User Datagram Protocol

User Datagram Protocol (**UDP**) es un protocolo del nivel de transporte basado en el intercambio de datagramas (Encapsulado de capa 4 Modelo OSI). Permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión, ya que el propio datagrama incorpora suficiente información de direccionamiento en su cabecera. Tampoco tiene confirmación ni control de flujo, por lo que los paquetes pueden adelantarse unos a otros; y tampoco se sabe si ha llegado correctamente, ya que no hay confirmación de entrega o recepción. Su uso principal es para protocolos como DHCP, BOOTP, DNS y demás protocolos en los que el intercambio de paquetes de la conexión/desconexión son mayores, o no son rentables con respecto a la información transmitida, así como para la transmisión de audio y vídeo en real, donde no es posible realizar retransmisiones por los estrictos requisitos de retardo que se tiene en estos casos.

1 Descripción técnica

User Datagram Protocol (UDP) es un protocolo mínimo de nivel de transporte orientado a mensajes documentado en el RFC 768 de la IETF.

En la familia de protocolos de Internet UDP proporciona una sencilla interfaz entre la capa de red y la capa de aplicación. UDP no otorga garantías para la entrega de sus mensajes (por lo que realmente no se debería encontrar en la capa 4) y el origen UDP no retiene estados de los mensajes UDP que han sido enviados a la red. UDP sólo añade multiplexado de aplicación y suma de verificación de la cabecera y la carga útil. Cualquier tipo de garantías para la transmisión de la información deben ser implementadas en capas superiores.

La cabecera UDP consta de 4 campos de los cuales 2 son opcionales (con fondo rojo en la tabla). Los campos de los puertos fuente y destino son campos de 16 bits que identifican el proceso de origen y recepción. Ya que UDP carece de un servidor de estado y el origen UDP no solicita respuestas, el puerto origen es opcional. En caso de no ser utilizado, el puerto origen debe ser puesto a cero. A los campos del puerto destino le sigue un campo obligatorio que indica el tamaño en bytes del datagrama UDP incluidos los datos. El valor mínimo es de 8 bytes. El campo de la cabecera restante es una suma de comprobación de 16 bits que abarca una pseudo-cabecera IP (con las IP origen y destino, el protocolo y la longitud del paquete UDP), la cabecera UDP, los datos y 0's hasta completar un múltiplo de 16. El checksum también es opcional en IPv4, aunque generalmente se utiliza en la práctica (en IPv6 su uso es obligatorio). A continuación se muestra los campos para el cálculo del checksum en IPv4, marcada en rojo la pseudo-cabecera IP.

El protocolo UDP se utiliza por ejemplo cuando se necesita transmitir voz o vídeo y resulta más importante transmitir con velocidad que garantizar el hecho de que lleguen absolutamente todos los bytes.

2 Puertos

UDP utiliza puertos para permitir la comunicación entre aplicaciones. El campo de puerto tiene una longitud de 16 bits, por lo que el rango de valores válidos va de 0 a 65.535. El puerto 0 está reservado, pero es un valor permitido como puerto origen si el proceso emisor no espera recibir mensajes como respuesta.

Los puertos 1 a 1023 se llaman puertos "bien conocidos" y en sistemas operativos tipo Unix enlazar con uno de estos puertos requiere acceso como superusuario.

Los puertos 1024 a 49.151 son puertos registrados.

Los puertos 49.152 a 65.535 son puertos efímeros y son utilizados como puertos temporales, sobre todo por los clientes al comunicarse con los servidores.

3 Uso en aplicaciones

La mayoría de las aplicaciones claves de Internet utilizan el protocolo UDP, incluyendo: el Sistema de Nombres de Dominio , donde las consultas deben ser rápidas y solo contaran de una sola solicitud, luego de un paquete único de respuesta, el Protocolo de Administración de Red, el Protocolo de Información de Enrutamiento (DEP) y el Protocolo de Configuración dinámica de host.

4 Principales características

Las características principales de este protocolo son:

1. Trabaja sin conexión, es decir que no emplea ninguna sincronización entre el origen y el destino.

2. Trabaja con paquetes o datagramas enteros, no con bytes individuales como TCP. Una aplicación que emplea

el protocolo UDP intercambia información en forma de bloques de bytes, de forma que por cada bloque de bytes enviado de la capa de aplicación a la capa de transporte, se envía un paquete UDP.

3. No es fiable. No emplea control del flujo ni ordena los paquetes.

4. Su gran ventaja es que provoca poca carga adicional en la red ya que es sencillo y emplea cabeceras muy simples.

5 Código de ejemplo (Java)

El siguiente ejemplo muestra cómo usar el protocolo UDP para una comunicación cliente/servidor:

Servidor:

public static void main(String[] args) { try { System.out.println(“server creado.....”); // 1. crear el servidor.. DatagramSocket socket = new DatagramSocket(45000); // 2. recibir mensaje desde el cliente... // 2.1 crear el paquete donde se recibe el mensaje. byte[] buffer = new byte[1024]; DatagramPacket paqueteCliente = new DatagramPacket(buffer, 1024); // 2.2 recibir el paquete. operacion bloqueante. System.out.println(“socket esperando....”); socket.receive(paqueteCliente); // 2.3 leer el paquete como string... String msj = new String(paqueteCliente.getData()); System.out.println(“desde " + paqueteCliente.getAddress().getHostAddress() + " desde el puerto " + paqueteCliente.getPort() + " se recibio:" + msj); // 3. enviar respuesta.. String resp = new Date().toString();// la hora como respuesta. // 3.1 crear datagrama de envio. // direccion destino.. InetAddress addr = paqueteCliente.getAddress();// la misma del // cliente. int port = paqueteCliente.getPort(); // el datagrama contiene la información del destino. DatagramPacket paqueteEnvio = new DatagramPacket(resp.getBytes(), resp.length(), addr, port); System.out.println(“enviando:"+new String(paqueteEnvio.getData())); // 3.2 enviar paquete... socket.send(paqueteEnvio); //4. cerrar el socket... socket.close(); } catch (IOException e) { // TODO Auto-generated catch block e.printStackTrace(); } }

Cliente:

public static void main(String[] args) { try { // 1. crear el socket por donde se enviara la peticion y se recibira // la respuesta.. DatagramSocket socket = new DatagramSocket(32000); // 2. crear datagrama para enviar la info. el datagrama contiene // toda la info necesaria para que llegue el msj String msj = “Hola Server.....”; // msj a enviar. String ip = “127.0.0.1"; int port = 45000; // 2.1 crear datagrama DatagramPacket paqueteEnvio = new DatagramPacket(msj.getBytes(), msj.length(), InetAddress.getByName(ip), port); // 2.2 enviar paquete. socket.send(paqueteEnvio); // 3. recibir respuesta... // 3.1 crear datagrama de recepcion. byte[] resp = new byte[1024]; DatagramPacket paqueteRecibido = new DatagramPacket(resp, resp.length); // 3.2 recibir paquete. socket.receive(paqueteRecibido); // 4. mostrar info... System.out.println(“Server respondio desde " + paqueteRecibido.getAddress().getHostAddress() + " por el puerto " + paqueteRecibido.getPort() + " se recibio:" + new String(paqueteRecibido.getData())); // 5. cerrar socket.close(); } catch (IOException e) { e.printStackTrace(); } }

6 Código de ejemplo (Python 3.x)

El siguiente ejemplo muestra cómo usar el protocolo UDP para una comunicación cliente/servidor:

Servidor:

import socketserver print(“Servidor iniciado...”) class MyUDPHandler(socketserver.BaseRequestHandler): def handle(self): data = self.request[0].strip() socket = self.request[1] print("{0} Ha escrito:".format(self.client_address[0])) print(data) socket.sendto(data.upper(), self.client_address) if __name__ == "__main__": HOST, PORT = “localhost”, 9999 server = socketserver.UDPServer((HOST, PORT), MyUDPHandler) server.serve_forever()

Cliente (Cambia *“localhost”* por la dirección IP del servidor.):

import socket import sys print(“Cliente iniciado...”) HOST, PORT = “localhost”, 9999 data = " ".join(sys.argv[1:]) sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) sock.sendto(bytes(data + "\n","utf8”), (HOST, PORT)) received = sock.recv(1024) print(“Enviado: {0}".format(data)) print(“Recibido: {0}".format(received))

7 Código de ejemplo (C)

El siguiente ejemplo muestra cómo usar el protocolo UDP para una comunicación cliente/servidor:

Servidor:

#include <winsock.h> #include <stdio.h> #pragma comment(lib,"ws2_32.lib”) const int BufLen = 1024; int main() { WSADATA wsaData; SOCKET RecvSocket; sockaddr_in RecvAddr; int Puerto = 2345; char RecvBuf[BufLen]; sockaddr_in SenderAddr; int SenderAddrSize = sizeof(SenderAddr); WSAStartup(MAKEWORD(2,2), &wsaData); RecvSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP); RecvAddr.sin_family = AF_INET; RecvAddr.sin_port = htons(Puerto); RecvAddr.sin_addr.s_addr = INADDR_ANY;

bind(RecvSocket, (SOCKADDR *) &RecvAddr, sizeof(RecvAddr)); recvfrom(RecvSocket,RecvBuf, BufLen,0,(SOCKADDR *)&SenderAddr,&SenderAddrSize); printf("%s\n",RecvBuf); closesocket(RecvSocket); WSACleanup(); }

Cliente (Cambia "127.0.0.1" por la dirección IP del servidor):

#include <winsock.h> #pragma comment(lib,"ws2_32.lib") int main() { WSADATA wsaData; SOCKET SendSocket; sockaddr_in RecvAddr; int Puerto = 2345; char ip[] = "127.0.0.1"; char SendBuf[] = "Hola!!!!"; WSAStartup(MAKEWORD(2,2), &wsaData); SendSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP); RecvAddr.sin_family = AF_INET; RecvAddr.sin_port = htons(Puerto); RecvAddr.sin_addr.s_addr = inet_addr(ip); sendto(SendSocket,SendBuf,strlen(SendBuf)+1,0,(SOCKADDR *) &RecvAddr,sizeof(RecvAddr)); WSACleanup(); }

8 Comparativa entre UDP y TCP (*Transmission Control Protocol*)

- UDP: proporciona un nivel de *transporte no fiable de datagramas*, ya que apenas añade la información necesaria para la comunicación extremo a extremo al paquete que envía al nivel inferior. Lo utilizan aplicaciones como NFS (*Network File System*) y RCP (comando para copiar ficheros entre ordenadores remotos), pero sobre todo se emplea en tareas de control y en la transmisión de audio y vídeo a través de una red. No introduce retardos para establecer una conexión, no mantiene estado de conexión alguno y no realiza seguimiento de estos parámetros. Así, *un servidor dedicado a una aplicación particular puede soportar más clientes activos cuando la aplicación corre sobre UDP en lugar de sobre TCP.*

- TCP: es el protocolo que proporciona un *transporte fiable de flujo de bits entre aplicaciones*. Está pensado para poder enviar grandes cantidades de información de forma fiable, liberando al programador de la dificultad de gestionar la fiabilidad de la conexión (retransmisiones, pérdida de paquetes, orden en el que llegan los paquetes, duplicados de paquetes...) que gestiona el propio protocolo. Pero la complejidad de la gestión de la fiabilidad tiene un coste en eficiencia, ya que para llevar a cabo las gestiones anteriores se tiene que añadir bastante información a los paquetes que enviar. Debido a que los paquetes para enviar tienen un tamaño máximo, cuanta más información añada el protocolo para su gestión, menos información que proviene de la aplicación podrá contener ese paquete (el segmento TCP tiene una sobrecarga de 20 bytes en cada segmento, mientras que UDP solo añade 8 bytes). Por eso, cuando es más importante la velocidad que la fiabilidad, se utiliza UDP. En cambio, *TCP asegura la recepción en destino de la información para transmitir*.

9 Transmisión de vídeo y voz

UDP es generalmente el protocolo usado en la transmisión de vídeo y voz a través de una red. Esto es porque no hay tiempo para enviar de nuevo paquetes perdidos cuando se está escuchando a alguien o viendo un vídeo en tiempo real.

Ya que tanto TCP como UDP circulan por la misma red, en muchos casos ocurre que el aumento del tráfico UDP daña el correcto funcionamiento de las aplicaciones TCP. Por defecto, TCP pasa a un segundo lugar para dejar a los datos en tiempo real usar la mayor parte del ancho de banda. El problema es que ambos son importantes para la mayor parte de las aplicaciones, por lo que encontrar el equilibrio entre ambos es crucial.

Todo este tipo de protocolos son usados en telemática.

10 Véase también

- Lista de números de puerto
- TCP
- SCTP

11 Enlaces externos

- RFC-768
- RFC-768 en español
- IANA Port Assignments
- The Trouble with UDP Scanning (PDF)
- Breakdown of UDP frame
- UDP on MSDN Magazine Sockets and WCF

12 Origen del texto y las imágenes, colaboradores y licencias

12.1 Texto

- **User Datagram Protocol** *Fuente:* https://es.wikipedia.org/wiki/User_Datagram_Protocol?oldid=85786873 *Colaboradores:* Macar~eswiki, Centeno, Moriel, Sauron, Hashar, Zwobot, Aloriel, Rsg, Tano4595, Barcex, Enric Naval, Caos, JMPerez, Rembiapo pohyiete (bot), Magister Mathematicae, Orgullobot~eswiki, Toxickore, Yrbot, FlaBot, GermanX, Sasquatch21, KnightRider, JRGL, Pieraco, Juan Antonio Cordero, Maldoror, Siabef, CEM-bot, Thijs!bot, Dajuam, PabloCastellano, Sparkyman, Siluancar, Isha, JAnDbot, Jugones55, TXiKiBoT, Biasoli, AlnoktaBOT, Jotego, Matdrodes, Shooke, Barri, AlleborgoBot, Muro Bot, BotMultichill, SieBot, Masterdjinn, DaBot~eswiki, Loveless, Leonaro, Mutari, Farisori, Botellín, BodhisattvaBot, SilvonenBot, Khaessar ule, Camilo, Vara-ULE~eswiki, MastiBot, Andreasmperu, Nallimbot, Salva84, ArthurBot, Almabot, Xqbot, Jkbw, Rubinbot, Varelariel, Igna, Panderine!, RedBot, PatruBOT, KamikazeBot, Mr.Ajedrez, Nanyta27, Jorge c2010, GrouchoBot, EmausBot, Savh, Sergio Andres Segovia, Adadon, ChuispastonBot, Caferrerb, WikitanvirBot, MagnusA.Bot, IRedRat, Addbot, Balles2601, Adamantike, Rubenprot, DefaultCR y Anónimos: 77

12.2 Imágenes

12.3 Licencia del contenido