ciTCP

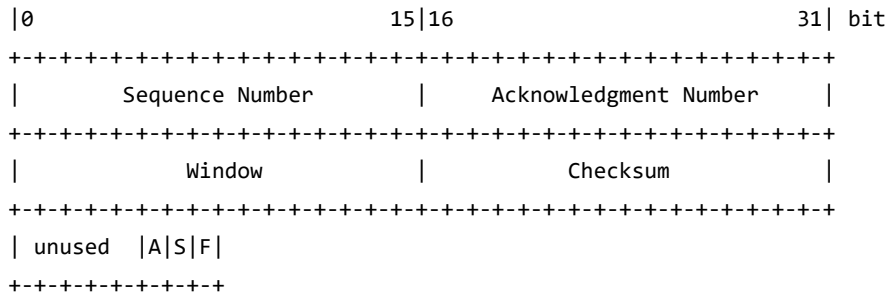
ciTCP is a client and server program that creates a reliable transport ontop of UDP, which is normally unreliable.

Principles of Reliable Transport

ciTCP will have the following features:

- Connection establishment
- Connection tear down
- Acknowledgement and sequence numbers
- Timeouts
- TCP Tahoe congestion control (slow start, congestion avoidance, and fast recovery)
- Receive and sending windows

ciTCP Header

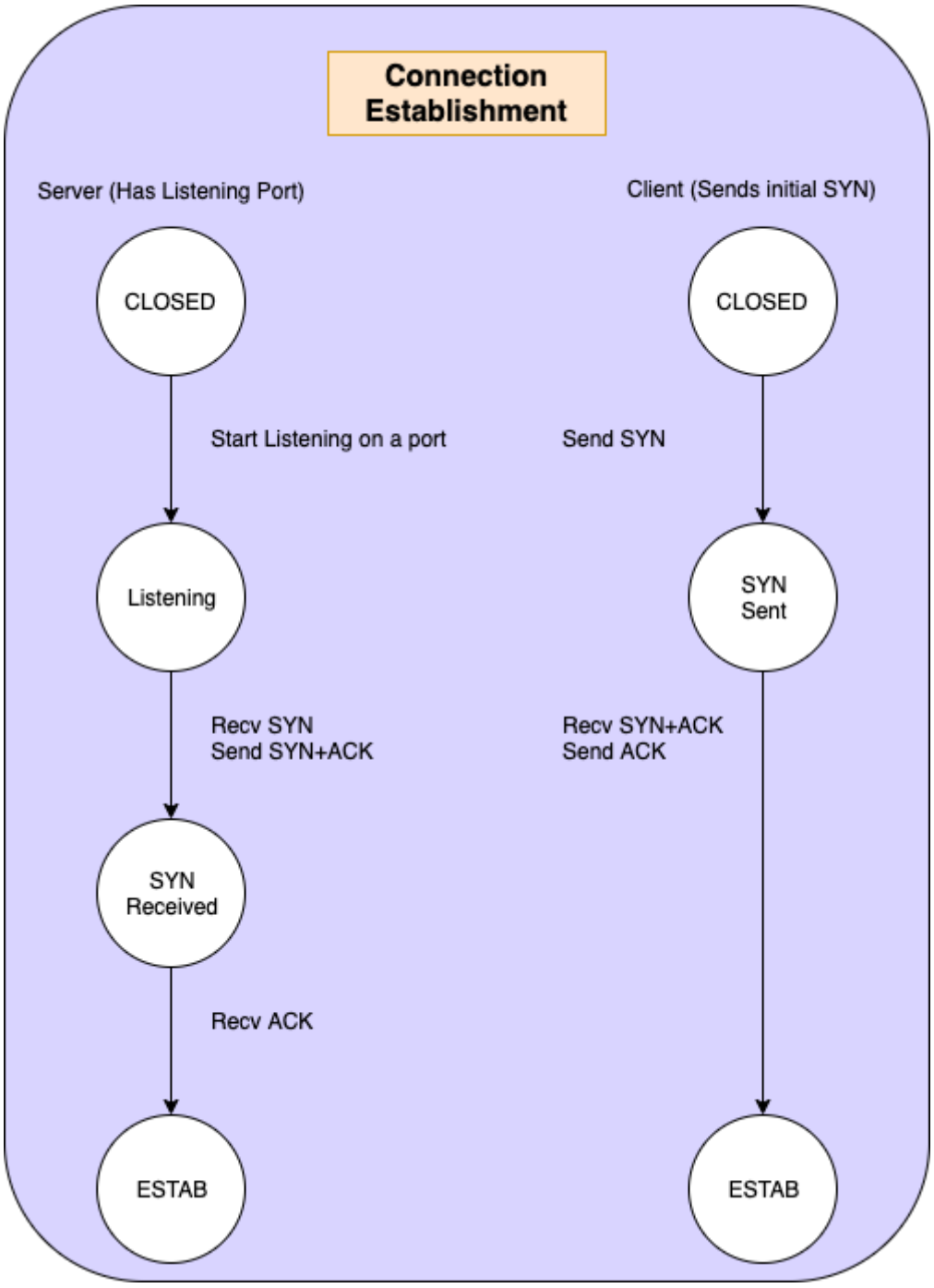


Connection Establishment

The connection establishment is the three way handshake.

1. The server listens on a port of an interface.
2. The client sends the server an SYN message ()
3. The server receives the SYN message and responds with a SYN+ACK
    - Acknowledging the SYN and sending it's own SYN
4. The client recieves the SYN+ACK and sends its own ACK
5. Both parties are connected

For the specific states and transitions refer to this diagram



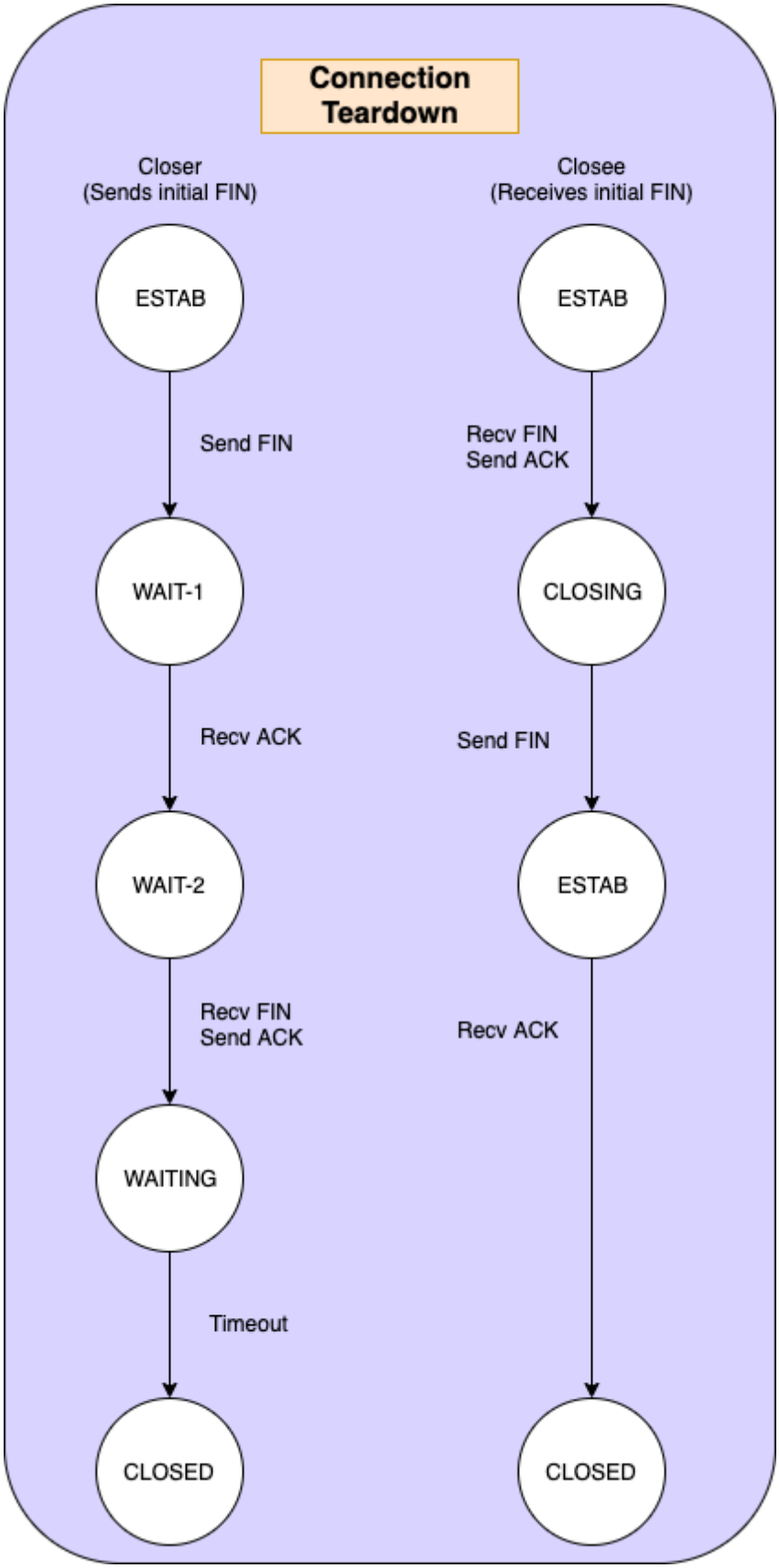
Connection Tear Down

The connection teardown is both parties finishing their connections and returning to a closed state

1. The closer sends the closee the FIN message
2. The closee receives the FIN message sends the closer an ACK message
    1. The closee then performs whatever actions it needs to close the connection
3. The closee then sends the FIN message to the closer
4. The closer receives the FIN message and responds with an ACK message
5. With both sides having sent and received a FIN and an ACK they wait in a timeout as to not have issues with late packets

6. The connection is now closed on both sides

For the specific states and transitions refer to this diagram



Acknowledgement and sequence numbers

The sequence and acknowledgement numbers are created by random number generation from their respective owning programs before the three way handshake starts. The sequence number is there to help ordering of packets. The acknowledgment number is there to tell the receiver that a segment of packets has been received.

Durring the three-way handshake the server's initial sequence number is established.
The receiver(client) does not send data (only control messages), so a sequence number from the receiver should not be sent.

Timeouts

All packets that expect responses will have a timeout for that response. Durring transmission, if the response times out, then we send that packet again and start a new timer. If the response is corrupted, we wait for the timeout to resend the message.

The timeout is ideally the same as the Round Trip Time or RTT.
The RTT is calculated by Exponential Weighted Moving Average or EWMA.

$$EWMA = EstRTT + 4 \cdot DevRTT$$

$$EstRTT = \left[(1-\alpha) \cdot EstRTT\right] + \left[\alpha \cdot SampleRTT\right]$$

$$DevRTT = \left[(1-\beta) \cot DevRTT\right] + \left[\beta \cdot |SampleRTT-EstRTT|\right]$$

$$\alpha = 0.125 \text{ and } \beta = 0.25$$

TCP Tahoe (Reno?) congestion control

Slow Start

During this phase of congestion control, we send start off with a Congestion Window Size of 1 and add one to it every time we get an ACK, until the receiver is congested. This effectively doubles the Congestion Window size every RTT. Once the receiver is "congested", the congestion control goes into congestion avoidance mode. We know the receiver is congested when there is a timeout or the congestion window size is greater than or equal to the receiver's advertised window size.

Congestion Avoidance

The Slow Start thresh hold is determined by half the window size as we leave the Slow Start phase. Once we have the ssthresh, we increase the window size by 1 per successful RTT.

When the RTT delivery is unsuccessful we decrease the window size.
If the delivery was unsuccessful because of 3 Duplicate acknoledgments, then we half the window size.
If the delivery was unsuccessful because of a timeout, we set the window size to one.
If the delivery was unsuccessful because of a timeout and the window size is one we double the timeout.

Fast Recovery

This is the act of skipping a second Slow start phase at the beginning of congestion avoidance and immediately moving onto the linear increase of window size.

The rate at which Messages are sent at is defined as:

$$Rate = (cwnd * MSS)/RTT \text{ as } \text{Bytes}/\text{Sec}$$

MSS = Maximum Segment Size

Receive and sending windows

Reveiving and Sending windows are the amount of data that a network can handle at any given time and are set by congestion control.

Some things To Consider

- How is the initial sequence number chosen?
  - By a random number generator created on each side of the connection independently.
- Does the timeout value have to change?
  - Yes, when congestion control determines that timeout needs to change, it doubles.
- How are the segment sizes decided?
  - The segment sizes are not determined by any feilds and therefore can theoretically be any size. I will use a maximum size of 2^16-1 since its small enough to not transfer something like an image in 1 packet, but large enough to not need endless packets.