Informe Obligatorio 2

Redes de Computadoras - 2015

Grupo 29
Bruno Amaral - 4.485.399-6
Martín Steglich - 4.607.084-9
Analía Percovich - 4.702.590-8

Indice

1) Introducción

1. Objetivo del trabajo

Aplicar los conceptos teóricos de capas de aplicación y transporte, el desarrollo de un servicio simple de comunicación para una LAN.

1. Descripción del problema

Se desea implementar una aplicación simple de comunicación en LAN que permita el intercambio de mensajes entre múltiples usuarios.
La misma utilizará el protocolo UDP y nivel de confiabilidad RDT 3.0 (sin considerar mensajes corruptos) y así ofrecer un servicio de mensajería confiable, permitiendo el intercambio de mensajes entre todos los clientes conectados simultáneamente a través de un servidor.

2) Fundamento teórico

1. Capa de aplicacion

La capa de aplicación es donde residen las aplicaciones de red y sus protocolos. Algunos ejemplos de estos protocolos son HTTP, SMTP y FTP, entre otros. El protocolo de la capa de aplicación se distribuye entre múltiples sistemas finales, con la aplicación en uno de ellos utilizando el protocolo para intercambiar paquetes de información con la aplicación en el otro sistema final.

2. Capa de transporte

Se encarga de aceptar los datos provenientes de la capa de aplicación, dividirlo en unidades más pequeñas (en caso de ser necesario), asegurarse que todos los datos enviados lleguen correctamente de extremo a extremo, determina el tipo de conexión (seguro orientado a conexión TCP, inseguro no orientado a conexión UDP).

3. UDP

User Datagram Protocol (UDP) es un protocolo del nivel de transporte basado en el intercambio de datagramas. 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.

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

y el origen no retiene estados de los mensajes 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. 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.

1. RDT 3.0

Este protocolo pertenece a la capa de transporte orientado a la conexión que brinda un servicio de transferencia confiable de datos con la excepción de que no realiza chequeo de correctitud de datos recibidos. El mismo trabaja sobre el protocolo IP, que provee un servicio de datagramas no confiable.

Para iniciar una transferencia a través del protocolo RDT se debe establecer conexión entre un extremo activo que inicia una conexión y un extremo pasivo que espera conexión, siendo el extremo activo siempre el solicitante del cierre de la conexión. Para cada sesión se utilizarán buffers de almacenamiento auxiliar, que guardarán datos a enviar o recibidos por RDT, hasta que la aplicación lo solicite, devolviendo el control a la aplicación mientras ésta no solicite o no entregue más datos. Una sesión RDT se identifica con los siguientes 4 elementos: {IP origen, puerto RDT origen, IP destino, puerto RDT destino}.

RDT permite la comunicación de datos en forma unidireccional desde el extremo que inició la conexión hacia el otro extremo, siendo responsabilidad del protocolo garantizar la entrega de los datos que se hayan perdido por cualquier motivo, ya sean fallas en la red o saturación de los buffers de envío y/o recepción del RDT.

2. Sockets y Threads

El socket es una interfaz que opera entre la capa de aplicación y la capa de transporte dentro de un host, por la cual un proceso recibe y envía mensajes desde y hacia la red. También se conoce como Interfaz de programación de aplicaciones (API Application programming interface) que funciona entre la aplicación y la red, ya que el socket es la interfaz de programación con la que se construyen las aplicaciones de red.

Un thread o hilo de ejecución es una característica que permite a una aplicación realizar varias tareas concurrentemente. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente. Los hilos de ejecución que comparten los mismos recursos, sumados a estos recursos, son en conjunto conocidos como un proceso.

Lo que es propio de cada hilo es el contador de programa, la pila de ejecución y el estado de la CPU (incluyendo el valor de los registros).

La técnica de Multithreading permite que varios threads de ejecución corran simultáneamente dentro del mismo programa sin interferir entre sí

3. Multicast

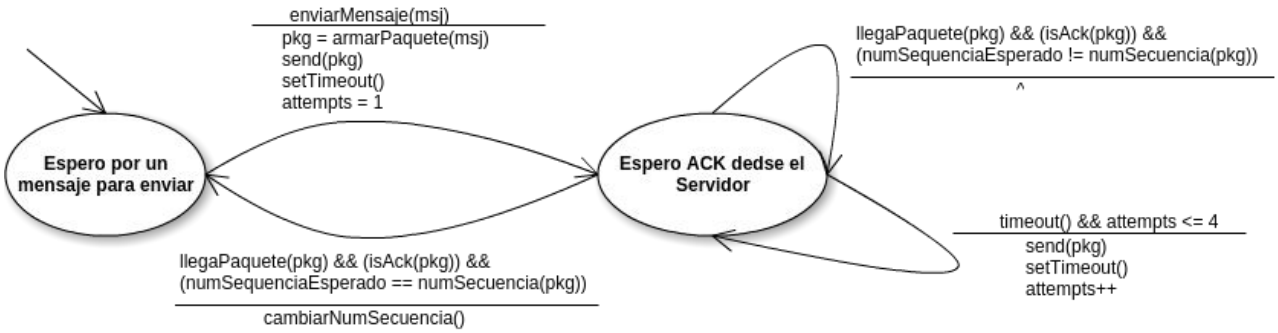
Multicast es un método de envío de paquetes (a nivel de IP) a múltiples destinos simultáneamente, que tan sólo serán recibidos por un determinado grupo de hosts. Para que el equipo reciba paquetes, antes deben de haberse “suscrito” a ese grupo.

3) Descripción de la solución

1. Descripción del protocolo

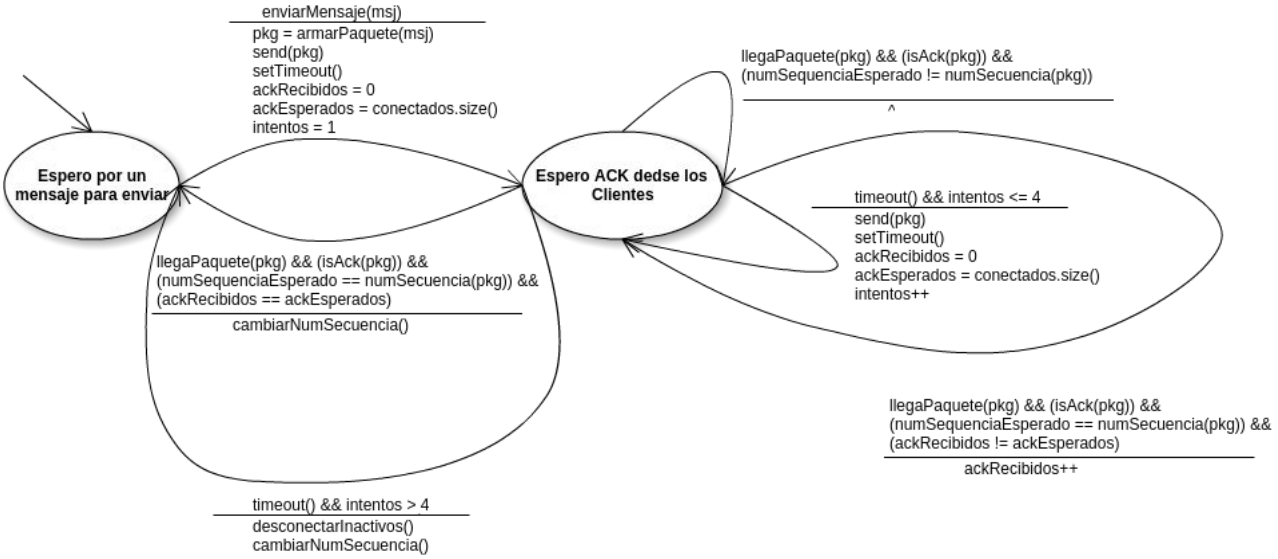
A continuación pasaremos a explicar el funcionamiento del protocolo de comunicación utilizado en la solución.

Lado Emisor del Cliente



El Cliente espera por un mensaje para enviar. Cuando llega un mensaje para enviar desde la capa de aplicación, crea el paquete a enviar, lo envía a través de UDP, setea un timeout e inicializa la variable attempts en 1 (cantidad de envíos).
Luego de enviar el mensaje se queda esperando por el mensaje de reconocimiento del Servidor. Si, mientras espera por el ACK desde el Servidor, recibe un ACK con un número de secuencia distinto al que espera, descarta el ACK recibido. En cambio, si los números de secuencia coinciden, actualiza el número de secuencia y espera por otro mensaje para enviar.
Si mientras se encuentra esperando por el ACK del Servidor, ocurre un fin de temporización, el Cliente reenvía el paquete, setea nuevamente el timeout y actualiza la variable attempts.

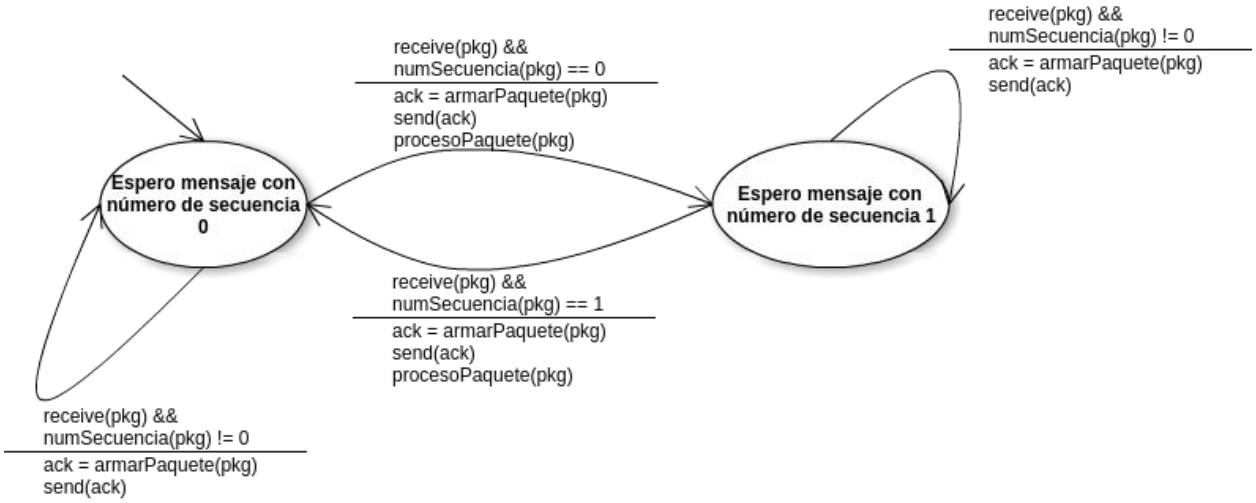
Lado Emisor del Servidor



El Servidor espera por un mensaje para enviar, Cuando llega un mensaje para enviar desde la capa de aplicación, crea el paquete, lo envía a través de UDP hacia el grupo de multicast, setea un timeout, inicializa la variable intentos en 1, setea la variable ackRecibidos en 0 y obtiene la cantidad de Clientes conectados en ese momento.
Luego de enviar el mensaje, se queda esperando por el mensaje de reconocimiento de cada uno de los clientes conectados.

Si, mientras espera por un ACK desde algún cliente, recibe un ACK con un número de secuencia distinto al que espera, descarta el ACK recibido. Si, al recibir el ACK, los números de secuencia coinciden pero aún no se han recibido todos los mensajes de reconocimiento de los clientes, vuelve a esperar por otro ACK y aumenta en uno la cantidad de ACK recibidos. Si, en cambio, ya se recibieron todos los mensajes de reconocimiento, se procede a cambiar el número de secuencia esperado y a esperar por un nuevo mensaje a enviar.
Si, en el momento de esperar un ACK, ocurre un fin de temporización y la cantidad de intentos es menor a 4, se procede a reenviar el paquete, obtener la cantidad de conectados, actualizar la variable intentos y setear nuevamente la variable ackRecibidos en 0. En cambio, si la cantidad de intentos es mayor a 4, se desconecta a todos los clientes que no hayan respondido con ACK y se procede a cambiar el número de secuencia.

Lado Receptor



Se espera por un mensaje con número de secuencia 0. Si se recibe un paquete con un número de secuencia diferente a 0, se envía un mensaje de reconocimiento con número de secuencia 0, se descarta el mensaje recibido y se vuelve a esperar por un mensaje con dicho número de secuencia. Si se recibe un paquete con número de secuencia 0, se envía un mensaje de reconocimiento para dicho mensaje y se procesa el mensaje recibido y se procede a esperar por un mensaje con número de secuencia 1.
Mientras se espera por un mensaje con número de secuencia 1, se realizan las mismas acciones que mientras se espera por un mensaje con número de secuencia 0.
El lado receptor tiene un comportamiento similar tanto en el Cliente como el Servidor.

2. Toma de decisiones

- Los paquetes enviados por nuestra aplicación tienen el siguiente formato:

<head>*hostOrigen*|*puertoOrigen*|*hostDestino*|*puertoDestino*|*numSeq*|*isAck*|*serverSeq****</head><data>****datos****</data>***

  - hostOrigen y puertoOrigen: Indican host y puerto del emisor del paquete.
  - hostDestino y puertoDestino: Indican host y puerto del receptor del paquete. Cuando el paquete debe ser recibido por todos los clientes, el valor de hostDestino es 0.0.0.0 y el de puertoDestino es 0.
  - numSeq: Número de secuencia del paquete enviado. Los posibles valores son 0 y 1.
  - isACK: Indica si el paquete enviado corresponde a un mensaje de reconocimiento. El valor 1 indica que es un mensaje de reconocimiento, el valor 0 indica que no.

- serverSeq: Número de secuencia esperado por el Servidor. Se utiliza para setear en el Cliente, en el momento del login, el número de secuencia actual del Servidor. En los mensajes enviados por el Servidor, este número coincide con el campo numSeq.
- datos: Mensaje de la capa de aplicación.

- Definimos el máximo tamaño de los mensajes (incluyendo el cabezal) en 512 bytes.
- Decidimos que 3 ACKs no validados por un cliente son suficientes para desconectarlo.
- Definimos un tiempo máximo de 0.9 segundos para esperar un ACK del Cliente.
- Implementación de la probabilidad de pérdidas. En el Cliente definimos una función que, utilizando los números aleatorios de java, indica si ocurrió una pérdida o no según la probabilidad de pérdidas definida. Por defecto dicha probabilidad es 0 y se puede cambiar en cualquier momento entrando a la opción específica para eso desde su interfaz.

3. Pseudocódigo

Lado Emisor Cliente:

```
Si Cliente conectado ->
        ObtengoPrimerMensajeAEnviar
        CreoPaquete
        AbroSocketUDP
        EnvioPaquete
        Mientras noReciboACK || intentos < 3 ->
                esperoACK
                Si llegaMensaje y esACK ->
                        Si numSecuencaRecibido == numSecuenciaEsperado ->
                                cambioNumeroSecuencia
                        Fin SI
                Sino ->
                        Si ocurreTimeOut ->
                                reenvioPaquete
                        Fin Si
                Fin SI
        Fin Mientras
        CierroSocketUDP
        BorroPrimerMensajeAEnviar
        Si noRecibiACK ->
                muestroMensajeError
        Fin SI
Fin Si
```

Lado Receptor Cliente:

```
Si Cliente conectado ->
        CreoSocketMulticast
        meUnoAlGrupoMulticast
        esperoMensaje

        CreoACk
        AbroSocketUDP
        EnvioACK
        CierroSocketUDP

        Si numSecuenciaRecibido == numSecuenciaEsperado ->
                ProcesoEncabezadoMensaje
                Si MensajeEsParaMi ->
                        ProcesoMensaje
                Sino
                        DescartoMensaje
                Fin Si
                cambioNumeroSecuencia
        Fin Si
        CierroSocketMulticast
Fin SI
```

Lado Emisor Servidor:

```
CreoMensajeAEnviar
CreoSocketUDP
EnvioMensajeMulticast
ACKRecibidos = 0
CreoSocketUDPreceptor
Mientras NoReciboTodosACK && intentos < 4 ->
        esperoACK
        Si llegaMensaje y esACK ->
                Si numSecuencaRecibido == numSecuenciaEsperado ->
                        ACKRecibidos++
                Fin SI
        Sino ->
                Si ocurreTimeOut ->
                        reenvioPaquete
                        ACKRecibidos = 0
                        intentos ++
                Fin Si
        Fin SI
Fin Mientras
Si intentos == 4 ->
        DesconectoClientesInactivos

ActualizoNumeroSecuencia
CierroSocketsUDP
```

Lado Receptor Servidor:

```
Mientras servidorVivo ->
        CreoSocket
        esperoMensaje

        Si origenEsCliente ->
                CreoACK
                AbroSocketUDP
                EnvioACK
                CierroSocketUDP

                Si numSecuenciaRecibido == numSecuenciEsperado ->
                        ProcesoEncabezadoMensaje
                        ProcesoMensaje
                        cambioNumeroSecuencia
                Fin Si
        Fin Si
        CierroSocket
Fin Mientras
```

3. Instrucciones para compilación y ejecución

Compilacion y ejecucion del Cliente:
1- Entrar a la carpeta Cliente y ejecutar el comando make.
2- Ejecutar java main.

Compilacion y ejecucion del Servidor:
1- Entrar a la carpeta Servidor y ejecutar el comando make.
2- Ejecutar ./Servidor