Práctica 8: Análisis del protocolo TCP

Trabajo previo

Para el correcto aprovechamiento de la práctica es conveniente que antes de acudir al laboratorio hayas estudiado el funcionamiento del protocolo TCP. Para ello, además de repasar los conceptos vistos en las clases de teoría y problemas, también debes estudiar el **capítulo 3 de Kurose**. Además, como en esta práctica se va a usar de forma intensiva el analizador de protocolos Wireshark, debes repasar el contenido de las prácticas en las que hemos empleado este analizador.

Objetivos de la práctica

Al acabar esta práctica serás capaz de interpretar en una captura de tráfico los mecanismos fundamentales involucrados en el funcionamiento del protocolo TCP, como el establecimiento y cierre de la conexión, el uso de las opciones TCP, funcionamiento de los reconocimientos y aspectos del control de congestión. También utilizarás nuevas herramientas del analizador de protocolos Wireshark que te permitirán crear gráficos para mostrar el intercambio de segmentos TCP. Y además constataremos algunas actualizaciones al comportamiento teórico básico que hemos analizado en las clases de teoría.

Hay que tener en cuenta que TCP, uno de los protocolos más importantes de Internet, se ha ido refinando y optimizando año tras año. En el año 1994 la pila TCP de Linux tenía 3000 líneas de código. Creció hasta las 18000 líneas en el año 2006 y en la actualidad está por encima de las 80000 (en lenguaje C). Esto hace que, por encima del funcionamiento teórico recogido en los estándares de TCP que hemos estudiado en clase, que siguen vigentes, existan un conjunto de técnicas y algoritmos adicionales que han optimizado su funcionamiento en nuestros días. Destacaremos algunas (muy pocas) de estas actualizaciones que te ayudarán a reflexionar sobre la complejidad del nivel de transporte y el apasionante mundo que esconde.

Establecimiento, cierre de la conexión TCP y ventana de recepción

Vamos a comenzar la práctica analizado tráfico previamente capturado para repasar todo lo que hemos tratado en las sesiones de teoría. Se trata de la descarga de la página www de la Universidad, concretamente el `GET /es/` que hemos realizado en algunas sesiones previas. Por ello, descarga el fichero **`UPVGet1.pcapng`** que está disponible en Poliformat y cárgalo en Wireshark. Ahora vamos a analizar tráfico TCP de este flujo con el objetivo de revisar el establecimiento de la conexión y las opciones incluidas en los segmentos de `SYN`. También vamos a ver cómo cada extremo de la conexión usa el *flag* `FIN` para indicar que desea cerrar la conexión y analizaremos el tamaño de la ventna de recepción.

Ejercicio 1:

a) Analiza el establecimiento de la conexión e identifica el protocolo en tres fases. ¿Qué MSS se elige para realizar la transferencia? ¿En qué segmentos aparece la opción MSS? ¿Qué otras opciones establecen cada uno de los extremos? ¿Siguen apareciendo una vez establecida la conexión?

b) Determina los números de secuencia iniciales de cada extremo. Como se ha explicado en las clases de teoría, cada extremo de la comunicación genera cada vez que se establece una nueva conexión un número de 32 bits como número de secuencia inicial, obtenido, principalmente, a partir de un valor de un temporizador local más una función pseudoaleatoria calculada sobre los parámetros de la conexión -direcciones IP origen y destino y puertos origen y destino - y una clave secreta). De esta forma se garantiza que no se va a repetir un número de secuencia inicial en instantes de tiempo próximos. Este número se da a conocer al otro extremo cuando se envía el segmento de inicio de conexión con el bit de `SYN` activo. A partir de esos números iniciales elegidos por cada extremo de la comunicación, éstos se van incrementando en función del intercambio de segmentos que se realice. Para facilitar el seguimiento de los números de secuencia, Wireshark, por defecto, resta a todos los números de secuencia el valor inicial, por lo que nos muestra unos **números de secuencia relativos** (es decir, como si empezaran con valor igual a 0 en el inicio de la conexión en ambos extremos). No obstante, es interesante comprobar el valor real o **absoluto**, que es el que está encapsulado en la estructura del segmento TCP. Para ello, cuando seleccionas un segmento y vemos los distintos niveles en la ventana intermedia con un mayor nivel de detalle (como vimos en la práctica 1), si desplegamos la información relativa al nivel TCP, podemos observar la información que nos proporciona Wireshark de, entre otros campos, los números de secuencia. En la Figura 1 hemos destacado las primeras palabras de la estructura del segmento TCP, que como sabéis están constituidas por los números de puerto (origen y destino, de 16 bits cada uno) y el número de secuencia. Tal como se muestra en la Figura 1, Wireshark nos detalla el número de secuencia relativo (etiquetado como *Sequence number*) del segmento dentro de su flujo y el absoluto *(Sequence number -raw-)* que es el real que se transmite en el segmento. Por último, Wireshark permite indicar, además, de forma explícita cómo queremos ver los números de secuencia (también afecta a lo que se visualiza en la ventana superior). Así, seleccionando un segmento TCP, con el botón derecho del ratón (***Protocol preferences***) podemos indicarle que queremos usar o no números relativos (ver Figuras 2 y 3). Si desactivamos esa opción veremos en todos los segmentos de la ventana superior el número de secuencia absoluto de cada segmento, aunque es más cómodo seguir usando en el resto de la práctica los números relativos.

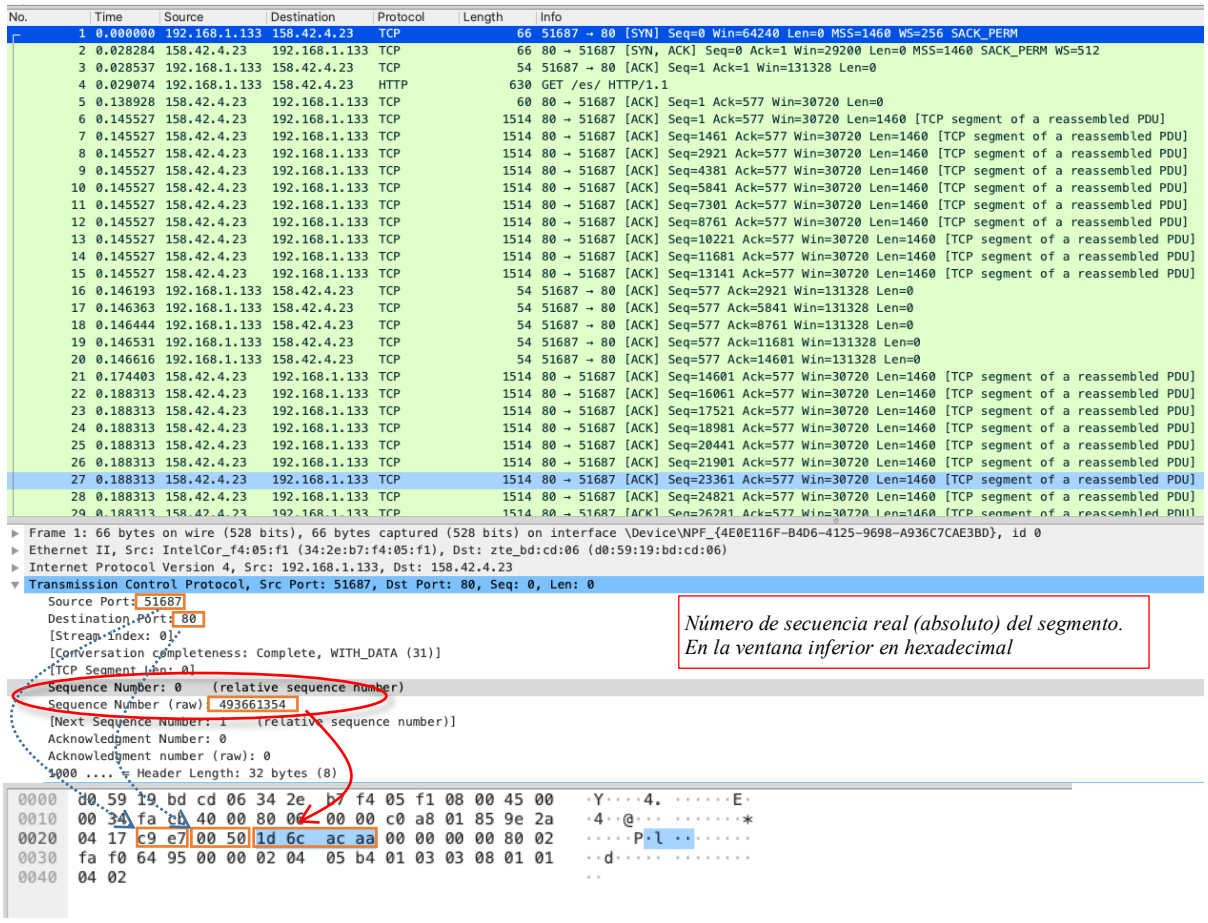

Figura 1. Números de secuencia absolutos y relativos.

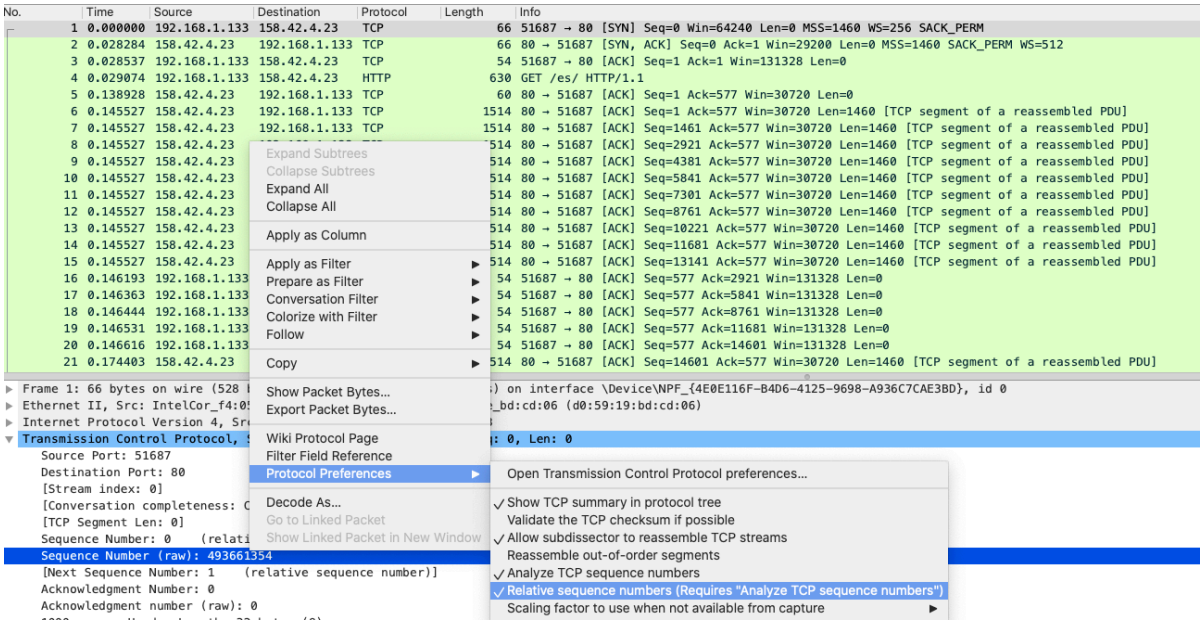

Figura 2. Modo de indicar cómo queremos ver los números de secuencia.

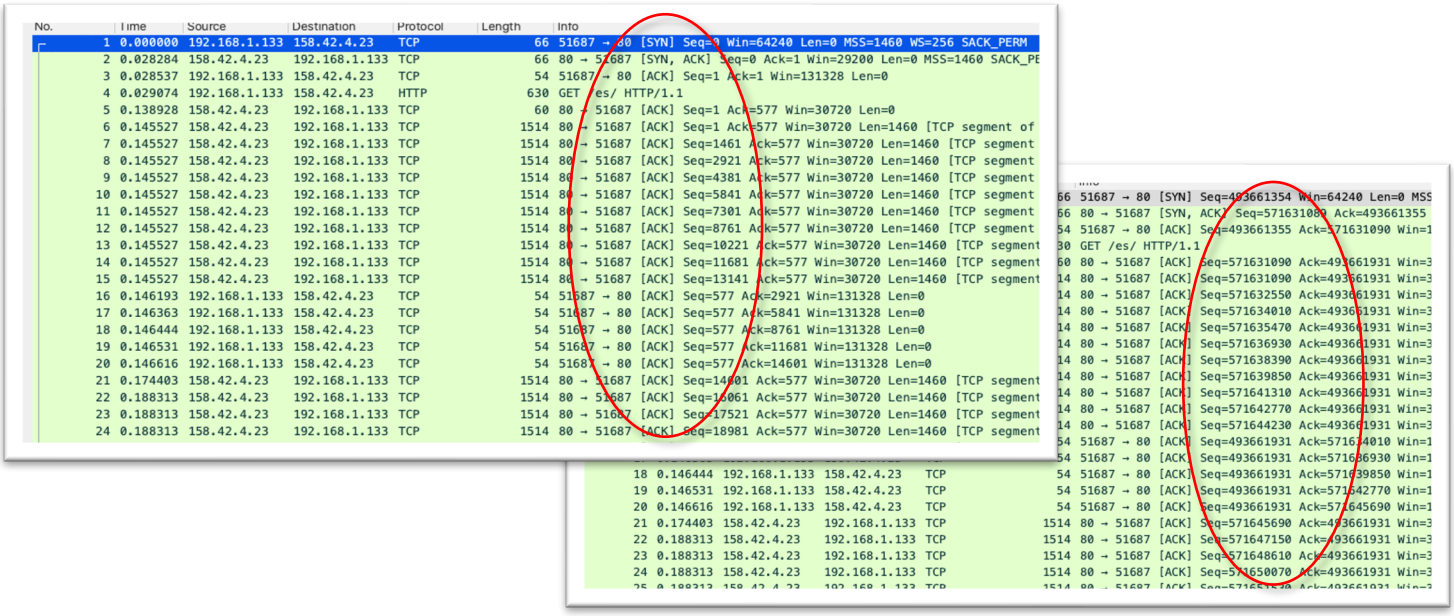

Figura 3. Números de secuencia relativos (izquierda) y absolutos (derecha).

Ejercicio 1 (continuación):

c) En este apartado vamos a analizar el cierre de la conexión. Para ello desplázate al final de flujo mostrado, que es donde verás el final de la conexión. ¿Quién toma la iniciativa? ¿En cuántas fases se realiza el cierre? ¿Cómo se modifican los números de secuencia y los de reconocimiento durante el cierre de la conexión? *Nota: Para ver claramente el cierre de la conexión no te fíes únicamente de dónde ves el bit de* `FIN` *en la ventana principal de Wireshark. Fíjate para ello en el segmento que en Wireshark ves como* `HTTP/1.1 200 OK` *y analiza su nivel de transporte en la ventana inferior buscando un* `FIN`.

d) Analicemos ahora el tamaño de la ventana de recepción. ¿Qué valor comunican cliente y servidor en el inicio de la conexión? ¿Se usa la opción de escalado de ventana? ¿Es la misma en los dos casos? Selecciona, por ejemplo, el segmento número 4, del `GET /es/` . Localiza en la ventana intermedia el valor `Window`. ¿Es igual a 513? ¿es ese el tamaño de la ventana del receptor? ¿Por qué Wireshark, debajo del anterior, indica `[Calculated Windows size: 131328]`?

Frecuencia de envío de reconocimientos y tamaño de la ventana de congestión

En esta sección vamos ver cómo se está realizando el envío de reconocimientos entre cliente y servidor y cómo evoluciona la ventana de congestión.

Ejercicio 2:

a) Usando el mismo flujo TCP del ejercicio anterior, ejecuta la opción del menú **Estadísticas → Gráfica de flujo**. A continuación, selecciona el flujo TCP de la ventana principal pinchando en la opción "**Limitar filtro de visualización**" que aparece en la nueva ventana (Figura 4). Selecciona también en la nueva ventana la opción "**Flow type: TCP flows**". Interpreta qué estás viendo.

b) A continuación, analiza la evolución de los reconocimientos (`ACK`) y responde a las preguntas siguientes. Para ello fíjate en los segmentos que envía el servidor web, su longitud y número de secuencia y observa cuándo el cliente envía los `ACK` y hasta qué byte están reconociendo. Esto lo puedes ver tanto en la ventana de **Estadísticas → Gráfica de flujo** que has abierto en el apartado previo como en la principal con el flujo seleccionado. Ten en cuenta que cuando marcas un segmento en la ventana de **Estadísticas → Gráfica de flujo**, se selecciona también en la ventana principal. (Nota: para saber cuál es el segmento del servidor que está reconociendo el cliente, analiza el número de secuencia del transmisor y súmale la longitud enviada; busca entonces el `ACK` que lleve ese número de reconocimiento cuando envía el cliente).

Céntrate en los primeros segmentos de la conexión y determina dos cosas: cuál es el tamaño inicial de la ventana de congestión del servidor de la UPV y si se emplean `ACK` retrasados. Para ello, analiza el número de segmentos de datos enviados inicialmente por el servidor. Teniendo en cuenta esto, ¿cuál es el tamaño inicial de ventana de congestión del servidor? Ahora fíjate en los reconocimientos que el servidor recibe del cliente ¿Por qué en la captura los `ACK` aparecen mucho después de los envíos que reconocen? A la vista de los resultados, ¿se están empleando reconocimientos retrasados? ¿cuántos segmentos enviados por el servidor son reconocidos por cada `ACK` enviado por el cliente?

Después de que el cliente envía 5 `ACK` (segmentos 16 a 20 de la captura), ¿qué información ha reconocido? ¿cuál debería ser el valor de la ventana de congestión del servidor después de recibir esos reconocimientos según lo visto en clase? Determinar ahora cuántos segmentos envía el servidor tras recibir esos cinco reconocimientos ¿Es congruente ese número con tu estimación del valor de la ventana de congestión del servidor? Reflexiona unos minutos y sigue leyendo…

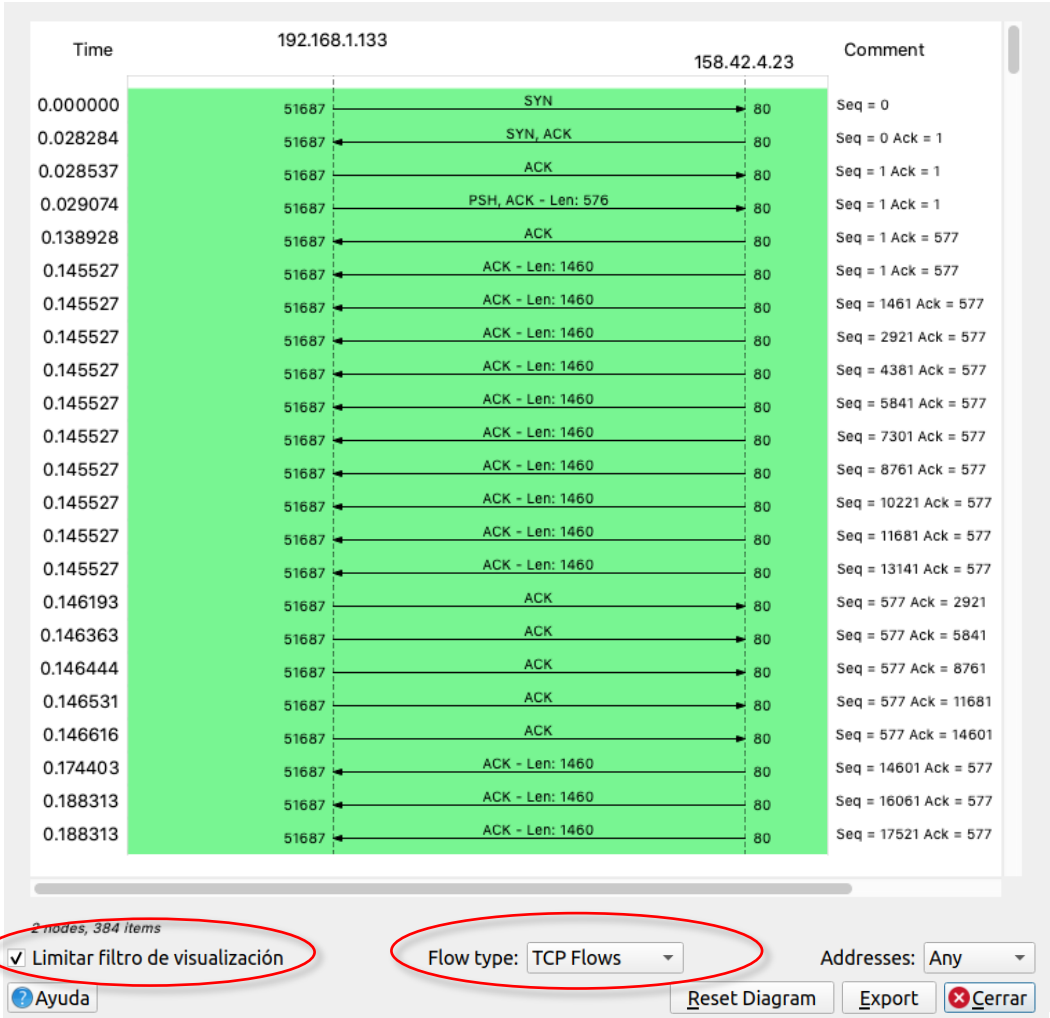

*Figura 4. Opción Wireshark **Estadísticas → Gráfica de flujo**.Flag RST*

*Hay implementaciones que incorporan la técnica denominada **TCP Congestion Control with Appropriate Byte Counting** (RFC 3465). En los sistemas que la implementan no se incrementa la ventana de congestión de forma constante con la recepción de cada* `ACK`*, sino que se incrementa tanto como la información reconocida por los segmentos de* `ACK`*. Así, si solo llega un* `ACK`*, pero reconoce 2, 3 ó más segmentos, la ventana se incrementa de forma acorde con la información reconocida.*

Por último, también es interesante saber que en las implementaciones actuales el número de segmentos que se envían al iniciarse una conexión no tiene que ser siempre igual a 2 (la norma RFC 5681 aporta algunas recomendaciones). Hay estudios que determinan, y así lo hacen la mayoría de sistemas actuales, que sería mejor incrementar el número de segmentos que se transmiten inicialmente. Muchos sistemas permiten enviar hasta 10 segmentos en el arranque de una conexión.

Reconocimientos selectivos (SACK)

En el establecimiento de la conexión del ejercicio 1 hemos visto que se va a permitir el uso reconocimientos selectivos. La opción `SACK` permite al receptor informar al emisor de que ha recibido un bloque de datos cuyo número de secuencia está dentro de la ventana de recepción, pero no coincide con el límite inferior de esa ventana. Es decir, ha recibido un segmento con un número de secuencia posterior al que debería haber llegado. Si la situación se repite y se pierden otra vez segmentos, el receptor puede acabar almacenando bloques de datos no contiguos en su buffer. Por cada bloque de datos recibido fuera de orden el receptor enviará un nuevo segmento con la opción `SACK`, que incluirá los bloques de datos posteriores al esperado recibidos más recientemente. De esta forma el emisor puede retransmitir los segmentos que se han perdido sin esperar a que venza su temporizador de retransmisión. Cuando lleguen los datos que rellenan el hueco, el receptor reconocerá los datos normalmente y avanzará el borde izquierdo de su ventana de recepción. El campo **Número de reconocimiento** de la cabecera TCP reconocerá todo el bloque de datos completo de forma global.

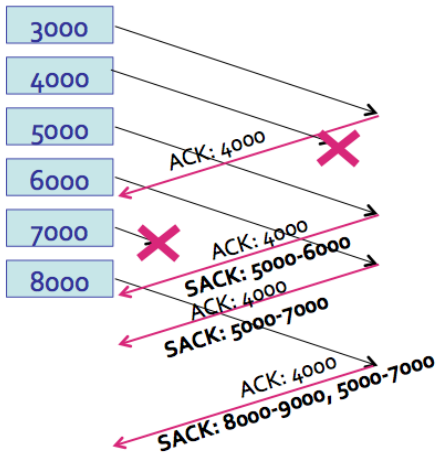

Figura 5. Ejemplo de uso de la opción SACK del protocolo TCP.

En la figura 5 puedes ver un ejemplo donde se pierden dos segmentos de datos, los que llevan número de secuencia 4.000 y 7.000. El receptor utilizará la opción `SACK` para avisar de que ha recibido otros bloques posteriores.

Es importante tener en cuenta que la opción `SACK` no modifica el significado del campo **Número de reconocimiento** de la cabecera TCP, cuyo valor aún especificará el borde inferior de la ventana de recepción.

Ejercicio 3:

Descarga el fichero **`UPVSacks.pcapng`** que está disponible en Poliformat y cárgalo en Wireshark. Es una captura de una de las peticiones a unos de los objetos referenciados en la página principal de la UPV que has analizado en la primera parte de la práctica. Verás que los primeros segmentos son muy similares a los que ya has analizado, pero, ahora, han ocurrido algunos errores en la comunicación y podemos analizar algunos detalles más del funcionamiento de TCP.

a) En este apartado vamos a ver en funcionamiento la opción `SACK`. En el tráfico capturado parece que todo funciona sin problemas hasta el segmento número 83. Apúntate su número de secuencia. Analizar ahora el siguiente, el número 84. ¿qué número de secuencia tiene? ¿cómo puede haber ocurrido eso? ¿qué está recibiendo el cliente en los segmentos 85 a 91?

b) Pasemos a los segmentos 92 a 97. ¿Qué información están reconociendo? Fíjate ahora en el 98 ¿qué se está reconociendo, teniendo en cuenta que ya aparecen los campos relativos a los reconocimientos selectivos?

c) Analiza los siguientes segmentos, del 99 al 104, ¿qué información reconocen? ¿Por qué siguen llevando los campos de los reconocimientos selectivos? ¿Por qué en el segmento 105 ya no están esos campos?

d) ¿Puedes identificar, por último, cada una de las fases del cierre de la conexión? ¿Quién la inicia, el cliente o el servidor?

Flag RST

En esta sección, analizaremos el uso del flag `RST`.

El efecto del flag `RST` es cerrar inmediatamente la conexión. Esto es ligeramente diferente de un `FIN`, que solo dice que el extremo que lo envía ya no va a transmitir ningún nuevo dato, aunque aún puede recibir algunos. Hay dos tipos de eventos que hacen que se transmita un `RST`:

a) Que la conexión se cancele explícitamente de forma abrupta. Por ejemplo, se mata el proceso que tiene abierto el socket.
b) Que TCP reciba ciertos tipos de segmentos no válidos. Por ejemplo, un segmento `SYN` destinado a un puerto que no está abierto en el sistema o un segmento destinado a una conexión que no existe.

Hay que tener en cuenta que en ocasiones se utilizan cortafuegos (*firewall*) que impiden el paso de los segmentos `SYN` destinados a puertos cerrados en el sistema destino. Estos segmentos se descartarán tanto sin avisar a TCP, lo que dificulta el trabajo de los atacantes al proporcionar menos información que una respuesta con un segmento `RST`, como generando mensajes ICMP indicando que la comunicación está prohibida en ese destino. El protocolo ICMP lo estudiaremos en el segundo cuatrimestre, pero podemos avanzar que permite que los hosts y routers envíen mensajes de control para la detección de fallos y problemas.

Ejercicio 4:

a) Realiza una captura mediante el analizador Wireshark de una conexión al puerto 81 del servidor web de la Universidad de Valencia (ojo, no de la UPV). Para ello en el navegador usa la URL http://www.uv.es:81. **Ten cuidado si tienes algún filtro activo y establécelo al puerto 81**. Analiza la captura realizada. ¿Cómo se indica que no hay un servicio en ese puerto? ¿Cómo se cierra la conexión? ¿Qué le contesta tu navegador? ¿Vuelve a intentar tu navegador la conexión? En caso afirmativo ¿cuántas veces?

b) Repite lo mismo pero esta vez intentando hacer la conexión con el servidor de nuestra universidad: http://www.upv.es:81. ¿Pasa lo mismo que en el caso anterior? ¿Qué motivos puedes encontrar para esos comportamientos?

c) Por último, intenta hacer la conexión con el puerto 81 del servidor del periódico deportivo marca: http://www.marca.es:81 ¿Pasa lo mismo que en los casos anteriores? ¿Qué motivos puedes encontrar para esos comportamientos?

Como se puede comprobar, los servidores anteriores se comportan de manera diferente. En el primer caso, como el puerto está cerrado (no hay un servidor escuchando en el puerto 81), TCP identifica la solicitud de conexión del cliente como un segmento no válido y, por lo tanto, responde con un segmento `RST`. En los otros dos casos está actuando un cortafuegos que impide el acceso a los puertos del sistema que no están abiertos. En el caso de la UPV, el cortafuegos genera un mensaje ICMP de tipo 3 ("*Destino inalcanzable*") y código 10 ("*Comunicación con host destino administrativamente prohibida*"), mientras en el caso del periódico Marca, el cortafuegos simplemente descarta el paquete y no envía ningún aviso.

Ejercicio opcional

Realiza una captura, mediante el analizador Wireshark, de una conexión a la página web de la Universidad (`http://www.upv.es/es/`) desde Firefox en **Linux**. Es conveniente que **limpies** el historial del navegador antes de realizar la captura. Además, para facilitar la interpretación de los datos, puede ser útil establecer un filtro de captura ("`port 80 and host www.upv.es`") desde el menú **Captura→Opciones** o uno de visualización ("`tcp.port == 80 and ip.addr == 158.42.4.23`") desde el filtro de la ventana principal.

Como en el ejercicio 1, localiza el primer flujo, el del `GET /es/ HTTP/1.1,` y analiza el MSS. Compara el MSS indicado al inicio de la conexión con el tamaño máximo del campo de datos utilizado durante la misma. Realiza esta comparación para diversos segmentos. La longitud de los datos contenidos en los segmentos recibidos ¿coincide con el MSS? ¿Es mayor o menor que el MSS? ¿Puede ser mayor?

Como habrás visto, para numerosos segmentos mostrados por Wireshark en la captura que has realizado, el tamaño de los datos contenidos en el segmento es claramente mayor que el MSS. Esto contradice, a priori, el significado de la opción MSS. ¿Qué está ocurriendo? *En redes de alta velocidad, muchas veces las tarjetas de red tienen capacidad para procesar por ellas mismas los segmentos TCP, descargando de esta manera a la CPU del ordenador. Con esto se consigue una importante mejora de prestaciones. Entre otras acciones, estas tarjetas de red son capaces de agregar varios segmentos consecutivos de forma que entregan al sistema operativo un segmento de mayor tamaño. Es decir, aunque por la red viajan segmentos con un campo de datos menor o igual que el MSS, estas tarjetas de red concatenan los datos de varios de estos segmentos para entregarlos juntos al sistema operativo como si fuera un único segmento. Para las operaciones de transmisión también existe el mismo funcionamiento concatenado, de forma que se le puede pasar a la tarjeta de red un volumen de datos mayor al MSS establecido y la tarjeta de red se encargará de dividirlo en segmentos del tamaño establecido. Por este motivo, Wireshark muestra segmentos de tamaño mayor que el MSS establecido al inicio de la conexión. Como la tarjeta de red, cuando se reciben nuevos segmentos, genera una interrupción al procesador cada vez que debe pasarle un segmento, uno de los beneficios de esta concatenación de segmentos es que reduce el número de interrupciones necesarias. Este menor número de interrupciones mejora las prestaciones globales del sistema dado que el procesador es interrumpido menos veces. Estas técnicas se conocen con el nombre de* TCP Segmentation Offload (TSO), Large Receive Offload (LRO), *y en entornos Windows TCP* Chimmney Offload *y* Receive Segment Coalescing (RSC).

A continuación analiza, como antes, la evolución de los reconocimientos (`ACK`) y responde a las preguntas del siguientes. Para ello fíjate en los segmentos que envía el servidor web, su longitud y número de secuencia y observa cuándo el cliente (tu ordenador) envía los `ACK` y hasta qué byte están reconociendo. Esto lo puedes ver tanto en la ventana de **Estadísticas → Gráfica de flujo** que has abierto en el apartado previo como en la principal con el flujo seleccionado.

¿Hay un `ACK` cada dos segmentos TCP? ¿Por qué? Intenta encontrar una explicación (Razona sobre las posibles ventajas de evitar el uso de `ACK` retardados durante los primeros RTT, cuando está en

ejecución el algoritmo de arranque lento y, por otra parte, los aspectos de la concatenación de segmentos antes mencionada).

Durante el inicio de una conexión, la mayoría de sistemas actuales tienen un modo denominado ***QuickACK****. En este caso no se entra en el envío diferido o retardado de* `ACK` *hasta instantes de tiempo posteriores. ¿Cuánto tiempo? Pues depende de las diversas implementaciones existentes: unas permanecen en este modo hasta que se transmite un determinado número de segmentos, otras tienen un temporizador que determina la transición al modo retardado y otras, además, analizan el tipo de tráfico para determinar si es interesante o no este modo (porque está generando más tráfico). ¿Qué se consigue con esto? El objetivo es enviar un* `ACK` *por cada segmento recibido y conseguir que la ventana de congestión crezca más rápido que si el receptor envía un* `ACK` *cada dos segmentos. Y a esto tenemos que unirle lo que hemos explicado antes de TCP* ***Congestion Control with Appropriate Byte Counting*** *(RFC 3465) Estas dos técnicas no son excluyentes y podemos encontrar implementaciones que utilicen las dos (por ejemplo, piensa que un sistema puede usar QuickACK para que el otro extremo –que no sabe si usa Appropriate Byte Counting o no- incremente en cualquier caso la ventana de congestión más rápido).*

Como has visto, a partir del comportamiento habitual descrito en el aula y analizado en la primera parte de la práctica, las optimizaciones de TCP que se han ido realizando (y seguirán proponiéndose nuevas) hacen que los comportamientos observados puedan diferir en algunas situaciones. En esta parte opcional de la práctica hemos destacado algunas de ellas. Hay muchas más. Cuando te encuentres con alguna, intenta encontrar una explicación y buscar información sobre esa optimización, porque ya tienes la base para entenderla.