DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 3

Capa de Transporte: Programación de protocolos end-to-end

14 de Noviembre de 2014 — Teoría de las Comunicaciones

Integrante	LU	Correo electrónico
Juan Manuel Tastzian	39/10	jm@tast.com.ar
Lucas Tolchinsky	591/07	lucas.tolchinsky@gmail.com
Nicolás Vallejo	500/10	nico_pr08@hotmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Introducción

1.1. Retransmission Timeout (RTO)

El *timeout de retransmisión* es un valor utilizado en protocolos como **TCP** que sirve para *asegurarse la entrega* de un paquete al recipiente, a pesar de la ausencia de todo *feedback* de su parte. Se encarga de reenviar los paquetes al no recibir confirmación de recepción.

La idea del **RTO** es, entonces, que dicho valor refleje el tiempo a esperar desde el envío de un paquete hasta el envío de la retransmisión correspondiente en caso de no recibir un reconocimiento del receptor.

Intuitivamente es fácil imaginar que un valor óptimo para el **RTO** es aquel que se aproxime de la mejor manera al **RTT** promedio de la comunicación, dado que un valor al **RTT** supondría el envío de una retransmisión antes de la llegada de la respuesta del receptor (en caso de haberse efectivizado), y un valor mayor al **RTT** supondría una pérdida de tiempo antes de retrasmitir.

De esta manera, es de esperar que a medida que se realicen progresivas comunicaciones entre dos host el **RTO** se vaya recalculando para reflejar así el tiempo recomendado de espera antes de retrasmitir un paquete.

Para computar el **RTO**, el emisor del paquete mantiene dos variables de estado, **SRTT** (*smoothed round-trip time*) y **RTTVAR**(*round-trip time variation*), así como también hace uso de ciertas constantes, α, β y K.

El algoritmo básico propuesto por el **RFC 6298** cuenta con los siguientes pasos:

- Para la primer medición de RTT, fijar:
  SRTT = RTT
  RTTVAR = $\frac{RTT}{2}$
  RTO = SRTT * max(G, K*RTTVAR), siendo G la granuralidad del reloj y K=4.

- Para subsecuentes mediciones de RTT, debe ejecutarse, en este orden:
  RTTVAR = (1 - β) * RTTVAR + β * |SRTT - RTT |
  SRTT = (1 - α) * SRTT + α * RTT
  RTO = SRTT * max(G, K*RTTVAR), siendo G la granuralidad del reloj y K=4.

El paper en cuestión sugiere utilizar $\alpha = \frac{1}{4}$ y $\beta = \frac{1}{8}$, valores que suponemos surgen de resultados empíricos en pruebas del algoritmo.

Cualquier implementación debe manejar el/los timer(s) de retransmisión de forma tal que *un segmento* ***nunca*** *es retransmitido demasiado temprano* (es decir, en menos de un RTO luego de la transmisión del segmento anterior).

A continuación, dejamos el algoritmo **recomendado** en el **RFC 6298** para el manejo del timer de retransmisión:

1. Cada vez que se envía un paquete con datos (incluyendo una retransmisión), si el timer no está corriendo, se inicia el mismo para que expire luego de RTO segundos (para el valor actual de RTO).
2. Cuando se hizo ACK de todos los datos, se apaga el timer de retransmisión.
3. Cuando se recibe un ACK que confirma nuevos datos, se reinicia el timer de retransmisión para que expire luego de RTO segundos (para el valor actual de RTO).

Cuando expira el timer de retransmisión, hacer lo siguiente:

4. Retransmitir el segmento más viejo que no haya sido confirmado por el receptor de TCP.
5. El emisor debe setear **RTO** $\leftarrow$ **RTO * 2** ("hacer *back off* del timer"). Como valor para acotar superiormente esta operación, se puede usar el valor de 60 segundos.

6. Comenzar el timer de retransmisión para que expire luego de *RTO* segundos (para el valor de *RTO* conseguido luego de hacer la operación de duplicación en el ítem número 5).

7. Si expira el timer esperando el *ACK* de un segmento *SYN* y la implementación está usando un *RTO* menor a 3 segundos, el *RTO* debe ser re-inicializado a 3 segundos cuando comienza la transmisión de datos (es decir, luego de finalizar el three-way handshake).

1.2. Sobre este Trabajo Pŕactico

El objetivo de este trabajo práctico será analizar el comportamiento del protocolo PTC provisto por la cátedra con respecto al cálculo del RTO, simulando en un esquema cliente-servidor en el ámbito de una red local delay y pérdida de paquetes en el envío de los **ACK** por parte del segundo.

Además, se experimentará alterando los valores de α y β propuestos por el algoritmo descripto en la sección anterior y se estudiará su impacto a la hora de calcular el RTO, con el fin de determinar algunos valores óptimos que lo acerquen tanto como sea posible al RTT real.

2. Desarrollo

Para el desarrollo de este Trabajo Pŕactico se realizaron modificaciones al código fuente del protocolo PTC presentado por la cátedra. Parte de las modificaciones se debieron a las consignas del trabajo, no obstante lo cual se aplicaron más cambios con el objeto de facilitar la realización de los experimentos.

Principalmente se modificó el archivo **handler.py**, en particular el método **send_ack**, que en un principio era el encargado de enviar paquetes PTC solamente con el flag ACK encendido. Las modificaciones en esta porción del código se correspondieron con la introducción de un delay a la hora de enviar este tipo de paquetes además de decidir si efectivamente se realizará el envío del paquete basado en una cierta probabilidad de dropeo de paquetes dada.

Estos nuevos valores de delay y probabilidad de dropeo se pasan por parámetro a la hora de inicializar un Socket PTC y se utilizan también al momento de inicialización de una instancia del protocolo. Por esto, se modificaron también los archivos **ptc_socket.py** y **protocol.py**, donde se agregaron atributos a las clases correspondiente que reflejaron estos valores de customización.

Acerca de la probabilidad que debe introducirse como parámetro, creemos necesario mencionar que esperamos un valor que pueda expresarse de la forma $\frac{1}{n}$, siendo n un número entero. Esto se decidió así para no complicar en demasía las modificaciones a realizar y porque consideramos que con probabilidades de esa forma era suficiente para realizar los análisis correspondientes.

La nueva versión de **send_ack** funciona, entonces, de la siguiente manera:
Dada la probabilidad de dropeo, la invierte para obtener un número entero n. Con ese valor de n como límite se obtiene un número entero entre 1 y n, haciendo uso de funciones nativas de python que hacen uso de distribución uniforme. Luego, el paquete se enviará si y solo si ese número obtenido es igual a 1, efectivizando la probabilidad de dropeo en $\frac{1}{n}$.

Si corresponde enviar el paquete entonces se aplica un delay de tantos segundos como se haya especificado a la hora de crear el socket PTC. Simulamos ese delay mediante un sleep.

Finalmente la última modificación que se realizó al código fuente fue el agregado de dos atributos a la clase RTOEstimator, que se corresponde con el alpha y beta a utilizar en los cálculos de estimación del RTO y los consiguiente cambios para que los cálculos los utilicen. Esto se hizo al notar al momento de experimentar que no se estaban modificando dichos valores.

2.1. Experimentación

Entendiendo que el **RTO** óptimo del protocolo es el **RTT**, lo primero que decidimos investigar fue cómo afectaban diferentes rangos de α y β a **PTC** en el escenario ideal sin pérdida de paquetes. Para esto fue necesario primero establecer los parámetros de nuestro experimento y las primeras pruebas arrojaron que alrededor de 50 paquetes eran más que suficientes para lograr que el cálculo del **RTO** se estabilizara.

Sabiendo esto ejecutamos la transferencia de 50 paquetes entre cliente y servidor para par (α, β) con $\alpha \in [0.1, 0.9]$ y $\beta \in [0.1, 0.9]$. Luego de realizadas estas pruebas pudimos definir cuál era el β que más acercaba el **RTO** al **RTT** promedio para cada α.

Habiendo seleccionado un β para cada α procedimos a realizar pruebas con probabilidad de dropeo de paquetes. Nuevamente, fue necesario definir qué probabilidades considerábamos apropiadas para realiar estos nuevos experimentos.

Consideramos que una red, para que sea efectivamente usable, no puede tener una gran probabilidad de pérdida de paquetes. Teniendo esto en cuenta imaginamos que una red donde un paquete tiene una probabilidad $\frac{1}{2}$ de perderse no sería muy útil y tomamos como límite este valor. Las probabilidades de pérdida que elegimos para realizar las pruebas fue 0,1, 0,3 y 0,5.

Este nuevo experimento nos permitió determinar si el mejor α, β seguía manteniéndose con pérdida

de paquetes. También vale mencionar que nos pareció interesante contrastar el comportamiento de estos valores contra los resultados obtenidos con los valores óptimos propuestos en el RFC 6298, $\alpha = 0{,}125$ y $\beta = 0{,}25$ para el último experimento.

3. Análisis

3.1. Experimento 1

Para este experimento planteamos un esquema cliente-servidor donde el cliente envía paquetes y el servidor solamente se encarga de reconocerlos.

El objetivo era observar la evolución de la estimación del RTO desde el lado del cliente dependiendo del par de α y β que se utilicen y en base a todos las combinaciones que se probaron, elegir las que a nuestro criterio nos dieran mejores resultados.

Es fácil desprender de la definición del RTO que idealmente este valor debería coincidir con el RTT de la transmisión. Luego, esperábamos observar que el a medida que se envían paquetes el RTO estimado converge al valor del RTT.

Como nos interesaba observar el impacto de estos valores en la progresión de la estimación del RTO, fijamos un valor de delay en 2 segundos cada vez que el servidor enviaba un ACK y no se definió probabilidad de dropeo. Así, nos asegurábamos un RTT de aproximadamente dos segundos (dado que ambos extremos de la conexión eran locales, el delay de la conexión podría despreciarse y el RTT dependería casi totalmente del delay forzado a mano a la hora del envío del ACK), de manera que se podía contrastar fácilmente en un gráfico, y además que todos los mensajes enviados eran correctamente reconocidos en tiempo y forma, sin dar lugar a retransmisiones.

Con esto en mente, comenzamos a hacer corridas del experimento con diversar combinaciones de α y β. Consideramos suficiente hacer ochenta y un combinaciones, que surgen de elegir para ambas variables valores desde 0.1 a 0.9 aumentando en cada corrida en 0.1 la varible analizada y combinándola con los nueve valores de la otra variable.Si bien esto es arbitrario, probamos algunos pocos valores en los intevalos sin analizar y decidimos que aumentar la granularidad no nos daría cambios significativos en los resultados.

Para analizar estos 81 resultados, partimos los datos en nueve gráficos[1], fijando cada valor de α y graficando las variaciones de los β.

En todos los gráficos, aparte del resultado de cada β con el α de dicha corrida, se encuentra una línea constante que representa el **RTT promedio** de cada corrida, que es el valor al que conjeturamos el RTO debería converger. Es notable que a pesar de haber fijado el delay en dos segundos, los RTT que se fueron obteniendo fueron ligeramente más pequeños, algo quizá atribuible a nuestra implementación. Sin embargo, dado que en todos los casos resultaba en valores casi constantes, consideramos que no enturbiaba la experimentación realizada.

Dijimos que ibamos a seleccionar de estos valores aquellos que mejor nos parecían. El criterio utilizado para dicha selección consiste en un equilibrio entre dos cuestiones: en primer lugar, la velocidad de convergencia al RTT, suponiendo mejor una convergencia más rápida; y en segundo lugar, la estabilidad de la progresión una vez cerca del RTT, tratando de evitar aquellas con comportamiento errático. Debe entenderse entonces que se contemplan ambos aspectos en la elección (ligeramente subjetiva) de dichos valores de α y β, y que, por lo tanto, un corrida que por ejemplo converge antes, no es necesariamente la que seleccionamos.

En los siguientes gráficos se puede observar la progresión de la estimación del RTO en función de la cantidad de paquetes enviados. En este experimento en particular la cantidad de envíos se condice con la cantidad de estimaciones del RTO, porque se forzó al cliente a que recién enviara otro paquete cuando hubiera recibido un mensaje de confirmación del servidor (cuyo envío en el servidor forzamos nosotros y que no es el ACK). Hicimos esto para tener más control en la corrida y en la obtención de los datos. Por esta razón, el eje x de las siguiente figuras podría entenderse tanto como paquetes enviados como cantidad de estimaciones del RTO.

En cada gráfico, además, está fijado un α y se muestran las nueve curvas que surgen de combinar dicho α con los β. Diferenciada de las demás curvas en azul se puede percibir la curva que se corresponde con el par α β elegido en cada gráfico.

[1] Aparte de estas imágenes, debajo de cada uno se encuentra un link a su versión interactiva, para poder apreciar mejor los datos.

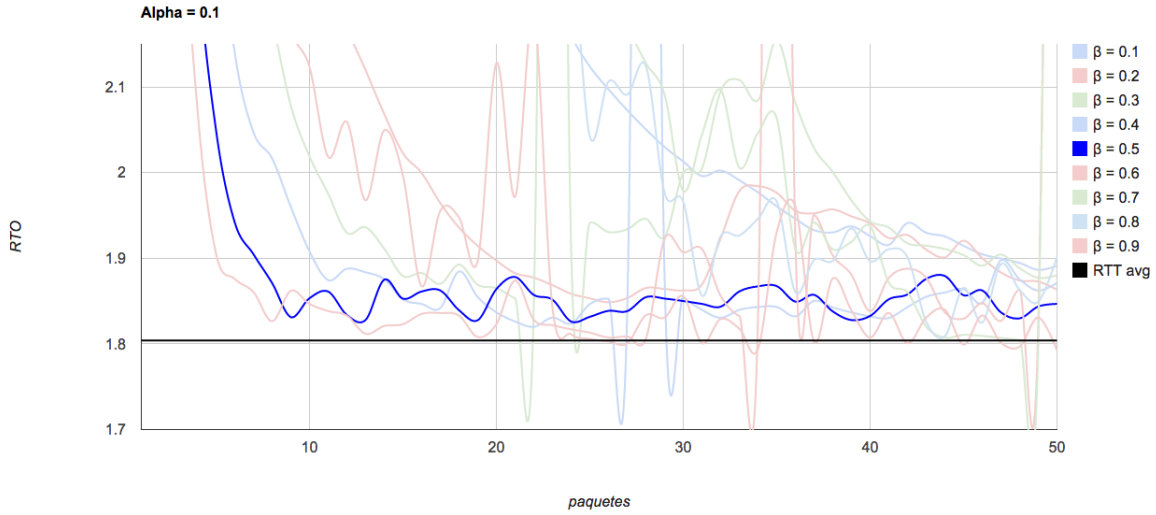

Gráfico interactivo en: http://goo.gl/i3q8vd

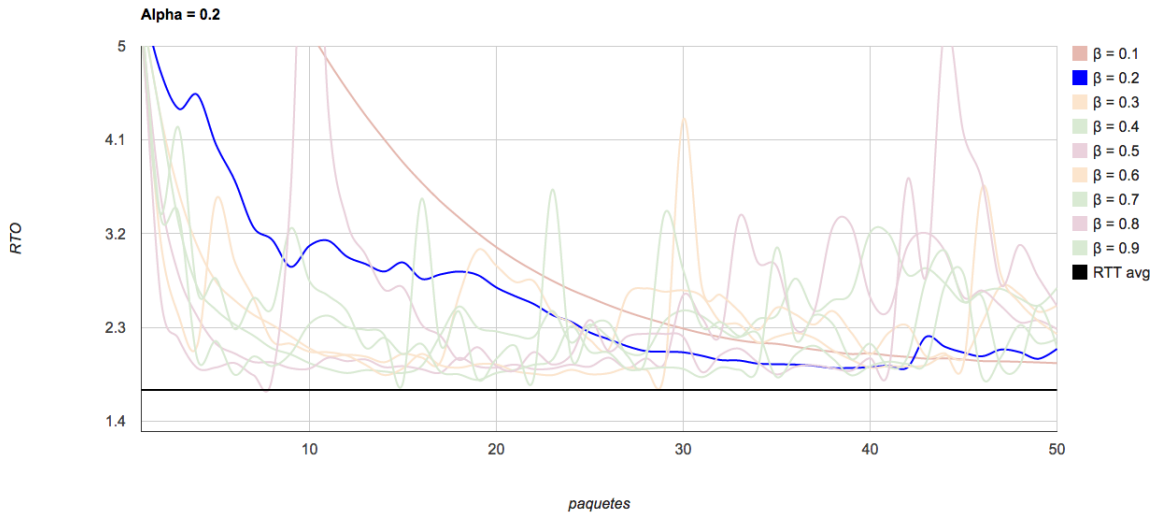

Gráfico interactivo en: http://goo.gl/QS0NjO

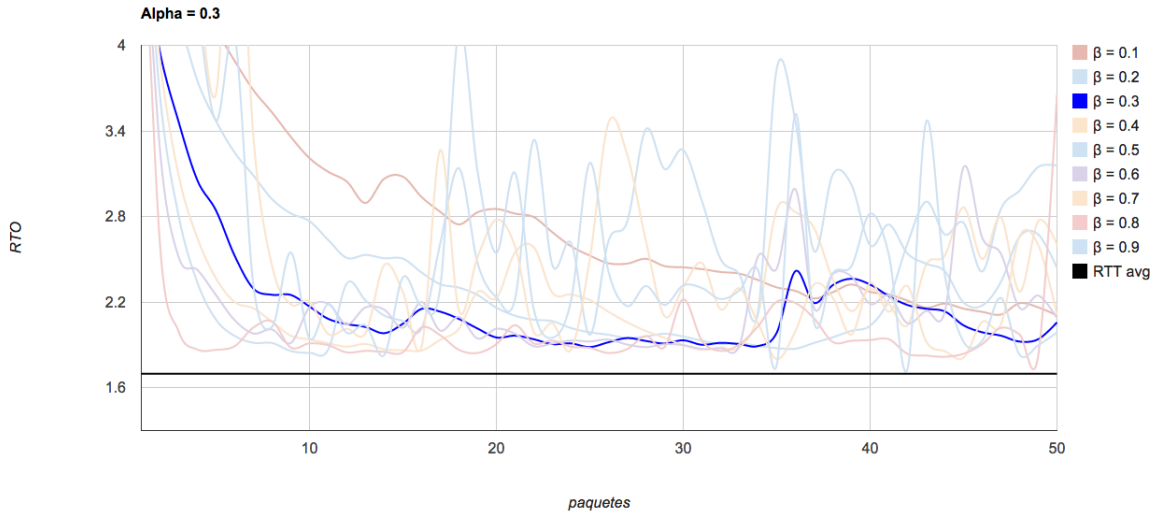

Gráfico interactivo en: http://goo.gl/cmSyGd

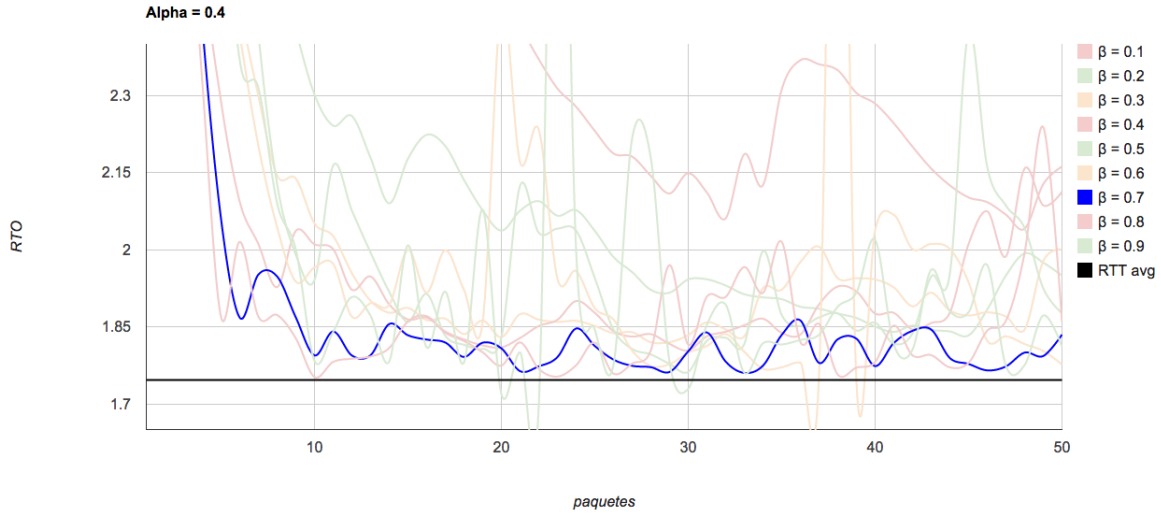

Gráfico interactivo en: http://goo.gl/Db08yU

Notar que estos primeros cuatro gráficos están a una escala más que pequeña que algunos de los que siguen a continuación. En particular, estos gŕaficos pueden resultar medio caóticos, pero dejan en claro que las curvas no son suaves sino que poseen varios picos. En esta escala pequeña queda un poco más claro el criterio de elección que hemos realizado, eligiendo en cada caso la curva y los α y β que nos pareció que convergieron a un velocidad razonables y que además fueron los que más se estabilizaron cerca del RTT promedio.

Es fácil también observar que todos decaen rápidamente de su valor inicial y se acercan al RTT y luego van fluctuando alrededor de dicho valor.

Los siguientes dos gráficos tienen una escala un poco más grande. En ellos se puede ver claramente como la estimación del RTO va decreciendo hasta acercarse a un valor parecido al RTT para luego ir fluctuando cerca de ese valor:

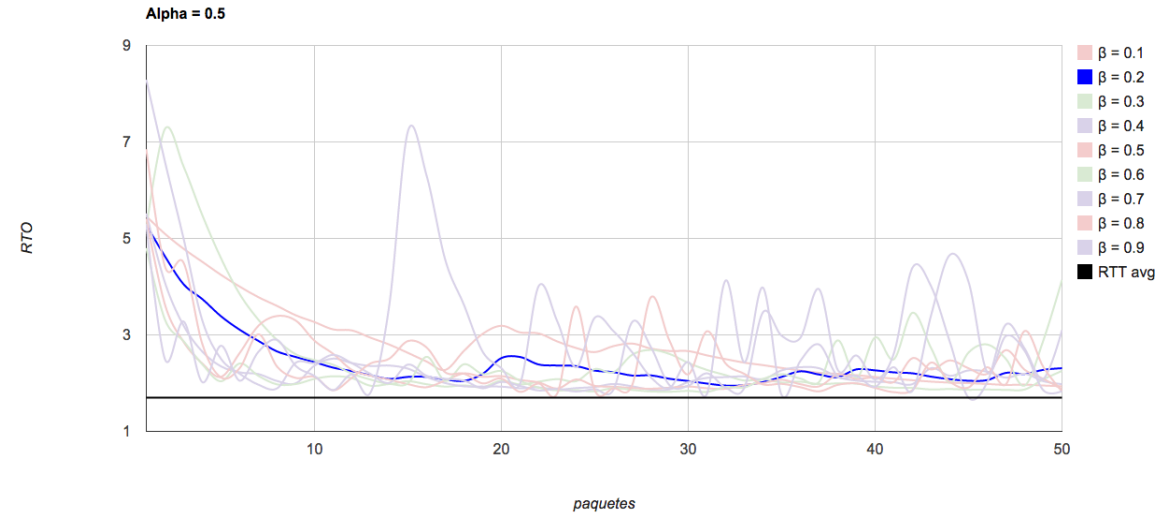

Gráfico interactivo en: http://goo.gl/VNV8xF

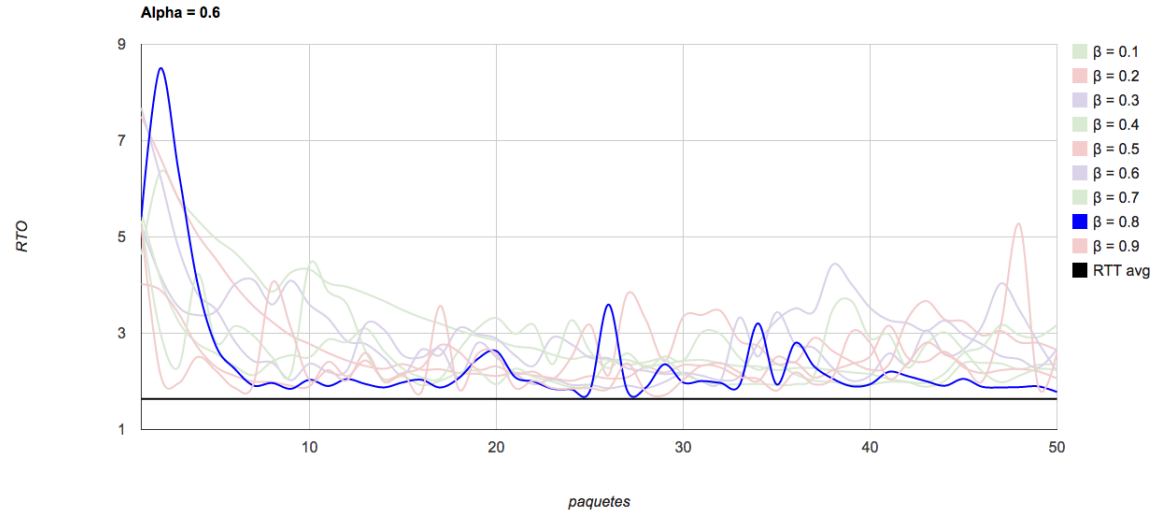

Gráfico interactivo en: http://goo.gl/CN4joc

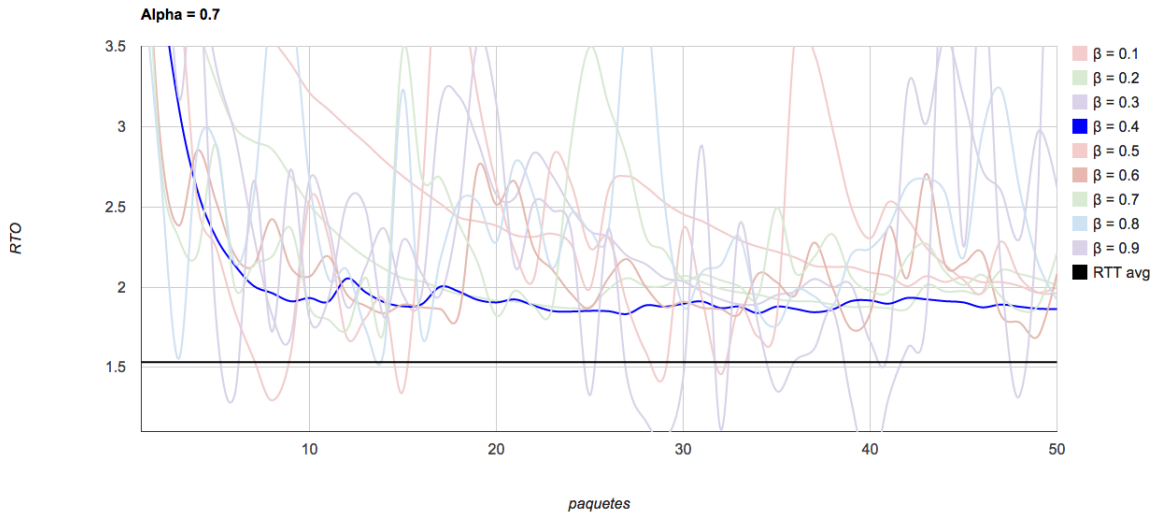

Gráfico interactivo en: http://goo.gl/Gc93dU

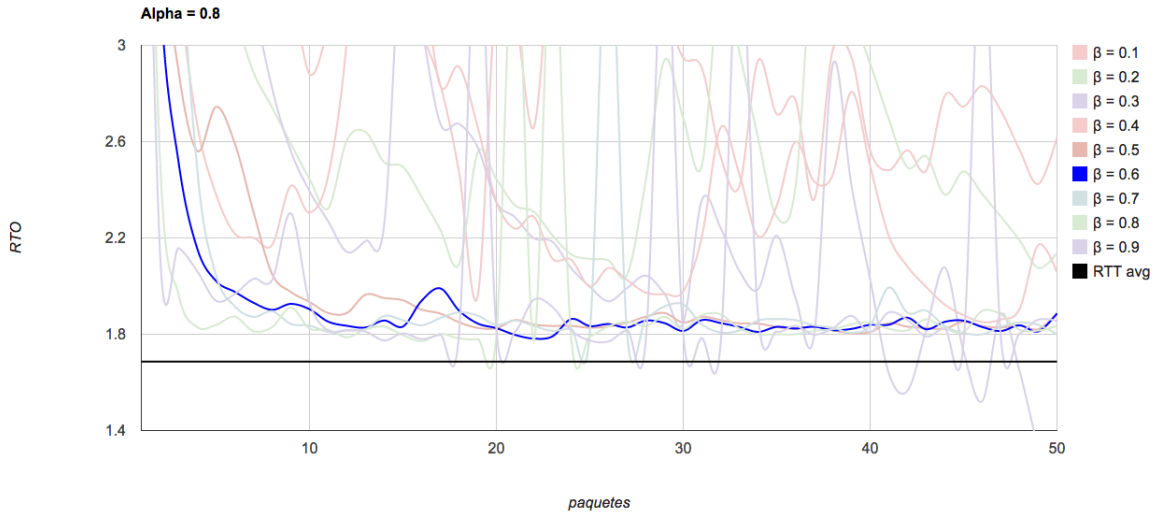

Gráfico interactivo en: http://goo.gl/RGiQc0

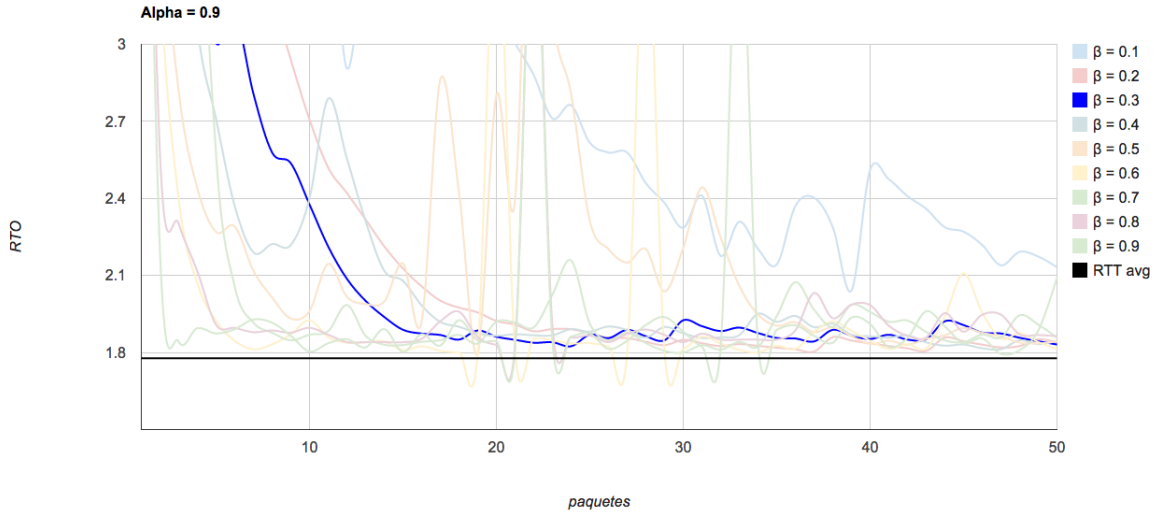

Gráfico interactivo en: http://goo.gl/QXH7cA

El siguiente gŕafico se corresponde con la contraposición de las curvas correspondientes a las α y β elegidos de cada uno de los gŕafico.Comparando los resultados, concluimos que para nuestra experimentación los valores que mejores resultados nos dieron fueron $\alpha = 0{,}4$ y $\beta = 0{,}7$.

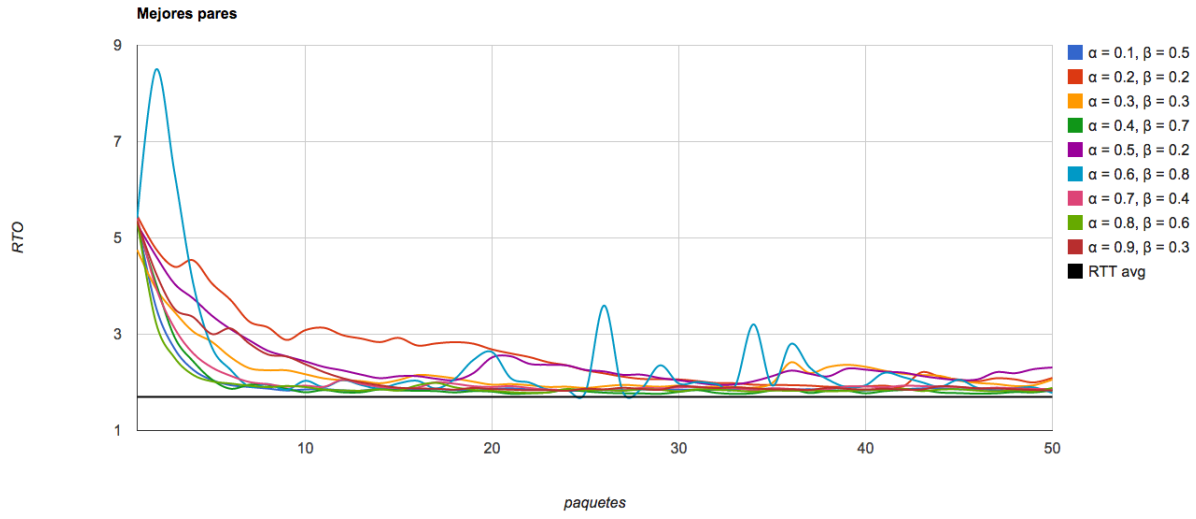

Gráfico interactivo en: http://goo.gl/FBA1p2

El gráfico anterior lo dejamos para dimensionar la convergencia y los valores en general. El siguiente gráfico es el que realmente justifica que de todas las opciones, la anteriormente mencionada sea la elegido, ya que es la que más cerca se encuentra del promedio de RTT, y además, la que más estable mantiene dicha cercanía, aunque en contraposición parece converger un poco más lentamente que la mayoría de las demás curvas del gráfico.

Este resultado parece chocar con los valores recomendados de $\alpha = 0{,}125$ y $\beta = 0{,}25$ como valores óptimos a utilizarse que se proponen en el RFC que refiere al cálculo del RTO. Si bien en esta etapa no se exper imentó con dichos valores, era sensato imaginar que de todas las corridas las que se correspondían con los α y β más cercanos a tales valores serían las que mejores resultados proveyeran.

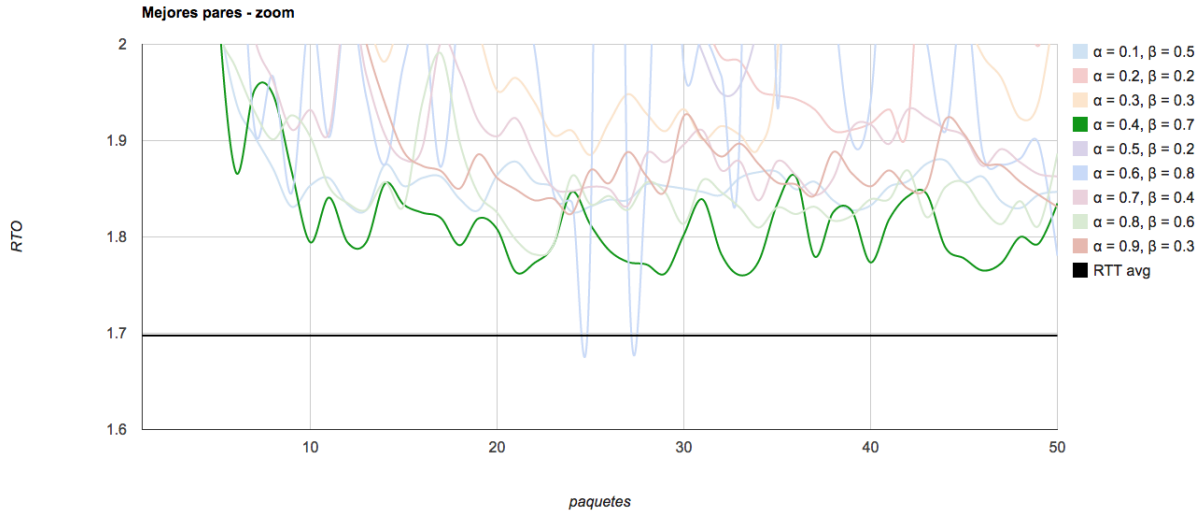

Gráfico interactivo en: http://goo.gl/RvuVpr

Para los próximos experimentos, utilizaremos estos que hemos determinado como mejores pares de α y β para analizar como se comportan los mismos cuando se agrega la nueva variable de **probabilidad de dropeo** de un paquete.

3.2. Experimento 2

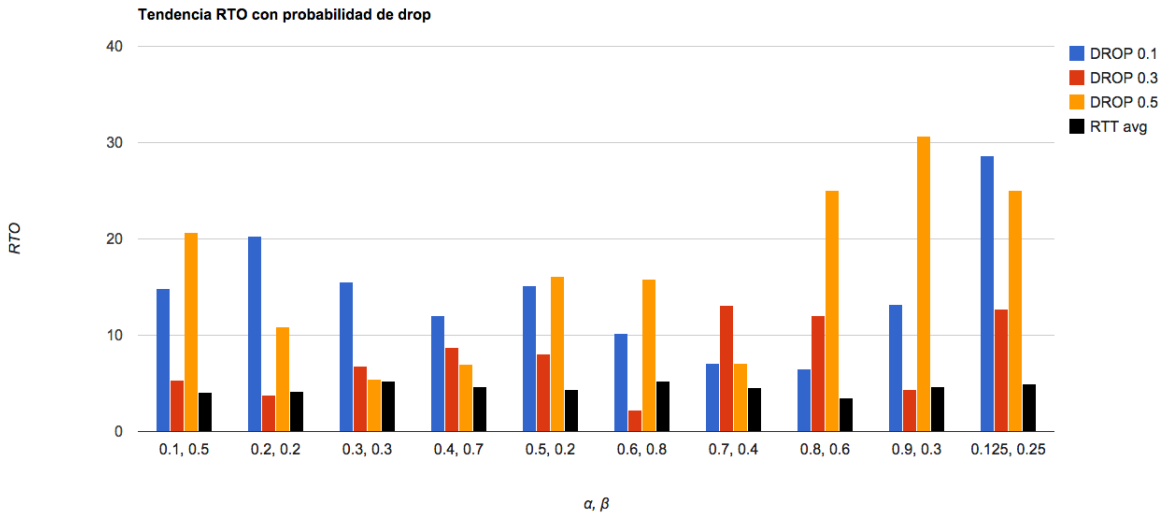

Gráfico interactivo en: http://goo.gl/6aM368