Universidad de Chile
Facultad de Cs. Físicas y Matemáticas
Departamento de Ciencias de la Computación
CC4303 Redes

Tarea 1

Profesor: José M. Piquer

Auxiliar: Catalina Álvarez

Objetivos

Esta tarea persigue construir una capa de transporte simple sobre UDP, usando una implementación base de Stop-and-wait a la que deben agregar números de secuencia. Para esto, se les entregan los fuentes en C de un cliente y una implementación de Stop-and-Wait correcta, sin números de secuencia alguno. Esta versión de Stop-and-Wait no funciona bien frente a delays ya que no puedo distinguir entre paquetes retrasados que dí por perdidos y verdaderos paquetes correctos, dando resultado a archivos con información duplicada o mal ordenada.

Para mejorar esto, se requiere incluir números de secuencias al protocolo. En nuestro caso, hemos decidido codificar los números de secuencia en 4 bytes, usando cada uno de ellos para representar un dígito de 0 a 9, dándonos un número de secuencia máximo 9999.

En esta tarea, se les pide implementar lo anterior y medir comportamiento de la solución en algunos escenarios adversos y compararse con TCP.

Aplicación

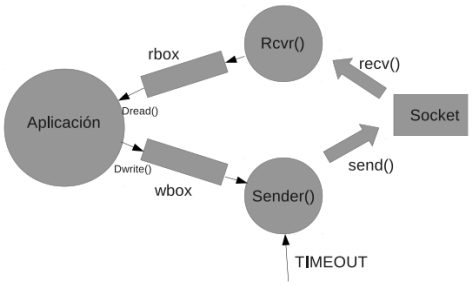
Es un cliente y un servidor que intercambian un archivo para medir ancho de banda en ambas direcciones. Se les provee el fuente del cliente. Ustedes sólo deben modificar el archivo que contiene la capa de datos (transporte).

Capa de datos provista

La capa datos original usa paquetes con un header de 2 bytes: el primero indica la conexión a la que pertenece (entre 0 y 255) y el segundo indica el tipo de paquete ('C','D' o 'A').

- **Número de conexión**: Es un identificador de conexión, entero de un byte entre 0 y 255.
- **Tipo de Paquete**:
    1. Conexión 'C': el cliente, inicialmente envía un paquete con sólo header, con tipo conexión y número de conexión 0 (byte en cero). El servidor responde con un paquete de tipo ACK y el número de conexión asignado (0 - 255). Este paquete se debe re-enviar 10 veces, esperando 3 segundos por el ACK, antes de abandonar.
    2. Datos 'D': es un paquete de datos normal. Se espera 1 segundo el ACK correspondiente y se re-intenta 10 veces antes de abandonar.
    3. ACK 'A': es un paquete de 2 bytes (sólo header), que indica qué paquete de la conexión fue recibido OK, lleva el número de secuencia al que corresponde.

La implementación de esta capa es un poco complicada, pues se necesita que opere en paralelo con la aplicación, recibiendo datos desde el socket y re-transmitiendo mientras la aplicación puede estar haciendo cualquier otra cosa. El diseñoo que sigue esta implementación se puede ver en este dibujo:



Capa de datos pedida

Se les pide modificar la capa datos anterior de modo de tener números de secuencia de 4 bytes.

Para esto, se requiere que se implemente lo siguiente:

- Incluir números de secuencia de 4 bytes, que codifican números hasta 9999.
- Cada paquete tiene un número de secuencia asignado, todos ellos consecutivos.

- Una vez que el paquete de número de secuencia N se envía, el enviador queda esperando la confirmación (ACK) con ese número de secuencia; cualquier otro se descarta.
- Cada vez que se recibe un paquete se debe verificar si su número de secuencia corresponde al esperado o se descarta.
- Una vez se recibe el ACK correcto (N), se envía el paquete con siguiente número de secuencia (N+1).
- En caso de llegar al último número de secuencia posible (9999), se da la vuelta y se comienza con 0 otra vez.

Ejecución

Para compilar la tarea, recomendamos hacer un Makefile siguiendo el ejemplo que se les entrega. Mantengan la separación de las funciones de comunicación de la aplicación para que puedan reusar la aplicación para las próximas tareas.

El Makefile, los fuentes entregados y los ejecutables incluidos son:

- **bwc-orig**: Es el cliente con Stop-and-Wait sin números de secuencia. El único archivo que ustedes deben modificar de aquí en adelante es Dataclient-seqn.c
- **bws-orig**: Es el servidor ejecutable del cliente original, sin números de secuencia.
- **bws**: Es el servidor ejecutable del nuevo cliente, con secuencias de 4 bytes.
- **bwc-tcp, bws-tcp**: Son cliente y servidor (vienen con código completo) donde la capa de datos se implementa simplemente con un socket TCP. La idea es que nuestro objetivo final es aproximarnos a los anchos de banda que ellos logran en las mismas condiciones.

Para probarlos, deben correr el siempre el servidor primero (con opción de debug):

```
% ./bws -d
```

y luego el cliente:

```
% ./bwc -d archivo-in archivo-out ::1
```

Se les proveen todos los fuentes para compilar el cliente:

```
% gcc bwc.c jsocket6.4.c Dataclient-seqn.c bufbox.c -o bwc-orig -lpthread -lrt
```

Pueden probar con los servidores ejecutables (bws) que vienen con los archivos, y usar localhost (::1 o 127.0.0.1).

Para probar con pérdidas, fuercen a localhost para que genere pérdidas aleatorias. En linux, usen "netem", usualmente basta hacer como superusuario (instalar paquete kernel-modules-extra):

```
% tc qdisc add dev lo root netem loss 20.0% delay 100ms
```

Y tienen 20 % de pérdida. Para modificar el valor, deben usar **change** en vez de add en ese mismo comando.

Una forma más simple de probar es darle las opciones de pérdida y delay a los servidores, quienes la implementan en su propia capa de datos. Pero esto no sirve para TCP (necesita que esto se implemente en el Sistema Operativo).

```
% ./bws -d -L 0.3 -D 0.5
```

Simula una pérdida con probabilidad 0.3 (30 %) y un delay de 0.5 segundos.

Cualquier duda o pregunta o reporte de bugs, dirigirse al foro de U-cursos.

Pruebas de eficiencia

Una vez terminada la tarea, se les pide comparar su eficiencia copiando el mismo archivo (pueden elegir cualquier archivo, pero se les recomienda de al menos decenas de kilobytes), con delay 0, 0.1 y 0.2, con pérdida de 0 y 10 %.

En los mismos escenarios, prueben la version TCP y reporten sus resultados.

Entrega

A través de U-cursos, **no se aceptan atrasos**. La evaluación de esta tarea es muy simple: el cliente debe funcionar bien con el servidor provisto.

Entreguen los fuentes y el makefile, o instrucciones, para compilar todo, y un LEEME.txt donde muestren los resultados de eficiencia que obtuvieron.