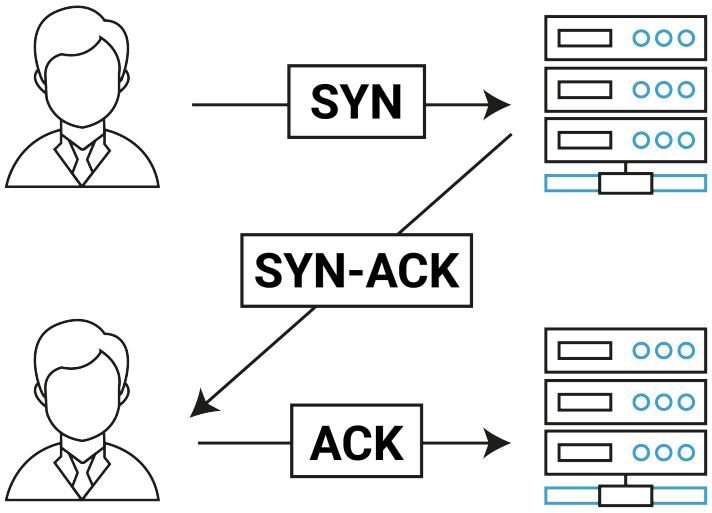
Tecnología Digital IV: Redes de Computadoras

Trabajo Práctico 1

Implementación de una Conexión Confiable sobre un Canal Inseguro Usando Scapy



Nicolas Boudourian, Benjamin Toledo

Segundo Semestre, 2024

Índice de contenidos

1 Introducción

En este trabajo práctico se implementa una conexión confiable entre un cliente y un servidor utilizando Python y la librería Scapy. El objetivo principal es garantizar que la comunicación entre ambos se mantenga íntegra, a pesar de las posibles fallas que ocurren en un canal inseguro simulado mediante la librería canalruidoso. Los problemas del canal incluyen la posibilidad de que los paquetes se pierdan, se corrompan o sufran retrasos.

La implementación sigue el protocolo de transmisión TCP, incluyendo el three-way handshake y el cierre correcto de la conexión, para garantizar la confiabilidad del canal. Además, se establecen mecanismos para la retransmisión de paquetes en caso de que los ACKs no lleguen dentro del tiempo de espera de 3 segundos.

2 Protocolo Implementado

2.1 Inicio de Conexión: Three-Way Handshake

Para iniciar una conexión confiable, el cliente sigue el protocolo three-way handshake de TCP, que incluye los siguientes pasos:

SYN: El cliente envía un paquete SYN al servidor para indicar que desea iniciar una conexión.

SYN-ACK: El servidor responde con un paquete SYN-ACK, confirmando la recepción del SYN y señalando que está dispuesto a establecer la conexión.

ACK: El cliente envía un paquete ACK para confirmar la recepción del SYN-ACK, completando así la conexión.

Durante este proceso, si alguno de los paquetes no es confirmado mediante el envío del ACK correspondiente en un plazo de 3 segundos, el cliente o el servidor retransmiten el paquete.

2.2 Cierre de Conexión: Protocolo de Terminación de TCP

El cierre de la conexión sigue el procedimiento habitual de terminación de TCP:

FIN: El servidor envía un paquete FIN para indicar que desea finalizar la conexión.

FIN-ACK: El cliente responde con un ACK y envía su propio FIN, indicando que ha recibido el mensaje de cierre y también desea cerrar la conexión.

ACK: El servidor envía el ACK final para confirmar el cierre.

Este proceso garantiza que ambas partes cierren la conexión de manera ordenada, sin dejar segmentos pendientes.

2.3 Retransmisiones y Timeout

El canal inseguro que utilizamos puede generar condiciones desfavorables para la transmisión de paquetes, como pérdida, corrupción o retraso. Para abordar esto, se implementó una política de retransmisión con un timeout de 3 segundos (arbitrario). Esto significa que, si un ACK no es recibido dentro de este tiempo, se retransmite el paquete. En la fase de experimentación, obtendremos conclusiones respecto al valor óptimo para el tiempo de "timeout".

La función ‘envio-paquetes-inseguro‘ de la librería ‘canalruidoso‘ es la encargada de simular estas condiciones adversas, lo cual nos obliga a implementar un sistema de retransmisiones. Este enfoque asegura que, a pesar de las fallas en la transmisión, los paquetes eventualmente lleguen a su destino sin errores.

2.4 Definición de Wrapper Send

Para tener un control de estado de cada paquete enviado por el canal inseguro, se optó por realizar un *wrapper* que utilice internamente la función `envio_paquetes_inseguro`. Al utilizar esta función, se verifica el resultado, ya sea una pérdida, un retraso (delay) o corrupción (en caso de ocurrir). Luego de haber detectado el resultado, se contabiliza para futuras estadísticas y mediciones.

Para identificar si existe retraso, se implementó el contador:

```
initial_time = time.time()

# Llamar a la funci n envio_paquetes_inseguro
result = f.envio_paquetes_inseguro(pkt)

# Medir el tiempo despu s del env o para ver si hubo retraso
end_time = time.time()
elapsed = end_time - initial_time
```

Si el tiempo de envío fue mayor a 3 segundos, se considera que el paquete tiene un retraso, y se incrementa el tiempo total de retraso en la ejecución:

```
if elapsed > 3:
    stats['delayed_packets'] += 1
    stats['total_delay_time'] += (elapsed - 1)
    return
```

Para identificar si se envió o se perdió el paquete, se verifica si la función `envio_paquetes_inseguro` retornó 0:

```
if result == 0:
    stats['lost_packets'] += 1
    return
```

Y para identificar corrupción, verificamos el `checksum` del paquete:

```
if pkt[TCP].chksum == 0x1234:
    stats['corrupted_packets'] += 1
    return
```

Esta implementación del *wrapper* se ha diseñado para facilitar la recopilación de estadísticas durante la experimentación. Esto nos permitirá evaluar el rendimiento de nuestra implementación de conexión segura, que utiliza un canal inseguro, y así poder llegar a conclusiones informadas sobre su funcionamiento.

3 Experimentos y Resultados

En esta sección, se proponen experimentos variando parámetros, conjuntos de datos, algoritmos y/o lenguajes de programación para estudiar el comportamiento de las implementaciones realizadas en cada caso.

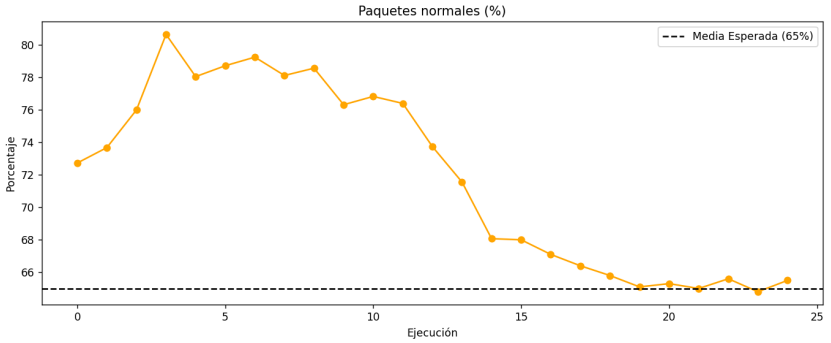
3.1 Cálculo del Porcentaje de Errores

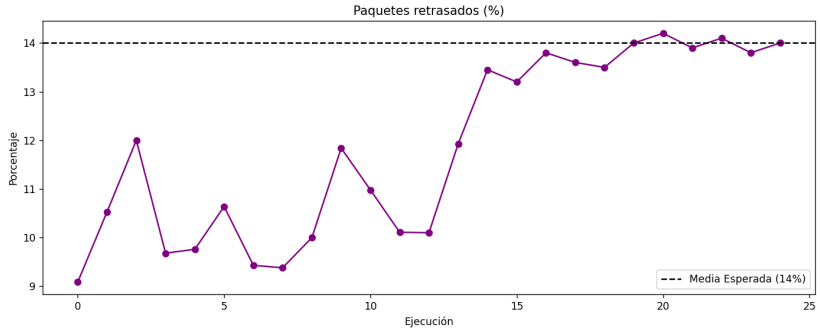
Para calcular el porcentaje de errores, ejecutaremos el código 25 veces, y en cada ejecución se enviarán al menos 6 mensajes. Esto nos da un mínimo de 150 mensajes, de forma que obtendremos una muestra representativa para luego generar gráficos generalizados.

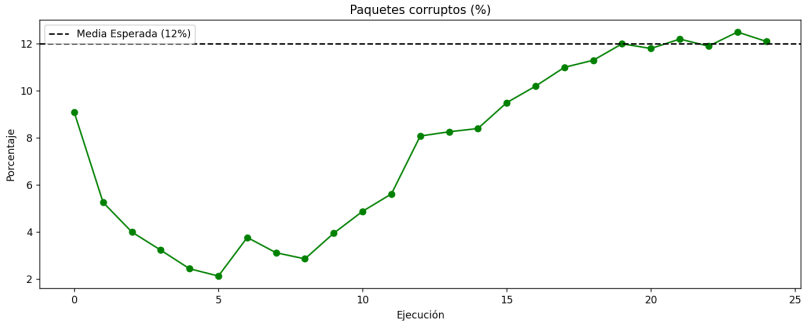
La hipótesis es que, a medida que aumentan las ejecuciones, los resultados acumulados deben aproximarse a los valores predeterminados para el "envío de paquetes en un canal inseguro". A continuación, se presentan dichos valores:

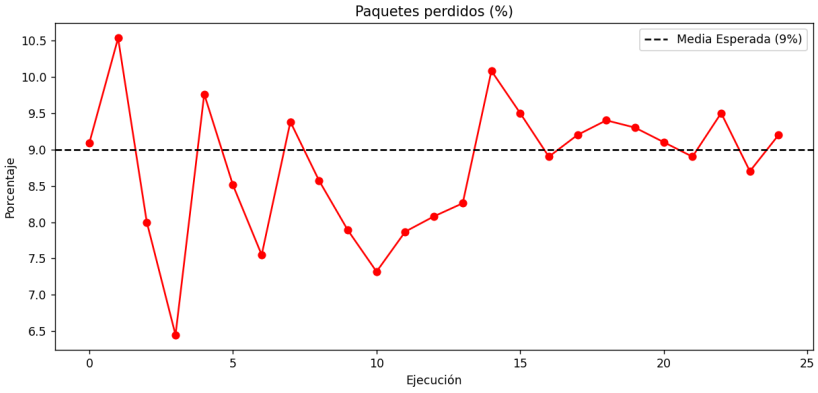
- Porcentaje delay = 14
- Porcentaje corrupcion = 12
- Porcentaje perdida = 9
- Porcentaje normal = 65

Después de 25 ejecuciones, obtuvimos un total de 200 mensajes en los cuales se registró el resultado de cada ejecución. A continuación, presentamos los gráficos de los resultados obtenidos:


Paquetes normales (%)
Media Esperada (65%)
Porcentaje
Ejecución


Paquetes retrasados (%)
Media Esperada (14%)
Porcentaje
Ejecución


Paquetes corruptos (%)
Media Esperada (12%)
Porcentaje
Ejecución


Paquetes perdidos (%)
Media Esperada (9%)
Porcentaje
Ejecución

4 Conclusión

El trabajo práctico realizado nos permitió comprender mejor el funcionamiento de la apertura y cierre de conexión entre hosts. Además, gracias al canal inseguro, pudimos implementar un algoritmo para garantizar la entrega de paquetes. Lo más interesante fue el análisis de la experimentación. A medida que la cantidad de paquetes aumenta, nos acercamos más al margen de error previsto por el "canal inseguro". Esto nos proporcionó un nuevo entendimiento y una teoría en la cual se puede predecir el comportamiento de un canal. A partir de esto, luego se podrían ajustar ciertos parámetros para hacer más eficiente la comunicación. Por ejemplo, si después de enviar 200 paquetes se observan ciertos valores de retransmisión y tiempos de pérdida, entonces podría ajustarse el timeout para que sea dinámico. También podría ser útil al elegir el protocolo de retransmisión.