Sistemas Telemáticos
Práctica 4: Control de Congestión en TCP

GSyC
Departamento de Teoría de la Señal y Comunicaciones y Sistemas Telemáticos y Computación
URJC

Marzo de 2018

1. Estados de una conexión TCP

En este apartado se observan los estados por los que va pasando un extremo de una conexión TCP.
Descomprime el escenario `estadosTCP-lab.tgz` y arranca las máquinas.

1.1. Establecimiento de la conexión

Una aplicación que funciona como servidor TCP comienza su ejecución aceptando conexiones de clientes. Hasta que haya alguna conexión de algún cliente la aplicación servidor TCP se encontrará en estado `LISTEN` y cambiará de estado en el momento en que el cliente se conecte.

Ve realizando los siguientes pasos, y responde en su momento a las preguntas que se indican:

1. Ejecuta `tcpdump` en `r1` en su interfaz `eth0`, no es necesario que guardes la captura en un fichero, especifica que sólo capture el tráfico TCP de la siguiente forma:

   ```
   tcpdump -i eth0 tcp
   ```

   Ensancha la ventana de `r1` todo lo que puedas para ver cada segmento capturado en una línea. Para interpretar la salida de `tcpdump` utiliza la información de la figura 1, al final del enunciado.

2. Arranca una aplicación servidor TCP que espere conexiones en el puerto 7777 en `pc2` utilizando la herramienta *netcat*, de la siguiente forma[1]:

   ```
   nc -l -p 7777 &
   ```

3. Para ver en qué estado se encuentran las conexiones TCP que tiene abiertas una máquina utilizaremos el comando `netstat -nat`. Para ejectuar repetidamente un comando utilizaremos el comando `watch`. Así, para comprobar el estado de la conexión TCP en el lado del servidor ejecuta en `pc2`:

   ```
   watch -n 0.5 netstat -nat
   ```

   La opción `-n` de `watch` permite especificar cada cuanto tiempo se repite la ejecución del comando que sigue a continuación; en este caso, cada 0.5 segundos.

4. Coloca en la pantalla las ventanas de los terminales de `pc1` y `pc2` para que puedas visualizarlas simultáneamente.

5. Explica en qué estado se encuentra la conexión en el servidor antes de arrancar el cliente.

6. Lanza en `pc1` una aplicación cliente TCP que se conecte al servidor de `pc2`:

   ```
   nc 12.0.0.2 7777
   ```

[1] La herramienta *netcat* (`nc`) permite lanzar de forma simple clientes o servidores de TCP o UDP, y se estudió en la asignatura de Arquitectura de Redes de Ordenadores.

7. Explica qué estados atraviesa el servidor hasta que el cliente se encuentra conectado [2].

8. Observa en el tráfico capturado en `r1` como se han intercambiado los 3 segmentos del establecimiento de la conexión.

9. Explica el valor de la ventana anunciada por el servidor.

10. Interrumpe la ejecución del comando `watch` en `pc2` con `Ctrl+C` y trae a primer plano de ejecución el comando `nc` en `pc2` ejecutando `fg`.

1.2. Intercambio de datos

Desde el cliente en `pc1` podemos escribir tantos datos como queramos por la entrada estándar para enviar a `pc2`. Vamos a comprobar que los datos que envía `pc1` a `pc2` se quedan almacenados en el buffer de recepción (*in*) de la conexión en el lado del servidor en `pc2` hasta que la aplicación servidor los lea.

Ve realizando los siguientes pasos, y responde en su momento a las preguntas que se indican:

1. Si has interrumpido la ejecución de `tcpdump` en `r1` en su interfaz `eth0`, vuelve a lanzarlo.

2. Detén la ejecución de `nc` en el lado servidor con `Ctrl+z`. Esto provocará que la ejecución del proceso servidor quede suspendida (pero no finalizada), y por tanto no realizará llamadas al sistema: en particular no leerá la información que reciba de TCP, que se irá acumulando en el buffer de recepción (*in*) de la conexión TCP.

3. Ejecuta en el lado servidor de nuevo el comando `netstat` de la siguiente forma:

```
watch -n 0.5 netstat -nat
```

4. En el proceso `nc` del cliente en `pc1` introduce una cadena de caracteres larga por la entrada estándar (por ejemplo pulsa un rato la tecla de alguna letra hasta que aparezcan 2 líneas enteras de esa letra por pantalla y después pulsa la tecla `Enter`). Esta cadena de caracteres se recibirá en la implementación de TCP en el lado servidor, pero la aplicación no leerá estos datos porque se encuentra detenida.

5. Fíjate en cómo los datos se han quedado almacenados en el *buffer* de recepción (*in*) del lado servidor (aparece etiquetado como `Recv-Q`).

   Explica el valor de `Recv-Q` teniendo en cuenta que has pulsado más caracteres para enviar en el cliente.

   Explica los segmentos enviados por el cliente que muestra la captura de tráfico de `r1`.

6. Interrumpe con `Ctrl+c` la ejecución de `netstat` en el lado servidor. Trae a primer plano la ejecución de `nc` en `pc2` (`fg`) y la ejecución de la aplicación servidor continuará. Observa cómo la aplicación servidor lee los datos del *buffer* de recepción (*in*) de la conexión y los muestra en la pantalla.

1.3. Finalización de la conexión

Ve realizando los siguientes pasos, y responde en su momento a las preguntas que se indican:

1. Si has interrumpido la ejecución de `tcpdump` en `r1` en su interfaz `eth0`, vuelve a lanzarlo.

2. Interrumpe la ejecución del cliente en `pc1` con `Ctrl+c`. La aplicación cliente mandará un segmento con el flag FIN activado.

3. El servidor `nc` está programado para terminar si el cliente le manda un segmento con el flag FIN activado, por este motivo, la aplicación `nc` en `pc2` finaliza la conexión mandando su segmento FIN+ACK. Cuando el servidor recibe el último ACK de `pc1` da por terminada la comunicación y finaliza su ejecución.

4. En `pc1` aunque la aplicación ha terminado, los recursos de la conexión TCP tardan un tiempo en liberarse y si ejecutas en `pc1`:

```
watch -n 0.5 netstat -nat
```

[2] Hemos añadido retardos a la interfaz de red de pc1 para que puedas visualizar esta transición entre estados ya que el cambio de estado normalmente se produce muy rápido y se aprecia fácilmente.

observarás que la conexión TCP tarda un tiempo en liberarse y fíjate en el estado en el que se encuentra.

Explica por qué se mantiene tanto tiempo en este estado.

5. En `pc2` la aplicación ha terminado y se han liberado inmediatamente los recursos de la conexión TCP. Si ejecutas en `pc2`:

```
watch -n 0.5 netstat -nat
```

observarás que ya no hay conexiones TCP abiertas en `pc2`.

Explica por qué ya no aparecen conexiones TCP en `pc2`.

2. Slow Start en el inicio de la conexión

En este apartado se observa el comportamiento del mecanismo de TCP arranque lento (*Slow Start*).

Carga en *Wireshark* la captura `slow-start.cap`. Ordena los paquetes por la columna `Time`. Selecciona el primer segmento que envía el cliente al servidor, y a continuación muestra en *Wireshark* el diagrama de la conexión mediante el menú: *Statistics→TCP Stream Graph→Time-Sequence Graph (tcptrace)*

Observa la gráfica de la captura, y cada uno de los segmentos. Trata de entender cómo afecta *Slow Start* el envío de nuevos segmentos desde que comienza la conexión.

Responde a las siguientes preguntas:

1. Indica el valor del *flightsize* en los siguientes instantes: 1'5s, 2'5s, 3'5s, 4'5s, 5s, 6s.
2. ¿En qué instantes de tiempo de esta captura es menor la ventana de control de flujo que la ventana de control de congestión?
3. ¿Por qué se envían inicialmente 3 segmentos con datos?
4. ¿Cuántos segmentos con nuevos datos se podrían enviar en el instante 3'5s?
5. ¿Cuántos segmentos con nuevos datos se podrían enviar en el instante 4'5s?

3. Control de Congestión tras Timeout

En este apartado se observa el comportamiento del mecanismo de control de congestión de TCP tras un *timeout*.

Carga en *Wireshark* la captura `ss-timeout.cap`. Ordena los paquetes por la columna `Time`.
Si en la columna "Protocol"ves en algunos paquetes el valor "Gryphon", desactiva el análisis de dicho protocolo en el menú: *Analize→Enabled Protocols*.
Selecciona el primer segmento que envía el cliente al servidor, y a continuación muestra en *Wireshark* el diagrama de la conexión mediante el menú: *Statistics→TCP Stream Graph→Time-Sequence Graph (tcptrace)*

1. Observa la captura y analiza cómo afecta *Slow Start* al envío de nuevos segmentos al inicio de la conexión. Entre el instante 3s y 3'5s se envían 6 segmentos. ¿Podrían haberse enviado más segmentos en ese instante? Razona la respuesta.
2. Localiza la retransmisión que se produce alrededor del instante 7'5s. Estudia el comportamiento de TCP tras el *timeout*. Analiza tanto la gráfica como cada uno de los segmentos. Responde a las siguientes preguntas:
   a) ¿Qué número de secuencia tiene el segmento retransmitido?
   b) ¿Qué valor se calcula para el umbral *ssthres*?
   c) ¿Qué tamaño tiene la ventana de congestión en el instante 8s?
   d) ¿Por qué se envían 2 segmentos alrededor del instante 9s?
   e) ¿Podría enviarse algún segmento más en el instante 9s? ¿Por qué?
   f) ¿En qué modo de control de congestión se haya la conexión en el instante 5s? ¿Por qué?
   g) ¿En qué modo de control de congestión se haya la conexión en el instante 10'5s? ¿Por qué?
3. ¿En qué modo de control de congestión termina la conexión?
4. ¿Qué valor aproximado tiene la ventana de congestión al final de la conexión?

4. Fast Retransmit / Fast Recovery

En este apartado se observa el comportamiento de TCP tras la recepción de 3 ACKs duplicados. Carga en wireshark la captura `fr-fr.cap` y ordena los paquetes de la captura según la columna `Time`.

Para identificar si una retransmisión es debida a *Fast Retransmit* comprueba las siguientes condiciones:

- Anteriormente a la retransmisión hay 4 ACKs (1 + 3 duplicados) o más sobre el número de secuencia que se está retransmitiendo (condición necesaria)
- Se envían más datos nuevos después de realizar la retransmisión y antes de recibir su ACK (condición suficiente)

Wireshark interpreta los segmentos TCP de las capturas e identifica si una retransmisión es debida a *Timeout* o *Fast Retransmit*. Sin embargo, a veces la captura no está bien ordenada según el eje temporal y las interpretaciones que hace *Wireshark* no son correctas. Por este motivo para saber por qué se produce una retransmisión es necesario analizar despacio los paquetes presentes en la captura.

Observando la captura, responde a las siguientes preguntas:

1. ¿Cuántas retransmisiones debidas a *timeout* se observan en la traza? Identifica en qué instante se producen.
2. ¿Cuántas retransmisiones debidas a *Fast Retransmit* se observan en la traza? Identifica en qué instante se producen.
3. Observa la primera ocasión en la que se produce *Fast Retransmit*. ¿En qué modo de control de congestión se entra tras efectuarse dicha retransmisión?
4. Poco después de producirse esa primera retransmisión debida a *Fast Retransmit* se envían nuevos segmentos de datos antes de recibir ningún ACK. ¿Cuántos nuevos segmentos de datos se envían? ¿Por qué se pueden enviar en ese instante? ¿Se podría enviar alguno más en ese periodo hasta que llegue el ACK del paquete retransmitido?
5. Observa en qué momento llega el primer ACK nuevo. ¿En qué modo de control de congestión se entra tras recibirse dicho ACK nuevo? Indica cuántos paquetes se envían en ese momento, y explica la razón de este número. Observa la evolución de la ventana de congestión desde este momento hasta la siguiente retransmisión.
6. Observa la segunda ocasión en la que se produce *Fast Retransmit*. Es en el paquete 114, aunque Wireshark no lo identifica correctamente. Estudia la evolución del control de congestión tras esta retransmisión.
7. Observa las otras 2 retransmisiones, que teambién son *Fast Retransmit*. Para cada una de ellas, estudia la evolución del control de congestión después de dicha retransmisión.

5. Control de Congestión y sondas de ventana

En este apartado se observa la interacción entre los mecanismos de control de flujo y de control de congestión de TCP.

Carga en wireshark la captura `sondas.cap`.

Observa la captura y comprueba el comportamiento de la ventana anunciada a partir del segmento 43. Responde a las siguientes preguntas:

1. Localiza en la traza los anuncios de ventana de tamaño 0 enviados por el servidor al cliente. ¿Qué segmentos son los que transportan estos anuncios?
2. Observa las sondas de ventana que envía el cliente en ese periodo. ¿Qué segmentos son los que transportan las sondas de ventana?
3. Estudia el comportamiento de TCP entre los segmentos 43 y 87. Responde a las siguientes preguntas:
   a) Indica cuál es el número de byte más alto que puede enviarse tras recibirse los siguientes segmentos: 41, 43, 51, 79.x
   b) ¿Cuántos bytes nuevos pueden enviarse tras recibirse el segmento 80?
   c) ¿Cuántos bytes de datos transportan los segmentos con sondas de ventana?
   d) Tras haberse cerrado la ventana, ¿en qué segmento vuelve a abrirse? ¿Cuál es el tamaño de la nueva ventana anunciada?

e) Tras recibirse el segmento 84, ¿qué limita al emisor, la ventana de control de flujo o la de congestión?

4. Indica en qué modo de control de congestión se encuentra la conexión TCP en los siguientes puntos:

   a) Antes del segmento 43.
   b) Entre el segmento 43 y 87
   c) Después del segmento 87

5. Indica en qué momentos de la conexión es la ventana de control de flujo la que limita al emisor, y en cuáles es la ventana de control de congestión.

6. ACKs selectivos (SACK)

En este apartado se observa el comportamiento de SACKs en TCP, en la implementación *New Reno*.

Carga en *Wireshark* la captura `sack.cap`. Es muy importante que ordenes los segmentos según la columna `Time`, ya que Wireshark los tiene ordenados según el número de segmento.

Podrás encontrar los segmentos que contienen asentimientos selectivos (SACK) en la captura si en la caja `Filter` de Wireshark escribes `tcp.options.sack` y pulsas `<Intro>`. Además, en versiones antiguas de wireshark (versiones anteriores a la 2.0.0, como las que existían en distribuciones Linux basadas en Ubuntu 14.04 o anteriores), en la gráfica *tcptrace* se identifican fácilment la información de los asentimientos selectivos mediante trazos azules. En futuras versiones de wireshark (versiones posteriores a la 2.3.0) se identificarán también estos segmentos en la gráfica mediante trazos rojos.

NOTA: En distribuciones Ubuntu o Linux Mint recientes, si ejecutas `wireshark-gtk` en vez de `wireshark` (tras instalar el paquete con `sudo apt-get install wireshark-gtk`), podrás acceder a las gráficas *tcptrace* antiguas en vez de a las modernas.

Responde de forma razonada a las siguientes preguntas:

1. Dado que SACK es una opción que pueden usar ciertas implementaciones de TCP, indica cómo indica un extremo de la conexión que acepta asentimientos selectivos del otro extremo. ¿En qué segmentos de la conexión ocurre ese acuerdo? ¿Qué número de opción de TCP se utiliza?

2. Indica cuál es el primer segmento de la captura donde se muestran asentimientos selectivos. ¿En qué lugar del segmento va esta información? ¿Qué número de opción se utiliza? ¿Qué números de secuencia se están asintiendo y qué números de secuencia se asienten selectivente? ¿Qué números de segmento son los que se están asintiendo, y cuáles se asienten selectívamente?

3. Explica si el segmento 78 es una retransmisión por timeout o es una retransmisión rápida.

4. Explica en qué modo de control de congestión está la máquina 11.0.0.10 justo después de haber enviado el segmento 78. Indica el valor de `threshold` en ese instante.

5. Justo antes de enviar el segmento 83, ¿qué segmentos puede suponer la máquina 11.0.0.10 que le faltan a 13.0.0.10?

6. Si 11.0.0.10 sabe que a 13.0.0.10 le faltan varios segmentos, ¿por qué justo después de enviar el segmento 83 no retransmite todos los que sabe que le faltan?

7. Explica si el segmento 262 es una retransmisión por timeout o es una retransmisión rápida.

8. Explica en qué modo de control de congestión está la máquina 11.0.0.10 justo después de haber enviado el segmento 262. Indica el valor de `threshold` en ese instante.

9. Al recibir el segmento 257, ¿qué segmento/s puede suponer la máquina 11.0.0.10 que le falta/n a 13.0.0.10?

10. A recibir el segmento 259, ¿qué segmento/s puede suponer la máquina 11.0.0.10 que le falta/n a 13.0.0.10?

11. Fíjate en la diferencias que hay entre el segmento 257 y 259. ¿Qué crees que ha provocado que la máquina 13.0.0.10 haya enviado ese asentimiento?

12. A recibir el segmento 277, ¿qué segmento/s puede suponer la máquina 11.0.0.10 que le falta/n a 13.0.0.10?

13. Al recibir el segmento 669, ¿qué segmento/s puede suponer la máquina 11.0.0.10 que le falta/n a 13.0.0.10?

14. Explica si el segmento 747 es una retransmisión por timeout o es una retransmisión rápida.

15. ¿Por qué hay 2 retransmisiones juntas (segmentos 747 y 749)?

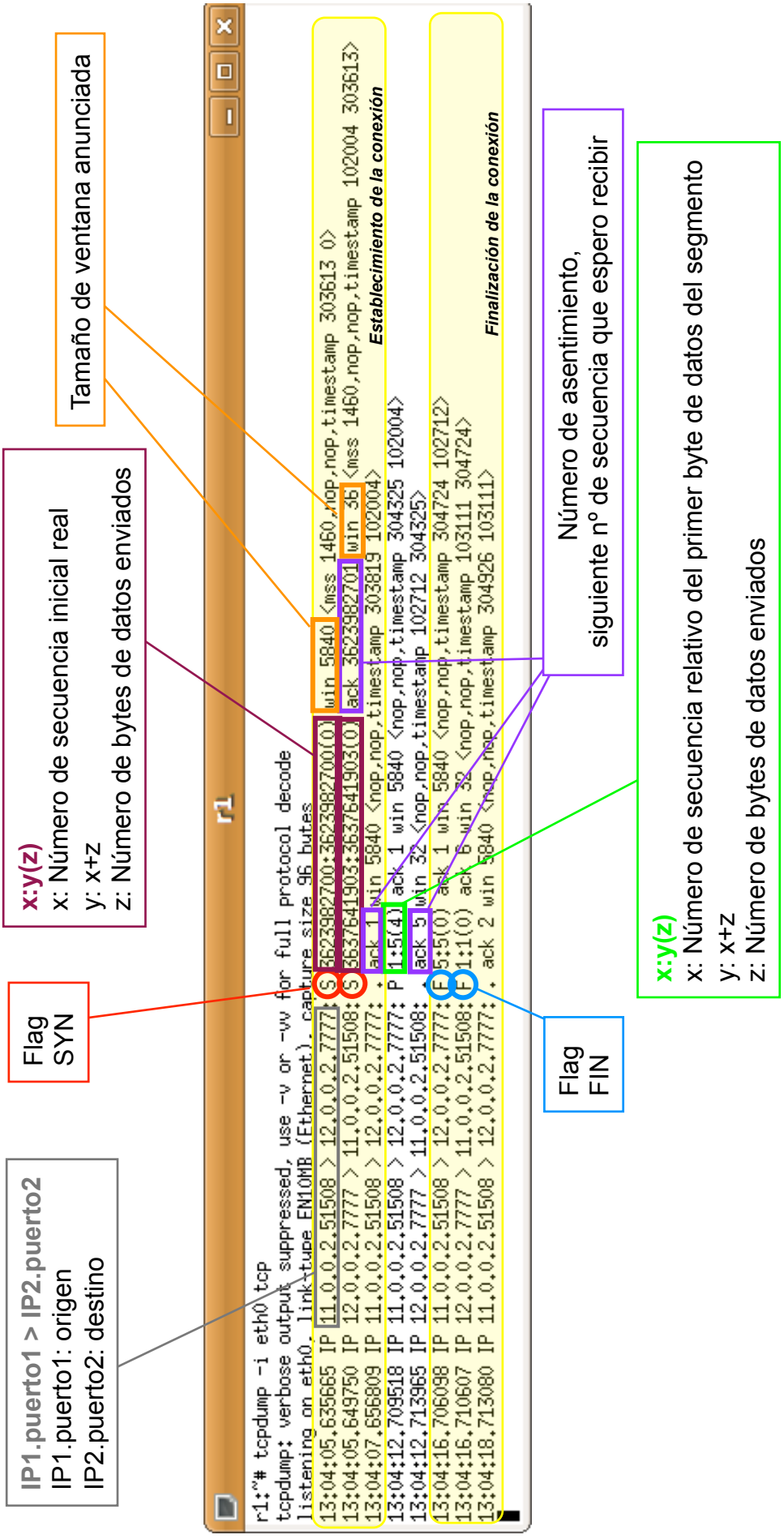

Figura 1: Conexión TCP utilizando tcpdump directamente en el terminal