Universidad de Costa Rica

Facultad de Ingeniería

Escuela de Ciencias de la Computación e Informática

Curso: CI-1320 – Redes de Computadoras

Profesora: Gabriela Barrantes

Proyecto: Fase 2

Estudiantes:

B53779 - Fabián Leandro Flores

B50761 - Carlos Azofeifa Aguirre

Fecha de entrega:

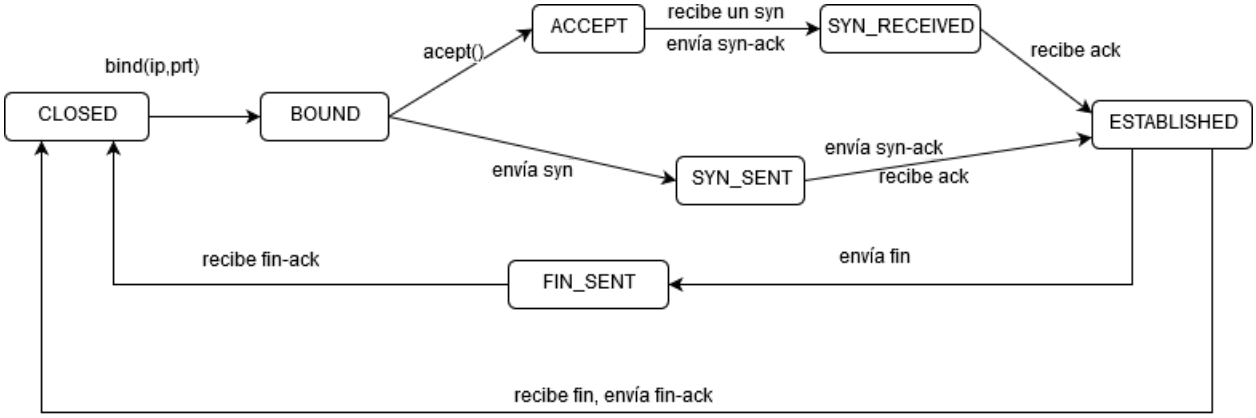
Lunes 29 de octubre 2018

1. Dependencias

Esta solución está implementada en Python 3, y para su correcta ejecución se debe tener una versión de python igual o superior a Python 3.6.

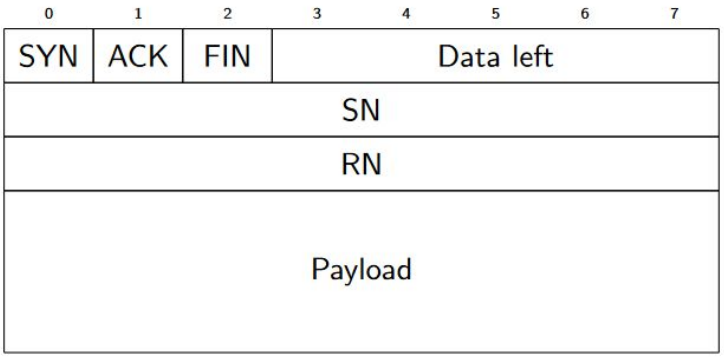
2. Diagrama de estados

La solución utiliza el siguiente diagrama de estados para crear y cerrar una conexión. Esto es un *three-way handshake* para el inicio de una conexión y un *two-way handshake* para el cierre de la conexión. Si bien el diagrama no incluye timeouts, cada estado tiene mecanismos para manejarlos.

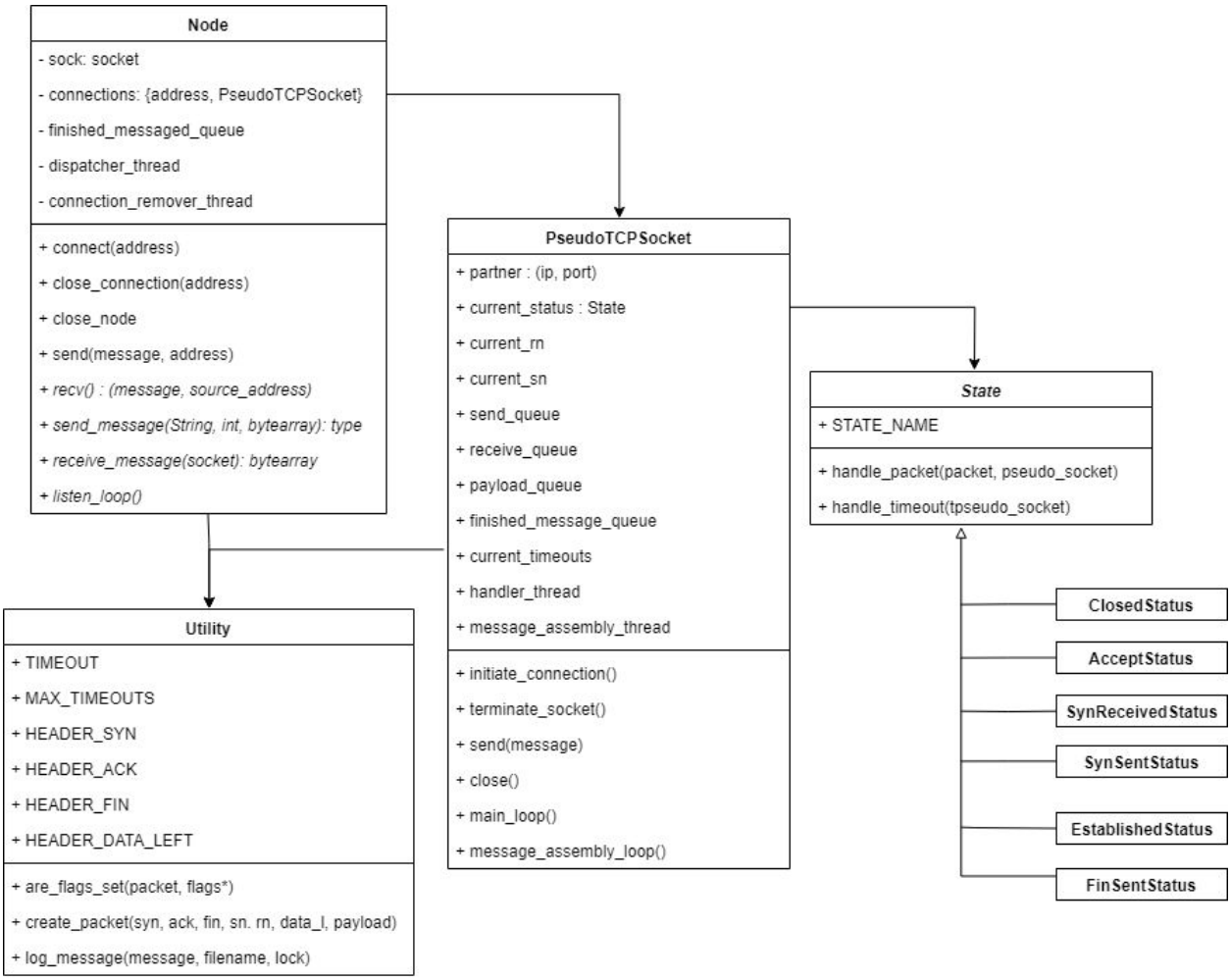


3. Decisiones de Diseño

1. Estructura del paquete: 0-30 bytes de payload, 1 bit de SYN, 1 bit de ACK, 1 bit de FIN, 5 bits que indican la cantidad de datos faltantes, 1 byte de SN, y 1 byte de RN.



2. Dentro del paquete no se envía la dirección y puerto del origen. En esta solución se asume que todos los nodos siempre usan la función bind(), y utilizan un único socket para realizar sus operaciones.
3. Mecanismo de conexión similar al *three-way handshake* de TCP:
    a. El nodo que inicia la conexión elige un SN al azar [0-255], y envía un paquete con este SN y el bit de SYN encendido, todo lo demás apagado.
    b. El segundo nodo recibe esta conexión, incrementa SN y lo toma como RN, elige un SN al azar [0-255], enciende los bits de SYN y ACK, y envía un paquete con esta información.
    c. Finalmente el primer nodo recibe este paquete, incrementa el SN recibido, lo toma como su SN, toma el RN recibido como su SN, y apaga el bit de SYN, dejando el bit de ACK encendido, y envía este paquete
4. Como timeout se eligió un tiempo de 5 milisegundos. Al ejecutar el comando ping entre dos computadoras en la misma red se observó un RTT máximo de 2ms, por lo que 5ms no debería ocasionar timeouts y aún así es un tiempo bastante pequeño.



5. Se utiliza un diseño por objetos (ilustrado arriba). Se definen las siguientes clases:

    a. Node: contiene N conexiones a otros nodos. Se encarga de escuchar mensajes en un socket y con un thread dispatcher envía los paquetes recibidos a la conexión apropiada para que lo maneje. Cada conexión es representada con un objeto PseudoTCPSocket.
    b. PseudoTCPSocket: representa una conexión en el nodo, con todas las variables de estado para manejar el algoritmo Stop and Wait.
    c. State: representa un estado de la máquina de estados. Por cada estado hay una clase que hereda de state y contiene la lógica que debe ejecutarse en caso de que se reciba un paquete o suceda un timeout.

6. Para el control del flujo de PseudoTCPSocket se utilizan 3 colas: **mensajes recibidos**, **mensajes procesados y listos para ser leídos**, y una de **mensajes por ser enviados**. De esta manera se generaliza la lógica en ambos nodos para conseguir que el enlace sea *full dúplex*. Basta con tener un hilo que ingrese los paquetes recibidos a la cola de mensajes, y otro que maneja el loop principal, el cual consiste en atender los siguientes eventos:
    a. Procesar un mensaje de la cola de mensajes: se determina si contiene un ACK, si es así se actualiza SN. Adicionalmente, si contiene datos se actualiza RN y se ingresan los bytes a la cola de mensajes procesados.
    b. Timeout: se reenvía el último mensaje para el cual no se ha recibido un ACK.
7. Además, todas las conexiones de un nodo comparten una cola de mensajes terminados. PseudoTCPSocket contiene un hilo que consume la cola de mensajes procesados para obtener los fragmentos. Cuando un se termina de recibir un mensaje, este hilo envía el mensaje terminado al nodo mediante la cola compartida, de manera que el nodo pueda entregar el mensaje a la capa superior.
8. Para indicar que un mensaje finalizó, el hilo principal de PseudoTCPSocket agrega un mensaje especial a la cola de mensajes procesados al leer un paquete con menos de 30 bytes de datos.
9. Node cuenta con un hilo para cerrar asincrónicamente las conexiones. Cuando un nodo se va a cerrar asincrónicamente (debido a que recibió un FIN o porque se excedió el máximo de timeouts, y no porque se llamó close) notifica a este hilo para que elimine su entrada en la tabla de conexiones.