Laboratorio 1: Redes de Computadores

Profesora: Erika Rosas Olivos
Ayudantes: Jorge Díaz M.- Felipe Samur S.

Abril 2021

Objetivos del Laboratorio

- Aprender a trabajar con sockets UDP y TCP en `Python` y `Go`
- Conocer y practicar la estructura cliente-servidor
- Aprender a analizar el tráfico de red usando la herramienta `Wireshark`.
- Familiarizarse con protocolos usados en Internet.

1. Introducción

Un socket es la interfaz entre la capa de aplicación y la capa de transporte, esto ya lo habrán estudiado en clases. Estas interfaces permiten establecer conexiones y comunicar dos aplicaciones que están corriendo en dos máquinas (o en la misma), todo esto en base a una estructura cliente-servidor.

Tanto `Python` como `Go` tienen formas de facilitarnos el trabajo con sockets. `Python` nos permite trabajar los sockets con la libreria `socket` y `Go` tiene un paquete llamado `net`, que nos permitirá trabajar sockets de dominio Unix.

Como ya adelantamos, la arquitectura que busca evaluar este laboratorio es la clásica *cliente-servidor*, que divide la carga de trabajo entre los servidores (proveedores de un servicio), y los clientes (quienes hacen uso del servicio provisto).

Parte de esta actividad incluye trabajar con `Wireshark`. Este es un software que permite analizar el tráfico red en tiempo real, además facilita la identificación de protocolos como TCP o UDP.

2. Tarea

2.1. Enunciado

El clásico "piedra, papel o tijeras", más conocido como *cachipún* por los chilenos, es un juego que ya es parte de nuestra idiosincrasia, por lo que asumimos que conocen sus simples reglas. Para esta primera parte del laboratorio se les encomendará la tarea de crear un "cachipún"de jugador contra la computadora. Para llevar a cabo tal tarea consideren el diagrama que aparece más abajo.

Como se comentó antes, deberán construir una arquitectura cliente-servidor, pero esta constará de tres nodos: Cliente, Servidor Intermediario, Servidor Cachipún.

2.2. Cliente

Este proceso cumple el rol de jugador. El Cliente debe satisfacer las siguientes tareas:

- Establecer una conexión TCP con el Servidor Intermediario.
- No debe conectarse directamente con el Servidor Cachipún.
- Debe mostrar en la consola los resultados del turno y un resultado final indicando si ganó la máquina o el jugador.
- El código de este programa debe estar escrito en `Python`.

2.2.1. Estructura de los Resultados

```
...
[*] Usted jugó Piedra
[*] El bot jugó Tijera
---
[*] El ganador de esta ronda fue usted
[*] El marcador actual es Jugador 3 , Bot  2
....
[*] El ganador de la partida fue: Jugador
```

2.3. Servidor Intermediario

Este nodo cumple el rol de comunicar al Cliente con el Servidor Cachipún. Para esto, se deben satisfacer las siguientes tareas:

- Mantener una conexión TCP con el Cliente.
- Conectarse, cuando sea requerido, con el Servidor Cachipún mediante una conexión UDP.
- Responder al Cliente con el mensaje que recibe del Servidor Cachipún.
- Este debe procesar el turno y enviar el resultado junto con la jugada del rival al Cliente.
- Alertar al Servidor Cachipún del término del juego para que este pueda terminar su ejecución.
- Debe terminar su ejecución cuando el Cliente indique el término del juego.
- El código de este programa debe estar en `Python`

2.4. Servidor Cachipún

Este nodo cumple el rol de Bot en la lógica de Jugador contra Bot (máquina, computadora, etc). Por lo tanto, este nodo debe jugar contra el Cliente ejecutando jugadas aleatorias hasta que se indique el final del juego. Para esto, se deben satisfacer las siguientes tareas:

- Abrir una conexión UDP para comunicarse con el Servidor Intermediario.
- Abrir otra conexión UDP en un puerto aleatorio cada vez que se pida una partida.
- Enviar mensajes al Servidor Intermediario y también recibir mensajes del mismo.
- Debe terminar su ejecución cuando lo indique el Servidor Intermediario.
- El código de este programa debe estar escrito en `Go`

2.5. Diagrama

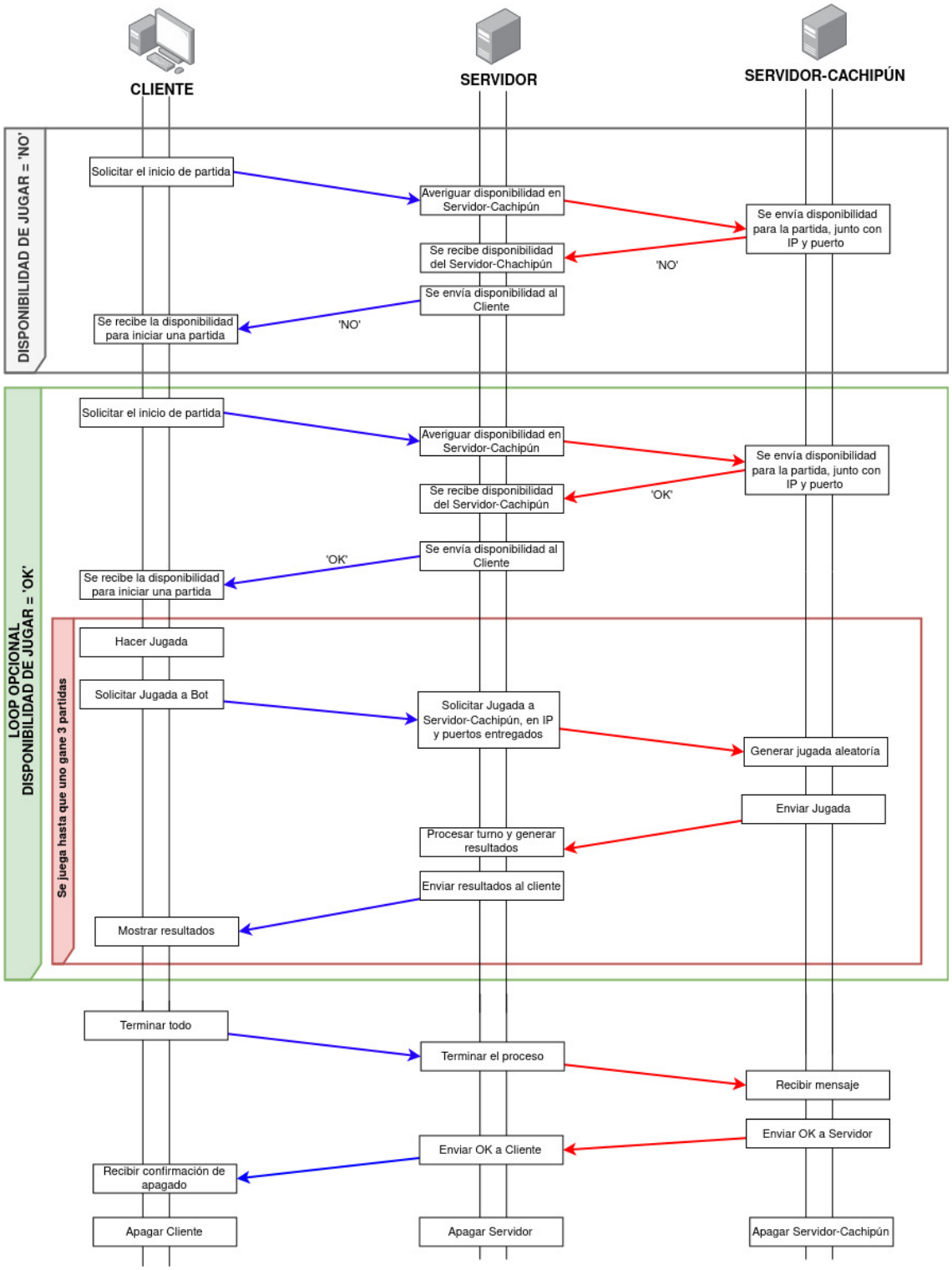

Figura 1: Diagrama del proceso del Laboratorio

3. Análisis de Tráfico

En esta sección, debe emplear la herramienta *Wireshark* para analizar las los paquetes relacionados a la aplicación que desarrolló y responder las siguientes preguntas:

1. Respecto al número de mensajes que se envían durante la ejecución del programa, basándose en el esquema de la Figura 1, ¿Cuantos mensajes se deberían enviar durante una partida? Considere ahora su código, ¿Cuantos mensajes se enviaron en una partida?

2. Referente a los mensajes realizados por las aplicaciones: ¿Qué tipos de protocolo espera ver? ¿Cuáles encontró? Justifique sus expectativas y las diferencias que encuentre.

3. Las interacciones vía TCP entre el Cliente y el Servidor Intermediario, ¿deben ocupan los mismos puertos a lo largo del tiempo? ¿Coincide con lo visto en Wireshark ? Fundamente.

4. Las interacciones vía UDP entre el Servidor Intermediario y el Servidor Cachipún, ¿Deben ocupar los mismos puertos a lo largo del tiempo?,¿Qué cambia al ejecutar varias partidas seguidas sin cerrar el terminal? ¿Coincide con lo visto en Wireshark ? Fundamente.

5. Respecto al contenido de los mensajes enviados entre cada par de entidades, ¿son legibles? ¿Por qué es o no legible el contenido del mensaje?

   Por cada pregunta debe agregar capturas de pantalla para respaldar su respuesta.

4. Bonus

Adicionalmente a lo anterior, se le ofrece un bonus de 20 puntos adicionales a la nota máxima del laboratorio (100pts), por la implementación de una extensión de la tarea. Este bonus contará solamente si y solo si su grupo implementó la tarea por completo, vale decir, si no se hizo el informe o faltó alguna parte del laboratorio pero si hizo el bonus este no contará.

4.1. Enunciado

Dado el contexto mencionado anteriormente, en este bonus ustedes deben implementar la opción de jugar cachipún en modo *multiplayer*, vale decir, ustedes implementarán las conexiones necesarias para poder conectar dos terminales de jugadores, con su respectivo servidor Intermediario, por medio del Servidor Cachipún de manera que puedan jugar entre ellos.

4.2. Cliente

El código del Cliente debe estar en `Python`.

En esta iteración, deberán ejecutarse dos terminales de clientes. Para mayor facilidad, le recomendamos definir una variable que identifique al primer y segundo jugador, ya sea al iniciar la ejecución del cliente o tener dos archivos de código, uno para cada jugador.

Sumado a lo pedido anteriormente, el cliente debe tener adicionalmente la opción de solicitar una partida multiplayer. En cada ronda de la partida, el jugador tendrá que recibir y mostrar por pantalla la jugada que realizó el rival junto con el resultado del turno.

Importante: Para la versión multiplayer, solo un jugador será el encargado de terminar el proceso del servidor cachipún (se retrata en el diagrama).

4.3. Servidor Intermediario

El código del servidor Intermediario debe estar en `Python`

Debe existir un Servidor Intermediario asociado a cada jugador mediante una conexión TCP al igual que en el enunciado del laboratorio.

El Servidor Intermediario debe enviar las jugadas del Cliente al Servidor Cachipún para que este se encargue de enviarla al rival, del mismo modo, el Servidor Intermediario debe recibir las jugadas del rival por medio del Servidor Cachipún, procesar los resultados del turno y enviarle estos resultados al cliente. Las conexiones del Servidor Intermediario con el Servidor Cachipún deben ser conexiones UDP.

Importante: Recuerde que definir un puerto diferente para cada Servidor.

4.4. Servidor Cachipún

El código del Servidor Cachipún debe estar en `Go`

Este servidor servirá como "puente.^{en}tre las jugadas de ambos rivales. El servidor debe ejecutarse en un puerto definido previamente donde cada jugador pedirá una partida online. El servidor debe iniciar la partida si y solo si los dos jugadores solicitaron la partida. Además, avisará a los servidores Intermediario es el puerto en el que se deben conectar para jugar la partida multiplayer. Recordar que dicho puerto debe ser aleatorio en cada partida.

Importante: Recuerde que el Servidor Cachipún debe cerrarse cuando el cliente (que usted asignó) seleccione cerrar el juego.

4.5. Diagrama

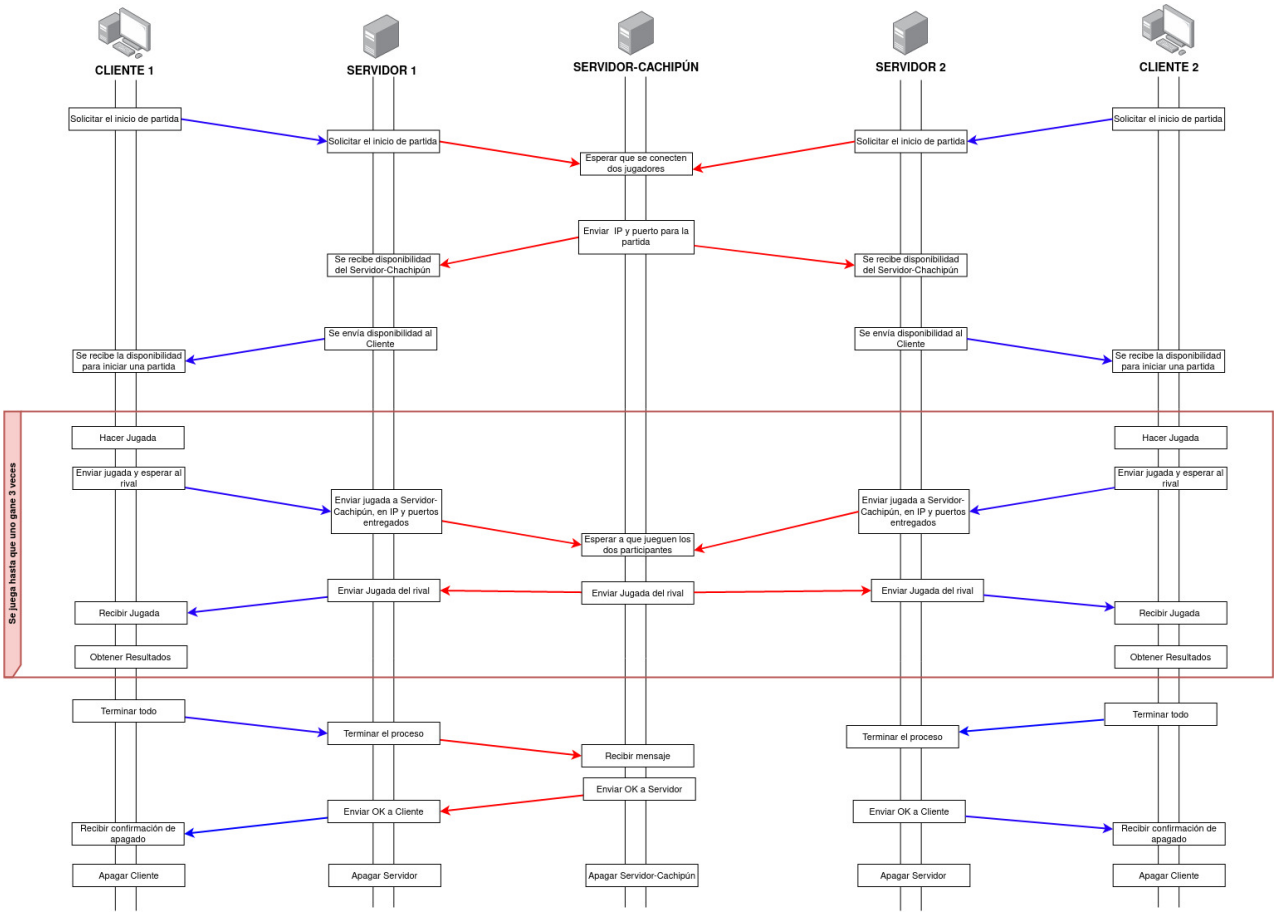

Figura 2: Diagrama Bonus Laboratorio

5. Reglas de Entrega

- La tarea se realiza en grupos de 2 personas. Excepcionalmente existirá un grupo de 3 personas que debe ser autorizados por los ayudantes.
- La fecha de entrega es el día jueves **13 de mayo de 2021** a las 23:55 hrs.
- El código debe correr en `Python 3.7`. Solo se permite utilizar la librería `socket`. Para las conexiones del código en `Go`, se debe utilizar la libreria `net`
- La entrega debe realizarse a través de Aula, en un archivo comprimido `.zip` o `.tar.gz`, y debe indicar el número de su grupo siguiendo el patrón: T1-Grupo[N°Grupo].zip, Ejemplo: T1-Grupo01.zip
- Debe entregar todos los archivos fuente necesarios para la correcta ejecución. Debe haber al menos un archivo para el Cliente, el Servidor Intermediario y el Servidor Cachipún. Recuerde que el código debe estar identado, comentado, sin warnings y sin errores.
- Debe entregar un README con nombre y rol de cada integrante, además de las instrucciones necesarias para ejecutar los archivos.
- Cada día de atraso se penalizará con un descuento de 20 puntos. Después de 3 días de atraso, la tarea será calificada con nota 0.
- Las copias serán evaluadas con nota 0.

6. Consultas

Para hacer sus consultas recomendamos hacerlas en el foro del ramo en Aula. Sin embargo, en caso de algo personal o urgente, los correos de los ayudantes son: `jorge.diazma@sansano.usm.cl` y `jorge.samur@sansano.usm.cl`. Cabe destacar que se responderan consultas vía Aula y/o correo electronico hasta 24 hrs antes de la fecha y hora de entrega (en este caso, hasta el 12 de mayo a las 23:55 hrs).