Práctica 6

1. **¿Cuál es el puerto por defecto que se utiliza en los siguientes servicios? Web / SSH / DNS / Web Seguro / POP3 / IMAP / SMTP**
   Investigue en qué lugar en Linux y en Windows está descripta la asociación utilizada por defecto para cada servicio.

Servicio	Puerto por defecto
Web	80 (HTTP)
SSH	22
DNS	53
Web Seguro	443 (HTTPS)
POP3	110
IMAP	143

   En Linux, la asociación de puertos por defecto para los servicios se encuentra en el archivo /etc/services. Este archivo es un archivo de texto que contiene una lista de servicios registrados, junto con sus números de puerto, protocolos y nombres de dominio.

   En Windows, la asociación de puertos por defecto para los servicios se encuentra en el archivo %SystemRoot%\System32\drivers\etc\services. Este archivo es similar al archivo /etc/services de Linux.

2. **Investigue qué es multicast. ¿Sobre cuál de los protocolos de capa de transporte funciona? ¿Se podría adaptar para que funcione sobre el otro protocolo de capa de transporte? ¿Por qué?**

   El multicast es una técnica que permite enviar un mensaje a un grupo de destinatarios de forma simultánea. A diferencia del broadcast, que envía un mensaje a todos los dispositivos de una red, el multicast solo envía el mensaje a los dispositivos que están interesados en recibirlo.

   La técnica del multicast funciona sobre UDP, ya que no necesita establecer una conexión y se podría usar un mismo socket (un proceso tiene asociado un socket) para recibir datos de varios procesos que se quieren comunicar con un proceso a la vez.

   Teóricamente podría intentarse adaptar multicast sobre TCP, pero sería demasiado complejo e iría en contra de la naturaleza del modelo ya TCP establece una conexión punto a punto entre un único emisor y receptor.

3. **Investigue cómo funciona el protocolo de aplicación FTP teniendo en cuenta las diferencias en su funcionamiento cuando se utiliza el modo activo de cuando se utiliza el modo pasivo ¿En qué se diferencian estos tipos de comunicaciones del resto de los protocolos de aplicación vistos?**

FTP requiere dos conexiones TCP. Una conexión de control y otra para la transferencia de datos**.** El cliente escoge cualquier puerto no privilegiado, (n > 1023) y genera conexión de control contra el puerto 21 del servidor. El servidor recibe los comandos por dicha conexión y responde/recibe por la conexión de datos aquellos que lo requieran. La conexión de datos se crea y de cierra bajo demanda. El estado de cada operación se transmite por el canal de control.

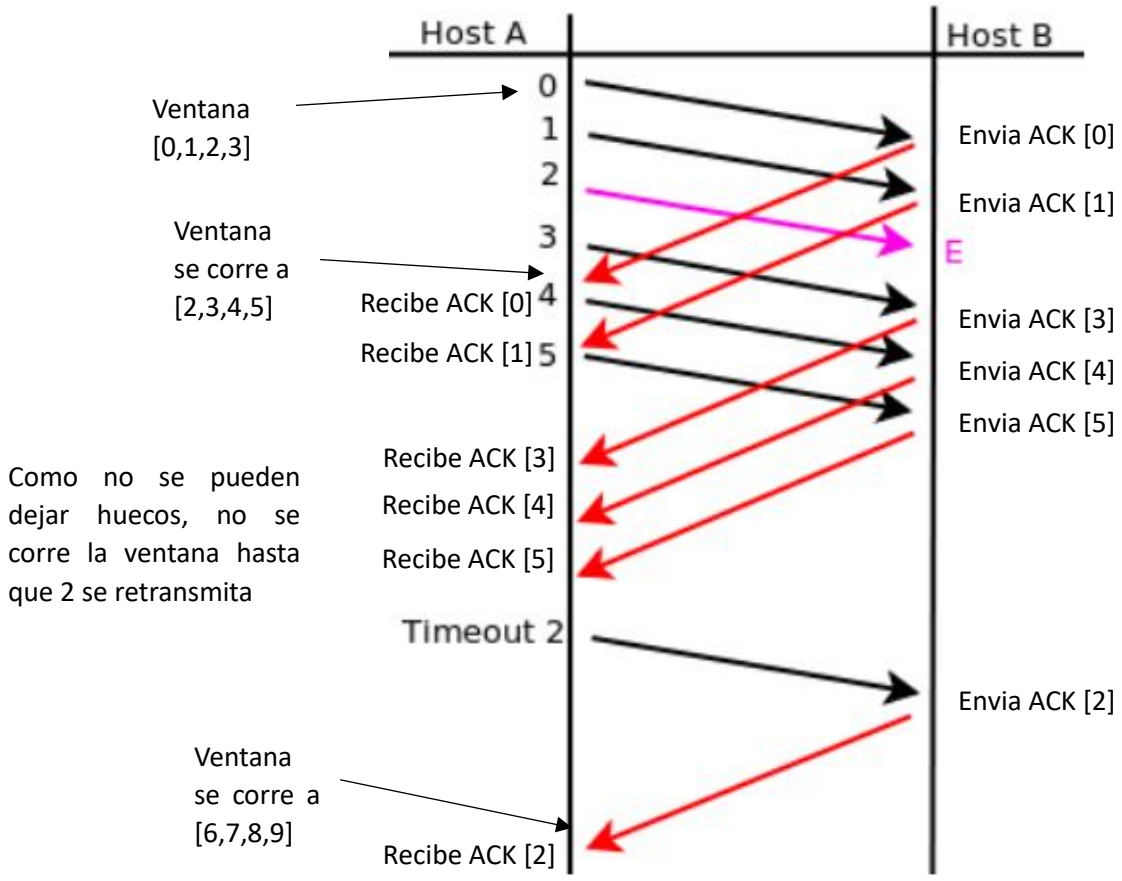
Modo Activo

- Conexión de control: port 21.
- Conexión de datos: port 20.
- El servidor de forma activa se conecta al cliente para generar la conexión de datos.

Modo Pasivo

- Conexión de control: port 21.
- Conexión de datos: port no privilegiado.
- El servidor de forma pasiva indica al cliente a que nuevo puerto debe conectarse. La conexión de datos la abre el cliente.

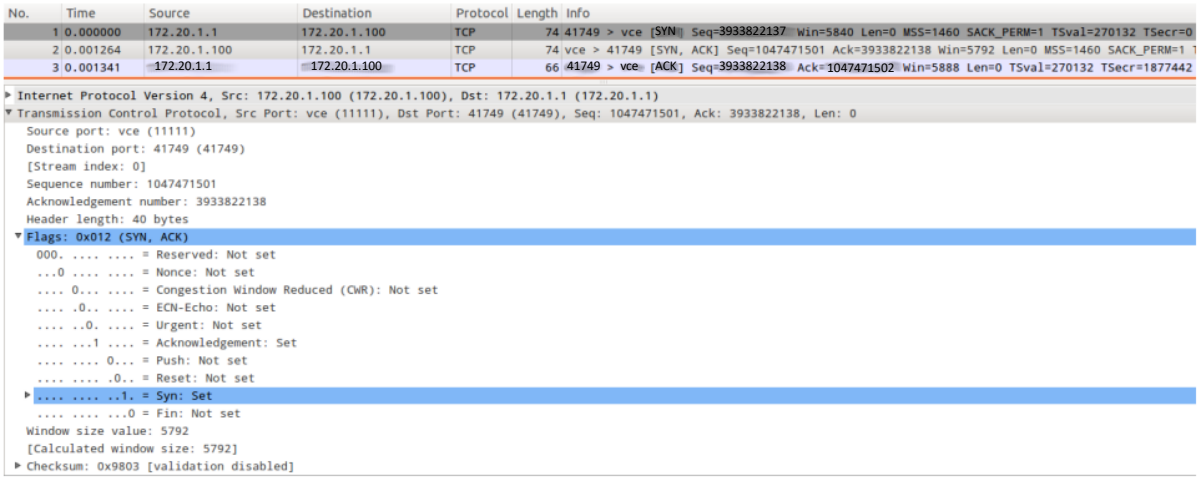
4. **Suponiendo Selective Repeat; tamaño de ventana 4 y sabiendo que E indica que el mensaje llegó con errores. Indique en el siguiente gráfico, la numeración de los ACK que el host B envía al Host A.**



5. **¿Qué restricción existe sobre el tamaño de ventanas en el protocolo Selective Repeat?**

   El tamaño de la ventana no debe exceder la mitad del tamaño total del espacio de números de secuencia. La razón detrás de esta restricción es evitar la posibilidad de que un número de secuencia se reutilice antes de que el ACK correspondiente haya llegado, ya que la ventana se implementa como un buffer circular, entonces si fuese más grande podría haber paquetes representados por la misma posición en el buffer lo que podría llevar a confusiones en la correcta interpretación de los frames.

6. **De acuerdo a la captura TCP de la siguiente figura, indique los valores de los campos borroneados.**



SYN ➔Comienzo de 3WH

3933822137 ➔Se que es ese porque el receptor (linea 2) me indica que espera (ACK) que se le envie el segmento 3833822138, por lo tanto el que le envie en 1 es 3833822138 – 1

172.20.1.1 ➔ IP Origen

172.20.1.100 ➔ IP Destino

41749 ➔ Puerto Destino

vce ➔ Puerto Origen

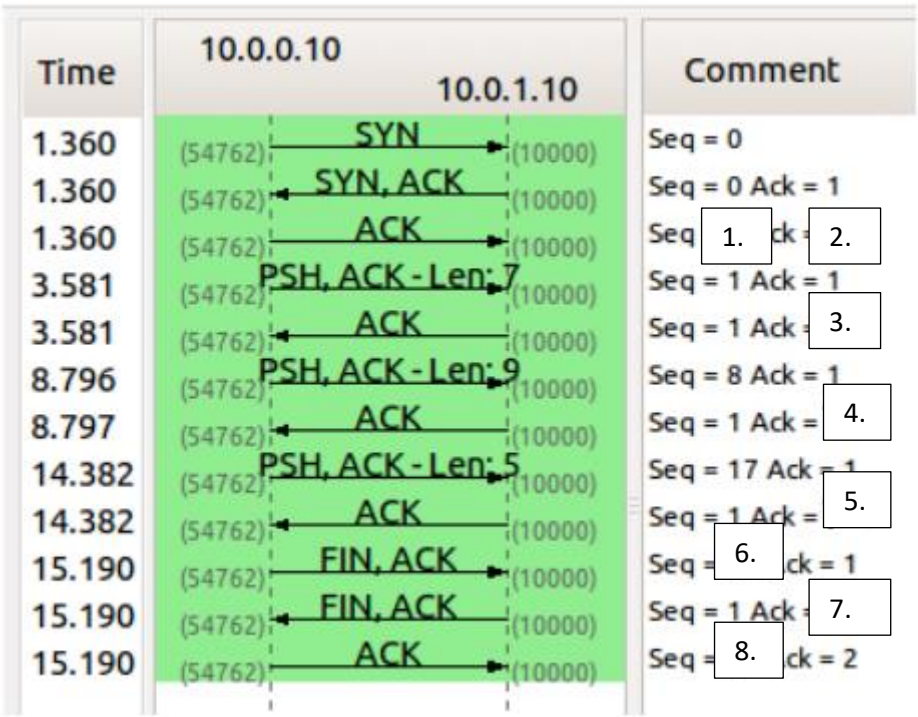
ACK ➔ Fin de 3WH

3933822138 ➔ En el segmento anterior a este se indicó que se esperaba 3933822138

1047471502 ➔ Se recibió 1047471501 , por lo que se espera recibir 1047471503

Las confirmaciones son “anticipativas”, indican el nro. de byte que esperan.

7. **Dada la sesión TCP de la figura, completar los valores marcados con un signo de interrogación.**



1. 1
2. 1
3. 8
4. 17
5. 22
6. 22
7. 23
8. 23

8. ¿Qué es el RTT y cómo se calcula? Investigue la opción TCP timestamp y los campos TSval y TSecr.

El RTT es el tiempo que tarda un paquete en viajar desde un host a otro y recibir un ACK de vuelta.

La opción de marcas de tiempo en TCP permite a los endpoints mantener una medición más precisa del tiempo de ida y vuelta (RTT) de la red entre ellos. Este valor ayuda a cada pila TCP a configurar y ajustar su temporizador de retransmisión. Hay otros beneficios, pero la medición RTT es el principal.

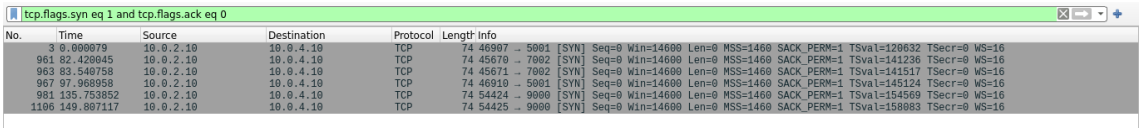
Para ello se incluye un Timestamp Value TSval en cada segmento que se envía. Los valores TSval se repiten en el lado opuesto de la conexión en el campo Timestamp Echo Reply TSecr . Entonces, cuando se confirma un segmento, el remitente de ese segmento puede simplemente restar su marca de tiempo actual del valor TSecr para calcular una medición precisa del tiempo de ida y vuelta (RTT).

$RTT = TSecr - TSval$

9. Para la captura dada, responder las siguientes preguntas.

a. ¿Cuántos intentos de conexiones TCP hay?

6

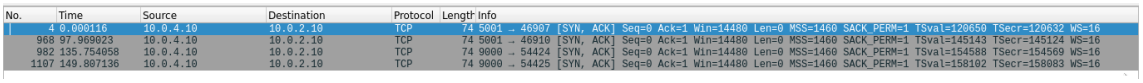


b. ¿Cuáles son la fuente y el destino (IP:port) para c/u?

Fuente	Destino
10.0.2.10:46907	10.0.4.10:5001
10.0.2.10:45670	10.0.4.10:7002
10.0.2.10:45671	10.0.4.10:7002
10.0.2.10:46910	10.0.4.10:5001
10.0.2.10:54424	10.0.4.10:9000
10.0.2.10:54425	10.0.4.10:9000

c. ¿Cuántas conexiones TCP exitosas hay en la captura? Cómo diferencia las exitosas de las que no lo son? ¿Cuáles flags encuentra en cada una?

4



Las exitosas tienen los flags SYN/ACK en 1, las fallidas tienen los flags RST/ACK en 1

d. Dada la primera conexión exitosa responder:

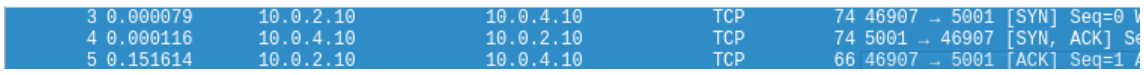
i. ¿Quién inicia la conexión?

La conexión es iniciada por 10.0.2.10:46907

ii. ¿Quién es el servidor y quién el cliente?

El cliente es el que inicia la conexión 10.0.2.10:46907 y el servidor es el destino 10.0.4.10:7002

iii. ¿En qué segmentos se ve el 3-way handshake?



iv. ¿Cuáles ISNs se intercambian?

1 –

```
Sequence Number: 0    (relative sequence number)
Sequence Number (raw): 2218428254
[Next Sequence Number: 1    (relative sequence number)]
Acknowledgment Number: 0
Acknowledgment number (raw): 0
```

2 –

```
Sequence Number: 0    (relative sequence number)
Sequence Number (raw): 1292618479
[Next Sequence Number: 1    (relative sequence number)]
Acknowledgment Number: 1    (relative ack number)
Acknowledgment number (raw): 2218428255
```

3 –

```
Sequence Number: 1    (relative sequence number)
Sequence Number (raw): 2218428255
[Next Sequence Number: 1    (relative sequence number)]
Acknowledgment Number: 1    (relative ack number)
Acknowledgment number (raw): 1292618480
```

Se intercambian los ISN 2218428254, 1292618479 y 2218428255.

v. ¿Cuál MSS se negoció?

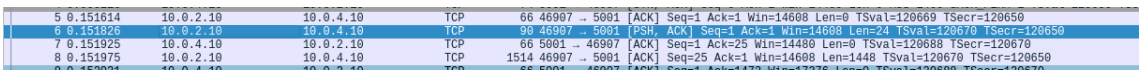
```
▾ Options: (20 bytes), Maximum segment size, SACK permitted,
  ▾ TCP Option - Maximum segment size: 1460 bytes
      Kind: Maximum Segment Size (2)
      Length: 4
      MSS Value: 1460
```

Se puede ver el segmento Nro 4 (siguiendo el orden de Wireshark)

vi. ¿Cuál de los dos hosts envía la mayor cantidad de datos (IP:port)?

10.0.2.10:46907, se incrementa su numero de secuencia (se incrementa cuando se envían datos), mientras que 10.0.4.10:7002 nunca lo incrementa (salvo en el 3WH). 10.0.2.10:46907 termina con el ISN relativo de 786458 y 10.0.4.10:7002 con 1.

e. Identificar primer segmento de datos (origen, destino, tiempo, número de fila y número de secuencia TCP).



(el azul)

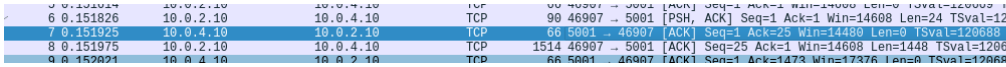
Origen: 10.0.2.10:46907
Destino: 10.0.4.10:7002
Tiempo: 0.151826

Nro. de fila: 6
Nro. de secuencia TCP: 1 (*221842855*)

i. **¿Cuántos datos lleva?**

Lleva 24 bytes.

ii. **¿Cuándo es confirmado (tiempo, número de fila y número de secuencia TCP)?**



(el azul)

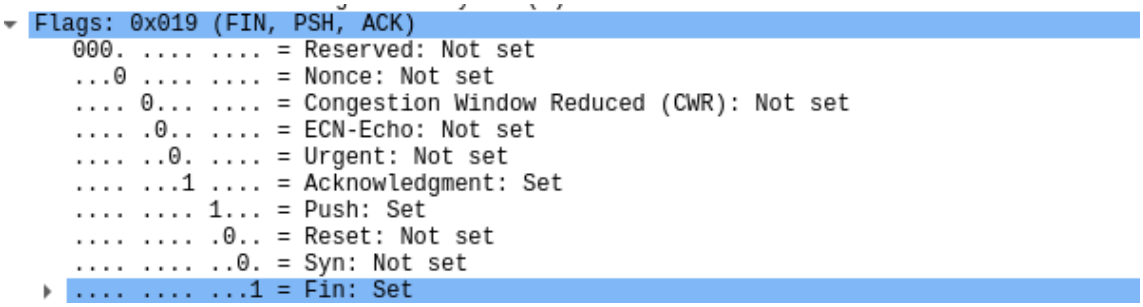
Tiempo: 0.151925
Nro. de fila: 7
Nro. de secuencia TCP: 1 (*1292618480*)

iii. **La confirmación, ¿qué cantidad de bytes confirma?**

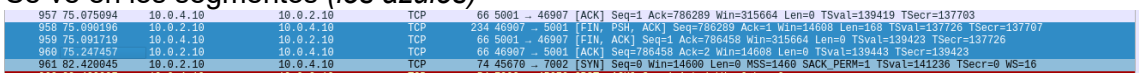
Confirma los 24 bytes, ya que indica que espera el byte nro 25.

f. **¿Quién inicia el cierre de la conexión? ¿Qué flags se utilizan? ¿En cuáles segmentos se ve (tiempo, número de fila y número de secuencia TCP)?**

La inicia 10.0.2.10:46907. Utiliza los flags FIN, PSH y ACK



Se ve en los segmentos *(los azules)*



10. **Responda las siguientes preguntas respecto del mecanismo de control de flujo**

a. **¿Quién lo activa? ¿De qué forma lo hace?**

El control de flujo lo activa el receptor enviando ventanas más chicas. Esto deja en evidencia que el receptor tiene poco espacio (o no tiene más lugar) para seguir recibiendo datos. Esto se realiza a través del

campo de tamaño de ventana en los encabezados de los segmentos TCP.

b. ¿Qué problema resuelve?

Resuelve el problema de la posible saturación o congestión de los buffers en los endpoints. Al indicar al emisor que reduzca la cantidad de datos que está enviando, evita que el receptor se sobrecargue.

c. ¿Cuánto tiempo dura activo y qué situación lo desactiva?

Cuanto tiempo dura activo depende del receptor (más que nada la velocidad en que lee la aplicación). El control de flujo está activo mientras el receptor envíe ventanas más pequeñas (indicando capacidad limitada). Durará activo hasta que el receptor envíe ventanas más grandes, lo que indica que tiene más capacidad para recibir datos.

En todo momento ambos extremos están actualizando su propia ventana.

11. Responda las siguientes preguntas respecto del mecanismo de control de congestión.

a. ¿Quién lo activa el mecanismo de control de congestión? ¿Cuáles son los posibles disparadores?

El control de congestión lo activa el emisor. El emisor limita la velocidad de transmisión de tráfico a través de su conexión en función de la congestión de red percibida. Este proceso es dinámico y adaptativo, y el emisor ajusta su velocidad de transmisión en respuesta a las condiciones cambiantes de la red.

Los posibles disparadores son:

- Fin de Temporización: La expiración del temporizador asociado con el envío de un segmento TCP puede ser interpretada como una señal de pérdida, indicando posiblemente congestión en la ruta.

- Recepción de **TRES** ACK Duplicados: La recepción de paquetes ACK duplicados procedentes del receptor también se interpreta como un suceso de pérdida. Este evento puede sugerir la pérdida de un paquete en la red debido a congestión.

b. ¿Qué problema resuelve?

El objetivo es que no se desborde la propia red. Esto ocurre cuando hay más tráfico de red del que la red puede manejar eficientemente, lo que puede resultar en la pérdida de paquetes, retrasos elevados y un rendimiento de red deficiente. El control de congestión busca evitar que la red se sobrecargue ajustando la tasa de transmisión de datos del emisor para que sea compatible con la capacidad de la red.

c. **Diferencie slow start de congestion-avoidance.**

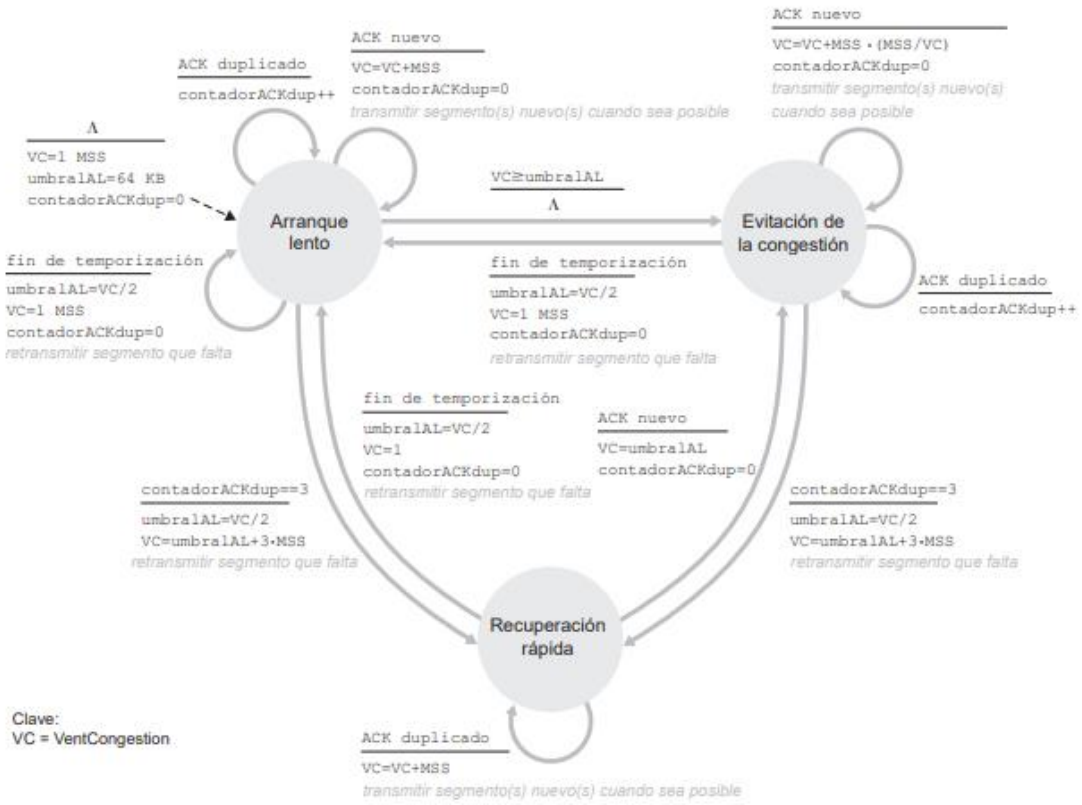
Arranque Lento (Slow Start):

- Inicio de la Conexión: Se utiliza al inicio de una conexión TCP.
- Tamaño de la Ventana de Congestión (VentCongestion): Inicializado con un valor pequeño (1 MSS, tamaño máximo de segmento).
- Crecimiento Exponencial: La ventana de congestión se duplica en cada periodo RTT.
- Finalización del Crecimiento Exponencial:
    - Al detectarse un suceso de pérdida (fin de temporización).
    - Cuando el valor de VentCongestion alcanza o sobrepasa el umbral de arranque lento (umbralAL).

Evitación de la Congestión (Congestion Avoidance):

- Transición desde Slow Start: Inicia cuando se detecta congestión y se sale del arranque lento.
- Tamaño de Ventana de Congestión (VentCongestion): Aproximadamente la mitad del valor cuando se detectó congestión por última vez.
- Crecimiento Lineal: Se incrementa en un MSS por RTT, más conservador que el crecimiento exponencial.
- Finalización del Crecimiento Lineal:
    - Al detectarse un suceso de pérdida (fin de temporización o tres ACK duplicados).
    - El valor de VentCongestion se fija en 1 MSS y se actualiza el umbral de arranque lento (umbralAL).
    - En el caso de pérdida detectada por tres ACK duplicados, se realiza un ajuste menos drástico del valor de VentCongestion y umbralAL, entrando en el estado de recuperación rápida.

https://www.youtube.com/watch?v=r9kbjAN2788 *(ver mas tarde)*



12. Para la captura dada, responder las siguientes preguntas.

a. ¿Cuántas comunicaciones (srcIP,srcPort,dstIP,dstPort) UDP hay en la captura?

a lot
En principio son 9, pero hay algunas que se tratan de la misma conversación, así que son 6.

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
10.0.30.10	8003	10.0.2.10	0	63	2,646	0	0	63	2,646	0.000000	4.7099	0	4,494
10.0.2.10	9000	10.0.3.10	13	1	46	1	46	0	0	20.872511	0.0000	—	—
10.0.2.10	9004	10.0.3.10	13	1	46	1	46	0	0	29.228498	0.0000	—	—
10.0.2.10	9004	10.0.3.10	4555	1	46	1	46	0	0	43.515947	0.0000	—	—
10.0.3.10	9045	10.0.2.10	9004	4	189	2	96	2	93	59.092837	7.6361	100	97
1.1.1.1	9045	10.0.2.10	9004	630	30k	0	0	630	30k	85.343725	17.8203	0	13k
10.0.2.10	53300	10.0.4.10	9045	1	46	1	46	0	0	112.609197	0.0000	—	—
10.0.2.10	59053	10.0.4.10	8003	5	235	3	139	2	96	118.382957	8.7621	126	87
10.0.2.10	8003	10.0.4.10	8003	2,320	2,473k	2,320	2,473k	0	0	169.166152	0.3529	56M	0

Fuente	Destino
10.0.2.10:0	10.0.30.10:8003
10.0.2.10:9004	10.0.3.10:9045
10.0.2.10:9004	1.1.1.1:9045
10.0.2.10:53300	10.0.4.10:9045
10.0.2.10:59053	10.0.4.10:8003
10.0.2.10:8003	10.0.4.10:8003

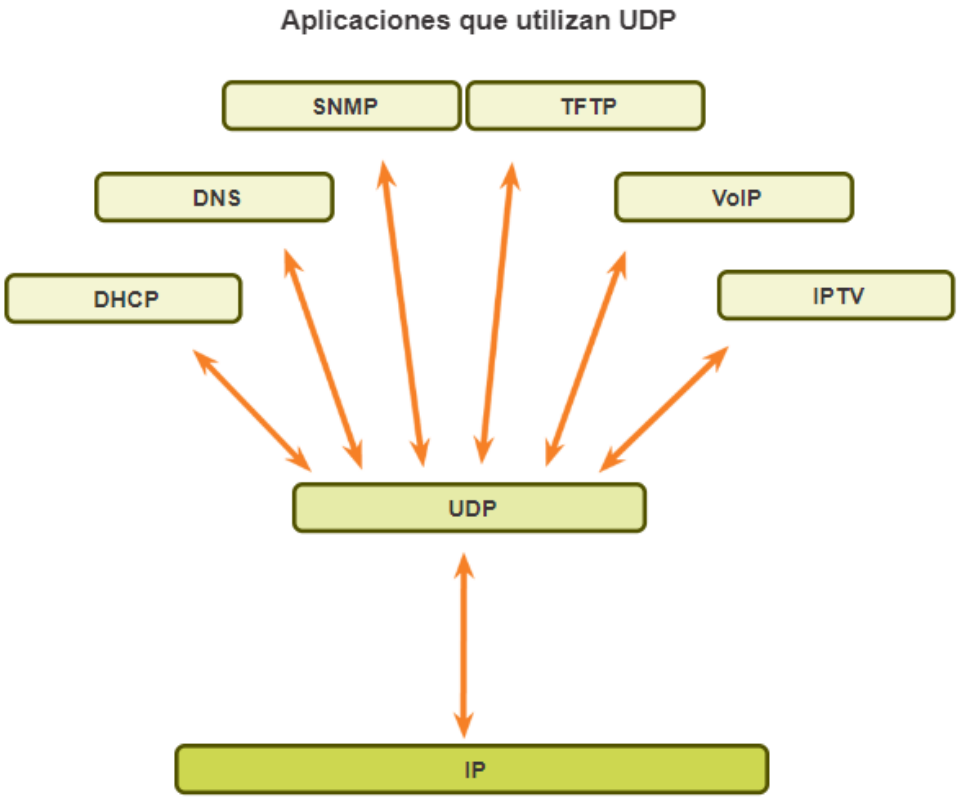
b. ¿Cómo se podrían identificar las exitosas de las que no lo son?

Se puede determinar con los mensajes ICMP

c. **¿UDP sigue el modelo cliente/servidor?**

Como no se establece una conexión, no sigue ningún modelo en particular, no tiene una estructura interna para definir roles específicos de cliente o servidor. Aun así, la mayoría de las aplicaciones que utilizan UDP suelen adoptar un modelo cliente/servidor según las necesidades del servicio que están proporcionando.

d. **¿Qué servicios o aplicaciones suelen utilizar este protocolo?**



e. **¿Qué hace el protocolo UDP en relación al control de errores?**

Lo único que ofrece es el checksum. Si detecta un error usando el checksum UDP no entrega el datagrama a la aplicación.

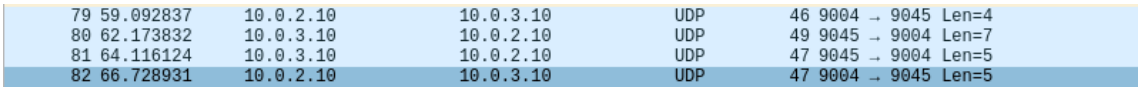
f. **Con respecto a los puertos vistos en las capturas, ¿observa algo particular que lo diferencie de TCP?**

En UDP el puerto origen puede ser 0 si no necesita una respuesta, puede ser simplemente un envio.

g. **Dada la primera comunicación en la cual se ven datos en ambos sentidos (identificar el primer datagrama):**

i. **¿Quién envía el primer datagrama (srcIP,srcPort)?**

10.0.2.10:9004



79	59.092837	10.0.2.10	10.0.3.10	UDP	46 9004 → 9045 Len=4
80	62.173832	10.0.3.10	10.0.2.10	UDP	49 9045 → 9004 Len=7
81	64.116124	10.0.3.10	10.0.2.10	UDP	47 9045 → 9004 Len=5
82	66.728931	10.0.2.10	10.0.3.10	UDP	47 9004 → 9045 Len=5

ii. **¿Cuantos datos se envían en un sentido y en el otro?**

12 bytes de 10.0.3.10:9045 a 10.0.2.10:9004
9 bytes de 10.0.2.10:9004 a 10.0.3.10:9045

h. **¿Se puede calcular un RTT?**

Con UDP no es posible calcular un RTT, ya que es un protocolo sin conexión y no tiene estados, por lo tanto no existen los ACK y estos son necesarios para el cálculo exacto de RTT. Nada te garantiza que el destino te responda y te responda apenas recibe el mensaje. Aun así, es posible estimar el RTT mediante técnicas externa o implementaciones específicas en la capa de aplicación.

Programación de sockets

Resuelva los siguientes ejercicios utilizando el lenguaje de programación que prefiera (por simpleza, se recomiendan Python o Ruby).

13. **Desarrolle un cliente y un servidor, donde el cliente envíe un mensaje al servidor y este último imprima en pantalla el contenido del mismo.**

    a. **Utilizando UDP.**

    https://wiki.python.org/moin/UdpCommunication

Sending

```python
import socket

UDP_IP = "127.0.0.1" # localhost
UDP_PORT = 5005 # non-privileged ports > 1023
MESSAGE = b"Long Live Taylor Swift!" # Must be bytes

print("UDP target IP: %s" % UDP_IP)
print("UDP target port: %s" % UDP_PORT)
print("message: %s" % MESSAGE.decode('utf-8'))

sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))
```

```
agusr@DESKTOP-E5THSHN MINGW64 /a/Agus/Escritorio/Facultad/3ER AÑO/2do Semestre/Redes/Practica/Practica 6
$ python3 senderUDP.py
UDP target IP: 127.0.0.1
UDP target port: 5005
message: Long Live Taylor Swift!
```

Receiving

```python
import socket

UDP_IP = "127.0.0.1" # localhost
UDP_PORT = 5005 # non-privileged ports > 1023

sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP
sock.bind((UDP_IP, UDP_PORT))

while True:
    data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
    print("received message: %s" % data.decode('utf-8'))
```

```
agusr@DESKTOP-E5THSHN MINGW64 /a/Agus/Escritorio/Facultad/3ER AÑO/2do Semestre/Redes/Practica/Practica 6
$ python3 receiverUDP.py
received message: Long Live Taylor Swift!
```

b. Utilizando TCP.

Client

```python
#!/usr/bin/env python

import socket


TCP_IP = "127.0.0.1"  # localhost
TCP_PORT = 5005
BUFFER_SIZE = 1024
MESSAGE = b"Long Live Taylor Swift!"  # Must be bytes

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
s.send(MESSAGE)
data = s.recv(BUFFER_SIZE)
s.close()

print("received data: %s" % data.decode('utf-8'))
```

Server

```python
#!/usr/bin/env python

import socket


TCP_IP = '127.0.0.1'
TCP_PORT = 5005
BUFFER_SIZE = 1024
```

```python
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

conn, addr = s.accept()
print ('Connection address:', addr)
while 1:
    data = conn.recv(BUFFER_SIZE)
    if not data: break
    print ("received data: %s" % data.decode('utf-8'))
    conn.send(data)  # echo
conn.close()
```

```
agusr@DESKTOP-E5TH5HN MINGW64 /a/Agus/Escritorio/Facultad/3ER AÑO/2do Semestre/Redes/Practica/Practica 6
$ python3 serverTCP.py
Connection address: ('127.0.0.1', 54651)
received data: Long Live Taylor Swift!
```

```
agusr@DESKTOP-E5TH5HN MINGW64 /a/Agus/Escritorio/Facultad/3ER AÑO/2do Semestre/Redes/Practica/Practica 6
$ python clientTCP.py
received data: Long Live Taylor Swift!
```

https://wiki.python.org/moin/TcpCommunication

14. **Compare ambas implementaciones. ¿Qué diferencia nota entre la implementación de cada una? ¿Cuál le parece más simple?**

    UDP es mucho mas sencillo, ya que no se tiene que establecer ninguna conexión, simplemente se tienen que enviar los datos.

Ejercicios de parcial

15. **Dada la salida que se muestra en la imagen, responda los ítems debajo.**

```
Netid  State      Local Address:Port      Peer Address:Port
udp    UNCONN               *:68                   *:*      (("dhclient",671,5))
udp    UNCONN               *:123                  *:*      (("ntpd",2138,16))
udp    UNCONN              :::123                 :::*      (("ntpd",2138,17))
tcp    LISTEN               *:80                   *:*      (("nginx",23653,19),("nginx",23652,19))
tcp    LISTEN               *:22                   *:*      (("sshd",1151,3))
tcp    LISTEN        127.0.0.1:25                   *:*      (("master",11457,12))
tcp    LISTEN               *:443                  *:*      (("nginx",23653,20),("nginx",23652,20))
tcp    LISTEN               *:3306                 *:*      (("mysqld",4556,13))
tcp    ESTAB         127.0.0.1:3306         127.0.0.1:34338  (("mysqld",4556,14))
tcp    TIME-WAIT  10.100.25.135:443     43.226.162.110:29148
tcp    ESTAB         127.0.0.1:48717        127.0.0.1:3306   (("ruby",28615,10))
tcp    ESTAB         127.0.0.1:3306         127.0.0.1:48717  (("mysqld",4556,17))
tcp    ESTAB         127.0.0.1:34338        127.0.0.1:3306   (("ruby",28610,9))
tcp    ESTAB      10.100.25.135:22     200.100.120.210:61576  (("sshd",13756,3),("sshd",13654,3))
tcp    LISTEN              :::22                  :::*      (("sshd",1151,4))
tcp    LISTEN              :1:25                  :::*      (("master",11457,13))
```

- **Suponga que ejecuta los siguientes comandos desde un host con la IP 10.100.25.90. Responda qué devuelve la ejecución de los siguientes comandos y, en caso que corresponda, especifique los flags.**

a. **hping3 -p 3306 –udp 10.100.25.135**

   ICMP Port Unreachable ya que el puerto no esta escuchando para UDP

b. **hping3 -S -p 25 10.100.25.135**

Va a devolver flag RST/ACK ya que no hay ningún proceso en estado LISTEN en ese puerto

c. **hping3 -S -p 22 10.100.25.135**

   Va a devolver flag SYN/ACK ya que hay un proceso en estado LISTEN para cualquiera en ese puerto.

d. **hping3 -S -p 110 10.100.25.135**

   Va a devolver flag RST/ACK ya que no hay ningún proceso en estado LISTEN en ese puerto

- **¿Cuántas conexiones distintas hay establecidas? Justifique**

  3

  1. 127.0.0.1:3306 con 127.0.0.1:34338
  2. 127.0.0.1:48717 con 127.0.0.1:3306
  3. 10.100.25.135:22 con 200.100.120.210:61576

  Si bien se ven 5, esas 2 que faltan son
  127.0.0.1:34338 con 127.0.0.1:3306
  127.0.0.1:3306 con 127.0.0.1:48717
  que son 1 y 2 pero con Local Address y Peer Address al revés ya que el comando presenta ambas para mostrar el flujo bidireccional de la comunicación.

16. Complete en la columna Orden, el orden de aparición de los paquetes representados en cada fila.

Host A				Host B			Orden
Seq	ACK	Len		Seq	ACK	Len	
100	2421	0	->				4
308	2821	0	->				13
			<-	1419	100	1002	3
156	2780	64	->				7
220	2780	47	->				9
			<-	2821	308	1418	14
			<-	2780	220	0	8
			<-	1	100	1418	**1**
100	2780	56	->				6
			<-	2780	308	0	11
267	2780	41	->				10
			<-	2780	308	41	12
			<-	2421	100	359	5
100	1419	0	->				2