TALLER UDP JAVA SOCKET COMPUTACIÓN EN INTERNET I

DANIELA CASTAÑO MORENO - A00401805 SIMÓN GARCÍA ZULUAGA - A00371828

> UNIVERSIDAD ICESI CALI, VALLE DEL CAUCA 2025

CAPTURAS DE WIRESHARK

Para iniciar, es necesario indicar que nuestros Peers estarán establecidos de la siguiente manera:

Peer A: 192.168.1.11

```
PS C:\Users\SIMON>
                      Get-NetIPConfiguration
InterfaceAlias
                 : Ethernet
InterfaceIndex
                    : 13
InterfaceDescription : Realtek PCIe GBE Family Controller
NetProfile.Name : GARCIAZULUAGA_PLUS
IPv4Address
                    : 192.168.1.11
IPv6DefaultGateway
IPv4DefaultGateway
                    : 192.168.1.1
                    : 200.21.200.10
DNSServer
                      200.21.200.80
```

• Peer B: 192.168.1.41

```
Copyright (C) Microsoft Corporation. All rights reserved.
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS C:\Users\kracr> Get-NetIPConfiguration
InterfaceAlias
                    : Wi-Fi
InterfaceIndex
                    : 13
InterfaceDescription : Qualcomm Atheros QCA9377 Wireless Network Adapter
NetProfile.Name
                      GARCIAZULUAGA_PLUS
IPv4Address
                     : 192.168.1.41
IPv6DefaultGateway
IPv4DefaultGateway
                    : 192.168.1.1
DNSServer
                      200.21.200.10
                      200.21.200.80
```

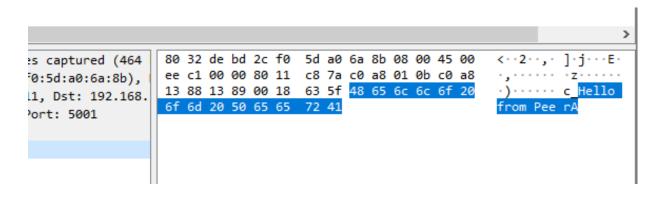
¿Es posible ver en la captura de Wireshark el contenido del mensaje enviado?

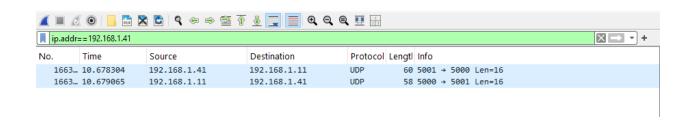
R// Sí, es posible ver claramente el contenido de los mensajes intercambiados entre el PeerA y el PeerB porque no tenemos un algoritmo que esté encriptando el mensaje.

PeerB to PeerA:

```
PS D:\AAUniversity\Semestre 10\Compunet 1\TallerUDP> d:; cd 'd:\A Open file in editor (ctrl + click)
ompunet 1\TallerUDP'; & 'C:\Program Files\Java\jdk-21\bin\java.exe' '@C:\Users\SIMON\AppDat
a\Local\Temp\cp bxmj31vhrz5ag4khdr7esmpl1.argfile' '-m' 'com.example.curriculum_sis/com.exa
mple.curriculum sis.ui.PeerA'
Waiting for a message...
Message from sender:
Hello from PeerB
PS D:\AAUniversity\Semestre 10\Compunet 1\TallerUDP>
         2c f0 5d a0 6a 8b 3c 91 80 32 de bd 08 00 45 00
                                                             ,·]·j·<· ·2····E·
   0010 00 2c 8d ad 00 00 80 11 29 8f c0 a8 01 29 c0 a8
                                                             .,....)...)..
   0020 01 0b 13 89 13 88 00 18 63 5e 48 65 6c 6c 6f 20
                                                                  ··· c^Hello
                                                             from Pee rB
   0030
         66 72 6f 6d 20 50 65 65 72 42 00 00
```

PeerA to PeerB:





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

```
R//
Source Port: 5001
Destination Port: 5000
Length: 24
Checksum: 0x635e [unverified]
[Checksum Status: Unverified]
[Stream index: 17]
[Stream Packet Number: 1]
> [Timestamps]
UDP payload (16 bytes)
```

Este checksum indica información ya que los checksum de la forma 0x6## son informativos. Ahora bien, el checksum 0x635e indica Información de comprobación inconsistente: El checksum ha detectado una inconsistencia en los códigos de comprobación basados en sectores. Por eso, en la captura de Wireshark aparece "[unverified]".

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

R// Proxy es un patrón que centraliza el acceso de la red, y en este caso como solo se tiene una clase de UDP connection, entonces podría facilitar la conexión. Adicionalmente el proxy permite agregar un nivel adicional de seguridad, validaciones y/o restricciones. El patrón command permite encapsular solicitudes complejas y generar una gran librería de solicitudes específicas (por el desacoplamiento entre emisor y receptor). El patrón observer se puede aplicar en la forma como un servidor funciona. Los peers se suscriben a otro y este les envía notificaciones. Esto se podría extender para que los suscriptores también le envien mensajes al peer que actúa de servidor para que pueda manejar las solicitudes (o incluso mandar info a otro peer al que no esté conectado directamente).

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

R// Implementado en el código de la entrega que está en el repositorio. Ahora permite el envío continuo de mensajes entre ambos peers hasta que se envíe el mensaje para terminar la comunicación.

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

R// Ya está implementado.

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

R//Para que se pueda realizar una comunicación tipo TCP debería validar que el mensaje haya llegado y que sea correcta la información que fue transmitida. Solo cuando el transmisor original verifica que la información se haya enviado, ya permite enviar el resto. En el caso de java esto se facilita mediante el uso de Socket y ServerSocket, ambos hacen parte de java.net.

- ¿Qué utilidades de codificación o seguridad agregaría al código?
 R// Algoritmo de encriptación de la información RSA para que no se pueda ver el mensaje directamente.
- BONUS: desarrollen una interfaz de usuario en JavaFX para este programa.

R// Implementado en el código que está en el repositorio.

LINK AL REPOSITORIO:

https://github.com/danielacastanomoreno/TallerUDP