# Temporizador de retransmisiones

Elaborado por: Ukranio Coronilla

Hasta ahora para el *timeout* utilizamos n segundos para esperar que una solicitud sea respondida, y si no sucede realizar la retransmisión. El problema en este caso es que el tiempo puede ser muy grande, y optimizarlo brindaría ventajas a la aplicación distribuida aumentando el desempeño.

Por esta razón el *timeout* debe ser un valor variable que se vaya ajustando a las condiciones de la red, como por ejemplo la cantidad de tráfico, la distancia y la velocidad de transmisión de la propia red. En la actualidad existen algunos algoritmos donde se hace una estimación del tiempo t en el cual podría llegar la siguiente respuesta.

El algoritmo de Jacobson es uno de los más importantes y se centra en el control de flujo TCP/IP, también se dice que salvó el colapso de internet a finales de los 80s y principio de los 90s. Su trabajo completo se encuentra en este servidor con el nombre de congavoid.pdf por si desea conocer su deducción. La explicación breve del algoritmo se da a continuación:

El valor estimado del *timeout* ***se va a recalcular con cada nueva recepción de datos*** para utilizarlo en la siguiente solicitud. En el algoritmo de Jacobson al *timeout* se le conoce como RTO (Temporizador de retransmisiones).

Para la primera estimación de RTO (cuando no se ha enviado ningún mensaje) utilizaremos:

```
RTO = srtt + 2*rttvar
```

donde `srtt` = 0 segundos y `rttvar` = 3 segundos

`srtt` es la media estimada de RTT y `rttvar` es la desviación estándar de la RTT.

Importante: RTT(Round-Trip delay Time) es el tiempo que tarda un paquete de datos enviado desde un emisor en volver a este mismo emisor habiendo pasado por el receptor de destino.

Después de la primera estimación se deberá calcular RTT y su valor substituirá a la variable `RTTmedida` en la siguiente ecuación. Observe que en esta primera estimación `srtt` vale cero pero no sucederá así en las subsiguientes.

```
delta = RTTmedida - srtt
```

Con este valor se obtienen los nuevos valores de `srtt` y `rttvar` substituyendo en las siguientes formulas:

```
srtt <- srtt + g * delta
rttvar <- rttvar + h (|delta| - rttvar)
```

donde el valor de g es 1/8 y h es 1/4.

Finalmente la nueva estimación de `RTO` se calculará con:

```
RTO = srtt + 4*rttvar
```

Este proceso se debe repetir para estimar cada RTO (timeout) subsiguiente.

## Ejemplo:

Antes de enviar una solicitud el cliente hace un estimado de RTO (timeout):

RTO = 0 + 2s * 3s = 6 segundos.

Al enviar la primera solicitud el cliente mide la RTT, y supongamos que es de 409 microsegundos, entonces:

```
delta = 409us – 0 = 409us
srtt <- 0 + 0.125 * 409us = 51.125us
rttvar <- 3s + 0.25* (|409us| - 3s) = 2250102.25 us
RTO = 51.125us + 4 * 2250102.25 us = 9000460.125 us = 9 segundos y 461 us
```

En la segunda iteración se ajusta el timeout a 9 segundos y 460 microsegundos, y supongamos que la medir RTT obtenemos 277 microsegundos, entonces:

```
delta = 277us – 51.125us = 225.875us
srtt <- 51.125us + 0.125 * 225.875us = 79.359us
rttvar <- 2250102.25 us + 0.25* (|225.875us| - 2250102.25 us) = 1687633.156 us
RTO = 79.359us + 4 * 1687633.156 us = 6750611.983 us = 6 segundos y 750612 us
```

En la tercera iteración se ajusta el timeout a 6 segundos y 750611 microsegundos, y continuamos así con el algoritmo en cada solicitud sucesiva.

1.- Hasta el momento el método setTimeout arma la señal SIGALRM en segundos mediante alarm() o en microsegundos con ualarm(). Por ello es importante modificarla para que acepte tiempos que combinan segundos con microsegundos. El siguiente es un ejemplo de uso de la función setitimer() muy similar al presentado en el programa 11-1 del manual “Programación de Sistemas Linux”:

```
#include <stdio.h>
#include <signal.h>
```

```
#include <sys/time.h>
#include <unistd.h>

void tratar_alarma(void)
{
   printf("Alarma activada \n");
}


int main(void)
{
   struct sigaction act;
   sigset_t mask;
   struct itimerval intervalo;

   intervalo.it_interval.tv_sec = 0;
   intervalo.it_interval.tv_usec = 0;
   intervalo.it_value.tv_sec = 2;
   intervalo.it_value.tv_usec = 500000;

   /* especifica el manejador */
   act.sa_handler = (void *)tratar_alarma; /*funcion a ejecutar */
   act.sa_flags = 0; /* Ninguna accion especifica */

   /* Se bloquea la señal 3 SIGQUIT */
   sigemptyset(&mask);
   sigaddset(&mask, SIGQUIT);
   sigprocmask(SIG_SETMASK, &mask, NULL);
   sigaction(SIGALRM, &act, NULL);
   for(;;)
   {
      setitimer(ITIMER_REAL, &intervalo, NULL);
      pause(); /* se suspende el proceso hasta que se reciba la señal */
   }
}
```

Modifique los métodos setTimeout y unsetTimeout para que incluyan esta nueva característica.

2.- Elabore un cliente que realiza n solicitudes repetidas al servidor de la práctica 9 “Servidor de dominio Internet tipo UDP”, donde n se recibe como parámetro en la línea de comandos. En cada solicitud debe imprimir las variables `RTTmedida` , `delta`, `srtt`, `rttvar`, `srtt` estimada, `rttvar` estimada y `RTO` estimado tal y como lo sugiere el algoritmo de Jacobson. Todas las variables se imprimen en microsegundos salvo la primera `RTTmedida` y `RTO` estimado que se deben imprimir con ambos campos de segundos y microsegundos.

En caso de que un mensaje se pierda, deberá realizar la retransmisión del paquete pero esta vez con el timeout estimado por el algoritmo de Jacobson.

Es muy importante obtener una buena precisión en los cálculos, por ello se recomienda utilizar el tipo de datos `long double` en todas las operaciones y aplicar la función ceill() en la última ecuación para encontrar los microsegundos de RTO.

3.- Para simular la perdida de paquetes modifique el servidor para que reciba en la línea de comandos el porcentaje de paquetes que se van a perder (el servidor no responderá a dichos paquetes) de modo que si recibe un 10 como parámetro, significará que solo responderá 90 de cada 100 paquetes elegidos al azar (use la función `rand`() para dicha simulación).

*Observaciones: Debe tener especial cuidado con timersub() pues al calcular delta puede devolver un valor erróneo cuando el tiempo final es menor que el tiempo inicial (realice una prueba de ello). En estos casos y cuando se saque el valor absoluto, se sugiere utilizar una sola variable para almacenar el tiempo total en microsegundos.*