Conexiones

November 14, 2014

Universidad de Buenos Aires - Departamento de Computación - FCEN

Integrantes:

- Gallardo, Guillermo L.U.: 32/10 gagdiez.c@gmail.com
- Fixman, Martin L.U.: 391/11 martinfixman@gmail.com
- Matayoshi, Leandro L.U.: 79/11 leandro.matayoshi@gmail.com
- Szyrej, Alexander L.U.: 642/11 alexander.szyrej@gmail.com

Contents

1 Introducción

Los protocolos confiables de transporte permiten al usuario abstraerse de los problemas de transmisión de la red, por ej, el protocolo TCP nos permite enviar mensajes correctamente a través de medios que pueden perder, desordenar o incluso modificar paquetes.

El protocolo PTC es una implementación de la catedra, el mismo está basado en TCP, por lo tanto, intenta asegurar que los paquetes lleguen de un punto a otro utilizando diversos mecanismos como control de flujo, realizar la conexión en etapas, mantener una ventana deslizante para enviar y recibir paquetes, etc.

En este trabajo estudiaremos como se comporta PTC, poniendo especial atención a la retransmisión de paquetes. Para ello primero emularemos escenarios donde existan delay y perdida de paquetes entre dos nodos comunicados. Luego, iremos modificando parámetros de la implementación para ver como ello afecta el reenvío de paquetes.

2 Métodos

El protocolo PTC utiliza un sistema de *acknowledge* para saber si un paquete fue recibido. En caso de no recibir un *ack* para un determinado paquete luego de un umbral de tiempo, procede a retransmitirlo. Dicho umbral, denominado *rto* se va estimando a medida que se reconocen los paquetes enviados usando las siguientes funciones:

$$RTTVAR = (1 - \beta)RTTBAR + \beta|SRTT - RTT|$$

$$SRTT = (1 - \alpha)SRTT + \alpha RTT$$

$$RTO = SRTT + max(1, K * RTTVAR)$$

Donde $K, \alpha, \beta \in \mathbb{Q}$, RTT es el valor de *rtt* calculado para el último *ack* recibido. En la implementación utilizada se inicializan los valores de la siguiente manera: $K = 4, \alpha = \frac{1}{8}$, $\beta = \frac{1}{4}$, $SRTT = RTT_0$, $RTTVAR = \frac{RTT}{2}$. Siendo RTT_0 el primer *rtt* calculado.

3 Desarrollo

En primera instancia se implementaron dos scripts en python que actuan uno como cliente y otro como servidor. Luego, se modificó la implementación de PTC para poder simular delay y perdida de paquetes (el contexto de una red), además de transformar en parámetros a los valores α y β para estudiar como se modifica el *rto* en base a los mismos.

3.1 Modificaciones a la Implementación

Se agregaron al sistema variables para manejar la probabilidad de que un paquete se pierda en el envío, el delay antes de contestar con un ACK, el valor de α y β.

3.1.1 rto

Se modificó para que los valores de α y β fueran variables seteables al momento de inicializar el *RTOEstimator*.

3.1.2 protocol

Se modificó la inicialización, ahora recibe los parámetros α, β, *perdida* y *delay*.

α, β se utilizan para inicializar el *RTOEstimator*.

Se modificó el método *send_and_queue*, ahora, espera *delay* ticks antes de enviar el paquete con una probabilidad de $1 - perdida$. En una primera instancia se intentó incluir el delay en la función *send_ack* del módulo *handler*, pero en la experimentación notamos que ciertos valores de *rtt estimado* quedaban muy por debajo del delay seteado, por lo tanto entendimos que algunos acks nos evadían, y se mandaban por otro método. Dado que el paquete se trackea antes del delay en *send_and_queue*, el cliente interpreta el tardío ack como una demora en la red y calcula el rtt/rto en base a esa demora.

Se modificó también el método *free* para que imprima el *rto* estimado durante la conexión.

Se agregaron métodos para poder saber en cualquier momento la cantidad de paquetes reenviados y el *rto*, *rtt* actuales.

3.1.3 ptc_socket

Se modificó el wrapper, ahora se debe inicializar con valores para α, β, *perdida* y *delay*, los mismos son utilizados al momento de crear el objeto *PTCProtocol* del paquete *protocol*.

Se creó el método *alumnos_change_delay* que recibe un entero y cambia el delay de la conexión mediante el método nuevo puesto en *protocol*.

3.2 Análisis

Previamente a la experimentación nos detuvimos a analizar la fórmula que determina el RTO en una retransimisión dada (Ver sección métodos).

α es utilizada para ponderar el valor de los RTT's anteriores con el último recibido. De la ecuación para calcular el SRTT podemos deducir que valores de α pequeños hacen que se le asigne mayor peso a las mediciones anteriores, mientras que valores de α mayores establecen una mayor influencia de la última muestra de RTT obtenida. Ambas opciones tienen ventajas y desventajas: la primera genera un RTT más estable, aunque no es lo suficientemente rápida como para adaptarse a los cambios. La segunda aproximación es buena en este último aspecto, aunque es susceptible a ser afectada en gran medida por fluctuaciones temporales.

La variable β utilizada para actualizar el valor de RTTVAR juega un papel totalmente análogo al de α. Valores pequeños hacen que se pondere en mayor medida el valor de las variaciones anteriores, mientras que valores altos le asignan mayor importancia al valor de la última diferencia entre la muestra del RTT obtenida y el valor del SRTT.

3.3 Experimentación

Para estudiar el comportamiento de PTC realizaremos los siguientes experimentos: *Valor de RTO variando* α *y* β, *Valor de RTO cuando la red se congestiona sin perdida*, *Cantidad de paquetes perdidos variando* α *y* β, *Tiempo total de transmisión en entorno con pérdidas.*

4 Análisis y Resultados

4.1 Valor de RTO variando α y β

Primero estudiamos como varía el *rto* cuando se utilizan los valores extremos de α y β. Se utilizó un valor de delay = *6 ticks* sin probabilidad de pérdida.

Puede verse en la *Figura 1*, que tomando $\alpha = 0$, $\beta = 0$ los cálculos ignoran por completo los nuevos valores de RTT y consideran únicamente los valores de SRTT anteriores. En este caso, el valor constante inicial.

La *Figura 2* muestra como al tomar ambas variables el valor 1, los cálculos ignoran los valores anteriores (tanto de variación de RTT (RTTVAR) como de SRTT), y modifican los resultados únicamente en función de la última muestra de RTT obtenida. El resultado es un gráfico en donde el RTO coincide exactamente con el SRTT, y ambos responden excesivamente a cualquier fluctuación que se produzca en el valor del RTT.

Por otra parte, la *Figura 3* muestra lo que sucede al ponderar de igual manera los valores anteriores con los nuevos, al tomar ambas variables con valor 0.5. Esto debería generar valores cercanos a los de las muestras anteriores, pero que sean capaces de responder a cambios significativos y permanentes del valor de RTT. En otras palabras, que alcancen un equilibrio entre ambas propuestas. Sin embargo, el comportamiento no fue el esperado.

El resultado es una serie de cálculos demasiado sensibles a las fluctuaciones temporales que responden excesivamente a estos cambios. Por ello hicimos una última prueba (*Figura 4*) donde fijamos el valor de α en 0.15 y β en 0.20 (cercanos al propuesto por el RFC 6298). Las conexiones en las distintas redes por lo general permanecen estables la mayor parte del tiempo, por lo que los cambios bruscos de *rtt* son poco probables. Por otro lado el valor de β permite agregar un poco de sensibilidad a los cambios en el valor de los *rtts*.

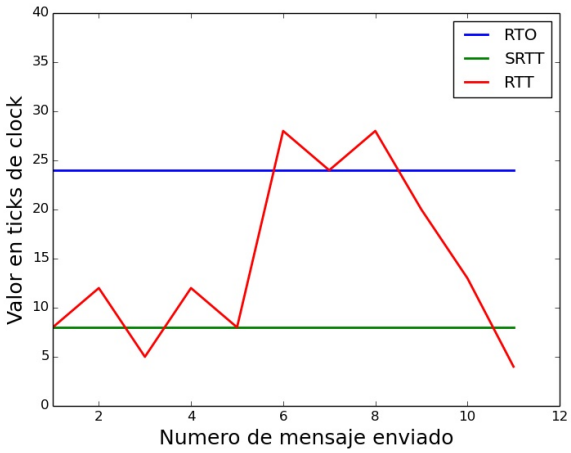

Figura 1

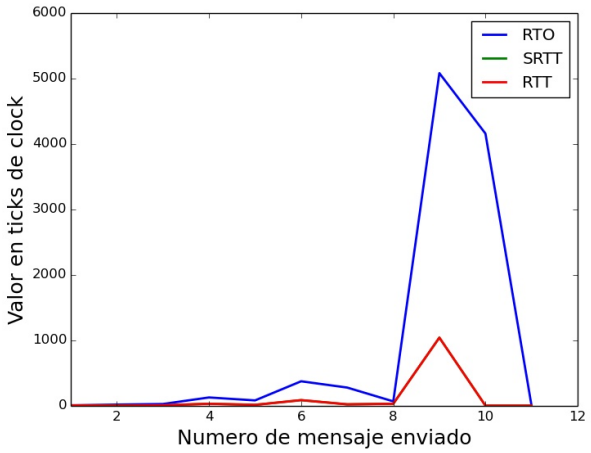

Figura 2

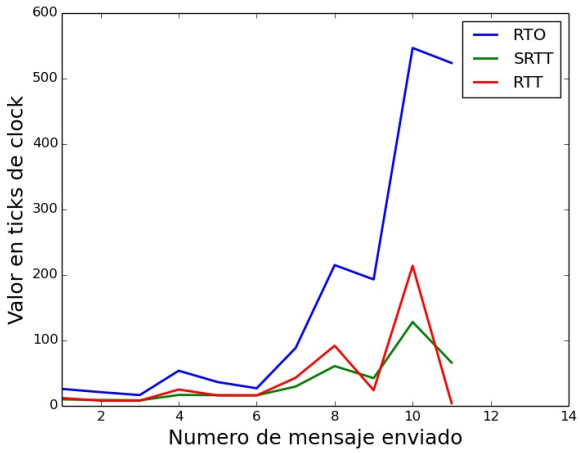

Figura 3

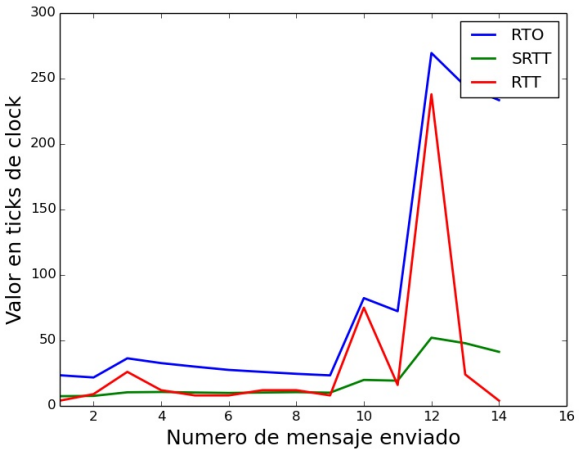

Figura 4

Finalmente intentamos medir muchos valores de *rto* para diversas convinaciones de α y β. Los resultados de medir el *rto* para un valor fijo de *25 ticks* y probabilidad de perdida cero luego de enviar 200 paquetes se pueden ver en la *Figura 4*.

4.2 Perdida de paquetes variando α y β

Para estudiar la perdida de paquetes se fijó el valor de de *rto* en *25 ticks* y no se simularon pérdidas. Sin embargo en varios casos hubo retransmisiones, la *Figura 5* muestra el resultado para el envío de 150 paquetes mientras que la *Figura 6* lo muestra para 300.

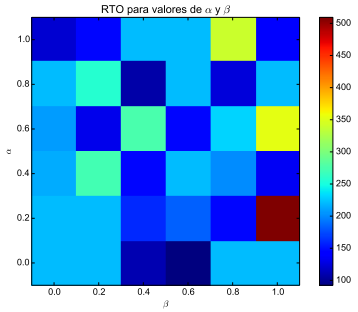

Figura 4

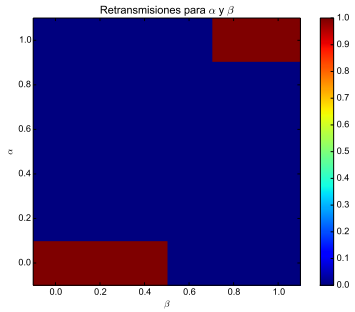

Figura 5

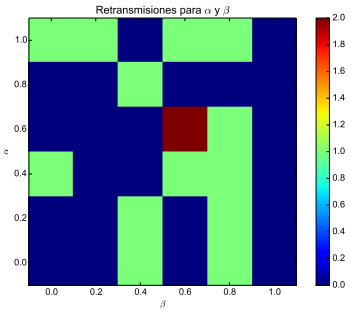

Figura 6

4.3 Valor de RTO cuando la red se congestiona sin perdida

Para este configuramos el cliente para que envíe 300 paquetes al servidor, pero luego de enviarse 150 paquetes, el delay de la red se duplicara o cuadruplicara.

El *Escenario 1* muestra el resultado en una red con paráemtros: $\alpha = \frac{1}{2}$, $\beta = \frac{1}{4}$ cuando no se congestiona, cuando la congestión causa el doble de delay, y cuando causa el cuadruple.

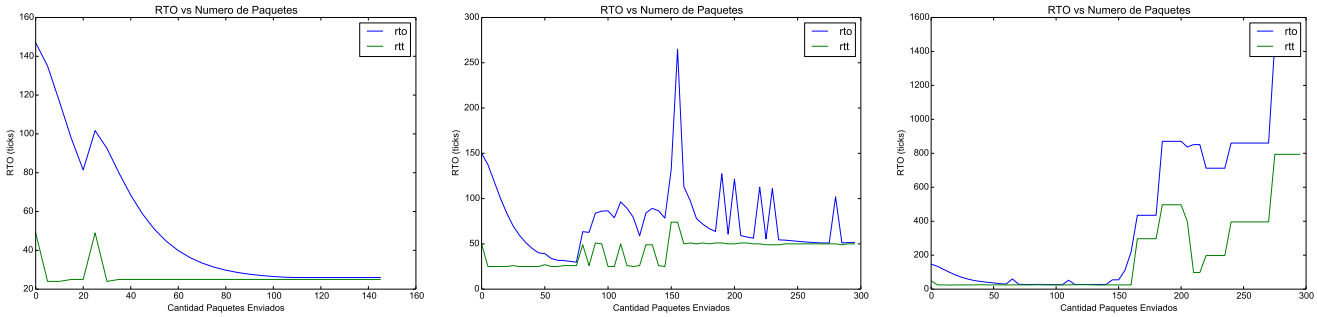

Figure 4: Escenario 1

El *Escenario 2* muestra el resultado en una red con paráemtros: $\alpha = \frac{1}{8}$, $\beta = \frac{1}{2}$ cuando no se congestiona, cuando la congestión causa el doble de delay, y cuando causa el cuadruple.

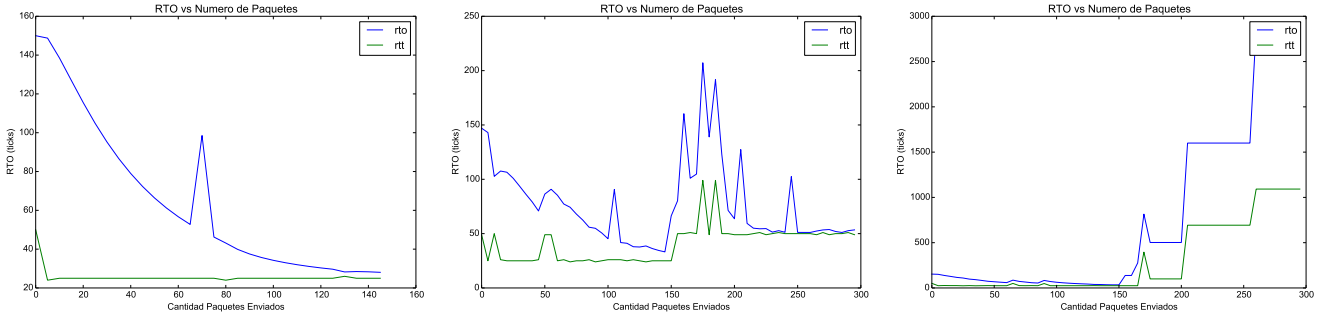

Figure 5: Escenario 1

En ambos casos sucedió que al duplicar el valor del delay el mensaje número 101 debió ser retransmitido. A su vez, cuando se cuadruplicó el valor de la red, el mensaje número 101 debió ser retransmitido dos veces. También se puede notar que en ambos casos luego de cuadruplicarse el delay no se logró volver a un *rto* cercano a el *rtt* real.

4.4 Tiempo total de transmisión en entorno con pérdidas

Para este experimento se configuró un escenario en el cual el cliente envía 300 mensajes al servidor, pasados los 150 el delay se incrementa de *25 ticks* a *50 ticks*. A su vez, la probabilidad de perder un paquete es de 0.1. Luego, utilizando wireshark, se calculó el tiempo que duró la conexión en base a los paquetes capturados y su timestamp.

Para los valores de $\alpha = 0$ y $\beta = 0$ el tiempo fue 42,19 seg, para los valores propuestos por el RFC el tiempo promedio fue 41,67 seg, para $\alpha = 0.15$ y $\beta = 0.2$ 42,39 seg, finalmente para $\alpha = 1$ y $\beta = 1$ el tiempo fue 44,34 seg.

5 Conclusiones

Todos los experimentos demostraron lo susceptible que es la medición de RTO a los parámetros de la red, así como también a los parámetros utilizados para estimarlo.

Primero vimos que cambiar los valores de α y β modifican significativamente el valor estimado de *rto*, al experimentar con valores muy pequeños, los cálculos no se adaptan de forma realista a las variaciones en los valores de RTT reales. Por el contrario, al tomar valores muy altos los mismos se ven demasiado influenciados por fluctuaciones temporales, dando como resultado valores demasiado altos o demasiado pequeños en comparación con el RTT real.

Luego vimos como los valores de α y β modifican la cantidad de retransmisiones, cabe destacar que las *Figuras 5 y 6* son parte de un experimento en el cual no se simuló perdida de paquetes, sin embargo , hubo retransmisiones. Esto se debe a que cuando el *rto* se acerca al *rtt* estimado es muy sensible a cambios internos de la pc, recordemos que los procesos de cliente y servidor están funcionando dentro de un sistema operativo hogareño.

Por último, vimos como el cálculo del *rto* afecta a la transmisión total de una conexión. Podemos ver que entre el primer resultado hay casi dos segundos de diferencia, teniendo en cuenta que el tiempo para enviar un mensaje entre ambos nodos es menor a 0.1 seg, podemos ver que la diferencia de performance es notable.

Para concluir queremos agregar que el valor que mejor funcionó en los experimentos fue el propuesto por el RFC, consiguiendo aproximarnos con $\alpha = 0.15$ y $\beta = 0.2$.