Control 2
Redes
Plazo de entrega: 16 de octubre 2024

José M. Piquer

P1: Protocolos Clásicos

1.1 Algoritmos clásicos

En general, se considera que Stop-and-Wait es mejor con delay muy bajo y poca pérdida, Go-Back-N es mejor con delay alto y poca pérdida y Selective Repeat es mejor con delay alto y pérdida alta.

Esto, porque no vale la pena pagar la complejidad de un algoritmo avanzado si el más simple funciona bien.

Ahora, supongamos que un físico loco inventó un enlace de gran capacidad (1 Gigabyte/s), casi sin pérdidas, pero de alto delay (5 segundos de RTT).

Responda las siguientes preguntas (justificando sus respuestas, puede usar el simulador del curso para experimentar):

1. ¿Qué protocolo recomendaría usar? *El escenario correspondería a un Go-Back-N, suponiendo que la pérdida es cero. Cualquier pérdida un poco mayor se notará mucho al retransmitir ventanas tan grandes.*

2. ¿Qué configuración del protocolo usaría? (parámetros que correspondan: timeout, tamaño ventana, etc). `La ventana tiene que ser un poco más que el tamaño del BDP por la pérdida. En este caso, un poco más de 5 Gigabytes. Si el RTT son 5 segundos constantes, un timeout de un poco más de 5 segundos, digamos 5.5s`

3. ¿Qué riesgos de ineficiencias tiene su elección? *El problema de un Go-Back-N en este escenario es que la ventana es enorme, por lo que cada retransmisión (debido a un error) cuesta mucho ancho de banda. Por ejemplo, si la ventana está llena y hay una pérdida, tendré que retransmitir 5 Gigabytes de datos. Si considero paquetes de 1 Kbyte, en*

la ventana caben 5 millones de paquetes. Si sólo tengo una pérdida en 5 millones, tendré que retransmitir toda ventana al menos una vez, generando un consumo del doble de lo necesario. Eso es un 0.0000002 % de pérdida. O sea, cuando el físico dice casi sin pérdidas, tiene que ser menos que eso. Cualquier otro escenario podría requerir usar un Selective-Repeat mejor.

4. Suponga un enlace más normal, con una pérdida de tipo 1 %, RTT de 10ms y ancho de banda 100 Megabytes/s: ¿qué protocolo recomendaría? ¿Con qué parámetros? *Ese sería un clásico Selective-Repeat: BDP de 1 Mbyte, con la pérdida lo aumentamos al doble, ventanas de 2 Mbytes, timeout de 15ms.*

5. Si tuviera que elegir cuál de estos dos enlaces comprar (al mismo precio), ¿cómo decidiría cuál tomar? *Primero, como vimos, hay que asegurarnos que la pérdida sea realmente cero. Si fuera así, habría que revisar el tipo de aplicaciones que uso en mi red: si principalmente son video-conferencias y conversaciones en tiempo real, siempre es mejor el enlace más normal. Pero, si principalmente hago transferencias de archivos enormes (por ejemplo un observatorio astronómico) es mucho mejor el enlace del físico loco.*

1.2 Números de secuencia

Selective Repeat tiene una ventana de recepción: si el paquete recibido cae dentro de la ventana, se acepta. Si no, hay dos posibilidades: si es un paquete anterior a la ventana debo descartarlo y enviar un ACK; si es un paquete posterior a la ventana, debo descartarlo sin enviar ACK.

El problema es que usamos siempre números de secuencia que se reutilizan, no son infinitos. Entonces, ¿Cómo hace el receptor del protocolo, cuando recibe paquetes fuera de la ventana, para no confundirse entre paquetes del pasado de paquetes del futuro si comparten el espacio de números de secuencia?

Realmente no hay cómo saberlo. Entonces, usamos propiedades del protocolo: mientras el receptor no haya enviado el ACK para el primer paquete de la ventana de recepción (por que no lo ha recibido aún), sabemos que el enviador no puede haber ido más lejos en el futuro que ese paquete más el tamaño máximo de ventana. Todo paquete en ese rango se considera futuro. Todo paquete fuera de ese rango se considera pasado.

Si la ventana del enviador es del mismo tamaño que la del receptor, todo el futuro cae dentro de la ventana de recepción.

P2: Simulador

2.1 BDP y Pérdida

Un problema con las ventanas de los protocolos clásicos es definir qué tamaño óptimo deben tener cuando conozco el BDP y la probabilidad de pérdida. Claramente es bueno que sean mayores al BDP, pero ¿qué tanto más grande?

Un ingeniero postula que la ventana tiene que ser lo más grande posible, es decir, todo lo que nos permita el protocolo y la memoria disponible.

Usemos el simulador para testear esta hipótesis, usando Selective Repeat con CACK.

El objetivo a lograr en el simulador es aproximarse al Useful Bandwidth obtenido sin pérdidas. Entonces, en una configuración con 5.000 de delay, 10.500 de timeout y 59 paquetes/minuto, una ventana de 11 paquetes cubre el BDP y (sin pérdidas) me da casi 1 de Useful Bandwidth. Al haber pérdidas, aumentará el consumo de ancho de banda. El Useful Bandwidth sólo disminuye si hay bloqueos: momentos en que el protocolo no puede seguir enviando paquetes nuevos a 1 paquete/s.

Usen el simulador: `http://users.dcc.uchile.cl/jpiquer/srgbn2.html` para poder definir ventanas grandes y no usar ventanas de congestión.

OJO: El simulador corre en javascript en el navegador, y eso hace que si la pestaña del simulador no está visible, el navegador la inactiva y deja de ejecutar hasta que es visible otra vez. Por lo tanto, para hacer pruebas un poco más largas, deben dejar la pestaña en primer plano y visible en la pantalla para medir bien los anchos de banda.

Responda las siguientes preguntas:

1. Una ventana de tamaño 40, ¿qué probabilidad de pérdida logra soportar sin bloquearse nunca? *Tipo 0.2*

2. Con pérdidas más grandes, ¿lograremos siempre encontrar una ventana que las soporte sin bloquearse nunca? *No, es cada vez más difícil lograrlo. A modo de ejemplo absurdo, si la pérdida es 100 % no hay forma de lograrlo.*

3. Si pudiéramos tener ventanas de tamaño infinito, ¿soportaríamos cualquier pérdida? *Con ventanas infinitas, podríamos no bloquearnos nunca, ya que no se acabaría nunca la ventana. Pero eso no haría que nos recuperaríamos, estaríamos retransmitiendo todo el tiempo muchos paquetes que no han logrado llegar y saturando infinitamente el ancho de banda.*

4. ¿El ingeniero, en teoría, tiene razón en su hipótesis? *Sí, si solo consideramos el lograr transmitir lo más posible sin bloquearnos, lo más grande que sea la ventana será siempre mejor.*

5. ¿Qué problema podríamos generar en un ambiente real si usamos ventanas enormes? *Saturación de ancho de banda, consumo infinito de memoria tanto en el emisor como en el receptor.*

6. ¿Qué pasaría con los números de secuencia en el caso de ventanas enormes? *Recordando que los números de secuencia deben ser el doble que el tamaño máximo de la ventana, nos encontramos que tendrán que ser de rango muy grandes, utilizando mucho espacio en los paquetes y generando mas overhead.*

2.2 Consumo de Ancho de Banda

En el simulador del curso, se muestran tres valores para el ancho de banda: el Consumo Total (gasto de ancho de banda acumulado), el Consumo Actual (consumo total en un delta de tiempo) y el Consumo útil (transmisión real lograda, sería como lo que mediría la aplicación).

Al haber pérdidas y retransmisiones, es normal que el consumo total suba, mientras que el útil baja. Aunque la generación de paquetes sea fija (típicamente 1 paq/s en nuestros ejemplos), el consumo total puede ser mayor que 1, ya que las retransmisiones se hacen por sobre la generación de paquetes nuevos. Ojo que la probabilidad de pérdida se aplica tanto a los paquetes como a los ACKs respectivos.

Experimentando con el simulador, en Selective Repeat con CACK, pruebe con 59 paquetes/minuto y 0.1 probabilidad de pérdida. Pensando en cómo funcionan los protocolos de retransmisión, responda las siguientes preguntas:

1. Mire cuál es el mejor valor de Consumo Útil que logra. ¿Cuál sería el valor teórico óptimo de Consumo útil? *Logramos tipo 0,92 de consumo útil. El ideal sería lograr 1, ya que si logramos corregir todos los errores sin nunca bloquearnos, debería ser posible.*

2. Mire cuánto gasta de ancho de banda total. ¿Cuál sería el valor teorico óptimo que uno esperaría obtener de consumo de ancho de banda total? *Me da 1.13 de ancho de banda total. Considerando una pérdida de 10 %, consumir un 10 % más de ancho de banda para corregir los errores es lo esperado.*

P3: Congestión

El algoritmo de la ventana de congestión logra efectivamente controlar el ancho de banda utilizado. Pero, la ventana en realidad fue inventada para otra cosa: para adaptarse al BDP del enlace.

Responda las siguientes preguntas:

1. Si hay pérdidas por congestión, ¿es como si cambiara el BDP de la conexión y por eso es bueno cambiar la ventana? *Versión 1: El BDP del enlace no cambia. Pero, al haber pérdidas por congestión, nosotros queremos disminuir el consumo de ancho de banda para disminuir la congestión. Por eso es bueno achicar la ventana, para así frenar la tasa de transmisión.*

   Versión 2: Si consideramos la conexión extremo a extremo en Internet, la congestión cambia la capacidad real de ancho de banda que queda disponible para mi conexión. Por lo tanto, la pérdida sí disminuye mi BDP disponible, y por eso es bueno achicar la ventana.

2. Al achicar la ventana de congestión: ¿se queda bloqueada la conexión esperando ACKs? ¿Es esto lo que se buscaba al achicar la ventana? *Sí, la idea es bloquear al emisor para que disminuya la tasa de envío. Achicar la ventana es la forma de lograrlo.*

3. Al achicar la ventana de congestión: ¿estamos modificando el BDP de la conexión? *Obviamente el BDP del enlace no lo podemos modificar. Pero, al bloquear al emisor estamos disminuyendo el ancho de banda que podemos utilizar, logrando disminuir el BDP realmente consumido por mi conexión.*

4. Proponen una solución alternativa: Podríamos controlar el ancho de banda utilizado (cuando comienzan las pérdidas) agregando un `sleep()` pequeño y adaptable según la pérdida, después de cada envío de paquete, manteniendo una ventana de tamaño constante. Esto ¿es como si mantuviera el BDP de la conexión? ¿Lograría el objetivo de limitar la congestión? *Es bastante equivalente a la solución anterior: al forzar una espera entre paquete y paquete, disminuyo el ancho de banda del que dispongo en la práctica. Por lo tanto, también disminuyo el BDP real del enlace y la ventana no se llenará tanto como antes. Por lo tanto, también serviría para limitar la congestión.*

5. Compare esta solución con la ventana de congestión: ¿sería mejor o peor? *Lo bueno de la nueva solución es que reparte los bloqueos en-*

tre todos los paquetes a enviar. La ventana de congestión bloquea un envío hasta que llegue el ACK de vuelta, pero luego transmite el contenido de la ventana en forma seguida, pudiendo saturar por momentos. Por supuesto, habría que implementar un algoritmo adaptativo para ir cambiando el tiempo del sleep() según las condiciones de pérdida actuales, podría ser tipo AIMD también (pero al revés: incremento del tiempo rápido y disminución lenta). Por otro lado, el mantener la ventana grande hace que tengo muchos paquetes pendientes, y no queda claro cómo manejo las retransmisiones: ¿Debo aumentar el timeout? ¿Debo esperar una vez que el timeout ocurrió?

Mi recomendación sería estudiar una mezcla entre estas dos ideas: achicar la ventana y separar los envíos por tiempos adaptables para evitar saturar el enlace.