-Redes Multimedia – Prácticas 2019
Práctica 2: Medidas de rendimiento

Turno y pareja: 2461_06

Integrantes:

Pablo Díez del Pozo

Alejandro Alcalá Álvarez

Fecha de entrega: 18/03/2020

Contenido

1 Introducción

En esta práctica nos hemos introducido en las medidas de calidad que tienen las redes multimedia, para ello hemos tenido que conocer como funcionan los trenes de paquetes y de esos trenes de paquetes hemos valorado las perdidas, los retardos y el ancho de banda que hay en ellos. Los resultados obtenidos los hemos aplicado al despliegue de un servicio de VoIP.

2 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)?

El valor de la marca tiempo que se envía en la cabecera del paquete es el tiempo de envio al servidor que esta truncado para que se pueda introducir en el campo timestamp de RTP.

La marca de tiempo se construye de la siguiente manera:

```
    for i in range(0,trainLength):
        #usamos la longitud del tren como identificador de fuente. De esta manera en destino podemos saber la
        #longitud original del tren. En el campo timestamp (32bits) sólo podemos enviar segundos y
        #centésimas de milisegundos (o decenas de microsegundos, segun se quiera ver) truncados a 32bits
        message=struct.pack('!HHII',0x8014,seq_number, int(time.time()*DECENASMICROSECS)&B_MASK,trainLength)+data
        sock_send.sendto(message,addr)
        seq_number+=1
```

Pero este tiempo lo tenemos que truncar porque el resultado que nos devuelve el método sobrepasa los 32 bits y por eso lo tenemos que truncar a decenas de microsegundos para que se pueda introducir en el campo timestamp de RTP.

La relación que hay es que el método time de la librería time devuelve los segundos transcurridos de enero de 1970 hasta que se haya enviado el paquete al servidor.

2. 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 encuentra 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

Después de recibir todos los paquetes que nos ha enviado el cliente ya podemos calcular las siguientes medidas de ancho de banda, retardos y porcentaje de perdidas. Primero, al recibir los paquetes creamos una lista con dos elementos:

- El primer elemento es el paquete que hemos recibido.
- El segundo elemento es la marca de tiempo en la que ha llegado el servidor.

Después hemos recorrido recorrido la lista resultante que nos ha salido al recibir los paquetes. Primero, hemos calculado los retardos obtenidos con el campo timestamp de la cabecera RTP, donde debemos quitar el truncado que hemos hecho para poder obtener los segundos de retardo. Todos estos retardos los hemos ido almacenando para luego poder calcular la media de los retardos, el retardo mínimo, máximo y el jitter de los paquetes recibidos.

```
print ('Retardo instantaneo en un sentido (s): ',(reception_time_trunc-send_time_trunc)/DECENASMICROSECS)
retardos.append((reception_time_trunc-send_time_trunc)/DECENASMICROSECS)
if npackets > 1:
```

A continuación, mostraremos los cálculos pertinentes para calcular las medidas que hemos comentado con anterioridad:

```
jitter=0
sumaRetardo = 0
for ret in retardos:
    sumaRetardo += ret
retardo_medio = sumaRetardo / len(retardos)
print ('Retardo Máximo en (s): ', max(retardos))
print ('Retardo Mínimo en (s): ', min(retardos))
print ('Retardo Medio en (s): ', retardo_medio)
sumatorio =  0
res = 0
for retardo in retardos:
    res = (retardo - retardo_medio)**2
    sumatorio += res
jitter = math.sqrt(sumatorio/len(retardos))
print ('Variación del retardo: ',jitter)
```

En segundo lugar, hemos obtenido los anchos de bandas instantáneos con el valor del tiempo que hemos guardado en la lista que hemos creado al recibir los paquetes. Todos estos anchos de bandas los hemos guardado para poder obtener el máximo y el mínimo de todos ellos. Después, hemos calculado el ancho de banda medio donde lo hemos calculado con la siguiente fórmula:

$$\frac{\left((nºpaquetes - 1) * longitud_{paquete}\right)}{(tiempoRec_N - tiempoRec_1)}$$

A continuación, vamos a mostrar como hemos calculado el ancho de banda instantáneo:

```
if npackets > 1:
    reception_time_i1 = packet_list[npackets-2][1]
    if (reception_time_i1 - reception_time) == 0:
        anchoBandaInstantaneo = tamanoPackets
    else:
        anchoBandaInstantaneo = tamanoPackets / (reception_time - reception_time_i1)
    anchosBandas.append(anchoBandaInstantaneo)
    print ('Ancho de Banda Instantáneo (b/s): ', anchoBandaInstantaneo)
```

No podemos calcular el ancho de banda del primer paquete recibido debido a que no tiene ancho de banda medible, por lo tanto, empezamos a calcular el ancho de banda desde el segundo paquete debido a que podemos obtener el tiempo del paquete que recibimos y también tenemos el tiempo de recepción del paquete anterior.

Ahora mostraremos como hemos calculado el ancho de banda medio, el máximo y el mínimo:

```
reception_time_1 = packet_list[0][1]
reception_time_N = packet_list[-1][1]
anchoBandaMedio = ((npackets - 1) * tamanoPackets)/ (reception_time_N - reception_time_1)
print ('Ancho de Bando Medio (b/s): ', anchoBandaMedio)
print ('Ancho de Banda Medio (Kb/s): ', (anchoBandaMedio/1000))
print ('Ancho de Banda Medio (Mb/s): ', (anchoBandaMedio/1000000))
print ('Ancho de Banda Medio (Gb/s): ', (anchoBandaMedio/1000000000))
print ('Ancho de Banda Instantáneo Máximo (b/s): ', max(anchosBandas))
print ('Ancho de Banda Instantáneo Mínimo (b/s): ', min(anchosBandas))
```

Por pantalla vamos a imprimir el ancho de banda en bps, Kbps, Mbps y Gbps. El máximo y el mínimo lo sacamos por pantalla como bps.

Por último, hemos sacado el porcentaje de perdidas que hemos obtenido de la siguiente manera:

```
packetLoss= trainLength - npackets
print ('Número de paquetes recibidos: ', npackets)
print ('Perdida de paquetes: ',packetLoss)
print ('Porcentaje de Perdida de paquetes: ', (packetLoss*100)/npackets)
```

Como en la estructura tenemos el número de paquetes que enviamos, solo debemos tener un contador en el bucle para poder contar los paquetes que hemos recibido en el servidor. Como mostramos con anterioridad el cálculo es trivial para obtener esta medida de calidad.

3. 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.

Primero, hemos obtenido a lo que puede llegar a transmitir la tarjeta de red con el comando ethtool <interfaz-red>. A continuación, mostraremos la salida de este comando en la terminal:

```
e336863@6A-26-6-26:~/RMult/P2/Code/codigoTrenPaquetes$ ethtool enp0s25
Settings for enp0s25:
        Supported ports: [ TP ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Supported pause frame use: No
        Supports auto-negotiation: Yes
        Supported FEC modes: Not reported
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Full
        Advertised pause frame use: No
        Advertised auto-negotiation: Yes
        Advertised FEC modes: Not reported
        Speed: 1000Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 1
        Transceiver: internal
        Auto-negotiation: on
        MDI-X: on (auto)
Cannot get wake-on-lan settings: Operation not permitted
        Current message level: 0x00000007 (7)
                               drv probe link
        Link detected: yes
```

Aquí podemos observar que la velocidad que transmite la tarjeta de red es de 1000 Mbps.

La configuración en localhost es la siguiente:

- Debemos abrir dos terminales.
- En una de ellas debemos escribir el siguiente comando: *python3 clienteTren.py 127.0.0.1 5004 100 1200*, donde el primer parámetro es la IP a la que queremos enviar, el segundo es

el puerto destino, el tercero es el número de paquetes que queremos enviar y el último es el tamaño de cada paquete.
- En la otra terminal debemos escribir el siguiente comando: *python3 servidorTren.py 127.0.0.1 5004*, donde el primer argumento es la IP del servidor y el segundo argumento es el puerto donde escucha.
Para ejecutar debemos primero que dar al *Enter* en la terminal que esta el segundo comando e inmediatamente debemos dar al *Enter* en la otro terminal.
La configuración en LAN es la siguiente:
- Debemos que tener dos ordenadores encendidos que estén conectados a la misma subred. Un ordenador hará de cliente y otro hará de servidor.
- En el ordenador cliente debemos abrir una terminal y escribir el siguiente comando: *python3 clienteTren.py 127.0.0.1 5004 100 1200*, donde el primer parámetro es la IP a la que queremos enviar, el segundo es el puerto destino, el tercero es el número de paquetes que queremos enviar y el último es el tamaño de cada paquete.
- En el ordenador servidor debemos abrir una terminal y escribir el siguiente comando: *python3 servidorTren.py 127.0.0.1 5004*, donde el primer argumento es la IP del servidor y el segundo argumento es el puerto donde escucha.
Para ejecutar debemos primero que dar al *Enter* en el ordenador servidor que esta el segundo comando e inmediatamente debemos dar al *Enter* en el ordenador cliente.

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

Dependiendo si enviamos los paquetes por local o por LAN, debido a que si enviamos los paquetes por local los paquetes no salen a la red y por lo tanto no tendrán la cabecera Ethernet. Si los paquetes se envían por LAN tendremos que añadir la cabecera Ethernet a la longitud del paquete. En el código hemos tenido en cuenta esa diferencia y hemos realizado la siguiente comprobación:

```
if ipListen == "127.0.0.1":
    tamanoPackets += IP_HDR_SIZE+UDP_HDR_SIZE+RTP_HDR_SIZE
else:
    tamanoPackets += IP_HDR_SIZE+UDP_HDR_SIZE+RTP_HDR_SIZE+ETH_HDR_SIZE
```

Así calculamos el ancho de banda con el tamaño del paquete completo y no solo con el nivel de aplicación. Dependiendo donde ejecutemos el código añadimos la cabecera Ethernet o no.

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

Depende si enviamos por local o por LAN. Si se envía por local no hay una trama mínima, pero si enviamos por LAN la trama mínima es de 46 bytes sin contar la capa de Ethernet en ninguno de los dos casos.

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

Tiene sentido enviar un máximo de 1500 bytes, debido a que si enviamos más bytes el paquete se fragmentará y no vamos a obtener el resultado deseado.

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

Dependiendo del parámetro a medir, puede ocurrir que se necesario utilizar longitudes distintas llegado el caso.
Los mejores resultados se consiguen con trenes de entre 100 y 1000 paquetes y el tamaño de los paquetes entre 1200 y 1400 bytes, debido a que obtenemos resultados muy acordes a las velocidades que pueden transmitir las tarjetas de red de los ordenadores de los laboratorios.

4. 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 2 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*.

El método utilizado para transmitir a una tasa determinado es hacer esperar un tiempo determinado entre paquete y paquete. El tiempo entre paquetes se calcula con la siguiente fórmula:

$$\Delta_r = \frac{Tamaño_{paquete}}{Tasa}$$

Por lo tanto, en el código hemos cambiado lo siguiente:

```
if dstIP == "127.0.0.1":
    time_wait = (dataIpLocal*8) / tasa_envio
else:
    time_wait = (dataIpEth*8) / tasa_envio
for i in range(0,trainLength):
    #usamos la longitud del tren como identificador de fuente. De esta manera en destino podemos saber la
    #longitud original del tren. En el campo timestamp (32bits) sólo podemos enviar segundos y
    #centésimas de milisegundos (o decenas de microsegundos, segun se quiera ver) truncados a 32bits
    message=struct.pack('!HHII',0x8014,seq_number, int(time.time()*DECENASMICROSECS)&B_MASK,trainLength)+data
    sock_send.sendto(message,addr)
    time.sleep(time_wait)
    seq_number+=1
```

5. 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.**

Para saber el ancho de banda que tenemos cada miembro de la pareja, primero tenemos que hacer pruebas con el clienteTren.py. A continuación, mostraremos el ancho de banda de cada miembro de la pareja:
Este ancho de banda corresponde al miembro de la pareja con el DNI: 53745614L. Es de alrededor de 300 Kbps.
Este ancho de banda corresponde al miembro de la pareja con el DNI: 53504265D. Es de alrededor de 200 Kbps.

```
Ancho de Bando Medio (b/s):  195016.70900885118
Ancho de Banda Medio (Kb/s):  195.01670900885117
Ancho de Banda Medio (Mb/s):  0.19501670900885118
Ancho de Banda Medio (Gb/s):  0.00019501670900885117
```

Al tener ya el ancho de banda solo es necesario ejecutar el clienteTren2.py con un ancho de banda menor al que nos ha salido con el clienteTren.py.

```
Ancho de Bando Medio (b/s):  282443.55337244587
Ancho de Banda Medio (Kb/s):  282.4435533724459
Ancho de Banda Medio (Mb/s):  0.28244355337244587
Ancho de Banda Medio (Gb/s):  0.00028244355337244585
```

DNI 1: 53745614L.
Ancho de banda estimado: 300 Kbps.
Retardo estimado: 60 ms.
Desviación estándar del retardo: 5 ms.
Porcentaje de pérdidas: 5%.

```
Número de paquetes recibidos:  952
Perdida de paquetes:  48
Porcentaje de Perdida de paquetes:  5.042016806722689
Retardo Máximo en (s):  0.07419
Retardo Mínimo en (s):  0.04368
Retardo Medio en (s):  0.06049212184873945
Variación del retardo:  0.00503258167056106
e336863@6A-26-6-26:~/RMult/P2/Code/codigoTrenPaquetes$
```

DNI 2: 53504265D.
Ancho de banda estimado: 200 Kbps.
Retardo estimado: 57 ms.
Desviación estándar del retardo: 17 ms.
Porcentaje de pérdidas: 2%.

```
Número de paquetes recibidos:  980
Perdida de paquetes:  20
Porcentaje de Perdida de paquetes:  2.0408163265306123
Retardo Máximo en (s):  0.10477
Retardo Mínimo en (s):  0.01502
Retardo Medio en (s):  0.05771401020408165
Variación del retardo:  0.017034929192448464
e336863@6A-26-6-26:~/RMult/P2/Code/codigoTrenPaquetes$
```

6. 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, calcule los valores de ancho de banda, retardo y jitter y compare estos datos con los resultados obtenidos con su programa.

Las medidas se han realizado en local, es decir, en un mismo ordenador. Por lo tanto, hemos abierto tres terminales más el programa Wireshark. En una terminal vamos a ejecutar el comando siguiente:
- *python3 clienteTren2.py 127.0.0.1 5004 1200 20000 100000*
En la otra terminal ejecutaremos el emulador que es el que nos ofrece perdidas y

retrasos a los paquetes que le llega del clienteTren2. - *./emulador 127.0.0.1 5004 127.0.0.1 5005 DNI_MiembroPareja* Y en la última terminal ejecutaremos el comando del servidor que es el siguiente: - *python3 servidorTren.py 127.0.0.1 5005* Por lo tanto, para coger las medidas de los paquetes que le llegan al servidorTren.py tenemos que el Wireshark capture los paquetes que lleguen al puerto 5005y eso se hace añadiendo el filtro siguiente: -*udp.dstport == 5005* Después, iniciamos la captura de tráfico, ejecutamos el comando del emulador, luego el comando del servidor y acto seguido ejecutaremos el comando del cliente y, así hasta esperar que haya terminado de recoger paquetes el servidorTren.py. Después, de tomar estas medidas nos damos cuenta de que nos salen parámetros muy parecidos a los que nos han salido en el ejercicio anterior.
DNI 1: 53745614L. Ancho de banda estimado: 300 Kbps. Retardo estimado: 60 ms. Desviación estándar del retardo: 5 ms. Porcentaje de pérdidas: 5%.
DNI 2: 53504265D. Ancho de banda estimado: 200 Kbps. Retardo estimado: 57 ms. Desviación estándar del retardo: 17 ms. Porcentaje de pérdidas: 2%.

7. 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 . Igualmente, indique el tamaño del buffer a configurar en el receptor de la llamada para amortiguar el efecto del *jitter*.

Podemos elegir entre dos codecs para el miembro de la pareja con el DNI1: 53745614L. Podemos elegir el codec G723.1 que tiene los siguientes parámetros: - Tiene 24 bytes de paquetización. - Tiene un MOS de 3.9. - Tiene un tiempo de paquetización de 30 ms. - Consume un ancho de banda de 21.9 Kbps. También se puede elegir el codec G729 que tiene los siguientes parámetros: - Tiene 20 bytes de paquetización. - Tiene un MOS de 3.92. - Tiene un tiempo de paquetización de 50 ms. - Consume un ancho de banda de 31.2 Kbps.
Las llamadas que se podrían realizar en el caso del codec G723.1. serían un total de 13 llamadas. Las llamadas que se podrían realizar en el caso del codec G729 serían un total de 9 llamadas.
Tendríamos un tamaño de buffer de alrededor de los 2000 bytes para que no haya

ninguna pérdida importante de paquetes.
Podemos elegir entre dos codecs para el miembro de la pareja con el DNI1: 53504265D. Podemos elegir el codec G723.1 que tiene los siguientes parámetros: - Tiene 24 bytes de paquetización. - Tiene un MOS de 3.9. - Tiene un tiempo de paquetización de 30 ms. - Consume un ancho de banda de 21.9 Kbps. También se puede elegir el codec G729 que tiene los siguientes parámetros: - Tiene 20 bytes de paquetización. - Tiene un MOS de 3.92. - Tiene un tiempo de paquetización de 50 ms. - Consume un ancho de banda de 31.2 Kbps.
Las llamadas que se podrían realizar en el caso del codec G723.1. serían un total de 13 llamadas. Las llamadas que se podrían realizar en el caso del codec G729 serían un total de 9 llamadas.
Tendríamos un tamaño de buffer de alrededor de los 2000 bytes para que no haya ninguna pérdida importante de paquetes.

8. 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.

Ahora para simular una llamada con los dos codecs elegidos con anterioridad tenemos que ejecutar siguiente comando en la terminal: - *python3 clienteTren2.py 127.0.0.1 5004 20000 24 21900* Aquí inyectamos paquetes de 24 bytes a una velocidad de 21.9 Kbps que es la tasa de envío a la que se envían los paquetes en el codec G723.1. Con esto simulamos una llamada de 10 minutos de duración con este codec. Ahora para simular una llamada con los dos codecs elegidos con anterioridad tenemos que ejecutar siguiente comando en la terminal: - *python3 clienteTren2.py 127.0.0.1 5004 20000 20 31200* Aquí inyectamos paquetes de 20 bytes a una velocidad de 31.2 Kbps que es la tasa de envío a la que se envían los paquetes en el codec G729. Con esto simulamos una llamada de 6 minutos de duración con este codec.
DNI: 53745614L Codec: G723.1

```
Ancho de Bando Medio (b/s):  19613.25333623328
Ancho de Banda Medio (Kb/s):  19.613253336233278
Ancho de Banda Medio (Mb/s):  0.019613253336233278
Ancho de Banda Medio (Gb/s):  1.961325333623328e-05
Ancho de Banda Instantáneo Máximo (b/s):  95869805.71428572
Ancho de Banda Instantáneo Mínimo (b/s):  4289.915491517189
Número de paquetes recibidos:  18964
Perdida de paquetes:  1036
Porcentaje de Perdida de paquetes:  5.4629824931449065
Retardo Máximo en (s):  0.08197
Retardo Mínimo en (s):  0.03613
Retardo Medio en (s):  0.058909533853617226
Variación del retardo:  0.006368406161722908
e336863@6A-26-6-26:~/RMult/P2/Code/codigoTrenPaquetes$
```

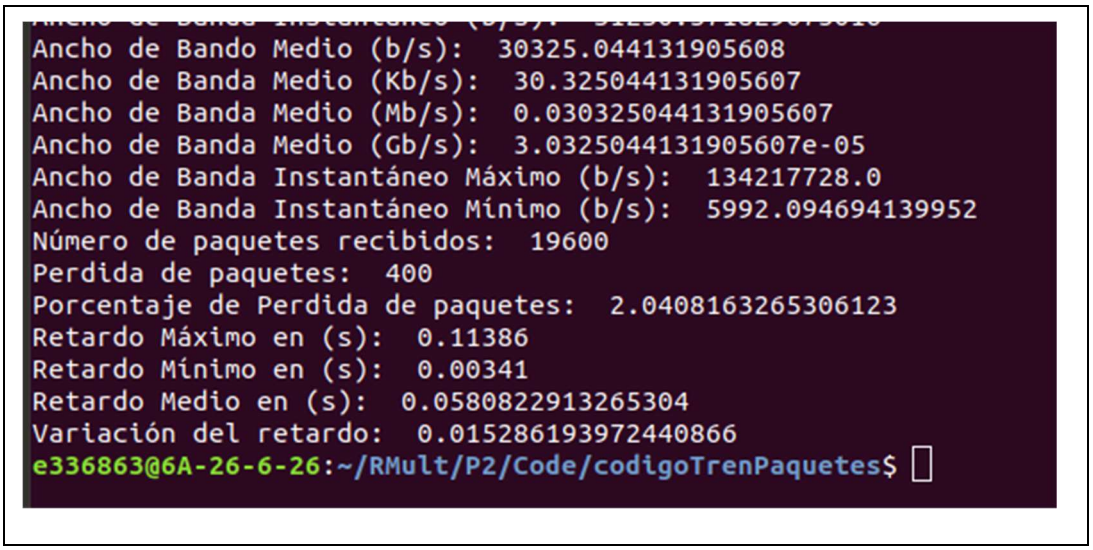
Codec: G729

```
Ancho de Bando Medio (b/s):  29486.140522292153
Ancho de Banda Medio (Kb/s):  29.486140522292153
Ancho de Banda Medio (Mb/s):  0.02948614052229215
Ancho de Banda Medio (Gb/s):  2.9486140522292152e-05
Ancho de Banda Instantáneo Máximo (b/s):  143804708.57142857
Ancho de Banda Instantáneo Mínimo (b/s):  6980.835301093277
Número de paquetes recibidos:  19061
Perdida de paquetes:  939
Porcentaje de Perdida de paquetes:  4.92628928177955
Retardo Máximo en (s):  0.08296
Retardo Mínimo en (s):  0.03249
Retardo Medio en (s):  0.057998377839567766
Variación del retardo:  0.006227920828074473
e336863@6A-26-6-26:~/RMult/P2/Code/codigoTrenPaquetes$
```

DNI: 53504265D
Codec: G723.1

```
Ancho de Bando Medio (b/s):  20278.511168706762
Ancho de Banda Medio (Kb/s):  20.27851116870676
Ancho de Banda Medio (Mb/s):  0.020278511168706763
Ancho de Banda Medio (Gb/s):  2.0278511168706763e-05
Ancho de Banda Instantáneo Máximo (b/s):  100663296.0
Ancho de Banda Instantáneo Mínimo (b/s):  5018.760956056119
Número de paquetes recibidos:  19610
Perdida de paquetes:  390
Porcentaje de Perdida de paquetes:  1.988781234064253
Retardo Máximo en (s):  0.1182
Retardo Mínimo en (s):  0.00355
Retardo Medio en (s):  0.05870281234064245
Variación del retardo:  0.015243444022613318
e336863@6A-26-6-26:~/RMult/P2/Code/codigoTrenPaquetes$
```

Codec: G729



3 Conclusiones

Las conclusiones que hemos sacado de esta práctica es como hacer buenas medidas para en la red que vamos a utilizar para poder meter un servicio de VoIP. La mejor manera de medir es con un tren de paquetes debido a que nos quitamos tráfico transversal y así poder medir bien el ancho de banda, los retardos, el jitter y las pérdidas de la red sobre la que estamos trabajando. Hemos aprendido que como saturemos la red con la llegada de múltiples paquetes podemos hacer disparar la perdida de paquetes debido a que llenamos el buffer del socket muy rápido y no puede hacerse con ello lo desecha. También hemos visto que el retraso puede ser negativo entre dos ordenadores debido a que no pueden estar sincronizados, pero la desincronización es de la orden de microsegundos no la tenemos mucho en cuenta a la hora de las medidas. Por lo tanto, hemos visto como varía los retardos, el ancho de banda, el jitter y las perdidas en una red local y una red entre dos ordenadores modificando el tamaño del enlace o modificando la velocidad a la que enviamos los paquetes, si enviamos los paquetes con el primer cliente tren los envía al máximo de la red y eso puede saturar el servidor.