**UNIVERSIDAD NACIONAL DE CÓRDOBA**

**FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES.**

**ASIGNATURA**

Comunicaciones de Datos

**CARRERA**

INGENIERIA EN COMPUTACION

**Trabajo Práctico III: Capa​ ​ de​ ​ Transporte. Comunicación​ ​ TCP​ ​ y ​ ​ UDP​ ​ con​ ​ sockets - Análisis​ ​ de​ ​ una​ ​ transferencia​ ​ TCP​ ​ con​ ​ FTP**

**Integrantes:**

* Alaniz Ferreyra, Alvaro
* Malano, Leandro

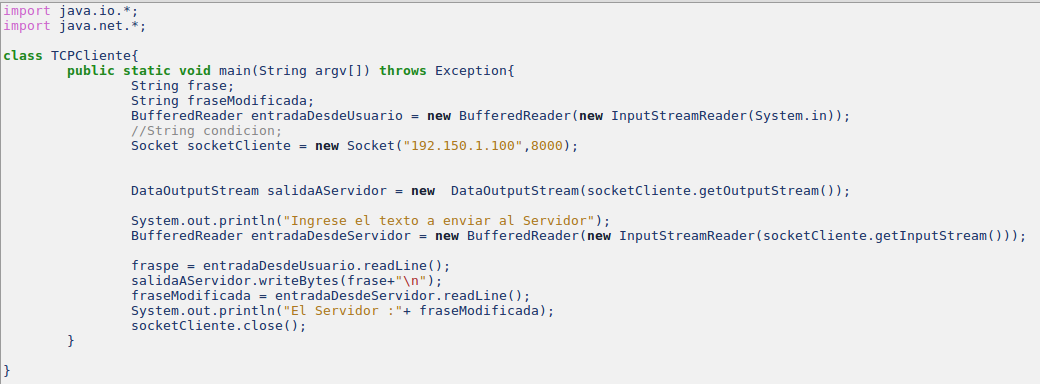
**Parte 1:**

**Cliente y Servidor TCP:**

Implementando un socket Cliente y un socket Servidor en dos terminales utilizando Java, logramos comunicarnos utilizando un puerto determinado en ambas terminales, en el cual para ver si esto funciona logramos enviar una cadena de caracteres (String) por consola del socket Cliente para que el servidor muestre este mensaje.

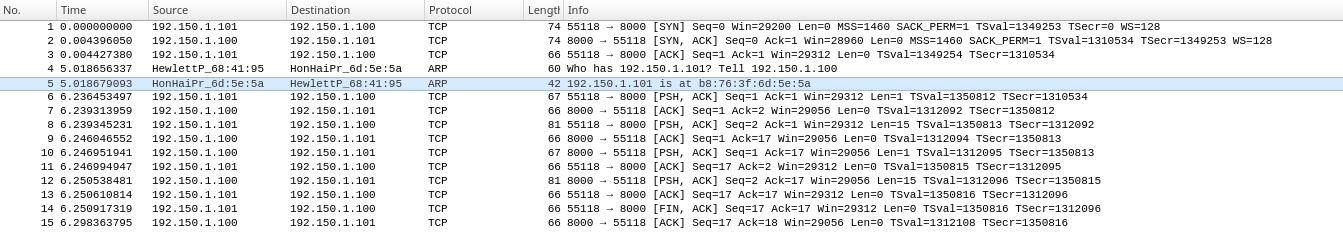
Para el primer caso utilizamos el protocolo de la capa de transporte TCP, en el segundo caso utilizamos UDP.

A continuacion mostraremos los codigos de los archivos TCPCliente.java y TCPServidor.java , ambos codigos fueron ejecutado en diferente máquinas, mediante la JVM (Java Virtual Machine).Primeros ejecutaremos el server que “atenderá” el cliente.





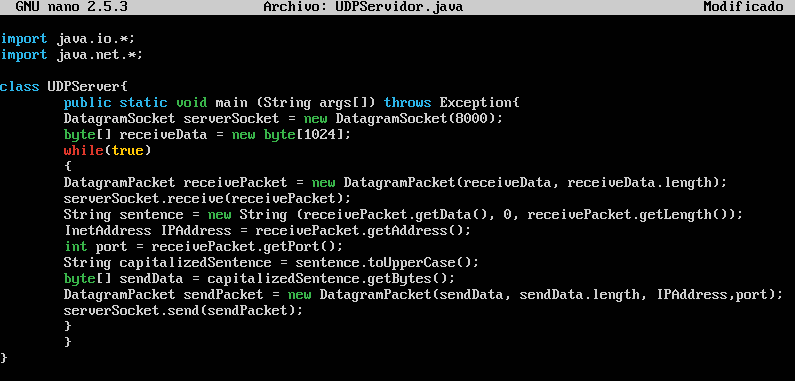
Ahora escucharemos utilizando wireshark los paquetes TCP para ver si es correcta esta implementación utilizada según la teoría:



Vemos que al comienzo realiza la conexión Cliente-Servidor, ya que este protocolo está orientado a la conexión, ahora haremos un paréntesis para explicar ARP ya que permite que se conozca la dirección física de una tarjeta de interfaz de red correspondiente a una dirección IP. Luego vemos que desde el paquete 6 hasta el 13 envía los datos a la capa superior, devolviendo un ACK en caso de que ese envío fue exitoso y por último en las líneas 14 y 15, establecemos el fin de la conexión, ya habiendo cumpliendo nuestro objetivo de manera correcta.

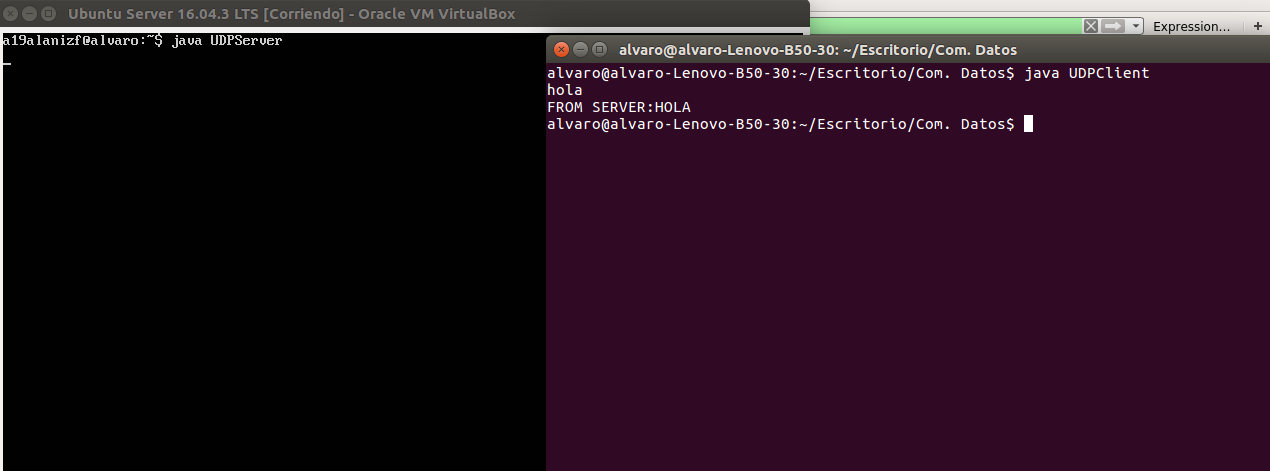
**Cliente y Servidor UDP:**

Primero, se procedió a realizar los códigos en lenguaje java del Cliente y del Servidor para UDP. Se detallan a continuación:

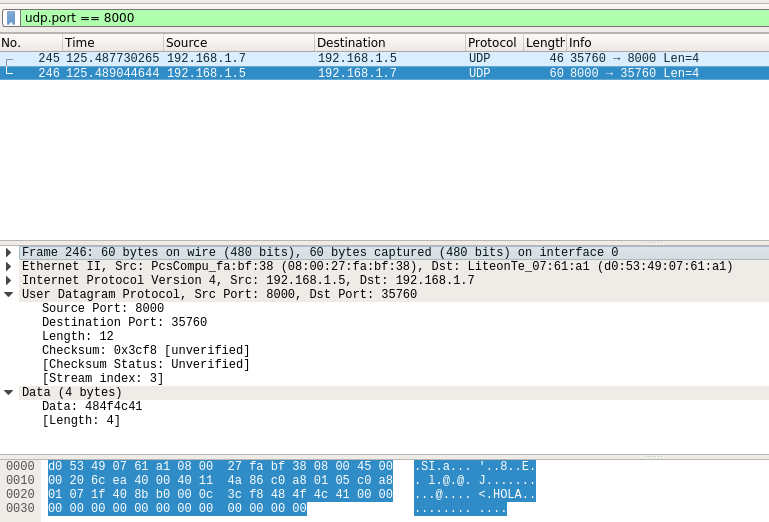


Como en el primer ejercicio con TCP, se ejecutó el código UDPCliente.java en el sistema operativo que acoge a la máquina virtual. El código UDPServer.java se ejecutó en la maquina virtual en Ubuntu Server 16.04.3 LTS.

Ya ejecutado, se envía un mensaje desde el cliente al servidor. Este lo recibe y devuelve el eco del primer mensaje pero en mayúsculas. En la siguiente imagen se ve dicha interacción en el terminal de Ubuntu.



A partir de esta interacción, se puede analizar mediante la herramienta Wireshark la conexión entre el servidor y el cliente, obteniendo lo siguiente:

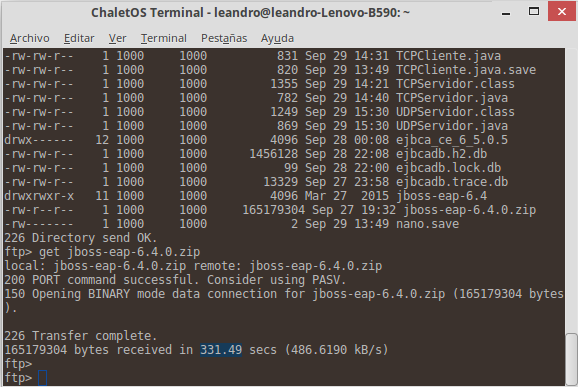


Se puede observar que se envían paquetes desde cliente a servidor y viceversa de 4 bytes desde el puerto 35760 al 8000 que es el caso de la solicitud del cliente al servidor. Mientras que en el segmento de data en el programa se ve el mensaje “HOLA” que es el que devuelve el servidor al cliente.

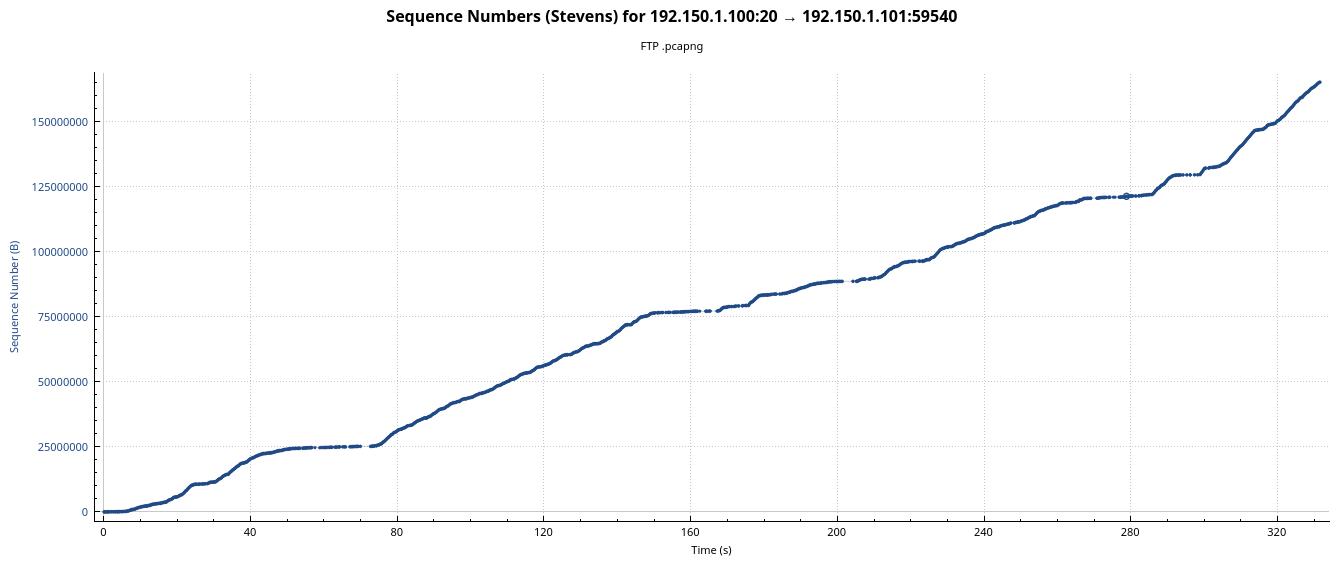
A su vez, se puede observar que UDP no requiere establecer ninguna sincronización previa antes de realizar el intercambio de datos. Esto significa que dicho protocolo no es orientado a conexión.

**Parte 2:**

Utilizando un Server FTP, desde una pc, accederemos desde otra pc remotamente para empezar a analizar este caso. Luego de loguearnos, ingresamos al sistema de archivo del host remoto y copiaremos nuestro archivo en cuestión utilizando el comando FTP get, como se muestra en la imagen siguiente.



* **Gráfico Time/Sequences​ ​ Stevens:**



Vemos que a medida que avanza el tiempo, el número de secuencia aumenta, por lo que significa que aumenta la cantidad de datos que se envía, a excepción de 4 cortes visiblemente en la gráfica, también podemos ver en una parte de la gráfica que el número de secuencia se “estanca” pero luego crece.

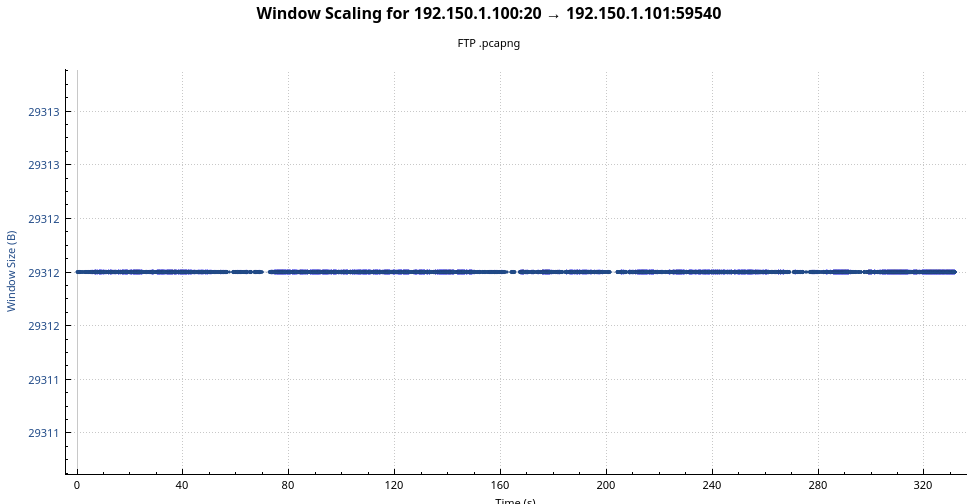
Luego, se pueden ver unas mesetas en el gráfico anterior. Al terminar dichas mesetas se da un crecimiento exponencial del número de secuencias hasta cierto punto donde vuelve a ser un crecimiento lineal. Este crecimiento exponencial está dado por el algoritmo de TCP Slow-start.

El algoritmo *Slow-start* tiene la función de equilibrar la velocidad de una conexión de red. Éste incrementa gradualmente la cantidad de datos transmitidos hasta que encuentra la capacidad máxima de carga de la red.

El inicio lento impide que la red se congestione regulando la cantidad de datos que se envía por ella. Se produce una “negociación” entre emisor y receptor al definir la cantidad de datos que se pueden transmitir con cada paquete, aumentando lentamente la cantidad de datos hasta que se alcanza la capacidad de la red (se produce la primera retransmisión por time-out). A partir de ese punto, se produce un crecimiento lineal en el envío de datos. Garantiza la transmisión de datos sin obstruir la red.

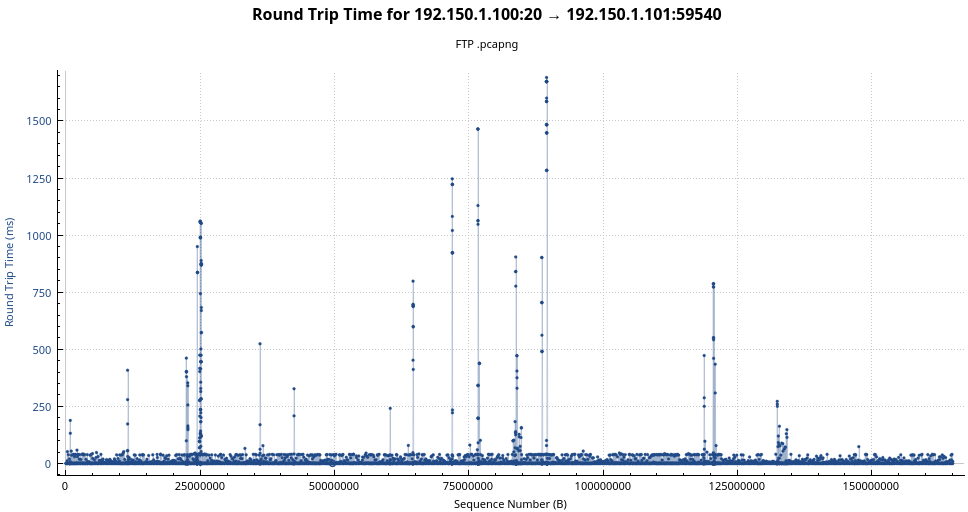
A partir de esta gráfica determinaremos la velocidad de transmisión de datos (kB/s), el tiempo de envío fue de 331 seg aproximadamente y los bytes transmitidos fueron 165.000.200 .Esto nos determina una velocidad de 499.093,65 KB/s, datos coherentes si comparamos con los obtenidos de la imagen anterior de la terminal.

* **Gráfico ​ Window Scaling:**



Podemos ver que el tamaño de la ventana estuvo siempre constante a simple vista, a excepción de los cortes que la gráfica presenta que coinciden en tiempo con el del grafico de Stevens, momento en que no se envia datos del server al cliente, la ventana es igual a cero.

* **Round​ ​ Trip​ ​ Time:**



Observamos en este gráfico que el tiempo de llegada del envío de paquetes es casi constante en excepciones en donde hay sobresaltos. el orden del tiempo RTT es del orden del 0,1 mS aproximadamente.

**Preguntas:**

*1.​ ​ ¿En​ qué​ casos​ conviene​ utilizar​ UDP​ por​ más​ que​ no​ implemente​ control​ de​ flujo? Nombre​ 3 ​protocolos​ ​de​ la​ ​capa​ ​de​​ aplicación​ que​ funcionan​ con​ UDP ​y ​por​ ​qué.*

UDP provoca menor sobrecarga en la red, debido a que no necesita inicializar ni dar cierre a las conexiones. En algunas aplicaciones esto es preferible (no disminuir la tasa de paquetes) a que probablemente se puedan perder paquetes, una consecuencia directa en este tipo de protocolo.

Además, dicho protocolo ayuda a tener un mayor control a nivel de capa de aplicación, ya que permite enviar paquetes cuando el usuario lo decide y no está a la espera que se den ciertas condiciones de inicio de conexión y de cierre como es el caso de TCP.

Su uso principal es para protocolos como DHCP, BOOTP, DNS, entre otros. En estos el intercambio de paquetes son mayores en la conexion y desconexion. No es rentable la pérdida de performance con respecto a la información transmitida, aun cuando se puedan perder paquetes de transmisión. Un ejemplo de esto, es la transmisión de audio y video en tiempo real, donde no es tan grave que se pierda un pixel de la pantalla, o se pierda algún dato de la transmisión de la voz. Es decir, se usa el protocolo UDP cuando necesitamos una mayor velocidad de transferencia de datos que, en si el mensaje que se transmite.

Funcionan con UDP:

DNS: debido a que no es tan crítico que no responda un servidor DNS al envío, porque puede recibir una respuesta de otro servidor. La consulta por otra parte debe ser lo mas rapida posible, que se logra con UDP.

VoIP: preferible una pérdida de paquetes de datos a que baje la velocidad de transmisión. Si no se recibió un paquete en un determinado momento, su reenvío no es necesario.

IPtv: similar a VoIP, más importante una robustez en la velocidad de transmisión que la pérdida de paquetes de datos.

*2.​ ​ En​ ​ la​ ​ cabecera​ ​ de​ ​ IPv4​ ​ hay​ ​ un​ ​ campo​ ​ llamado​ ​ “protocolo”,​ ​ que​ ​ indica​ ​ el​ ​ protocolo que​ ​ está​ ​ transportando​ ​ el​ ​ paquete​ ​ IP,​ ​ por​ ​ ejemplo​ ​ UDP​ ​ o ​ ​ TCP.​ ​ ¿Por​ ​ qué​ ​ no​ ​ hace​ ​ falta un​ ​ campo​ ​ parecido​ ​ en​ ​ la​ ​ cabecera​ ​ UDP/TCP​ ​ que​ ​ indica​ ​ algo​ ​ como​ ​ HTTP,​ ​ DNS...?*

Tanto HTTP como DNS y otros más protocolos de la capa de aplicación tienen asignados puertos estandarizados y conocidos, por los cuales se los puede identificar. Estos van del 0 al 1023 y fueron asignados por IANA. Por ejemplo, se sabe que el puerto 80 es usado por HTTP, 20 y 21 por FTP o el 53 por DNS.

*3.​ ​ ¿Por​ ​ qué​ ​ FTP​ ​ utiliza​ 2 ​puertos​ ​ en​ ​ vez​ ​ de​ ​ uno​ ​ solo?*

FTP usa un puerto para realizar la conexión de control, en la cual se realiza el login y se envían los comandos para modificar el directorio del servidor. Esta conexión se mantiene activa a lo largo de toda la sesión del usuario para poder tener un estado del usuario. En el puerto destinado al flujo de paquetes, se intercambian todos los datos dependiendo del comando ingresado en la conexión de control.

El protocolo envía exactamente un archivo por cada conexión de datos y luego se cierra. Si el usuario desea transmitir otro archivo, se debe crear una nueva conexión de datos. Por lo tanto, como ambas conexiones funcionan de manera totalmente diferente, se emplean dos puertos. FTP usa la conexión de control en el puerto 21, diferente al puerto 20, destinado al flujo de datos.