Universidad de Costa Rica

Escuela de Computación e Informática

CI-1320
Curso de Redes

Fase II del Proyecto

Estudiantes:
Fabián Alvarez Chévez
carné: B60359
Alejandro Chacón
carné: B51867

Profesor:
Gabriela Barrante

30 de octubre de 2018

UNIVERSIDAD DE
COSTA RICA

1. Diagrama de la máquina de estados

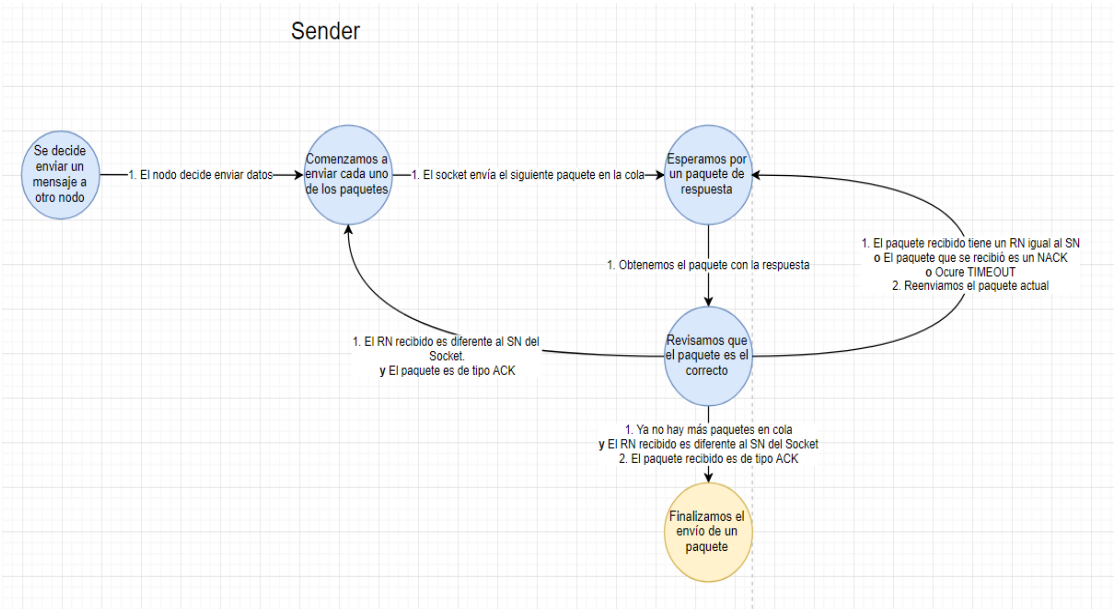

Figura 1: Nodo que envía paquetes

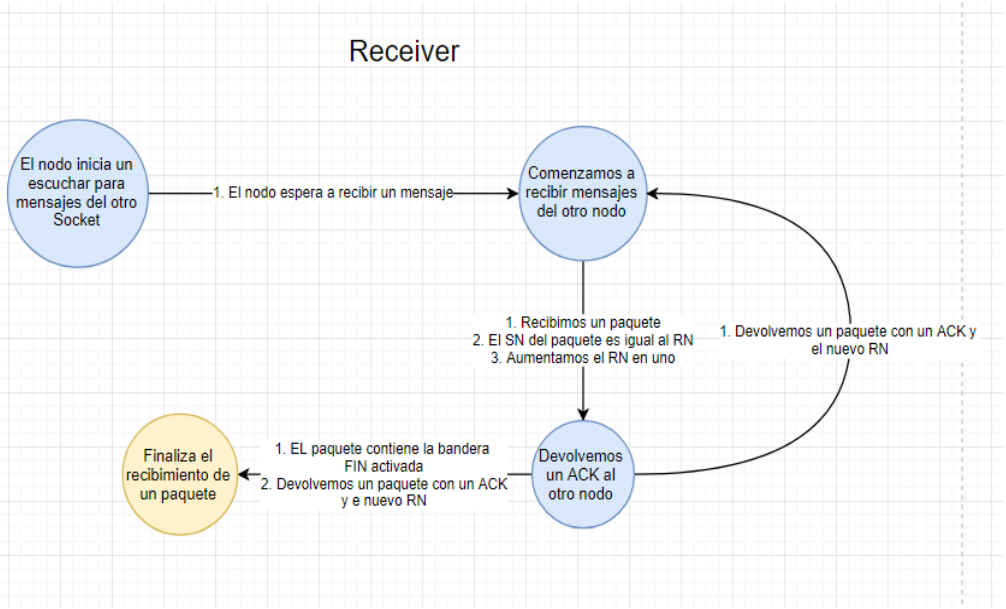

Figura 2: Nodo que recibe paquetes

2. Descripción de todas las decisiones de diseño

- La bitacora la manejamos mediante una librería de python, la cual nos permite crear y escribir en un archivo de texto. Se decidió que la bitacora trendrá los mensajes de inicio de handshake del cliente, el ACK del server, la respuesta de parte del cliente y por último, la confirmación de que el handshake se realizó correctamente en el server. Además también registramos cada mensaje que se envía y su respectivo ACK. En el caso que un paquete se pierda lo registramos en la bitacora y continua el proceso de pasar el mensaje. Por último cuando el mensaje terminó de enviarse el server registra en la bitácora lo que recibió.

- La probabilidad de pérdida de paquetes la escogimos con base en varias pruebas realizadas, donde revisamos la bitácora con distintos valores y observamos la cantidad de veces que se perdía un paquete. Por lo tanto decidimos escoger 0.15, para no estar perdiendo constantemente tantos paquetes, pero para tener las suficientes pérdidas para realizar pruebas y ver el comportamiento al reenviar un paquete.

- La estructura de datos que utilizamos para guardar los paquetes fue una cola, esto por el hecho de que es una estructura que ya es **thread save** por lo que no hay que preocuparse en el uso de candados cuando se modifica la cola, otra razón por la que se utilizó es porque ya contiene un método con el que se puede ejecutar el *timeout* sin necesidad de uno preocuparse por ello. Por lo tanto a la hora de que un método solicita algún paquete de la cola, sí el paquete no llega este no se va a quedar esperando para siempre sino que después de cierto tiempo continua. Para finalizar cuando se ejecuta el método para obtener un dato de la cola este se queda esperando por lo que no ejecuta ninguna otra linea de código a menos de que le haya llegado un dato o se haya ejecutado el *timeout*.

- Para controlar la forma en la que el despachador distribuye los paquetes se decidió implementar 3 colas diferentes, una para recibir paquetes que contienen datos a esta le llamamos **messageMailbox**, otra es para recibir paquetes que contienen un ACK a esta cola le llamamos **ackMailbox** y para finalizar le agregamos una tercera cola que se utiliza para el manejo de conexiones, es decir cualquier mensaje que involucra crear o cerrar una conexión se agrega a esta cola, la cual se llama **connMailbox**.

- Para saber el estado de un *Socket* agregamos tres variables cada una de ellas sirve para poder completar alguno de los métodos que ejecuta cada *Socket*. La primera

variable se llama **alive**, con ella sabemos sí un nodo puede seguir recibiendo nuevas conexiones y paquetes o no. La segunda variable es **connected** la cuál se utiliza cuando un nodo está realizando el *handshake*, como este no se ha terminando la conexión no está completamente establecida, esto nos permite identificar las nuevas conexiones rápidamente. La última variable es **closing** y se utiliza para identificar cuando un *Socket* se está cerrando, de forma que sea más simple identificarlo.

- Una decisión de diseño importante fue la selección de cuales métodos del Socket TCP íbamos a implementar, en este caso lo que nosotros decidimos fue utilizar todos los métodos que el nodo TCP de la primera fase utilizaba, en otras palabras el código que nosotros utilizamos en la segunda fase fue básicamente copiar y pegar el de la otra fase, solo que en vez de utilizar el Scket TCP, utiliza nuestro nodo pseudo TCP. La razón de esto es que nos ayudo a entender mejor la forma de implementar todos los métodos que se necesitaban para poder tener un protocolo de transporte confiable.

3. Requerimientos adicionales

- El primer requerimiento es el formato en el que el programa debe recibir el paquete. Nosotros asumimos que el orden en que vienen los datos es el siguiente **Ip origen, Puerto origen, Ip destino, Puesrto destino, Syn, RN, SN, Ack, Fin, Relleno, Mensaje**. Además del orden el tamaño es otro aspecto que se debe considerar, el tamaño de cada una de las partes del paquete es la siguiente **4 bytes, 2 bytes, 4 bytes, 2 bytes, 1 bit, 1 bit, 1 bit, 1 bit, 1 bit, 3 bits, 1 byte** Para finalizar el último requerimiento que está relacionado al paquete es que todo debe ser recibido en bytes y debe ir acomodado en *big endian*.

- Un *socket* en el nodo no puede estar comunicándose con dos o más *sockets* de otro nodo, sí un nodo quiere tener dos comunicaciones con un mismo nodo, debe generar dos *sockets* y cada uno de estos se debe comunicar con un *socket* diferente.