Reporte de laboratorio: Envío de archivos por el protocolo UDP

Integrantes:
-Julian Mora (202012747)
-Juan Carlos Eduardo Nunes Ariza (202010826)
- Leandro Esteban Yara Ramírez (202013928)

Enlace de GitHub al repositorio:

https://github.com/LeandroYara/ServidorUDP_Grupo3

Enlace al video explicativo:

https://uniandes-my.sharepoint.com/:v:/g/personal/j_morav_uniandes_edu_co/EVixkL8r7tJMuRy_eVuw41IB83y2lgXNorKTV76PJkp9hg?e=KyHUkr

Enlace a la carpeta de las capturas de tráfico:
Capturas de tráfico

Proceso de solución

Servidor: Para este se crea un archivo en el que se define inicialmente la IP del servidor, el puerto en el que va a escuchar su socket TCP y el tamaño en Bytes de los paquetes enviados. Al inicio del método main se define el valor inicial de los clientes, se le pide al usuario el archivo por enviar y la cantidad de clientes concurrentes que manejara la aplicación y se crea una barrera para controlar la concurrencia. Luego se obtiene el nombre y el tamaño del archivo con el fin de enviarlos a los usuarios.

```
9   IPS = "127.0.0.1"
10  PORTS = 5005
11  ADDRS = (IPS, PORTS)
12  SIZE = 1024
13
14  def main():
15
16      cuentaCliente = 1
17
18      tipo = input("Ingrese 1 si quiere enviar el archivo de 10MB o ingrese 2 si quiere enviar el archivo de 5MB: ")
19      clientesSimultaneos = int(input("Ingrese el numero de usuarios concurrentes que quiere aceptar: "))
20
21      barrera = threading.Barrier(clientesSimultaneos)
22
23      if tipo == "1":
24          filePath = "ArchivosEnvio/100MB.bin"
25      elif tipo == "2":
26          filePath = "ArchivosEnvio/250MB.bin"
27
28      fileName = os.path.basename(filePath)
29      fileSize = os.path.getsize("ArchivosEnvio/" + fileName)
30
```

Para recibir las comunicaciones desde los clientes se utiliza un socket TCP que, al aceptar una petición, envía al cliente su número establecido por orden de llegada, recibe el número del puerto que utiliza el cliente y le envía el número de personas que se espera atender concurrentemente. Ya con esos datos se crea un thread sobre una función encargada de

atender a los clientes a través de sus sockets UDP, se aumenta el número de clientes y se inicializa el thread.

```
    sockTCP = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sockTCP.bind(ADDRS)
    sockTCP.listen()
    print(f"El servidor esta escuchando en ({IPS}, {PORTS})...")

    while True:
        conn, addr = sockTCP.accept()
        conn.send(cuentaCliente.to_bytes(2, 'little'))
        puertoCod = conn.recv(1024)
        puertoNuevo = int.from_bytes(puertoCod, 'little')
        conn.send(clientesSimultaneos.to_bytes(2, 'little'))
        thread = threading.Thread(target=handle_client, args=(addr, barrera, fileName, fileSize, cuentaCliente, puertoNuevo, clientesSimultaneos, IPS, PORTS))
        cuentaCliente += 1
        thread.start()
        print(f"[CONEXIONES ACTIVAS]: {threading.active_count() - 1}")

```

Dentro de cada thread, cuando todas las conexiones esperadas hayan llegado, se abre la barrera, se crea un socket UDP que envía el nombre y tamaño del archivo al socket UDP del cliente y se abre el archivo por enviar para establecer el primer bloque que se va a enviar junto a la barra de progreso de envío y el contador de tiempo inicial.

```

def handle_client(addr, barrera, fileName, fileSize, cuentaCliente, puertoNuevo, clientesSimultaneos, IPS, PORTS):

    barrera.wait()
    sockUDP = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sockUDP.bind((IPS, PORTS + cuentaCliente))

    sockUDP.sendto(fileName.encode(), (addr[0], puertoNuevo))
    sockUDP.sendto(fileSize.to_bytes(3, 'little'), (addr[0], puertoNuevo))

    print(f"Enviando {fileName} al cliente {cuentaCliente}...")
    file = open("ArchivosEnvio/" + fileName, "rb")
    data = file.read(SIZE)

    progress = tqdm.tqdm(range(fileSize), f"Enviando {fileName}", unit="B", unit_scale=True, unit_divisor=1024)
    now = datetime.datetime.now()
    inicio = time.time()

```

Para enviar el archivo, va leyendo si el siguiente bloque no está vacío, lo envía al cliente y actualiza el progreso del envío. Cuando no hay más datos por leer se cierra el archivo y se obtiene el contador de tiempo final con el que calcula el tiempo de envío.

```


    while(data):
        if(sockUDP.sendto(data, (addr[0], puertoNuevo))):
            data = file.read(SIZE)
            progress.update(len(data))
    file.close()
    print(f"{fileName} enviado al cliente {cuentaCliente}!")

```

Se obtiene el tiempo actual y se genera un archivo de Log en modo de escritura para este envío, donde se muestra el nombre del archivo y su tamaño, el cliente al que se envió, el puerto del socket del servidor y el tiempo de transferencia.

```
70
71      final = time.time()
72      tiempoProceso = final - inicio
73      print(f"El tiempo de procesamiento y envio para el cliente {cuentaCliente} es: {tiempoProceso}")
74
75      year = str(now)[:4]
76      month = str(now)[5:7]
77      day = str(now)[8:10]
78      hour = str(now)[11:13]
79      minute = str(now)[14:16]
80      second = str(now)[17:19]
81
82      save_path = 'Logs/'
83      file_name = 'S'+ str(cuentaCliente) + '-' + year + '-' + month + '-' + day + '-' + hour + '-' + minute + '-' + second + '-' + fileName + '-' + str(clientesSimultaneos) + '-' + 'log.txt'
84      completeName = os.path.join(save_path,file_name)
85      newFile = open(completeName, 'w')
86      newFile.write('El archivo enviado fue: ' + fileName +'\n')
87      newFile.write('El archivo tiene un tamano de: ' + str(fileSize) + ' bytes\n')
88      newFile.write('El cliente al que le fue enviado es: ' + str(cuentaCliente) +'\n')
89      newFile.write('El puerto del servidor es: ' + str(PORTS + cuentaCliente))
90      newFile.write(f'El tiempo de transferencia para este cliente fue: {tiempoProceso} segundos\n')
91
```

Cliente: Primero se crea un socket TCP para establecer la conexión con el servidor. A través de este recibe su número de cliente, con el cual calcula el número del puerto UDP sumándose al número de puerto TCP y lo envía al servidor para, por último, recibir el número de conexiones simultáneas que espera el servidor.

```
18
19      sockTCP = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
20      sockTCP.bind(ADDRC)
21      sockTCP.connect(ADDRS)
22      print(f"Cliente conectado al servidor {IPS}:{PORTS}")
23      numeroCod = sockTCP.recv(1024)
24      numeroCliente = int.from_bytes(numeroCod, 'little')
25      print("Numero de cliente: " + str(numeroCliente))
26      puertoNuevo = PORTC + numeroCliente
27      print(f"Puerto actual: {puertoNuevo}")
28      sockTCP.send(puertoNuevo.to_bytes(3, 'little'))
29      conexCod = sockTCP.recv(1024)
30      conexionesSimultaneas = int.from_bytes(conexCod, 'little')
31      print(f"Conexiones simultaneas esperadas: {conexionesSimultaneas}")
32
```

Luego se crea el socket UDP con el puerto generado y, a través de este, se recibe el nombre del archivo y su tamaño en el servidor. Con la información recolectada se crea un archivo vacío a modo de escritura binaria para escribir lo que se recibe del servidor, se obtiene la fecha actual y se inicia el contador de tiempo.

```
32
33      sockUDP = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
34      sockUDP.bind((IPC, puertoNuevo))
35
36      fileCod, addr = sockUDP.recvfrom(SIZE)
37      fileName = fileCod.decode()
38      sizeCod, addr = sockUDP.recvfrom(SIZE)
39      serverSize = int.from_bytes(sizeCod, 'little')
40
41      print("Se establecio la comunicación con el servidor")
42      print("IP y puerto del servidor: " + "(" + str(addr[0]) + ", " + str(addr[1]) + ")")
43      print("Nombre del archivo por recibir: " + fileName)
44      print("Tamano del archivo por recibir: " + str(serverSize))
45
46      file = open(f"ArchivosRecibidos/{numeroCliente}-Prueba-{conexionesSimultaneas}", 'wb')
47
48      now = datetime.datetime.now()
49      inicio = time.time()
50
```

Para recibir los bloques lee la cabecera del socket para ver si hay datos por leer. Si es así, se recibe del socket del servidor el bloque y lo escribe en el archivo. De lo contrario, indica que terminó de leer el archivo, lo cierra y rompe el ciclo.

```
50
51      while True:
52          ready = select.select([sockUDP], [], [], timeout)
53          if ready[0]:
54              data, addr = sockUDP.recvfrom(SIZE)
55              file.write(data)
56          else:
57              print ("%s Terminado!" % fileName)
58              file.close()
59              break
60
```

Al terminar de leer el archivo, obtiene el contador final y calcula el tiempo del proceso, calcula el tamaño del archivo del lado del cliente y obtiene los datos de la fecha de la operación.

```
60
61      final = time.time()
62      tiempoProceso = final - inicio
63      print(f"Tiempo de comunicacion y envio: {tiempoProceso}")
64
65      filesize = os.path.getsize(f"ArchivosRecibidos/{numeroCliente}-Prueba-{conexionesSimultaneas}")
66
67      year = str(now)[:4]
68      month = str(now)[5:7]
69      day = str(now)[8:10]
70      hour = str(now)[11:13]
71      minute = str(now)[14:16]
72      second = str(now)[17:19]
73
```

Por último, genera el archivo del Log en modo de escritura y registra en el si el archivo se envió completamente comparando los tamaños del servidor y el cliente, el nombre y el tamaño del archivo, el cliente al que se le envió junto al puerto del socket UDP y el tiempo de transferencia.

```
73
74      save_path = 'Logs/'
75      file_name = 'C'+ str(numeroCliente) + '-' + year + '-' + month + '-' + day + '-' + hour + '-' + minute + '-' + second + '-' + fileName + '-' + str(conexionesSimultaneas) + '-' + 'log.txt'
76      completeName = os.path.join(save_path,file_name)
77      newFile = open(completeName, 'w')
78      if serverSize == filesize:
79          newFile.write('El archivo se ha enviado exitosamente'+'\n')
80      else:
81          newFile.write('El archivo ha tenido un fallo o paquete faltante al enviarse'+'\n')
82      newFile.write('El archivo enviado fue: ' + fileName +'\n')
83      newFile.write('El archivo tiene un tamano de: ' + str(filesize) + ' bytes\n')
84      newFile.write('El cliente al que le fue enviado es: ' + str(numeroCliente) +'\n')
85      newFile.write('El puerto del cliente es: ' + str(puertoNuevo))
86      newFile.write(f'El tiempo de transferencia para este cliente fue {tiempoProceso} segundos\n')
87
```

Pruebas de latencia:

Prueba 1: *100 MB para 1 cliente*

Transferencia exitosa	Puerto por cliente	Puerto por servidor	Bytes recibidos	Bytes transmitidos por el servidor	Tiempo de transferencia	Tasa de transferencia
No	12045	5006	104809472	104857600	4.521 segundos	23.183 MB/seg

Prueba 2: *250 MB para 1 cliente*

Transferencia exitosa	Puerto por cliente	Puerto por servidor	Bytes recibidos	Bytes transmitidos por el servidor	Tiempo de transferencia	Tasa de transferencia
No	33098	5006	262128640	262144000	12.969 segundos	20.212 MB/seg

Prueba 3: *100 MB para 5 clientes*

Transferencia exitosa	Puerto por cliente	Puerto por servidor	Bytes recibidos	Bytes transmitidos por el servidor	Tiempo de transferencia	Tasa de transferencia
No	53276	5006	101954560	104857600	9.317 seg	10.943 MB/seg
No	19649	5007	104153088	104857600	9.415 seg	11.068 MB/seg
No	40325	5008	104843264	104857600	9.457 seg	11.086 MB/seg
No	28060	5009	103467008	104857600	9.378 seg	11.033 MB/seg
No	11005	5010	104501248	104857600	9.389 seg	11.13 MB/seg

Prueba 4: *250 MB para 5 clientes*

Transferencia exitosa	Puerto por cliente	Puerto por servidor	Bytes recibidos	Bytes transmitidos por el servidor	Tiempo de transferencia	Tasa de transferencia
No	48271	5006	239524864	262144000	17.937 seg	13.354 MB/seg
No	23740	5007	240753664	262144000	18.676 seg	12.891 MB/seg
No	16203	5008	244147200	262144000	19.021 seg	12.836 MB/seg
No	52615	5009	257448960	262144000	17.457 seg	14.748 MB/seg
No	10284	5010	251114496	262144000	17.779 seg	14.124 MB/seg

Prueba 5: *100 MB para 10 clientes*

Transferencia exitosa	Puerto por cliente	Puerto por servidor	Bytes recibidos	Bytes transmitid	Tiempo de transferen	Tasa de transferen

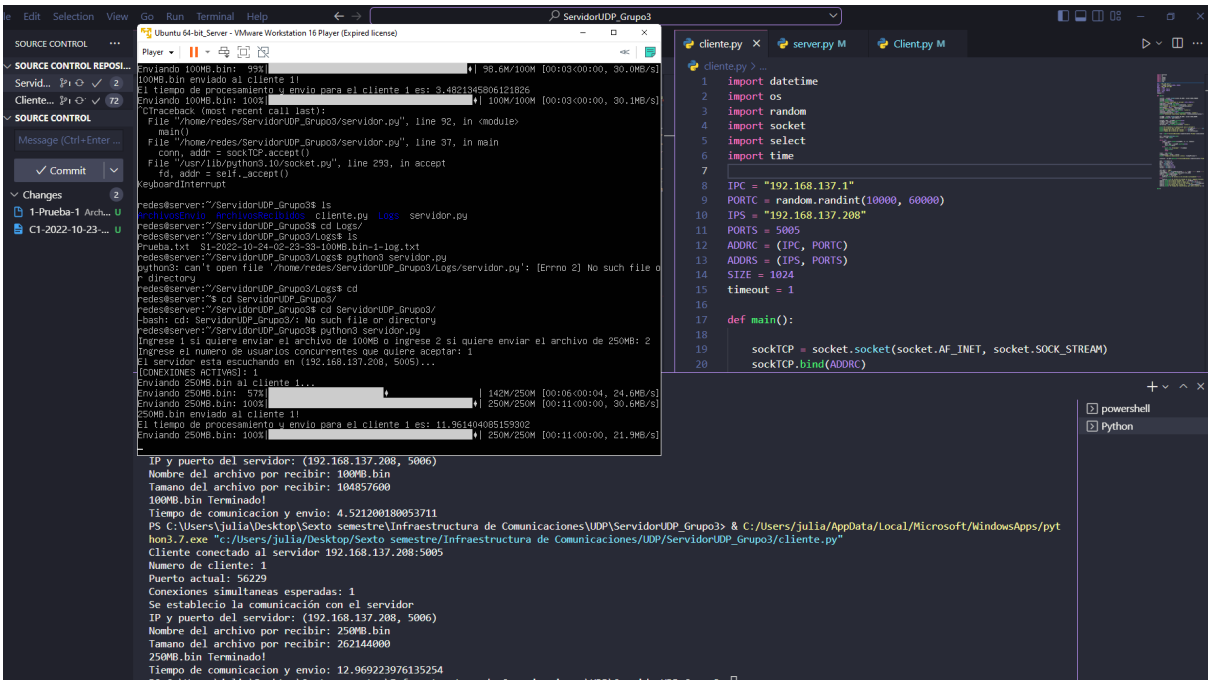
				os por el servidor	cia	cia
Si	52615	5006	104857600	104857600	21.127 seg	4.963 MB/seg
No	17261	5007	10338508	104857600	20.909 seg	4.945 MB/seg
No	37080	5008	104604672	104857600	20.787 seg	5.032 MB/seg
No	38199	5009	103974912	104857600	21.218 seg	4.9 MB/seg
No	12463	5010	104301568	104857600	21.393 seg	4.875 MB/seg
No	18994	5011	104493056	104857600	20.927 seg	4.993 MB/seg
No	20845	5012	104780800	104857600	21.173 seg	4.949 MB/seg
No	49188	5013	104857600	104857600	20.999 seg	4.993 MB/seg
No	53706	5014	104728576	104857600	20.935 seg	5.003 MB/seg
No	15271	5015	103843840	104857600	21.201 seg	4.898 MB/seg

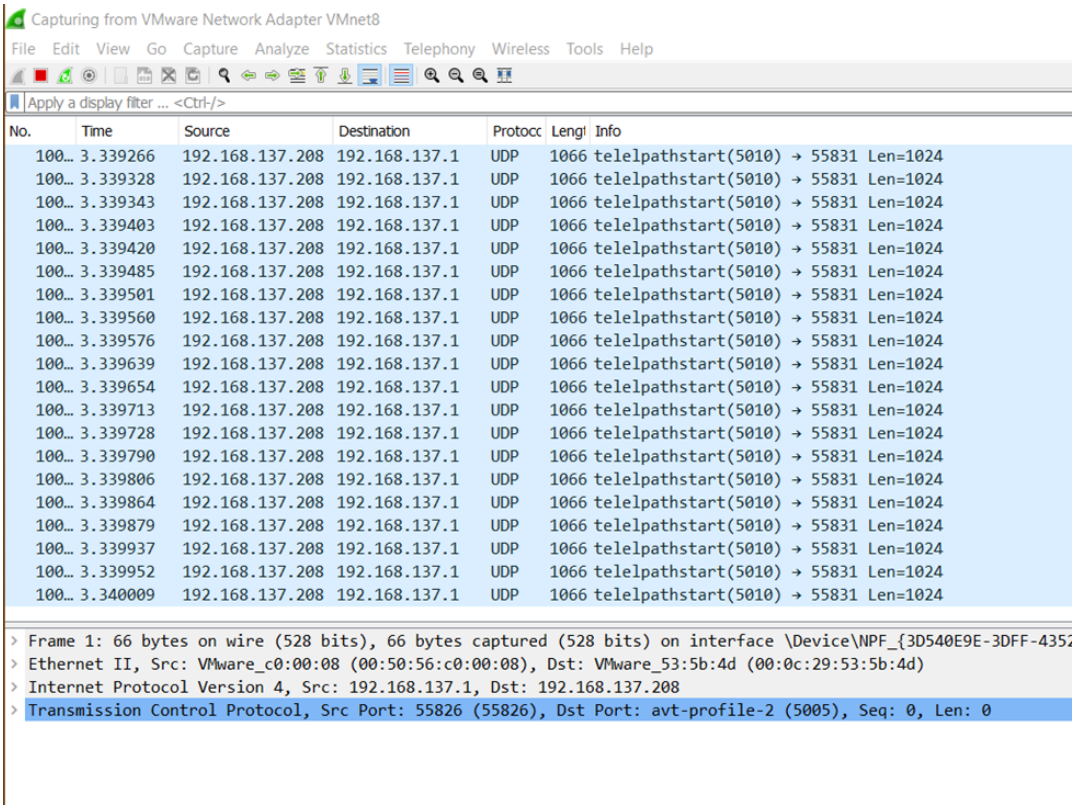
Prueba 6: *250 MB para 10 clientes*

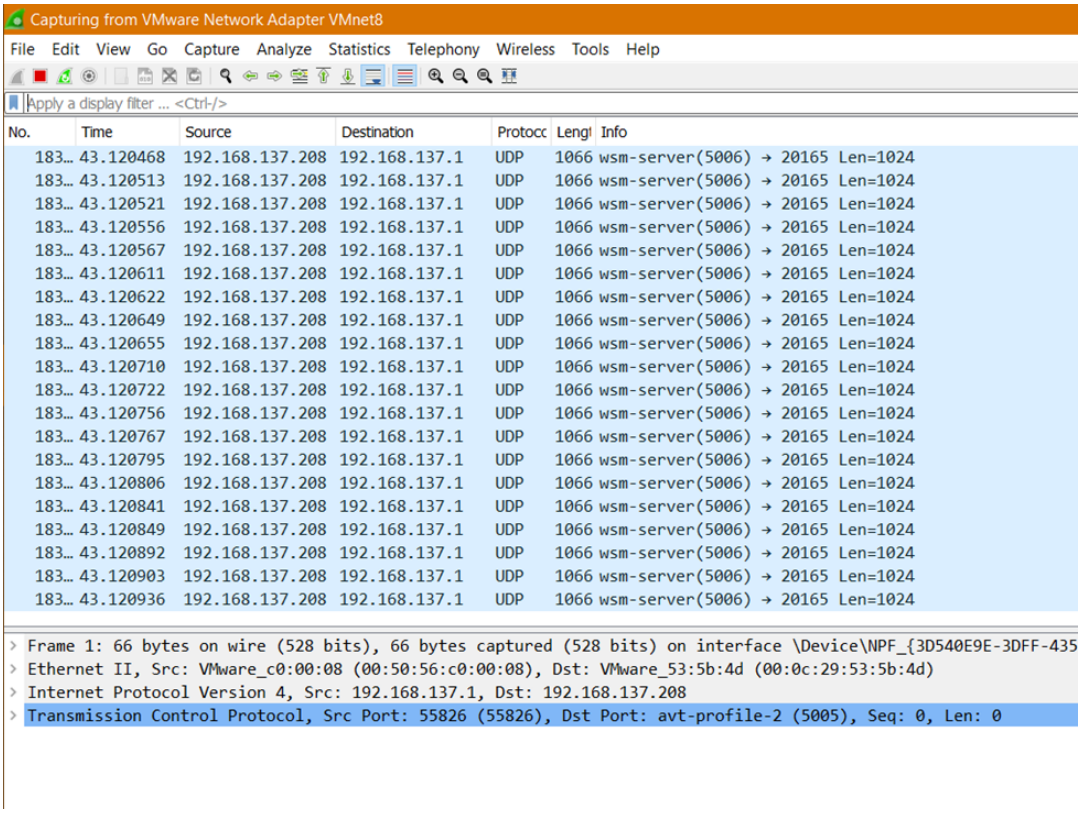
Transferencia exitosa	Puerto por cliente	Puerto por servidor	Bytes recibidos	Bytes transmitidos por el servidor	Tiempo de transferencia	Tasa de transferencia
No	50716	5006	255816704	262144000	65.116 seg	3,929 MB/seg
No	36271	5007	254669824	262144000	64.578 seg	3,944 MB/seg
No	26302	5008	256391168	262144000	65.276 seg	3,928 MB/seg

No	19826	5009	251800576	262144000	65.236 seg	3,86 MB/seg
No	13298	5010	255964160	262144000	64.782 seg	3,951 MB/seg
No	27308	5011	255768576	262144000	63.71 seg	4,015 MB/seg
No	16297	5012	249767936	262144000	64.751 seg	3,857 MB/seg
No	49201	5013	254792704	262144000	65.217 seg	3,907 MB/seg
No	32865	5014	254968832	262144000	64.467 seg	3,955 MB/seg
No	20477	5015	251913216	262144000	64.006 seg	3,936 MB/seg

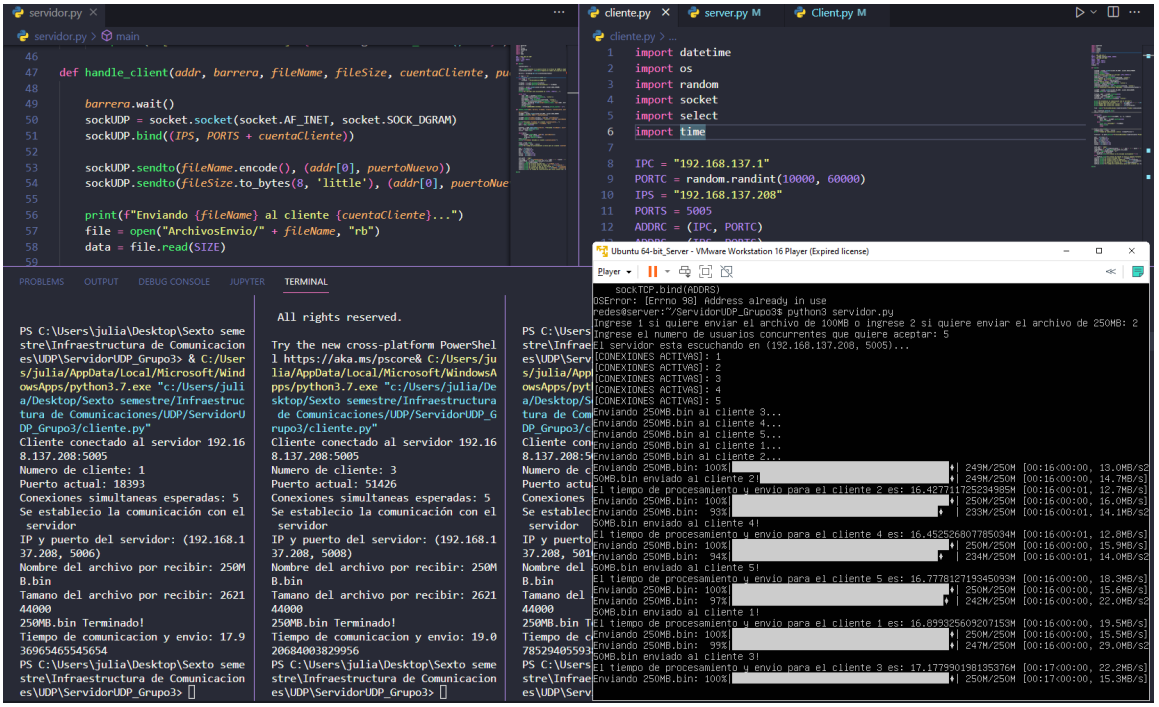
Pruebas 1 y 2, archivos de 100 MB y 250 MB para 1 cliente, respectivamente:

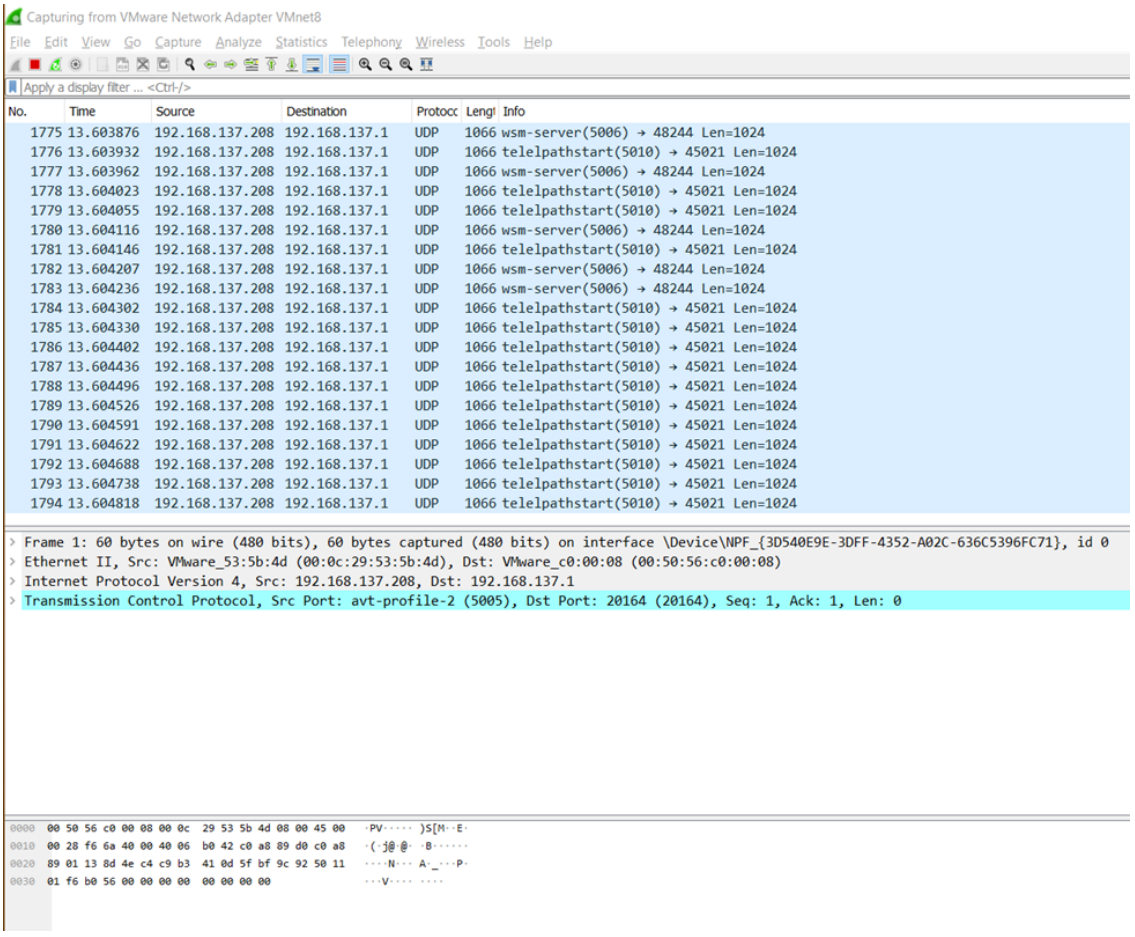



Capturing from VMware Network Adapter VMnet8
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocc Lengt Info
100… 3.339266 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339328 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339343 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339403 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339420 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339485 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339501 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339560 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339576 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339639 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339654 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339713 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339728 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339790 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339806 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339864 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339879 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339937 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.339952 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
100… 3.340009 192.168.137.208 192.168.137.1 UDP 1066 telelpathstart(5010) → 55831 Len=1024
Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{3D540E9E-3DFF-4352
Ethernet II, Src: VMware_c0:00:08 (00:50:56:c0:00:08), Dst: VMware_53:5b:4d (00:0c:29:53:5b:4d)
Internet Protocol Version 4, Src: 192.168.137.1, Dst: 192.168.137.208
Transmission Control Protocol, Src Port: 55826 (55826), Dst Port: avt-profile-2 (5005), Seq: 0, Len: 0

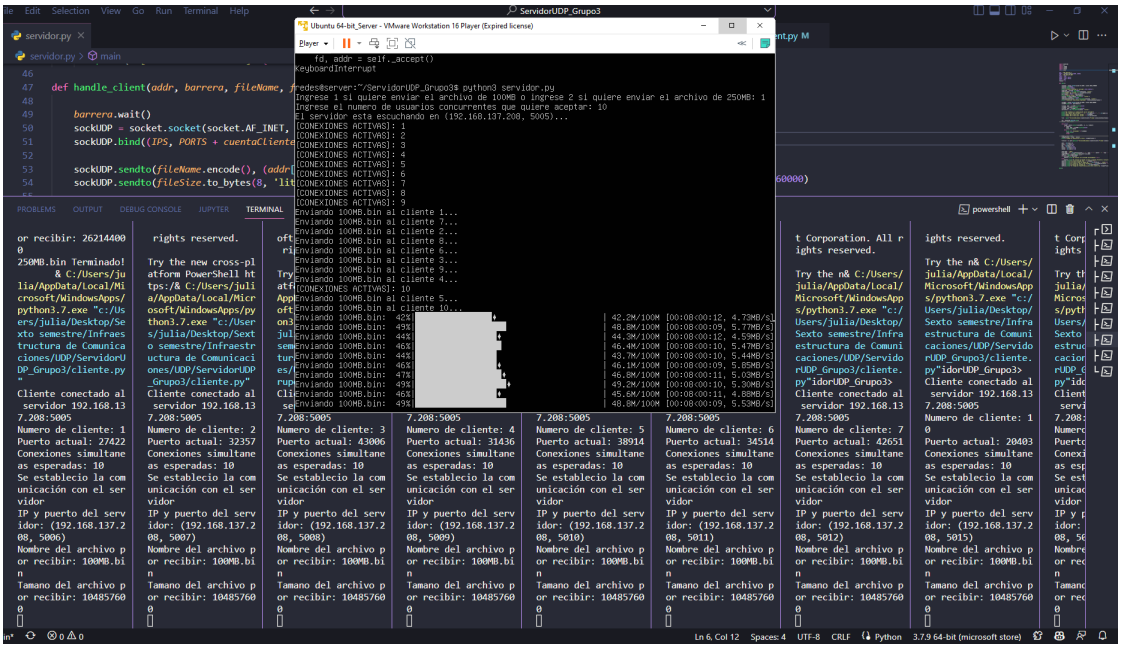

Capturing from VMware Network Adapter VMnet8
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocc Lengt Info
183… 43.120468 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120513 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120521 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120556 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120567 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120611 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120622 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120649 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120655 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120710 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120722 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120756 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120767 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120795 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120806 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120841 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120849 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120892 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120903 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
183… 43.120936 192.168.137.208 192.168.137.1 UDP 1066 wsm-server(5006) → 20165 Len=1024
Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{3D540E9E-3DFF-435
Ethernet II, Src: VMware_c0:00:08 (00:50:56:c0:00:08), Dst: VMware_53:5b:4d (00:0c:29:53:5b:4d)
Internet Protocol Version 4, Src: 192.168.137.1, Dst: 192.168.137.208
Transmission Control Protocol, Src Port: 55826 (55826), Dst Port: avt-profile-2 (5005), Seq: 0, Len: 0

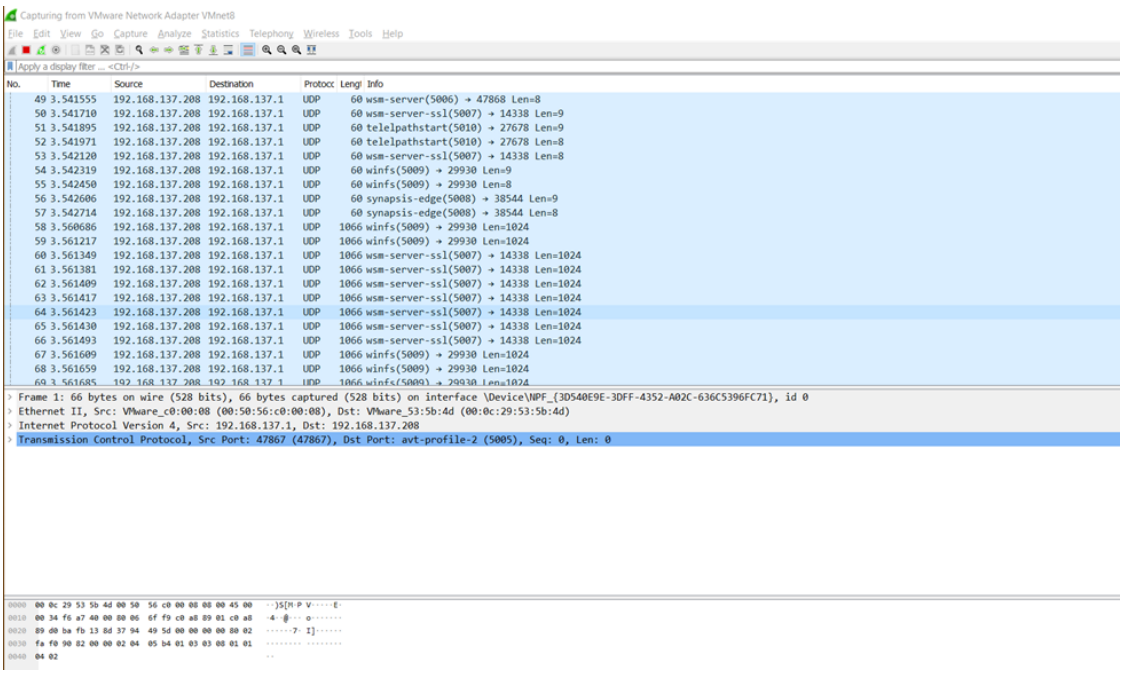
Prueba 3, envío de archivo 100 MB a 5 clientes:



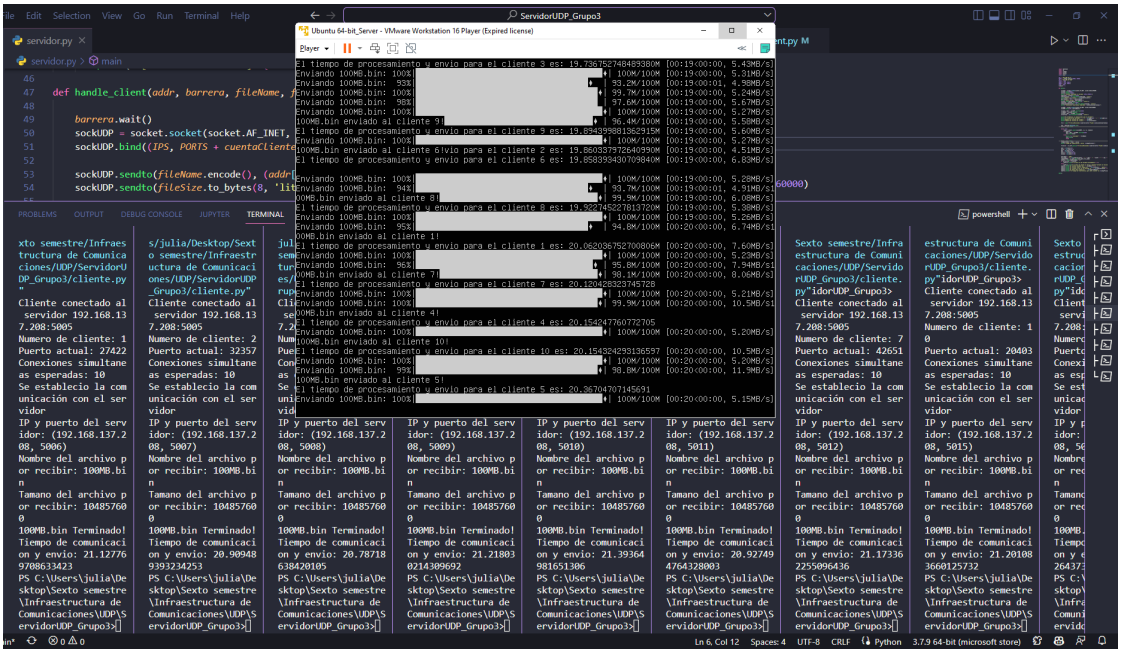


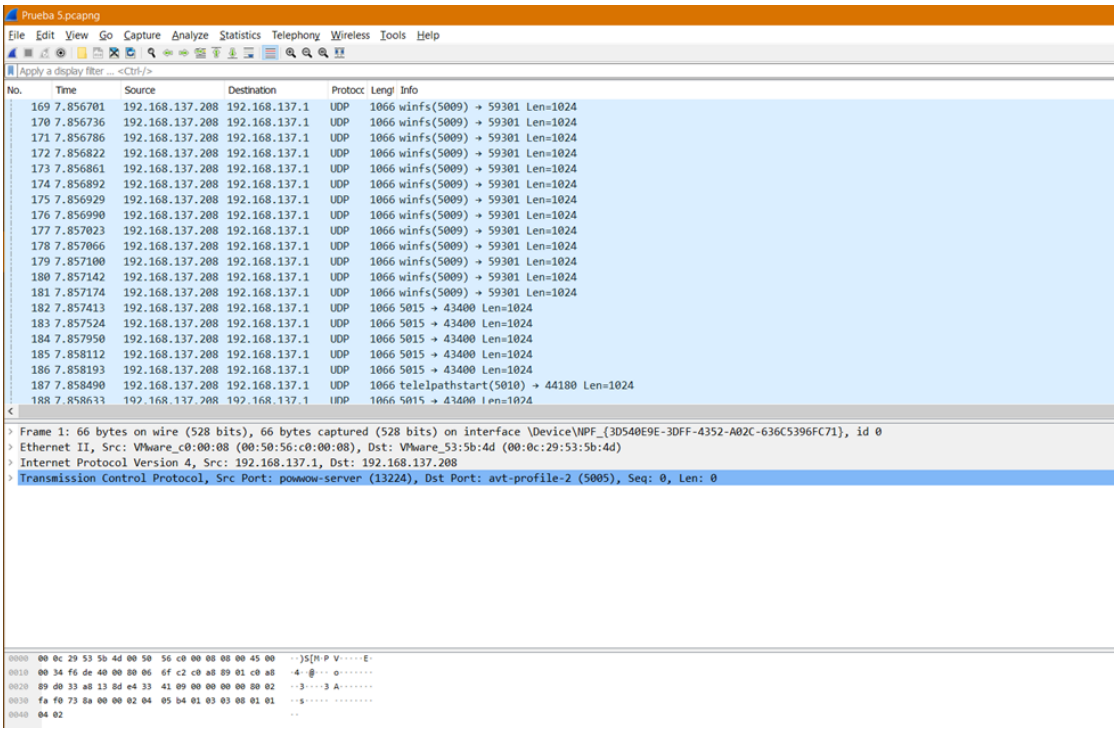
Prueba 4, envío de archivo 250 MB a 5 clientes:



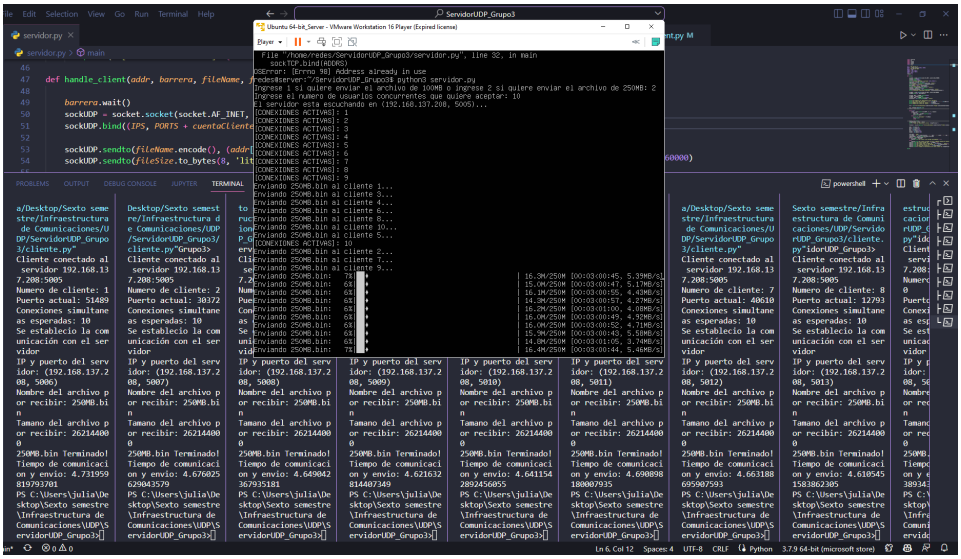


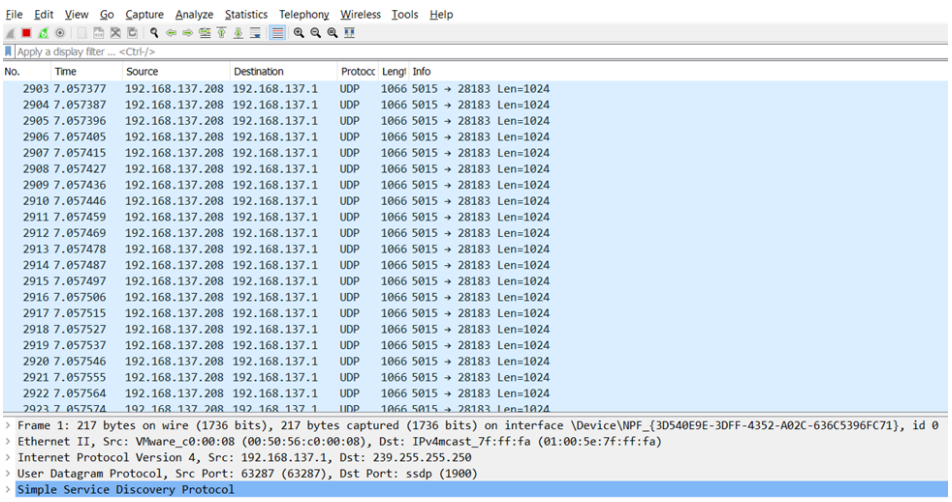
Prueba 5, envío de archivo 100 MB a 10 clientes:





Prueba 6, envío de archivo 250 MB a 10 clientes:





Análisis de resultados:

Luego de realizar todas las pruebas correspondientes a ambos laboratorios, tanto TCP como UDP, podemos llegar a la conclusión de que la transferencia de archivos promedio es menor cuando se implementa como UDP. Sin embargo, cuando la implementación es del tipo UDP, se presentaron mayores pérdidas en los archivos y algunas transferencias no fueron exitosas, contrastando con las de TCP, las cuales funcionaron de manera adecuada.

Preguntas:

1. Si tuviera que desarrollar un servicio de streaming de video con una arquitectura Cliente/Servidor donde la transmisión fuera en multidifusión. Mencione cuales serían las consideraciones técnicas que tendría en cuenta para el desarrollo del servicio, sea lo más detallado posible.

Se tendrá en cuenta un buffer con capacidad del tamaño del archivo,
Es necesario generar copias de los datos cuando los enlaces destino son distintos
Se debe tener habilitado los puertos de comunicación
Usar procesos ETL para extraer, transformar y cargar datos en distintas bases de datos

2. ¿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. Justifique su respuesta.

De por si el protocolo no genera retroalimentación del envío entre el cliente-servidor y da lugar a la posibilidad de incurrir en pérdida de paquetes, aunque existen técnicas de ordenamiento (numeración secuencial) y confirmación (acks redundantes) similares a las que se aplican en TCP que mejoran la tasa de entrega del protocolo UDP.

3. ¿Se podrían considerar servicios de emisión de contenidos que funcionen con el protocolo TCP? Justifique su respuesta.

No sería lo adecuado si se espera transmisión en tiempo real, ya que habrá retrasos considerables y el contenido no sería fluido por lo tanto no sería de buena calidad la comunicación. No satisface los requerimientos propios del negocio. Sin embargo, si no importa eso, y por ejemplo la transmisión se hace y posterior a que se acabe es que el usuario va a usarla entonces si es posible hacerlo.