiento y fragmentación se pueden aprovechar las numerosas características que hacen tan atractivo al protocolo, como el soporte de comunicaciones en tiempo real, la autoconfiguración de sistemas y seguridad. Para este proyecto en particular, se propone la creación de un mecanismo de control eficiente a través de la red para posicionar una video-cámara, ya que gracias a la introducción de flujos etiquetados (con prioridades) y los servicios de restricciones de tiempo real, hacen que este protocolo pueda tener la capacidad de tratar el tráfico sensible a retardos más eficientemente que IPv4.

CAPÍTULO III

3. APLICACIÓN CLIENTE-SERVIDOR IPV6

INTRODUCCIÓN

En este capítulo, basado en [18 y 20], se describe la estructura del direccionamiento del socket para IPv6. También se introducen las nuevas características de escritura de la etiqueta y prioridad del flujo, así como los cambios necesarios que son requeridos, para que una aplicación cliente-servidor soporte éste nuevo protocolo y opere en conjunto con el protocolo IPv4.

3.1 INTERFAZ SOCKET

Un socket es un punto de comunicación a través del cual un proceso puede enviar o recibir información. Los procesos tratan a los sockets como descriptores de archivos, lo que provoca que las operaciones de envío/recepción de mensajes sean parecidas a la lectura/escritura en un archivo.

El tipo de socket define la forma en la que se transfiere la información a través del socket. En este caso en particular, utilizaremos los sockets tipo *stream*, los cuales se definen de la siguiente manera.

3.2 SOCKETS *STREAM* (TCP, TRANSPORT CONTROL PROTOCOL)

Los sockets *stream* utilizan un servicio orientado a conexión, donde los datos se transfieren sin encapsulamiento en registros o bloques. Si se rompe la conexión entre los procesos, éstos serán informados.

El protocolo de comunicaciones con *streams* es un protocolo orientado a conexión, ya que para establecer una comunicación utilizando el protocolo TCP, hay que establecer en primer lugar una conexión entre un par de sockets. Esto es, mientras uno de los sockets atiende peticiones de conexión (servidor), el otro solicita una conexión (cliente). Una vez que los dos sockets estan conectados, se pueden utilizar para transmitir datos en ambas direcciones.

La razón por la que se escogió el socket tipo *stream*, es porque el protocolo TCP es un protocolo *ordenado*, lo que garantiza que todos los paquetes que se envíen serán recibidos en el socket destino en el mismo orden en que se han enviado. Además, es el más indicado para la implementación de servicios de red, como un control remoto, dado que UDP es un protocolo con un servicio de entrega de paquetes sin conexión no confiable que resulta en posibles desordenamientos, duplicación o pérdida de paquetes.

Para la programación de los sockets en IPv6, es necesario realizar algunos cambios para que soporte éste nuevo protocolo, y opere en conjunto con el protocolo IPv4. Las estructuras y funciones que requieren de estos cambios son:

- Estructura: **sockaddr_in**, esta estructura determina el espacio para la dirección fuente y destino del protocolo. Actualmente contiene 4 octetos para cada un de las direcciones en IPv4, sin embargo, no es suficiente espacio para las direcciones de IPv6 ya que se requiere de 16 octetos para ambas direcciones.

- Función: **gethostbyname ( )**, esta función obtiene información a partir del nombre del *host*. Se requiere de modificaciones para que soporte ambos protocolos, así como las nuevas aplicaciones de IPv6.

- Función: **gethostbyaddr ( )**, esta función obtiene información de un *host* a partir de una dirección. Se requiere de modificaciones para que soporte ambos protocolos.

- Funciones: **inet_ntoa ( ), inet_addr ( )**; estas funciones convierten las direcciones binarias a una forma estándar legible. Estas funciones contienen 32-bits para las direcciones de IPv4, por lo que es necesario diseñar dos funciones análogas para convertir tanto direcciones IPv4 como IPv6.

3.3 CAMBIOS EN LA INTERFAZ SOCKET IPV6

Como se mencionó anteriormente, nuevas interfaces son necesarias para soportar IPv6, e incluyen: escritura de la etiqueta, prioridad, y los campos de la cabecera del límite del salto. Las nuevas opciones del socket son necesarias para controlar el envío y la recepción de los paquetes del *multicast* IPv6. Los cambios que se requieren para la interfaz socket IPv6 que utilizaremos para la aplicación del control de la cámara, son los siguientes:

- Familias de direcciones y protocolos
- Estructura de direcciones
- Estructura de direcciones del socket
- Funciones del socket
- Direcciones wildcard IPv6
- Traducción "Nodename to Address"
- Funciones de conversión de direcciones

Cada uno de estos cambios se describen a continuación.

3.3.1 FAMILIAS DE DIRECCIONES Y PROTOCOLOS

La nueva familia de direcciones, **AF_INET6**, está definida en *<sys/socket.h>*. Las definiciones de **AF_INET6** se distinguen entre la estructura de datos original **sockaddr_in** y la nueva estructura de datos **sockaddr_in6**.

Por otro lado, la nueva familia de protocolos, **PF_INET6**, está definida en *<sys/socket.h>*. Usualmente se define teniendo el mismo valor que corresponde a la familia de direcciones.

```
# define    PF_INET6  AF_INET6
```

El **PF_INET6** se utiliza en el primer argumento de la función del **socket( )** para indicar que se está creando un socket IPv6.

3.3.2 ESTRUCTURA DE DIRECCIONES

La nueva estructura de datos para llevar a cabo un sólo direccionamiento IPv6 se define como sigue:

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

        struct in6_addr {
            uint8_t  s6_addr[16];      /* IPv6 address */
        }
```

Esta estructura contiene un arreglo de 16 elementos de 8-bits cada uno, que hacen un direccionamiento de 128-bits para IPv6.

3.3.3 ESTRUCTURA DE DIRECCIONES DEL SOCKET

La estructura del **sockaddr_in**, es la estructura de datos de direcciones para IPv4. Se utiliza para pasar los direccionamientos entre las aplicaciones y el sistema en las funciones del socket. La estructura siguiente se define para llevar los direccionamientos IPv6:

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

struct sockaddr_in6 {
     sa_family_t      sin6_family;     /* AF_INET6 */
     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    /*conjunto de interfaces para un
                                       alcance*/
       };
```

Donde:

- Campo **sin6_family:** identifica una estructura **sockaddr_in6**. Este campo se sobrepone al campo **sa_family** cuando el buffer borra una estructura de datos del **sockaddr**. El valor de este campo debe ser **AF_INET6**.

- Campo **sin6_port:** campo de 16-bits que corresponden al número de puerto de TCP o UDP.

- Campo **sin6_flowinfo:** campo de 32-bits y contiene dos partes de información: el tipo de tráfico y la etiqueta de flujo. Este campo debe ir en cero para implementaciones de prioridad usando la estructura **sockaddr_in6** para una aplicación de operación de recepción.

- Campo **sin6_addr:** campo de 128-bits para el direccionamiento de IPv6. Este campo es una estructura sencilla **in6_addr**, descrita anteriormente.

- Campo **sin6_scope_id:** campo de 32 bits, identifica el conjunto de interfaces apropiadas para alcanzar las direcciones transportadas en el campo **sin6_addr**.

El orden de los elementos en la estructura es diseñado específicamente para alinearse a 64-bits. Esto está hecho para obtener un funcionamiento óptimo en arquitecturas de 64-bits.

3.3.4 FUNCIONES DEL SOCKET

Las aplicaciones cliente-servidor llaman a la función **socket( )** para crear una descripción del socket, y éste represente un punto final de la comunicación. Los argumentos de la función del **socket( )** indican al sistema qué protocolo se está usando, y qué estructura de direccionamiento del formato será utilizado en funciones subsecuentes.

De esta forma, para crear un socket IPv6/TCP, haremos la llamada de la siguiente manera:

```
s = socket(PF_INET6, SOCK_STREAM, 0);
```

Una vez que la aplicación haya creado un socket **PF_INET6**, debe utilizar la estructura del direccionamiento **sockaddr_in6** para pasar los direccionamientos al sistema.

Por lo tanto, la creación del socket IPv6 en la implementación cliente-servidor desarrollada, queda de la siguiente forma:

SERVIDOR

`if ( (sd = socket(PF_INET6,SOCK_STREAM, 0)) == -1) {`
`perror("socket");`
`exit(1);`
`}`

CLIENTE

`if ( (sd = socket(PF_INET6, SOCK_STREAM, 0)) == -1) {`
`perror("socket");`
`exit(1);`
`}`

Tabla 3.1 Creación del socket IPv6, cliente-servidor

Las funciones que la aplicación utiliza para pasar los direccionamientos en el sistema son:

```
bind( )
connect( )
sendmsg( )
sendto( )
```

El sistema utilizará la estructura del direccionamiento **sockaddr_in6** para regresar las direcciones a las aplicaciones que están utilizando los socketes **PF_INET6**. Las funciones que regresan un direccionamiento del sistema a una aplicación son:

```
accept( )
recvfrom( )
recvmsg( )
getpeername( )
getsockname( )
```

Estas funciones no necesitan cambios de sintáxis para soportar IPv6.

3.3.5 DIRECCIONES WILDCARD IPV6

Dado que la dirección IPv6 es una estructura (struct **in6_addr**), una constante simbólica puede ser usada para inicializar la variable de la dirección IPv6, pero no puede ser usada para una asignación. Por lo tanto, los sistemas proveen una dirección *wildcard* IPv6, en dos formas.

La primera forma, que es la que utilizaremos para la aplicación, es considerando a la dirección *wildcard* como una variable global llamada **in6addr_any**, la cual está en la estructura **in6_addr**, y definida en la librería *<netinet/in.h>*. La variable **in6addr_any** es usada de la misma forma que la variable **INADDR_ANY** de IPv4. Por ejemplo, si se deja que el sistema seleccione la dirección fuente, la aplicación utilizará el siguiente código:

```
struct sockaddr_in6   sin6;
   . . .
sin6.sin6_family = AF_INET6;
sin6.sin6_flowinfo = 0;
sin6.sin6_port = htons(23);
sin6.sin6_addr = in6addr_any;  /*structure assignment */
   . . .
  if (bind(s, (struct sockaddr *) &sin6, sizeof(sin6)) == -1)
```

Siguiendo la estructura **sockaddr_in6** y las direcciones **wildcard** IPv6, la implementación cliente-servidor desarrollada, quede de la siguiente forma:

SERVIDOR	CLIENTE

Sin6.sin6_family = AF_INET6;
Sin6.sin6_port = htons(PUERTO);
Sin6.sin6_flowinfo = 0;
Sin6.sin6_addr = in6addr_any;
	pin.sin6_family = AF_INET6;
	pin.sin6_port = htons(PUERTO);
	sin6.sin6_flowinfo = 0;
	pin.sin6_addr = in6;

Tabla 3.2 Estructura sockaddr_in6, cliente-servidor

La otra forma de representar la variable **in6addr_any** es considerándola como una constante simbólica llamada **IN6ADDR_INIT**, definida en la librería *<netinet/in.h>*. Esta constante puede ser usada para inicializar una estructura **in6_addr**, como se muestra a continuación:

```
struct in6_addr anyaddr = IN6ADDR_ANY_INIT;
```

3.3.6 TRADUCCIÓN "NODENAME TO ADDRESS"

La función más común para la traducción de direcciones es la función **gethostname( )**, sin embargo, está función es inadecuada para muchas aplicaciones. Primeramente, porque no provee ninguna forma para que el emisor especifique el tipo de dirección deseada; y segundo, porque muchas implementaciones de esta función no son lo suficientemente seguras. La función llamada **gethostbyname2( )** definida en el RFC 2133, [19], también es inadecuada, primero porque

requiere una opción global (**RES_USE_INET6**) cuando la dirección IPv6 la requiere y segundo porque una bandera es necesaria para proveer al emisor con un control adicional sobre los tipos de direcciones requeridas.

Por lo tanto, la siguiente función definida en el RFC 2553, [18], es la que se utilizará en la aplicación, ya que soporta la traducción "Nodename to Address" una manera correcta y segura.

```
# include <sys/socket.h>
# include <netdb.h>

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

En el argumento ***name*** puede escribirse el nombre del *host*, o bien la dirección correspondiente del *host*.

En el argumento ***af*** se especificará la familia de direcciones en cualquiera de las siguientes dos formas **AF_INET** o **AF_INET6**.

El valor de ***error_num*** es regresado al emisor vía un apuntador, con el código de error adecuado en ***error_num*** para soportar un retorno seguro del código de error. El argumento ***error_num*** será fijado a uno de los siguientes valores:

- **HOST_NOT_FOUND**: No se ha encontrado el *host*.
- **NO_ADDRESS**: El servidor reconoce la petición y el nombre del *host*, pero ninguna dirección está disponible.
- **NO_RECOVERY**: Cuando ocurre una falla inesperada del servidor, la cual no se puede recuperar.
- **TRY_AGAIN**: Cuando ocurre un error temporal y posiblemente transitorio, tal como un fallo del servidor para responder.

El argumento ***flags***, especifica los tipos de direcciones que se están buscando y los tipos de direcciones que son regresados. Se hace notar, que si la bandera toma el valor **AI_DEFAULT**, muchas aplicaciones debieran poderse manejar. Esto es, muchas aplicaciones usan la siguiente llamada:

```
hptr = gethostbyname (name);
```

con

```
hptr = getipnodebyname(name, AF_INET6, AI_DEFAULT, &error_num);
```

Ahora bien, si en las aplicaciones se desea tener un control sobre el tipo de direcciones que se buscan o regresan, se pueden especificar otras combinaciones en el argumento ***flags***.

Una bandera en **0** implica una interpretación estricta del argumento ***af***:

- Si la bandera está en **0** y el argumento ***af*** es **AF_INET**, entonces el emisor solicita únicamente direcciones IPv4 (se hace mediante una consulta al registro **A**). Si la consulta es exitosa, entonces las direcciones IPv4 son retornadas, y el miembro ***h_lenght*** de la estructura **hostent** será 4, de otra forma la función retornará un apuntador nulo.

- Si la bandera está en **0** y el argumento ***af*** es ***AF_INET6***, entonces el emisor solicita únicamente direcciones IPv6, (se hace mediante una consulta al registro **AAAA**). Si la consulta es exitosa, entonces las direcciones Ipv6 son retornadas, y el miembro ***h_lenght*** de la estructura **hostent** será 16, de otra forma la función retornará un apuntador nulo.

Otras constantes definidas como lógicas **OR** dentro del argumento ***flag***, pueden modificar la respuesta de la función. Estas son:

- Si la bandera **AI_V4MAPPED** es especificada y el argumento ***af*** es **AF_INET6**, entonces, el emisor acepta las direcciones IPv4-*mapped* IPv6. Esto es, si no se ha

encontrado el registro **AAAA** entonces la consulta se hace en el registro **A,** y el registro encontrado (el que sea) es retornado como dirección IPv4-*mapped* IPv6. El miembro ***h-length*** será 16. La bandera **AI_V4MAPPED** es ignorada a menos que el argumento ***af*** sea igual **AF_INET6**.

- La bandera **AI_ALL** es utilizada en conjunto con la bandera **AI_V4MAPPED**; y es únicamente utilizada con la familia de direcciones IPv6. Cuando la bandera **AI_ALL** es lógicamente un **OR** con la bandera **AI_V4MAPPED**, entonces el emisor solicita todas las direcciones: IPv6 y IPv4-*mapped* IPv6. Una primera consulta se hace al registro **AAAA** y si ésta es exitosa, entonces las direcciones IPv6 son regresadas. Otra consulta es entonces hecha al registro **A** y las que sean encontradas se regresarán como direcciones IPv4-*mapped* IPv6 (el miembro ***h_leght*** será 16). Si ambas consultas fallan, entonces se retornará un apuntador nulo. La bandera **AI_ALL** es ignorada a menos que el argumento ***af*** sea igual **AF_INET6**.

- La bandera **AI_ADDRCONFIG** especifica que una consulta al registro **AAAA** podría ocurrir únicamente si el nodo tiene al menos una dirección fuente IPv6 configurada; y una consulta al registro **A** podría ocurrir únicamente si al menos una dirección fuente IPv4 está configurada.

Por ejemplo, si el nodo no tiene una dirección fuente IPv6 configurada, y el argumento ***af*** es igual a **AF_INET6,** y el nombre del nodo buscado tiene ambos registros (**AAAA**, **A**), entonces:

(a) Si sólo la bandera **AI_ADDRCONFIG** es especificada, la función retornará un apuntador nulo.

(b) Si las banderas **AI_ADDRCONFIG | AI_V4MAPPED** son especificadas, entonces los registros **A** serán retornados como direcciones IPv4-*mapped* IPv6.

Las banderas **AI_V4MAPPED** y **AI_ADDRCONFIG**, conforman una bandera especial con valor **AI_DEFAULT**, definida de la siguiente manera:

```
# define   AI_DEFAULT (AI_V4MAPPED | AI_ADDRCONFIG)
```

Se hace notar que la función **getipnodebyname( )** debe permitir que el argumento ***name***, sea el nombre de un nodo o una cadena literal de direcciones. Esto es, direcciones IPv4 o direcciones IPv6. Esto evita que las aplicaciones tengan que llamar a la función **inet_pton()** para manejar cadenas literales de direcciones.

Existen 4 casos, los cuales dependen de las cadenas de direcciones literales y del valor del argumento ***af***. Los dos casos más simples son:

1. Cuando el argumento ***name*** es una dirección IPv4, y el valor del argumento ***af*** es igual a **AF_INET**; o cuando el argumento ***name*** es una dirección IPv6 y el argumento ***af*** es igual a ***AF_INET6***. Entonces, los valores de los miembros retornados de la estructura **hostent** son:

   h_name: apunta a una copia del argumento de ***name***.
   h_aliases: es un apuntador nulo.
   h_addrtype: es una copia del argumento ***af***.
   h_length: puede tener el valor de 4 para **AF_INET** o 16 para **AF_INET6**.
   h_addr_list[0]: es un apuntador de 4-bytes o 16-bytes, contiene la dirección en binario.
   h_addr_list[1]: es un apuntador nulo.

2. Cuando el argumento ***name*** es una dirección IPv4 y el argumento ***af*** es igual a **AF_INET6**, y la bandera es igual a **AI_V4MAPPED**, entonces, una dirección IPv4-*mapped* IPv6 es retornada. Entonces, los valores de los miembros retornados de la estructura **hostent** son:

   h_name: apunta a direcciones IPv6 que contienen direcciones IPv4-mapped IPv6.
   h_aliases: es un apuntador nulo.
   h_addrtype: tiene el valor de **AF_INET6**.
   h_length: tiene el valor de 16.

h_addr_list[0]: es un apuntador de 16-bytes, contiene la dirección en binario.

h_addr_list[1]: es un apuntador nulo.

Si la bandera **AI_V4MAPPED** es asignada (con o sin la bandera **AI_ALL**), entonces se retornará la dirección IPv4-*mapped*, de otra manera se retornará un apuntador nulo.

Cuando el argumento ***name*** es una dirección IPv6 y el argumento ***af*** es igual a **AF_INET**, ocurre un error. Por lo tanto, la función retornará un apuntador nulo y el argumento de ***error_num*** será igual a **HOST_NOT_FOUND**.

Una vez definida la función **getipnodebyname** junto con los argumentos que esta función utiliza, la implementación del cliente queda de la siguiente forma:

CLIENTE

`if ( (hp = getipnodebyname(host,AF_INET6,AI_DEFAULT,&error_num)) == 0) {`
`perror("getipnodebyname");`
`exit(1);`
`}`

Tabla 3.3 Función getipnodebyname, cliente

3.3.7 FUNCIONES DE CONVERSIÓN DE DIRECCIONES

Existen dos funciones para convertir las direcciones IPv4 binarias a una forma decimal, estas son: **inet_addr( )** e **ine_ntoa( )**. De la misma manera, las aplicaciones IPv6 necesitan funciones similares. Las siguientes dos funciones convierten tanto direcciones IPv6 e IPv4:

```
# include <sys/socket.h>
# include <arpa/inet.h>

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

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

La función **inet_pton( )** convierte una dirección de la forma de presentación estándar a una forma binaria. Los argumentos de esta función son las siguientes:

El argumento ***af***, especifica la familia de las direcciones, éste puede soportar a la familia **AF_INET** o **AF_INET6**.

El argumento ***src***, apunta a la cadena que esta siendo transmitida.

El argumento ***dts***, apunta a un *buffer* dentro del cual la función almacena la dirección de una forma numérica, la dirección es retornada en *orden de octetos estándar de red (NBO, Network Byte Order)*.

La función **inet_pton( )** retorna un **1** si la conversión es exitosa, **0** si la entrada es una cadena IPv4 no válida o una cadena de dirección IPv6; o **-1** con el argumento ***errno*** en **EAFNOSUPPORT** si el argumento ***af*** es desconocido. Las aplicaciones de los emisores deben asegurarse que el *buffer* referido por el argumento ***dst***, sea lo suficientemente largo para contener las direcciones numéricas, (4 bytes para **AF_INET** o 16 bytes para **AF_INET6)**.

Si el argumento ***af*** es **AF_INET**, la función acepta una cadena de la forma estándar IPv4, es decir, de la forma:

ddd.ddd.ddd.ddd

Donde: **ddd** es uno de los tres dígitos del número decimal entre 0 y 255. Cabe mencionar que muchas aplicaciones que utilizan las funciones **inet_addr( )** e **inet_aton( )**, aceptan entradas no estándares, por ejemplo, números hexadecimales, números octales, etc. La función **inet_pton( )** no acepta estos formatos.

Si el argumento ***af*** es **AF_INET6**, entonces la función acepta una cadena en una de las formas estándares IPv6.

La función **inet_ntop( )** convierte la dirección numérica en una cadena de texto apropiada para la presentación de dicha dirección. Los argumentos de esta función son los siguientes:

El argumento ***af*** especifica la familia de la dirección, éste puede soportar la familia **AF_INET** o **AF_INET6**.

El argumento ***src*** apunta a un *buffer* que contiene la dirección IPv4 si el argumento ***af*** es **AF_INET**, o a la dirección IPv6 si el argumento ***af*** es **AF_INET6**. La dirección debe estar en *orden de octetos estándar de red.*

El argumento ***dst***, apunta a un *buffer* donde la función almacenará el resultado de la cadena de texto.

El argumento ***size***, especifica el tamaño del *buffer*. La aplicación debe especificar un argumento ***dst*** no nulo.

Para direcciones IPv6, el *buffer* debe estar como mínimo en 46 octetos; para las direcciones IPv4, el *buffer* debe estar como mínimo en 16 octetos. Con el objeto de permitir a las aplicaciones declarar fácilmente el *buffer* de tamaño adecuado para almacenar las direcciones IPv4 o IPv6 en forma de cadena, se definen las siguientes dos constantes:

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

    # define INET_ADDRSTRLEN    16
    # define INET6_ADDRSTRLEN   46
```

La función **inet_ntop( )** retorna un apuntador al *buffer* que contiene la cadena de texto si la conversión a sido exitosa, en caso contrario se retornará un apuntador nulo. En caso de falla, el argumento ***errno*** se fija **EAFNOSUPPORT** si el argumento ***af*** es invalido o **ENOSPC**, si el tamaño del *buffer* resultante es inadecuado.

Una vez definidas las funciones que convierten las direcciones binarias a una forma decimal, en la implementación se utiliza la función **inet_ntop( ),** quedando en el cliente de la siguiente forma:

CLIENTE
`inet_ntop(AF_INET6,(void *)&in6,abuf, sizeof(abuf));`

Tabla 3.4 Función inet_ntop, cliente

CONCLUSIONES

En el capítulo se describe la programación del socket en IPv6. Como se mencionó, un socket es un punto extremo de comunicación, que permite una comunicación bidireccional entre dos procesos que pueden ejecutarse en la misma máquina o no. También, permiten la utilización del protocolo IPv6 y TCP, utilizados en esta aplicación para la comunicación de procesos situados en distintos nodos.

Para trabajar con sockets IPv6, es necesario la instalación del *Release* IPv6 para Solaris 7 FCS, [26], y las utilerías utilerías (p.e. gcc y *Release* IPv6) necesarias para la compilación de los sockets. Cabe mencionar, que es necesario la instalación del Sistema Operativo Solaris 7 FCS y las para el Sistema Operativo especificado, de lo contrario la aplicación cliente-servidor implementada, no funcionará. Por lo tanto, en la implementación cliente-servidor realizada, se hace uso de la familia **PF_INET6**, las estructuras **in6_addr, sockaddr_in6**, **hostent *getipnodebyname,** la función **inet_ntop**, y el protocolo TCP; de esta forma se asegura el funcionamiento óptimo en la aplicación , garantizando la entrega de paquetes de forma ordenada y segura.

CAPÍTULO IV

4. PROTOCOLO DE COMUNICACIÓN ENTRE CLIENTE Y SERVIDOR

INTRODUCCIÓN

En el siguiente capítulo, estableceremos las características del modelo cliente-servidor. Cabe mencionar, que los servicios solicitados al servidor los envía el cliente a través de mensajes que contienen los datos necesarios para su ejecución. Cuando esto se realiza correctamente, el servidor responde enviando los resultados (acuses de recibo de los comandos) solicitados. También, se describe el protocolo de comunicación entre el cliente y servidor para asegurar el envío correcto de paquetes entre ellos.

4.1 COMUNICACIÓN ENTRE CLIENTE-SERVIDOR

El modelo cliente-servidor es aquel en el cual un proceso ofrece algún servicio a otros procesos que realizan el papel de clientes. Los servicios que provee el servidor pueden ser varios; en el caso de la presente tesis, el servidor podrá posicionar una video-cámara de acuerdo a los movimientos que el cliente solicite.

Para obtener la comunicación entre estos procesos, se creó el socket descrito en el Capítulo III. Los sockets son objetos que crean el acceso al cliente y al servidor. El servidor asigna al socket a una dirección y establece una cola para esperar solicitud de conexión al socket.

Una vez que la conexión se lleva a cabo, se establece un canal de comunicación *full-duplex* entre los dos procesos (cliente-servidor).

Ahora bien, lo primero que tenemos que definir es qué funciones queremos que realice nuestro servidor y por otro lado, qué requerimientos necesita nuestro cliente. La idea es que se desarrolle un programa servidor que espere peticiones de un cliente, para posicionar una video-cámara de acuerdo a los movimientos que éste solicite. La aplicación cliente-servidor se presenta, esquemáticamente en la siguiente figura (figura 4.1):

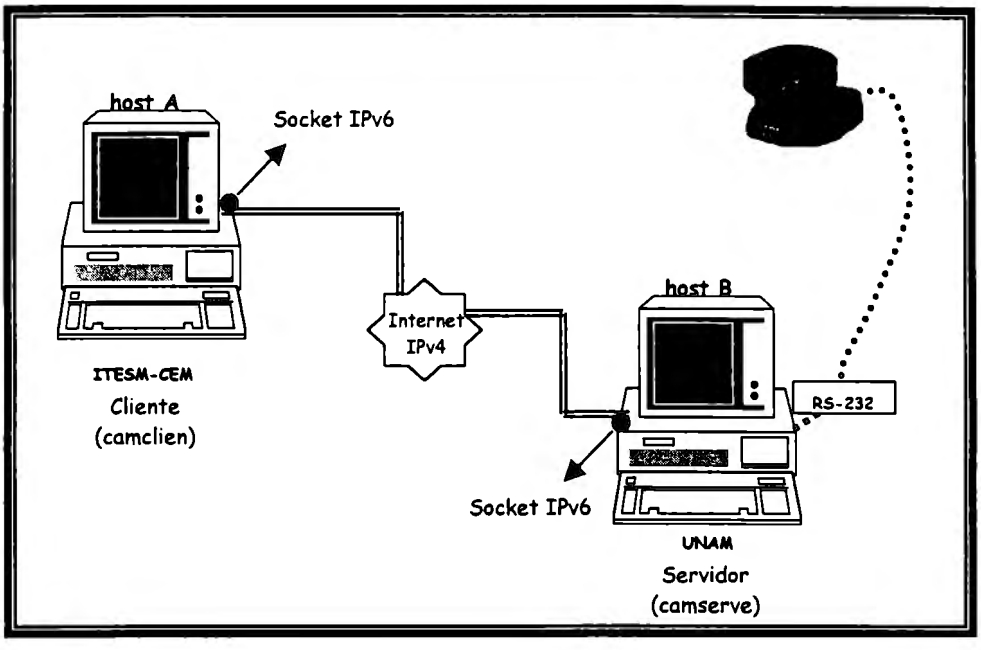

Figura 4.1 Esquema cliente-servidor

Podemos observar que el *host* A y el *host* B se encuentran comunicados mediante un socket. Por lo tanto, el primer paso es establecer la conexión entre el cliente y el servidor; esto se realizará ejecutando al servidor que se denominó "camserve" para que éste este esperando la conexión del cliente, el cual se nombró "camclien". Una vez establecida la conexión se procede a realizar las tareas.

Las tareas que el servidor realiza son:

1. Espera la conexión de algún cliente.

2. Si la conexión a sido realizada con éxito, el servidor envía un reconocimiento positivo al cliente para que éste proceda a hacer sus peticiones.
3. Espera los comandos (movimientos) del cliente.
4. Abre el puerto RS-232, [27].
5. Si el comando se recibe exitosamente, el servidor envía un reconocimiento positivo al cliente y sigue en espera de más comandos.
6. Se escribe el comando al puerto RS-232, para efectuar el movimiento de la cámara.
7. Si el comando no se ha recibido durante cierto tiempo, el servidor cierra la conexión.
8. Si el comando contiene un error, el servidor envía un mensaje al cliente, de retransmisión de comando y regresa al paso 3.
9. Espera cierre de conexión.
10. Cierra el puerto RS-232.

Las tareas que el cliente realiza son:

1. Realiza la conexión con el servidor.
2. Espera reconocimiento (ack) de conexión.
3. En caso de que el reconocimiento (ack) de conexión no llegue, el cliente solicita de nuevo la conexión.
4. Solicita la entrada de comandos al usuario (arriba, abajo, derecha, izquierda, home).
5. Envía el comando que el usuario a introducido, al servidor.
6. Espera reconocimiento de comando (ack, error, timeout).
7. En caso de que el reconocimiento (ack, error, timeout) no llegue, el cliente retransmite el comando introducido.
8. Si el comando introducido no existe, se solicita al usuario una entrada válida.
9. Envía el nuevo comando o retransmite el comando al servidor, si éste lo solicita.
10. Cierra la conexión establecida con el servidor.

Para entender mejor el proceso que realiza el cliente, en la figura 4.2 se detallan los estados y eventos que éste realiza.

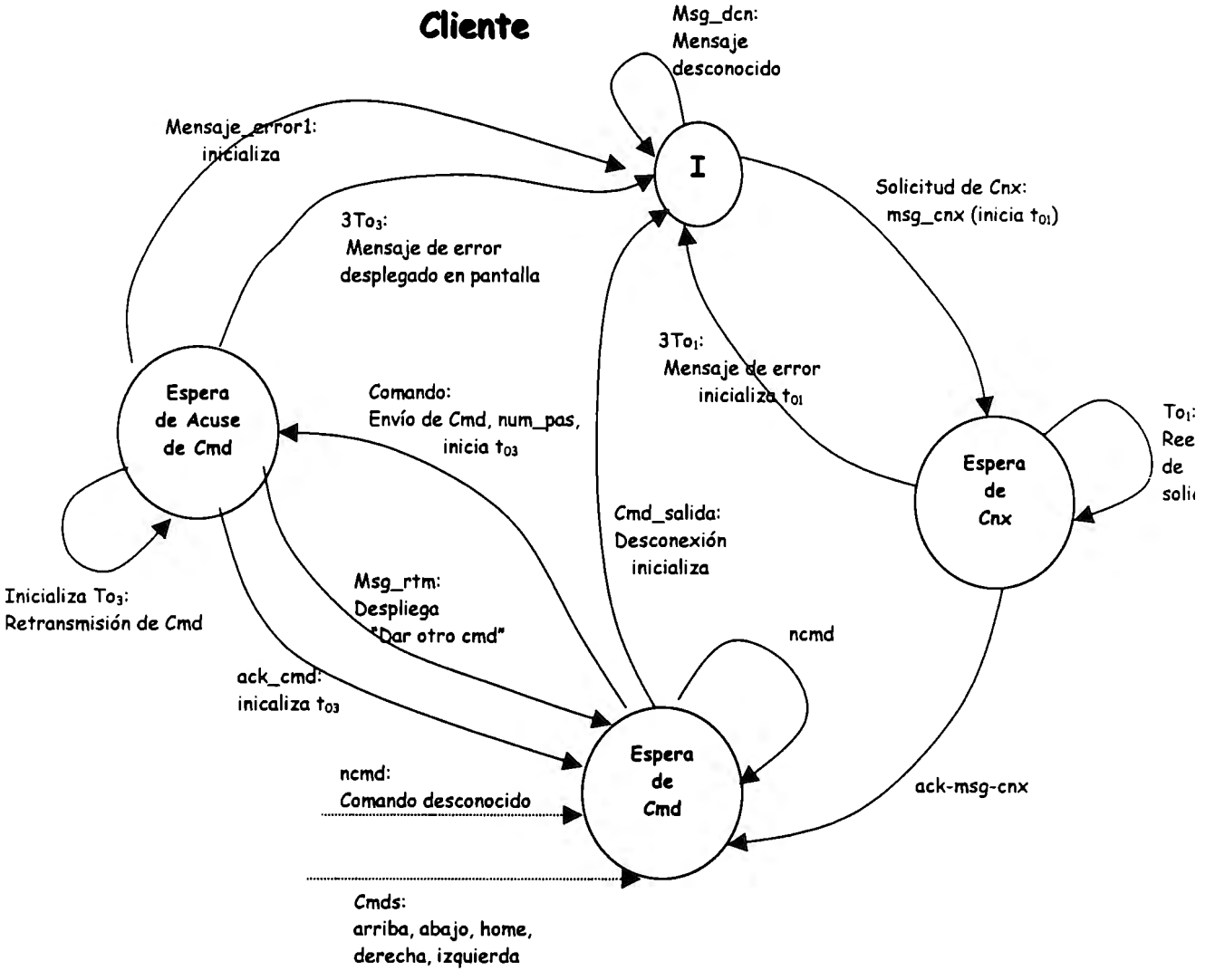

Figura 4.2 Proceso del cliente

Se observa que el proceso del cliente cuenta con los siguientes 4 estados:

- ***I (Estado Inicial)***: si al estado le llega un mensaje desconocido permanece en el mismo estado, si envía la solicitud de conexión pasa al estado ***Espera de Cnx***.

- ***Espera de Cnx (Espera de Conexión)***: este estado recibe la solicitud de conexión y envía el reconocimiento de este mensaje para pasar al estado ***Espera de Cmd***, si no recibe el reconocimiento en cierto tiempo, éste solicita un nuevo mensaje de conexión. Si en tres intentos (T_{01}) de espera no llega el mensaje, retornará al estado ***I***.

- ***Espera de Cmd (Espera de Comando)***: este estado recibe la entrada de comandos de parte del usuario (las flechas no indican que este estado sea un estado inicial). Si se ha introducido un comando inválido, sigue en el mismo estado, en caso contrario envía el comando y el número de pasos y pasa al estado de ***Espera de Acuse de Cmd***. Si se a introducido el comando de salir de la conexión, se retornará al estado ***I***.

- ***Espera de Acuse de Cmd (Espera de Acuse de Comando)***: este estado recibe el comando y espera el reconocimiento de este comando para pasar al estado ***Espera de Cmd***. Se inicializa el temporizador T_{03} y si en cierto tiempo no se recibe dicho acuse, retransmite el comando y pasa al estado ***Espera de Cmd***. Si en tres intentos no se recibe el acuse o se expira el temporizador se retornará al estado ***I***.

En el caso del servidor, éste cuenta sólo con 2 estados (figura 4.3).

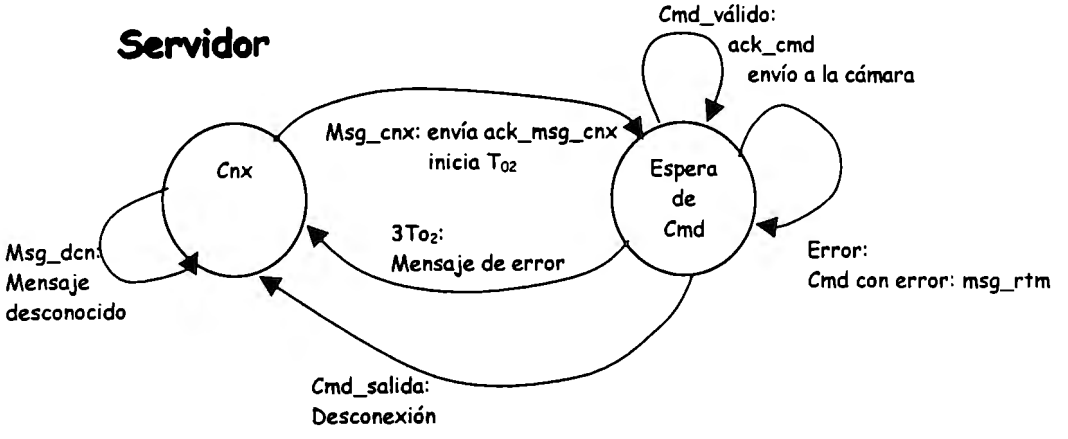

Figura 4.3 Proceso del servidor

La función de estos estados se describen a continuación:

- ***Cnx (Conexión)***: si el estado recibe un mensaje desconocido, permanece en el mismo estado. Si ha llegado la solicitud de conexión envía el reconocimiento de ésta para pasar al estado ***Espera de Cmd***.

- ***Espera de Cmd (Espera de Comando)***: este estado realiza varias acciones. Primero, si en cierto tiempo no recibe los comandos o éstos contienen errores, se envía mensaje de retransmisión de comandos y permanece en espera de los comandos. Segundo, si se ha recibido un comando válido envía reconocimiento, escribe en el puerto de la cámara el comando y sigue en espera de más comandos. Tercero, si se ha recibido el comando de salida, se retornará al estado ***Espera de Cnx***. Finalmente, si no se ha recibido un mensaje del cliente en tres intentos, se retornará al estado ***Espera de Cnx***.

Ahora bien, los mensajes que se generan en la comunicación cliente-servidor se muestran a continuación en la figura 4.4.

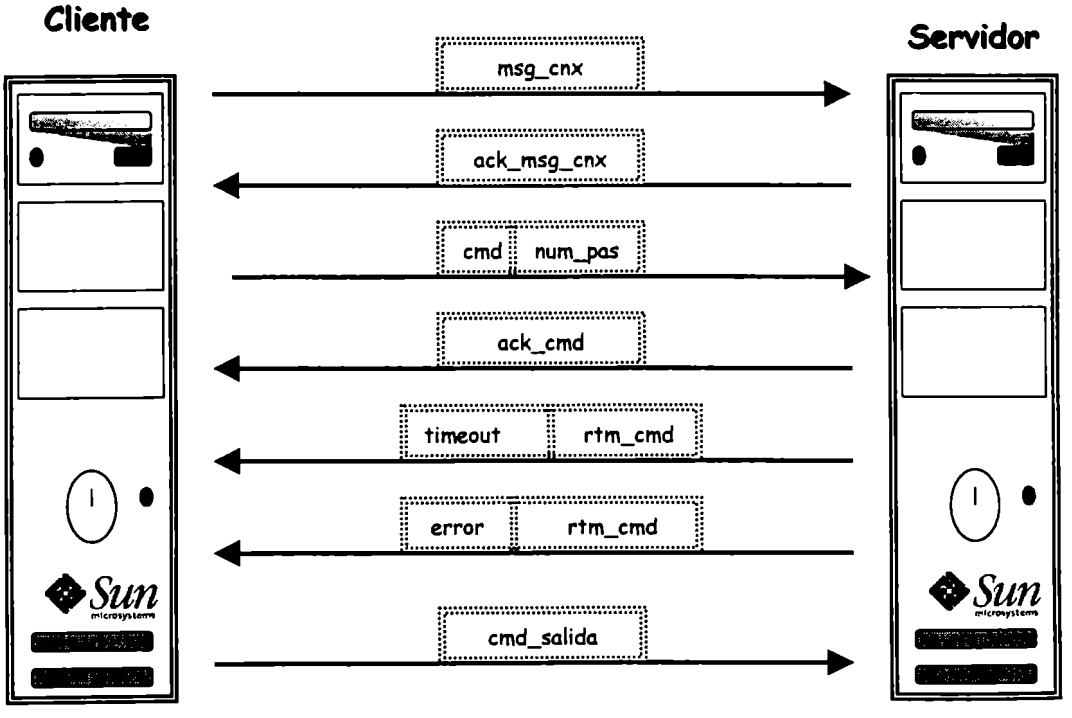

Figura 4.4 Mensajes transmitidos entre cliente-servidor

CONCLUSIONES

Implementando este protocolo (descrito en el Capítulo) en la aplicación cliente-servidor junto con el protocolo TCP, podemos garantizar un flujo *full-dúplex* integral, confiable y controlado. Tras solicitar al TCP que establezca una conexión, los programas de aplicación pueden usarla para transmitir y recibir datos garantizando la entrega de datos en orden y sin duplicación. Todos los mensajes que envía el cliente al servidor, viajan a través de la red, incluidos mensajes con datos, acuses de recibo y mensajes para establecer y terminar la conexión.

Para hacer un servicio completamente confiable, el protocolo retransmite los mensajes perdidos haciendo uso de un tiempo de expiración. Este tiempo de expiración, se calculó de acuerdo al tiempo que toma un comando en su ejecución, ya que depende directamente del número de "pasos". Por ello, el cliente calcula estos temporizadores de forma dinámica dependiendo del número de "pasos" que se hayan solicitado. Por otro lado, como se concluyó en el Capítulo II, IPv6 es un protocolo que puede tener la capacidad de tratar el tráfico sensible a retardos de forma más eficiente que IPv4. Podemos concluir entonces, que las rutinas cliente-servidor en IPv6 son de gran utilidad ya que permiten la comunicación entre máquinas haciendo uso de la red; en este caso, nos permite posicionar una video-cámara a distancia. Sin embargo, las pruebas contundentes de la mejora del desempeño, no pueden ser calculadas actualmente ya que toda comunicación viaja encapsulada por IPv4.

CAPÍTULO V

5. PRUEBAS OPERATIVAS GENERALES

INTRODUCCIÓN

En este capítulo se presenta una de las pruebas operativas generales realizadas con la aplicación cliente-servidor haciendo uso del protocolo IPv6. El objetivo es posicionar una video-cámara en sus movimientos básicos: arriba, abajo, derecha, izquierda y home.

5.1 SECUENCIA DE LA PRUEBA OPERATIVA

El prototipo de red se realizó a través de la conexión de las siguientes máquinas:

CLIENTE **(acorral)**	**SERVIDOR** **(dogbert)**
Workstation Type: SUNW, Ultra-5. IPv6 Address: 3ffe:1cff:0:fd::3 Domain: cem.itesm.mx Operating System: Solaris 7 FCS, SunOS 5.7	Workstation Type: SUNW, SparcStation-4. IPv6 Address: 3ffe:1cff:0:fd::1 Domain: dgsca.unam.mx Operating System: Solaris 7 FCS, SunOS 5.7

En la máquina servidor está conectada la video-cámara Sony EVI-D30, la cual tiene las siguientes características, [28]:

- High Speed, Wide Range Pan/tilter
- X12 Optical Zoom, High Speed Auto-Focus Lens
- 6 Position Preset
- Auto Tracking/Motion Detector
- RS232C Serial Control
- IR remote Commander
- Time, Date Generator

Una vez establecida la conexión cliente-servidor (acorral-dogbert), se mandaron los siguientes comandos de control para posicionar la video-cámara a partir de la posición home, con la siguiente secuencia:

- 10 movimientos a la Izquierda
- 6 movimientos Abajo
- 13 movimientos a la Derecha
- 5 movimientos Arriba

Los resultados obtenidos de esta prueba se presentan a continuación mostrando los mensajes desplegados en pantalla, por el cliente y el servidor. Cabe mencionar que sólo se muestra el resultado de una prueba operativa, ya que todas las pruebas operativas mostraron resultados similares.

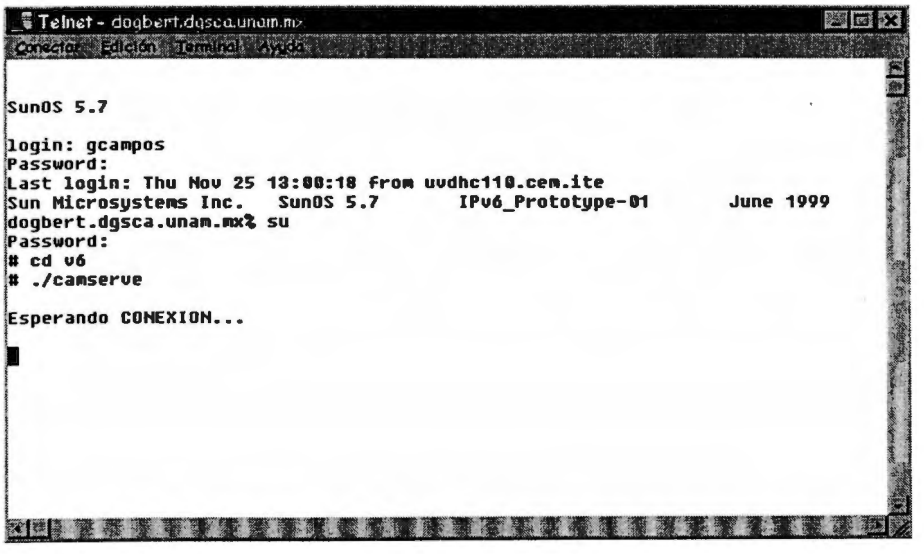

Figura 5.1 Servidor (dogbert) esperando conexión del cliente (acorral)

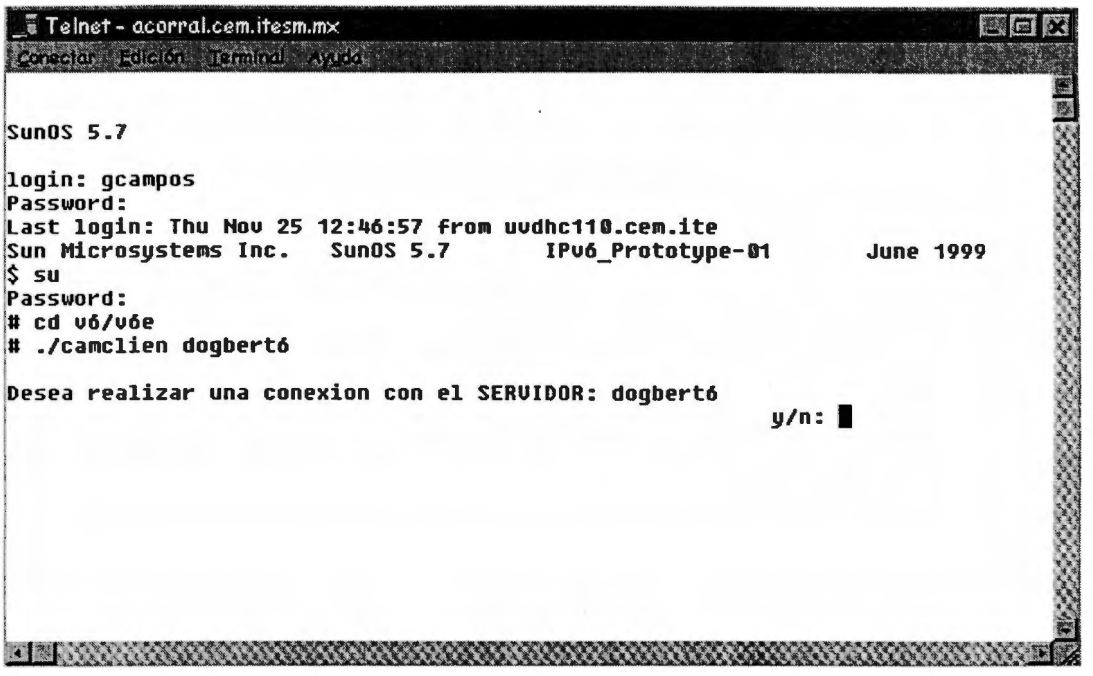

Figura 5.2 Cliente (acorral) realizando conexión al servidor (dogbert)

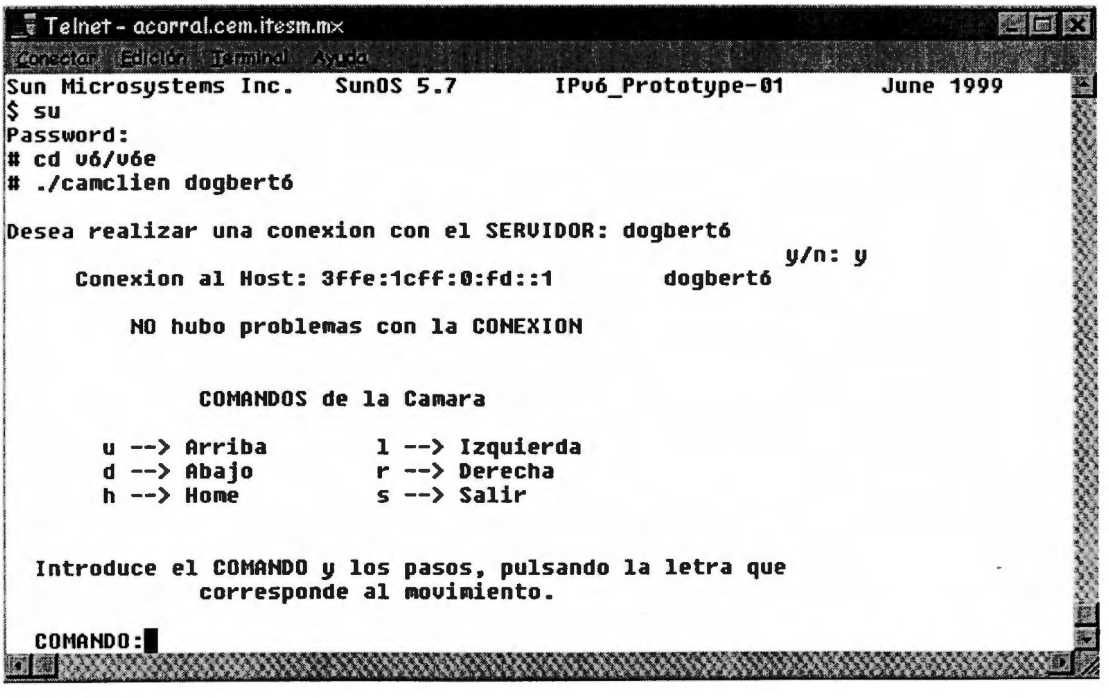

Figura 5.3 Conexión realizada con el servidor (dogbert)

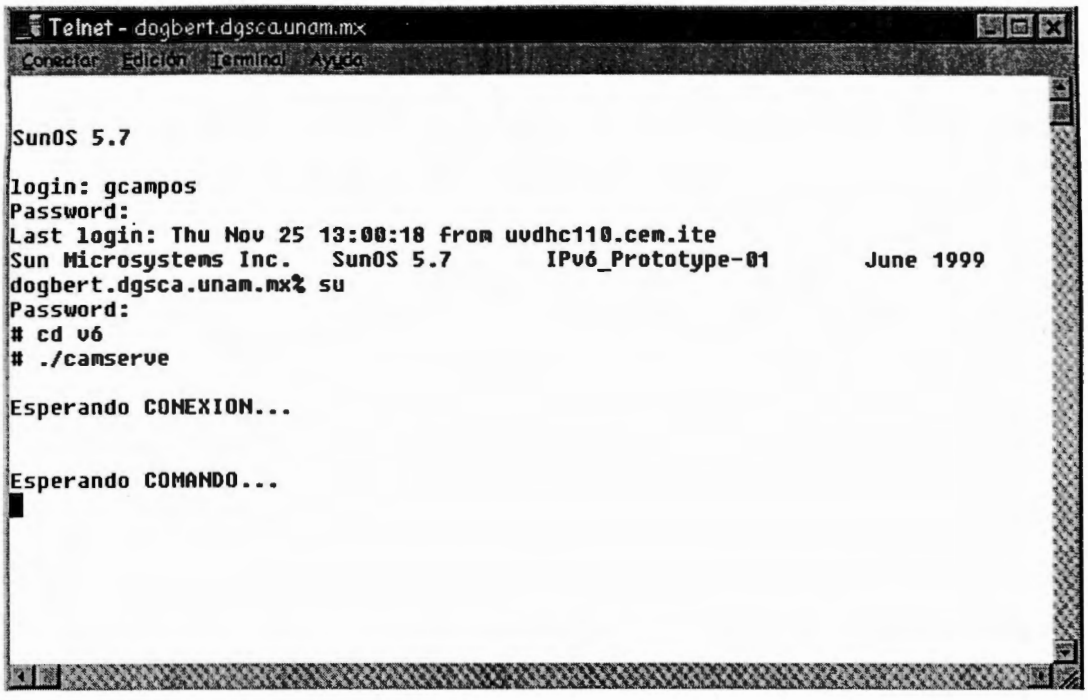

Figura 5.4 Servidor (dogbert) en espera de comandos

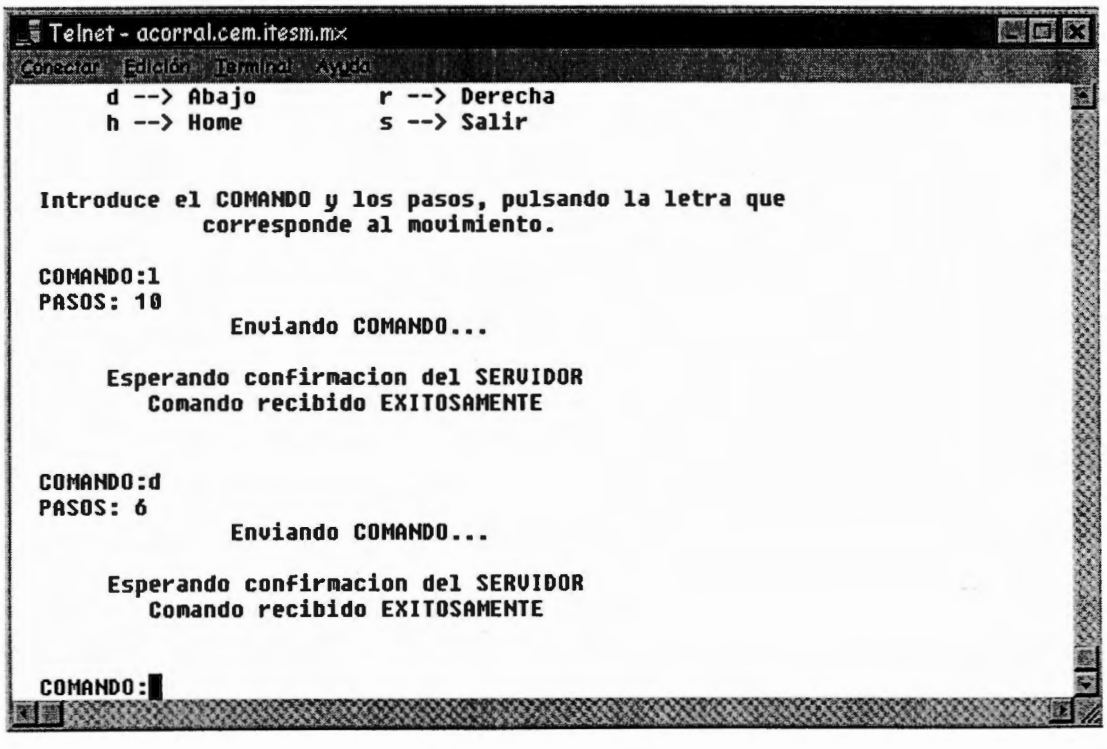

Figura 5.5 Envío de comandos Izquierda y Abajo

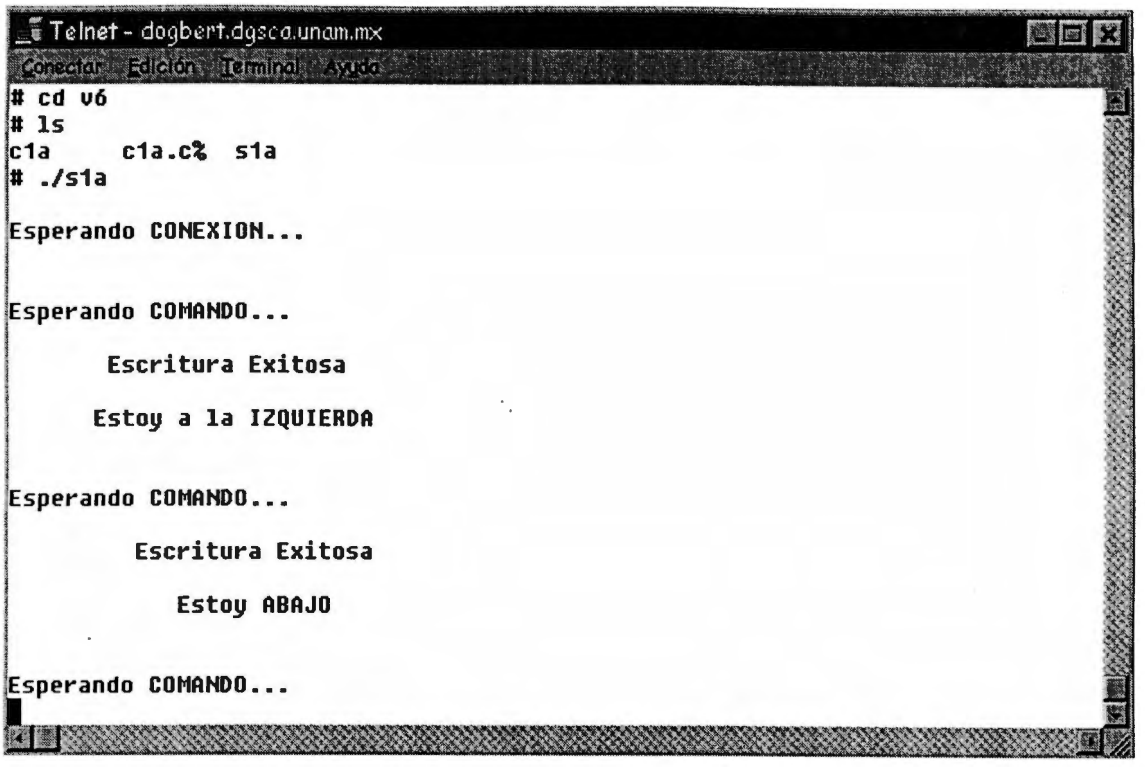

Figura 5.6 Recepción de comandos Izquierda y Abajo

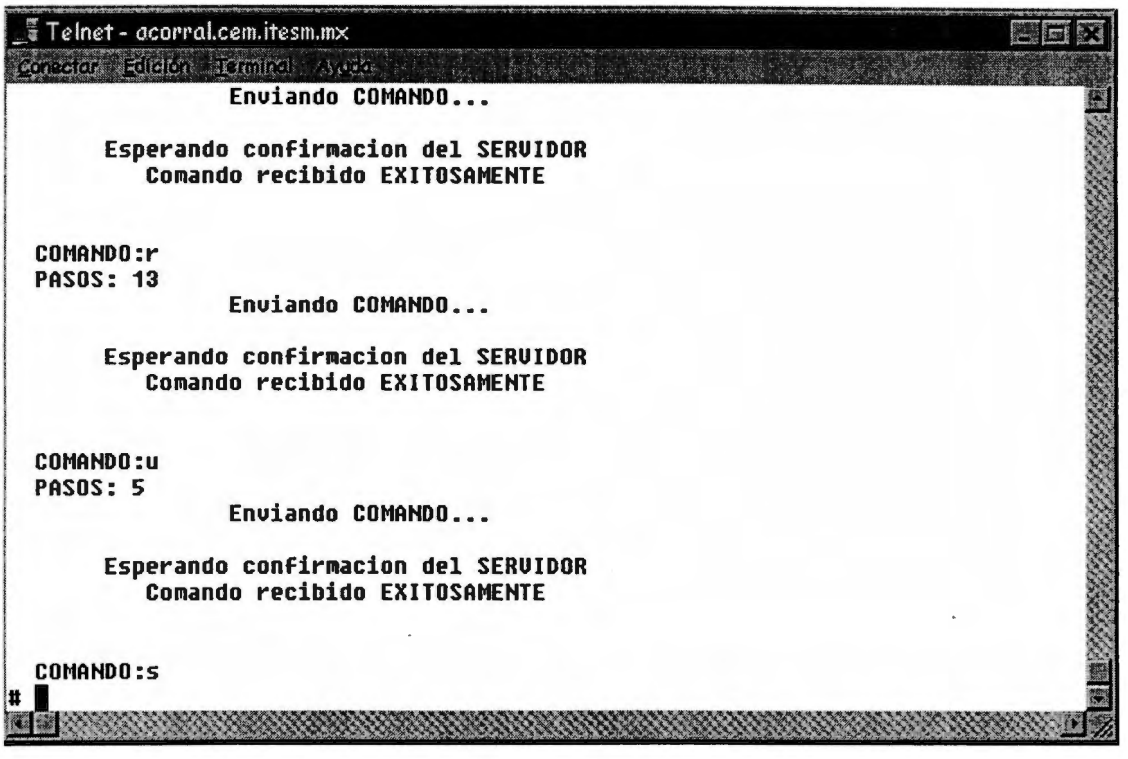

Figura 5.7 Envío de comandos Derecha y Arriba, y cierre de conexión

Figura 5.8 Recepción de comandos Derecha y Arriba, y cierre de conexión

CONCLUSIONES

Las pruebas operativas desarrolladas mostraron (en el Capítulo sólo se muestran los resultados de una de las pruebas operativas realizadas, dado que todas presentaron resultados satisfactorios), que la programación del socket en IPv6 funcionó exitosamente, ya que se cumplió con el objetivo de posicionar la video-cámara en sus movimientos básicos (arriba, abajo, derecha, izquierda y home). Además, se ha demostrado que es factible manejar el direccionamiento de 128 bits y que el encabezado de este nuevo protocolo es más fácil de utilizar haciendo uso de la programación "Applications Program Interfece" API en IPv6. También se comprobó que los *hosts* y opcionalmente los ruteadores pueden transportar el tráfico de IPv6 a través de topologías de enruteamiento de IPv4 por encapsulamiento, permitiendo a los nodos actualizados interactuar con los nodos de IPv4.

Para comprobar y garantizar las demás características que el protocolo IPv6 ofrece, es necesario permitir a las aplicaciones especificar una "Calidad de Servicio" (QoS) de red en cuestiones tales como la velocidad de transmisión, el retardo limitado y los límites de variación del mismo, el rendimiento y la planificación. Esto se podrá realizar a medida que los ruteadores se configuren con IPv6 para que reconozcan la prioridad que cada paquete contenga. Además, de una investigación de cómo transmitir video en IPv6, para garantizar la entrega de éste en tiempo real.

CAPÍTULO VI

6. CONCLUSIONES

Durante el desarrollo de este proyecto, se pudo observar que el protocolo IPv6 conserva muchas de las características de diseño que hacen a IPv4 tan exitoso. Al igual que IPv4, IPv6 opera sin conexiones, es decir, cada datagrama tiene una dirección destino y se enruta de forma independiente; también permite al emisor seleccionar el tamaño del datagrama y el máximo número de saltos que un datagrama puede realizar antes de ser eliminado. Además, el IPv6 utiliza direcciones más largas y añade algunas características nuevas, tales como: el formato de encabezado, encabezado de extensión, manejo de audio y video, autentificación y privacidad. Algo importante que remarcar, es que IPv6 revisa completamente el formato de los datagramas, remplazando el campo de opción de longitud variable del IPv4 por una serie de encabezados de formato fijo.

Una vez analizados los formatos de los encabezados de IPv6, la forma de direccionamiento, enruteamiento y fragmentación de los paquetes, se pueden aprovechar las numerosas características que hacen tan atractivo al protocolo, como el soporte de comunicaciones en tiempo real, la autoconfiguración de sistemas y seguridad. Para este proyecto en particular, se realizó un mecanismo de control eficiente a través de la red para posicionar una video-cámara. Se considera eficiente ya que gracias a la introducción de flujos etiquetados (con prioridades) y los servicios de restricciones de tiempo real, hacen que este protocolo pueda tener la capacidad de tratar el tráfico sensible a retardos más eficientemente que IPv4.

Para el desarrollo del mecanismo de control, fue necesario la programación del socket en IPv6, en el cual se hizo uso de la familia **PF_INET6**, las estructuras **in6_addr, sockaddr_in6**, **hostent *getipnodebyname,** la función **inet_ntop**, el protocolo TCP y el establecimiento de un protocolo entre el cliente y servidor, para asegurar el funcionamiento óptimo en la aplicación

cliente-servidor, garantizando una transmisión *full-dúplex* y la entrega de los comandos de forma ordenada y segura.

Para trabajar con sockets IPv6, fue necesario la instalación del *Release* IPv6 para Solaris 7 FCS, y las utilerías (p.e. gcc y *Release* IPv6) necesarias para la compilación de los sockets. Cabe mencionar, que es necesario la instalación del Sistema Operativo Solaris 7 FCS y las utilerías para el Sistema Operativo especificado, de lo contrario la aplicación cliente-servidor implementada, **no funcionará**.

Podemos concluir entonces, que el socket creado en IPv6 funcionó de manera correcta, ya que nos permitió de manera eficiente la comunicación entre el cliente y el servidor para posicionar exitosamente una video-cámara haciendo uso de la red convencional. Además, se han demostrado algunas características que el protocolo IPv6 ofrece, tales como: manejar el direccionamiento de 128 bits, y la compatibilidad con el protocolo IPv4. Ya que los *hosts* y opcionalmente los *routers* pueden transportar el tráfico de IPv6 a través de topologías de enruteamiento de IPv4 por encapsulamiento, permitiendo a los nodos actualizados interactuar con los nodos de IPv4. Debido al éxito en el proyecto (ya que se cumplieron los objetivos propuestos), se puede concluir, tentativamente, que IPv6 es "mejor" que IPv4 y que muy probablemente IPv6 pronto pase a ser el nuevo protocolo estándar que opere en Internet.

6.1 TRABAJO A FUTURO

La minimización de algunos problemas del control de la cámara se pueden lograr a través de implementar un control en lazo cerrado. Aunque hay que remarcar que si hay una pérdida de conexión cliente-servidor debido a problemas de red, no hay técnica que resuelva el problema de control hasta que la red se reestablezca.

En función del trabajo a futuro indicado anteriormente, y aprovechando las características que IPv6 proporciona, la señal de "feedback" que podría utilizarse es el de la imagen que la video-cámara captase, lo cual requerirá investigar cómo manejar la transmisión de imagen en tiempo real con IPv6.

BIBLIOGRAFÍA

[1] Christian Huitema, *IPv6 the New Internet Protocol*, Prentice Hall, 1996.

[2] Comer E. Douglas, *TCP/IP Vol. I*, Prentice Hall, 3° Edición, p.497-518.

[3] Comer E. Douglas, *Redes de Computadoras, Internet e Interredes*, Prentice Hall, 1° Edición, p. 239-249.

[4] IPv6
IPv6 Networking for the 21st Century
http://www.ipv6.org

6bone te testben for deployment of IPv6
http://www.6bone.net

[5] Especificaciones IPv6
IPv6 Related Specifications
http://www.ipv6.org/specs.html

Internet Protocol Version 6
http://telecom.noc.udg.mx/~kenya/cen-contenido.html

Exposición IPng
http://a01-unix.gsyc.inf.uc3m.es/~baudoux/présentacion.html

Protocolo IPv4 vs Protocolo IPv6
http://www.disc.ua.es/asignaturas/rc/trabajos/ip/indice.html

Protocolo Internet Version 6 (IPv6)
http://www.ldc.usb.ve/~93-25298/Tema15/index.html

IPv6 Specifications and RFC's
http://gate.ticl.co.uk/ipv6rkCD/docs/fdocidx.htm

[6] Steve King, Ruth Fax, Dimitry Haskin, Wenken Ling, Tom Meehan, Robert Fink, Charles E. Perkins, *The Case for IPv6*, INTERNET DRAFT, January 1999.
http://www.ietf.org/internet-drafts/draft-ietf-iab-case-for-ipv6-04.txt

[7] S. Bradner, A. Mankin, *The Recommendation for the IP Next Generation Protocol*, RFC1752, January 1995.
http://www.iol.unh.edu/training/general/rfc1752.txt

[8] S. Deering, R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, RFC2460, December 1998.
ftp://venera.isi.edu/in-notes/rfc2460.txt

[9] C. Partridge, *Using the Flow Label Field in IPv6*, RFC1809, June 1995
http://www.cis.ohio-state.edu/htbin/rfc/rfc1809.html

[10] A. Conta, S. Deering, *Generic Packet Tunneling in IPv6 Specification*, RFC2473, December 1998.
ftp://venera.isi.edu/in-notes/rfc2473.txt

[11] D. Estrin, T. Li, Y. Rekhter, K. Varadhan, D. Zappala, *Source Demand Routing Protocol (SDRP)*, RFC1940, May 1996.
http://www.cis.ohio-state.edu/htbin/rfc/rfc1940.html

[12] J. McCann, S. Deering, J. Mogul, *Path MTU Discovery for IP version 6*, RFC1981, August 1996.
http://www2.hunter.com/docs/rfc/rfc1981.html

[13] S. Kent, R. Atkinson, *IP Authentication Header*, RFC2402, November 1998.
http://andrew2.andrew.cmu.edu/rfc/rfc2402.html

[14] S. Kent, R. Atkinson, *IP Encapsulating Security Payload (ESP)*, RFC2406, November 1998.
http://www.sunsite.auc.dk/RFC/rfc/rfc2406.html

[15] S. Kent, R. Atkinson, *Security Architecture for the Internet Protocol*, RFC2401, November 1998
ftp://ftp.isi.edu/in-notes/rfc2401.txt

[16] R. Hinden, S. Deering, *IP Version 6 Addressing Architecture*, RFC2373, July 1998.
ftp://ftp.isi.edu/in-notes/rfc2373.txt

[17] R. Elz, *A Compact Representation of IPv6 Addresses*, RFC1924, April 1996.
http://andrew2.andrew.cmu.edu/rfc/rfc1924.html

[18] R. Gilligan, S. Thomson, J. Bound, W. Stevens, *Basic Socket Interface Extensions for IPv6*, RFC2553, April 1997.
ftp://ftp.isi.edu/in-notes/rfc2553.txt

[19] R. Gilligan, S. Thomson, J. Bound, W. Stevens, *Basic Socket Interface Extensions for IPv6*, RFC2133, April 1997.
http://rfc.fh-koeln.de/rfc/html/rfc2133.html

[20] Technical Standard: Networking Services (XNS) Issue 5.2 Draft 3.0
http://www.opengroup.org/orc/DOCS/XNS/webcom.htm

[21] A. Conta, S. Deering, *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, RFC2463, December 1998.
ftp://venera.isi.edu/in-notes/rfc2463.txt

[22] W. Stevens, M. Thomas, *Advanced Sockets API for IPv6*, RFC2292, February 1998.
ftp://venera.isi.edu/in-notes/rfc2292.txt

[23] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, RFC1889, January 1996.
http://info.internet.isi.edu:80/in-notes/rfc/files/rfc1889.txt

[24] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, *Resource ReSerVation Protocol (RSVP)*, RFC2205, September 1997.
http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2205.txt

[25] Internet 2
Corporación Universitaria para el Desarrollo de Internet (CUDI)
http://www.internet2.edu.mx/antece.htm

Internet2 o la Próxima Generación de Internet (Primera Parte)
http://web.ati.es/PUBLICACIONES/novatica/1997/127/intdos.html

Internet2 o la Próxima Generación de Internet (Segunda Parte)
http://web.ati.es/PUBLICACIONES/novatica/1997/128/intdos-2.html

Internet 2
http://www.internet2.edu/html/about-i2.html

Internet2 UNAM
http://www.internet2.unam.mx/

[26] SUN
http://playground.sun.com/pub/solaris2-ipv6/html/solaris2-ipv6.html

Release Notes
http://www.ipv6.org/solaris-quick.html
http://playground.sun.com/pub/solaris2-ipv6/html/BOOK.htm

[27] Serial Programing Guide for POSIX Compliant Operating Systems
http://dns-gate.easysw.com/~mike/serial/

[28] Sony EVI-D30 Teleconferencing Cameras
http://www.avtg.com/SonyEVI-D30-specs.htm

[29] InterNIC
http://www.internic.net/

[30] Internet 2/VbNS
http://web.missouri.edu/iats/rsc/talks/morenet_03-19-99/sld001.htm

[31] Internet 2/ NGI
http://www.nanog.org/mtg-9901/ppt/almes/almes/sld001.htm