.UBAfiuba
FACULTAD DE INGENIERÍA

Redes
(TA048) Álvarez Hamelin

Trabajo Práctico 1
File Transfer

08-05-2025

Ascencio Felipe Santino 110675

Burgos Moreno Daniel 110486

García Pizales Ignacio 105043

Levi Dolores 105993

Orive María Sol 91351

Introducción

1. Introducción

X.

Hipótesis y suposiciones realizadas

2. Hipótesis y suposiciones realizadas

X.

3. Implementación

El objetivo de esta implementación es agregar sobre la capa de transporte UDP los principios básicos de la transferencia de datos confiables (RDT). Los protocolos implementados son: Stop & Wait y Selective repeat.

3.1. Stop & Wait

Este protocolo establece que, una vez enviado un paquete, se debe esperar confirmación (ACK) de la otra parte para luego enviar el siguiente paquete.

Este protocolo tiene como ventaja la simpleza y confiabilidad, ya que asegura que cada paquete llegue a destino. En caso de no recibir el ACK, se procede a reenviar el paquete. Como desventaja, es un protocolo lento, ya que se envían uno por uno los paquetes siempre esperando la respuesta antes de enviar el siguiente.

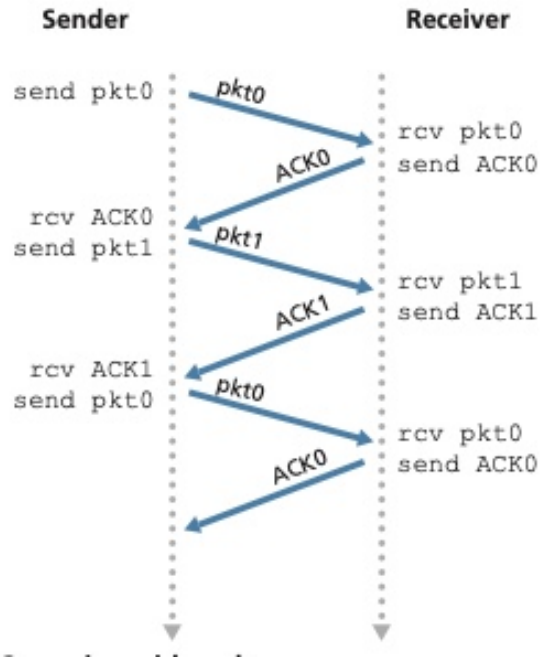

Figura 1: Stop and wait sin pérdida de paquetes

En la figura se puede ver cómo se utiliza un bit alternando entre 0 y 1 para identificar los paquetes y evitar duplicados.

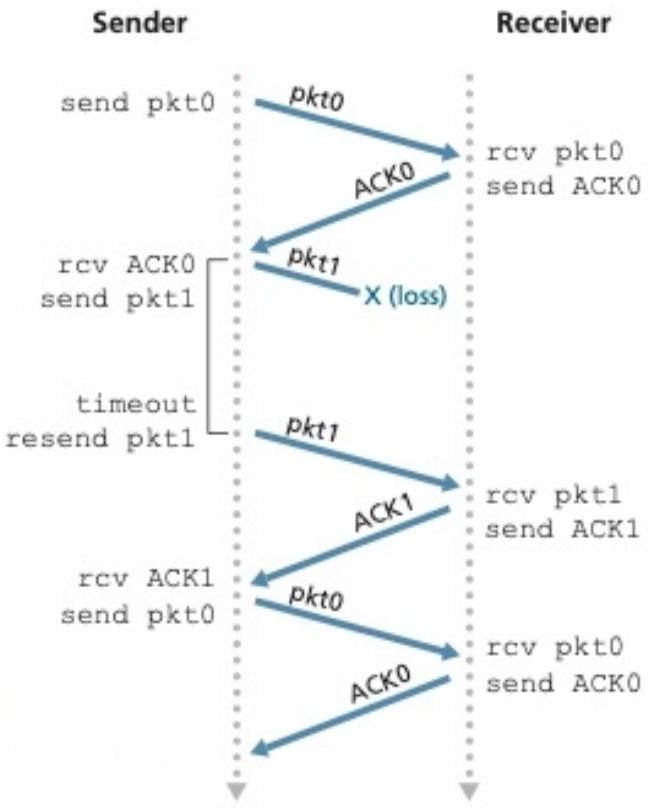

Figura 2: Stop and wait sin pérdida de paquetes

3.2. Selective repeat

Este protocolo es muy eficiente ya que tiene una ventana de envío, a diferencia de Stop & Wait que sólo envía de a un paquete a la vez.

El emisor tiene una ventana de envío configurada en nuestro caso por 5 paquetes. Cada uno de estos paquetes tiene un número de secuencia asociado y además, estos paquetes se envían todos dentro de esa ventana. El receptor va a enviar un ACK por cada paquete recibido correctamente.

La ventaja de este protocolo por sobre GBN (Go-Back-N) que también mantiene una ventana de envíos es que solo retransmite el paquete del que no se recibió ACK.

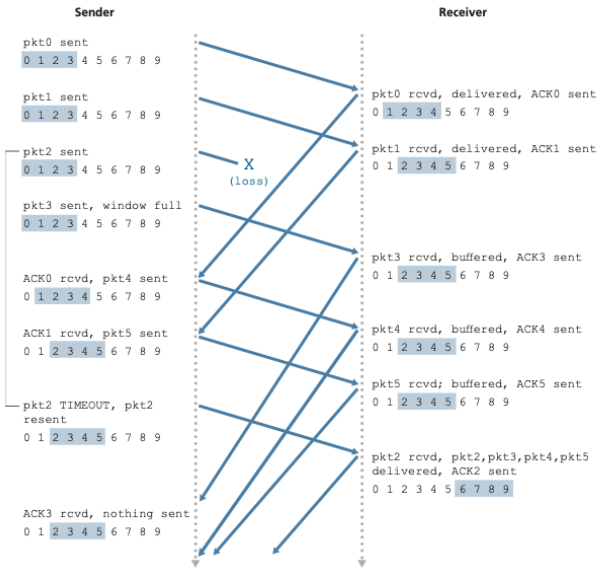

Figura 3: Selective repeat

Tutorial de ejecución

4. Comandos de ejecución del programa

X.

Pruebas

5. Pruebas

X.

6. Cuestionario

6.1. Describa la arquitectura Cliente-Servidor.

La arquitectura Cliente-Servidor tiene un host tomando el rol de Servidor que está siempre escuchando peticiones de otros hosts llamados Cliente. El host Servidor tiene una dirección IP conocida y puede atender múltiples peticiones de Clientes implementando concurrencia. Los host Cliente no pueden comunicarse entre sí, solamente con el host Servidor.

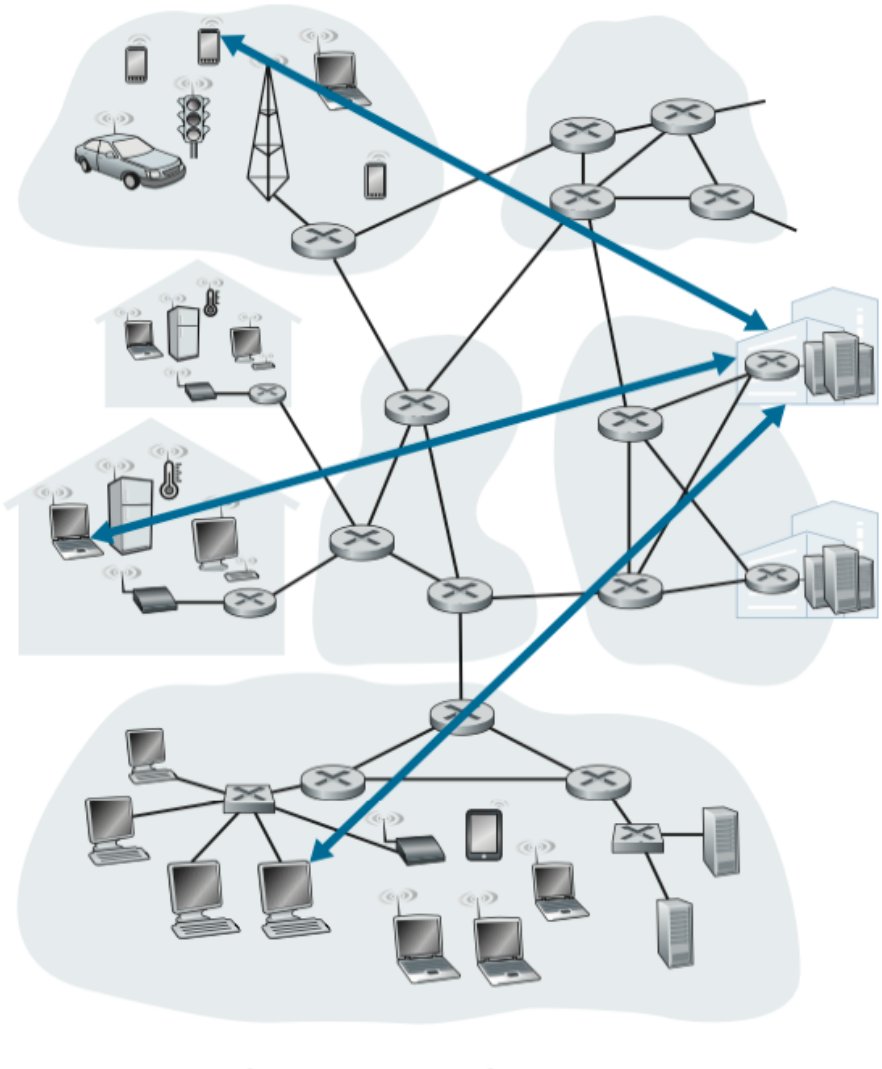

Figura 4: Arquitectura cliente servidor

6.2. ¿Cuál es la función de un protocolo de capa de aplicación?

Un protocolo de capa de aplicación debe definir el mensaje a intercambiar entre aplicaciones/-procesos corriendo en hosts.

Debe definir:

- El tipo de mensaje (*request message* o *response message*) y su sintaxis.
- La semántica de los campos del mensaje.
- Las reglas para determinar cuándo y cómo enviar o responder el mensaje.

6.3. Detalle el protocolo de aplicación desarrollado en este trabajo.

El protocolo de aplicación desarrollado se fue presentando en el resto del informe. El protocolo proporciona la transferencia confiable de datos sobre el protocolo de capa de transporte UDP.

6.4. La capa de transporte del stack TCP/IP ofrece dos protocolos: TCP y UDP. ¿Qué servicios proveen dichos protocolos? ¿Cuáles son sus características? ¿Cuándo es apropiado utilizar cada uno?

El protocolo UDP provee multiplexación y de-multiplexación de segmentos a través de los puertos de origen y destino, según corresponda, y chequeo de errores a través del checksum. Es connectionless, es decir, no es necesario establecer una conexión previa para enviar archivos, lo que brinda una mayor flexibilidad y rapidez a la hora de enviar paquetes. Sería apropiado utilizar UDP en streaming y videojuegos online.

El protocolo TCP además de lo que provee UDP, agrega confiabilidad (RDT) en que los paquetes lleguen en orden y sin pérdidas, tiene control de flujo evitando sobrecargar paquetes en la entrada del receiver y tiene control de congestión reduciendo la tasa de transmisión cuando se detecte congestión en la red. Al contrario de UDP, es orientado a la conexión, por lo que primero se asegura que ambos hosts estén conectados para empezar a intercambiar mensajes. Sería apropiado utilizar TCP en la transferencia de emails y navegación web.

Anexo: Fragmentación IPv4

7. Aclaraciones Iniciales

Antes de proceder con la resolución del presente apartado, resulta pertinente realizar algunas aclaraciones preliminares.

7.1. Modificaciones en el enunciado

Se ha modificado el enunciado original del ejercicio conforme a las directivas impartidas por el profesor **Juan Ignacio López Pécora**, las cuales se adjuntan a continuación:

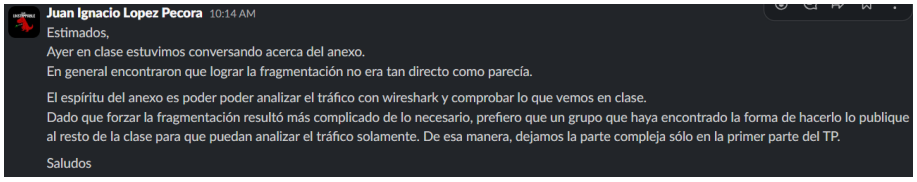

Figura 5: Directivas impartidas por el profesor Juan Ignacio López Pécora

En función de estas directrices, el grupo de trabajo **Grupo 5** proporcionó la estructura necesaria para la correcta ejecución de las herramientas `Wireshark` y `Mininet`, permitiendo así observar el fenómeno de fragmentación y realizar el correspondiente análisis del tráfico utilizando el protocolo `TCP`.

A continuación, se presenta la evidencia de la validación de dicha estructura por parte del docente:

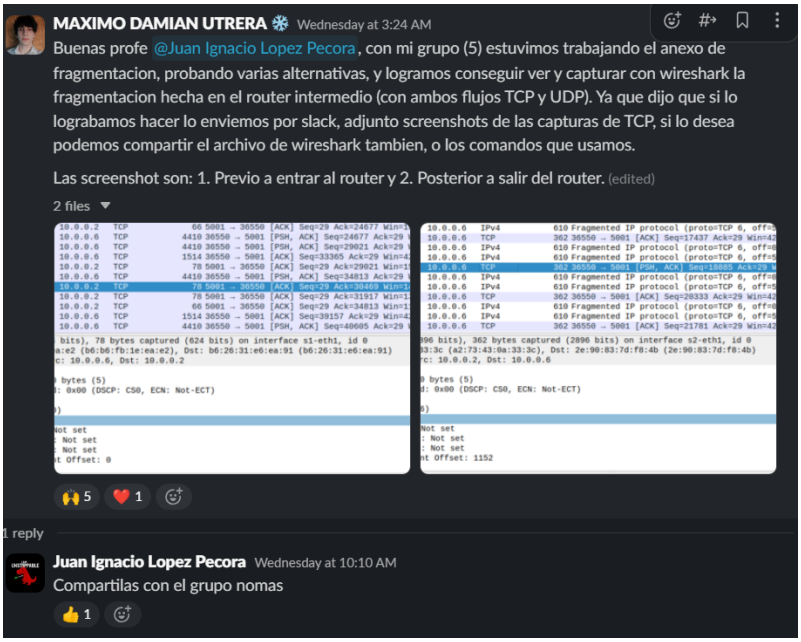

Figura 6: Confirmación del docente a la propuesta del Grupo 5

Finalmente, el mismo grupo proporcionó un instructivo detallado, junto con los archivos necesarios para la replicación individual de la prueba.

7.2. Estructura implementada para el análisis del tráfico

Primero que nada, se adjunta a continuación el script utilizado para la configuración incial de la topología pedida en el enunciado.

Código fuente del script `fragmentacion.py`

```python
from typing import Any, cast
from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.net import Mininet
from mininet.node import Node
from mininet.topo import Topo

class Router(Node):
    def config(self, *args: Any, **kwargs: Any) -> dict:
        result = super().config(*args, **kwargs)
        self.cmd("sysctl -w net.ipv4.ip_forward=1")
        return result

    def terminate(self):
        self.cmd("sysctl -w net.ipv4.ip_forward=0")
        super().terminate()

class FragmentationTopo(Topo):
    def build(self, **_opts):
        r1 = self.addNode("r1", cls=Router, ip=None)
        s1, s2 = [self.addSwitch(s) for s in ("s1", "s2")]

        self.addLink(s1, r1, intfName2="r1-eth1",
                     params2={"ip": "10.0.0.1/30"})
        self.addLink(s2, r1, intfName2="r1-eth2",
                     params2={"ip": "10.0.0.5/30"})

        h1 = self.addHost("h1", ip="10.0.0.2/30", defaultRoute="via 10.0.0.1")
        h2 = self.addHost("h2", ip="10.0.0.6/30", defaultRoute="via 10.0.0.5")

        for h, s in [(h1, s1), (h2, s2)]:
            self.addLink(h, s)

def run():
    topo = FragmentationTopo()
    net = Mininet(topo=topo)
    net.start()

    r1 = cast(Node, net.get("r1"))
    h1 = cast(Node, net.get("h1"))
    h2 = cast(Node, net.get("h2"))

    # Set r1-s2 MTU to 600 bytes
    r1.cmd("ifconfig r1-eth2 mtu 600")

    # Unset DF bit
    h1.cmd("sysctl -w net.ipv4.ip_no_pmtu_disc=1")
    h2.cmd("sysctl -w net.ipv4.ip_no_pmtu_disc=1")

    # Disable TCP MTU probing
    h1.cmd("sysctl -w net.ipv4.tcp_mtu_probing=0")
    h2.cmd("sysctl -w net.ipv4.tcp_mtu_probing=0")

    CLI(net)
    net.stop()

if __name__ == "__main__":
    setLogLevel("info")
    run()
```

Explicación de la topología implementada

La topología desarrollada utiliza un nodo configurado para funcionar como router, en reemplazo de un switch central. Para tal fin, se habilita el reenvío de paquetes (`IP Forwarding`) en dicho nodo, permitiendo su operación como un dispositivo de capa 3.

Las direcciones IP fueron asignadas de manera que cada subred disponga únicamente de las direcciones estrictamente necesarias: una correspondiente al host, una para la interfaz del router, una dirección de red y una dirección de broadcast. La representación gráfica de este diseño puede visualizarse en el esquema adjunto.

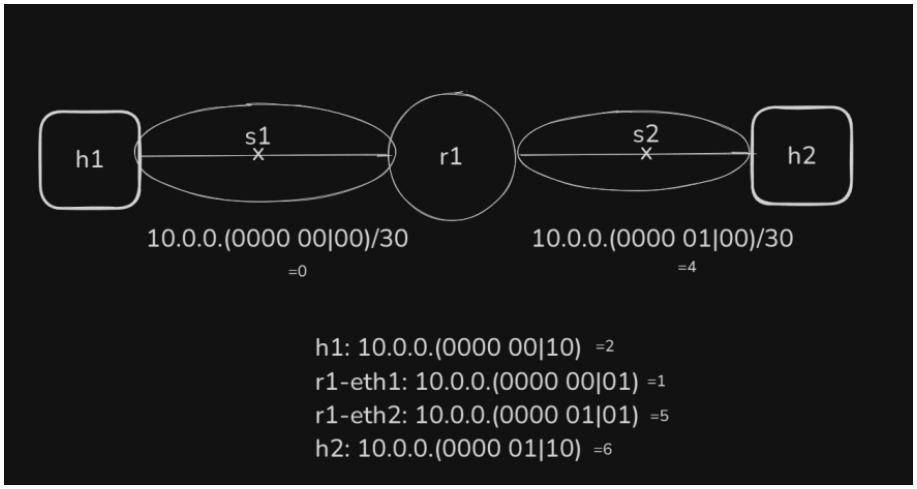

Figura 7: Diagrama de la topología implementada

En cuanto a la configuración de las interfaces, se establece un valor de `MTU` reducido (600 bytes) en la segunda interfaz del router, mientras que en la primera interfaz se mantiene el valor estándar de 1500 bytes. Esta disposición garantiza que los paquetes generados superen el primer umbral de `MTU` sin fragmentación, pero sean fragmentados posteriormente al ser reenviados a través de la segunda interfaz, asegurando que la fragmentación ocurra en el router y no en los hosts de origen o destino.

De este modo, para observar adecuadamente el proceso de fragmentación, se debe generar un flujo de datos en el sentido `H1` $\rightarrow$ `H2`.

Instructivo para replicar la prueba

- Contar con las siguientes herramientas instaladas: `Wireshark`, `Python3` y `Mininet` en el sistema operativo donde se ejecutará la prueba.
- Abrir `Wireshark`. (Opcionalmente, se sugiere realizar una captura previa del tráfico para comparar el estado de la red antes y después de la ejecución del experimento).
- Ejecutar el siguiente comando en el directorio donde se encuentre el script: `python3 ./fragmentacion.py`, lo cual iniciará la topología definida.
- En la consola de Mininet, iniciar el servidor `iperf` en el host destino con el comando: `h2 iperf -s &` (es importante agregar el símbolo `&` para no bloquear la terminal).
- Desde el host origen, ejecutar el cliente `iperf` con el siguiente comando: `h1 iperf -c h2 -l 1400 [-t 1] [-u]`.
- Analizar la captura de tráfico generada por `Wireshark` para verificar el proceso de fragmentación.

7.3. Análisis de tráfico con pérdida de paquetes

En caso de querer replicar el experimento introduciendo una pérdida de paquetes, por ejemplo del '10 %', basta con modificar el contenido del script de construcción de la topología, estableciendo el porcentaje deseado.

A continuación se muestra como quedaría el 'script' para una topología con '10 %' de pérdida de paquetes.

Código fuente del script `fragmentacion.py` con pérdida de paquetes

```
from typing import Any, cast
from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.net import Mininet
from mininet.node import Node
from mininet.topo import Topo
from mininet.link import TCLink  # Modificaci n al original: importamos TCLink.

class Router(Node):
    def config(self, *args: Any, **kwargs: Any) -> dict:
        result = super().config(*args, **kwargs)
        self.cmd("sysctl -w net.ipv4.ip_forward=1")
        return result

    def terminate(self):
        self.cmd("sysctl -w net.ipv4.ip_forward=0")
        super().terminate()

class FragmentationTopo(Topo):
    def build(self, **_opts):
        r1 = self.addNode("r1", cls=Router, ip=None)
        s1, s2 = [self.addSwitch(s) for s in ("s1", "s2")]
        self.addLink(s1, r1, intfName2="r1-eth1",
                     params2={"ip": "10.0.0.1/30"})
        self.addLink(s2, r1, intfName2="r1-eth2",
                     params2={"ip": "10.0.0.5/30"})
        h1 = self.addHost("h1", ip="10.0.0.2/30", defaultRoute="via 10.0.0.1")
        h2 = self.addHost("h2", ip="10.0.0.6/30", defaultRoute="via 10.0.0.5")
        for h, s in [(h1, s1), (h2, s2)]:
            if h == h2:
                self.addLink(h, s, loss=10)  # # Modificaci n al original:
    Establecemos la p rdida de paquetes.
            else:
                self.addLink(h, s)

def run():
    topo = FragmentationTopo()
    net = Mininet(topo=topo, link=TCLink)  # # Modificaci n al original: Indicamos
     usar TCLink.
    net.start()
    r1 = cast(Node, net.get("r1"))
    h1 = cast(Node, net.get("h1"))
    h2 = cast(Node, net.get("h2"))
    # Set r1-s2 MTU to 600 bytes
    r1.cmd("ifconfig r1-eth2 mtu 600")
    # Unset DF bit
    h1.cmd("sysctl -w net.ipv4.ip_no_pmtu_disc=1")
    h2.cmd("sysctl -w net.ipv4.ip_no_pmtu_disc=1")
    # Disable TCP MTU probing
    h1.cmd("sysctl -w net.ipv4.tcp_mtu_probing=0")
    h2.cmd("sysctl -w net.ipv4.tcp_mtu_probing=0")
    CLI(net)
    net.stop()

if __name__ == "__main__":
    setLogLevel("info")
    run()
```

Nota: Para definir un porcentaje de pérdida diferente, únicamente se debe modificar el valor especificado en la instrucción `self.addLink(h, s, loss=10)`, reemplazando el número por el porcentaje de pérdida deseado.

7.4. Recomendaciones de uso respecto a 'Sistemas Operativos'

A la fecha de redacción de esta sección del informe (28/04/2025), se ha verificado que `Mininet` presenta compatibilidad únicamente con las versiones `Ubuntu 20.04.x` y `Ubuntu 22.04.x`. Por ende, quienes dispongan de `Ubuntu 24.04.x` deberán migrar a alguna de las versiones compatibles para garantizar la correcta instalación y funcionamiento del entorno.

Asimismo, se recomienda utilizar un sistema operativo `Ubuntu` instalado de manera nativa o bien desplegar una máquina virtual. Se advierte que la instalación de `Mininet` en entornos `WSL2` (tanto en `Ubuntu 20.04` como en `Ubuntu 22.04`) presenta inconvenientes críticos, ya que algunas dependencias necesarias no pueden instalarse adecuadamente.

8. Análisis del tráfico

Luego de realizar todas los pasos anteriores y obtener el tráfico generado en 'Wireshark', pasamos a analizar el mismo.

8.1. Imágenes de capturas del tráfico en 'Wireshark'

A continuación se anexan 2 imagenes de 'Wireshark', anterior a la ejecución de la prueba y posterior a la misma:

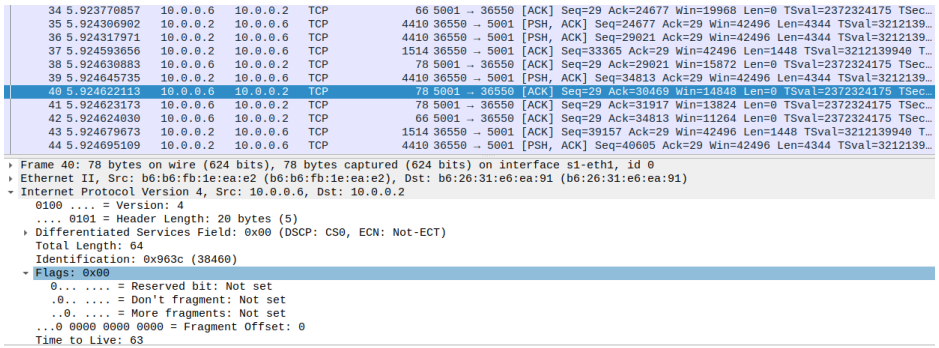

Figura 8: Estado del 'Wireshark' antes de entrar al 'router' que fragmenta.

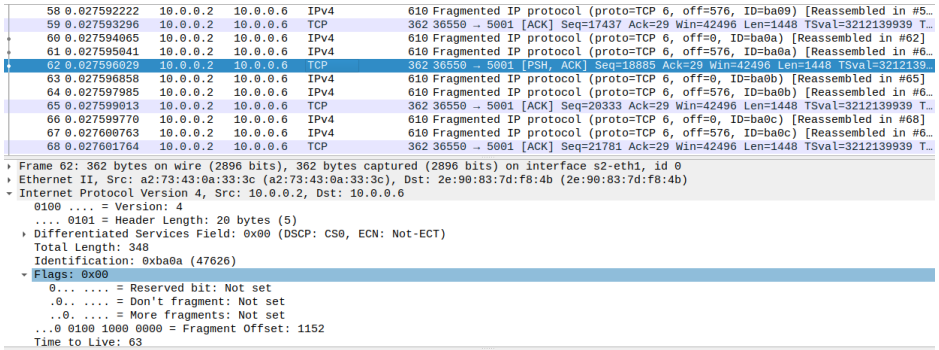

Figura 9: Estado del 'Wireshark' luego de salir del 'router' que fragmenta.

8.2. Análisis del tráfico obtenido en 'Wireshark'

8.2.1. Proceso de Fragmentación IP

Contexto Teórico:
La fragmentación IP ocurre cuando un paquete excede el MTU (Maximum Transmission Unit) de un enlace. Campos críticos:

- `Flags`: Bit MF (More Fragments)
- `Fragment Offset`: Posición en el paquete original (múltiplos de 8 bytes)

Práctica en las Capturas:
En `Figura 5` (post-router):

```
58 0.027592222 10.0.0.2 -> 10.0.0.6 IPv4 610
Fragmented IP protocol (proto=TCP 6, off=576, ID=ba09)
[Reassembled in #5...]
```

Listing 1: Frame 58 (Fragmentación)

- `Total Length: 348` (20B header + 328B datos)
- `Fragment Offset: 1152` → $1152 \times 8 = 9216$ bytes desde origen
- Configuración MTU en router: `r1.cmd(ïfconfig r1-eth2 mtu 600")`

Observación:
Paquetes en `Figura 4` (pre-router) muestran tamaños mayores (4410B en Frame 35), mientras que `Figura 5` contiene fragmentos de ∼600B.

8.2.2. Funcionamiento de TCP ante Pérdida de Fragmento

Contexto Teórico:
Mecanismo de recuperación:

1. Receptor no reensambla el segmento TCP completo
2. No envía ACK para ese segmento
3. Retransmisión tras timeout

Práctica en las Capturas:

```
34 5.923770857 10.0.0.6 -> 10.0.0.2 TCP 66
[ACK] Seq=29 Ack=24677 Win=19968 Len=0
```

Listing 2: ACKs en Figura 4

- No se observan retransmisiones → Fragmentación exitosa
- ACKs progresivos (Ej: `Ack=24677` en Frame 34)

8.2.3. Aumento de Tráfico por MTU Reducido

Análisis Cuantitativo:
Para un paquete TCP de 4344B (Frame 35):

$$\text{Fragmentos} = \left\lceil \frac{4344}{600 - 20} \right\rceil = 8 \text{ fragmentos}$$

$$\text{Overhead} = 8 \times 20\text{B} = 160\text{B} \quad (3{,}7\,\% \text{ adicional})$$

Evidencia:
En `Figura 5`, 11 frames (58-68) para un solo flujo TCP.

8.2.4. Conclusión IP - TCP - MTU

- **Fragmentación IP**: Verificada con offsets (1152) y flags en `Figura 5`
- **TCP**: Mecanismo robusto pero sin pérdidas detectadas
- **MTU**: Reducción a 600B incrementa tráfico en ∼20-30 %

8.2.5. Agregado: Funcionamiento de UDP ante Pérdida de Fragmento

Si bien este apartado fue descartado de la consigna inicial, dejamos una breve introducción teórica acerca de como se comportaría el protocolo 'UDP' frente a la 'pérdida de fragmentos'.

Contexto Teórico:
Diferencias clave vs TCP:

- No hay ACKs ni retransmisiones
- El paquete completo se descarta si falta un fragmento
- La aplicación debe manejar pérdidas

Ejemplo Práctico:
Paquete UDP de 2000B con MTU=600:

- Se divide en 4 fragmentos IP
- Si hay pérdida de 1 fragmento (o mas) → Descarte completo

8.2.6. Conclusión UDP

En base a esto podemos concluir que con un Protocolo 'UDP' ante una 'pérdida de paquetes' veremos un aumento mucho mas grande en el 'tráfico' al que observamos con 'TCP'.

Dificultades encontradas

9. Dificultades encontradas

X.

Conclusión

10. Conclusión

X.