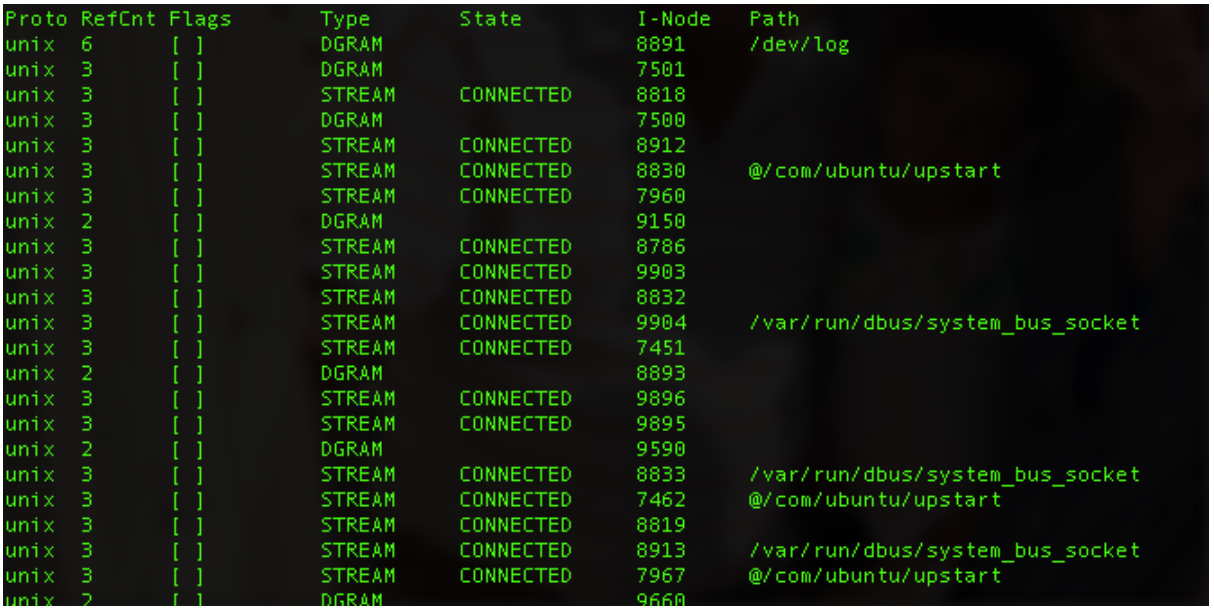
LABORATORIO 5 – TRANSMISIÓN DE DATOS SOBRE UDP

Camilo Montenegro - 201531747
Carlos Peñaloza - 201531973

4.4 Intercambios de archivos en UDP

Se realizó un programa cliente y servidor para realizar transferencia de objetos por UDP entre un cliente y un servidor. Las pruebas en wireshark y las estadísticas del servidor permitieron ver que se pierden aproximadamente la mitad de los archivos.
Cuando se realizó la prueba con múltiples clientes, todas las máquinas tenían la misma ip local pues no se implementó un mecanismo para cambiarlas.
Luego se realizó una aplicación para transferir archivos con udp. Se pudo ver que no es posible mandar archivos por medio de udp sin implementar un mecanismo para envío confiable pues se pierden muchos paquetes impidiendo que el archivo llegue bien. Por el anterior motivo el hash nunca fue igual a aquel calculado antes de enviar.

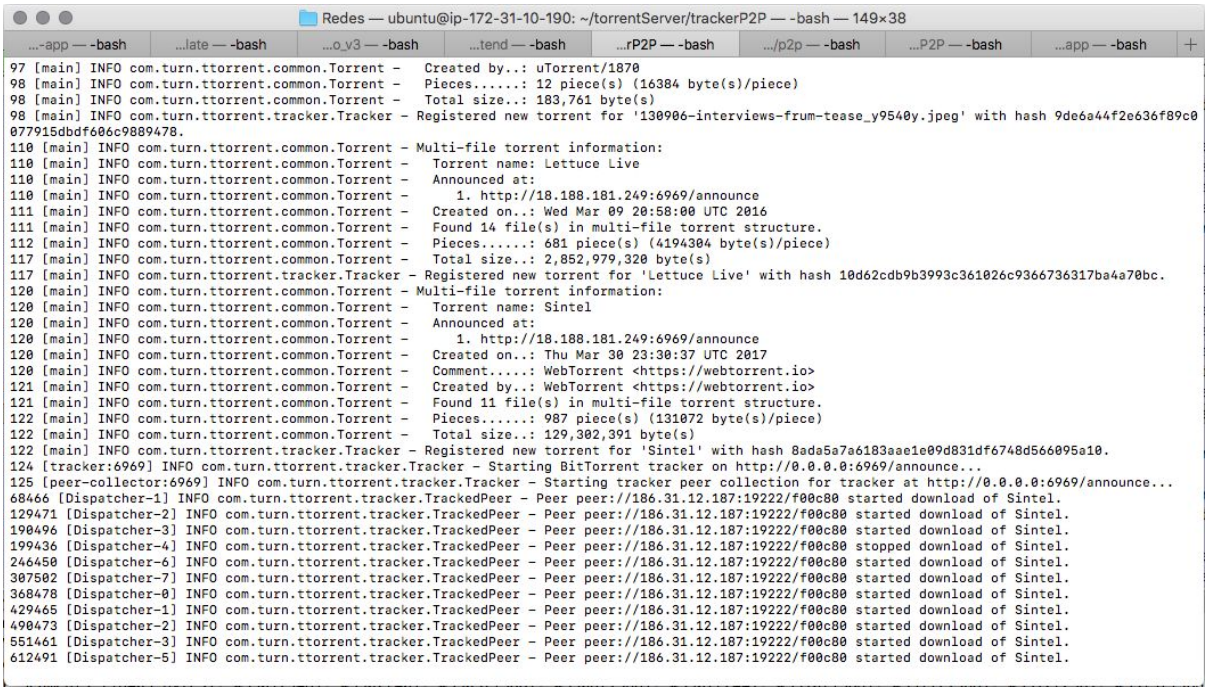


```
Proto RefCnt Flags       Type       State         I-Node   Path
unix  6      [ ]         DGRAM                    8891     /dev/log
unix  3      [ ]         DGRAM                    7501
unix  3      [ ]         STREAM     CONNECTED     8818
unix  3      [ ]         DGRAM                    7500
unix  3      [ ]         STREAM     CONNECTED     8912
unix  3      [ ]         STREAM     CONNECTED     8830     @/com/ubuntu/upstart
unix  3      [ ]         STREAM     CONNECTED     7960
unix  2      [ ]         DGRAM                    9150
unix  3      [ ]         STREAM     CONNECTED     8786
unix  3      [ ]         STREAM     CONNECTED     9903
unix  3      [ ]         STREAM     CONNECTED     8832
unix  3      [ ]         STREAM     CONNECTED     9904     /var/run/dbus/system_bus_socket
unix  3      [ ]         STREAM     CONNECTED     7451
unix  2      [ ]         DGRAM                    8893
unix  3      [ ]         STREAM     CONNECTED     9896
unix  3      [ ]         STREAM     CONNECTED     9895
unix  2      [ ]         DGRAM                    9590
unix  3      [ ]         STREAM     CONNECTED     8833     /var/run/dbus/system_bus_socket
unix  3      [ ]         STREAM     CONNECTED     7462     @/com/ubuntu/upstart
unix  3      [ ]         STREAM     CONNECTED     8819
unix  3      [ ]         STREAM     CONNECTED     8913     /var/run/dbus/system_bus_socket
unix  3      [ ]         STREAM     CONNECTED     7967     @/com/ubuntu/upstart
unix  2      [ ]         DGRAM                    9660
```

Nombre/Tamaño Fichero	Recibido	Archivo Wireshark
dosmegas	No	Test1000.pcapng
sietemegas	No	Test10000.pcapng
75megas	No	Test100000.pcapng

4.5 Intercambios de archivos en Bittorrent

Para realizar las pruebas de intercambio usando Bittorrent, se realizó una aplicación cliente y otra aplicación “servidor” que es mejor conocida como Tracker. Tanto el cliente como el tracker fueron realizados usando la librería Ttorrent. Dicho cliente maneja tres métodos esenciales para cumplir los requerimientos de ser cliente o peer: cargar el archivo torrent a compartir, descargar el archivo y compartir el archivo, es decir que cumple el rol de semilla. En cuanto al tracker, cumple la función esencial de anunciar cada archivo .torrent e iniciar procesos en el puerto 6969 y exponiendo el servicio /announce. El tracker se encuentra desplegado en una máquina virtual de aws bajo una ip pública dinámica.
En cuanto a las pruebas, se utilizó una carpeta .torrent la cual contiene archivos de menos de 2MiB, de aproximadamente 5MiB y más de 100MiB. Dicho archivo en sus propiedades tiene registrado como tracker el realizado para este laboratorio. Es decir que al momento de iniciar la descarga de archivos usando cualquier cliente torrent, este se debe redirigir al tracker creado con el fin de obtener los diferentes peers en donde se encuentra el archivo a descargar. En la captura de pantalla anexa, se muestra cuando el cliente se conecta e inicia la descarga de archivos. Debido a que el cliente, en este caso, está renovando la concesión con el tracker se generan varios dispatchers para el mismo proceso. Para los casos de prueba, se pudo realizar la descarga de los archivos en donde el de 2MiB tomaba mucho menos tiempo que en los archivos de 5MiB y 100MiB. En todos los casos, el tiempo previo a la descarga es alto debido a que debe primero contactar al tracker y luego hacer handshake con distintos peers, si es el caso, y luego sí proceder a hacer la descarga de los diferentes paquetes. Los resultados y captura de paquetes se encuentran en el archivo de Wireshark.
En la captura de Wireshark se puede evidenciar que inicialmente debe crear un canal confiable con el tracker mediante el protocolo TCP, luego de realizar el handshake, el cliente hace una petición Get al servicio expuesto /announce para que el cliente pueda utilizar tanto el protocolo TCP como BitTorrent para el intercambio de información y descarga de archivos de los diferentes peers. De igual forma, en la captura de Wireshark se puede evidenciar que el archivo llega al cliente por paquetes, se recibe el paquete y notifica al origen que se recibió esa parte del archivo. Asimismo, en Wireshark es posible ver el tamaño del paquete que se envió, dicho valor es variable en cada conexión. Si el cliente ya cuenta con el paquete recibido, le responde de igual forma al origen con ACK pero no procesa el paquete. Este proceso se repite por cada peer disponible para la descarga de archivos. Finalmente, cuando ya se tiene el archivo completo, procesado y en orden, se puede revisar la cantidad de paquetes recibidos y el cliente inicia el proceso de seeding.
La implementación en Java se encuentra anexa como un archivo comprimido.



4.6 Intercambios de archivos en TCP

Resultados de pruebas

Nombre/Tamaño Fichero	Tiempo Prom. [ms]	Recibido	Archivo Wireshark
Fichero1MiB.JPG	767.328982	Si	Captura_1Mib.pcapng
Fichero9MiB.mov	2510.871143	Si	Captura_9Mib.pcapng
Fichero88MiB.pcapng	2660.7107226	Si	Captura_88Mib.pcapng

En la etapa de transferencia de archivos en TCP se realizó una aplicación cliente servidor en Java. Inicialmente, el servidor se encuentra en una máquina virtual AWS corriendo en el puerto 3333 por default pero es posible cambiarlo. Dicha máquina no está restringida a una única conexión con un cliente, puede tener varias conexiones activas al mismo tiempo sin incurrir en problemas. Luego de realizar el handshake inicial con el cliente, el servidor envía los archivos disponibles de descarga al cliente el cual retorna el mensaje con una petición de descarga del archivo seleccionado. Posteriormente, el servidor inicia el envío del archivo mediante un socket único el cual se mantiene activo indefinidamente.
Por el otro lado se encuentra el cliente, éste se encuentra realizado en Java y permite ser usado con una interfaz gráfica. La interfaz únicamente se inicia si se pudo establecer un socket con el servidor. Asimismo, la interfaz al momento del deploy muestra una lista con los archivos disponibles para descargar, cuando es seleccionado el archivo se hace click en

download e inicia la descarga del fichero en donde se muestra una barra de carga y posteriormente un label el cual indica si la descarga fue exitosa o si tuvo algún error. Finalmente, la interfaz muestra una lista con los archivos disponibles en la carpeta destino la cual se actualiza cada vez que llega un nuevo archivo desde el servidor.
En cuanto a las capturas de Wireshark y filtrando en el puerto tcp 3333, se muestra el proceso de conexión TCP, desde la etapa de handshake con el servidor hasta la traza de cada envío de paquetes. Dependiendo del archivo del fichero hay un menor o mayor envío de paquetes desde el servidor, aparentemente por secuencia. En cuanto a la longitud de cada paquete, es la misma o tiende a variar muy poco en cada captura realizada en Wireshark.

Repositorio con el código fuente de la aplicación cliente servidor bajo TCP, las instrucciones de deploy se encuentran en el README.md: https://github.com/ca-montenegro/TcpTransferapp

6. Comparación BitTorrent, TCP, UDP

Dado que UDP es un protocolo no orientado a conexión y el cual el tiempo es un factor relevante en su uso, el tiempo de transmisión de archivos usando UDP es considerablemente menor que usando TCP. Lo anterior también se cumple dado que en comparación con TCP, UDP no realiza verificación de errores con el fin de recuperar o corregir los mismo, es un protocolo del mejor esfuerzo más no de entrega confiable y no hace control de flujo. Por el contrario y dado que TCP cumple más funcionalidades que UDP, es un protocolo que toma más tiempo en la transmisión de archivos pero se garantiza que el archivo llega al destino completo, en orden y no dañado. Dado que BitTorrent usa como protocolo en la capa de transporte a TCP, es de igual forma un protocolo que toma más tiempo al enviar y recibir archivos.

¿Es posible desarrollar aplicaciones UDP que garanticen la entrega confiable de archivos? Qué consideraciones deben tenerse en cuenta para garantizar un servicio de entrega confiable utilizando dicho protocolo.

Si es posible realizar aplicaciones UDP que garanticen la entrega confiable de archivos. El protocolo se encarga de sus funciones básicas (Mejor esfuerzo de envío) la aplicación debe encargarse de implementar los requerimientos adicionales, tales como garantizar el envío confiable. Una implementación que daría solución al requerimiento adicional puede ser que en el servidor al enviar un archivo, cada segmento mantenga en su data la cantidad de paquetes a enviar en total. En el cliente se debería implementar una función que verifique dicho valor, es decir que mantenga un conteo de paquetes recibidos y los compare con el valor enviado inicialmente por el servidor en cada paquete. Si es el caso que no se recibe todos los paquetes, el cliente debe solicitar al servidor que vuelva a enviar los paquetes. De esta forma se cumple con el requisito de entrega confiable, más no es susceptible al tiempo.