CAPITULO I. INTRODUCCIÓN

En este capítulo se explican los conceptos básicos de las comunicaciones y la arquitectura TCP/IP. Se tocan algunas funciones básicas para la programación de aplicaciones con sockets y se trata una visión breve de las aplicaciones cliente/servidor. También se tocan los aspectos relevantes del direccionamiento IP y su implementación.

Capítulo I

Introducción

I.1. Comunicaciones TCP/IP

El término TCP/IP por lo general significa todo y cualquier cosa relacionada a los protocolos específicos de TCP (Transmission Control Protocol) e IP (Internet Protocol). Esto puede incluir otros protocolos, aplicaciones e incluso el medio o interfaz de red. Un ejemplo de estos protocolos son: UDP (User Datagram Protocol), ARP (Address Resolution Protocol) e ICMP (Internet Control Message Protocol). También una muestra de estas aplicaciones son: TELNET, FTP (File Transfer Protocol) y DNS (Domain Name Service). Un término más preciso para TCP/IP es llamarla arquitectura TCP/IP, un ejemplo de red que implementa la arquitectura TCP/IP es la llamada **Internet**.

I.1.1. Estructura básica de la arquitectura TCP/IP

Para entender la arquitectura TCP/IP primero se debe comprender la siguiente estructura lógica, figura I.1.1. Esta es la estructura lógica de los protocolos en capas dentro de una terminal en Internet. Cada terminal que se comunica utilizando la arquitectura TCP/IP tiene la estructura lógica que se muestra en la figura I.1.1. Esta estructura lógica determina el comportamiento de las terminales en Internet.

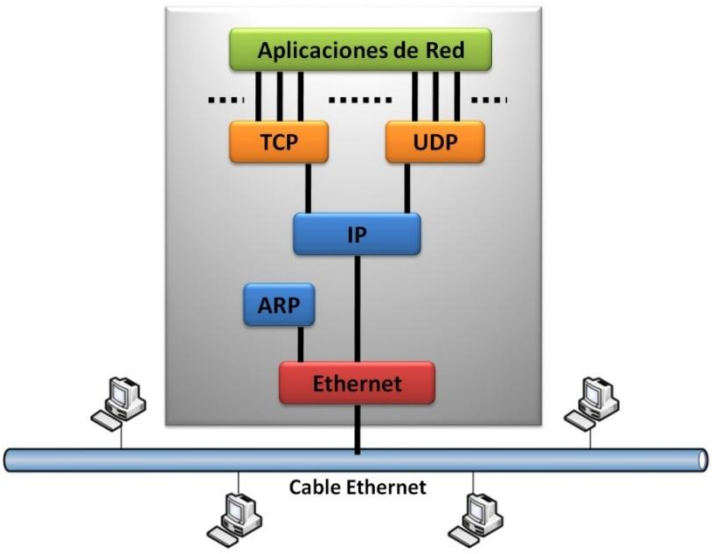


Figura I.1.1. Nodo básico de red TCP/IP.

Las capas representan el procesamiento de los datos a medida que pasan a través de la terminal, y las líneas que conectan las capas muestran el camino de los datos. Como se observa en la figura I.1.1, las terminales se conectan a un medio de transmisión que en éste caso es un cable de tipo Ethernet. En la interfaz de red Ethernet está presente la dirección física o dirección MAC y en la capa de red está la dirección lógica o dirección IP.

I.1.2. Terminología

El intercambio de datos entre capas del modelo OSI (Open System Interconnection) es desarrollado por las unidades de datos de protocolo, también llamadas PDU (Protocol Data Unit). El PDU de datos contienen los datos del usuario y el PDU de control se utiliza para administrar el comportamiento del protocolo. El PDU de los protocolos en Internet depende de la capa en que se ubique. A continuación se presentan los términos básicos (ver figura I.1.2):

Segmento: Un segmento es la unidad de transmisión de extremo a extremo en el protocolo TCP. Un segmento consta de un encabezado TCP seguido de datos de la capa de aplicación. Un segmento se transmite encapsulado dentro de un datagrama IP (paquete) a través de Internet.

Mensaje: Un mensaje es la unidad de transmisión en un protocolo de capa de transporte. Entonces se puede decir que un segmento TCP es un mensaje. Un mensaje se compone de un encabezado de un protocolo de capa de transporte seguido por los datos del protocolo de la capa de aplicación. Para ser transmitido de extremo a extremo a través de la Internet, un mensaje debe ser encapsulado dentro de un datagrama IP.

Datagrama IP: Un datagrama IP es la unidad de transmisión de extremo a extremo en el protocolo IP. Un datagrama IP consiste de un encabezado IP seguido por los datos de capa de transporte; es decir, un encabezado IP seguido por un mensaje.

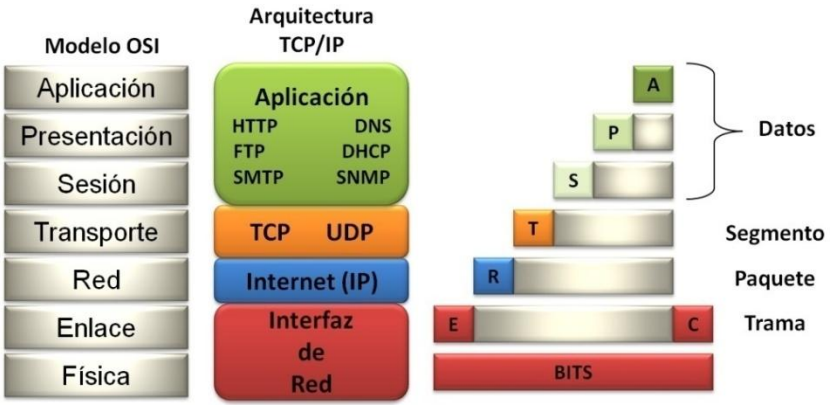


Figura I.1.2. Unidad de datos de protocolo en la arquitectura TCP/IP.

Paquete: Un paquete es la unidad de datos que se pasa a través de la interfaz entre la capa de red (Internet) y la capa de enlace. Un paquete está estructurado por un encabezado IP seguido de los datos. Un paquete puede ser un datagrama IP completo o un fragmento de un datagrama IP.

Trama: Una trama es la unidad de transmisión en un protocolo de capa de enlace, y consta de un encabezado de capa de enlace seguido por un paquete.

I.1.3. Flujo de datos

A continuación se explica cómo fluyen los datos hacia abajo a través de la pila de protocolo que se muestra en la Figura I.1.1. Para una aplicación que utiliza el protocolo TCP, los datos pasan desde la aplicación hasta el protocolo TCP. Para las aplicaciones que utilizan UDP, los datos pasan desde la aplicación hasta el protocolo UDP. FTP es una aplicación típica que utiliza TCP y su pila de protocolos en este ejemplo es FTP/TCP/IP/Ethernet. SNMP (Simple Network Management Protocol) es una aplicación que utiliza UDP y su pila de protocolos en este ejemplo es SNMP/UDP/IP/Ethernet.

El protocolo TCP, el protocolo UDP y el driver de Ethernet son multiplexores de `n a 1`, pues conmutan muchas entradas a una salida y por lo tanto también son demultiplexores de `1 a n`, pues conmutan una entrada a muchas salidas de acuerdo con el tipo de campo en el encabezado del protocolo, ver figura I.1.3.

Cuando una trama Ethernet llega al controlador de red de Ethernet desde el medio de transmisión, los datos se pasa hacia arriba para que sea el protocolo ARP (Address Resolution Protocol) o protocolo IP (Protocolo de Internet) quien lo procese. El valor del campo **ethertype** en el protocolo Ethernet determina si los datos de la trama Ethernet se pasan al protocolo ARP o protocolo IP como se muestra en la I.1.3.

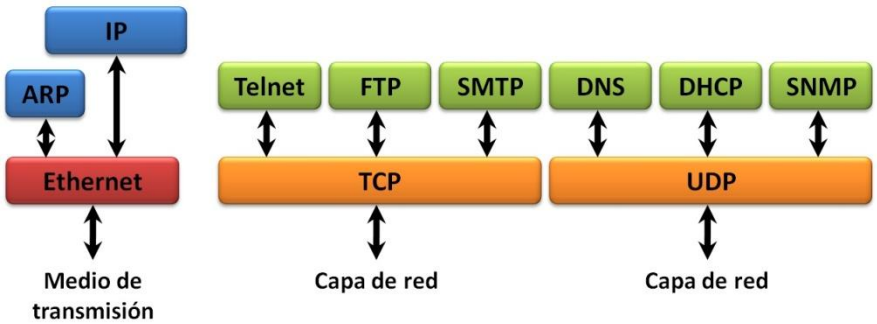


Figura I.1.3. Multiplexión de n a 1 y demultiplexión de 1 a n.

Cuando Ethernet pasa sus datos al protocolo IP (capa de red), los datos del datagrama IP se pasan hacia arriba a TCP o UDP; esto se determina por el valor que este configurado en el campo **protocolo** del encabezado IP.

Si los datos del datagrama IP (datagrama UDP) son enviados al protocolo UDP, los datos del datagrama UDP (los mensajes de la aplicación) se mandan hacia arriba al protocolo de la capa de aplicación, esto basado en el valor del campo **puerto destino** del encabezado UDP. Si los datos del datagrama IP (segmento TCP) son enviados al protocolo TCP, los datos del segmento TCP (los datos de la aplicación) se mandan hacia arriba al protocolo de capa de aplicación, esto basado en el valor del campo **puerto destino** del encabezado TCP.

La multiplexión hacia abajo es fácil de realizar porque a partir de cada punto de partida hay únicamente una trayectoria hacia abajo; cada protocolo de una capa añade su información de encabezado por lo que el paquete puede ser demultiplexado en la terminal de destino.

Los datos de la capa de aplicación que son enviados a los protocolos TCP o UDP, y el datagrama UDP o segmento TCP son siempre encapsulados como datagramas IP y se envían hacia abajo al controlador de red de la interfaz para que sean enviados a través del medio de transmisión.

Aunque los protocolos de la arquitectura TCP/IP son compatibles con muchas interfaces de red, Ethernet se utiliza para todos los ejemplos, ya que es la interfaz de red más común utilizada por debajo del protocolo IP. La terminal en la Figura I.1.1 tiene una sola interfaz de red Ethernet, la cual tiene una dirección física o dirección MAC (Media Access Control) de 6 bytes y son únicas para cada interfaz de red dentro de una red. La dirección física o dirección MAC se encuentra grabada en la memoria ROM de la interfaz de red.

Las terminales también tiene una dirección lógica o dirección IP de 4 bytes de longitud. Esta dirección IP se encuentra en un archivo de configuración en el sistema operativo. La dirección IP debe ser única para Internet.

Una terminal conectada en la red, siempre conoce su dirección IP y su dirección física.

I.1.4. Dos interfaces de red

Si una terminal está conectada a dos redes Ethernet como se muestra en la figura I.1.4; entonces, la terminal tendrá dos direcciones MAC y por lo tanto tendrá dos direcciones IP, pertenecientes a cada segmento de red.

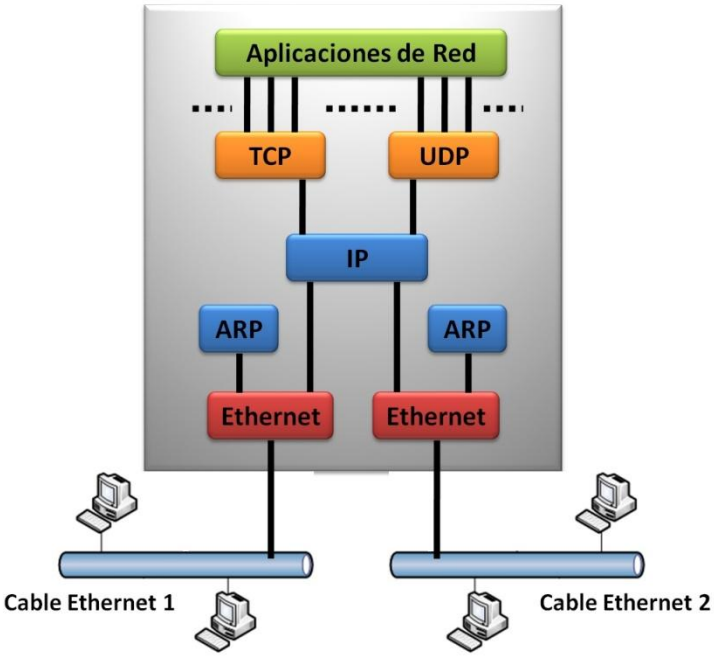


Figura I.1.4. Nodo básico de red TCP/IP, con dos interfaces.

Se observa en la estructura que para terminales con más de una interfaz de red, el protocolo IP es un multiplexor de `n a m` y un demultiplexor de `m a n`, figura I.1.5

Se realiza esta multiplexión en cualquier dirección para acomodar los datos entrantes y salientes. El modulo del protocolo IP con más de una interfaz de red es más complejo que con una sola interfaz, ya que se pueden enviar datos de una red a otra. Los datos pueden llegar de cualquier interfaz de red y ser enviados por otra interfaz de red.

El proceso de enviar un paquete IP hacia otra red se llama ruteo o reenvío de paquetes IP. Una terminal que ha sido dedicada a la labor de reenviar paquetes IP es llamada ruteador IP.

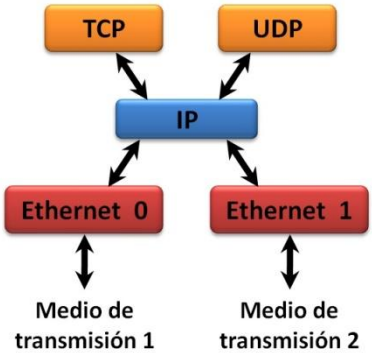


Figura I.1.5. Multiplexor de n a m y demultiplexor de m a n.

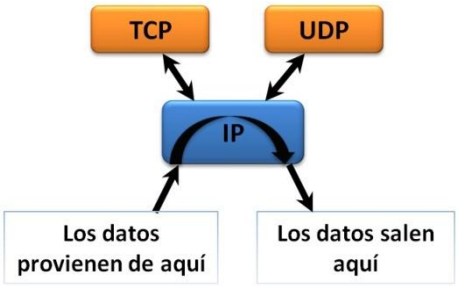


Figura I.1.6. Ejemplo de reenvío IP para paquetes IP.

Como se observa en la figura I.1.6, el paquete IP transmitido nunca toca los módulos de los protocolos TCP y UDP en el ruteador IP. Algunas implementaciones de los ruteadores IP no tienen los módulos de los protocolos TCP o UDP.

I.1.5. El protocolo IP crea una red lógica única

El protocolo IP es fundamental para el éxito de la arquitectura TCP/IP. Cada protocolo o controlador de red añade su encabezado al mensaje, ya que el mensaje pasa a través de la pila de protocolos. De la misma forma cada protocolo o controlador de red quita el encabezado correspondiente del mensaje, cuando el mensaje sube por la pila de protocolos hacia la aplicación. El encabezado IP contiene la dirección IP la cual es utilizada para construir una red lógica única a partir de múltiples redes físicas; a esta interconexión de redes físicas se le da el nombre de **Internet**. Entonces, se puede decir que el Internet es un conjunto de redes físicas interconectadas que limitan el rango de un paquete IP.

I.1.6. Independencia de la interfaz de red

El protocolo IP le da independencia al hardware de la interfaz de red. Si se desea desarrollar una nueva red física, se puede poner en servicio mediante la implementación de un nuevo controlador que se conecta a Internet por debajo del protocolo IP. Por lo tanto, las aplicaciones de red permanecen intactas y no son afectadas por los cambios de hardware en la capa de interfaz de red.

I.1.7. Interoperabilidad

Si dos equipos en Internet se pueden comunicar se dice que interoperan; si una implementación de la arquitectura TCP/IP es buena se dice que tiene interoperabilidad. Los usuarios de terminales de propósito general se benefician de la instalación de Internet porque les da interoperabilidad con las terminales en el mercado. Generalmente cuando se tiene una computadora, esta es capaz de interoperar; pero si la computadora no puede interoperar y la interoperabilidad no se puede añadir, podríamos decir que esta ocupa un lugar raro y que solo puede ser útil al propietario.

I.2. La Interfaz de sockets

La forma más básica de construir un sistema de comunicación es implementando la Interfaz de Programación de Aplicaciones de Socket (API de Socket). Como se menciona en el manual de socket (`man socket`), éste es un punto final de comunicación al que se le puede relacionar una dirección IP o nombre. Cada socket en uso tiene un tipo y puede tener uno o más procesos asociados. Los sockets existen dentro de los dominios de comunicación. Un dominio de la comunicación es una abstracción introducida para agrupar las propiedades comunes de los procesos de comunicación a través de los sockets.

I.2.1. Tipos de sockets

Los sockets son programados de acuerdo a las propiedades de comunicación que el usuario requiera. Los procesos son configurados para comunicarse sólo entre sockets del mismo tipo.

Varios tipos de sockets están actualmente disponibles:

Sockets de flujo (SOCK_STREAM)

Provee un flujo bidireccional, fiable, secuenciado, sin duplicidad de datos y sin límite de registro. Aparte de la naturaleza bidireccional del flujo de datos, un par de sockets de flujo conectados proporcionan una interfaz casi idéntica a la de los pipes que crean un canal unidireccional de flujo de bytes que sirve para la comunicación entre dos procesos (padre e hijo, dos hermanos, etc).

Sockets de datagrama (SOCK_DGRAM)

Soporta un flujo bidireccional de datos que no está garantizado a ser secuenciado, confiable, o sin duplicidad. Esto es, un proceso recibiendo mensajes con un socket datagrama puede encontrar mensajes duplicados, y posiblemente en un orden distinto del que fueron enviados. Una característica importante de un socket de datagramas, es que los límites de registro en los datos se conservan. Los sockets de datagramas modelan de cerca las facilidades que se encuentran en muchas de las redes de conmutación de paquetes actuales (por ejemplo, Ethernet).

Sockets crudos (SOCK_RAW)

Provee a los usuarios un acceso a los protocolos de comunicación por debajo de la capa de aplicación, los cuales respaldan la abstracción de los sockets. Estos sockets son normalmente orientados a datagrama, aunque sus características exactas dependen de la interfaz proporcionada por el protocolo.

Los sockets crudos no están diseñados para los usuarios en general; éstos han sido proporcionados principalmente para cualquier persona interesada en desarrollar nuevos protocolos de comunicación o en el acceso a algunas de las características más ocultas de un protocolo existente.

I.2.2. Creación de sockets

Para crear un socket, se utiliza la función `socket()`

```
sock_descr=socket(int domain, int type, int protocol);
```

Esta función solicita que un socket sea creado en un dominio determinado (`domain`) y que sea de un tipo específico (`type`) dentro del dominio. Un protocolo particular puede también ser solicitado, pero si se deja sin especificar el atributo `protocol` (con valor de 0), el sistema selecciona un protocolo apropiado de los protocolos que conforman el dominio de comunicación y que puede apoyar al tipo de socket solicitado. La función `socket()` devuelve un descriptor de socket que puede ser utilizado en las funciones posteriores que operan en sockets.

El dominio es especificado como una de las constantes indiscutibles definidas en <`sys/socket.h`>. Para el dominio de Internet, esta constante es `AF_INET`. Los tipos de sockets también se definen en este archivo; puede ser especificado como `SOCK_STREAM, SOCK_DGRAM, o SOCK_RAW.`

Para crear un socket de flujo en el dominio Internet, se usa la siguiente sinopsis:

```
tcp_socket=socket(AF_INET, SOCK_STREAM, 0);
```

Con esta llamada se crea un socket de flujo con el protocolo TCP, el cual proporcionar el soporte de comunicación.

El protocolo por default que se elige para el argumento `protocol` en la llamada socket es 0 y es correcto para la mayoría de las configuraciones; aun así, es posible especificar un protocolo diferente de 0.

Cuando se llama a la función `socket`(), esta puede fallar por cualquier razón, incluyendo:

- falta de memoria (ENOBUFS)
- solicitud por un protocolo desconocido (EPROTONOSUPPORT)
- solicitud por un tipo de socket para el cual no hay protocolo de soporte (EPROTOTYPE).

I.2.3. Vinculación de sockets con nombres locales

Un socket es creado sin un nombre o dirección; entonces, hasta que un nombre o dirección sea ligado a un socket los procesos no tienen manera de hacer referencia a éste, y en consecuencia no puede recibir ningún mensaje. Los procesos de comunicación están sujetos por una "asociación".

En el dominio de Internet, una asociación está compuesta de direcciones IP y puertos locales y remotos. En la mayoría de los dominios, las asociaciones deben ser únicas; en el dominio de Internet éstas nunca deben ser duplicadas <`dirección IP local, puerto local, dirección IP remota, puerto remoto`>.

La función `bind()` permite a un proceso especificar la mitad de una asociación (<`dirección IP local, puerto local`>) mientras que la funciones `connect()` y `accept()` son usadas para realizar una asociación de socket.

Vincular nombres a sockets puede ser bastante complejo. Por suerte, en lo general no tiene que vincular explícitamente una dirección IP y número de puerto a un socket, ya que las funciones `connect()` y `send()` vinculan automáticamente una dirección apropiada cuando son usadas con un socket vinculado.

La función `bind()` tiene esta sinopsis:

```
bind(int sockfd, const struct sockaddr *addr,socklen_t addrlen);
```

El nombre vinculado (`*addr`) es una cadena de bytes de longitud variable que es interpretada por los protocolos. Su interpretación puede variar dependiendo del dominio de comunicación (esta es una de las propiedades que constituyen el dominio). Como se mencionó anteriormente, los nombres de dominio de Internet contienen una dirección IP y número de puerto.

Para vincular una dirección IP y un puerto, se usa la siguiente sinopsis:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

...
struct sockaddr_in srcaddr;
...
bind(tcp_socket, (struct sockaddr *)&srcaddr, sizeof (srcaddr));
```

I.2.4. Estableciendo conexión

El establecimiento de conexión es usualmente una tarea asimétrica; un proceso es un cliente y el otro proceso es un servidor. Cuando el servidor esta dispuestos a ofrecer sus servicios, vincula un socket a una dirección IP y un puerto bien conocido, lo asocia con el servicio y luego pasivamente escucha en su socket. Un proceso no relacionado puede entonces conectarse con el servidor.

El cliente solicita los servicios del servidor mediante el inicio de una conexión al socket del servidor. Para iniciar la conexión, el cliente utiliza la función `connect()`. Esto se configura de la siguiente forma:

```
struct sockaddr_in server;
...
connect(tcp_socket, (struct sockaddr *)&server, sizeof (server));
```

Donde la estructura `server` contiene la dirección IP y el número de puerto al que el cliente desea conectarse. Si el socket del cliente no está vinculado en el momento de llamar a la función `connect()`, el sistema selecciona automáticamente y vincula un nombre al socket si es necesario (esto es la forma en cómo direcciones locales son vinculadas a un socket).

I.2.5. Escuchar en un socket

Para que el servidor reciba la conexión de un cliente, el servidor debe después de vincular su socket:

- Indicar disposición a escuchar las solicitudes de conexión entrantes.
- Realmente aceptar la conexión.

Para indicar la disposición a escuchar las solicitudes de conexión, el servidor utiliza la función `listen()`:

```
listen(tcp_socket, 5);
```

El argumento `backlog = 5` de la función `listen()` especifica el número máximo de conexiones pendientes que pueden estar en cola de espera para la aceptación del proceso servidor.

Si una conexión es solicitada mientras la cola del servidor está llena, la conexión no se rechazará. En su lugar, los mensajes individuales que componen la solicitud serán ignorados, forzando al cliente a volver a intentar la conexión. Mientras que el cliente vuelve a reenviar la solicitud de conexión, el servidor tiene tiempo para hacer espacio en su cola de conexiones pendientes.

Si la función `connect()` regresa el valor ECONNREFUSED, el cliente no será capaz de decir si el servidor está activo o no. Si el servidor sigue ignorando las solicitudes de conexión, es posible obtener el error ETIMEDOUT.

I.2.6. Aceptando una conexión

Con un socket indicando la disposición para aceptar conexiones, el servidor puede aceptar una conexión de la siguiente forma:

```
struct sockaddr_in client;
...
clientlen = sizeof (client);
newsock = accept( tcp_socket, (struct sockaddr *)&client,
&clientlen);
```

Cuando una conexión es aceptada un nuevo descriptor es regresado (junto con un nuevo socket).

Si el servidor quiere averiguar quién es su cliente, éste puede proporcionar un buffer para el nombre del socket del cliente. El argumento `clientlen` es inicializado por el servidor para indicar cuánto espacio es asociado con la estructura `client`, y es entonces modificado o regresado para indicar el verdadero tamaño de la estructura. Si el nombre del cliente no es de interés, la estructura puede ser un puntero NULL.

La función `accept()` normalmente es bloqueante, no va a regresar nada hasta que haya una conexión disponible o hasta que la función sea interrumpida por una señal a el proceso.

Por otra parte, un proceso no puede indicar que aceptará conexiones de un solo cliente o de clientes específicos. El proceso de usuario se encarga de considerar de quien es la conexión y de cerrar la conexión si no desea hablar con el proceso.

I.2.7. Transferencia de datos

Con una conexión establecida los datos se pueden comenzar a transmitir. Para enviar y recibir datos se puede elegir entre varias funciones.

Si las entidades del mismo nivel en cada extremo de una conexión se enlazan, se puede enviar o recibir un mensaje sin especificar el destino. En este caso, puede utilizar la funciones `read()` y `write()`:

```
#include <unistd.h>
read(int tcp_socket, void *buf, size_t count);
write(int tcp_socket, const void *buf, size_t count);
```

Además de `read()` y `write()`, se pueden utilizar las funciones `recv()` y `send()` de la siguiente forma:

```
#include <sys/types.h>
#include <sys/socket.h>
send(int tcp_socket, const void *buf, size_t len, int flags);
recv(int tcp_socket, void *buf, size_t len, int flags);
```

Aunque `recv()` y `send()` son prácticamente idénticas a `read()` y `write()`, el argumento extra `flags` es importante (los valores de `flags` son definidos en <`sys/socket.h`>). Una o más de las siguientes banderas pueden ser especificadas:

`MSG_OOB`: Envía o recibe datos fuera de banda.
`MSG_PEEK`: Examina los datos sin leerlos.
`MSG_DONTROUTE`: Envía datos sin ruteo de paquetes.

Datos fuera de banda es una opción específica para sockets de flujo. La opción enviar datos sin ruteo, es aplicada a los paquetes que son utilizados para el proceso de administración de la tabla de ruteo y es poco probable que sea de interés para el usuario normal.

Por otro lado, la capacidad de ver previamente los datos puede ser muy útil. Cuando MSG_PEEK se especifica con la función `recv()`, pues hace que la operación de recepción devuelva datos del principio de la cola de recepción sin quitarlos de allí. Así, una próxima llamada de recepción devolverá los mismos datos; esto es, la siguiente función `read()` o `recv()` aplicada al socket regresara los datos previamente vistos.

I.2.8. Cerrando el socket

Una vez que el socket ya no es de interés, éste puede ser cerrado por la aplicación, utilizando la función `close()`:

```
#include <unistd.h>
int close(int tcp_socket);
```

Si los datos están asociados con un socket del tipo SOCK_STREAM y se llama a la función `close()`, el sistema continuará anteponiendo la transmisión de

los datos. Sin embargo, si después de un período largo de tiempo los datos todavía no han sido entregados, éstos serán descartados. Si no tiene datos en espera, puede utilizar `shutdown()` en el socket antes de cerrarlo:

```
#include <sys/socket.h>
int shutdown(int sockfd, int how);
```

Donde el argumento `how` es 0 si no está muy interesado en la lectura de datos, 1 si no hay más datos para ser enviados, o 2 si no hay datos para ser enviados o recibidos.

I.2.9. Sockets no orientados a conexión (datagrama)

Hasta ahora, se ha visto todo sobre sockets que siguen el modelo orientado a conexión. Pero hay también el soporte para comunicaciones no orientadas a conexión o de datagrama, las cuales se implementan en las redes de conmutación de paquetes. Un socket de datagramas ofrece una interfaz simétrica para el intercambio de datos, y aunque los procesos siguen siendo clientes y servidores no hay ningún requisito de que se establezca conexión; pero si es requisito que cada mensaje incluya la dirección de destino.

Los sockets de datagrama se crean de la misma forma que los sockets de flujo. Si una dirección local especifica es necesaria, la función `bind()` tiene que ser llamada antes de la transmisión de datos. De lo contrario, el sistema establecerá la dirección y puerto cuando los datos sean enviados por primera vez.

Para enviar los datos, se utiliza la función `sendto()`:

```
#include <sys/types.h>
#include <sys/socket.h>
sendto(int udp_socket, const void *buf, size_t len, int flags,
const struct sockaddr *dest_addr, socklen_t addrlen);
```

Los argumentos `udp_socket, buf, len` y las banderas se configuran igual que con `send()`. La estructura `dest_addr` y `addrlen` indican la dirección del destinatario del mensaje y la longitud de la misma.

Cuando se está usando una interfaz de datagramas no fiable, es poco probable que algunos errores sean reportados a la terminal que transmite. Cuando un mensaje no se puede enviar (por ejemplo, una red es inaccesible), la función `sendto()` devolverá -1 y la variable global `errno` contendrá un número de error.

Para recibir mensajes con un socket de datagramas sin conexión, utilizamos la función `recvfrom()`:

```
#include <sys/types.h>
#include <sys/socket.h>
recvfrom(int udp_socket, void *buf, size_t len, int flags,
              struct sockaddr *dst_addr, socklen_t *addrlen);
```

Como se mencionó anteriormente, la longitud de la estructura o `addrlen` es un argumento de valor en los resultados, que contiene inicialmente el tamaño de la memoria asociada con la estructura `dst_addr`, y modificado en la recepción para indicar el tamaño real de la dirección de la terminal que envío el datagrama.

I.3. Direccionamiento IP

Cada equipo de cómputo que está conectado a Internet a través de un ISP (Internet Service Provider), de un proveedor Internet móvil, o en una red corporativa requiere de una dirección IP ya sea pública o privada. Para poder administrar de forma correcta y solucionar los problemas de conectividad entre redes IP, es importante conocer y comprender todos los aspectos relacionados con el direccionamiento IP.

Uno de los aspectos más importantes que se deben conocer en las redes TCP/IP es la asignación de direcciones IP. Se debe tener en cuenta que cada interfaz de red dentro de una subred IP debe tener una dirección IP única y correcta.

I.3.1. Tipos de direcciones IP

Las direcciones IP son direcciones lógicas que operan a nivel de capa de red; estas direcciones lógicas son de 32 bits y puedes ser de los siguientes tipos:

Unidifusión: Las direcciones IP de unidifusión (unicast) son aquella que se asignan solamente a una interfaz de red conectada dentro de una subred IP. Este tipo de direcciones se utilizan para realizar una comunicación de uno a uno.

Difusión: Una dirección IP de difusión (broadcast) está diseñada para que una terminal pueda enviar un mensaje a todas las terminales dentro de la misma subred o red IP. Este tipo de direcciones se utilizan para realizar comunicación de uno a todos.

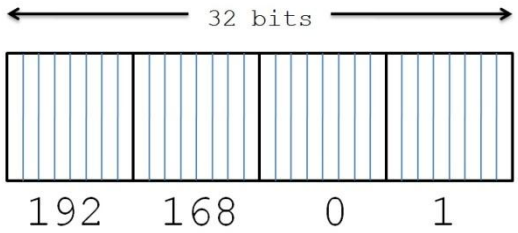


Figura I.3.1. Expresión de las direcciones IP de 32 bits.

Multidifusión: Las direcciones IP de multidifusión (multicast) están diseñadas para que una o varias terminales puedan escuchar un mensaje, ya sea que estén en el mismo o en diferentes subredes IP. Este tipo de direcciones son utilizadas en comunicaciones de una a varias terminales, siempre y cuando pertenezcan éstas al grupo de multidifusión.

I.3.2. Estructura de las direcciones IP

Las direcciones IP son direcciones de 32 bits que se expresan en 4 octetos de forma decimal como se muestra en la figura I.3.1. Cada octeto en la dirección IP puede tener un valor máximo de 255. Las direcciones IP de unidifusión contienen un ID de red y un ID de host, figura I.3.2.

El ID de red (dirección de red) o ID de subred sirve para identifica las terminales dentro de la red IP o subred IP; una subred IP coincide con un segmento de red de broadcast limitado por el ruteador IP.

El ID de host o dirección de host identifica a la terminal dentro de la subred IP. Una terminal puede ser cualquier dispositivo que tenga una interfaz de red como un ruteador, un switch o un host. Como ya se había mencionado el ID de host debe ser único dentro de la subred IP.

I.3.3. Clases de direcciones IP

Debido a la rápida expansión de lo que ahora se conoce como Internet, se concluyó que la asignación basada en clases no era suficiente, dado que las direcciones IP se terminarían muy rápido. Para evitar este problema, actualmente el direccionamiento en Internet no se basa en clases. Sin embargo, la forma más fácil de comprender el direccionamiento en Internet es a través de clases.

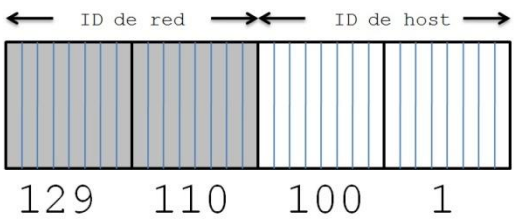


Figura I.3.2. Estructura de una dirección IP con ID de red e ID de host.

Las autoridades del Internet definieron en el RFC 791 las direcciones IP de unidifusión en términos de clases, para crear redes de diversos tamaños. El objetivo del RFC era generar un diseño de redes de la siguiente forma:

- Pocas redes con una gran cantidad de direcciones IP de host (Clase A).
- Un número mediano de redes con un número moderado de direcciones IP de host (Clase B).
- Una gran cantidad de redes con pocas direcciones IP de host (Clase C).

El resultado de esto fue la terminación de direcciones teniendo un alto crecimiento de Internet. Por ello las clases con gran cantidad de direcciones IP de host se subdividen en ID de subredes e ID de host.

I.3.3.1. Direcciones de Clase A

Como se mencionó, las direcciones IP de clase A fueron diseñadas para tener una gran cantidad de direcciones IP de host. Como se muestra en la figura I.3.3 los primeros 8 bits son para el ID de red y los otros 24 bits son para el ID de host.

Para identificar las direcciones de clase A, el primer bit del ID de red siempre será 0; entonces, los valores del primer octeto van desde 1 (`00000001`) hasta 127 (`01111111`), lo que quiere decir que en la clase A solo se tienen 127 redes: desde la `1.0.0.0` hasta la `127.0.0.0`.

Las direcciones IP de host validas se calcula con la siguiente fórmula:

$$\mathbf{Host=2^n - 2}$$

donde `n` es el número de bits para el ID de host y `-2` porque en el ID de host no se utilizan los bits en cero porque indican esta red: `1.0.0.0`; y ni todos los bits en uno porque indican una dirección de broadcast: `1.255.255.255`.

Entonces si son 24 bits para el ID de host, la cantidad de direcciones IP de host que se pueden tener en una dirección IP de red de clase A será:

$$\mathbf{Host = 2^{24} - 2 = 16,777,214}$$

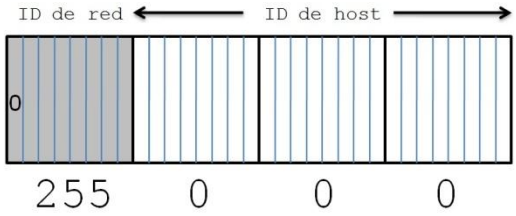


Figura I.3.3. Formato y mascara de subred de las direcciones de clase A: un octeto de ID de red y tres octetos de ID de host.

La máscara de red en las direcciones de clase A es `255.0.0.0`, los bits en uno indican el ID de red y los bits en cero indican el ID de host.

I.3.3.2. Direcciones de Clase B

Las direcciones IP de clase B fueron diseñadas para tener una cantidad moderada de direcciones IP de red, con una cantidad moderada de direcciones IP de host. Como se muestra en la figura I.3.4 los primeros 16 bits son para el ID de red y los otros 16 bits son para el ID de host.

Para identificar las direcciones de clase B, los dos primeros bits del ID de red siempre serán 10; entonces, los valores del primer octeto van desde 128 (**10**000000) hasta 191 (**10**111111), pero como son dos octetos para el ID de red en la clase B se tendrán las siguientes redes:

```
128.0.0.0, 128.1.0.0, 128.2.0.0,........., 128.255.0.0
129.0.0.0, 129.1.0.0, 129.2.0.0,........., 129.255.0.0
....          ....      ....                 ....
191.0.0.0, 191.1.0.0, 191.2.0.0,........., 191.255.0.0
```

Esto indica que por cada valor en el primer octeto se tendrán 256 redes, como se observa en la figura I.3.4 en el primer octeto los dos primeros bits nunca cambian, por lo tanto el número de redes que se pueden tener en las direcciones de clase B son:

$$\mathbf{Redes = 2^{14} = 16{,}384}$$

Las direcciones IP de host validas y posibles se calcula con la siguiente fórmula:

$$\mathbf{Host=2^{n} - 2}$$

donde `n` es el número de bits para el ID de host y `-2` porque en el ID de host no se utilizan los bits en ceros porque indica la dirección de la red: `128.10.0.0`, y ni todos los bits en uno porque indica una dirección de broadcast: `128.10.255.255`.

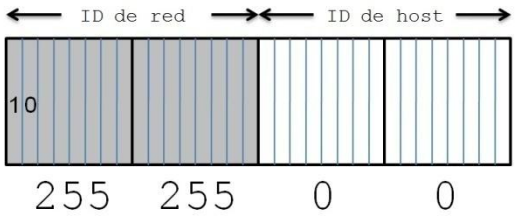


Figura I.3.4. Formato y mascara de subred de las direcciones de clase B: dos octetos de ID de red y dos octetos de ID de host.

Entonces si son 16 bits para el ID de host, la cantidad de direcciones IP de host que se pueden tener en una dirección IP de red de clase B será:

$$\mathbf{Host = 2^{16} - 2 = 65,534}$$

La máscara de red en las direcciones de clase B es 255.255.0.0, los bits en uno indican el ID de red y los bits en cero indican el ID de host.

I.3.3.3. Direcciones de Clase C

Las direcciones IP de clase C fueron diseñadas para tener una gran cantidad de direcciones IP de red con pocas direcciones IP de host. Como se muestra en la figura I.3.5 los primeros 24 bits son para el ID de red y los otros 8 bits son para el ID de host.

Para identificar las direcciones de clase C, los tres primeros bits del ID de red siempre serán 110; entonces, los valores del primer octeto van desde 192 (11000000) hasta 223 (11011111), pero como son tres octetos para el ID de red en la clase C se tendrán las siguientes redes:

Solo para 192, se tienen 2^{16} = 65,536 direcciones de red

192.**0.0**.0, 192.**0.1**.0,......, 192.**0 .255**.0
...
. . .
192.**255.0**.0, 191.**255.1**.0,....., 192.**255.255**.0

Esto indica que por cada valor en el primer octeto se tendrán 65,536 redes; como se puede ver en la figura I.3.5 en el primer octeto los tres primeros bits nunca cambian, por lo tanto el número de redes que se pueden tener en las direcciones de clase C son:

$$\mathbf{Redes = 2^{21} = 2,097,152}$$

Y la cantidad de direcciones IP de host se calcula con la siguiente fórmula:

$$\mathbf{Host=2^{n} - 2}$$

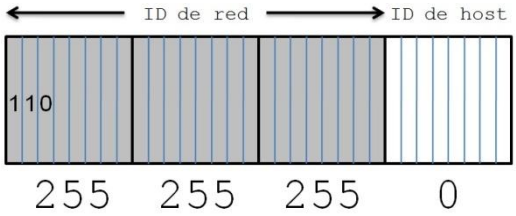


Figura I.3.5. Formato y mascara de subred de las direcciones de clase C: tres octetos de ID de red y un octeto de ID de host.

donde n es el número de bits para el ID de host y −2 porque en el ID de host no se utilizan todos los bits en cero porque indican esta red: 192.10.10.0; y ni todos los bits en uno porque es una dirección de broadcast: 192.10.10.255.

Entonces si son 8 bits para el ID de host, la cantidad de direcciones IP de host que se pueden tener en una dirección IP de red de clase C será:

$$\mathbf{Host = 2^8 - 2 = 254}$$

La máscara de red en las direcciones de clase C es 255.255.255.0, los bits en uno indican el ID de red y los bits en cero indican el ID de host.

I.3.4. Subredes y Máscara de Subred

Si se tiene una red de clase B en la cual se tienen conectados 65,534 host en la misma red de broadcast, como se muestra en la figura I.3.6; técnicamente sería posible realizarlo, pero de forma practica la comunicación entre las terminales no sería posible por la acumulación de tráfico de broadcast.

Ahora si se tiene una dirección de red de clase A, donde la cantidad de host en la misma red de broadcast puede ser hasta de 16,777,214 host. Eso sería imposible de forma práctica.

La creación de subredes está diseñada para que el uso de espacio de direcciones sea más eficaz y se tengan redes con un dominio de broadcast de menor tamaño; lo adecuado sería tener 254 host como máximo en el dominio de broadcast, pero se podrían tener desde 2 host en cada subred.

Los bits de ID de red son fijos, fueron asignados de forma global; los bits de host son variables y son los que se pueden tomar para crear IDs de subred e IDs de host.

Considerando la dirección de clase B 148.204.0.0, los dos primeros octetos son el ID de red y no pueden ser modificados; los siguientes dos octetos son los que se pueden tomar para crear subredes. Por ese motivo, se toman los 16 bits de host, 8 para la parte del ID de subredes y se dejan 8 para el ID de host. Para calcular el número de subredes se utiliza la formula:

$$\mathbf{Subredes = 2^n - 2}$$

Entonces la cantidad de subredes con 8 bits será:

$$\mathbf{Subredes = 2^8 - 2 = 254}$$

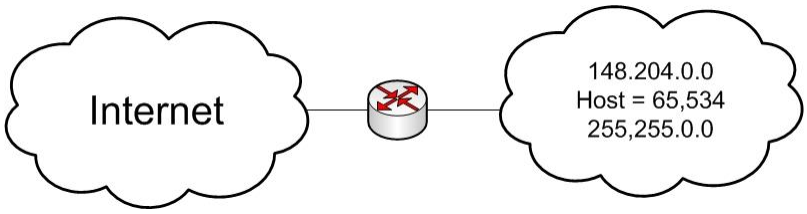


Figura I.3.6. La red 148.204.0.0 de clase B antes de ser dividida en subredes.

Como se ve en la figura I.3.7 las subredes de la dirección de red de clase B `148.204.0.0`, van desde la `148.204.1.0` hasta `148.204.254.0` con 254 host cada una.

I.3.4.1. Máscara de subred

Para que las terminales conozcan la dirección IP de red o de subred, requieren una configuración adicional para distinguir entre los bits del ID de red o ID de subred y los bits del ID de host de una dirección IP.

Entonces se define el uso de una máscara de bits que sirve para diferenciar entre los bits del ID de red y los bits del ID de host. Esta máscara de bits se llama máscara de red o de subred.

Los bits en uno indican la posición perteneciente al ID de red y los bits en cero indican la posición perteneciente al ID de host.

Tabla I.3.1. Notación de la máscara de red en decimal y prefijo.

Clase de dirección	Máscara de red	Bits para la máscara de red	Prefijo de red
Clase A	255.0.0.0	11111111 00000000 00000000 00000000	/8
Clase B	255.255.0.0	11111111 11111111 00000000 00000000	/16
Clase C	255.255.255.0	11111111 11111111 11111111 00000000	/24

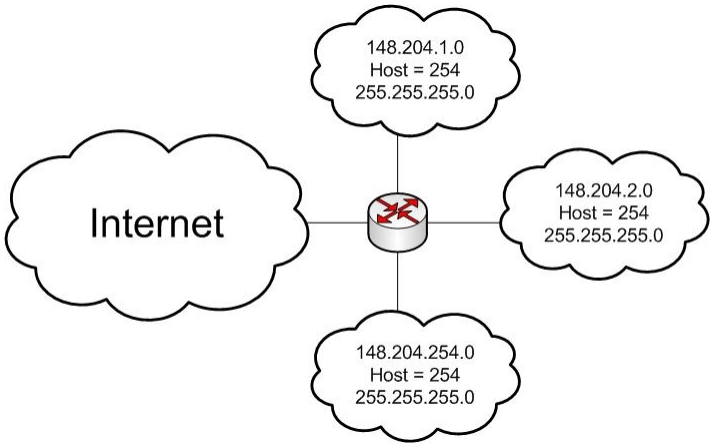
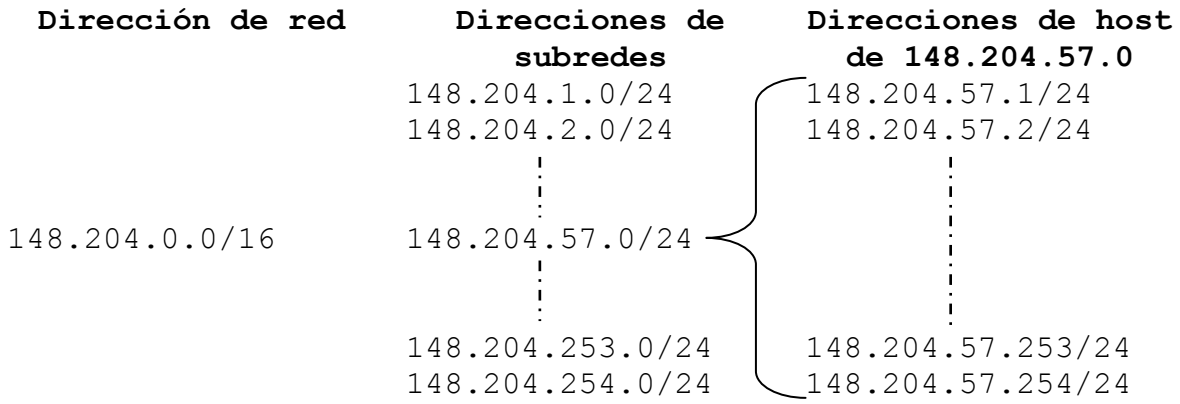


Figura I.3.7. La red 148.204.0.0 de clase B después de ser dividida en subredes.

Como se ve en la figura I.3.6 la dirección IP de red `148.204.0.0` tiene una máscara de red `255.255.0.0` o prefijo `/16`. Cuando se divide ésta red en subredes tomamos de los bits del ID de host 8 bits para crear subredes, los cuales en la máscara de red se tienen que poner en uno; entonces, con esto pasa a ser una máscara de subred como se observa en la figura I.3.7, la máscara de subred es `255.255.255.0`.

Entonces, la división de las subredes quedaría de la siguiente forma:



I.3.4.2. Determinación de la dirección IP de subred

Al configurar en un host o en un ruteador la dirección IP de host y la máscara de subred en sus interfaces de red, éstos pueden determinar que redes o subredes tienen conectadas.

Entonces para determinar la dirección IP de la subred se aplica una operación lógica AND entre la dirección IP de host y la máscara de subred, el resultado de la operación será la dirección IP de la subred.

Si a un ruteador con dos interfaces de red se le instala la siguiente configuración en la interfaz de red `e0: 148.204.1.254` con máscara de subred `255.255.255.0`, y en su interfaz de red `e1: 148.204.254.1` con máscara de subred `255.255.255.0`; entonces, el ruteador determina cuales son las subredes que tiene conectadas en sus interfaces.

```
     148.204.1.254       10010100.11001100.00000001.11111110
&255.255.255.0      11111111.11111111.11111111.00000000
      148.204.1.0       10010100.11001100.00000001.00000000

     148.204.254.1       10010100.11001100.11111110.00000001
&255.255.255.0      11111111.11111111.11111111.00000000
     148.204.254.0       10010100.11001100.11111110.00000000
```

Tabla I.3.2. Notación de la máscara de red en decimal y prefijo.

Dirección IP de red con prefijo	Máscara de subred (bits de subred)	Rango de las direcciones IP de subredes
148.204.0.0./20	255.255.**1111**0000.0	148.204.**0001**0000.0 a 148.204.**1110**0000.0
148.204.0.0./21	255.255.**11111**000.0	148.204.**00001**000.0 a 148.204.**11110**000.0
148.204.0.0./22	255.255.**111111**00.0	148.204.**000001**00.0 a 148.204.**111110**00.0
148.204.0.0./23	255.255.**1111111**0.0	148.204.**0000001**0.0 a 148.204.**1111110**0.0
148.204.0.0./24	255.255.**11111111**.0	148.204.**00000001**.0 a 148.204.**11111110**.0

De esta forma el ruteador aprende que en su interfaz de red `e0` tiene conectada la subred `148.204.1.0`, y en la interfaz de red `e1` tiene conectada la subred `148.204.254.0`.

Se muestra en la tabla I.3.2 la dirección IP de clase B con diferentes valores en el prefijo y el rango de subredes que se generan.

Ejercicio I.3.1. Como administrador de red, se le asigna la dirección de red 190.10.0.0 y su jefe le dice que cree la mayor cantidad de subredes que tengan 100 host únicamente.

1. ¿Cuántas subredes se crean?
2. ¿Cuántos host por subred?
3. ¿Cuál es la máscara de subred?
4. Indique el rango de subredes e
5. Indique el rango de los host en la subred 200.

I.3.5. Uso de la máscara de subred

Como ya se mencionó, para conocer la dirección IP de subred tanto el host como el ruteador se aplica una operación AND con la dirección IP de host y la máscara de subred.

```
dirección IP de host
 &máscara de subred
dirección IP de subred
```

I.3.5.1. Uso de la máscara de subred en el host

Cuando una terminal requiere comunicarse con una terminal remota, lo primero que hace es investigar si la terminal con dirección IP destino está en la misma subred o está fuera de la subred; para ello, aplica una operación AND entre la dirección IP origen y su máscara de subred, y entre la dirección IP destino y su máscara de subred.

dirección IP origen	dirección IP destino
&máscara de subred	&máscara de subred
IP de subred A	IP de subred B

Compara el resultado de la IP de subred A con el de la IP de subred B, si son iguales las dos IP de las subredes, la terminal origen aprende que el destino está en la misma subred, pero si los resultados son diferentes quiere decir que el destino está fuera de la subred. Por ejemplo, supongamos que la terminal A desea comunicarse con la terminal B como se muestra en la figura I.3.8; entonces, la terminal A aplica las operaciones AND como sigue:

148.204. 1 .3	148.204. 1 .2
&255.255.255.0	&255.255.255.0
148.204. 1 .0	148.204. 1 .0

Como se puede observar los dos resultados son iguales, por lo tanto la terminal A aprende que la terminal B está en la misma subred; entonces, investiga su dirección física con el protocolo ARP para poder enviarle los mensajes.

En otro caso, si la terminal A requiere comunicarse con la terminal C, entonces la terminal A aplica las operaciones AND como sigue:

148.204. 1 .3	148.204.254.2
&255.255.255.0	&255.255.255.0
148.204. 1 .0	148.204.254.0

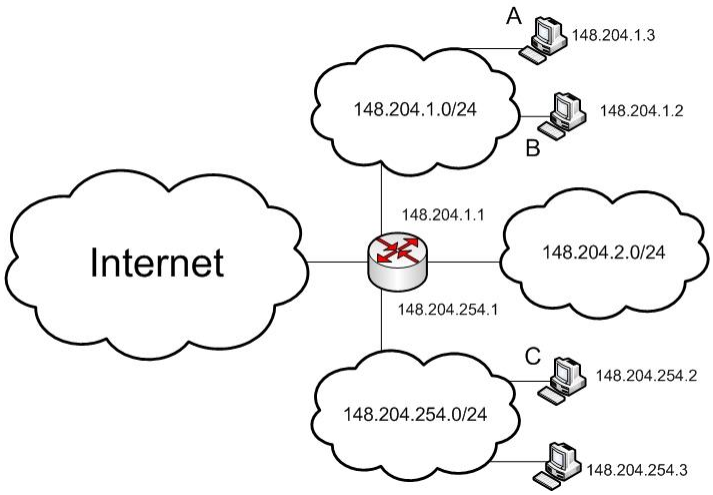


Figura I.3.8. Uso de la máscara de subred en el host .

Se puede ver que los resultados son diferentes, por lo tanto la terminal A aprende que la terminal C esta fuera de la subred; por lo tanto los paquetes se los tiene que enviar a la puerta de enlace, que en este caso es la dirección IP `148.204.1.1`. Al igual que en el caso anterior, la terminal A necesita conocer la dirección física de la puerta de enlace, para poder enviarle los paquetes.

I.3.5.2. Uso de la máscara de subred en el ruteador

Los ruteadores son dispositivos de conectividad que operan a nivel de capa de red, los cuales realizan la función de ruteo de paquetes que es la función de buscar, elegir e implementar la mejor trayectoria hacia un destino; entonces, guardan los destinos en una tabla llamada tabla de ruteo. Ver tabla I.3.3

Tabla I.3.2. Tabla de ruteo del ruteador R1. Ver figura I.3.9.

Destino	Mascara de subred	Siguiente HOP	Tipo	Métrica
148.204.1.0	255.255.255.0	148.204.1.1	DIR	1
148.204.2.0	255.255.255.0	148.204.2.1	DIR	1
148.204.254.0	255.255.255.0	148.204.2.2	REM	2

Por ejemplo, si la terminal A requiere enviar un mensaje a la terminal B, ver figura I.3.9. Cuando un datagrama IP llega al ruteador R1, éste analiza la dirección IP destino del datagrama IP para determinar a qué siguiente ruteador debe ser transmitido. Entonces, el ruteador R1 toma la dirección IP destino del datagrama IP y le aplica la máscara de subred de la primera línea de su tabla de ruteo:

```
  148.204.254.3
&255.255.255.0
---------------
  148.204.254.0
```

El ruteador R1 compara el resultado `148.204.254.0` con el destino de la primera línea `148.204.1.0`, como el resultado es diferente, se pasa a la segunda línea y aplica la máscara de subred de esa línea:

```
  148.204.254.3
&255.255.255.0
---------------
  148.204.254.0
```

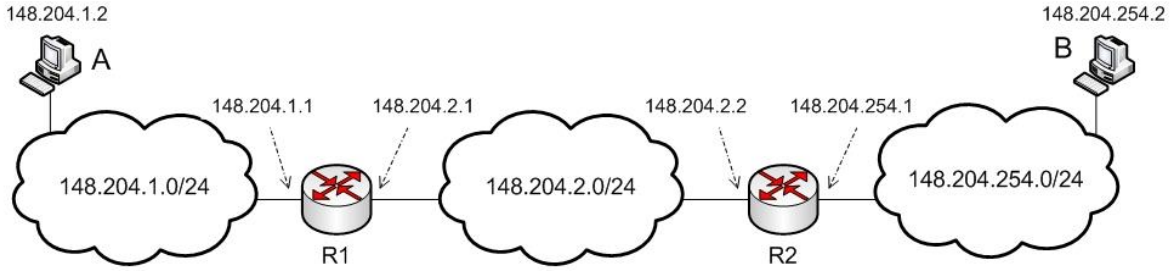


Figura I.3.8. Uso de la máscara de subred en el ruteador.

Compara `148.204.254.0` con `148.204.2.0` y como son diferentes aplica la máscara de subred de la tercera línea:

```
   148.204.254.3
  &255.255.255.0
  ---------------
   148.204.254.0
```

Compara el resultado `148.204.254.0` con el destino de la tercera línea que es `148.204.254.0`; entonces, como son iguales retransmite el paquete al siguiente HOP con dirección `148.204.2.2` que es la dirección IP cercana del ruteador R2.

Ejercicio I.3.2. Elabore las tablas de ruteo (como la de la tabla I.3.2 de la figura I.3.9 si la dirección IP de red es 192.1.1.0 y utilizamos el prefijo /28.

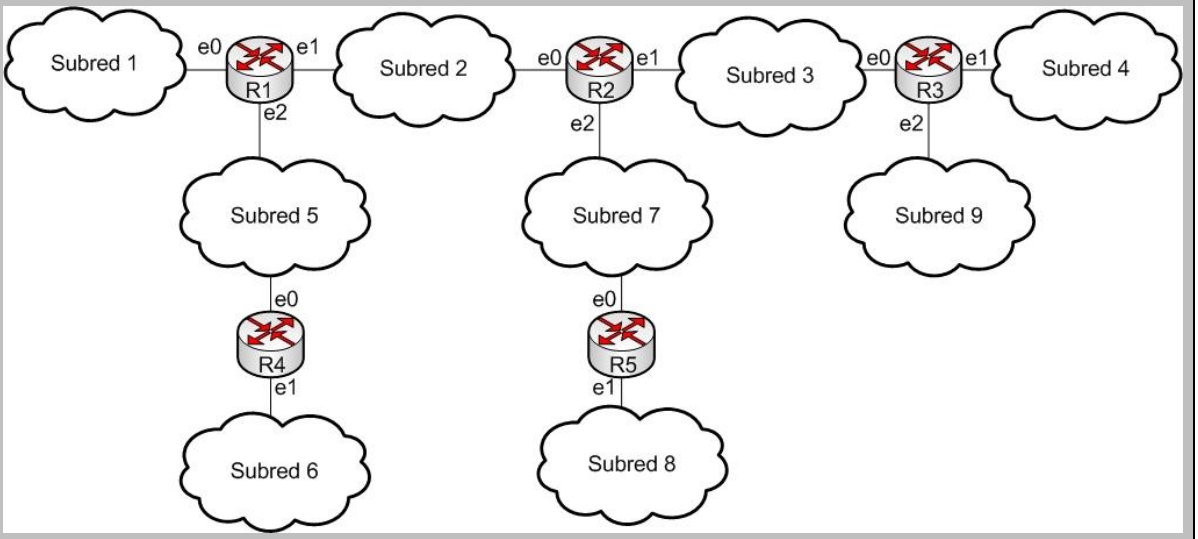


Figura I.3.8. Elaboración de tablas de ruteo con 192.1.1.1/28.

I.4. Aplicaciones cliente/servidor

El paradigma más utilizado en el desarrollo de aplicaciones distribuidas es el modelo cliente/servidor. En este esquema, las aplicaciones cliente solicitan los servicios de un proceso servidor. Esto implica la asimetría en el establecimiento de la comunicación entre el cliente y el servidor, como se mencionó en el apartado I.2.

El cliente y el servidor requieren una correcta configuración de sus argumentos antes de que un servicio pueda ser prestado (y aceptado). Este conjunto de convenciones comprende un protocolo que debe implementarse en ambos extremos de una conexión. Dependiendo de la situación, el protocolo

puede ser simétrico o asimétrico. En un protocolo simétrico, cada lado puede desempeñar la función de maestro o esclavo. En un protocolo asimétrico, un lado está siempre como el maestro y el otro como esclavo.

Un ejemplo de un protocolo simétrico es el Protocolo de Trasferencia de Archivos Trivial (TFTP) utilizado en Internet. Un ejemplo de un protocolo asimétrico es el Protocolo de Transferencia de Archivos (FTP), también utilizado en Internet. No importa si el protocolo específico utilizado en la obtención de un servicio es simétrico o asimétrico, cuando se accede a un servicio siempre hay un proceso cliente y un proceso servidor.

En primer lugar, se deben considerar las propiedades de los procesos del servidor; entonces, se podrán considerar las propiedades de los procesos de cliente.

I.4.1. Servidor

Un proceso servidor normalmente presta un servicio a través de un puerto bien conocido por las solicitudes de servicio de los clientes. Es decir, el proceso de servidor permanece latente hasta que un cliente solicita una conexión a la dirección del servidor. En ese momento, el proceso servidor se activa para dar servicio al cliente, realizando las acciones apropiadas a las solicitudes de los clientes.

Empleando esquemas alternativos como un "servidor de servicio" puede ser utilizado para eliminar un cumulo de procesos del servidor que obstruyen al sistema sin dejar de ser más inactivos del tiempo. Para los servidores de Internet, este esquema ha sido implementado vía `inetd`, el llamado SuperServidor Internet.

El servidor `inetd` lee un archivo de configuración en el arranque y escucha a una variedad de puertos basados en el contenido del archivo. Cuando una conexión es solicitada a un puerto que está escuchando en `inetd`, `inetd` ejecuta el programa servidor adecuado para atender al cliente.

I.4.2. Cliente

Un cliente suele ser un programa que se ejecuta bajo demanda de un usuario y que precisa del acceso a algún recurso localizado en una terminal remota. Para conseguir este acceso, lo que hace es ponerse en contacto con el servidor que administra ese recurso de la terminal remota.

I.4.3. Interacción entre el cliente y el servidor

Por ejemplo, en una aplicación como FTP, el cliente es una aplicación que permite al usuario indicar con que terminal conectarse (dirección IP y puerto), tener acceso a los directorios de la terminal remota y leer o enviar archivos entre la terminal local y la terminal remota. El cliente establece una conexión TCP con el servidor FTP, y los datos que necesita para realizar dicho establecimiento de conexión son la dirección IP y el puerto 21 o el nombre del servidor para que el protocolo DNS (Domain Name Service) resuelva la dirección IP.

Una vez establecida la conexión, el cliente FTP envía comandos especiales que el servidor entiende para cambiar de directorio, leer un archivo, desplegar la lista de archivos, etc. El servidor es, por tanto, capaz de responder a los comandos enviados por el cliente.

I.4.4. Clasificación de los servidores

Hay servidores que reciben peticiones de servicio y las atienden vía TCP, y otros utilizan UDP. Algunos servidores incluso pueden dar servicio a través de TCP y/o UDP en función de lo que le convenga a él o al cliente; estos servidores son llamados multiprotocolo. Otro aspecto de los servidores, es que algunos suelen estar configurados para atender a múltiples clientes, de la siguiente forma:

- Secuencial: Mientras el servidor atiende a un cliente, los demás tienen que esperar, o bien se les niega el servicio.
- Concurrente: El servidor es capaz de atender a múltiples clientes al mismo tiempo.

Una terminal que desee prestar varios servicios simultáneamente puede disponer de múltiples servidores ejecutándose concurrentemente. Cada servidor estará esperando peticiones de servicio con un protocolo y un puerto dado. Es poco probable o inclusive nulo que una petición de un cliente sea atendida por un servidor equivocado; toda la información enviada por los clientes contiene los campos protocolo y puerto, lo que hace que la pila de protocolos TCP/IP la entregue siempre al servidor adecuado.

I.4.5. Nombres de servicios

Para que un cliente acceda a un servicio, es necesario que el cliente conozca la dirección IP y el puerto por el cual ese servidor está prestando el servicio; ejemplo, un servidor HTTP presta el servicio a través del puerto 80. Sin la dirección IP y el puerto no se podría establecer una conexión TCP, y tampoco se podrían recibir los mensajes UDP. Para resolver esto, se pueden utilizar varias técnicas:

- Enterarse por algún medio ajeno a la red. El número de puerto del servidor puede ser un parámetro facilitado al cliente.
- Utilizar un número establecido de forma general. Esto es uno de los medios más utilizados, especialmente en el caso de aplicaciones estándar. En Internet, los puertos de las aplicaciones estándar se llaman puertos bien conocidos; ejemplo, FTP: puerto 21, Telnet: puerto 23, HTTP: puerto 80, DNS: puerto 23, TFTP: puerto 69, etc. Actualmente, los puertos bien conocidos están reservados hasta el 5000, por lo que no puedes ser utilizados para procesos corrientes. Todos los servidores en Internet prestan servicio a través de los puertos bien conocidos: todos los servidores HTTP en Internet prestan servicio a través del puerto 80.
- Utilizar un archivo para traducir entre un nombre de servicio y el puerto asociado al mismo. Es el principio que se implementa, pues para una persona es más fácil aprenderse una serie de nombres o etiquetas que aprenderse una serie de números.

Si se utiliza la última opción, el archivo `/etc/services` contiene listas de servidores y puertos con el siguiente formato:

```
nombreServicio númeroPuerto/protocolo nombreAlternativo1 nombreAlternativo2...
```

`nombreServicio`: es único dentro del archivo `/etc/services`; no se permiten nombres de servicio duplicados. Ha de coincidir con el nombre de servicio especificado en el archivo `/etc/inetd.conf`.

`númeroPuerto`: es el número del puerto en el que el daemon **inetd** está a la escucha y a través del cual se conecta el daemon **inetd**. El número del puerto puede estar en el rango de 1 a 65535 (2^{16} = 65536) y debe ser exclusivo.

`protocolo`: puede ser tcp o udp.

Un ejemplo del archivo `/etc/services`:

```
tcpmux          1/tcp           # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp          quote
msp             18/tcp          # message send protocol
msp             18/udp
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp-data        20/tcp
ftp             21/tcp
```

Este archivo se puede consultar usando funciones especiales, como `getservbyname()` y `getservbyport()`:

```
#include <netdb.h>

struct servent *getservbyname(const char *name,
const char *proto);
struct servent *getservbyport(int port, const char *proto);
```

La primera de estas funciones obtiene un resultado del tipo `struct servent` a partir de un nombre de servicio y un protocolo (tcp o udp). La segunda hace lo mismo, pero toma como entrada un número de puerto. La estructura `serevent` tiene la siguiente forma:

```
struct servent {
    char  *s_name;       /* official service name */
    char **s_aliases;    /* alias list */
int    s_port;       /* port number */
    char  *s_proto;      /* protocol to use */
}
```

Los campos de la estructura `serevent` devueltos por la funciones anteriores son:

`*s_name`: Nombre oficial del servicio
`**s_aliases`: Una lista de nombres alternativos
`s_port`: El número de puerto del servidor
`*s_proto`: El nombre del protocolo para dar el servicio

El que el nombre de un servicio se encuentre en el archivo `/etc/services` de una terminal, no indica que el servicio esté disponible. Para dar el servicio se necesita de un proceso servidor escuchando en el puerto correspondiente.

La elección del puerto en el que se va a colocar un servidor no debe ser arbitraria. Existe una serie de puertos reservados que solo se pueden usar por el administrador del sistema (root). La constante `IPPORT_RESERVED` indica la frontera entre los puertos bien conocidos y los accesibles para los usuarios normales.

I.4.6. Comunicación cliente/servidor no orientado a conexión

Cuando se habla de comunicación no orientada a conexión se entiende que el protocolo que se implementa en la familia AF_INET es UDP, el cual envía datos en forma de datagramas UDP. Por lo que no hay garantía en la entrega de los mensajes que se envían, ni tampoco hay garantía de una entrega en secuencia.

Lo único que se tiene garantizado es que el datagrama UDP llegara sin ningún error y que la aplicación sea identificada.

La secuencia de llamadas para una comunicación no orientada a conexión se muestra en la figura I.4.1; como se puede observar, la llamada de las funciones es la misma tanto en el cliente como en el servidor, la diferencia está en que el servidor inicia primero la escucha por la petición de servicio y ya que atendió al cliente regresa a ponerse en espera para atender otra petición de servicio. En el caso del cliente cuando recibe la respuesta a su petición, cierra el canal de comunicación.

I.4.7. Comunicación cliente/servidor orientado a conexión

En la comunicación orientada a conexión el tipo de socket que se emplea es el `SOCK_STREAM` y el protocolo que se implementa es TCP, el cual proporciona a los procesos de usuario un canal fiable y full-dúplex. El protocolo TCP asegurara que los segmentos lleguen en secuencia, sin errores y con identificación de la aplicación.

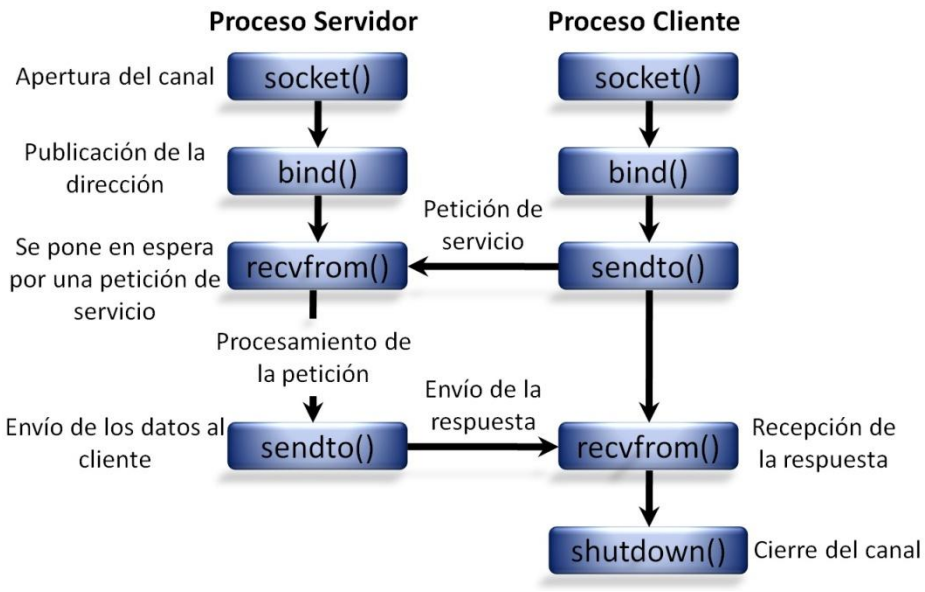


Figura I.4.1. Secuencia de llamadas para una comunicación no orientada a conexión.

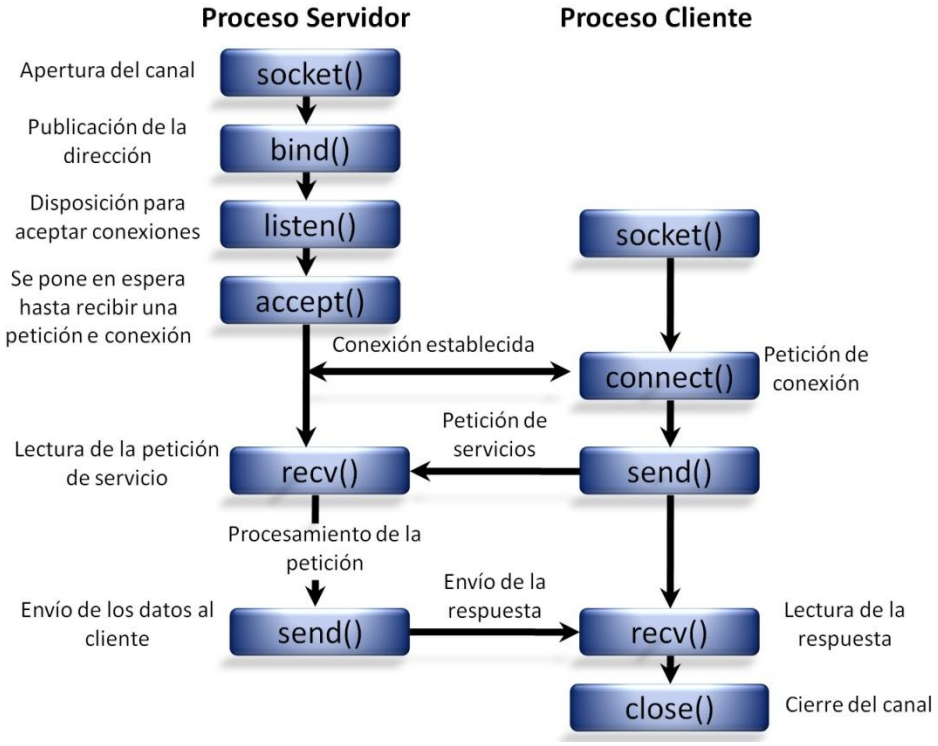


Figura I.4.2. Secuencia de llamadas para una comunicación orientada a conexión .

La secuencia de llamadas para una comunicación orientada a conexión se muestra en la figura I.4.2; aquí el servidor realiza las llamadas para abrir el canal de comunicación, vincula el socket con la dirección IP y el puerto con el que va a dar servicio, anuncia su disposición para aceptar conexiones y se pone en escucha para recibir las conexiones de los clientes. En el caso del cliente, este abre su canal de comunicación y establece conexión con la función `connect()`.

Todas estas llamadas para las comunicaciones orientadas a conexión y no orientadas a conexión fueron tratadas en el apartado I.2.

Ejercicio I.4.1. Elabore un programa cliente y un programa servidor con sockets no orientados a conexión para transmitir un mensaje cualquiera del cliente al servidor; el servidor deberá enviar una cadena "listo" de confirmación.

Ejercicio I.4.2. Elabore un programa cliente y un programa servidor con sockets orientados a conexión para transmitir un archivo en modo binario; el servidor deberá solicitar y validar un login y un password para poder iniciar la transferencia del archivo.