TP2: Conexiones

Teoría de las Comunicaciones

Departamento de Computación
FCEN - UBA

23.10.2013

1. Introducción

En este trabajo práctico ejercitaremos las nociones del nivel de transporte estudiadas en la materia a través de la implementación y análisis de un protocolo sencillo. En primera instancia, daremos una especificación de este protocolo en la sección 3.1. Luego presentaremos las tareas a desarrollar en la sección 3.2. Por último, la sección 3.3 describe el material que la cátedra pondrá a disposición como punto de partida para el desarrollo del trabajo.

2. Normativa

- Fecha de entrega (por mail): hasta el miércoles 10 de diciembre.
- El código junto con el informe deberá haber sido enviado por correo antes de esa fecha con el siguiente formato:
  to: tdc-doc at dc uba ar
  subject: debe tener el prefijo [tdc-conexiones]
  body: nombres de los integrantes y las respectivas direcciones de correo electrónico
  attachment: el código fuente desarrollado
- Se deberá entregar el informe impreso y abrochado con la lista de integrantes y sus correos electrónicos (los mismos que fueran enviados por mail).

El día estipulado como taller de prueba del TP2 (martes 12 de noviembre), cada grupo deberá acudir a los laboratorios de la facultad con el trabajo realizado a los efectos de validar el desarrollo con los docentes. Además, se dará la posibilidad de llevar a cabo las pruebas experimentales en ese mismo momento. El informe y los resultados del análisis podrán entregarse durante los días siguientes.

3. Enunciado

El trabajo consiste, esencialmente, en implementar el cliente de un protocolo de transporte simplex y luego analizar la eficiencia y el desempeño del protocolo en el contexto de una red local. A continuación veremos de qué trata este protocolo.

3.1. Especificación

El protocolo que estudiaremos, **PTC** (*Protocolo de Teoría de las Comunicaciones*), puede ubicarse dentro de la capa de transporte del modelo OSI tradicional. Fue concebido como un protocolo de exclusivo uso didáctico que permita lidiar en forma directa con algunas de las problemáticas usuales de la capa de transporte: establecimiento y liberación de conexión, control de errores y control de flujo.

3.1.1. Características básicas

Desde un punto de vista general, **PTC** presenta las siguientes características:

- **Unidireccionalidad**: se trata de un protocolo simplex en el que un cliente activo envía datos y un servidor pasivo los recibe, sin la posibilidad de enviar sus propios datos (a diferencia de TCP).
- **Orientación a conexión**: contempla procesos formales de establecimiento y liberación de conexón. Esta conexión permite que tanto el cliente como el servidor generen estado para lograr luego una comunicación efectiva.
- **Confiabilidad**: a través de un algoritmo de ventana deslizante, garantiza que los datos enviados por el cliente lleguen correctamente a destino.

En lo que sigue daremos un panorama más detallado de estas características, aunque en primer lugar mostraremos cómo es el formato de los paquetes.

3.1.2. Formato del segmento

El encabezado es de tamaño fijo: 12 bytes. Sólo exhibe campos para representar los puertos de origen y destino, para secuenciar y reconocer los paquetes y para indicar el propósito del paquete (i.e., flags). La imagen 1 ilustra cómo se disponen estos campos.

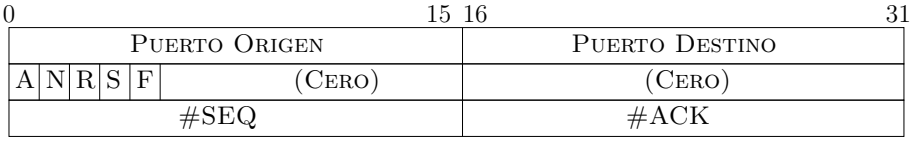

Figura 1: formato del encabezado de **PTC**

Los campos A, N, R, S y F, todos de 1 bit, representan los flags:

- A es el flag de ACK; prendido sólo en los mensajes del servidor para reconocer la correcta recepción de un paquete.
- S es el flag de SYN; es enviado sólo por el cliente al requerir establecer una nueva conexión.
- F es el flag de FIN; es enviado únicamente por el cliente cuando desea finalizar una conexión existente.
- Los flags N y R actualmente no tienen uso.

Inmediatamente después de este encabezado siguen los datos, de longitud arbitraria aunque entre 1 y 1000 bytes. El número de secuencia no trabaja a nivel de byte como ocurre en TCP sino que identifica a toda la porción de los datos contenida en el paquete.

Así como un segmento TCP se encapsula dentro de un datagrama IP, los paquetes de **PTC** también viajarán dentro de IP. Para que los hosts puedan reconocer este hecho, el campo `proto` del header IP debe definirise con valor 202, tradicionalmente sin uso (el protocolo TCP se identifica con el valor 6).

3.1.3. Establecimiento y liberación de conexión

Antes de poder enviar sus datos, el cliente debe establecer activamente una conexión con el servidor deseado. Para ello, debe enviar un segmento con el flag SYN prendido y con un valor arbitrario en el campo #SEQ que indique el número de secuencia inicial que utilizará el cliente para identificar sus paquetes. El resto de los campos pueden tener un valor arbitrario; no serán tenidos en cuenta por el servidor.

Por su parte, el servidor responderá con un segmento con el flag ACK prendido y el campo #ACK reconociendo el valor de #SEQ previamente recibido. Una vez hecho esto, el servidor considerará que la conexión está establecida. Lo mismo hará el cliente al recibir este reconocimiento. En caso de no llegar, el

cliente eventualmente retransmitirá el paquete inicial (más detalles sobre el mecanismo de retransmisión en la sección 3.1.4). Observar que no se requiere un SYN por parte del sevidor: al no enviar datos, no es necesario que éste sincronice su número de secuencia con el cliente.

Para cerrar la conexión, la secuencia a seguir es similar, aunque cambiando el flag de SYN por el flag de FIN. El segmento FIN enviado por el cliente también debe ir secuenciado como cualquier otro segmento de datos previamente enviado. En caso de no recibir el ACK respectivo, el cliente también retransmitirá el FIN.

3.1.4. Ventana deslizante y retransmisiones

Para garantizar la confiabilidad, **PTC** implementa ventana deslizante, particularmente Go-Back-N [1]. De esta manera, el servidor tiene una ventana de recepción de tamaño 1, mientras que el cliente tiene una ventana de emisión de tamaño que por simplicidad se fija en un valor constante arbitrario (potencialmente mayor que 1). Llamaremos `SEND_WINDOW` a este valor.

El cliente, entonces, podrá enviar hasta `SEND_WINDOW` paquetes en simultáneo antes de bloquearse esperando por los sucesivos reconocimientos. Cada ACK recibido permitirá desplazar la ventana y así hacer lugar para el envío de nuevos segmentos. Además, los ACKs del servidor pueden ser acumulativos (i.e., un paquete ACK puede reconocer más de un paquete con números de secuencia contiguos enviados por el cliente).

Al enviar un segmento con datos, el cliente también encolará este segmento en la cola de retransmisión. Éste permanecerá allí hasta ser eventualmente reconocido. Por otro lado, el cliente también define un tiempo máximo de espera `RETRANSMISSION_TIMEOUT` para esperar por estos reconocimientos. De superarse este tiempo, se asumirá que el paquete se extravió en los rincones de la red y por ende será retransmitido junto con todo segmento posterior aguardando en la cola de retransmisión.

Se define también un número máximo admisible de retranmisiones, `MAX_RETRANSMISSION_ATTEMPTS`. Si algún segmento del cliente debiera ser retransmitido más veces que esta cantidad, se debe asumir que la conexión se perdió, y se pasará a cerrarla sin enviar FIN.

3.1.5. Estados

En lo que sigue veremos los posibles estados que pueden atrevesar tanto el cliente como el servidor:

Estado	Descripción	Estado anterior
`CLOSED`	Representa la ausencia de conexión	`FIN_SENT` o ninguno
`SYN_SENT`	El cliente desea iniciar una conexión y ha enviado un SYN	`CLOSED`
`ESTABLISHED`	El cliente recibió el ACK del SYN y ya puede enviar datos	`SYN_SENT`
`FIN_SENT`	El cliente desea terminar la conexión y ha enviado un FIN al servidor	`ESTABLISHED`

Cuadro 1: estados del cliente

Estado	Descripción	Estado anterior
`CLOSED`	Representa la ausencia de conexión	`FIN_RECEIVED` o ninguno
`SYN_RECEIVED`	El servidor recibió un SYN y debe enviar el ACK respectivo	`CLOSED`
`ESTABLISHED`	El servidor ha enviado el ACK y está a la espera de recibir datos	`SYN_RECEIVED`
`FIN_RECEIVED`	El servidor recibió un FIN y debe enviar el ACK respectivo	`ESTABLISHED`

Cuadro 2: estados del servidor

3.2. Tareas a desarrollar

3.2.1. Primera parte: implementación

Tomando como punto de partida el código parcial suministrado por la cátedra (ver sección 3.3), se debe completar la implementación en Python del cliente del protocolo **PTC** . Puntualmente, se debe completar lo siguiente en el archivo `client.py`:

- El método `handle_incoming` de `PTCClientProtocol`, que es invocado cada vez que llega un paquete desde el servidor. El argumento `packet` indica el paquete recibido.
- El método `handle_timeout` de `PTCClientProtocol`, que es invocado siempre que el tiempo de espera del primer paquete encolado en la cola de retransmisión se agota.
- La lógica de la clase `ClientControlBlock`, que maneja las variables de la ventana deslizante del protocolo. Fundamentalmente, se debe poder verificar si es posible enviar (método `send_allowed`) y además definir métodos para determinar si un ACK es aceptado y para reajustar la ventana dado un ACK aceptado.

3.2.2. Segunda parte: experimentación

Utilizando el servidor provisto por la cátedra y la implementación completa del cliente del punto anterior, realizar las siguientes pruebas en el contexto de una red local (LAN):

- Enviar archivos de distinto tamaño (e.g., 1 KB, 5 KB, 10 KB, 50 KB, 100 KB, etc.) y medir el tiempo total de transmisión de los mismos. Tomar la medida representativa como el promedio de un número N de experimentos.
- Variar el tamaño de la ventana de emisión y repetir los envíos.
- Observar en Wireshark cómo impacta lo anterior en la cantidad de retransmisiones.

Para efectuar estas mediciones posiblemente se deba agregar código ad-hoc en `client.py`.

3.2.3. Tercera parte: análisis

A partir de los experimentos hechos, graficar lo siguiente y sacar conclusiones:

- Throughput percibido (i.e., cantidad de bits por segundo) en función del tamaño de archivo.
- Throughput en función de la ventana de emisión, para un tamaño de archivo fijo.
- Cantidad de retransmisiones en función de la ventana de emisión, para un tamaño de archivo fijo.

3.3. Material de la cátedra

La cátedra provee el siguiente material que deberá tomarse como punto de partida para desarrollar el trabajo:

- Código fuente en Python formado por:
  - Wrappers para cliente y servidor, ofreciendo una interfaz similar a la de los sockets tradicionales (`ptc.py`). Se deberá utilizar esta interfaz a la hora de realizar la experimentación.
  - Implementación parcial del cliente (`client.py`)
  - Implementación completa del servidor (`server.py`).
  - Constantes diversas del protocolo (`constants.py`).
  - Mapeo de bytes a paquetes de alto nivel y viceversa (`packet_parser.py` y `packet.py`) .
  - Abstracción del manejo de sockets (`soquete.py`).
  - Manejo de threads y eventos (`worker.py` y `event.py`). El cliente y el servidor internamente utilizan threads que se encargan de manejar los timeouts, el recibimiento de nuevos paquetes, etc.
  - Implementación de buffers de datos y de la cola de retransmisión (`buffers.py`).

- Plugin de Wireshark que contiene un disector del protocolo: esto permite la visualización detallada de los paquetes que enviará y recibirá el cliente. Para instalarlo, se debe copiar el archivo `.so` adecuado (según la arquitectura de la computadora a utilizar) al directorio de plugins globales de Wireshark. Éste puede conocerse accediendo al menú `Help` → `About Wireshark` → `Folders`.

  El disector fue compilado para versiones de Wireshark posteriores a la 1.6, y es altamente posible que no funcione en versiones previas. Por este motivo, sugerimos revisar la versión a utilizar y hacer la actualización si fuese necesario.

Referencias

[1] Tanenbaum, A. *Computer Networks*, 3ra Ed. Capítulo 3: págs. 207-213.