**UNIVERSIDAD DE TARAPACÁ**



**FACULTAD DE INGENIERÍA**

Departamento de Ingeniería en Computación e Informática



**Taller 1: Sockets Cliente-Servidor**

**Autor**: Iván Cardemil

Patricio Tudela

**Curso**: Sistemas Distribuidos

**Profesor**: Diego Aracena

8 de Septiembre del 2019, Arica, Chile

# 

Contenido

**[1. Introducción](#_Toc18873594)** [3](#_Toc18873594)

[**2. Objetivos** 4](#_Toc18873595)

[**2.1 Propósito** 4](#_Toc18873596)

[**2.2 Objetivo general** 4](#_Toc18873597)

[**2.3 Objetivos específicos** 4](#_Toc18873598)

[**3. Desarrollo** 5](#_Toc18873599)

[**3.1 Estudio Sockets, Modelo Cliente-Servidor** 5](#_Toc18873600)

[**3.2 Diseño de la aplicación** 7](#_Toc18873601)

[**3.2.1 Descripción** 7](#_Toc18873602)

[**3.2.2 Casos de Uso y diagramas de secuencia** 8](#_Toc18873603)

[**3.2.3 Arquitectura** 17](#_Toc18873604)

[**3.2.4 Pruebas de conexión** 19](#_Toc18873605)

[**3.2.5 Pruebas de funcionamiento** 22](#_Toc18873606)

[**4. Conclusión** 27](#_Toc18873607)

[**5. Referencias** 28](#_Toc18873608)

# **1. Introducción**

En el presente documento se presenta el estudio y taller realizado para el curso de sistemas distribuidos, el cual consiste en realizar un estudio sobre los sockets y el modelo cliente-servidor con la finalidad de realizar un sistema distribuido que diera cuenta de lo aprendido en el estudio. El tema del sistema a desarrollar es libre, por lo que el equipo de trabajo decidió realizar un sistema lúdico que permitiera a dos clientes conectarse a un servidor y así jugar una partida de cartas.

# 

# **2. Objetivos**

## **2.1 Propósito**

En el siguiente documento se enseña el resultado del estudio del modelo cliente-servidor y el manejo de sockets. Además, se realiza una detallada descripción del sistema desarrollado utilizando sockets.

## **2.2 Objetivo general**

Investigar sobre el modelo cliente-servidor y desarrollar un sistema distribuido utilizando sockets.

## **2.3 Objetivos específicos**

* Estudiar los aspectos teóricos del modelo cliente-servidor.
* Diseñar y describir un sistema distribuido que utilice sockets.
* Asociar el resultado obtenido con el contenido teórico.

# **3. Desarrollo**

## **3.1 Estudio Sockets, Modelo Cliente-Servidor**

La arquitectura cliente-servidor es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.

Algunos ejemplos de aplicaciones computacionales que usen el modelo cliente-servidor son el Correo electrónico, un Servidor de impresión y la World Wide Web.

La separación entre [cl](https://es.wikipedia.org/wiki/Cliente_(inform%C3%A1tica))i[ente](https://es.wikipedia.org/wiki/Cliente_(inform%C3%A1tica)) y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un solo programa. Los tipos específicos de servidores incluyen los servidores web, los servidores de archivo, los servidores del correo, etc. Mientras que sus propósitos varían de unos servicios a otros, la arquitectura básica seguirá siendo la misma.

La red cliente-servidor es una red de comunicaciones en la cual los clientes están conectados a un servidor, en el que se centralizan los diversos recursos y aplicaciones con que se cuenta; y que los pone a disposición de los clientes cada vez que estos son solicitados. Esto significa que todas las gestiones que se realizan se concentran en el servidor, de manera que en él se disponen los requerimientos provenientes de los clientes que tienen prioridad, los archivos que son de uso público y los que son de uso restringido, los archivos que son de sólo lectura y los que, por el contrario, pueden ser modificados, etc. Este tipo de red puede utilizarse conjuntamente en caso de que se esté utilizando en una red mixta.

Una típica aplicación de red consiste en dos programas, un programa cliente y un programa servidor, que residen en diferentes end systems. Cuando estos dos programas son ejecutados, un proceso del cliente y otro proceso del servidor son creados, y estos procesos se comunican entre sí, escribiendo y escuchando a través de un SOCKET. Por ende al crear una aplicación de red la tarea como programador es escribir un código para el programa de cliente y servidor.

En una aplicación de red, las implementaciones siguen los estándares del RFC. Como por ejemplo seguir el estándar del protocolo FTP permitirá a dos programadores, uno escribiendo el código del cliente y otro el del servidor, comunicar sus programas sin problema alguno, como por ejemplo el navegador Firefox comunicándose con el servidor web Apache de compañías distintas.

También existe otro tipo de aplicación red en la cual un desarrollador o un equipo, crea ambos programas, en el cual tiene el control absoluto de lo que va en el código. La desventaja será que no pueda desarrollar un código que interopere con otros programas de desarrolladores independientes.

Durante la fase de desarrollo de la aplicación red la primera decisión de los desarrolladores es si correr la aplicación utilizando el protocolo TCP o UDP. Mientras que TCP es orientado a la conexión y provee un canal de tramas segura en el cual los datos son transferidos entre dos end systems. UDP es sin conexión y envía paquetes independientes desde un end system a otro sin ninguna garantía de arribo.

## 

## **3.2 Diseño de la aplicación**

### **3.2.1 Descripción**

Para el desarrollo de la actividad el equipo de trabajo decidió desarrollar un sistema lúdico. Este consiste en un juego de cartas por turnos, donde cada jugador comienza con cinco cartas numeradas desde el cero hasta el tres, cuyo objetivo es el de no sobrepasar un límite pre-establecido. Es por eso que en cada turno el jugador deberá lanzar una carta al pozo, donde, para ganar, deberá tener en cuenta el punto anterior.

Una vez que alguno de los jugadores cruce el límite, el sistema le entregará un punto al jugador contrario. El juego mostrará el resumen de los puntos de cada jugador y ofrecerá una partida de revancha, donde se cambiarán las manos de los jugadores de forma aleatoria. En caso que los jugadores acepten una partida de revancha, el primer turno será asignado al jugador que ganó la partida anterior.

El sistema termina cuando uno de los jugadores cierra la aplicación cliente.

Las tecnologías utilizadas para desarrollar la aplicación fueron:

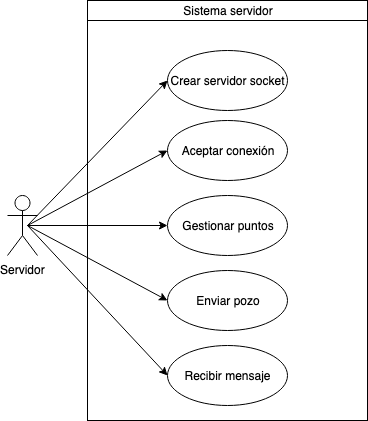
* Java como lenguaje de programación, debido a sus librerías y documentación relacionada en el ámbito de la programación con sockets. Además de cumplir con la características de portabilidad de los programas desarrollados con esta tecnología.
* Netbeans como entorno de desarrollo.

### **3.2.2 Casos de Uso y diagramas de secuencia**

A continuación se realizará la descripción de los casos de uso para el actor cliente y servidor, con su respectivo diagrama de secuencia.

#### 3.2.2.1 Caso de uso del servidor

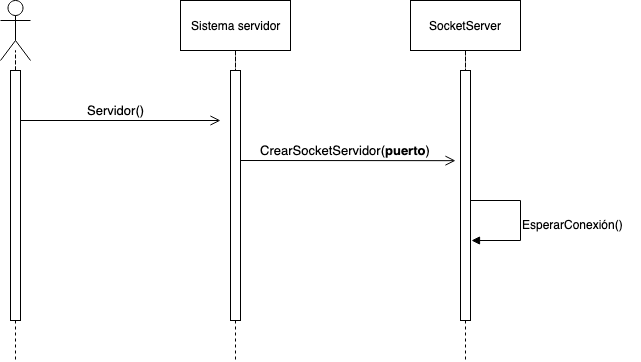
Como se puede apreciar en el siguiente diagrama, se definen las funciones básicas del servidor, como son crear servidor, aceptar conexión de un nuevo cliente, enviar pozo a los clientes, recibir mensajes desde los clientes y gestionar los puntos que ganan los jugadores a lo largo de las partidas.



##### 

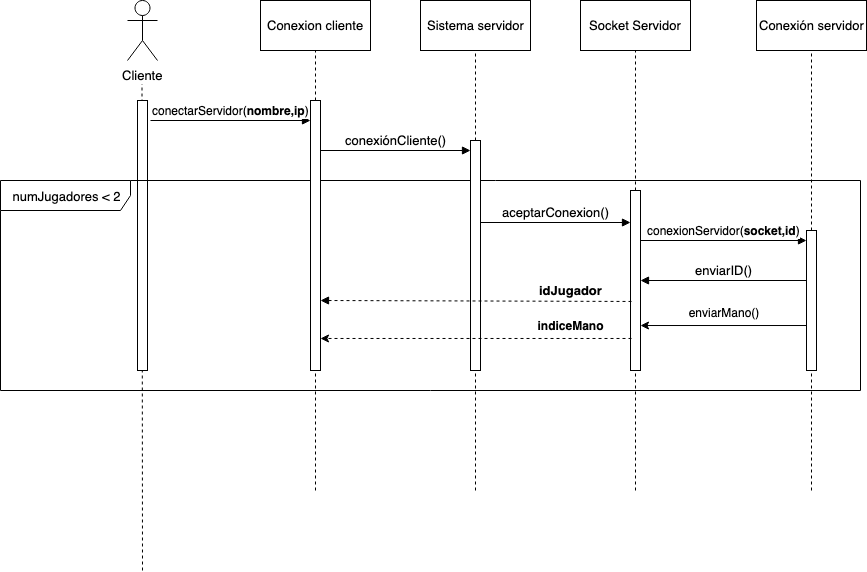
##### 3.2.2.1.1 Diagrama de secuencia: Crear servidor socket

El siguiente diagrama representa el proceso de creación del servidor socket, a la espera de nuevos clientes.



##### 3.2.2.1.2 Diagrama de secuencia: Aceptar conexión

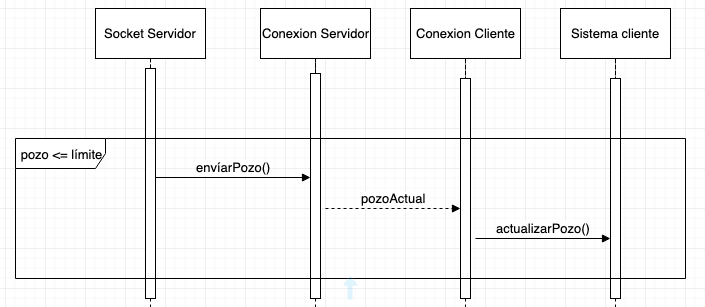
En este caso de uso se puede apreciar la petición de ingresar al servidor, donde la clase “Conexión cliente” es la encargada de realizar la comunicación entre el cliente y el servidor. Por otra parte, la clase “Conexión servidor” es la clase encargada de gestionar la lógica del juego desde el Servidor.



##### 

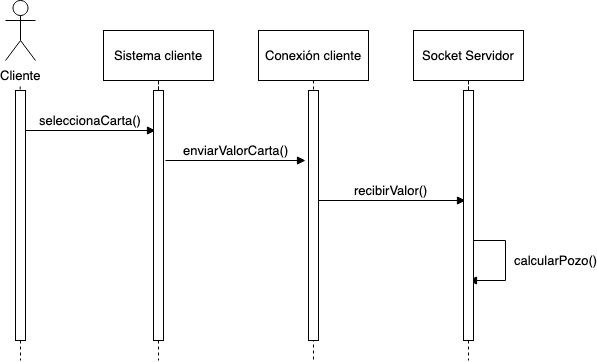
##### 3.2.2.1.3 Diagrama de secuencia: Enviar pozo

El siguiente diagrama describe la acción de enviar datos desde el servidor hacia el cliente. Acá se puede destacar la condición en la que el pozo debe ser menor o igual al límite para que siga enviando el valor del pozo. Este es el caso de uso más utilizado durante la partida, ya que es el que permite observar a los jugadores la jugada que realizó el contrincante.



##### 3.2.2.1.4 Diagrama de secuencia: Recibir mensaje

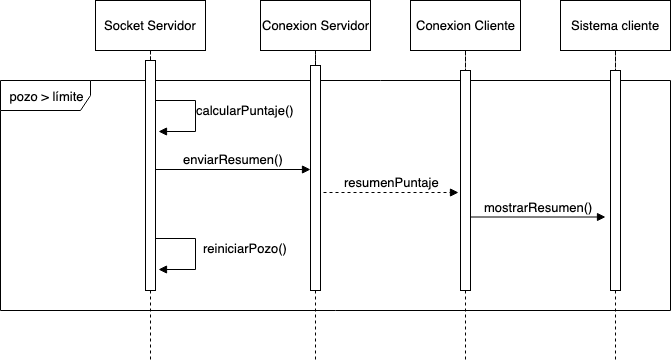
Como se puede apreciar en este sencillo diagrama, el cliente envía el valor de una carta al servidor y este realiza el cálculo del pozo con el objetivo de determinar si la partida puede continuar o se da por finalizada.



##### 

##### 3.2.2.1.5 Diagrama de secuencia: Gestionar puntos

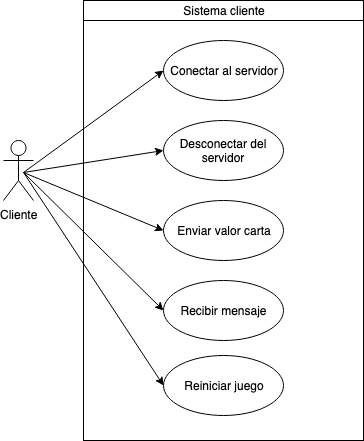
En el siguiente diagrama se puede observar cómo se envía el mensaje cuando finaliza la partida. Como se puede apreciar, es en el socket servidor donde se calcula el puntaje de cada jugador, se envía hacia a la clase encargada de gestionar la comunicación con los clientes y envía un mensaje para que sea visualizado en la aplicación cliente. Además, finaliza la tarea reiniciando el pozo a cero.



#### 

#### 3.2.2.2 Caso de uso del cliente

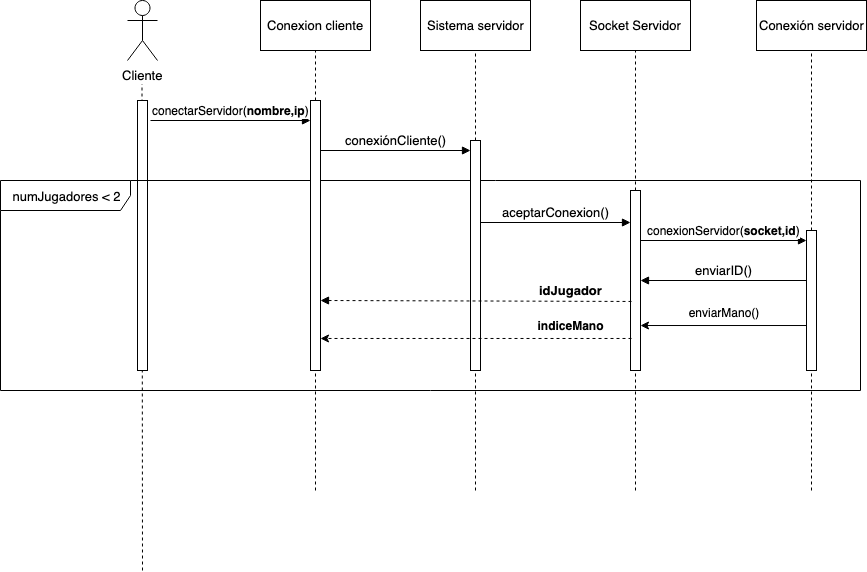
Los casos de uso definidos para el cliente del sistema desarrollado son los que se presentan en el siguiente diagrama, donde destaca la conexión y desconexión al servidor, el envío del valor de la carta seleccionada por el jugador hacia el servidor, la recepción de mensajes y el reinicio del juego.



##### 

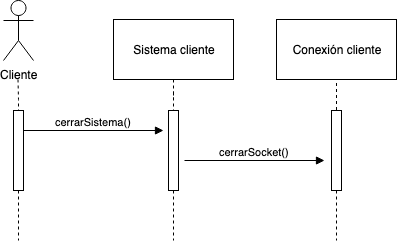
##### 3.2.2.2.1 Diagrama de secuencia: Conectar al servidor

Como se presentó en el diagrama de aceptar conexión, cuando el usuario desea ingresar al servidor, deberá ingresar un nombre de usuario y la dirección ip del servidor. Este proceso se logra por medio de la clase intermedia de “Conexión cliente” que es la encargada de realizar las gestiones por parte del cliente para que este se conecte al servidor. Antes de aceptar la conexión, el servidor revisa si ya ha superado su capacidad máxima. Si el número de jugadores es menor que dos, entonces el servidor envía al cliente por medio del socket la identificación del jugador y el índice, generado al azar, de la mano que utilizará el jugador en la partida. Cabe destacar que el servidor solo manda el índice de la mano, ya que estas están cargadas en una matriz en la aplicación del lado del cliente.



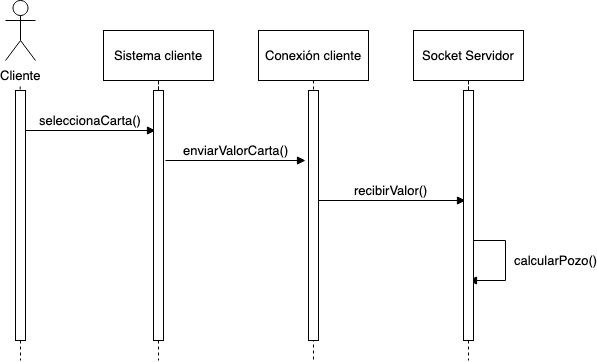
##### 2.2.2.2.2 Diagrama de secuencia: Desconectar al servidor

Al momento en que el cliente cierra la aplicación, este cierra el socket del lado del cliente.



##### 3.2.2.2.3 Diagrama de secuencia: Enviar valor carta

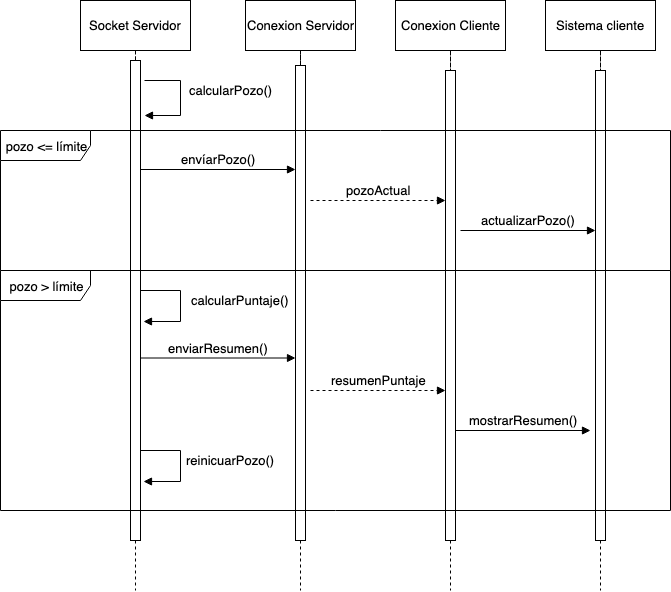
En el siguiente caso de uso, el cliente selecciona una carta de su mano y se envía el valor seleccionado hacia el servidor, como se puede apreciar en el diagrama la clase intermedia de “Conexión cliente” es el encargado de comunicarse con el socket del servidor. Una vez que el servidor recibe el valor de la carta, este recalcula su el valor del pozo, para reenviarlo al jugador contrario.



##### 

##### 3.2.2.2.4 Diagrama de secuencia: Recibir mensaje

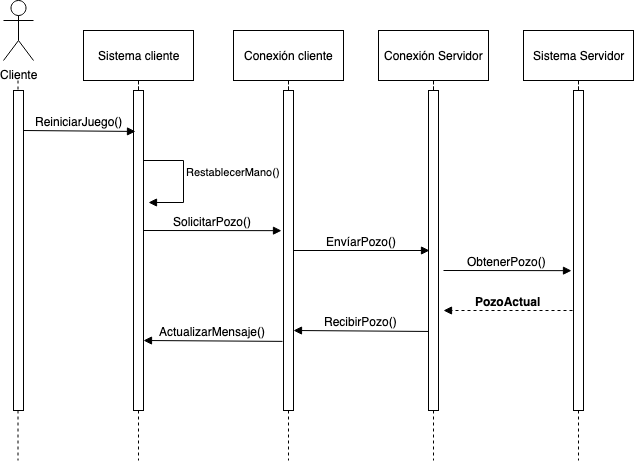
En el siguiente diagrama se puede apreciar el proceso de recibir mensajes de parte del cliente, donde se nos presentan dos escenarios. El primero es en el caso que el pozo sea menor que el límite, por lo que el proceso de enviar mensajes desde el servidor es el habitual, donde la clases conexión servidor y conexión cliente son las encargadas de gestionar el envío y recepción de mensajes. El segundo escenario corresponde al caso en el que el pozo es mayor que el límite establecido, por lo que el servidor deberá calcular el puntaje de los jugadores y enviar un resumen del juego, donde se muestra el nombre de cada jugador y el puntaje de cada uno. Al final el escenario descrito, el socket deberá reiniciar el valor del pozo a cero.



##### 

##### 3.2.2.2.5 Diagrama de secuencia: Reiniciar Juego

Como se puede apreciar en el siguiente diagrama de secuencia, el caso de uso inicia con el cliente reiniciando el juego, por lo que actualiza su mano y solicita el valor del pozo al servidor, que sería cero.



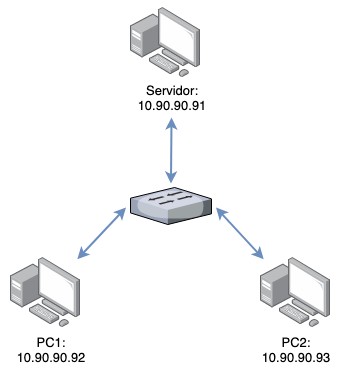
### **3.2.3 Arquitectura**

#### 3.2.3.1 Topología del sistema

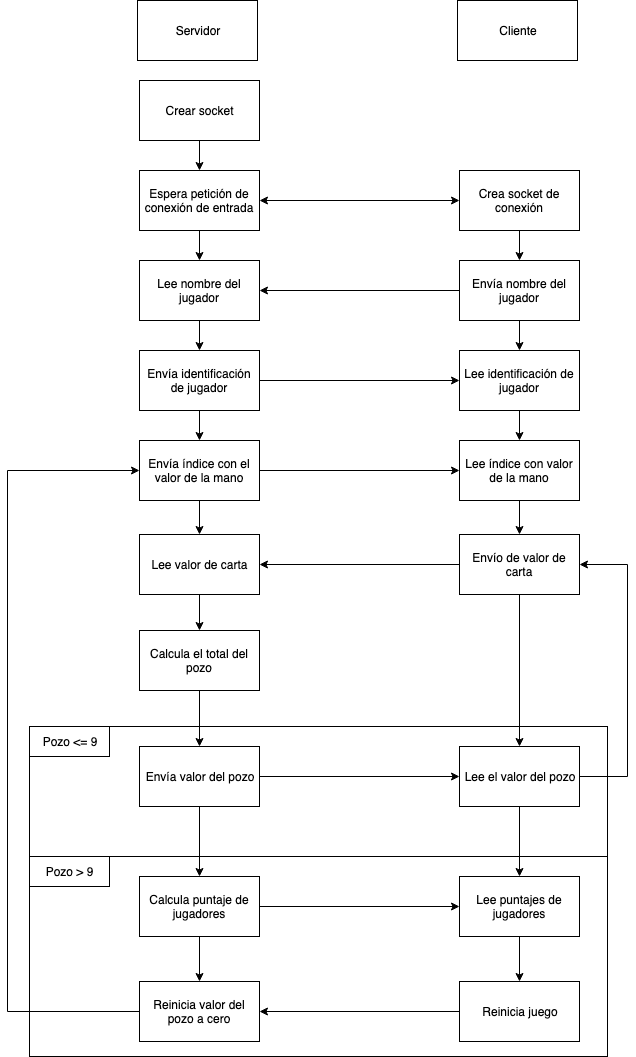
La arquitectura del sistema se puede apreciar en la siguiente figura, donde el servidor es el encargado de administrar la comunicación entre dos clientes ubicados en distintos computadores. El servidor solo puede manejar dos comunicaciones de forma simultánea, por lo que estaríamos hablando de un sistema no persistente, ya que no permite entrar a un tercer cliente mientras el servidor esté en su capacidad máxima.

Como se puede apreciar en la figura, se utilizaron tres computadores y un switch para levantar el sistema, donde una de los computadores cumplió el rol de servidor y los otros dos hicieron de clientes.

Para realizar la conexión de forma correcta, el cliente debe indicar la IP y el puerto del servidor. De acuerdo al sistema desarrollado, el usuario solo deberá colocar la dirección IP, ya que la el puerto (**51734**) ya está predefinido en la aplicación cliente.



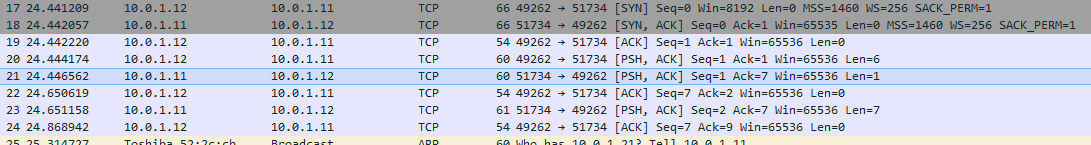
#### 3.2.3.2 Diagrama de intercambio de mensajes



### **3.2.4 Pruebas de conexión**

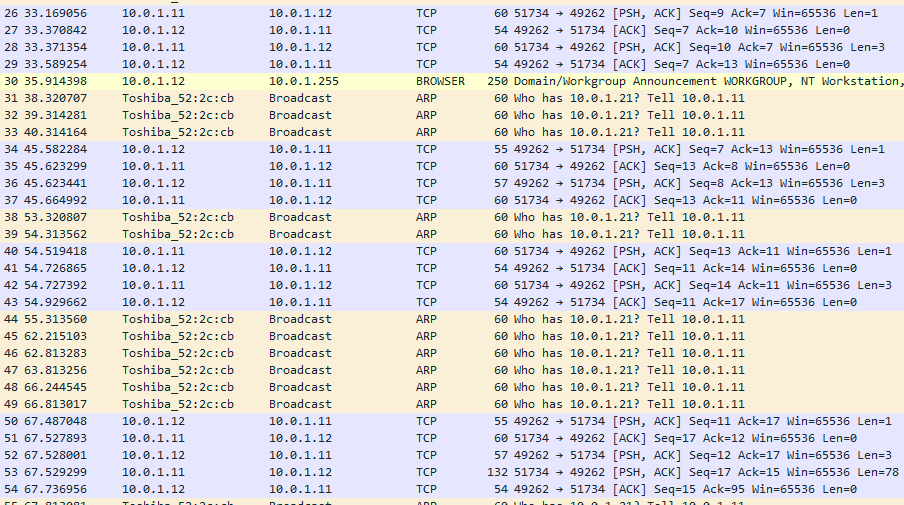
Al realizar las pruebas el equipo utilizó la aplicación Wireshark para visualizar el movimiento de paquetes entre los dispositivos. Para esto se capturó los movimientos en la red del dispositivo servidor y uno de los dispositivos clientes.

A continuación podemos ver los movimientos en la red, de parte del dispositivo cliente, el cual tiene una ip 10.0.1.12 y se conecta al dispositivo con ip 10.0.1.11 el cual es el servidor.



Una vez inicializado el juego, serán elegidas cartas las cuales serán notificadas al servidor para actualizar el pozo que luego enviará al cliente. En la imagen se ven representados como 4 ráfagas de mensajes con sus respectivos ACK.

Finalmente una vez que el juego haya llegado a su fin el servidor enviará un mensaje indicando el marcador de la partida, esto se aprecia en la ráfaga de 5 mensajes.

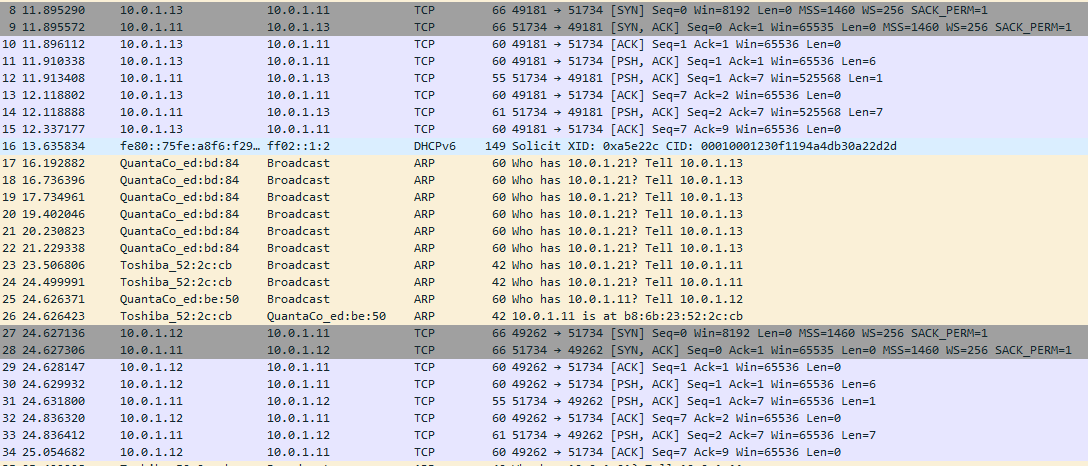


Si el cliente desea retirarse del juego podrá cerrar la aplicación, lo que arroja el mensaje que la conexión a sido cerrada, representada por este mensaje en rojo.

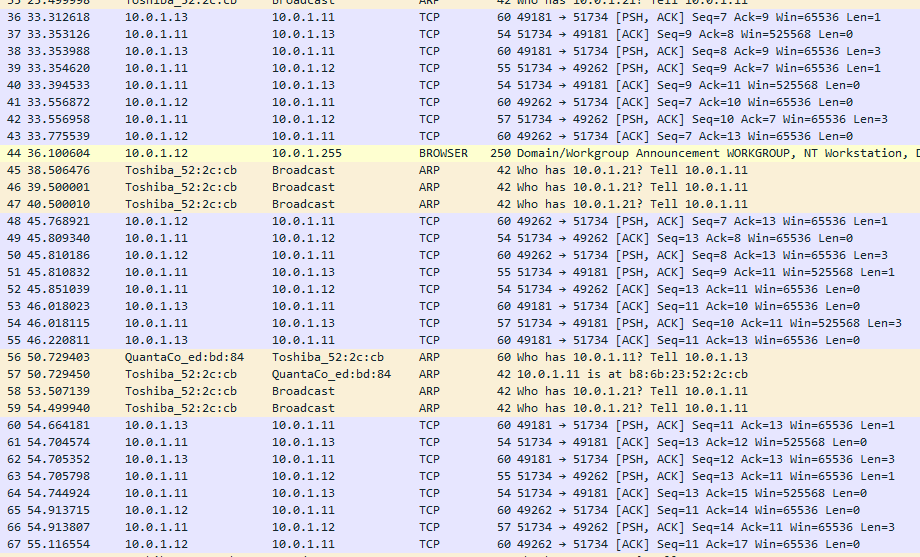


Ahora veremos el movimiento de red de parte del servidor.

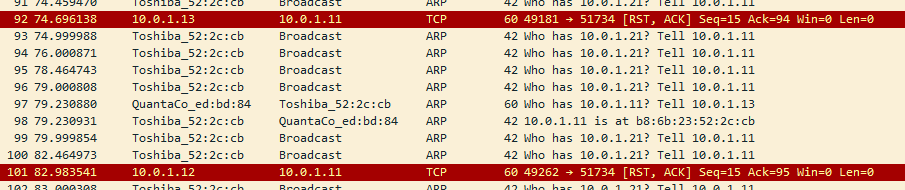
La siguiente imagen indica que los dispositivos 10.0.1.12 y 10.0.1.13 han realizado la conexión al servidor exitosamente.



Al momento que un cliente envía la carta el servidor actualizará el pozo total, que seguidamente enviará a cada uno de los clientes. Como se aprecia en la imagen los clientes toman turnos para enviar sus cartas, y entre medio el servidor les va enviando el pozo actualizado.

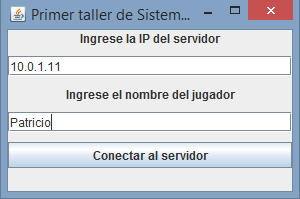
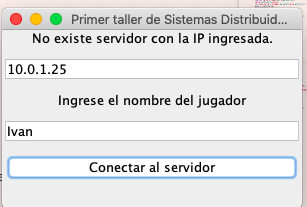


En el momento que los clientes cierren la aplicación, wireshark captura los mensajes de cierre en color rojo, detectando el origen y destinatarios.



### **3.2.5 Pruebas de funcionamiento**

En la siguiente imagen se puede apreciar la ventana que permite “Conectar al servidor”, donde el usuario deberá ingresar la Ip del servidor y un nombre. El sistema es capaz de manejar dos errores. El primero es en caso que la IP del servidor no exista, el programa mostrará un mensaje de alerta. El otro caso de error se da cuando el jugador no ingresa el nombre, por lo que el sistema reclamará para que se complete esa casilla.

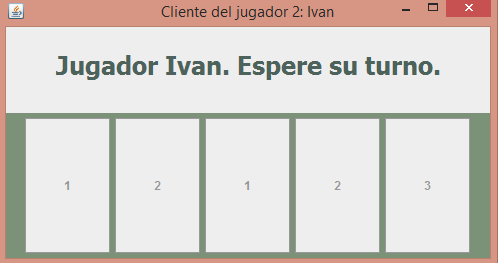
 

En la siguiente pantalla se pueden ver las dos áreas fundamentales del juego, la primera corresponde al panel que posee una etiqueta (JLabel), que en el primer turno indicará las acciones que debe realizar cada jugador. En el otro panel se puede observar el conjunto de botones que representan a las cartas del juego.

Esta figura corresponde al jugador que tiene que hacer la primera jugada, por lo que el panel de botones se encuentra disponible. Una vez seleccionado una carta, el panel de botones se bloqueará y la etiqueta se actualizará con el valor de la carta seleccionada.



La siguiente imagen corresponde a la pantalla del segundo jugador, en ella se encuentra una etiqueta donde indica que el jugador deberá esperar hasta que el primer jugador realice su jugada. Es por ese motivo que su panel de cartas se encuentra bloqueado.

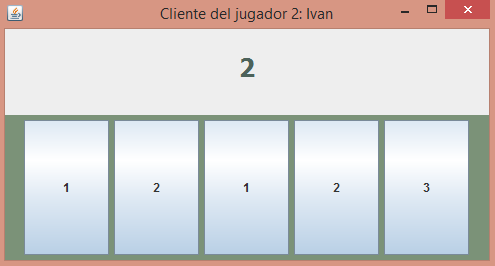


En las siguientes imágenes muestran cuando él un jugador selecciona una carta de su mano y esa acción se ve reflejada en la pantalla del otro jugador. Como se puede apreciar, el jugador uno selecciona una carta con valor dos, por lo que la carta seleccionada desaparece y esto desencadena la actualización del área que posee la etiqueta, que antes contenía un mensaje y el bloqueo del área de cartas.

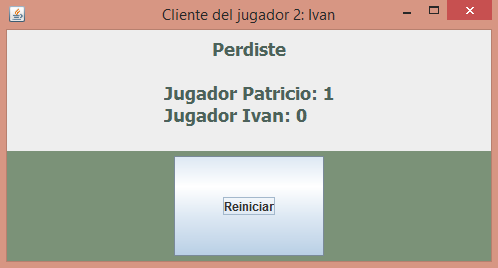


Como se describió en la imagen anterior, la acción realizada por el jugador uno, se vió reflejada en la pantalla del jugador dos. Además, luego de ocurrido este hecho, el área de botones del jugador dos se desbloqueara para que puede realizar su acción.

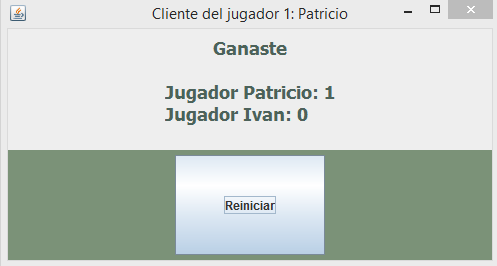
Estas acciones ocurrirán hasta que uno de los jugadores sobrepase el límite establecido, que en este caso sería el valor nueve.



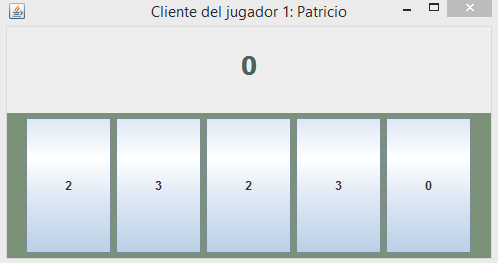
Las siguientes imágenes muestran cuando la partida ha terminado, mostrando el marcador y los puntos que tiene cada jugador. Estos son guardados por el servidor por lo cual puede ir acumulándolos cada vez que gane un jugador. El jugador que ganó la partida recibe el mensaje “Ganaste” mientras que el perdedor recibe el mensaje “Perdiste”. Además son ocultadas las cartas y reemplazadas con un botón “Reiniciar” lo cual permite a los jugadores volver a jugar otra partida. El jugador que ganó la ronda anterior será el que comienza primero en la siguiente.



Acá podemos ver la pantalla del jugador que a perdido esta ronda. El cual también posee el botón “Reiniciar”.



Ahora en las siguientes imágenes podemos apreciar lo que sucede cuando los dos jugadores presionan el botón “Reiniciar” ya que el cliente 1 ganó la ronda anterior, este empieza primero, mientras que el cliente 2, quien perdió la ronda anterior, deberá esperar a que el cliente 1 juegue su carta.



Como vemos el cliente 2 permanecerá con sus cartas bloqueadas hasta que el cliente uno juegue termine su turno.



Este proceso se repetirá por cuantas rondas deseen jugar los participantes. En algún momento ellos saldrán de la aplicación cerrando la ventana.

# 

# 

# **4. Conclusión**

Este taller ha aportado al equipo importantes conocimientos sobre el las aplicaciones red y sus requerimientos para ser desarrolladas utilizando sockets y un modelo cliente-servidor. Ya que el tema era de elección libre el equipo eligió un sistema sencillo que diera cuenta de lo requerido, sin realizar un sistema complejo o extravagante.

Si bien resultó sencillo seguir los estándares para realizar una conexión entre dispositivos, se presentaron ciertos problemas al codificar el juego para que utilizara estas conexiones, se debió diseñar con detalle el orden de los mensajes y los eventos que ocurrirían en ciertos casos. Luego del desarrollo de esta aplicación red el equipo quedó conforme con el producto, ya que logró ser robusto, intuitivo y fácil de usar.

Como trabajo a futuro el equipo propuso ideas para mejorar la aplicación y las mecánicas del juego. Cómo añadir más jugadores a las partidas, mejorar la interfaz para hacerla más agradable, cambiar mecánicas a la hora de jugar cartas, como cartas comodines o trampas.

# 

# **5. Referencias**

* Network Programming Socket. 2002. Material entregado por el profesor.
* P. J. Deitel y H. M. Deitel. «Como Programar con Java» 5ta Edición. 2004. Capítulo 18.
* Modelo Cliente servidor [Modelo Cliente Servidor](https://es.wikipedia.org/wiki/Cliente-servidor).