

Control 2 (Pauta)

Redes

Plazo de entrega: 26 de octubre 2022

José M. Piquer

P1: PROTOCOLOS CLÁSICOS

1.1 Stop-and-Wait

Vimos en la Tarea 2 como el receptor stop-and-wait podía usarse igual para atender un enviador con go-back-n. Supongamos el mismo caso (números de secuencia 0-9) y el enviador ahora implementa selective-repeat. ¿Podemos hacer lo mismo y que el receptor stop-and-wait siga funcionando bien? Explique. (Si ven la literatura oficial, siempre dicen que Selective-repeat tiene ventanas de envío y recepción del mismo tamaño. Acá les pedimos que violen el dogma y piensen qué pasa si lo implementamos así).

HINT: revisen con cuidado el procesamiento en el receptor de los paquetes distintos al esperado. ¿Debo enviar un ACK para ese paquete? ¿Para el último recibido bien? ¿Otro?

La diferencia de selective-repeat con go-back-n es que ahora no retransmitimos la ventana completa, sino sólo los paquetes que no han recibido ACK a tiempo. Si la ventana del receptor es de tamaño 1, implementando un Stop-and-Wait simple, ignoraremos los paquetes fuera de rango, pero debemos decidir qué ACK responder. Ahora los ACKs no son acumulativos (como en Go-Back-N) por lo que si envío un ACK para 3, el emisor puede seguir esperando un ACK para 0 y no avanzar nunca. Entonces, debemos responder con el ACK correspondiente al paquete ignorado, si es un paquete antiguo (retransmitido). Pero, si es un paquete en el futuro, debo responder con un ACK del último paquete recibido correcto solamente. Entonces, el receptor debe distinguir estas dos situaciones, y para eso debe conocer el tamaño de la ventana de envío. Eso implica que no puedo usar el receptor clásico de Stop-and-Wait y debo modificarlo.

1.2 Secuencias y ventanas

OJO: Esta pregunta contiene 5 sub-preguntas que debe responder.

Vimos en clases que se puede usar sólo la mitad de los números de secuencia disponibles para el tamaño máximo de ventana del enviador en selective repeat. Explique por qué. Luego muestre un escenario en

que, si usáramos uno más que la mitad, fallaría, en un ejemplo de número de secuencia entre 0-5 y una ventana de tamaño 4. ¿Cuál es la regla si el número de secuencia es impar (ejemplo 0-6)? ¿Qué pasa si selective repeat usa una ventana de recepción más grande que la del emisor? ¿Cuál es la regla de tamaño máximo de ventana para Go-Back-N (explique por qué)?

El caso a evitar es que transmito toda la ventana, se recibe bien, pero se pierden todos los ACKs. Entonces, el emisor está con una ventana completa de envío llena y el receptor está esperando una ventana completa siguiente a esa. Lo importante es que, entre las dos, no compartan ningún número de secuencia.

En el ejemplo, tengo ventanas de tamaño 6, y transmito 4 paquetes: 0-3. Se me pierden los 4 ACKs, el emisor retransmite 0-3 y el receptor espera 4,5,6,0. En este caso, va a recibir el paquete 0 (de la ventana anterior) creyendo que es el paquete 0 de la nueva ventana y lo va a aceptar erróneamente. Si tengo un número impar N de secuencias, sólo puedo usar ventanas iguales al caso que tuviera $N - 1$ y dejar un número de secuencia sin usar. En el caso 0-6 sólo puedo usar ventanas de tamaño 3.

Si el receptor usa una ventana más grande que la del emisor, lo único importante es que esa ventana no exceda el tamaño máximo definido por los números de secuencia. El protocolo queda correcto, pero no ganamos nada con esa ventana mayor y seguimos limitados por el tamaño del emisor, por lo que es un gasto inútil de memoria.

En Go-Back-N puedo usar ventanas más grandes para los mismos números de secuencia, y me basta con dejar un número de secuencia libre entre una ventana y la siguiente. Si tengo N números de secuencia disponibles, puedo usar ventanas de tamaño máximo $N - 1$. Esto se debe a que, en el mismo escenario anterior, un emisor con secuencias 0-5 y una ventana de tamaño 5, transmite paquetes 0-4, se pierden los ACKs, el receptor espera ahora el 5 y sólo necesito que ese número nunca esté en la ventana anterior, por eso no puedo tener ventana de tamaño 6 en este caso. Pero, con tamaño $N - 1$ está perfecto.

P2: VENTANAS CORREDERAS

OJO: Esta pregunta contiene 2 sub-preguntas que debe responder.

Vimos que el tamaño óptimo de ventana es justo el BDP (Bandwidth-Delay Product) del enlace. Sin embargo, al calcularlo, usamos el RTT (Round-Trip-Time y no el delay. Explique por qué es el RTT el que importa.

El tiempo que demoro entre enviar el primer paquete y recibir su ACK es RTT. Todo ese tiempo, yo quisiera poder seguir transmitiendo paquetes, y no quedarme bloqueado esperando el ACK. Por eso, es RTT el tiempo que importa.

Sin embargo, si lo que queremos es calcular cuántos bytes “cabén” en el enlace en tránsito entre el origen y el destino, efectivamente es

el delay el que nos permite calcular eso. Al usar el RTT, casi duplicamos ese tamaño. Explique por qué eso es correcto a pesar de la contradicción aparente.

Efectivamente, en el enlace “cabén” solo la mitad de esos bytes. La otra mitad debe poder ser almacenada en los buffers del receptor, entre la ventana de recepción y los buffers del socket y la aplicación para que esto pueda funcionar bien. Si no hay espacio para esos bytes (que pueden ser muchos), no servirá de nada tener ventanas de tamaño de ese BDP.

P3: TCP Y VENTANA DE CONGESTIÓN

1.1 Congestión

OJO: Esta pregunta contiene 3 sub-preguntas que debe responder.

La ventana de congestión se crea para limitar el consumo de ancho de banda de TCP, partiendo del supuesto de que ese consumo es el que está saturando la red y que, al disminuirlo, tendremos menos pérdidas. Sin embargo, en redes inalámbricas, tengo pérdidas por ruido e interferencias que no tienen nada que ver con congestión. Suponga que, al ocurrir un timeout y forzar retransmisión, el código puede usar una variable global que me indica la calidad de la señal inalámbrica (0=pésima, 100=óptima). ¿Podríamos modificar el algoritmo de la ventana de congestión para aprovechar de usar esa información? ¿Qué haría Ud? ¿Qué riesgos tiene su algoritmo?

Uno podría reaccionar frente a la pérdida más o menos exageradamente según el parámetro de calidad. Por ejemplo, en vez de dividir por 2 la ventana, podríamos ponderarlo y dividir la ventana por $(1 + n)$ donde n es la calidad dividida por 100. Entonces, con un 0 de calidad, dejamos la ventana igual y con un 100 de calidad, la dividimos a la mitad. El riesgo de esto es que haya ruido en la señal y congestión a la vez, en cuyo caso este algoritmo no disminuirá su consumo, pudiendo ayudar a saturar la red.

1.2 Timeouts

Vimos que TCP trata de calcular un timeout óptimo, que se aproxime lo mejor posible al RTT actual, pero considerando la varianza actual también. Explique por qué esto es tan importante. ¿Qué pasa si mi valor de timeout es demasiado pequeño? ¿Qué pasa si es demasiado grande?

Un timeout demasiado pequeño hará que retransmitiremos paquetes inútilmente. En el extremo, si siempre es menor al RTT, gastaremos el doble del ancho de banda requerido en un esquema tipo selective repeat. Un timeout demasiado grande hace muy lenta la recuperación frente a pérdidas, y nos genera tiempos de bloqueo esperando una respuesta que ya sé que no llegará nunca. En el extremo, un timeout infinito, deja bloqueado para siempre el protocolo.