Proyecto Final

Carlos Saenz

***Resumen-*** En el presente informe usted verá el análisis de transferencia de paquetes para nos archivos alojados en un servidor. Bajo el software de Python, donde realizaremos dos scripts para dos clientes FTP y con sockets de flujo, estableceremos conexión servidor para poner en ejecución la transferencia del archivo vía cliente/servidor, adicionalmente estudiaremos el jitter. Además, durante la transferencia del archivo, mediante Wireshark analizaremos las tasas de transferencia de datos por paquete o en promedio.

# Elaboración de script FTP

Para la elaboración del cliente FTP se utilizaron los módulos *ftplib* y *os.* Inicialmente, como puede ver en las líneas 7, 8 y 9 se crearon tres variables donde se asignaba el URL del servidor donde necesitábamos acceder, el usuario y la clave. Ya mencionado anteriormente, nosotros, para poder acceder a las librerías del servidor, era necesario una identificación. Nuestros identificadores era*user: c\_iele1400* y *password: uniandes2020*. Posteriormente se utilizaron las palabras reservadas *try/except*, debido a que en condiciones ideales siempre se podría establecer conexión, pero era necesario pensar en aquellas excepciones que no nos permitirían establecer conexión. Para eso, se utilizó la palabra reservada *Exception*, que retornaría al cliente un mensaje con la razón de por qué no se pudo establecer conexión. Una vez iniciado sesión en el servidor, mediante el comando *ftp.welcome()* nos retornaría el mensaje de bienvenida estándar establecido por el servidor. En las líneas 17, 18 y 19 se ingresó al directorio donde se encontraban los archivos y mediante el comando *ftp.retrlines(“LIST”)* tendríamos como retorno una lista de aquellos archivos presentes en el directorio del servidor. Posteriormente se le pide al cliente que ingrese el archivo que desea descargar como la ruta en donde sea guardar la descarga, ahí toma uso la librería *os.chdir*, que nos da la función de permitir el acceso a la ruta de donde el cliente quiere guardar la descarga y guardar la descarga allí. Una vez identificada la ruta del archivo y la ruta de donde dese el cliente guardar la descarga, nuestro cliente a partir de la línea 30, empieza con la transferencia del archivo. Este abre el archivo que queremos transferir a nuestra computadora y lo transcribe en Bytes mediante el comando *with open(archivo, “wb”) as f:* y el comando *ftp.retrbinary(“RETR “ + archivo, f.write).* Este último archivo transfiere el archivo a descargar a nuestra computadora en modo binario. Una vez realizada la descarga, se retorna a nuestro cliente un mensaje confirmando la transferencia e informando el peso en bytes del archivo. El siguiente except, presente en la línea 40, toma como excepción cualquier error dado por el protocolo FTP. Una vez finalizada la transferencia, se cierra unilateralmente la conexión FTP con el servidor.

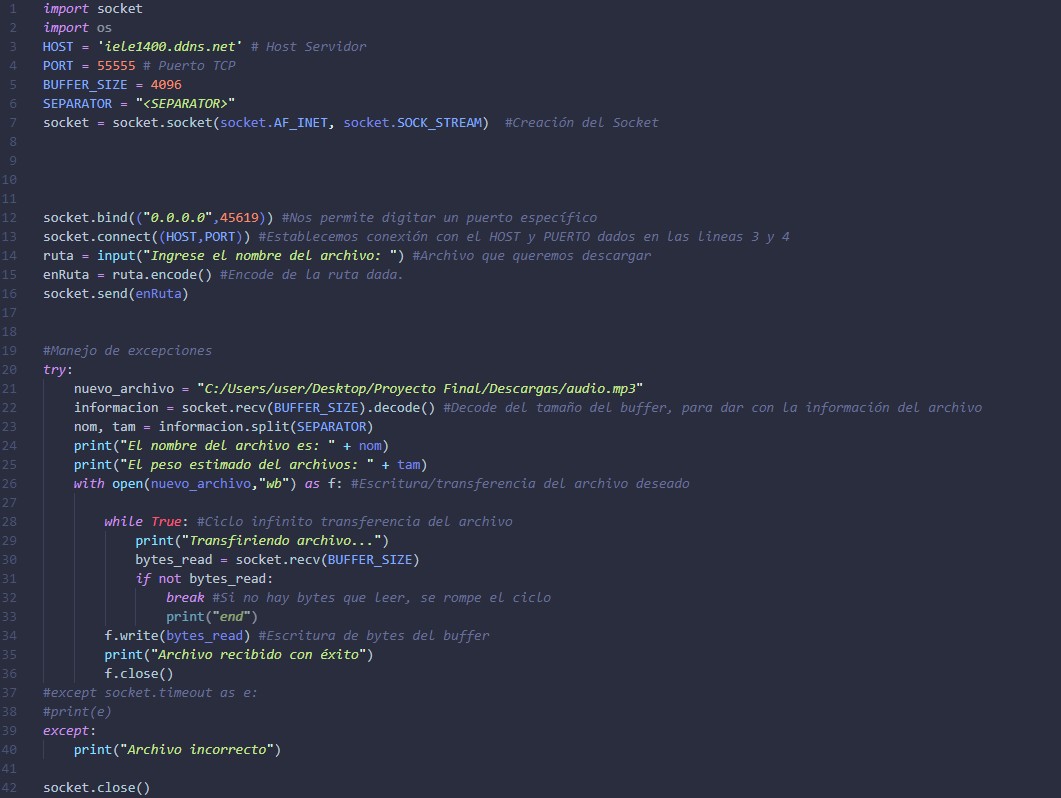


**Figura 1:** Script del cliente FTP

# Elaboración del cliente mediante Sockets de Flujo

La elaboración del cliente mediante sockets de flujo fue sin duda el reto a programar. Para este script se utilizó la librería socket. Inicialmente se estableció el nombre del host para *iele1400.ddns.net*, con el número de puerto. Adicionalmente, antes de crear el socket, se creó una variable llamada SEPARATOR, asignándole *<SEPARATOR>*. En la línea 7 puede ver cómo se creó el socket, el comando *AF\_INET* especificaba que trabajaríamos con una dirección IPv4 y el *SOCK\_STREAM* especificaba que trabajaríamos sobre sockets TCP. En las líneas 9 a 13 se establece conexión con el servidor, ingresando un *socket.bind* que nos ayudaría a asignar el socket con una dirección IP, además de darnos la opción de asignar un puerto TCP personalizado, no uno definido por el servidor [5]. Posterior a esto se pide el nombre del archivo a descargar, para así codificar la información con

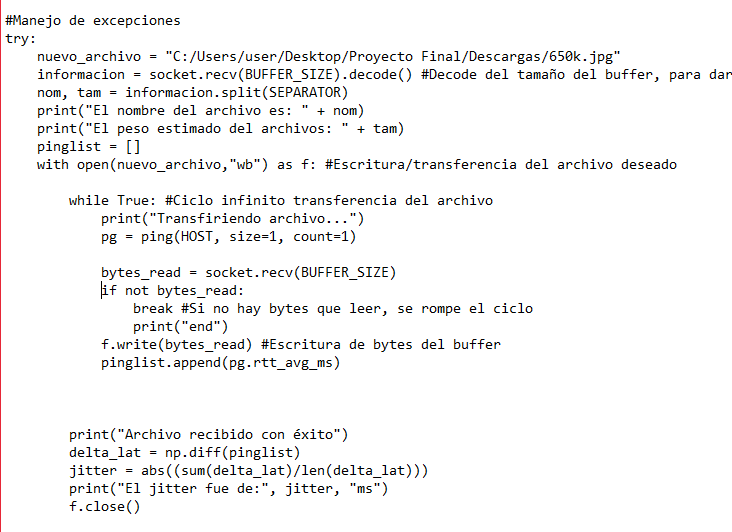
*.encode()* y enviar el socket. A continuación, nos tocaba pensar en dos cosas, ¿qué pasa si el archivo existe o NO existe? Para eso se utilizó el método *try/except*. Si el archivo existe, se le asignará la ruta en donde queremos guardar nuestra descarga, decodificamos mediante *.decode()* la información del buffer para saber tanto el nombre del archivo como su peso correspondiente en bytes. Posterior a esto, con el comando *with(open)* ubicado en la línea 21, se abre un archivo nuevo en la ruta que usted especificó para guardar el archivo, y este nuevo archivo se abre con un formado de *write bytes*, es decir, escribirá información del archivo en bytes. Se crea un ciclo infinito para escribir y transferir el archivo. Una vez realizada la transferencia, simplemente se cierra el archivo creado para evitar que se corrompa. Finalmente, el except considerando un posible archivo incorrecto. Por último, para evitar errores en el socket, se cierra la conexión con el comando socket.close(). Es importante notar que el socket.bind() tendrá un número de puerto asignado por nosotros, en este caso serán números al azar, exceptuando el último dígito, ya que este dígito me identificaría a mí con el número 9, es decir, xyzw9.



**Figura 2:** Script del cliente con Sockets de Flujo

Para la elaboración del cálculo de Jitter se utilizaron los módulos *pythonping* y *numpy* dada la facilidad que nos da para los cálculos. Inicialmente, después de la línea 28 se inserta el comando

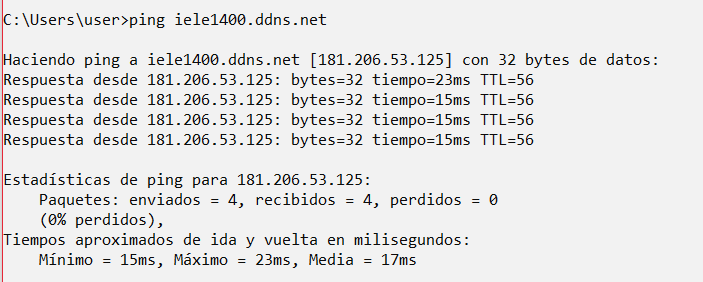
pg = ping(HOST, size=1, count=1). Previamente se habría creado una lista vacía en donde se insertará la latencia promedio mediante el comando pinglist.append(pg.rtt\_avg\_ms). Una vez la lista con los datos, simplemente, con ayuda de numpy, aplicamos la fórmula que está en la guía.



**Figura 3:** Actualización código cálculo Jitter

# Análisis de conectividad y tasa de transferencia de datos instantánea/promedio

Inicialmente es necesario conocer si hay conectividad con el servidor, para esto simplemente ejecutamos el comando ping en nuestra consola hacia el servidor.



**Figura 4:** Comando *ping.*

En la imagen 3 confirmamos conexión con el servidor, así que procedemos al análisis de paquetes.

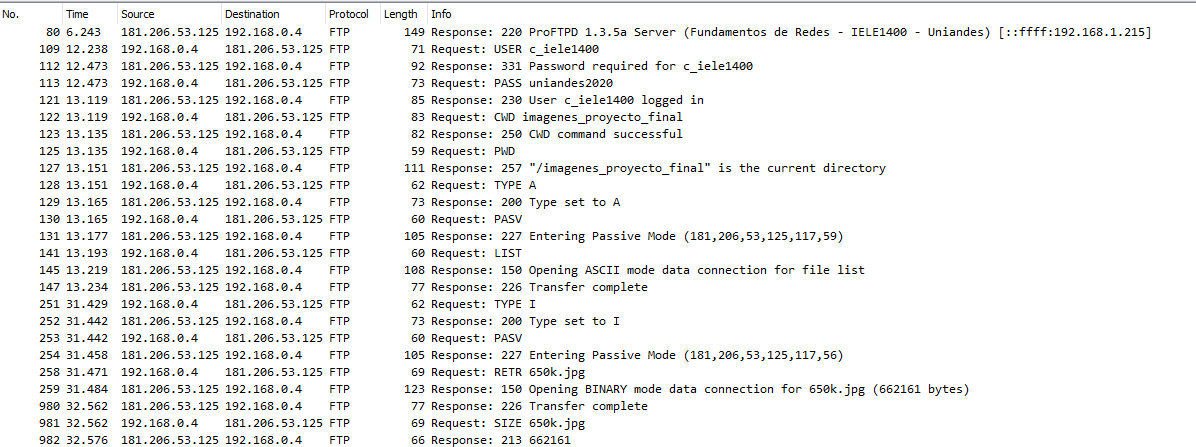
Dado que es claro que tenemos dos archivos en nuestro servidor, se realizarán pruebas y análisis de transferencias de datos, con su respectivo jitter. Este análisis se dividirá por archivo, empezando por el archivo más pequeño, con formato jpg, 650k.jpg. Posterior, se hará con el archivo de formato mp4, 45m.mp4

1. Para el archivo de formato jpg se harán 3 tomas de paquetes en distintas horas para considerar posibles cambios por la congestión de red tanto mía, como del servidor. **Primera prueba hecha el día jueves 25 de mayo de 2021 a las 10:00 pm**

Para el cliente FTP:



**Figura 5:** Filtro realizado en Wireshark.



**Figura 6:** Captura de paquetes realizada en el cliente FTP. Toma #1

Posterior a realizar el filtro, como lo muestra la figura 4, nos arroja los paquetes dados en la figura 5. Estos paquetes son los asociados a la transferencia en la red al ejecutar nuestro script FTP Python. A partir de lo anterior, se puede establecer lo siguiente:

|  |  |  |  |
| --- | --- | --- | --- |
| Tiempo relativo acumulado | Tamaño del Paquete | Tiempo de transferencia individual | Tasa instantánea |
| 6.243 | 149 | 0.0732 | 2035.519126 |
| 12.238 | 71 | 5.9455 | 11.94180473 |
| 12.473 | 92 | 0.003 | 30666.66667 |
| 12.473 | 73 | 0.000111 | 657657.6577 |
| 13.119 | 85 | 0.4038 | 210.5002476 |
| 13.119 | 83 | 0.000194 | 427835.0515 |
| 13.135 | 82 | 0.01569 | 5226.258764 |
| 13.135 | 59 | 0.00001 | 5900000 |
| 13.151 | 111 | 0.0159 | 6981.132075 |
| 13.151 | 62 | 0.000076 | 815789.4737 |

|  |  |  |  |
| --- | --- | --- | --- |
| 13.165 | 73 | 0.01371 | 5324.580598 |
| 13.165 | 60 | 0.000093 | 645161.2903 |
| 13.177 | 105 | 0.011714 | 8963.633259 |
| 13.193 | 60 | 0.016225 | 3697.996918 |
| 13.219 | 108 | 0.02594 | 4163.454125 |
| 13.234 | 77 | 0.01553 | 4958.145525 |
| 31.429 | 62 | 18.194 | 3.40771683 |
| 31.442 | 73 | 0.013134 | 5558.093498 |
| 31.442 | 60 | 0.000124 | 483870.9677 |
| 31.458 | 105 | 0.01598 | 6570.713392 |
| 31.471 | 69 | 0.012795 | 5392.731536 |
| 31.484 | 123 | 0.013087 | 9398.639872 |
| 32.562 | 77 | 1.03559 | 74.35375004 |
| 32.562 | 69 | 0.000209 | 330143.5407 |
| 32.576 | 66 | 0.013545 | 4872.646733 |

Como podemos ver, el tiempo relativo acumulado es el mismo que vemos en la figura 5, así como el tamaño del paquete. ¿Qué es el tiempo de transferencia individual y la tasa instantánea por paquete? El tiempo de transferencia individual nos lo da Wireshark, este valor se asigna para el protocolo TCP como Timestamps, que es el tiempo que lleva desde que se recibió el paquete. La tasa instantánea está dada por la fórmula:

⁄

𝑡𝑎𝑚𝑎ñ𝑜 𝑑𝑒𝑙 𝑝𝑎𝑞𝑢𝑒𝑡𝑒

𝑡𝑖𝑒𝑚𝑝𝑜 𝑑𝑒 𝑡𝑟𝑎𝑛𝑠𝑓𝑒𝑟𝑒𝑛𝑐𝑖𝑎 𝑖𝑛𝑑𝑖𝑣𝑖𝑑𝑢𝑎𝑙

. Este resultado tiene unidades de 𝐵𝑦𝑡𝑒𝑠 𝑠𝑒𝑔𝑢𝑛𝑑𝑜.

Es pertinente encontrar nuestra tasa de transferencia promedio. Para esto, se hará el siguiente procedimiento matemático:

̅̅̅̅̅̅̅

650 𝐾𝐵

𝐾𝐵⁄

25.164 𝐾𝐵⁄𝑠

𝑇𝑎𝑠𝑎 = 25.83 𝑠 = 25.164

𝑠 ,

125 = 0.20128 𝑀𝑏𝑝𝑠



Archivo 650k.jpg

Tasa instantánea respecto al tiempo Tasa Media

1000000

900000

800000

700000

600000

500000

400000

300000

200000

100000

0

0,000

0,005

0,010

0,015

0,020

0,025

0,030

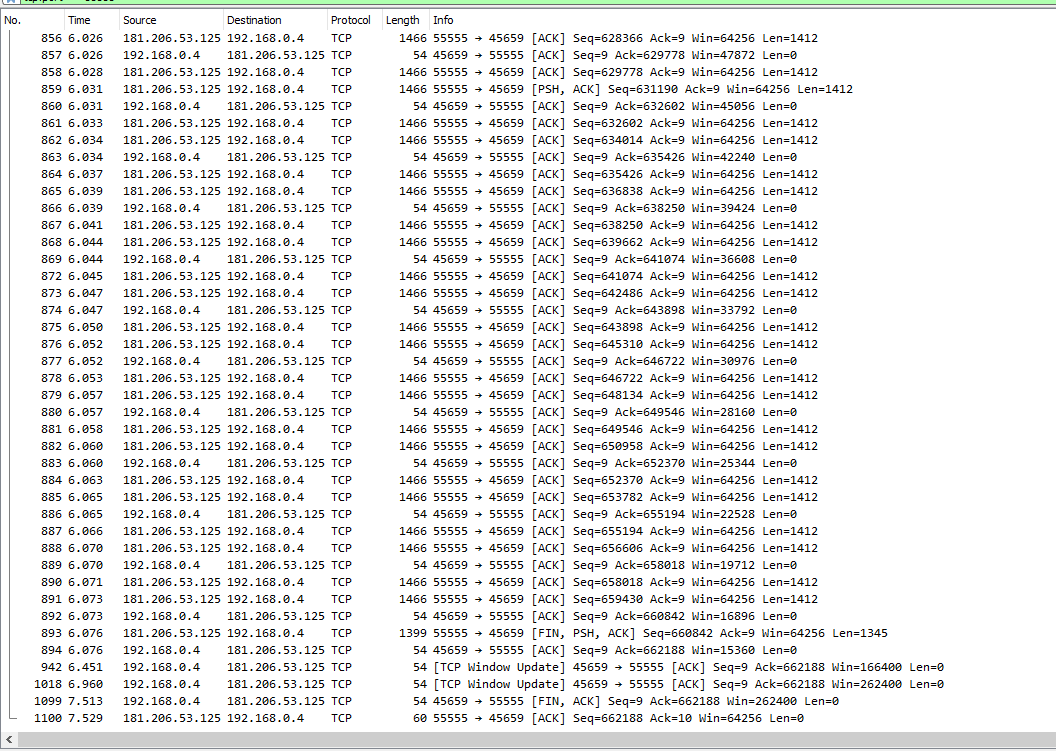
0,035

La gráfica anterior demuestra el comportamiento expuesto anterior. Nuestros valores en naranja representan nuestra tasa de transferencia promedio, mientras que los puntos azules es cada tasa de transferencia de datos por paquete.

Para el cliente con Sockets de Flujo:



**Figura 7:** Filtro realizado en Wireshark.



**Figura 8:** Captura de paquetes realizada con el cliente con Sockets de Flujo. Toma #1

Esta captura contiene muchos más paquetes que los que contiene la figura 5, por eso se realizó un filtro mucho más específico con el puerto del servidor. Estudiemos la diferencia en las tasas de transferencia con este cliente. Dada la gran cantidad de paquetes asociado a este puerto, se hará el análisis con 60 paquetes, asignando 20 al tramo inicial, 20 al tramo medio y 20 al tramo final.

|  |  |  |  |
| --- | --- | --- | --- |
| Tiempo relativo acumulado | Tamaño del Paquete | Tiempo de transferencia individual | Tasa instantánea |
| 1.455 | 66 | 0 | #¡DIV/0! |
| 1.467 | 66 | 0.011 | 6000 |
| 1.467 | 54 | 0.01132 | 4770.318021 |
| 4.552 | 62 | 3.085 | 20.09724473 |
| 4.565 | 60 | 3.10966 | 19.29471389 |
| 4.565 | 79 | 0.00016 | 493750 |
| 4.565 | 1466 | 0.000167 | 8778443.114 |
| 4.565 | 54 | 0.00017 | 317647.0588 |
| 4.568 | 1466 | 0.0027 | 542962.963 |
| 4.569 | 1466 | 0.000712 | 2058988.764 |
| 4.569 | 54 | 0.00011 | 490909.0909 |
| 4.573 | 1466 | 0.0046 | 318695.6522 |
| 4.573 | 1466 | 0 | #¡DIV/0! |
| 4.573 | 54 | 0.000015 | 3600000 |
| 4.574 | 1466 | 0.001113 | 1317160.827 |
| 4.575 | 1466 | 0.000087 | 16850574.71 |
| 4.575 | 54 | 0.00008 | 675000 |
| 4.576 | 1466 | 0.001143 | 1282589.676 |
| 4.576 | 1466 | 0.0008 | 1832500 |
| 4.576 | 54 | 0.00005 | 1080000 |
| 4.582 | 1466 | 0.006271 | 233774.5176 |
| 5.008 | 1466 | 0.000023 | 63739130.43 |
| 5.008 | 54 | 0.000005 | 10800000 |
| 5.011 | 1466 | 0.002884 | 508321.7753 |
| 5.014 | 1466 | 0.003229 | 454010.5296 |
| 5.014 | 54 | 0.00007 | 771428.5714 |
| 5.015 | 1466 | 0.000784 | 1869897.959 |
| 5.018 | 1466 | 0.00237 | 618565.4008 |
| 5.018 | 54 | 0.00005 | 1080000 |
| 5.021 | 1466 | 0.003494 | 419576.4167 |
| 5.021 | 1466 | 0.000022 | 66636363.64 |
| 5.021 | 54 | 0.000005 | 10800000 |
| 5.022 | 54 | 0.000431 | 125290.0232 |
| 5.024 | 1466 | 0.00248 | 591129.0323 |
| 5.027 | 1466 | 0.003172 | 462168.9786 |

|  |  |  |  |
| --- | --- | --- | --- |
| 5.027 | 54 | 0.000006 | 9000000 |
| 5.028 | 1466 | 0.000794 | 1846347.607 |
| 5.03 | 1466 | 0.002417 | 606537.0294 |
| 5.03 | 54 | 0.000005 | 10800000 |
| 5.034 | 1466 | 0.003451 | 424804.4045 |
| 5.034 | 1466 | 0.003471 | 422356.6695 |
| 6.057 | 54 | 0.000006 | 9000000 |
| 6.058 | 1466 | 0.001491 | 983232.7297 |
| 6.06 | 1466 | 0.001721 | 851830.3312 |
| 6.06 | 54 | 0.000004 | 13500000 |
| 6.063 | 1466 | 0.003194 | 458985.598 |
| 6.065 | 1466 | 0.001493 | 981915.6062 |
| 6.065 | 54 | 0.000005 | 10800000 |
| 6.066 | 1466 | 0.001727 | 848870.8743 |
| 6.07 | 1466 | 0.003213 | 456271.3974 |
| 6.07 | 54 | 0.0000006 | 90000000 |
| 6.071 | 1466 | 0.001439 | 1018763.03 |
| 6.073 | 1466 | 0.001698 | 863368.669 |
| 6.073 | 54 | 0.00007 | 771428.5714 |
| 6.076 | 1399 | 0.003737 | 374364.4635 |
| 6.076 | 54 | 0.000013 | 4153846.154 |
| 6.451 | 54 | 0.37453 | 144.1807065 |
| 6.96 | 54 | 0.50917 | 106.0549522 |
| 7.513 | 54 | 0.553112 | 97.62941321 |
| 7.529 | 60 | 0.015597 | 3846.893633 |

Realizado el procedimiento anterior, se calcularon las tasas de transferencia instantánea por paquete y el tiempo de transferencia individual. Calculemos la tasa de datos promedio, se puede hipotetizar que es parecida a la ya expuesta anteriormente. En este caso, dado que no utilizamos todos los paquetes, en el tamaño usaremos el tamaño total de los paquetes dividido en el tiempo acumulado de todos los paquetes, así

̅𝑇̅̅𝑎̅̅𝑠̅𝑎̅ =

702.298 𝐾𝐵

7.73 𝑠 = 90.85

𝐾𝐵⁄𝑠 ,

90.85 𝐾𝐵⁄𝑠

125 = 0.72 𝑀𝑏𝑝𝑠



Archivo 650k.jpg

90000000

80000000

70000000

60000000

50000000

40000000

Tasa Instantánea

Tasa Media

30000000

20000000

10000000

0

1

2

3

4

5

6

7

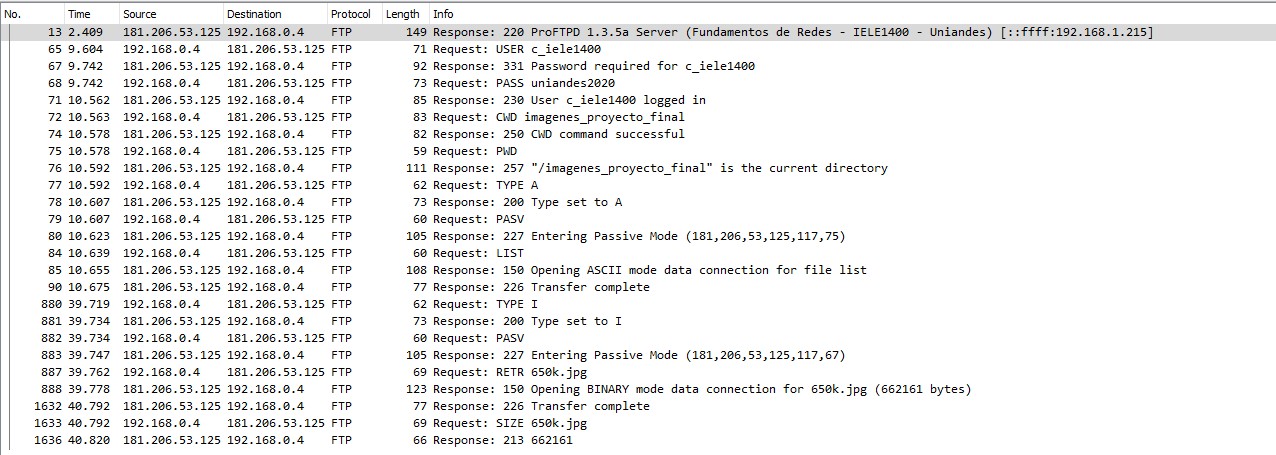
8

Para nuestra primera toma, para el archivo de 650 KB, hubo una diferencia en la tasa de transferencia de 0.52 𝑀𝑏𝑝𝑠. Haremos este mismo análisis en dos tomas más.

Además, importante recalcar que tuvimos un Jitter de 0.013 ms. Estudiaremos el cambio del Jitter en distintas horas.

# Segunda prueba hecha el día 21 de agosto de 2021 a las 10:00 am

Para el cliente FTP:



**Figura 9:** Captura de paquetes realizada con el cliente FTP. Toma #2

En este caso, no necesitaremos calcular individualmente la tasa de transferencia por paquete, simplemente calcularemos la tasa de transferencia promedio a ver cuánto ha variado.

Sumando los timestamps de los 24 paquetes obtenidos, tenemos lo siguiente:

̅̅̅̅̅̅̅

650 𝐾𝐵

𝐾𝐵⁄

25.164 𝐾𝐵⁄𝑠

𝑇𝑎𝑠𝑎 = = 16.91

38.42 𝑠

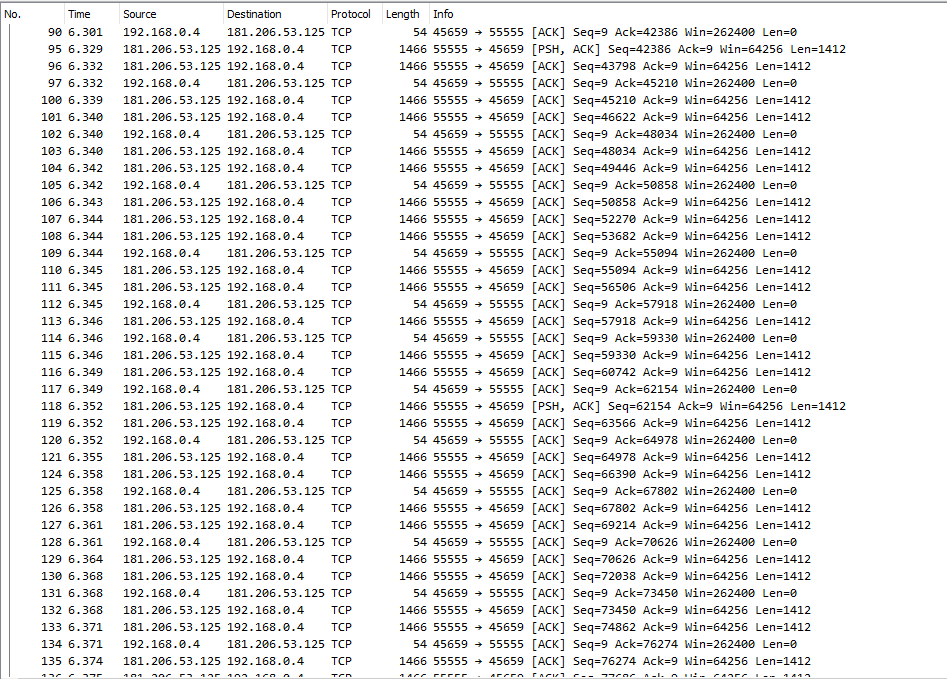
Para el cliente con Sockets de Flujo:

𝑠 ,

= 0.13 𝑀𝑏𝑝𝑠

125

Para el filtro de la figura 6, tenemos lo siguiente:



**Figura 10:** Captura de paquetes realizada con el cliente de Sockets de Flujo. Toma #2

Al ser 719 paquetes, se tomará al igual que para la primera toma, 60 paquetes, se sumarán sus timestamps y calcularemos la tasa de transferencia promedio.

̅𝑇̅̅𝑎̅̅𝑠̅𝑎̅ =

505.680 𝐾𝐵

6.28 𝑠 = 80.52

𝐾𝐵⁄𝑠 ,

90.85 𝐾𝐵⁄𝑠

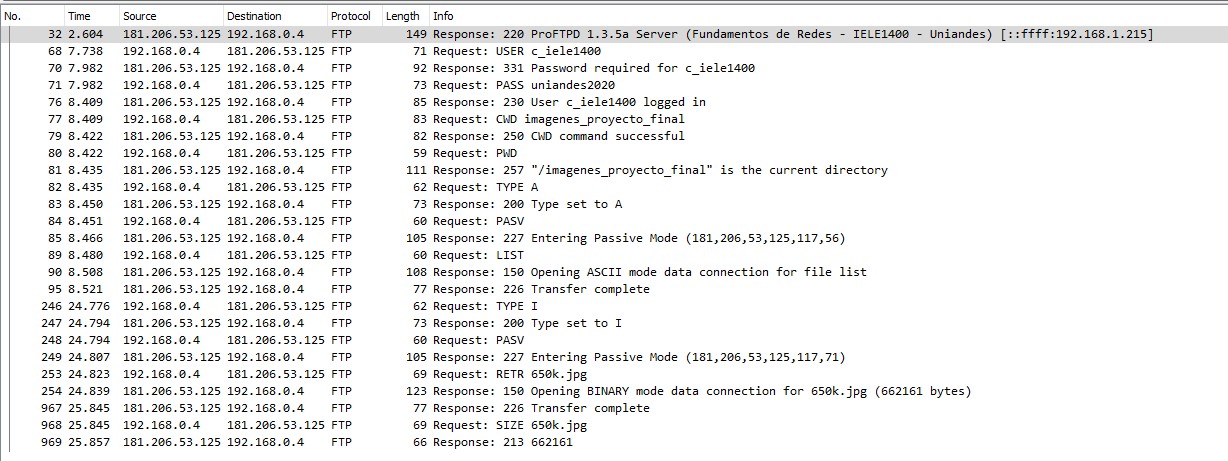
125 = 0.644 𝑀𝑏𝑝𝑠

En esta toma #2 las tasas de transferencia de datos bajaron considerablemente. La tercera muestra se realizará en horas de la madrugada, suponiendo una tasa más alta a las expuestas anteriormente.

Además, el Jitter tuvo un valor de 0.015.

# Toma #3 Realizada viernes 28 de may. de 2021 a las 11:00 pm

Para el cliente FTP:



**Figura 11:** Captura de paquetes realizada con el cliente FTP. Toma #3

̅𝑇̅̅𝑎̅̅𝑠̅𝑎̅ =

650 𝐾𝐵

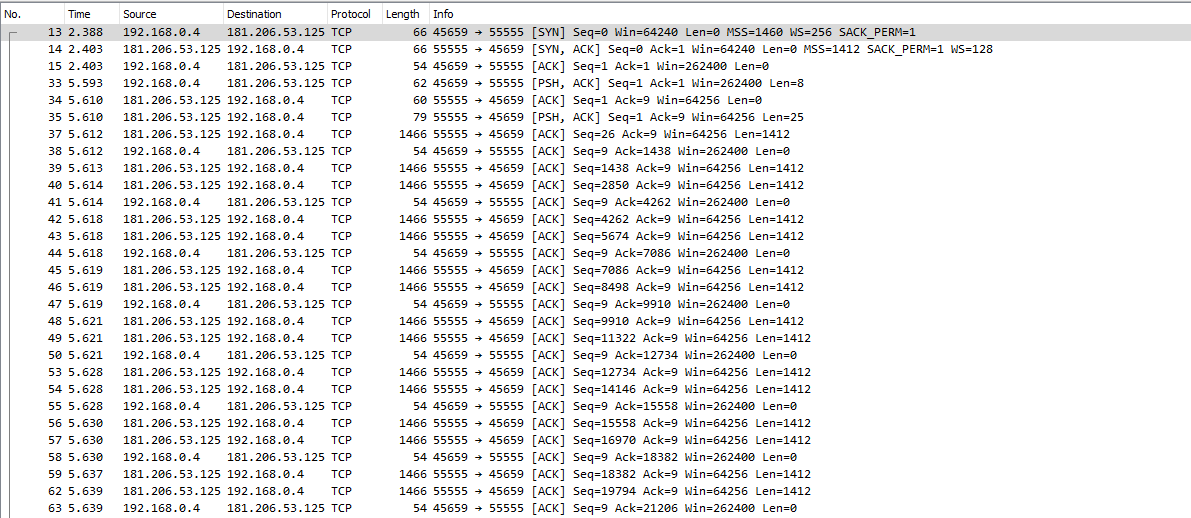
22.97 = 28.29

𝐾𝐵⁄𝑠 ,

25.164 𝐾𝐵⁄𝑠

125 = 0.22 𝑀𝑏𝑝𝑠

Para el cliente con Sockets de Flujo:



**Figura 12:** Captura de paquetes realizada con el cliente con Sockets de Flujo. Toma #3

̅𝑇̅̅𝑎̅̅𝑠̅𝑎̅ =

687.503 𝐾𝐵

6.69 𝑠 = 102.76

𝐾𝐵⁄𝑠 ,

90.85 𝐾𝐵⁄𝑠

125 = 0.822 𝑀𝑏𝑝𝑠

Al haber realizado 3 tomas diferentes, se puede concluir lo siguiente:

* 1. La tasa de transferencia con Sockets de Flujo es más alta.
  2. La hora hizo que la tasa de transferencia variara.
  3. Así como la tasa de transferencia con sockets de flujo es más alta, también genera muchos más paquetes que el cliente FTP.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Hora | FTP | Sockets |
| Toma #1 | Jueves, 27 may 10:00 pm | 0.20 Mbps | 0.72 Mbps |
| Toma #2 | Viernes, 28 may 10:00 am | 0.13 Mbps | 0.64 Mbps |
| Toma #3 | Viernes, 28 may 11:00 pm | 0.22 Mbps | 0.82 Mbps |

Es bastante claro que la toma 3 tendría una mayor tasa de transferencia, ya que se puede asumir que nadie o muy poca gente estaría congestionando tanto mi red como la red del servidor. Por esto se hipotetizó con anterioridad que en horas de la madrugada sería más alta la tasa de transferencia.

La diferencia en los procesos de ambos scripts, y la razón de por qué los sockets tienen mayor tasa de transferencia, es debido a su forma de actuar. Nuestro cliente FTP tiene la capacidad para entrar al servidor para descargar archivos, lo que hacía nuestro cliente con sockets de flujo era establecer una conexión TCP cliente/servidor para que el cliente nos transmitiera los datos.

Por otro lado, nuestro Jitter realmente no cambió, debido a que no hubo tanto retraso en la transferencia de paquetes.

1. Para el archivo de formato mp4 se harán 3 tomas de paquetes en distintas horas para considerar posibles cambios por la congestión de red tanto mía, como del servidor. **Primera prueba hecha el día viernes 28 de mayo de 2021 a las 8:00 pm**

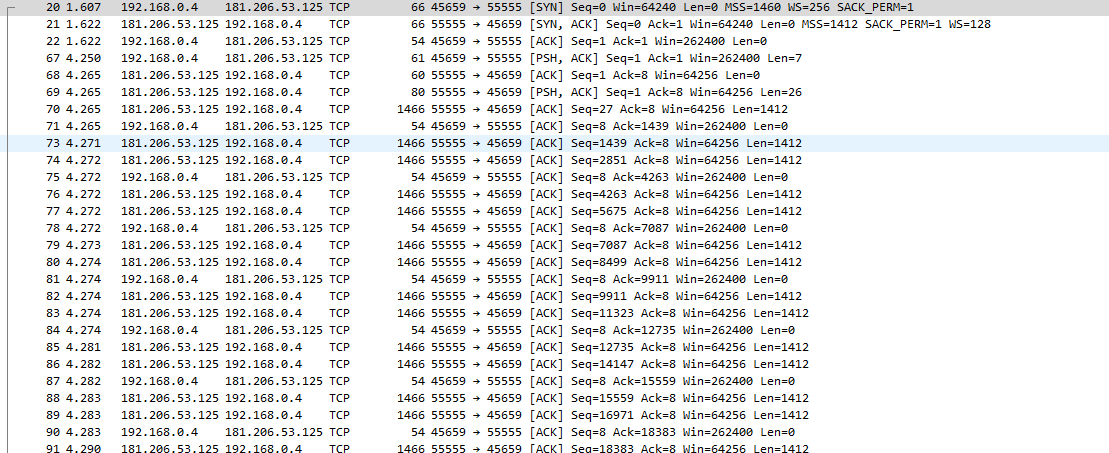
Para el cliente con Sockets de Flujo:

Aplicamos el siguiente filtro para ver los paquetes asociados a la comunicación cliente/servidor:



**Figura 13:** Filtro realizado en Wireshark.

Al aplicar el filtro, observamos los siguientes 50976 paquetes:



**Figura 14:** Captura de paquetes realizada con el cliente de Sockets de Flujo.

Realizaremos el mismo procedimiento ya realizado anteriormente, lo único que cambia es que trabajaremos con un archivo más grande, con formato mp4. Se seleccionarán 60

paquetes aleatorios. Es importante aclarar que esa cantidad de paquetes se debe a que es un archivo más grande que el ya analizado.

Tras analizar los paquetes, tenemos lo siguiente:

|  |  |  |  |
| --- | --- | --- | --- |
| Tiempo Relativo Acumulado | Tamaño del Paquete | Tiempo de transferencia individual | Tasa instantánea |
| 1.607 | 66 | 0 | #¡DIV/0! |
| 1.622 | 66 | 0.015092 | 4373.177843 |
| 1.622 | 54 | 0.000049 | 1102040.816 |
| 4.25 | 61 | 2.627985 | 23.21170022 |
| 4.265 | 60 | 0.01479 | 4056.795132 |
| 4.265 | 80 | 0.000081 | 987654.321 |
| 4.265 | 1466 | 0.000104 | 14096153.85 |
| 4.265 | 54 | 0.000011 | 4909090.909 |
| 4.271 | 1466 | 0.006309 | 232366.4606 |
| 4.272 | 1466 | 0.000863 | 1698725.377 |
| 4.272 | 54 | 0.00001 | 5400000 |
| 4.272 | 1466 | 0.000404 | 3628712.871 |
| 4.272 | 1466 | 0.000108 | 13574074.07 |
| 4.272 | 54 | 0.00008 | 675000 |
| 4.273 | 1466 | 0.000684 | 2143274.854 |
| 4.274 | 1466 | 0.000374 | 3919786.096 |
| 4.274 | 54 | 0.00008 | 675000 |
| 4.274 | 1466 | 0.000079 | 18556962.03 |
| 4.274 | 1466 | 0.000203 | 7221674.877 |
| 4.274 | 54 | 0.00005 | 1080000 |
| 4.405 | 54 | 0.00008 | 675000 |
| 4.406 | 1466 | 0.000877 | 1671607.754 |
| 4.408 | 1466 | 0.002263 | 647812.6381 |
| 4.408 | 54 | 0.00006 | 900000 |
| 4.411 | 1466 | 0.003523 | 416122.6228 |
| 4.412 | 1466 | 0.000021 | 69809523.81 |
| 4.412 | 54 | 0.00005 | 1080000 |
| 4.414 | 1466 | 0.002846 | 515108.9248 |
| 4.418 | 1466 | 0.003219 | 455420.9382 |
| 4.418 | 54 | 0.000007 | 7714285.714 |
| 4.418 | 1466 | 0.000875 | 1675428.571 |
| 4.42 | 1466 | 0.001752 | 836757.9909 |
| 4.42 | 54 | 0.000006 | 9000000 |
| 4.425 | 1466 | 0.005136 | 285436.1371 |
| 4.425 | 1466 | 0 | #¡DIV/0! |

|  |  |  |  |
| --- | --- | --- | --- |
| 4.425 | 54 | 0.000014 | 3857142.857 |
| 4.427 | 1466 | 0.001295 | 1132046.332 |
| 4.432 | 1466 | 0.005086 | 288242.2336 |
| 4.432 | 1466 | 0 | #¡DIV/0! |
| 4.432 | 54 | 0.00001 | 5400000 |
| 292.259 | 54 | 0.0783009 | 689.6472454 |
| 292.272 | 186 | 0.013599 | 13677.47629 |
| 292.272 | 1466 | 0.00004 | 36650000 |
| 292.272 | 54 | 0.00007 | 771428.5714 |
| 292.273 | 1466 | 0.000946 | 1549682.875 |
| 292.274 | 1466 | 0.000326 | 4496932.515 |
| 292.274 | 54 | 0.00007 | 771428.5714 |
| 292.276 | 1466 | 0.001911 | 767137.6243 |
| 292.277 | 1466 | 0.001441 | 1017349.063 |
| 292.277 | 54 | 0.000009 | 6000000 |
| 292.278 | 1466 | 0.001016 | 1442913.386 |
| 292.278 | 1466 | 0 | #¡DIV/0! |
| 292.278 | 54 | 0.000007 | 7714285.714 |
| 292.279 | 1466 | 0.001211 | 1210569.777 |
| 292.28 | 284 | 0.000396 | 717171.7172 |
| 292.28 | 54 | 0.000007 | 7714285.714 |
| 292.509 | 54 | 0.229841 | 234.9450272 |
| 293.16 | 54 | 0.650657 | 82.99303627 |
| 293.44 | 54 | 0.279585 | 193.143409 |
| 293.462 | 60 | 0.022439 | 2673.91595 |

Ahora, para calcular la tasa de transferencia promedio, realizamos lo siguiente:

̅𝑇̅̅𝑎̅̅𝑠̅𝑎̅ = 49018407 × 8 = 1′336.272 𝑏𝑝𝑠 ≈ 1.32𝑀𝑏𝑝𝑠

293.42 𝑠

49018407 B es la suma total de todos los paquetes, multiplicado por 8 para tener esa unidad en bites, 293.42 es el tiempo acumulado de la transferencia.



Archivo 45m.mp4

3

1350000

2.5

1340000

2

1330000

1.5

1320000

1

0.5

1310000

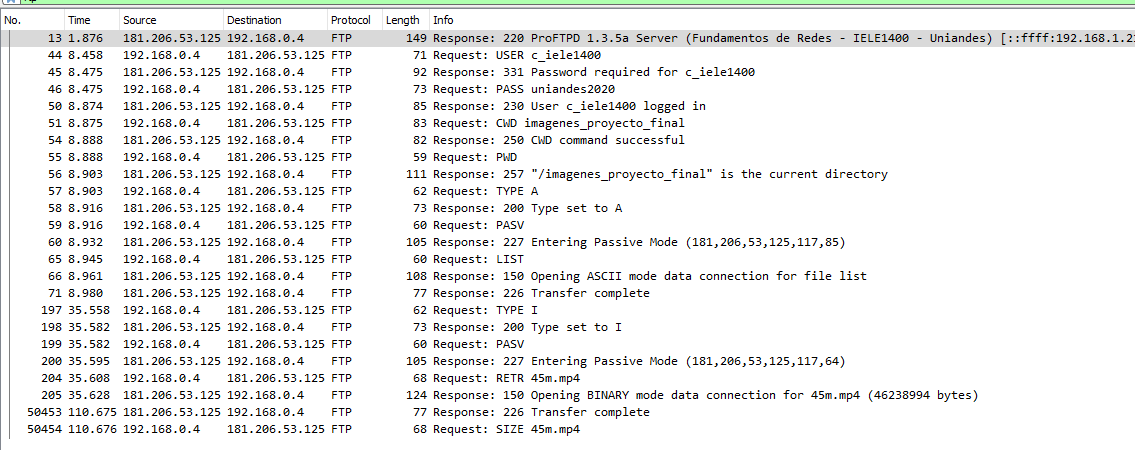
0

1300000

0 50 100 150 200 250 300 350

Por otro lado, nuestro Jitter fue de 0.0034 ms. Para el cliente FTP:

Se realiza el filtro de la figura 4, dando con:



**Figura 15:** Captura de paquetes realizada con el cliente FTP. Realicemos el análisis:

|  |  |  |  |
| --- | --- | --- | --- |
| Tiempo relativo acumulado | Tamaño del paquete | Tiempo de transferencia individual | Tasa instantánea |
| 1.876 | 149 | 0.061594 | 2419.06679 |
| 8.458 | 71 | 6.527392 | 10.8772386 |

|  |  |  |  |
| --- | --- | --- | --- |
| 8.475 | 92 | 0.016601 | 5541.83483 |
| 8.475 | 73 | 0.00008 | 912500 |
| 8.874 | 85 | 0.345049 | 246.341824 |
| 8.875 | 83 | 0.000446 | 186098.655 |
| 8.888 | 82 | 0.000285 | 287719.298 |
| 8.888 | 59 | 0.000054 | 1092592.59 |
| 8.903 | 111 | 0.01511 | 7346.12839 |
| 8.903 | 62 | 0.000166 | 373493.976 |
| 8.916 | 73 | 0.012269 | 5949.95517 |
| 8.916 | 60 | 0.000088 | 681818.182 |
| 8.932 | 105 | 0.015931 | 6590.92336 |
| 8.945 | 60 | 0.01287 | 4662.00466 |
| 8.961 | 108 | 0.016414 | 6579.74899 |
| 8.98 | 77 | 0.018629 | 4133.34049 |
| 35.558 | 62 | 26.5788 | 2.3326862 |
| 35.582 | 73 | 0.023669 | 3084.20297 |
| 35.582 | 60 | 0.000104 | 576923.077 |
| 35.595 | 105 | 0.012781 | 8215.31962 |
| 35.608 | 68 | 0.013045 | 5212.72518 |
| 35.628 | 124 | 0.020251 | 6123.15441 |
| 110.675 | 77 | 74.99426 | 1.02674525 |
| 110.676 | 68 | 0.00288 | 23611.1111 |

Para la tasa de transferencia promedio tenemos lo siguiente:

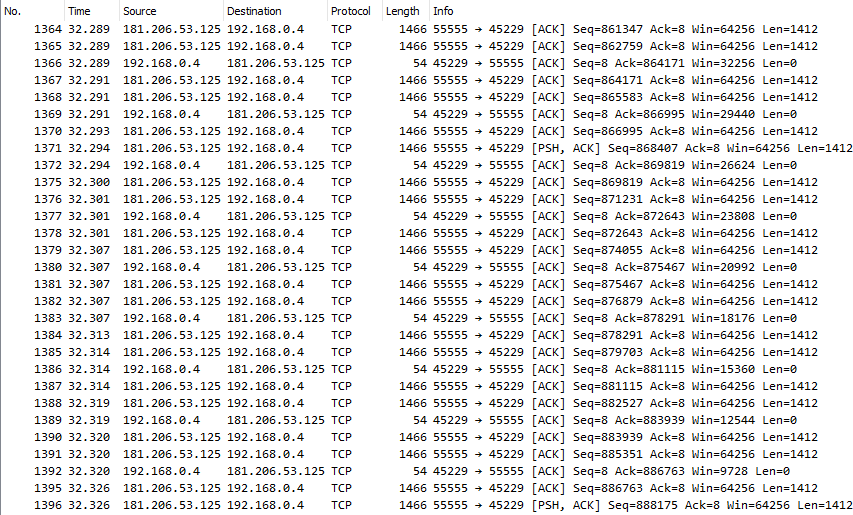
̅𝑇̅̅𝑎̅𝑠̅̅𝑎̅ = 45.546 𝑀𝐵 = 0.41 𝑀𝐵⁄ = 3.21 𝑀𝑏𝑝𝑠 108.688 𝑠

𝑠

45.546 Megabytes corresponden al peso del archivo y 108.688 segundos corresponden a la suma del tiempo de transferencia por paquete.

# Segunda prueba realizada el sábado 29 de may. de 2021 a las 11:00 am

Para el cliente de Sockets de Flujo:



**Figura 16:** Captura de paquetes realizada con el cliente de Sockets de Flujo. #Toma 2

Realizamos la tabla para 60 paquetes aleatorios. En este caso cogeremos paquetes con un peso de 1466 Bytes.

|  |  |  |  |
| --- | --- | --- | --- |
| Tiempo relativo acumulado | Tamaño del paquete | Tiempo de transferencia individual | Tasa instantánea |
| 30.136 | 1466 | 0.000165 | 8884848.48 |
| 30.153 | 1466 | 0.01692 | 86643.026 |
| 30.153 | 1466 | 0.000285 | 5143859.65 |
| 30.155 | 1466 | 0.00157 | 933757.962 |
| 30.157 | 1466 | 0.001541 | 951330.305 |
| 30.158 | 1466 | 0.00115 | 1274782.61 |
| 30.158 | 1466 | 0.000157 | 9337579.62 |
| 30.159 | 1466 | 0.000976 | 1502049.18 |
| 30.16 | 1466 | 0.000571 | 2567425.57 |
| 30.162 | 1466 | 0.002005 | 731172.07 |
| 30.162 | 1466 | 0.002375 | 617263.158 |
| 30.165 | 1466 | 0.000815 | 1798773.01 |
| 30.165 | 1466 | 0.015463 | 94806.9585 |
| 30.181 | 1466 | 0.001422 | 1030942.33 |
| 30.182 | 1466 | 0.001422 | 1030942.33 |
| 30.182 | 1466 | 0.000084 | 17452381 |
| 30.187 | 1466 | 0.004662 | 314457.314 |
| 30.188 | 1466 | 0.000545 | 2689908.26 |

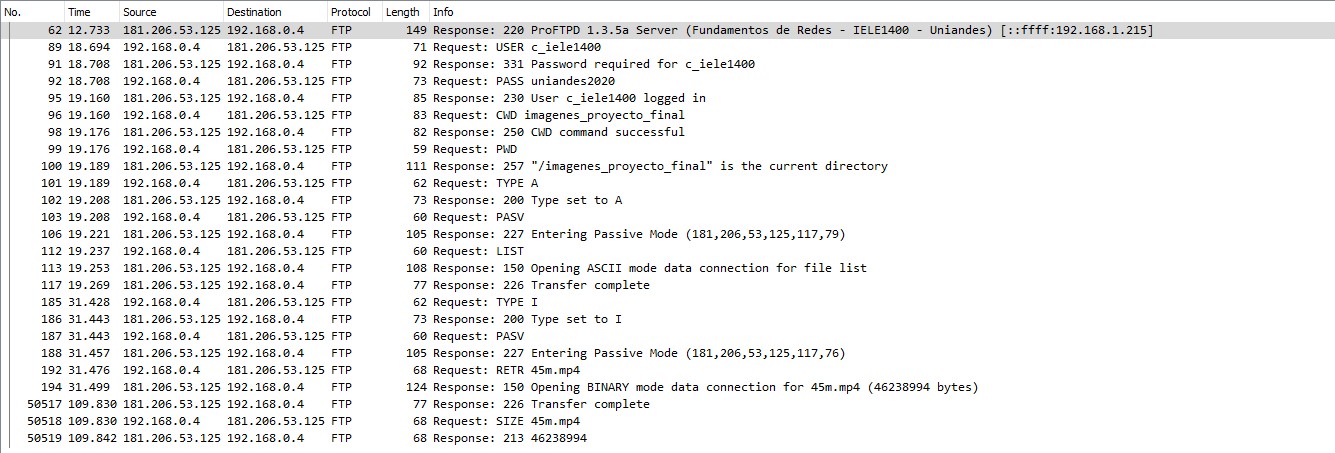
|  |  |  |  |
| --- | --- | --- | --- |
| 30.197 | 1466 | 0.009092 | 161240.651 |
| 30.198 | 1466 | 0.001429 | 1025892.23 |
| 122.269 | 1466 | 0.001412 | 1038243.63 |
| 122.272 | 1466 | 0.00319 | 459561.129 |
| 122.274 | 1466 | 0.001789 | 819452.208 |
| 122.275 | 1466 | 0.001409 | 1040454.22 |
| 122.278 | 1466 | 0.003214 | 456129.434 |
| 122.28 | 1466 | 0.001797 | 815804.118 |
| 122.282 | 1466 | 0.001373 | 1067734.89 |
| 122.285 | 1466 | 0.003203 | 457695.91 |
| 122.287 | 1466 | 0.001824 | 803728.07 |
| 122.288 | 1466 | 0.001403 | 1044903.78 |
| 122.291 | 1466 | 0.003158 | 464217.859 |
| 122.293 | 1466 | 0.001813 | 808604.523 |
| 122.294 | 1466 | 0.001403 | 1044903.78 |
| 122.298 | 1466 | 0.003162 | 463630.614 |
| 122.299 | 1466 | 0.001804 | 812638.581 |
| 122.301 | 1466 | 0.001401 | 1046395.43 |
| 122.304 | 1466 | 0.003209 | 456840.137 |
| 122.306 | 1466 | 0.002127 | 689233.662 |
| 122.306 | 1466 | 0.00394 | 372081.218 |
| 122.31 | 1466 | 0.003925 | 373503.185 |
| 211.997 | 1466 | 0.000406 | 3610837.44 |
| 211.999 | 1466 | 0.002517 | 582439.412 |
| 212.002 | 1466 | 0.003199 | 458268.209 |
| 212.004 | 1466 | 0.001595 | 919122.257 |
| 212.006 | 1466 | 0.001592 | 920854.271 |
| 212.009 | 1466 | 0.003181 | 460861.364 |
| 212.01 | 1466 | 0.00163 | 899386.503 |
| 212.012 | 1466 | 0.00156 | 939743.59 |
| 212.759 | 1466 | 0.0000064 | 229062500 |
| 212.76 | 1466 | 0.000703 | 2085348.51 |
| 212.76 | 1466 | 0.000085 | 17247058.8 |
| 212.762 | 1466 | 0.002362 | 620660.457 |
| 212.764 | 1466 | 0.001514 | 968295.905 |
| 212.764 | 1466 | 0.000066 | 22212121.2 |
| 212.766 | 1466 | 0.001841 | 796306.355 |
| 212.766 | 1466 | 0.000607 | 2415156.51 |
| 212.767 | 1466 | 0.000904 | 1621681.42 |
| 212.767 | 1466 | 0.000019 | 77157894.7 |
| 212.77 | 1466 | 0.002736 | 535818.713 |
| 212.777 | 1466 | 0.006539 | 224193.302 |

Teniendo ya estos datos, podemos resolver para la tasa de transferencia promedio:

𝑇̅̅̅𝑎̅̅𝑠̅𝑎̅ = 49018407 × 8 = 1′842996 𝑏𝑝𝑠 ≈ 1.84𝑀𝑏𝑝𝑠 212.777 𝑠

Con un Jitter de 0.00036 ms, lo que explicaría la tasa tan elevada. La red, en teoría, no tiene congestión.

Para el cliente FTP:



**Figura 17:** Captura de paquetes realizada con el cliente FTP. #Toma 2

̅𝑇̅̅𝑎̅̅𝑠̅𝑎̅ = 45.156

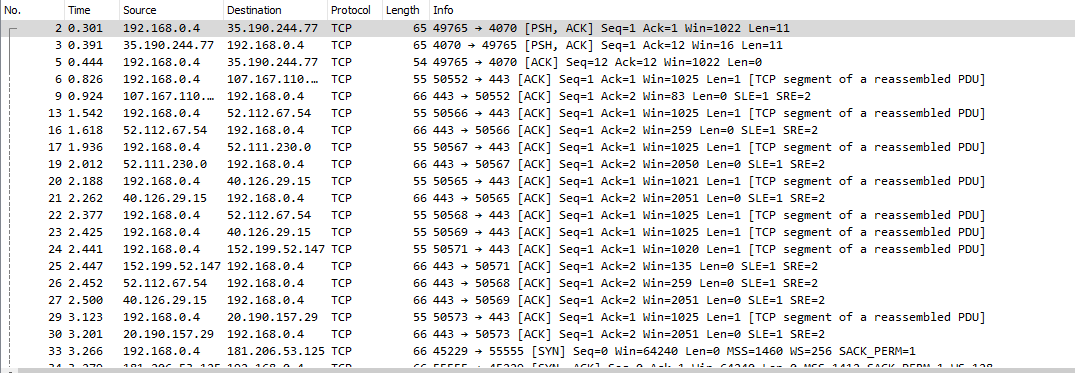
107.325 𝑠

= 0.42 𝑀𝐵⁄𝑠 = 3.36 𝑀𝑏𝑝𝑠

Donde 45.156 es el peso del archivo en MB y 107.325 es el tiempo de transferencia individual acumulado.

# Tercera toma realizada el día sábado 29 de may. de 2021 a las 7:00 pm

Para el cliente con Sockets de Flujo:

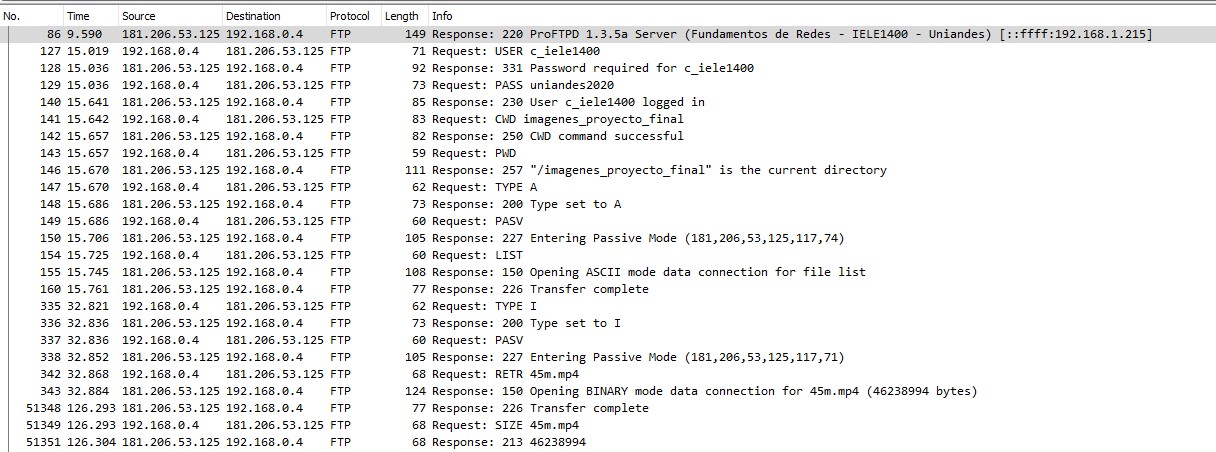


**Figura 18:** Captura de paquetes realizada con el cliente con Sockets de Flujo. #Toma 3

Calculamos la tasa:

𝑇̅̅̅𝑎̅̅𝑠̅𝑎̅ = 49018407 × 8 = 1′700.153 𝑏𝑝𝑠 ≈ 1.7𝑀𝑏𝑝𝑠 230.654 𝑠

El Jitter fue de: 0.00301 ms Para el cliente FTP:



**Figura 19:** Captura de paquetes realizada con el cliente FTP. #Toma 3

̅𝑇̅̅𝑎̅𝑠̅̅𝑎̅ = 45.546 𝑀𝐵 = 0.41 𝑀𝐵⁄ = 3.21 𝑀𝑏𝑝𝑠

𝑠

120.304 𝑠

Respecto al análisis ya hecho, se puede deducir que la tasa de transferencia de datos promedio varía, no solo respecto a la hora y día que se realiza, sino también respecto al tamaño del archivo. Nótese en el archivo de 650 KB, la tasa de transferencia de datos más alta fue para el cliente de sockets con una tasa de 0.822 𝑀𝑏𝑝𝑠, mientras que la tasa más alta para el archivo de 45 MB fue de 3.21 𝑀𝑏𝑝𝑠. Claramente para el archivo de mayor tamaño tiene mayor tasa de transferencia, y si bien sí varía con el día y la hora, la tendencia se mantenía. Además, retomando lo anterior dicho, la fecha y hora también hacen que esta tasa varíe mucho. Debido a que habrá menos gente haciendo pruebas en el servidor, el servidor tendrá menos congestión, por lo tanto, una mayor tasa de transferencia de datos.

1. Determinar la cantidad de fragmentos en que se dividen los archivos descargados utilizando cálculos manuales y comparar con los resultados que entrega Wireshark utilizando sockets de flujo y FTP. Hacer una tabla comparativa donde se evidencien las tendencias de las mediciones y el cálculo realizado.

Tamaño del MTU = 1500 Bytes aproximadamente,

Para el archivo de 45MB

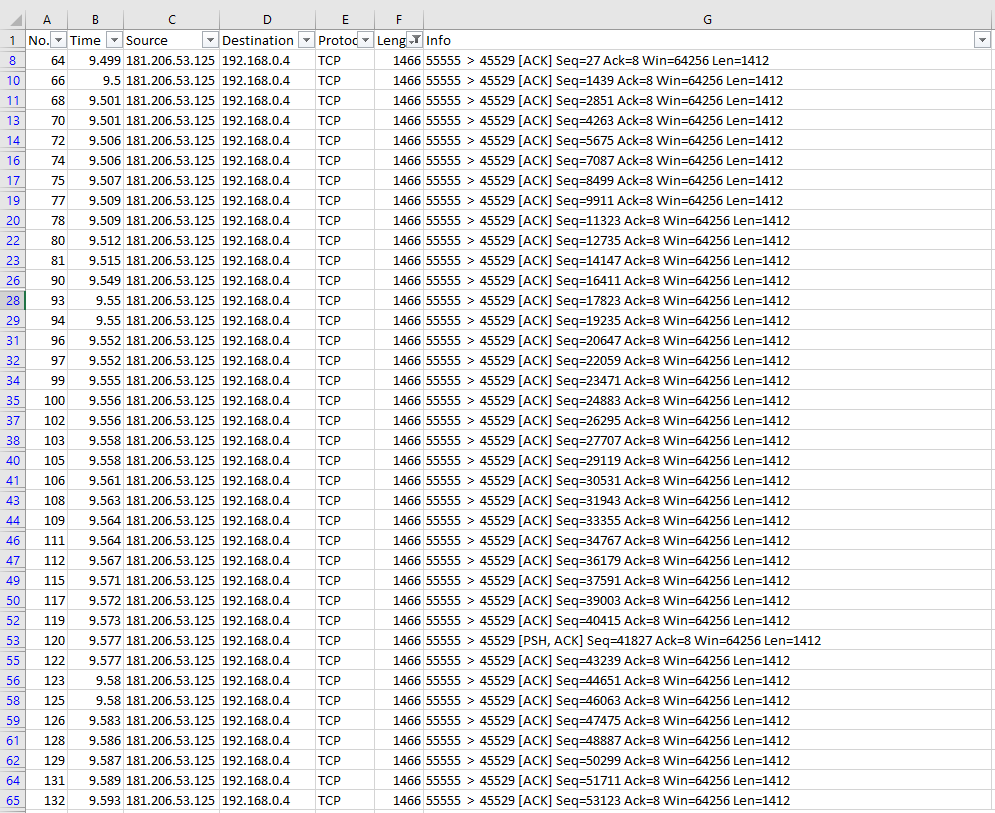
45 𝑀𝐵 → 45′185.920 𝐵𝑦𝑡𝑒𝑠 𝑝𝑜𝑟 𝑙𝑜 𝑡𝑎𝑛𝑡𝑜,

451895920

1500

= 30123.94 𝑝𝑎𝑞𝑢𝑒𝑡𝑒𝑠 (𝑉𝑎𝑙𝑜𝑟 𝑡𝑒ó𝑟𝑖𝑐𝑜)

Para saber la cantidad de paquetes que hubo por cliente, bajo una captura de paquetes, se exportará a Excel y se hará un filtro sobre los paquetes con un peso de 1466 Bytes, como podrá ver en la siguiente imagen:



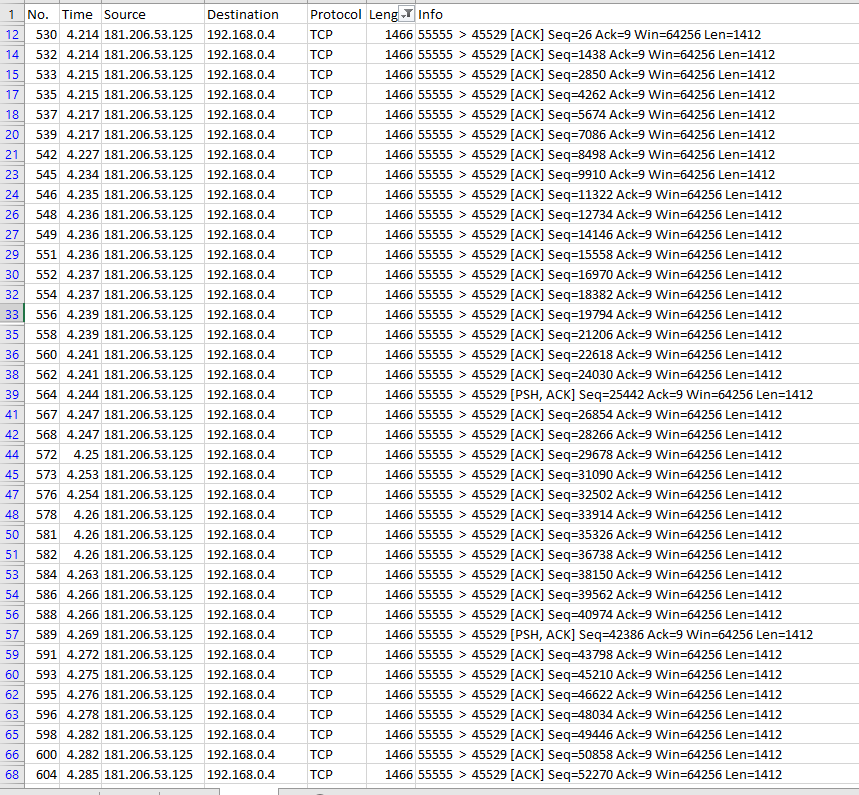
**Figura 20:** Filtrado de paquetes. Cliente con Sockets de Flujo para el archivo de 45 MB

Al ver el recuento de filas, damos con un total de paquetes de 32584. Para el archivo de 650KB

45 𝑀𝐵 → 665600 𝐵𝑦𝑡𝑒𝑠 𝑝𝑜𝑟 𝑙𝑜 𝑡𝑎𝑛𝑡𝑜,

665600

1500 = 443.73 𝑝𝑎𝑞𝑢𝑒𝑡𝑒𝑠 (𝑉𝑎𝑙𝑜𝑟 𝑡𝑒ó𝑟𝑖𝑐𝑜)



**Figura 21:** Filtrado de paquetes. Cliente con Sockets de Flujo para el archivo de 650 KB

Al ver el recuento de filas, damos con un total de paquetes de 467.

|  |  |  |
| --- | --- | --- |
|  | Valor teórico | Valor experimental |
| 650 KB | 443.73 | 467 |
| 45 MB | 30123.94 | 32584 |

Es claramente evidente que el archivo más grande necesita fragmentar en muchos más paquetes para hacer la transferencia del archivo. Además, podemos ver que el valor teórico siempre es menor que el valor experimental, y puede ser por distintas razones tales como:

* 1. Congestión de la red.
  2. Se tomaron paquetes irrelevantes como encabezados.

Bibliografía

1. Á. L. Cobo, «Protocolo de Transferencia de Archivos (FTP),» Pueblo Nuevo, 2009.
2. R. González Duque, Python para todos, 2014.
3. "¿Qué es Jitter?", *Speedcheck.org*. [Online]. Available: https://[www.speedcheck.org/es/wiki/jitter/.](http://www.speedcheck.org/es/wiki/jitter/) [Accessed: 22- May- 2021].
4. "VoIP Foro - QoS - jitter - Causas, soluciones y valores recomendados", *Voipforo.com*. [Online]. Available: [http://www.voipforo.com/QoS/QoS\_Jitter.php.](http://www.voipforo.com/QoS/QoS_Jitter.php) [Accessed: 22- May- 2021].

[5]s. bind(), S. Ghosh, J. Rach, S. Ghosh and P. Murry, "socket connect() vs bind()", *Stack Overflow*, 2021. [Online]. Available: https://stackoverflow.com/questions/27014955/socket-connect- vs-bind. [Accessed: 22- May- 2021].