Universidad de Costa Rica

Escuela de Ciencias de la Computación e Informática

CI-1320 Redes de Computadoras

Grupo 1

Prof. Gabriela Barrantes

Fase 2 del proyecto (PseudoTCP)

BackNet Boys:

Jefferson Alvarez Bonilla, B61144

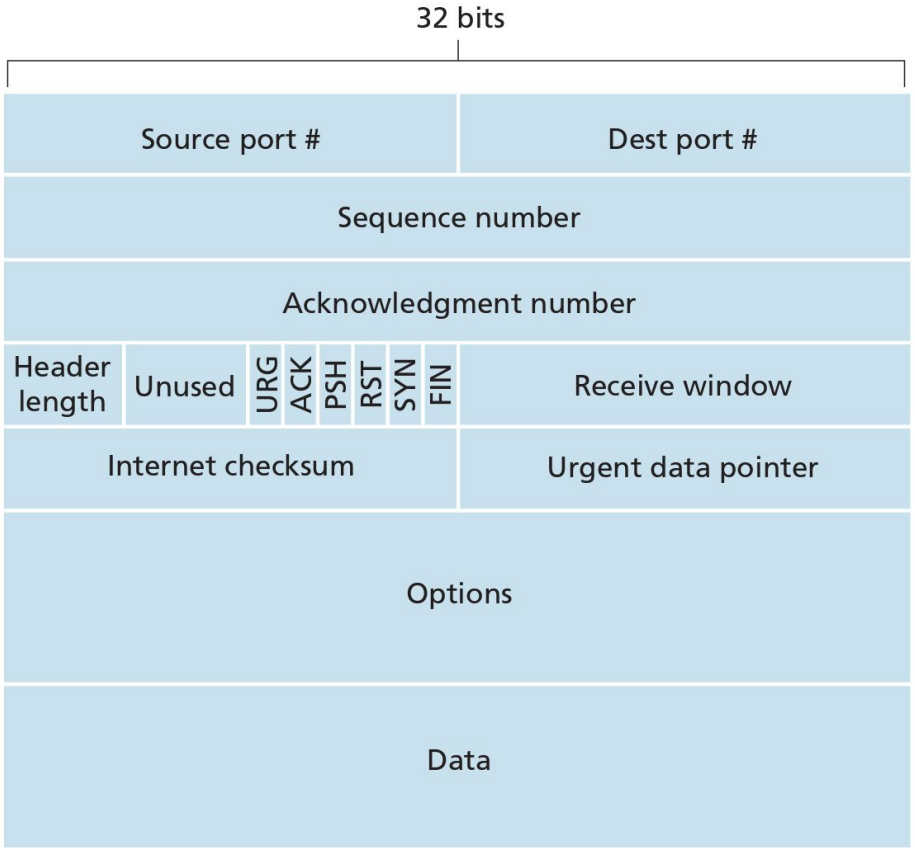
Walter Bonilla Jimenez, B60380

Luis Porras Ledezma, B65477

II Semestre 2018

Decisiones importantes de diseños tomadas:

1. La principal decisión de diseño tomada fue la forma en la que se decidió resolver el problema planteado, para esto se decidió crear una clase “socketPseudoTCP” que encapsula al socket UDP e incorporaría todos lo métodos que normalmente tiene un socket de tipo TCP(send, recv, bind, listen, accept, connect y close), permitiéndole así comportarse como TCP (con Stop and Wait cómo ARQ). Esto se decidió así pues nos parecio la forma más natural de resolver el problema.
2. Otra decisión importante de diseño fue el decidir manejar en forma separada el id de la conexión (es decir el número de puerto del emisor o receptor) del SN y RN, a diferencia de TCP que unifica ambos conceptos. Esto se desicion pues se querían manejar SN y RN con 0 o 1.
3. Como se mencionó en el punto anterior se tomó la decisión de utilizar los valores 0 y 1 como los únicos posibles para los SN y RN. Esto se decidió aprovechando que el ARQ a utilizar era Stop and Wait por lo que estos valores serían más que suficiente.
4. La estructura del segmento que se decidió usar fue: [puerto origen] = 2 bytes, [puerto destino] = 2 bytes, [SN] = 1 byte, [RN] = 1 byte, [tamaño del header] = 1 byte, [banderas(SYN, ACK y FIN)] = 1 byte, y finalmente [data] = 8 bytes. Esto se decidió de esta forma siguiendo la figura del libro que mostraba cómo está compuesto el header de TCP, la cual se muestra a continuación.



5. Otra decisión importante fue el uso de la clase queue.Queue de Python para manejar las colas de mensajes recibidos de los sockets. Se decidió utilizar esta clase pues esta está recomendada a la hora de trabajar con threads, pues esta incorpora un candado

de forma automática, que bloquea la cola en operaciones de get y put, además esta clase posee un timeout implementado, el cual fue el que se aprovechó para realizar los timeouts necesarios en el protocolo Stop and Wait.

6. Otra decisión tomada fue el uso de un diccionario de la std de Python para manejar la estructura con los connection sockets que devuelve el método accept del lado servidor, esta estructura utiliza como llave la ip y el puerto del emisor (origen), esto pues el principal propósito de esta estructura ayudar en la demultiplexación de mensajes.

Requerimientos adicionales descubiertos durante la programación:

Al principio nos dimos cuenta que para la transmisión confiable lo mínimo que se necesita son:

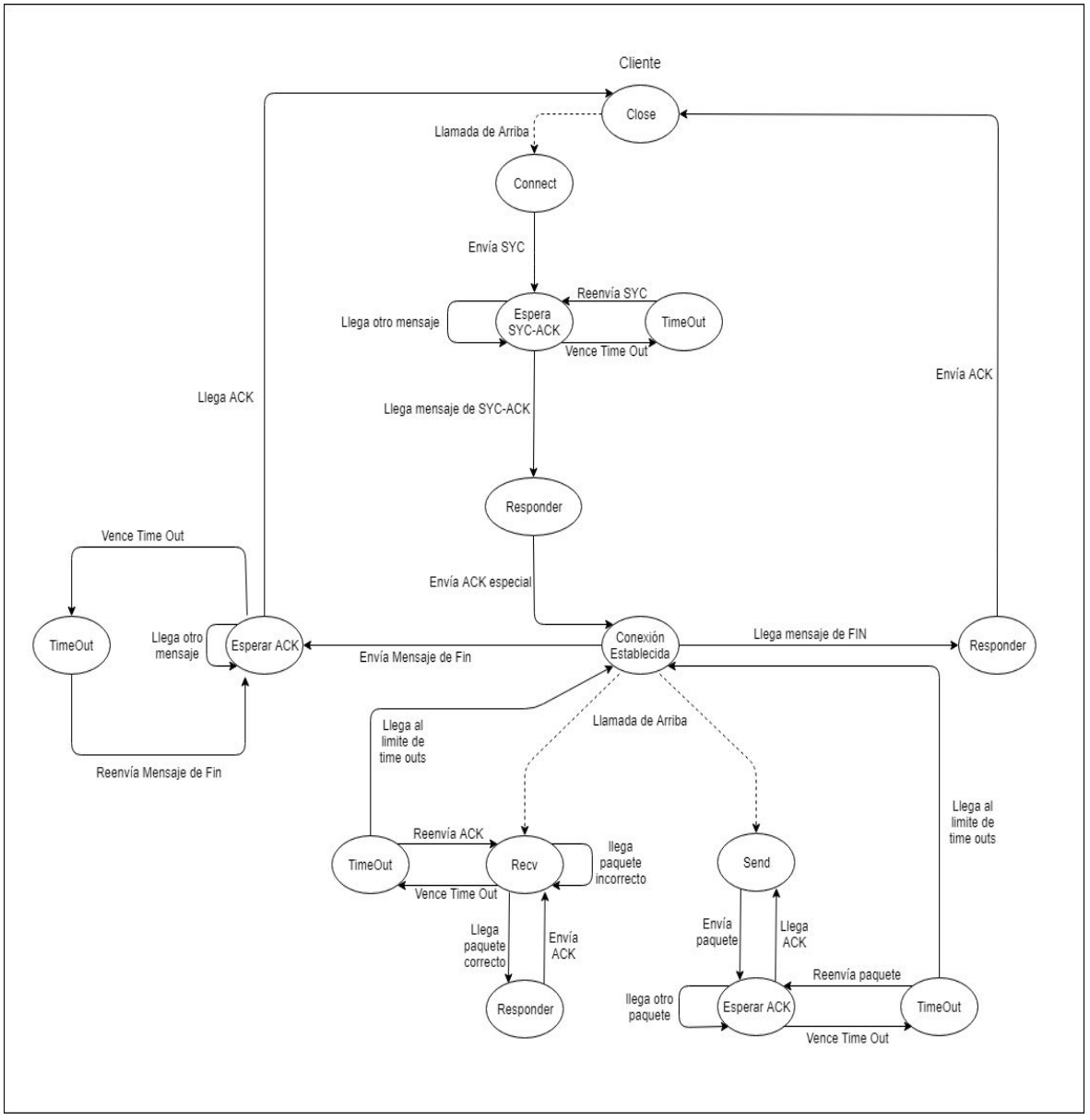
- Banderas de SYN, ACK y FIN: estas con el fin de establecer conexiones y cerrarlas.
- Identificar las conexiones: no dimos cuenta que era necesario guardar el número de puerto e IP para identificar con quien nos comunicamos.
- Manejo de concurrencia sobre las estructuras compartidas: gracias a que elegimos la clase Queue de python pudimos solucionar el tema de la concurrencia bastante fácil pues es una estructura que ya posee un lock interno.
- Manejo del “time out”: de la misma forma, la estructura Queue de python nos brindo una forma fácil de poder manejar el “time out” ya que nos brinda la posibilidad de usar uno.

Luego al avanzar en el proyecto nos dimos cuenta que teníamos algunos problemas al diferenciar archivos, puesto que no teníamos forma de identificar el inicio y fin de un archivo. Por lo tanto, tuvimos que agregar:

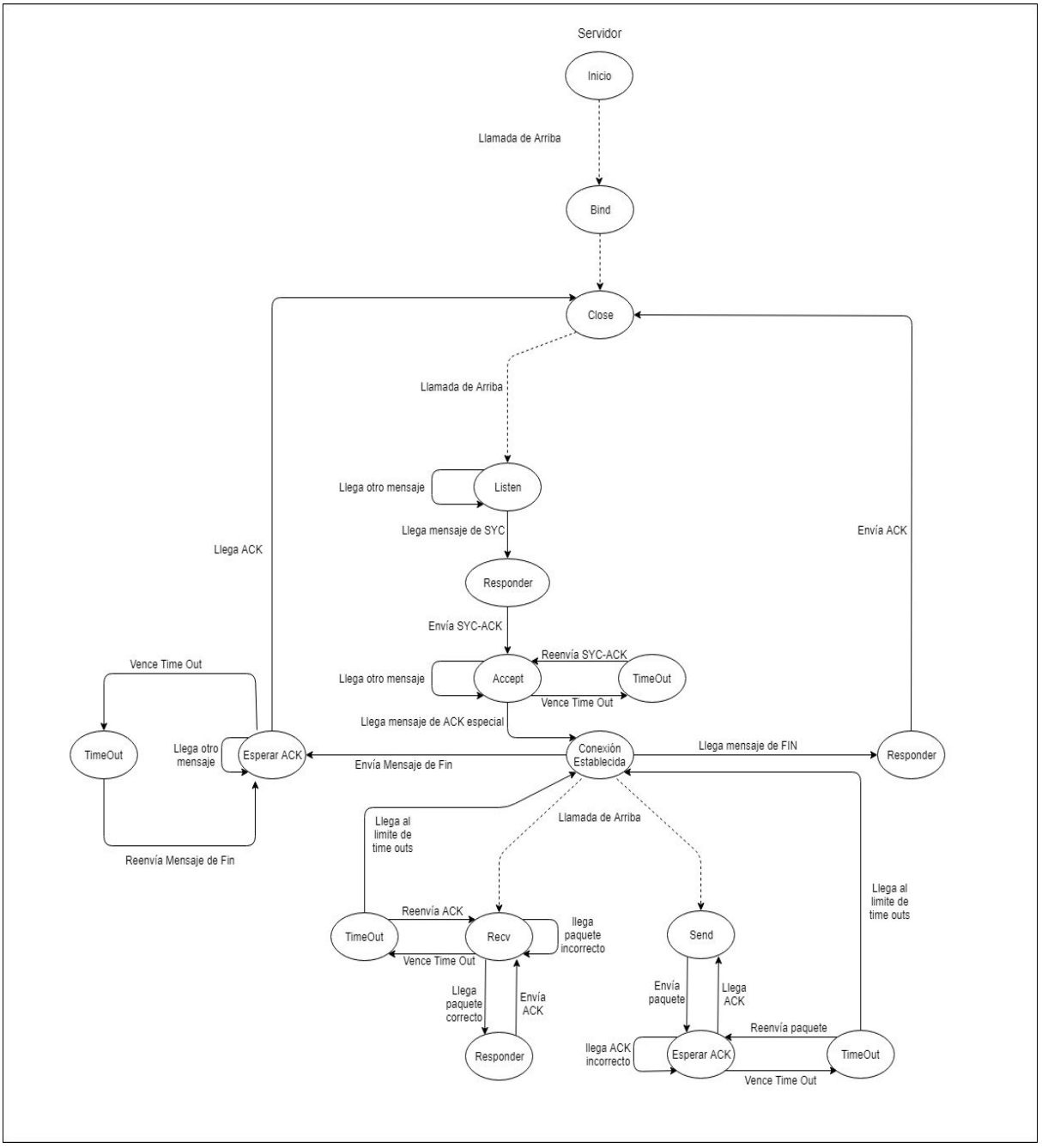
- Banderas demarcadoras de inicio y fin: estas banderas denominadas START y END nos ayudaron a solucionar el problema a la hora de identificar el inicio y fin de los archivos.

Máquinas de Estado:

Cliente



Servidor


Servidor
Inicio
Llamada de Arriba
Bind
Close
Llamada de Arriba
Llega otro mensaje
Listen
Llega mensaje de SYC
Llega ACK
Envía ACK
Responder
Envía SYC-ACK
Reenvía SYC-ACK
Llega otro mensaje
Accept
TimeOut
Vence Time Out
Vence Time Out
Llega mensaje de ACK especial
TimeOut
Llega otro mensaje
Esperar ACK
Envía Mensaje de Fin
Conexión Establecida
Llega mensaje de FIN
Responder
Reenvía Mensaje de Fin
Llega al limite de time outs
Llamada de Arriba
Llega al limite de time outs
Reenvía ACK
TimeOut
Recv
llega paquete incorrecto
Vence Time Out
Send
Llega paquete correcto
Envía ACK
Envía paquete
Llega ACK
Reenvía paquete
Responder
llega ACK incorrecto
Esperar ACK
TimeOut
Vence Time Out