Taller UDP Connection

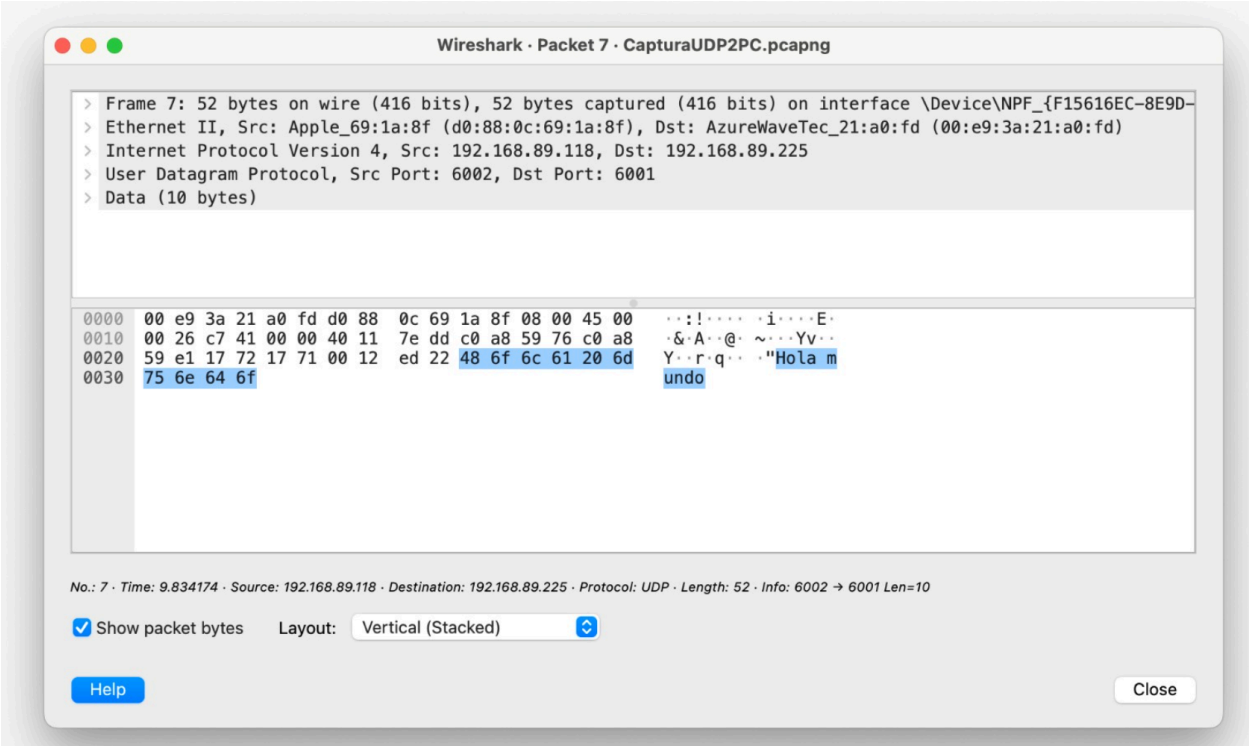
Grupo: Gabriel Escobar - Vanessa Sánchez

Solución del taller:

Código fuente del proyecto: https://github.com/Gab27x/UDP-assignment

Capturas de Wireshark de cada Peer el nombre de las capturas deber´a tener el formato PeerX.pcapng, con base a estas capturas deben responder a estas preguntas:

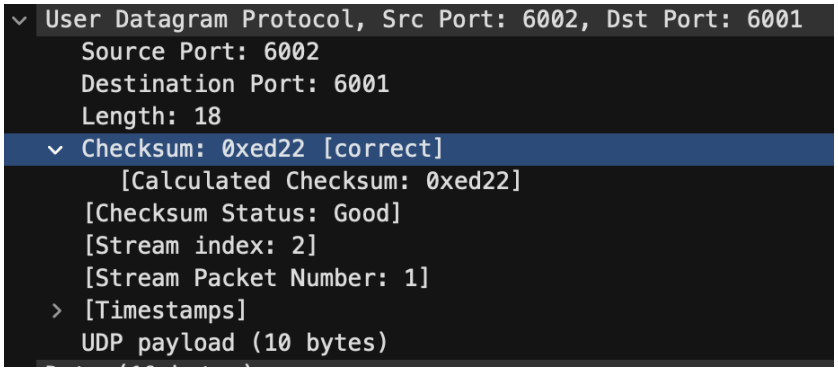
En el repositorio se encuentra la captura, esta es una foto del resultado:



1. ¿Es posible ver en la captura de Wireshark el contenido del mensaje enviado?

Como podemos evidenciar en la captura se puede ver que el contenido del mensaje era hola mundo, este es visible ya que el mensaje no se encuentra cifrado.

2. ¿Cuál es el checksum de la captura? ¿Explique/investiguen por qué este checksum?



   El checksum de la captura es **0xed22**.
   Este checksum se calcula sumando los datos del paquete en palabras de 16 bits, junto con un "pseudo-encabezado" que incluye partes del encabezado IP. Después de realizar dicha suma, se realiza el complemento a uno del resultado y se coloca en el campo del checksum correspondiente.

   Finalmente, cuando el paquete es recibido, el receptor realiza el mismo cálculo. Si el checksum resultante es todo unos (en binario), el paquete se considera válido, como sucede en este caso.

3. ¿Qué patrones de diseño/arquitectura aplicaría al desarrollo de un programa basado en red como este?

   Para realizar el programa se utilizó el patrón de diseño singleton para crear una clase UPDConnection para gestionar la lógica de escucha y de envío.

   Pero sabemos que se podría implementar el patrón Observer, el cual permitiría notificar de forma eficiente a múltiples componentes de la aplicación cuando ocurran eventos relevantes en la conexión UDP, como la recepción de nuevos datos o errores de red.

   También se podría implementar el patrón Decorator. Este patrón nos permitiría encapsular la lógica de cifrado y descifrado en objetos separados, los cuales podrían ser aplicados a los objetos que gestionan el envío y recepción de mensajes de manera dinámica, sin necesidad de modificar su código interno.

4. Modifique el código provisto de tal forma que: el hilo de recepción no ’muera’ una vez recibido el mensaje

De esta manera se cumpliria lo solicitado:

```java
@Override  new *
public synchronized void run() {
    DatagramPacket packet = new DatagramPacket(new byte[16], length: 16);
    System.out.println("Waiting...");

    while (true) {
        try {
            // Recibir la información
            this.socket.receive(packet);
            String msj = new String(packet.getData()).trim();
            System.out.println(msj);

            packet.setLength(16);
        } catch (IOException e) {

            e.printStackTrace();
        }
    }
}
```

No se encontrara en la implementación final ya que se quería posibilitar el responder el mensaje.

5. Modifique el código provisto de tal forma que la lógica de transmisión de paquetes quede en un hilo aparte

```java
public void sendDatagram(String msj,String ipDest, int portDest){  1 usage  Gab27x

    new Thread(() -> {
        try{
            socket = new DatagramSocket();

            InetAddress ipAddress = InetAddress.getByName(ipDest);
            DatagramPacket packet = new DatagramPacket(msj.getBytes(),msj.length(),ipAddress,portDest);
            socket.send(packet);


        }catch(SocketException | UnknownHostException e){
            e.printStackTrace();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

    }).start();


}
```

6. Investiguen qué modificaciones son necesarias para implementar este mismo sistema pero para la comunicación TCP en java.

   En primer lugar, para modificar la implementación del sistema de UDP a TCP, es necesario implementar un *ServerSocket* del lado de la lógica del servidor que gestione la escucha de mensajes y que deba aceptar la comunicación (método *accept()*). Además, la entrada y salida de los datos se manejaría por medio del *BufferedReader* y *BufferedWriter*. Y finalmente, en TCP se debe realizar la lógica de establecimiento de la conexión; mientras que en UDP se envía un datagrama sin establecer conexión alguna o garantizar la entrega del mensaje, en TCP el servidor debe escuchar en un puerto específico y aceptar conexiones mientras que el cliente se conecta a este puerto.

7. ¿Qué utilidades de codificación o seguridad agregarían al código?

   Para poder agregar seguridad a la implementación de UDP, existen diferentes alternativas que se pueden tomar.

   En primer lugar, se podría implementar un manejo más específico de las excepciones, que indiquen que fallos se estan presentando y además, validar las entradas (puerto de escucha, puerto de destino y dirección IP de destino).
   Por otro lado, se podría realizar un cifrado de los mensajes a enviar o recibir por medio de la generación de claves y cifrado en la parte de quien envía y posterior descifrado en el lado del receptor. Se pueden implementar algoritmos de cifrado como AES (Advanced Encryption Standard).

Finalmente, otras medidas de seguridad que se podrían tomar a nivel de código sería la implementación de HMACSHA256, un tipo de algoritmo hash con clave que se usa como código de autenticación de mensajes (este se puede usar para determinar si se ha alterado un mensaje enviado a través de un canal no seguro, siempre que el remitente y el receptor compartan una clave secreta) o con implementando una lógica de *logs*, que permitan registrar y monitorear las actividades del programa.