Redes Multimedia – Prácticas 2019

Práctica 2: Medidas de rendimiento

Turno y pareja: 4312\_02

Integrantes:

Javier Martín González

Jorge Gutiérrez Díaz

Fecha de entrega: 18-03-2019

Contenido

[1Introducción 2](#__RefHeading___Toc285_732697610)

[2Realización de la práctica 2](#__RefHeading___Toc287_732697610)

[3Conclusiones 5](#__RefHeading___Toc289_732697610)

# Introducción

En esta práctica se va a realizar el rendimiento de una red según diferentes parámetros cuando se envía un tren de paquetes. Dentro de las características a configurar se encuentra por ejemplo la cantidad de paquetes y el tamaño de los mismos. Las pruebas se van a realizar en unos casos sobre un mismo equipo y en otros con dos diferentes.

Por un lado, tenemos un servidorTren en el que se implementarán los correspondientes cálculos para obtener las mediciones de ancho de banda y tiempo para además imprimirlo por pantalla. De igual forma se harán análisis del tráfico a través de Wireshark.

Por otro lado, hay un clienteTren que se nos proporciona, pero tenemos que hacer una versión alternativa (clienteTren2) que permita introducir una tasa binaria por línea de comandos y así́ limitar la tasa de transferencia a una menor o igual a la máxima para obtener unas medidas reales del retardo. Para hacer las pruebas, se utiliza un emulador que también se nos facilita junto a la práctica con el que en función del número de DNI introducido (en nuestro caso lo probamos con el de los dos integrantes de la pareja), se simula una red con unas determinadas pérdidas de paquetes, anchos de banda y retardos.

Para terminar, y una vez que tenemos los datos del análisis, plantearemos un servicio VoIP que funcione bien bajo las características de nuestro tipo de red en concreto.

# Realización de la práctica

## 

1. Se entrega un programa que envía trenes de paquetes en los que se puede configurar la longitud del tren y la longitud del campo de datos de los paquetes por encima del nivel de aplicación. Los paquetes poseen un campo de número de secuencia y otro de marca de tiempos. Para ello se utiliza la cabecera típica de RTP, según se proponía en la práctica 0. Los parámetros se entrarán por línea de comandos siguiendo la sintaxis:

*clienteTren.py ip\_destino puerto\_destino longitud\_tren longitud\_datos*

Estudie en detalle el funcionamiento del programa, pues deberá modificarlo en un apartado posterior. ¿Qué valor de marca de tiempo se envía? ¿Qué relación tiene con el tiempo *epoch* (segundos desde 1970)?

|  |
| --- |
| En primer lugar se realiza una comprobación de la cantidad de parámetros introducidos. En el caso de que se haya introducido un número incorrecto de parámetros, el programa devuelve un mensaje mostrando como iniciar el cliente de forma correcta.  A continuación, el programa recoge los parámetros y comprueba que el tamaño de datos sea lo suficientemente grande como para que pueda ser enviado y lo suficientemente pequeño como para que no se produzca fragmentación.  Por último, añadimos a la cabecera RTP de cada paquete un número de secuencia que empieza desde 0 y aumenta con cada paquete, un identificador común para cada paquete que sea la longitud del tren a enviar, y el tiempo en que se envió cada paquete (segundos y centésimas de segundo) dentro del timestamp. |
| Dentro de la cabecera RTP, el valor del timestamp se puede modificar tanto en segundos como en centésimas de segundos truncados a 32 bits. En nuestro caso, obtenemos el tiempo del envio del paquete en segundos y lo multiplicamos por la constante DECENASMICROSECS para pasarlo a decenas de microsegundos. Por último, el resultado lo pasamos a través de una máscara para truncarlo a 32 bits. |
| El tiempo *epoch* es el tiempo transcurrido desde el 1 de enero de 1970 y que dura hasta nuestros días. El tráfico en la red utiliza este formato de tiempo para reflejar las distintas marcas de tiempo, por lo tanto, utilizamos este formato para imitar el tráfico real de la red. La función de time.time() devuelve el tiempo en segundos desde el tiempo *epoch* como un numero de punto flotante. |

1. Se debe completar un programa que recibe los trenes de paquetes, de forma que mida y visualice por pantalla anchos de banda (instantáneos, máximo, medio y mínimo), retardos en un sentido (instantáneos, máximo, medio y mínimo), variación del retardo, y pérdida de paquetes (%).Tenga en cuenta que los paquetes que se envíen a la red contendrán igualmente las cabeceras de las capas inferiores (RTP, UDP, IP, Ethernet) según se mostraba en la Error: no se encontró el origen de la referencia, por lo que dicha longitud también debe ser tenida en cuenta a la hora de medir el ancho de banda. En el caso de utilizar la interfaz local no habrá cabecera Ethernet. El servidor deberá ejecutarse según la siguiente sintaxis:

*servidorTren.py ip\_escucha puerto\_escucha*

|  |
| --- |
| Se han creado tres listas para guardar el ancho de banda, el retardo de los paquetes y el tiempo de llegada de estos. Como las mediciones pueden realizarse en localhost o entre dos equipos, se ha utilizado un *if* que sume al tamaño de cabeceras totales la cabecera de *ethernet* en el caso que se haga las mediciones entre dos equipos.  A la llegada de un paquete, se guarda el retardo y el tiempo de llegada del paquete en sus respectivas listas. A partir del segundo paquete recibido, se calcula el ancho de banda instantáneo sumando el tamaño de datos del paquete con el tamaño total de las cabeceras y se multiplica por 8 para pasarlo a bits. El resultado se divide entre la diferencia del tiempo de llegada del último paquete con el anterior, y el ancho de banda obtenido se guarda en la lista de anchos de bandas. El ancho de banda que se muestra por pantalla es el calculado pero dividido entre 1000000 para pasarlo a Mb/s.  Para el ancho de banda y retardo medio, máximo y mínimo se ha aplicado las funciones *statistics,mean, max* y *min* de listas para obtener los resultados respectivamente. Además, se ha calculado el factor de dispersión en el servidor como el tiempo transcurrido entre el último bit del primer paquete y el último bit del último paquete del tren.  Por último, se calcula el porcentaje de paquetes perdidos dividiendo la diferencia de la longitud del tren menos la cantidad de paquetes recibidos, entre la longitud del tren. Todo se ha multiplicado por 100 para obtener el porcentaje. El *jiiter* se obtiene con la varianza de la lista de retardos mediante la función de *statistics.variance.* |

1. Pruebe ambos programas en la interfaz local y entre dos equipos conectados en la red de área local. Realice varias medidas variando la longitud del tren y la longitud de los paquetes. Responda a las siguientes preguntas. Utilice el Wireshark en el receptor para ver los paquetes del tren y contrastar las respuestas.

|  |
| --- |
| Se utilizó el comando *ifconfig* en el equipo donde se ejecuta el servidor para obtener la *ip* de este*.* Se ejecuta el cliente desde un ordenador diferente y envia los paquetes a la ip donde se ejecuta el servidor. Wireshark corre en el mismo equipo que el cliente y filtramos los paquetes RTP con *ip* destino al equipo del servidor. |

* 1. ¿Qué valores ha empleado para contabilizar las distintas cabeceras?

|  |
| --- |
| IP = 20, ETH = 26, UDP = 8, RTP = 12 |

* 1. ¿Cuál es la longitud mínima de una de las tramas que se envían? ¿Por qué?

|  |
| --- |
| 6 bytes. El valor mínimo de datos que puede enviar *Ethernet* es de 46 bytes, y la suma de todas las cabeceras que se envían es de 40 bytes. Si se resta el valor mínimo de la cabecera Ethernet con la suma de las cabeceras, se obtiene que el valor mínimo de las tramas debe ser de 6 bytes. |

* 1. ¿Cuál es la longitud máxima de datos que tiene sentido utilizar? ¿Por qué?

|  |
| --- |
| Si se aplica la misma técnica, pero con el cambio de que el valor máximo de la cabecera Ethernet es de 1500 bytes, se obtiene que el valor máximo de las tramas es de 1460 bytes. |

* 1. ¿Con qué longitudes de tren y datos se consiguen mejores resultados? ¿Por qué?

|  |
| --- |
| Se obtienen mejores resultados con longitudes de trenes grandes y tamaños de paquetes grandes porque así evitamos que el tráfico interferente de la red interfiera en los resultados. En nuestro caso, con trenes de 1000 paquetes y de 1000 bytes de datos se obtenían buenos resultados. |

1. Para poder medir adecuadamente retardos y *jitter* es necesario que el cliente envíe a una tasa inferior a la de la red, de forma que se elimine el efecto del cuello de botella sobre el tren. Modifique el programa proporcionado en el apartado para que permita configurar también la tasa de envío de los paquetes. Su sintaxis será:

*clienteTren2.py ip\_destino puerto\_destino longitud\_tren longitud\_datos [tasa\_binaria]*

Si no se indica la tasa binaria, se transmitirá a la tasa máxima posible, lo que permite hacer una estimación del ancho de banda, que se puede utilizar posteriormente para hacer adecuadamente las medidas de retardo y *jitter*.

|  |
| --- |
| En el caso de que se ejecute en programa sin indicar una tasa binaria, implica transferir a tasa máxima, que es lo equivalente a los resultados obtenidos en los apartados anteriores y no se apreciará ninguna diferencia.  Si por el contrario se ejecuta el programa indicando un valor de tasa, para calcular el tiempo de retardo en el sleep, se divide el tamaño del paquete entre dicho valor. En función de si se utiliza o no la dirección localhost (127.0.0.1) el tamaño del paquete puede variar. Si es entre un mismo ordenador no se añade la cabecera ethernet, en cambio entre dos ordenadores con diferentes direcciones IP, si se haría. |

1. Realice medidas con clienteTren2.py y servidorTren.py, utilizando el emulador compilado que se puede descargar de Moodle. El emulador es un programa que simula retardos variables, pérdidas (similar al realizado en la práctica 1) pero también anchos de banda. Dicho ejecutable tiene la siguiente sintaxis:

*emulador ip\_escucha puerto\_escucha ip\_destino puerto\_destino DNI*

donde ip\_esucha y puerto\_escucha son la dirección IP y puerto donde escucha el emulador, ip\_destino y puerto\_destino son la dirección IP y puerto a donde el emulador reenvía lo que recibe, y DNI es un número de DNI (sin letra). El programa, siempre en base al número del DNI que se proporcione, impone una combinación de ancho de banda, retardo, variación del retardo y porcentaje de pérdida de paquetes único.

Deduzca a partir de las medidas realizadas qué valores de ancho de banda, retardo, variación del retardo y pérdidas se están aplicando en el emulador para el DNI de ambos miembros de la pareja. **Tenga en cuenta que para medir los retardos de forma correcta es necesario limitar la tasa de transferencia a una inferior o igual a la medida.**

|  |
| --- |
| La tasa que hemos introducido ha sido el ancho de banda medio de cada DNI de los integrantes de la pareja (uno por cada ejecución). Hemos utilizado el emulador proporcionado, y entre dos ordenadores y con los puertos de escucha 5004 y 5006. Con nuestros números del DNI lo que hemos logrado ha sido simular una red con unos valores de pérdidas y retrasos acordes al valor introducido, generando así dos situaciones diferentes y que no serán igual al del resto de parejas.  Asimismo, hemos ejecutado por un lado el servidorTren escuchando en el puerto 5006 para obtener los resultados de la red y por el otro el clienteTren2 con una tasa binaria de 10 hacia la dirección de destino del servidor.  Los resultados se pueden ver a continuación. |
| DNI 1: 53747580  Ancho de banda estimado: 4,56 kb/s  Retardo estimado: 1,089 s  Desviación estándar del retardo: 0,00022 s  Porcentaje de pérdidas: 2,6% |
| DNI 2: 50638681  Ancho de banda estimado: 4,53 kb/s  Retardo estimado: 1,086 s  Desviación estándar del retardo: 0,00034 s  Porcentaje de pérdidas: 3,6% |

1. Capture el tráfico de las medidas realizadas con el emulador y analice con Wireshark el tráfico recibido y a partir de los tiempos de llegada, marcas de tiempo y longitudes, compare estos datos con los resultados obtenidos con su programa.

|  |
| --- |
| La forma de arrancar los ejecutables de cara a las mediciones, en este caso es igual que en el anterior, con la diferencia de que hemos activado Wireshark para que escuche la interfaz eth0 antes de arrancar el servidor y el cliente, y así obtener todos los paquetes que se han ido enviando. A continuación, se han calculado las medidas en base a la captura de paquetes realizada.  Para ello, hemos puesto un filtro para el puerto 5006 de UDP (udp.port==5006), y hemos decodificado los paquetes como RTP. Hemos cambiado otros ajustes como la forma de visualización de los tiempos y para el análisis hemos tenido en cuenta los números de secuencia y los timestamps.  Para el ancho de banda y obtener el porcentaje de pérdidas, hemos navegado a la pestaña Telephony < RTP < Stream Analysis.  Los datos de desviación y retardo los hemos calculado en un Excel en base a los datos exportados. |
| DNI 1: 53747580  Ancho de banda estimado: 6,39 kb/s  Retardo estimado: 1,02 s  Desviación estándar del retardo: 0,141 s  Porcentaje de pérdidas: 1,80% |
| DNI 2: 50638681  Ancho de banda estimado: 6,25 kb/S  Retardo estimado: 1,04 s  Desviación estándar del retardo: 0,22 s  Porcentaje de pérdidas: 4,10% |

1. Se desea establecer un servicio de VoIP sobre una red cuyos parámetros de calidad son los del emulador. Explique razonadamente qué códec y tiempos de paquetización deberá utilizar en ambos casos para adaptarse de la mejor manera posible al canal, y cuantas llamadas simultáneas se podrían soportar en ese caso (se supone una red full-duplex). Para valorar dicho códec y tiempo de paquetización puede utilizar [esta página](http://www.cisco.com/en/US/tech/tk652/tk698/technologies_tech_note09186a0080094ae2.shtml). Igualmente, indique el tamaño del buffer a configurar en el receptor de la llamada para amortiguar el efecto del *jitter*.

|  |
| --- |
| No se puede realizar un servicio VoIP ya que al utilizar la fórmula para calcular el códec, al ser nuestro retardo medio de 1089 ms sobrepasa el retardo máximo que puede tener cualquiera de los códec. |
| No se ha podido instalar el códec y por lo tanto no se ha podido calcular. |
| No se ha podido instalar el códec y por lo tanto no se ha podido calcular. |
| No se puede realizar un servicio VoIP ya que al utilizar la fórmula para calcular el códec, al ser nuestro retardo medio de 1086 ms sobrepasa el retardo máximo que puede tener cualquiera de los códec. |
| No se ha podido instalar el códec y por lo tanto no se ha podido calcular. |
| No se ha podido instalar el códec y por lo tanto no se ha podido calcular. |

1. Simule el servicio planteado en el apartado 7 con las herramientas codificadas previamente y evalúe si el resultado responde a la predicción realizada. Indique razonadamente los parámetros utilizados para generar el tren de paquetes.

|  |
| --- |
| Al no haber implementado el servicio, no se pueden obtener los parámetros. |
| Al no haber implementado el servicio, no se puede evaluar las medidas. |

# Conclusiones

Esta práctica nos ha permitido experimentar como poder implementar un servicio de VoIP y la importancia que tienen las medidas de rendimiento de la red en él. Hay que tener en cuenta estas medidas para implementar el códec que mejor se adapte a nuestras circunstancias.

Además, hemos podido comprobar la diferencia que supone transmitir información desde *localhost* y obtener unos resultados excelentes, a transmitir entre dos equipos distintos donde comienza a observarse resultados con peores medidas. Al transmitir con el emulador entre medias de los dos equipos, hemos podido experimentar una calidad más cercana a lo que se transmite por la red hoy en día.

Por último, hemos comprobado que las situaciones donde se mide mejor el rendimiento mediante trenes de paquetes son aquellas donde se envían trenes con longitudes grandes y con paquetes de gran tamaño. Aun así, hay que tener cuidado con no enviar un tren de longitud excesivo para no sobrecargar la red, ni enviar paquetes con suficientes datos para que se produzca fragmentación, ya que se harían peores medidas.