Tarea 2 Redes

Entrega: 9 de noviembre 2015

1. Objetivos

Esta tarea persigue construir una capa de transporte simple sobre UDP, que permita un protocolo confiable usando Go-Back-N y optimzado con timeouts adaptables (a la TCP), NACKs y secuencias de 2 bytes con ventanas más grandes.

Para esto, deben modificar la tarea1, primero los números de secuencia (header de 4 bytes), luego los NACKs y luego los timeouts.

Usaremos ventanas correderas más grandes en el protocolo y para eso necesito primero pasar a números de secuencia entre 0 y 65535 (dos chars sin signo), luego armar una ventana de transmisión de tamaño `WIN_SZ` y mantener el receptor con una ventana de un buffer. Luego implementar NACKs que apuran la retransmisión. Finalmente implementar timeouts inteligentes, que se van adaptando al RTT y su varianza.

En esta tarea, se les pide dar esos pasos y medir comportamiento de la solución en algunos escenarios adversos, buscando el tamaño de ventana óptimo para cada situación y compararse con TCP. Busque entender por qué usar ventanas demasiado grandes funciona mal, incluso sin errores inducidos (probar, por ejemplo, ventanas de tamaño 20.000 paquetes con delay pero sin errores).

2. 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. Uds sólo deben modificar el archivo que contiene la capa de datos (transporte).

3. Capa Datos pedida

Se les pide modificar la capa datos anterior de modo de tener números de secuencia entre de dos bytes, y que el enviador implemente un Go-Back-N simple, con ventana de envío de tamaño `WIN_SZ`, que será variable. El servidor ejecutable entregado viene pre-compilado con ventana de transmisión de 200 paquetes y eso no lo pueden cambiar (pero como es Go-Back-N no necesita ser igual a la de Uds).

Además deben implementar timeouts a la TCP como especificado acá y NACKs que deben ser enviados por Uds y entendidos por Uds si el servidor se los envía, siguiendo el protocolo especificado acá.

4. Implementación

Vayan implementando estas funcionalidades en orden, de modo de garantizar que el programa funciona bien en la primera parte antes de seguir a la siguiente. Es más

fácil de implementar y depurar así.

4.1. Go-Back-N

Se les pide implementar Go-Back-N con números de secuencia de dos bytes, y una ventana de tamaño fijo de `WIN_SZ` paquetes, y poder compilarla con distintos tamaños para probar eficiencia. Como siempre, implementando que un ACK para n implica ACK para todo $i \leq n$.

Para implementar esto, en un ambiente en que los números de secuencia se reutilizan, puede usar la función siguiente:

```
/* retorna TRUE si x está entre min y max en un rango circular */
int between(int x, int min, int max) {      /* min <= x < max */
    if(min <= max) return (min <= x && x < max);
    else           return (min <= x || x < max);
}
```

Para implementar secuencias en 2 bytes, portables en orden de red, pueden usar estas funciones auxiliares:

```
int get_seqn(unsigned char *p) { /* retorna DSEQ desde el buffer */
    uint16_t val;

    val = (*p << 8) + *(p+1);
    return ntohs(val);
}

void put_seqn(unsigned char *p, int seq) { /* copia seq a DSEQ en el buffer */
    uint16_t val;

    val = htons((uint16_t) seq);
    *p = val >> 8;
    *(p+1) = val & 0xff;
}
```

Como el servidor implementado genera NACKs, en la primera versión pueden simplemente interpretar un NACK para i como un ACK para $i - 1$ y no retransmitir anticipadamente, para probar que está todo bien.

4.2. NACK

Luego, implementen un Fast Retransmit con un NACK: normalmente el receptor, al recibir un paquete distinto al que está esperando, genera un ACK para el último recibido correcto (el anterior al esperado). En esta tarea, después de tres paquetes seguidos distintos al que está esperando, genera un NACK para el paquete que está esperando. El enviador, al recibir un NACK para i, hace como que eso implica un ACK para $i - 1$ y una petición para que generen una retransmisión anticipada de i y de la ventana a

parti de ahí. Basta con adelantar el timeout correspondiente a ese paquete para implementarlo. Sin embargo, como el receptor tiene una ventana de tamaño uno, el enviador debe ignorar los NACKs sucesivos que recibirá para ese mismo paquete. En el fondo, debo retransmitir una sola vez el paquete y su ventana, aunque reciba muchos NACKs seguidos. Para esto, basta recordar el último NACK recibido y no generar más retransmisiones para él. Si vuelve a perderse el paquete de datos, no es grave, ya que igual mantenemos un timeout vigente para toda retransmisión.

4.3. Timeout adaptable

Implementen un timeout adaptable, que va midiendo RTT, de la siguiente forma:

1. RTT y rdev se inicializan a 1s y a 0 respectivamente.
2. timeout se inicializa a 1s.
3. Cada vez que transmito un paquete, anoto la hora de su envío
4. Cuando recibo el ACK para ese paquete, calculo un rtt nuevo.
5. Defino:

```
double diff = RTT-rtt_nuevo;
RTT = RTT - diff*0.125;
rdev = 0.75*rdev + fabs(diff))*0.25;
timeout = (RTT + 4*rdev) * 1.1;
```

6. timeout siempre debe acotarse entre 0.01s y 3.0s
7. Cuando retransmito un paquete, debo evitar recalcular RTT al recibir su ACK (Karn)
8. al retransmitir la ventana por timeout, siempre duplico el timeout actual por si acaso.
9. solo el paquete que generó el timeout debe figurar con un retry.

5. Ejecución

Para esta tarea, se les entrega:

- bws y bwc

  Son el servidor y el cliente ejecutables de esta tarea, con todo implementado y una ventana de transmisión de 200 paquetes.

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

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

y luego el cliente:

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

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

6. Pruebas de Eficiencia

Una vez terminada la tarea, se les pide comparar su eficiencia copiando el mismo archivo (/etc/services es un buen candidato), con delay 0, 0.1 y 0.2, con pérdida de 0 y 10 %, y diferentes tamaños de ventana. Compárelos con la tarea 1. ¿Qué tamaño de ventana es ideal? ¿Por qué una ventana grande hace que sea muy lento?

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

7. 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 para archivos de más de 65536 paquetes (tanto de texto como binarios deben funcionar). Se probará que implemente todas las funcionalidades requeridas. Una buena implementación de los números de secuencia lleva un 4.0, manejar bien los NACKs lleva un 5.0 y los timeouts un 7.0.

Entreguen los fuentes y el makefile necesario para compilar todo, y un LEEME.txt donde muestren los resultados de eficiencia que obtuvieron y su explicación del efecto de las ventanas grandes.

Las preguntas y respuestas en el foro de U-cursos serán obligatorias y evaluadas, como puntos de bonus en las tareas, así que ¡participen!