

Sistemas Distribuidos

**Comunicación integrada entre distintas maquinarias**

Profesor: Rodrigo Pavez  
Alumnos: Cristobal Henriquez, Yorch Sepúlveda, Daniela Paredes.

*1 de abril de 2020*

# **Introducción**

En las últimas décadas las tecnologías han tenido un crecimiento enorme y con ello los sistemas que lo administran. Es por ello que cada vez se requiere que sean tolerantes ante cualquier circunstancia, como por ejemplo los sistemas bancarios, los sistemas de transacciones en las multitiendas, entre otras, en donde las personas requieren que estos sistemas sean confiables, seguros y robustos. Es por ello que en este informe se explicará en base a un caso real, pero a gran escala, algunas fallas y soluciones de un sistema para una empresa de combustibles.

# **Alcance del proyecto**

A continuación se describen la totalidad de las características deseables tanto para el **“Proyecto 2”**  como el **“Proyecto 3”**.

**Características del proyecto de la unidad anterior (Proyecto 2):**

* Tanto las sucursales como la empresa de combustibles poseen una base de datos diferentes.
* La empresa es la única que puede modificar el precio del combustible mientras que las sucursales sólo reciben los cambios de ésta.
* Un surtidor no puede actualizar su precio si se encuentra en operación.
* Cuando se realiza una transacción en una sucursal, es necesario que ésta la envíe a la empresa.
* Cuando se inicia una transacción, es la empresa la que crea y almacena el nombre de la sucursal (IP), asumiendo que ésta es estática.

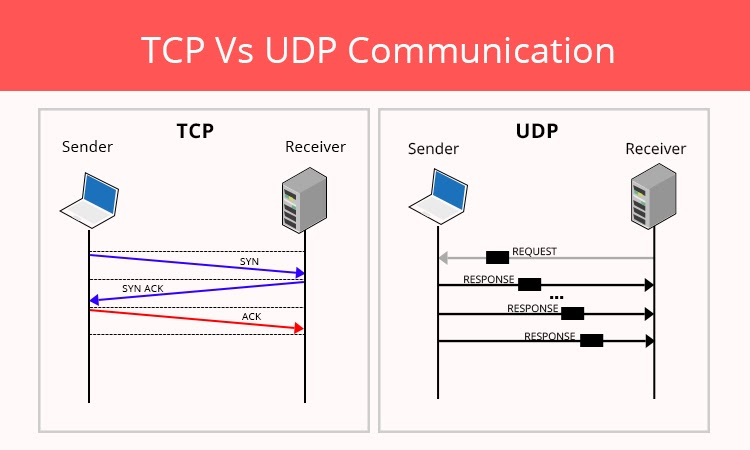
**Características del proyecto de la unidad actual (Proyecto 3):**

1. Se debe diseñar una arquitectura que permita mantener el sistema lo más estable posible, reduciendo los efectos por caídas en el sistema.
   1. Estas caídas pueden ser ocasionadas por ataques a ciertas máquinas, fallos en la comunicación o incluso por fallas en la parte fısica de algún equipo en el lado del servidor.
2. Ninguna venta debería verse alterada por algún inconveniente en la transmisión de datos o por fallas en el servidor, por lo que se solicita mantener una replicación del servidor en dos áreas distintas.
3. En caso de existir algún fallo en alguno de los surtidores, a la hora de realizar una venta (no en el servidor) se debería informar y el surtidor debería estar de regreso en el menor tiempo posible, dejando almacenado el registro de la fecha, hora y tiempo de la caída.
4. Se debería asegurar que el precio de los combustibles no se vea afectado por problemas en la comunicación.
5. Se debería asegurar la posibilidad de obtener reportes en todo momento por parte de la casa matriz.
6. En caso de perder la comunicación por caída de la red, el surtidor no debería dejar realizar cargas de combustible. Si en algún caso el fallo se produce mientras se estaba realizando la carga, se debe asegurar que el registro final llegue a la casa matriz o sucursal tan pronto exista comunicación.

# **Solución**

Para la resolución de este trabajo, se debe adaptar el proyecto realizado en la unidad anterior, es por esto que se decide rehacer dicho proyecto ya que presentaba problemas que interfieren con el cumplimiento de los nuevos requisitos. Por ejemplo, el primer requisito nuevo (punto 1 del item anterior), que solicita un sistema lo más estable posible, reduciendo los efectos por caídas, lo cual lamentablemente era incompatible con lo presentado anteriormente, ya que se contaba con caídas inesperadas.

El cambio más importante realizado en el proyecto de la unidad 2 fue pasar de UDP a TCP lo que implica un cambio en el manejo de la información, que se puede apreciar en la Figura 1:



**Figura 1:** Diferencias entre TCP y UDP

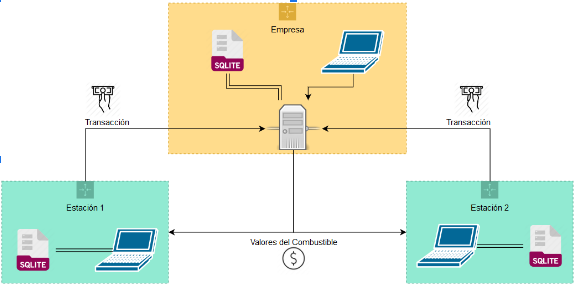
Este proyecto se basa en el protocolo TCP, en el cual pueden existir muchas conexiones simultáneas pero una de éstas es la encargada de abrir un *socket* en un determinado puerto TCP y se queda a la escucha de nuevas conexiones. Es común referirse a esto como apertura pasiva, y determina el lado servidor de una conexión. El lado cliente de una conexión realiza una apertura activa de un puerto enviando un paquete SYN inicial al servidor como parte de la negociación en tres pasos. En el lado del servidor (este receptor también puede ser una PC o alguna estación terminal) se comprueba si el puerto está abierto, es decir, si existe algún proceso escuchando en ese puerto, pues se debe verificar que el dispositivo de destino tenga este servicio activo y esté aceptando peticiones en el número de puerto que el cliente intenta usar para la sesión. En caso de no estarlo, se envía al cliente un paquete de respuesta con el bit RST activado, lo que significa el rechazo del intento de conexión. En caso de que sí se encuentre abierto el puerto, el lado servidor respondería a la petición SYN válida con un paquete SYN/ACK. Finalmente, el cliente debería responderle al servidor con un ACK, completando así la negociación en tres pasos (SYN, SYN/ACK y ACK) y la fase de establecimiento de conexión. Es interesante notar que existe un número de secuencia generado por cada lado, ayudando de este modo a que no se puedan establecer conexiones falseadas (*spoofing*)

El motivo del cambio de protocolo utilizado en el proyecto de la unidad 2, fue que en ese proyecto utilizamos UDP priorizando la velocidad sobre la robustez del sistema, sin embargo al crecer el proyecto y agregando así nuevas necesidades del cliente, entre las que se encuentra la robustez del sistema, es que se vuelve inevitable el utilizar TCP.

## **Lógica detrás de la solución implementada**

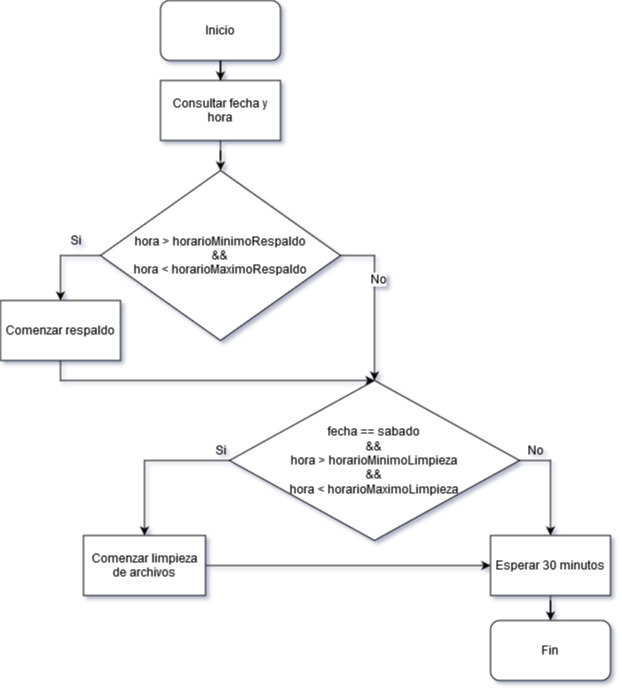
A grandes rasgos, en el proyecto 2 la empresa (la cual actúa como servidor) envía de forma simultánea los valores de los combustibles a las sucursales cada vez que se realiza un cambio en éstos. Por otro lado, las sucursales sólo enviarán sus transacciones cuando estas ocurran tras almacenarlas en su base de datos local.

La arquitectura física elegida para dar solución al problema fue la arquitectura cliente servidor, como se puede apreciar en la Figura 2.

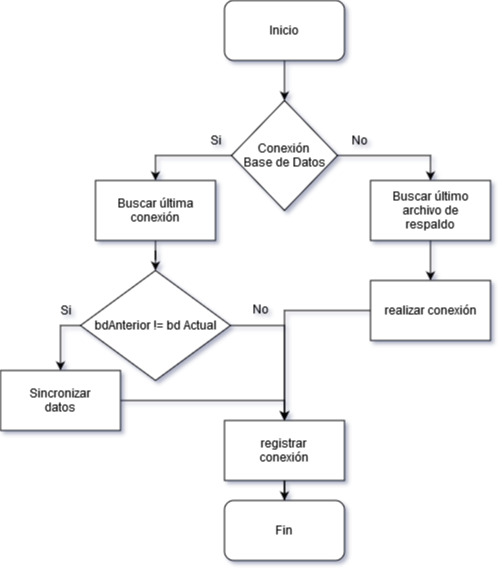
  
**Figura 2:** Arquitectura física

En cuanto al proyecto 3 se toma como base el proyecto 2 pero con la finalidad de que este último sea tolerante a fallos. Por ese mismo motivo se realizaron algunos cambios menores que pueden visualizarse en las Figuras 14 y 15, que se encuentran en el anexo. A continuación se detalla cuales son las fallas encontradas y cómo se logran solucionar.

|  |  |
| --- | --- |
| **Fallos** | **Solución** |
| Caída de la base de datos | Se decidió que en caso de que la base de datos principal llegara a fallar, el sistema debe conectarse al último respaldo. No obstante, el programa debe hacer una sincronización entre ambas bases de datos si la principal volviera a estar disponible (ver **Figura 4**) |
| Backup de la base de datos | Se decidió que todos los días, de 3:01 a.m. a 3:59 a.m., el sistema debe realizar un respaldo de archivos en una carpeta aislada. Así mismo, para poder evitar una saturación de archivos fue necesario implementar una limpieza de éstos. (ver **Figura 3**) |
| Datos sin utilizar | Debido a que como se decidió que el sistema cada treinta minutos realice una operación de respaldo y limpieza, es normal que haya más de un archivo de respaldo diario. Para evitar el exceso de archivos sin utilizar, todos los sábados de 4:01 a 4:59 se realizará una limpieza de archivos. (Ver **Figura 3**) |
| Sincronización de datos | Como se mencionó anteriormente, de existir un problema en la base de datos y luego ésta volviera a estar disponible, es natural que presente una falta de información. Para evitar ésto y mantener las configuraciones actuales del programa (en este caso, el valor del combustible) ambos, servidor y sucursales deben sincronizar sus datos entre su base de datos principal y la última de respaldo utilizada. (ver **Figura 4**). |



**Figura 3:** Flujo de ejecución de limpieza



**Figura 4:** Flujo de ejecución de conexión

# Conclusión

Un sistema tolerante a fallos hace que las personas tengan completa libertad de usarlos, sobre todo en sistemas financieros como los que tienen los bancos y las grandes tiendas. Además se requiere la interacción entre distintas entidades, por lo tanto, otro factor importante es que no pierda la comunicación entre ellos, y en caso de que suceda esto, no exista pérdida de datos. Es por ello que, estos sistemas requieren tener un respaldo para asegurar que la información no se pierda en su totalidad en caso de que las fuentes de almacenamiento principales tengan pérdidas considerables. Por lo tanto, un sistema tolerante a fallos es un sistema robusto, confiable y que entrega seguridad al momento de usarlo. Es por esto que se creó un sistema con una alta tolerancia a fallos y que nos permitió comprender el funcionamiento de sincronización de datos, respaldo de archivos y problemas de conexión entre distintas entidades, además de la utilización de los distintos protocolos de transporte de datos (TCP y UDP). Por otra parte cabe destacar que se logra diferenciar cómo operan las capa de transporte que opera a nivel de software y la capa de red que opera a nivel de hardware. Ambos se encargan de la obtención y transmisión de transporte de paquetes de datos pero en capas diferentes.

Gracias a los nuevos requisitos solicitados en el presente proyecto es que se logró una alta comprensión en cuanto a cuándo utilizar los protocolos TCP y UDP, ya que en el proyecto anterior se utilizó TCP por su velocidad y como se aclaró en el apartado de “Solución” en el informe, se debió cambiar a UDP dada la exigencia presente en el requisito 1 del nuevo proyecto, lo cual permitió experimentar y trabajar con ambos protocolos.

# Trabajo futuro

1. Implementar la reconexión automática entre el servidor y el cliente en caso de sufrir alguna falla el servidor y terminar su conexión.
2. Cambiar la herramienta utilizada para la implementación de la base de datos por una más robusta, ya que esta fue elegida exclusivamente por su rapida y facil implementacion. Sqlite tiene consultas a nivel de programa, uno puede enviar una consulta sql a través de una variable y esta se ejecutará sin problemas, al contrario si se usa un motor más robusto como postgres, se puede crear un proceso de almacenado para que dichas variables sean almacenadas, pero no ejecutadas en la base de datos.

# Anexos

## ReadMe

### Consideraciones

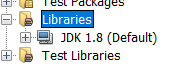
* Se utiliza Netbeans 8.2 para desarrollar la aplicación.
* La versión de Java es 1.8.0\_241
* Las librerías externas a utilizar se encuentran en la carpeta dist de ambas aplicaciones

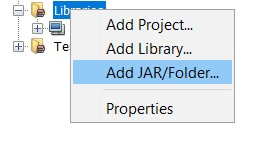
**Figura 4:** Rutas en donde se encuentran las librerías a utilizar en la aplicación

#### Pasos para la ejecución

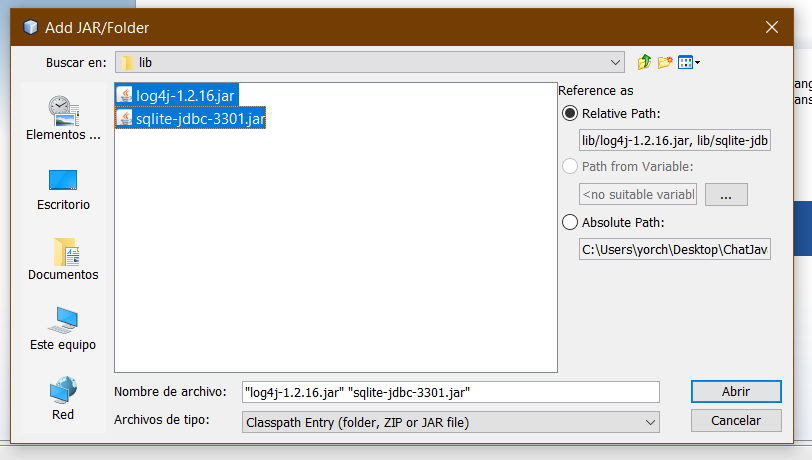
1. Abrir en Netbeans el proyecto Empresa. Luego se debe importar las librerías **sqlite-jdbc-3302.jar** y **log4j-1.2.16.jar** en netbeans de la siguiente manera:



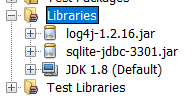
**Figura 5:** Carpeta que debe contener las librerías que la aplicación debe utilizar para su correcto funcionamiento



**Figura 6:** Opción que el usuario debe seleccionar para ingresar las librerías que la aplicación de utilizar, debido a que éstas se encuentran con la extención .**jar**



**Figura 7:** Selección de las librerías a importar



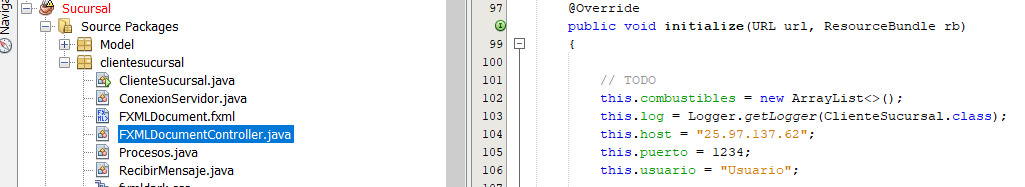
**Figura 8:** Muestra cómo se debe visualizar la carpeta “**Libraries**” de la aplicación

1. Antes de ejecutar el proyecto, se debe crear una red en Hamachi y conocer la ip en la que se encuentra ejecutando el servidor además del puerto por el cual se realizará la conexión, debido a que es necesario que la empresa y las sucursales se deben encontrar en la misma red.



**Figura 9:** Ip del servidor en la aplicación LogMeIn Hamachi

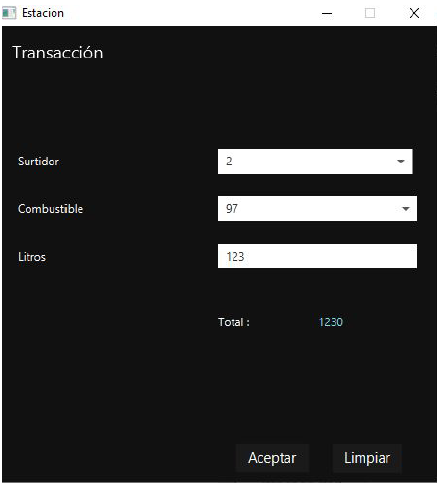
1. Tras realizar los pasos anteriores, se compila el servidor (llamado empresa) y éste queda en espera hasta que un cliente establezca conexión. Para poder realizar el traspaso de paquetes, es necesario que ambos proyectos se encuentren en el mismo puerto.
2. Realizada la ejecución del servidor, es necesario volver al **Paso 1** pero en este caso, para el proyecto cliente (llamado sucursal).
3. Se debe ingresar en el proyecto **“Sucursal”** la ip y el puerto en donde se establece conexión con el servidor.

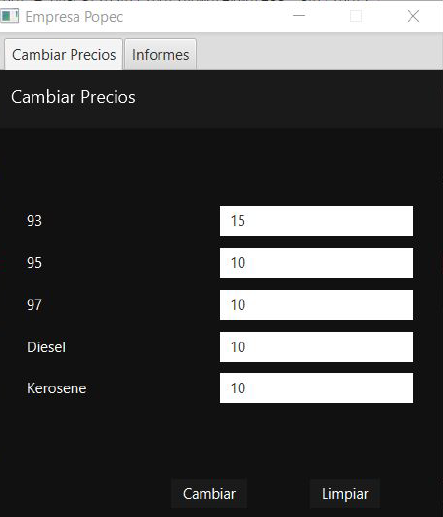


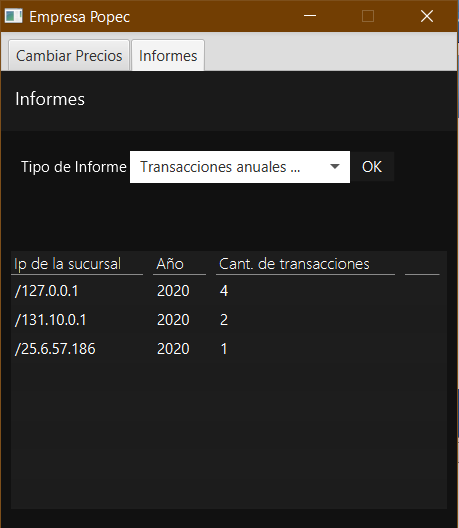
**Figura 10:** Modificación de la variable host con la ip del Servidor en la aplicación Sucursal

1. Una vez hecho lo anterior se procede a ejecutar el proyecto **“sucursal”** y establecerá conexión con el servidor, siempre y cuando hamachi se encuentre bien configurado (Computadores conectados en red).
2. Es necesario desactivar el firewall para que la conexión se pueda realizar. Para comprobar si está bien hecha o no, basta con hacer ping entre computadores utilizando las IP otorgadas por hamachi.

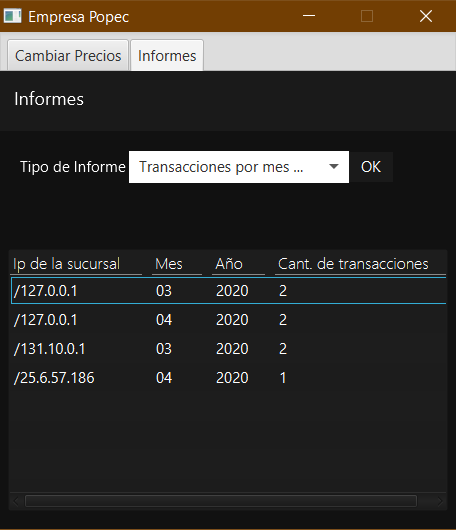
## Interfaz

  
**Figura 11:** Interfaz Sucursal

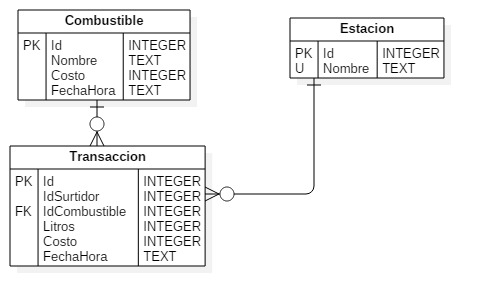
  
**Figura 12:** Interfaz Empresa



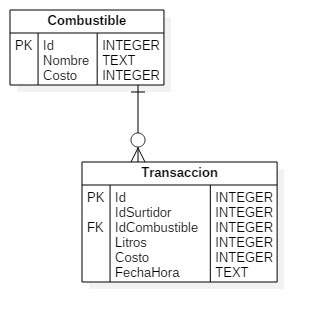
**Figura 13:** Interfaz empresa, Informes 1.



**Figura 13:** Interfaz empresa, Informes 2.



**Figura 14**: Modelo de Datos Empresa



**Figura 15**: Modelo de Datos Sucursal.