UDP header revisited

<table>
<tr><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td colspan="4">Version</td><td colspan="4">IHL</td><td colspan="6">DSCP</td><td colspan="2">ECN</td><td colspan="16">Total Length</td></tr>
<tr><td colspan="16">Identification</td><td colspan="3">Flags</td><td colspan="13">Fragment Offset</td></tr>
<tr><td colspan="8">Time To Live</td><td colspan="8">Protocol</td><td colspan="16">Header Checksum</td></tr>
<tr><td colspan="32">Source IP Address</td></tr>
<tr><td colspan="32">Destination IP Address</td></tr>
<tr><td colspan="32">Options (if IHL > 5)</td></tr>
<tr><td colspan="16">Source port</td><td colspan="16">Destination port</td></tr>
<tr><td colspan="16">Length</td><td colspan="16">Checksum</td></tr>
<tr><td colspan="32">Application data</td></tr>
</table>

TCP

TCP

Consists of 3 primary phases:

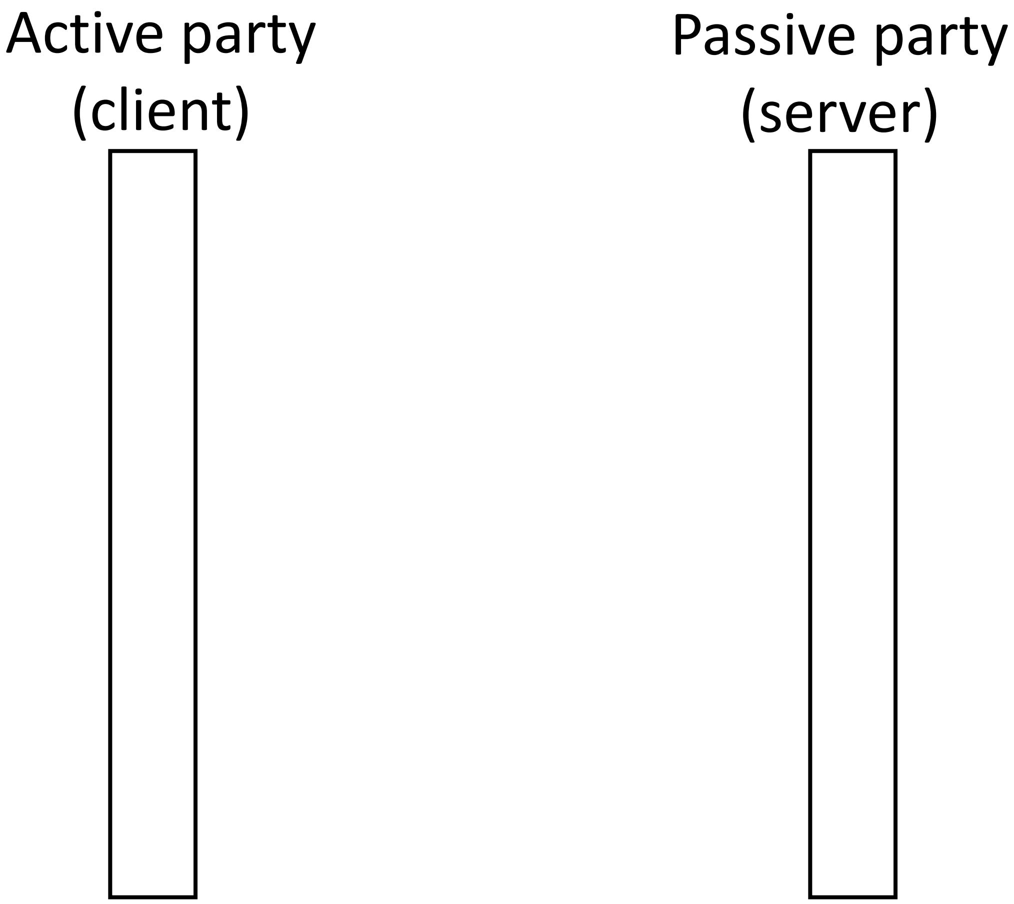
- Connection Establishment (Setup)
- Sliding Windows/Flow Control
- Connection Release (Teardown)

Connection Establishment

- Both sender and receiver must be ready before we start the transfer of data
  - Need to agree on a set of parameters
  - e.g., the Maximum Segment Size (MSS)
- This is signaling
  - It sets up state at the endpoints
  - Like “dialing” for a telephone call

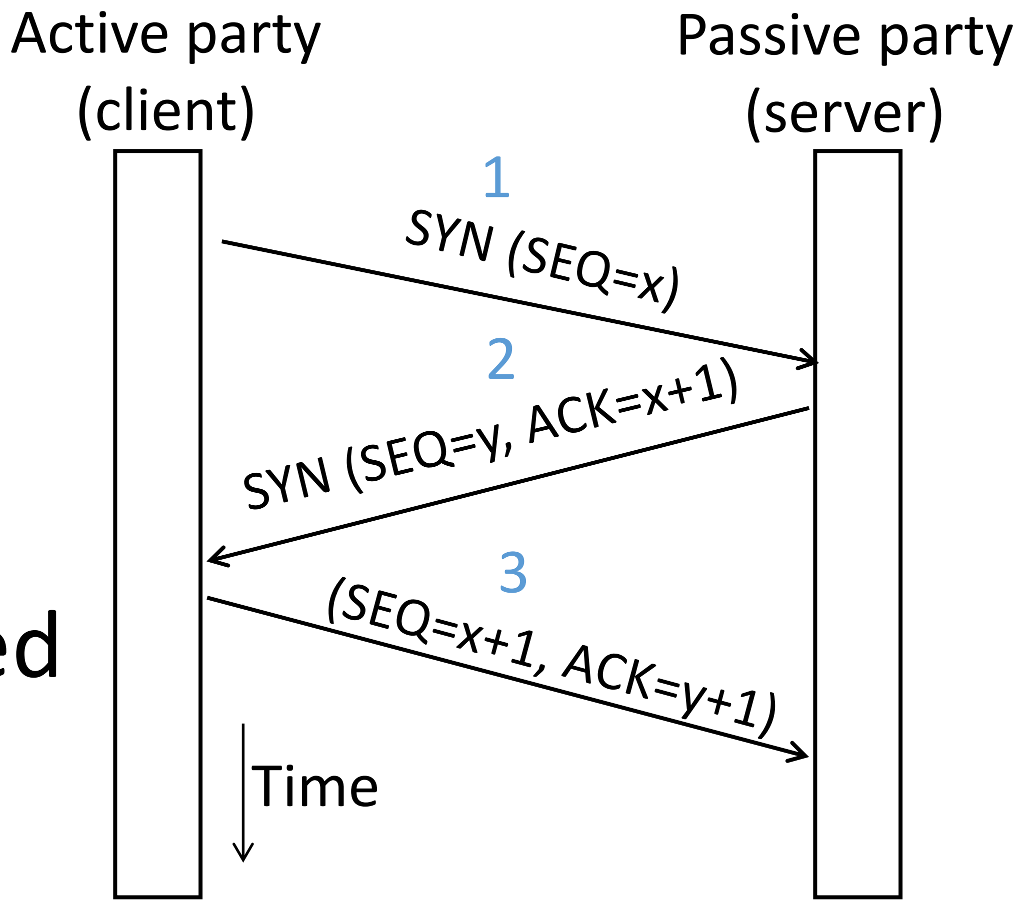
Three-Way Handshake

- Used in TCP; opens connection for data in both directions
- Each side probes the other with a fresh Initial Sequence Number (ISN)
    - Sends on a SYNchronize segment
    - Echo on an ACKnowledge segment
- Chosen to be robust even against delayed duplicates



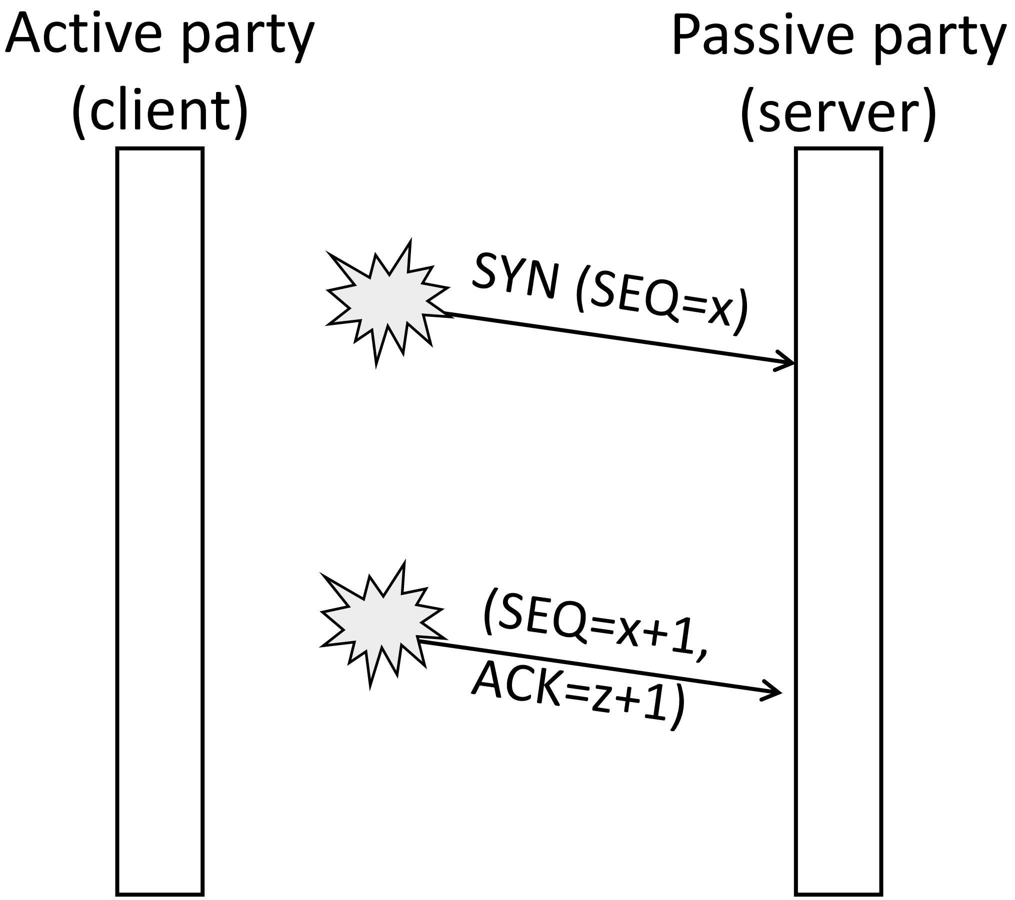
Three-Way Handshake (2)

- Three steps:
  - Client sends SYN(x)
  - Server replies with SYN(y)ACK(x+1)
  - Client replies with ACK(y+1)
  - SYNs are retransmitted if lost
- Sequence and ack numbers carried on further segments



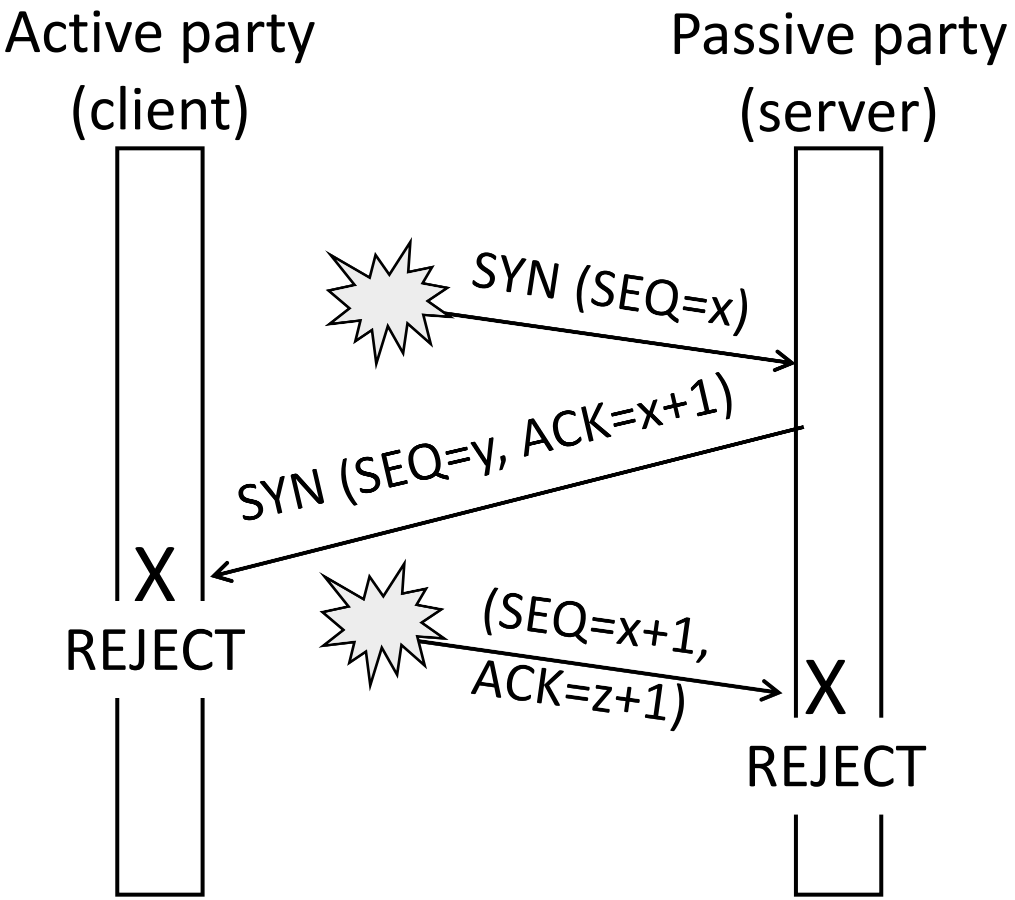
Three-Way Handshake (3)

- Suppose delayed, duplicate copies of the SYN and ACK arrive at the server!
  - Improbable, but anyhow …



Three-Way Handshake (4)

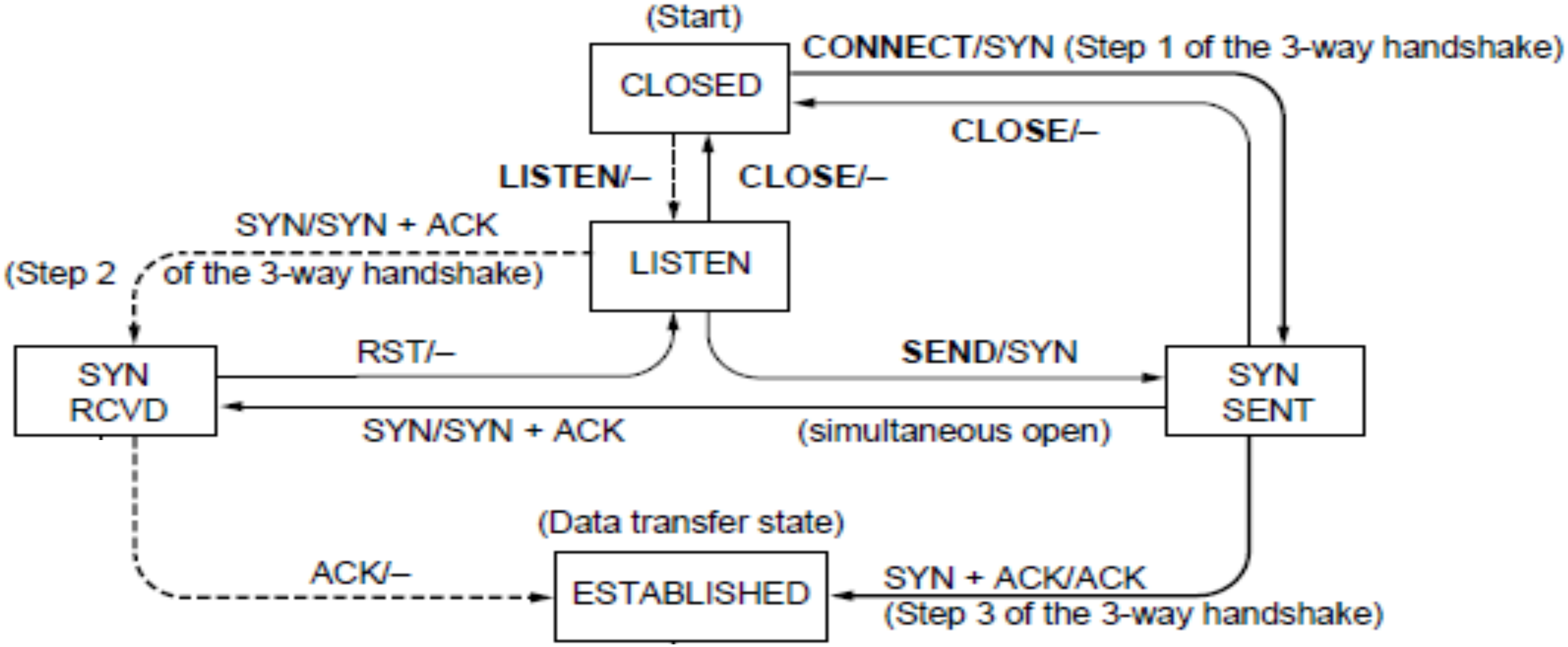
- Suppose delayed, duplicate copies of the SYN and ACK arrive at the server!
  - Improbable, but anyhow …
- Connection will be cleanly rejected on both sides ☺



TCP Connection State Machine

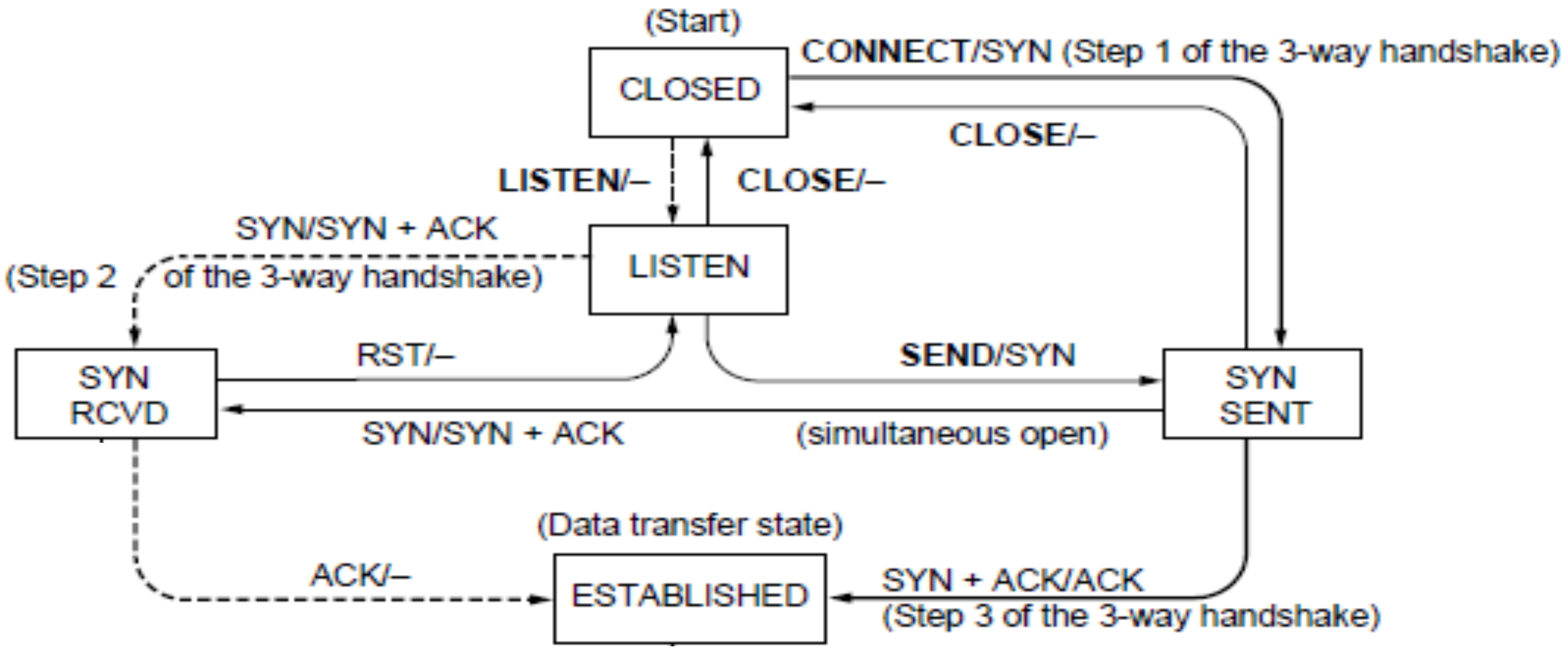
- Captures the states ([]) and transitions (->)
  - A/B means event A triggers the transition, with action B

Both parties run instances of this state machine



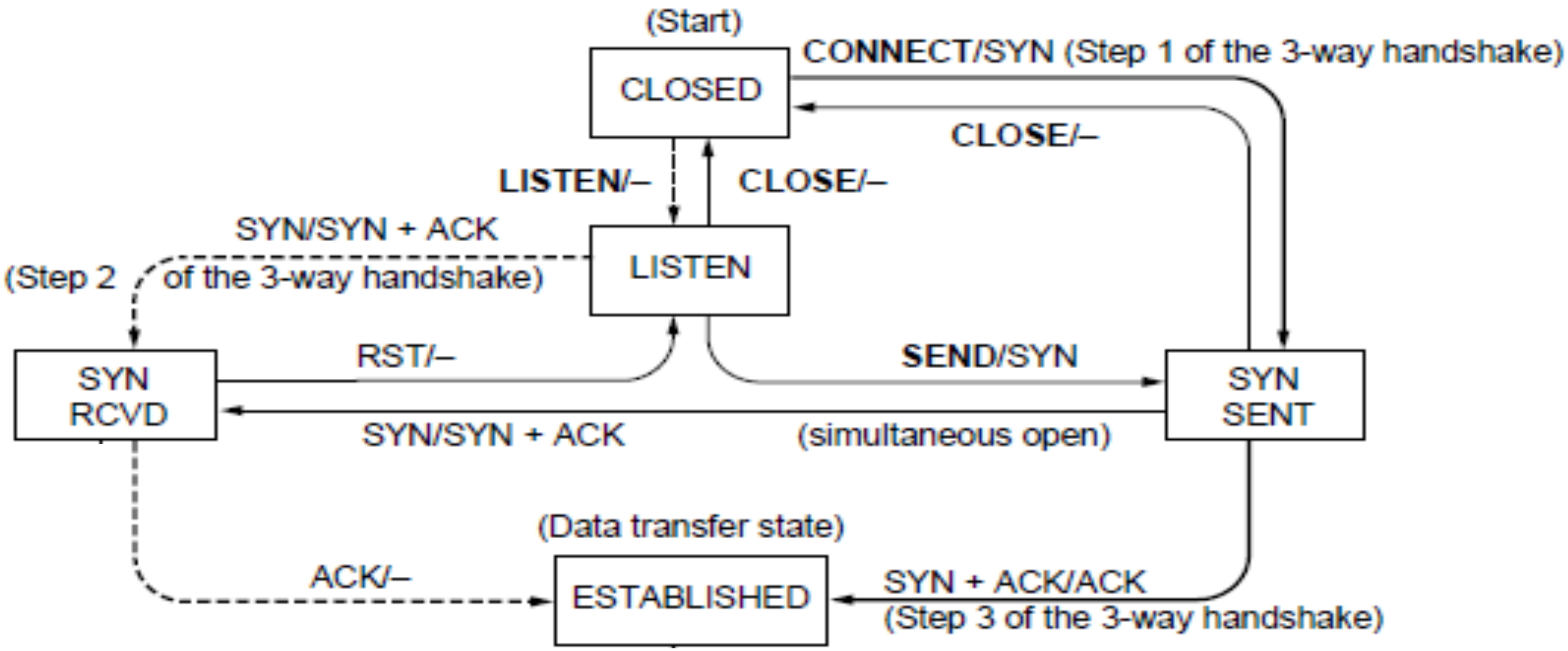
TCP Connections (2)

- Follow the path of the client:



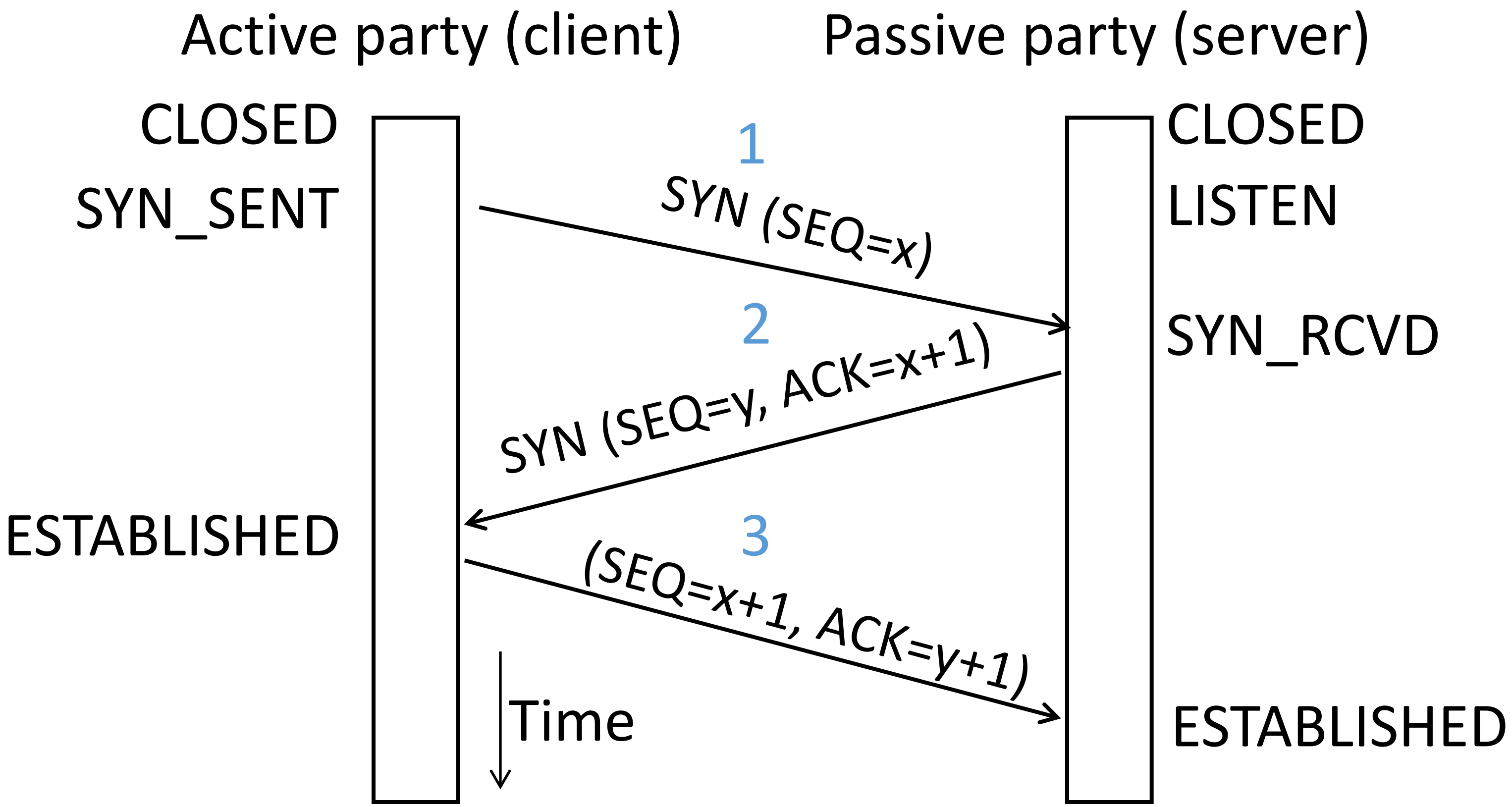
TCP Connections (3)

- And the path of the server:



TCP Connections (4)

- Again, with states …



TCP Connections (5)

- Finite state machines are a useful tool to specify and check the handling of all cases that may occur
- TCP allows for simultaneous open
  - i.e., both sides open instead of the client-server pattern
  - Try at home to confirm it works ☺

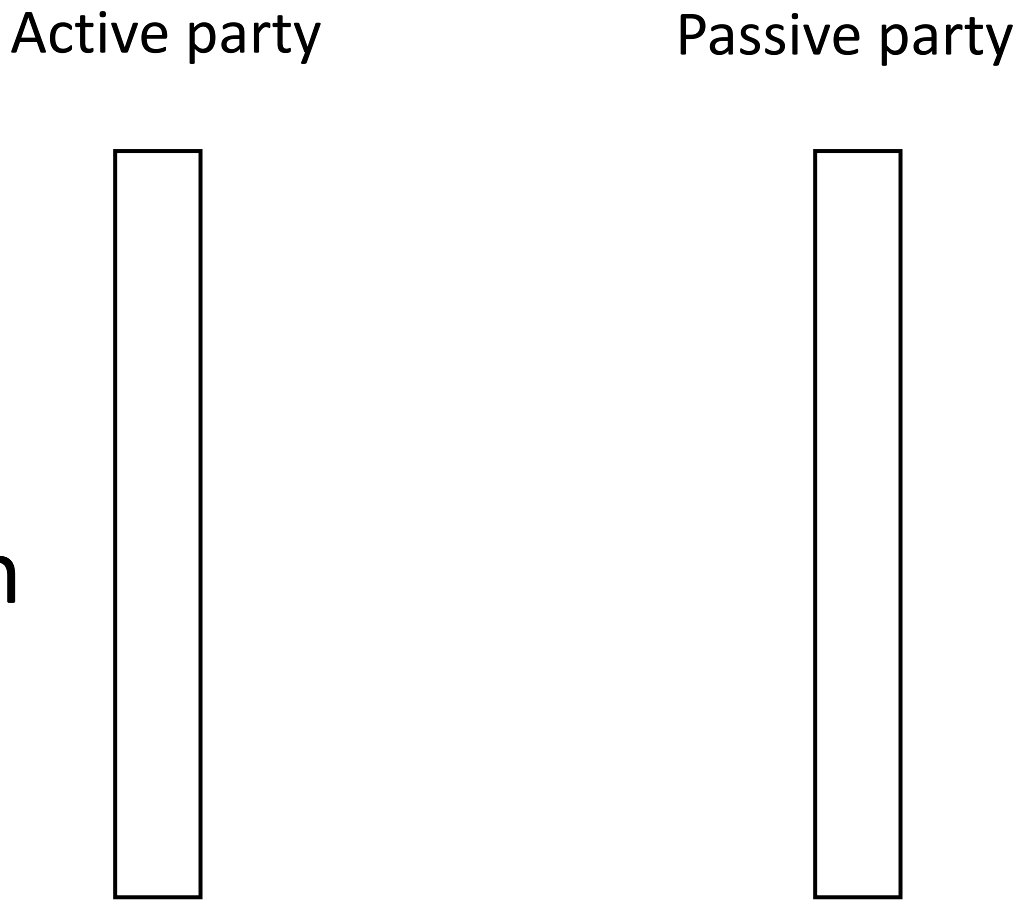
Connection Release

- Orderly release by both parties when done
  - Delivers all pending data and “hangs up”
  - Cleans up state in sender and receiver
- Key problem is to provide reliability while releasing
  - TCP uses a “symmetric” close in which both sides shutdown independently

TCP Connection Release

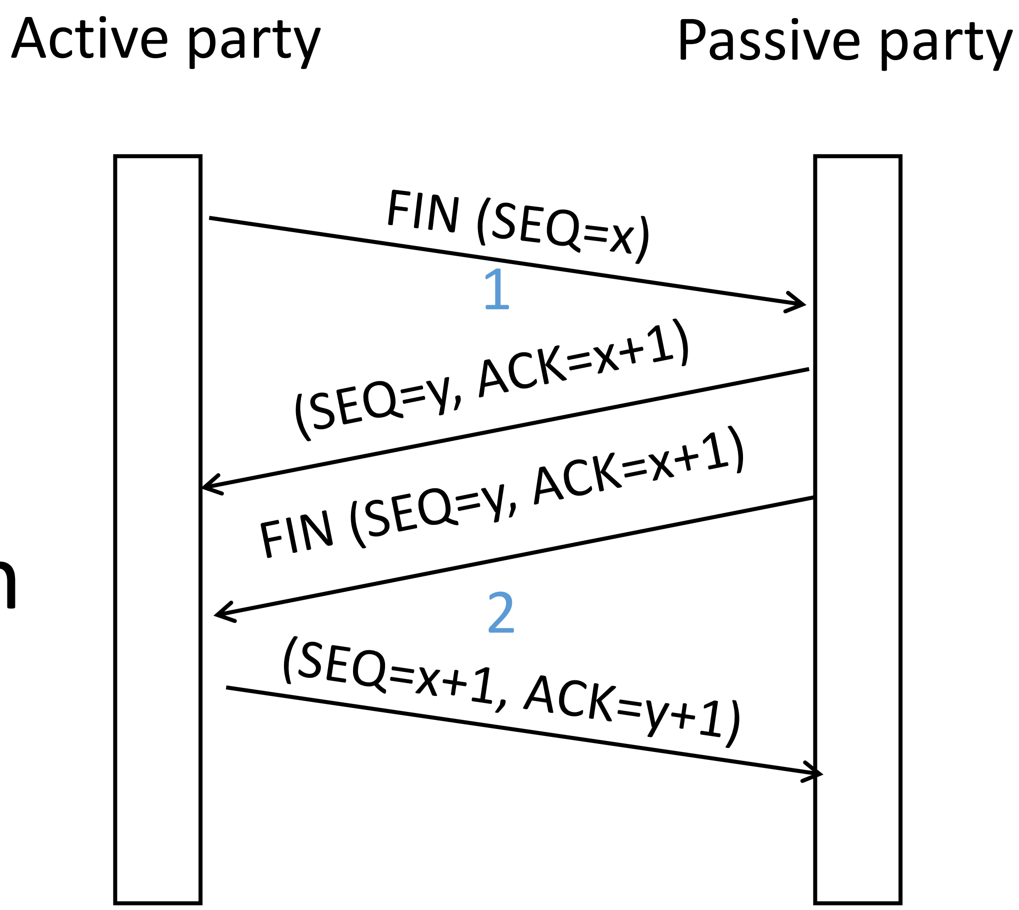
- Two steps:
  - Active sends FIN(x), passive ACKs
  - Passive sends FIN(y), active ACKs
  - FINs are retransmitted if lost

- Each FIN/ACK closes one direction of data transfer



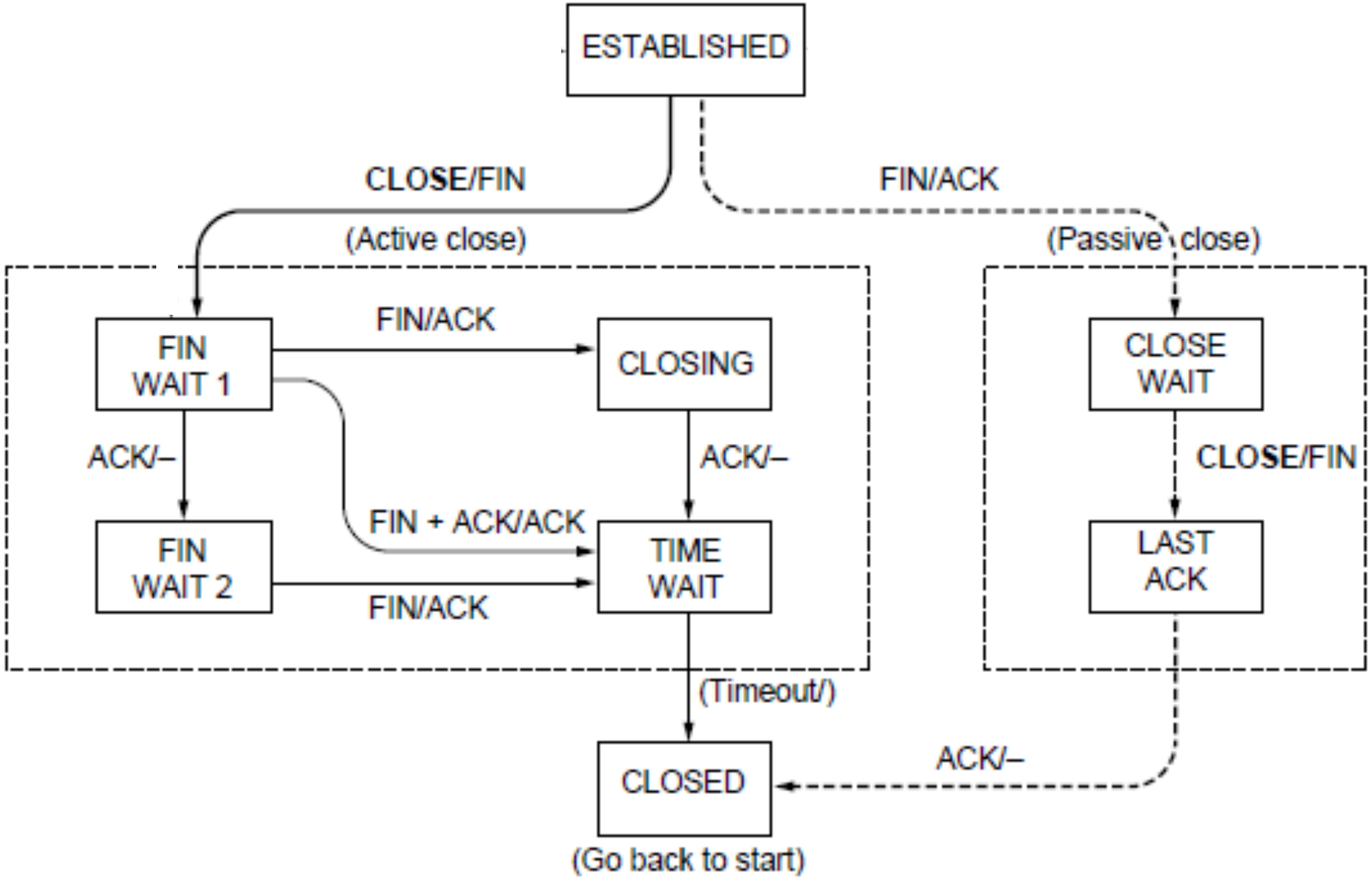
TCP Connection Release (2)

- Two steps:
  - Active sends FIN(x), passive ACKs
  - Passive sends FIN(y), active ACKs
  - FINs are retransmitted if lost
- Each FIN/ACK closes one direction of data transfer



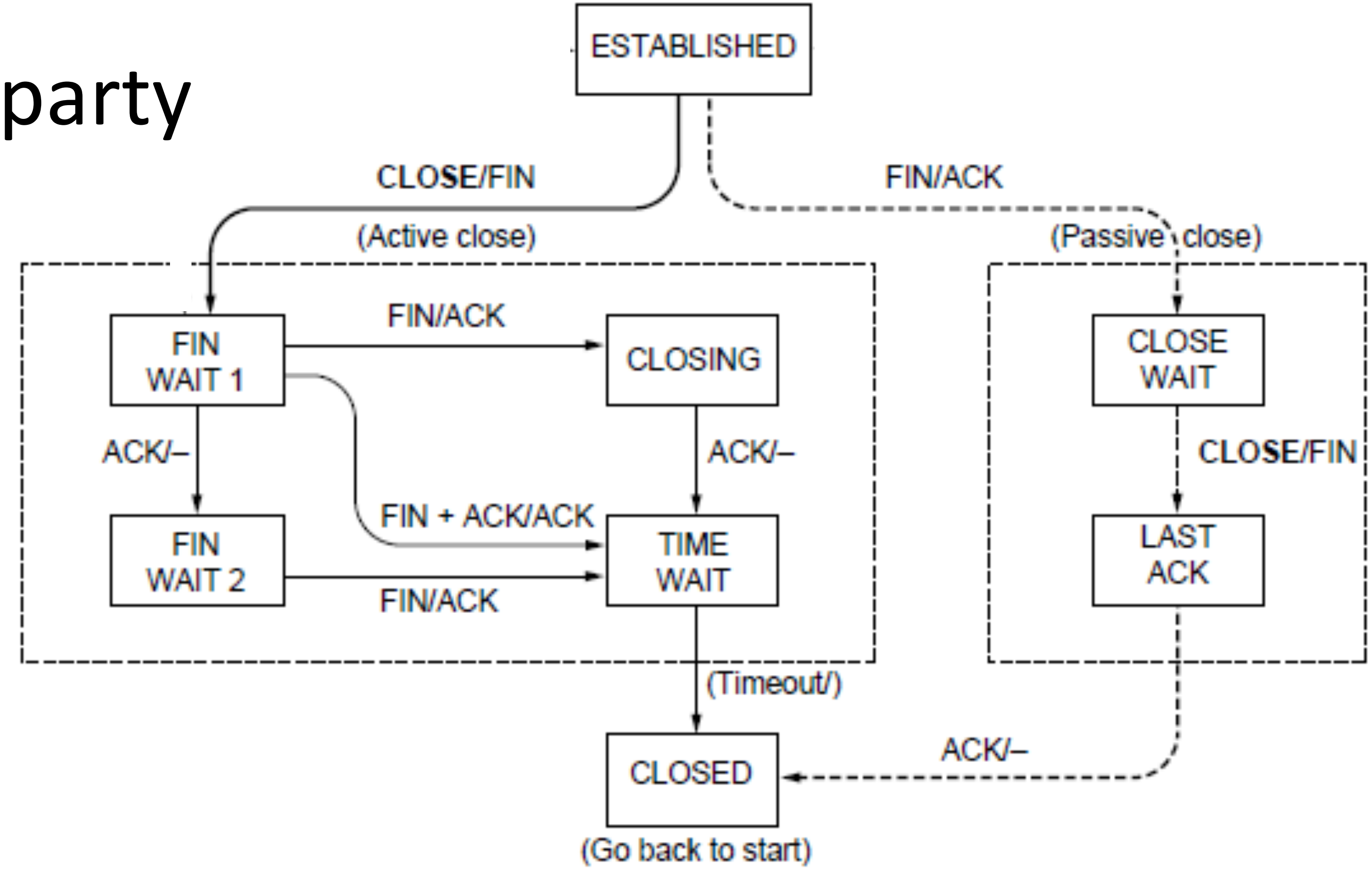
TCP Connection State Machine

Both parties run instances of this state machine



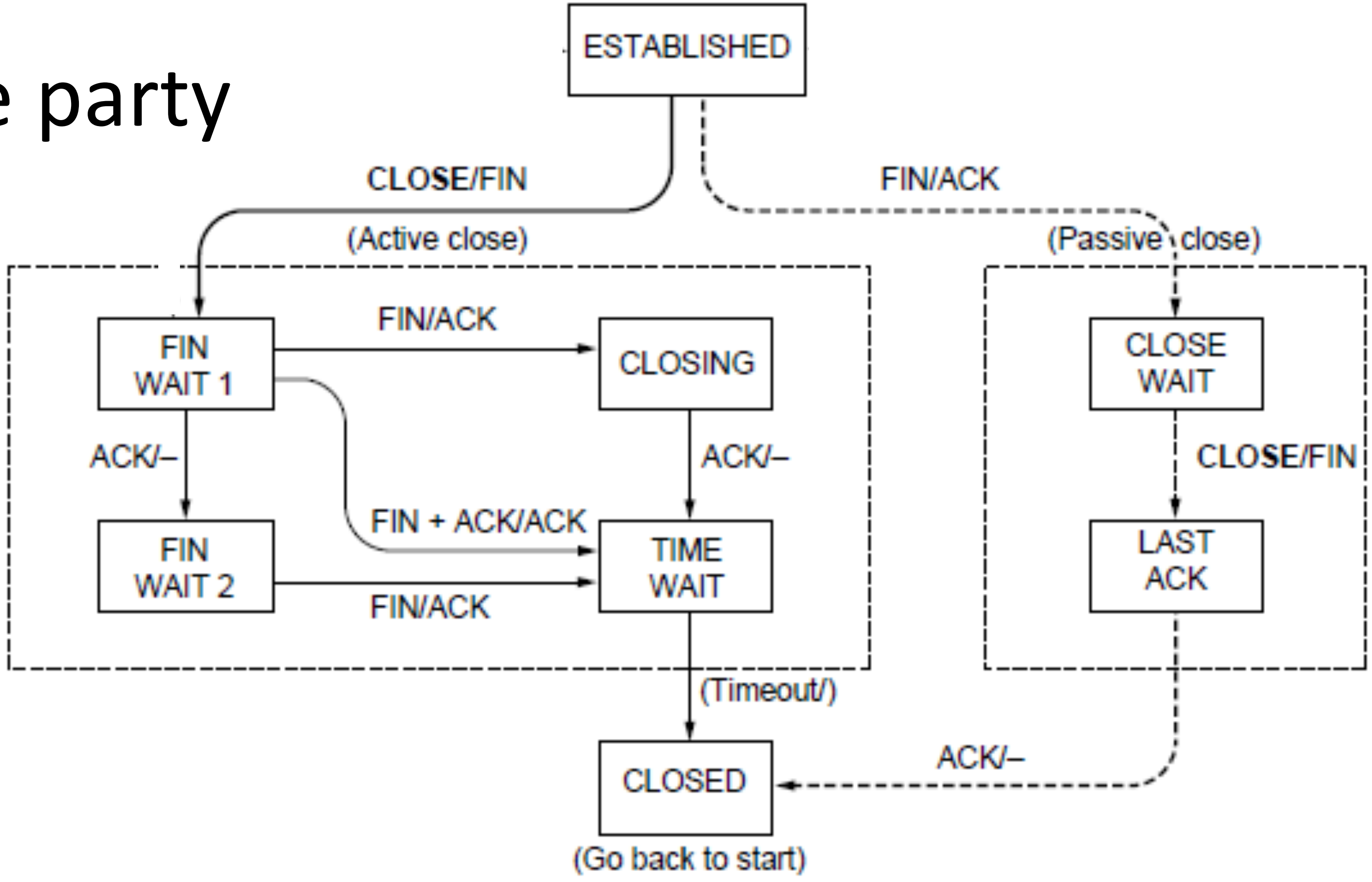
TCP Release

- Follow the active party



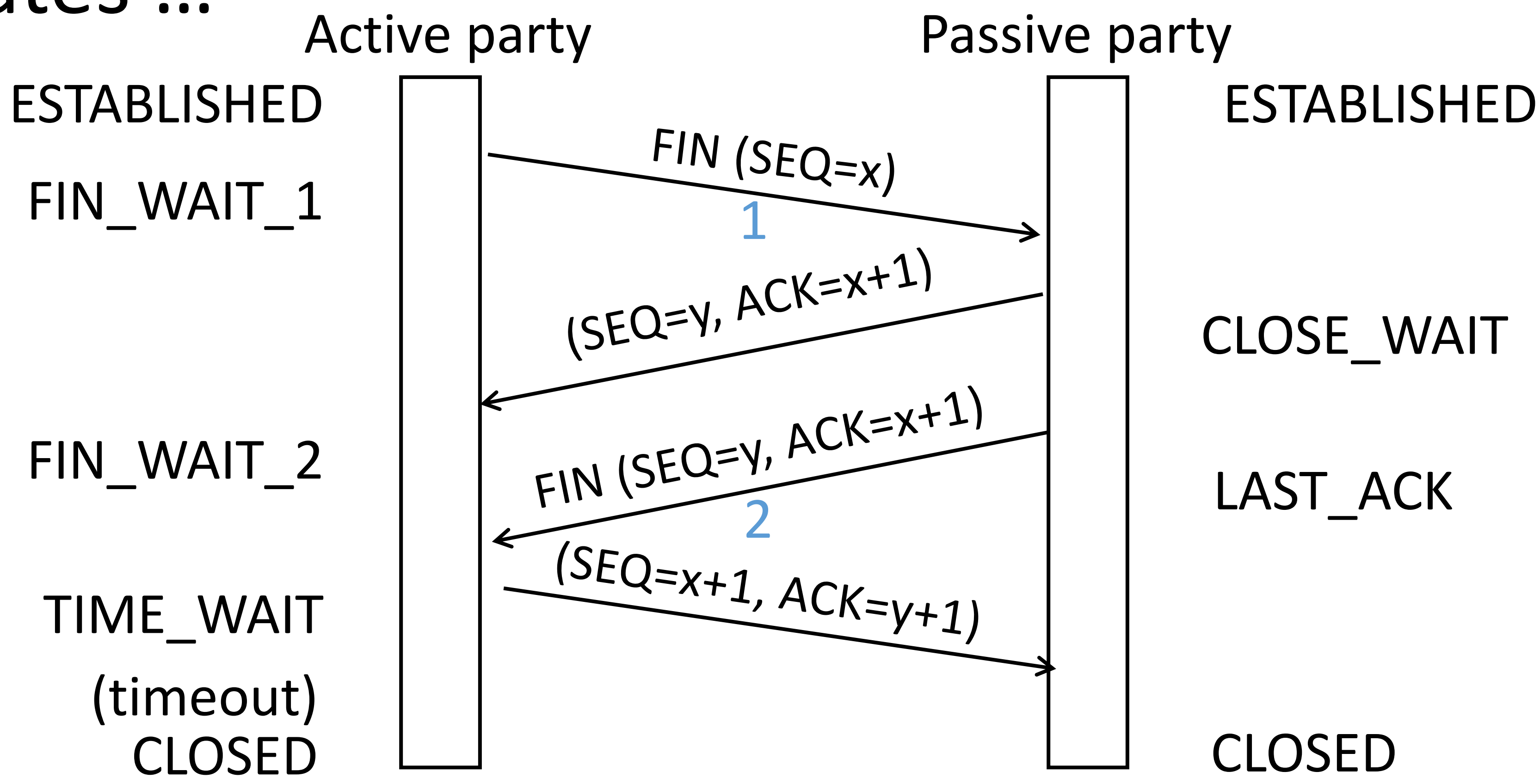
TCP Release (2)

- Follow the passive party



TCP Release (3)

- Again, with states …



TIME_WAIT State

- Wait a long time after sending all segments and before completing the close
  - Two times the maximum segment lifetime of 60 seconds
- Why?

TIME_WAIT State

- Wait a long time after sending all segments and before completing the close
  - Two times the maximum segment lifetime of 60 seconds
- Why?
  - ACK might have been lost, in which case FIN will be resent for an orderly close
  - Could otherwise interfere with a subsequent connection

Flow Control

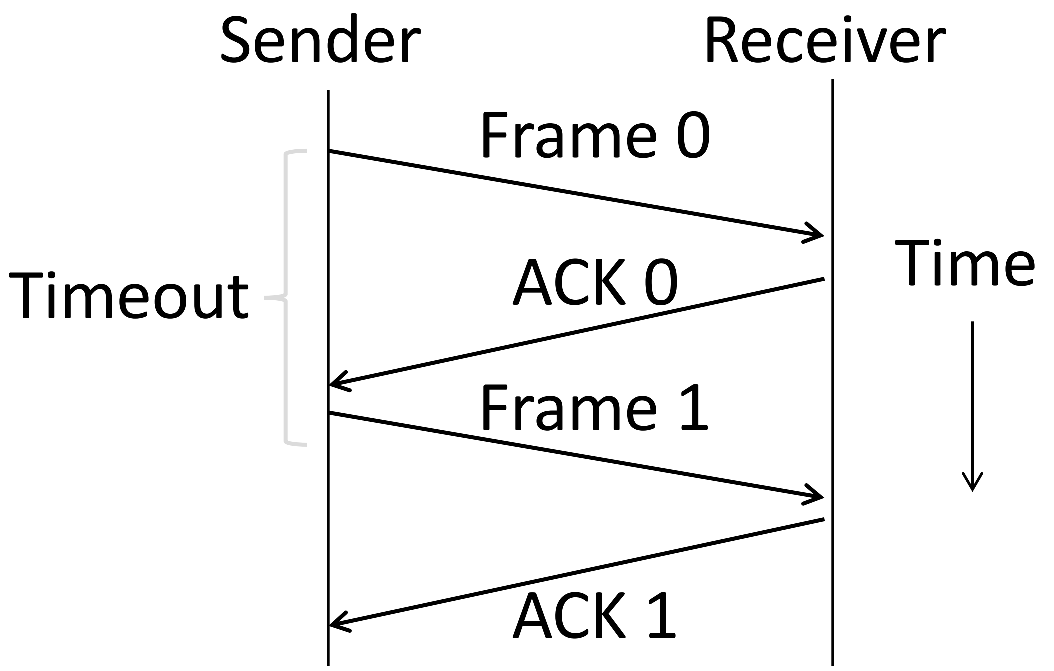
Flow control goal

Match transmission speed to reception capacity

- Otherwise data will be lost

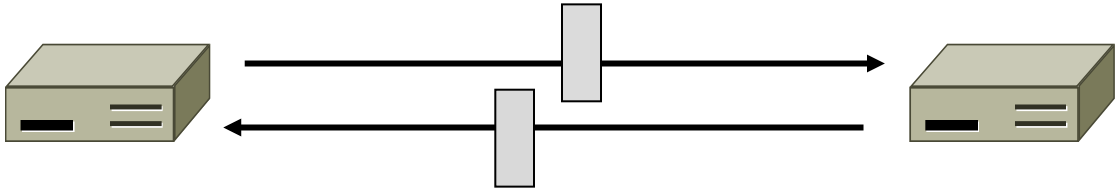
ARQ: Automatic repeat query

- ARQ with one message at a time is Stop-and-Wait



Limitation of Stop-and-Wait

- It allows only a single message to be outstanding from the sender:
  - Fine for LAN (only one frame fits in network anyhow)
  - Not efficient for network paths with longer delays

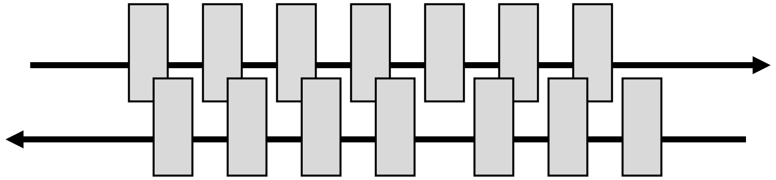


Limitation of Stop-and-Wait (2)

- Example: B=1 Mbps, D = 50 ms
  - RTT (Round Trip Time) = 2D = 100 ms
  - How many packets/sec?
    - 10
  - Usage efficiency if packets are 10kb?
    - $(10,000 \times 10) / (1 \times 10^6) = 10\%$
  - What is the efficiency if B=10 Mbps?
    - 1%

Sliding Window

- Generalization of stop-and-wait
  - Allows W packets to be outstanding
  - Can send W packets per RTT (=2D)



  - Pipelining improves performance
  - Need W=2BD to fill network path

Sliding Window (2)

What W will use the network capacity with 10kb packets?

- Ex: B=1 Mbps, D = 50 ms
  - 2BD = 2 x 10^6 x 50/1000 = 100 Kb
  - W = 100 kb/10 = 10 packets

- Ex: What if B=10 Mbps?
  - W = 100 packets

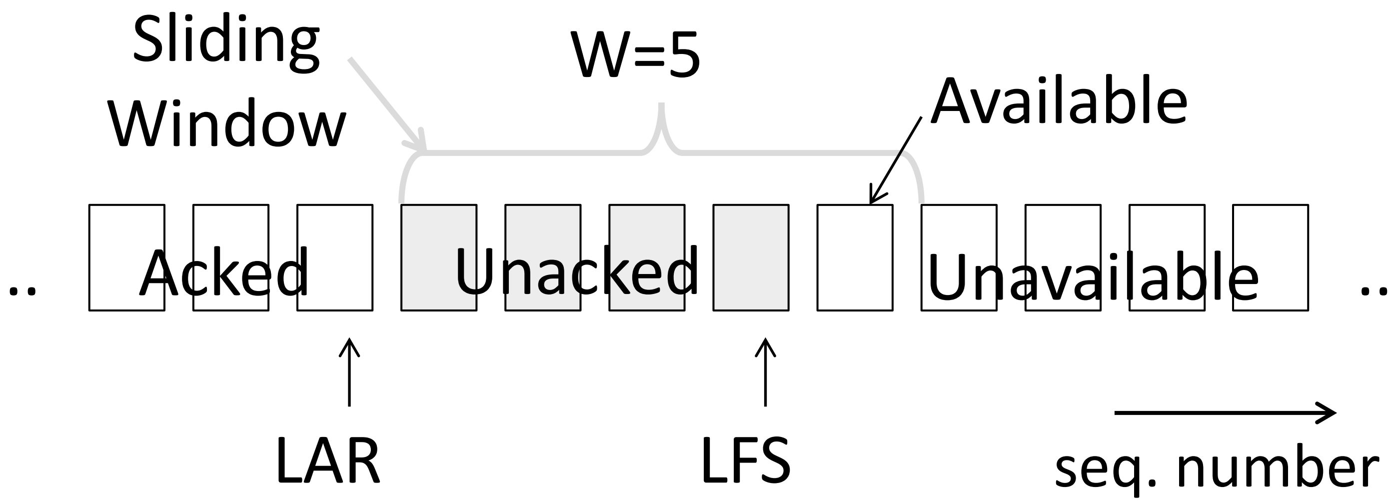
Sliding Window Protocol

- Many variations, depending on how buffers, acknowledgements, and retransmissions are handled

- <u>Go-Back-N</u>
  - Simplest version, can be inefficient
- <u>Selective Repeat</u>
  - More complex, better performance

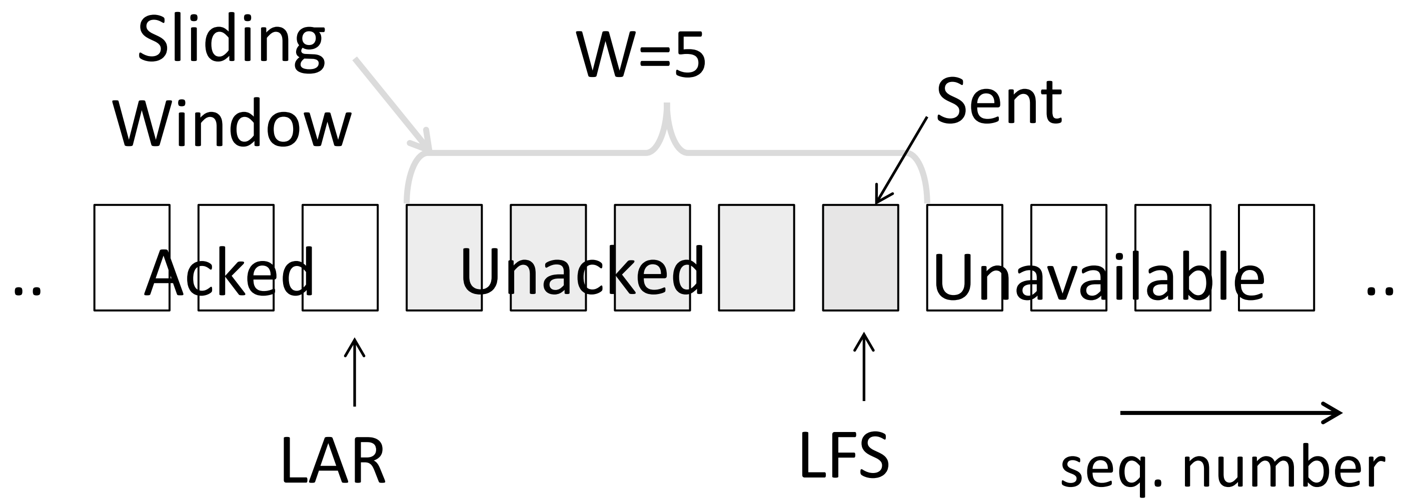
Sender Sliding Window

- Sender buffers up to W segments until they are acknowledged
  - LFS=LAST FRAME SENT, LAR=LAST ACK REC'D
  - Sends while LFS – LAR ≤ W



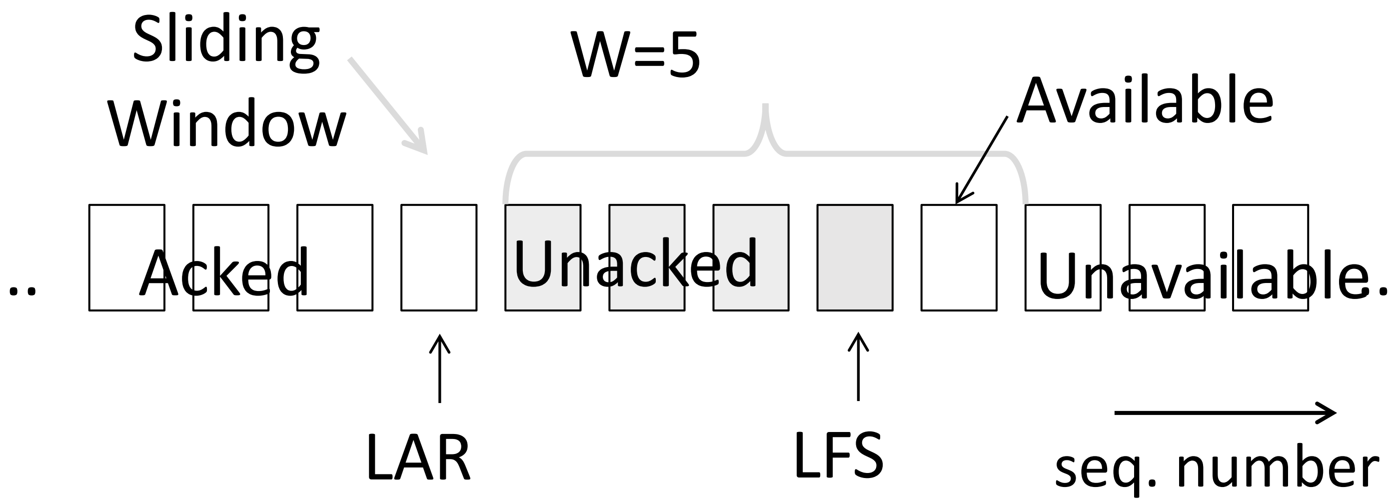
Sender Sliding Window (2)

- Transport accepts another segment of data from the Application ...
  - Transport sends it (LFS–LAR → 5)



Sender Sliding Window (3)

- Next higher ACK arrives from peer…
  - Window advances, buffer is freed
  - LFS–LAR → 4 (can send one more)



Receiver Sliding Window – Go-Back-N

- Receiver keeps only a single packet buffer for the next segment
  - State variable, LAS = LAST ACK SENT
- On receive:
  - If seq. number is LAS+1, accept and pass it to app, update LAS, send ACK
  - Otherwise discard (as out of order)

Receiver Sliding Window – Selective Repeat

- Receiver passes data to app in order, and buffers out-of-order segments to reduce retransmissions
- ACK conveys highest in-order segment, plus hints about out-of-order segments
  - Ex: I got everything up to 42 (LAS), and got 44, 45
- TCP uses a selective repeat design; we'll see the details later

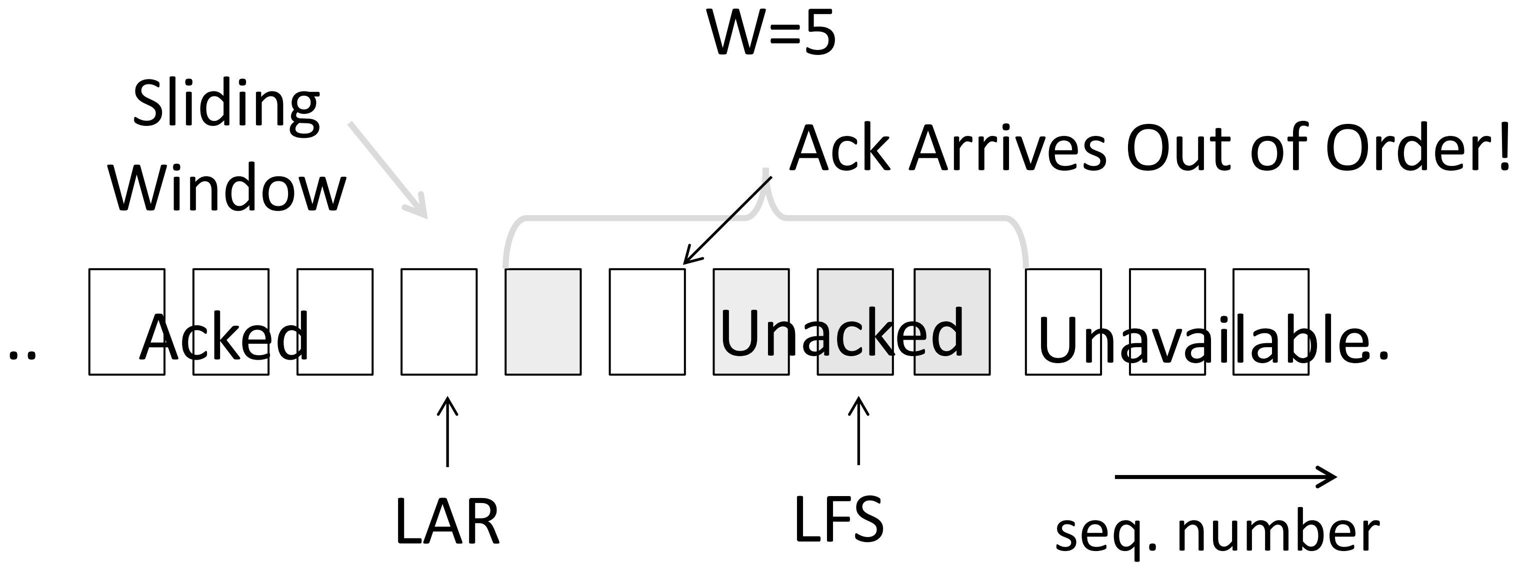
Receiver Sliding Window – Selective Repeat (2)

- Buffers W segments, keeps state variable LAS = LAST ACK SENT

- On receive:
  - Buffer segments [LAS+1, LAS+W]
  - Send app in-order segments from LAS+1, and update LAS
  - Send ACK for LAS regardless

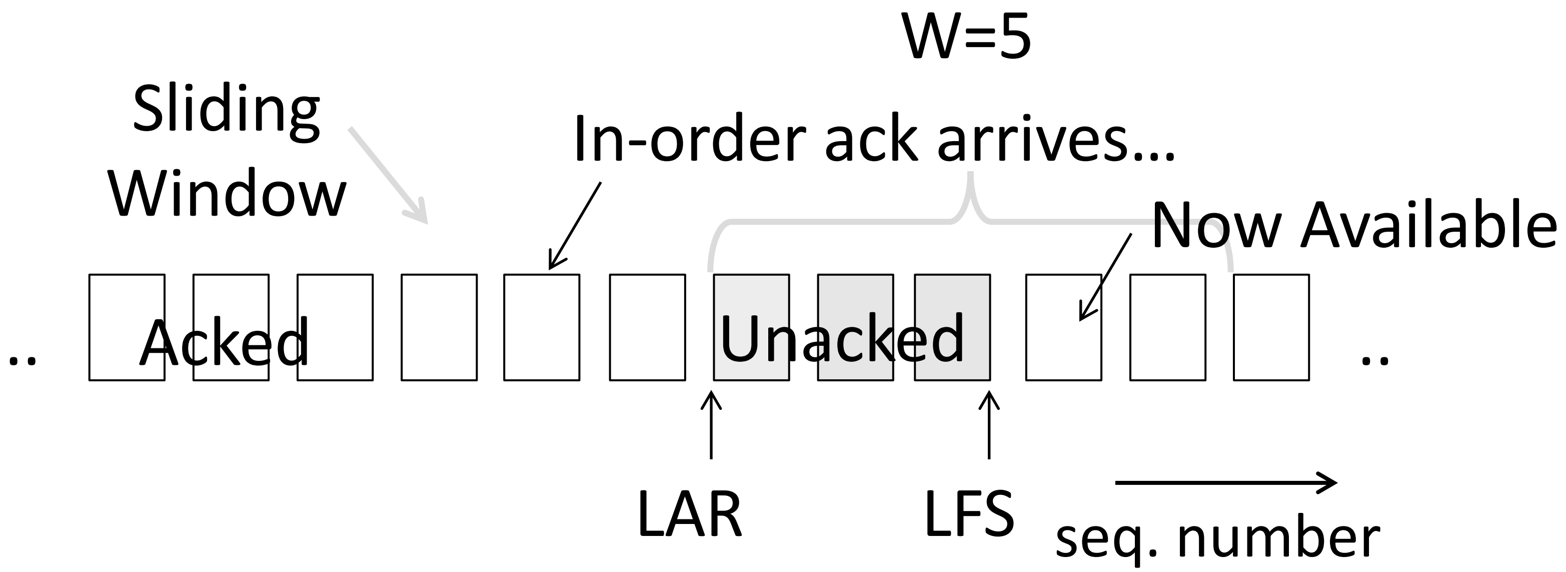
Sender Sliding Window – Selective Repeat

- Keep normal sliding window
- If out-of-order ACK arrives
  - Send LAR+1 again!



Sender Sliding Window – Selective Repeat (2)

- Keep normal sliding window
- If in-order ACK arrives
  - Move window and LAR, send more messages



Sliding Window – Retransmissions

- Go-Back-N uses a single timer to detect losses
  - On timeout, resends buffered packets starting at LAR+1
- Selective Repeat uses a timer per unacked segment to detect losses
  - On timeout for segment, resend it
  - Hope to resend fewer segments

Sequence Numbers

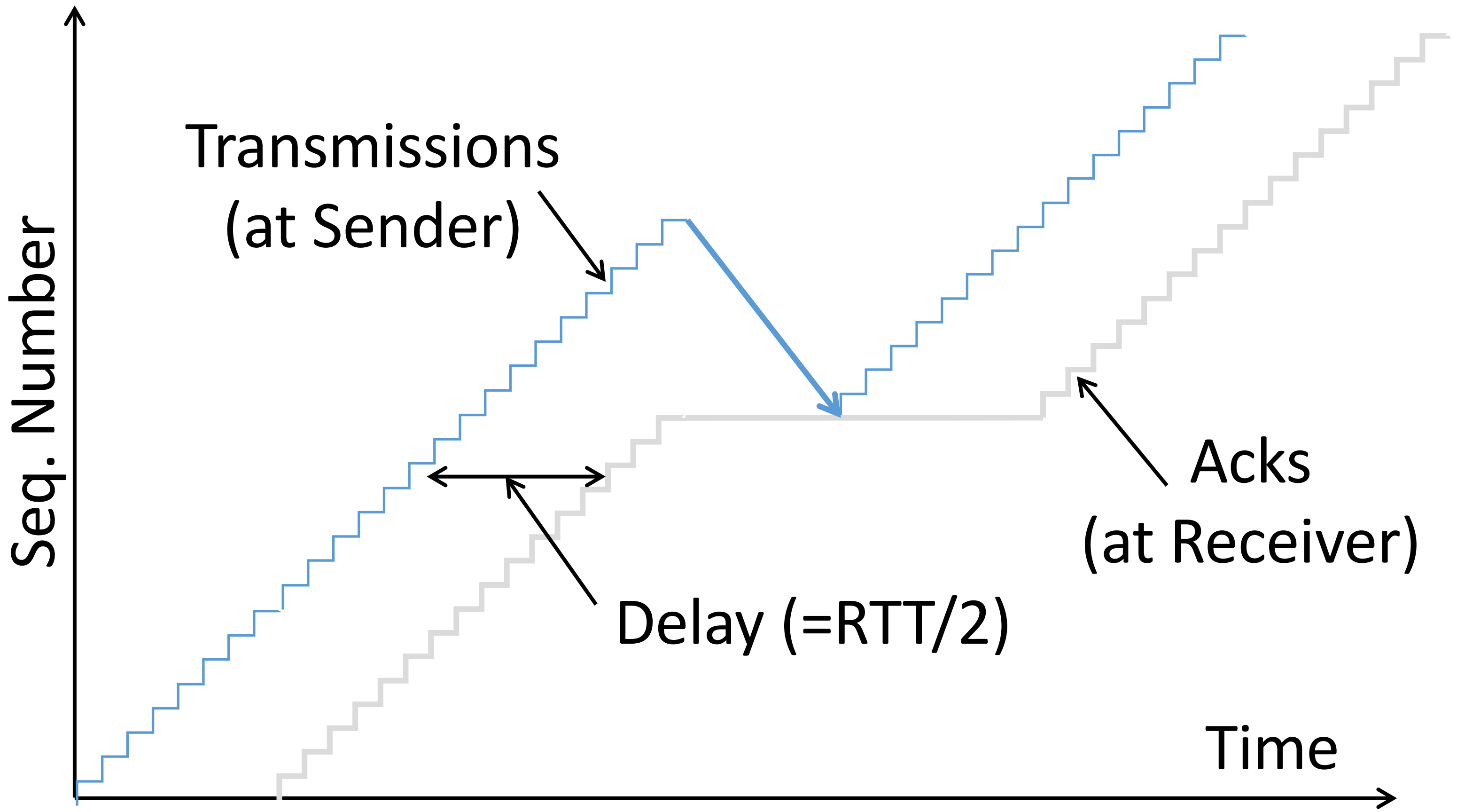
Need more than 0/1 for Stop-and-Wait … but how many?

- For Selective Repeat: 2W seq numbers
  - W for packets, plus W for earlier acks
- For Go-Back-N: W+1 sequence numbers

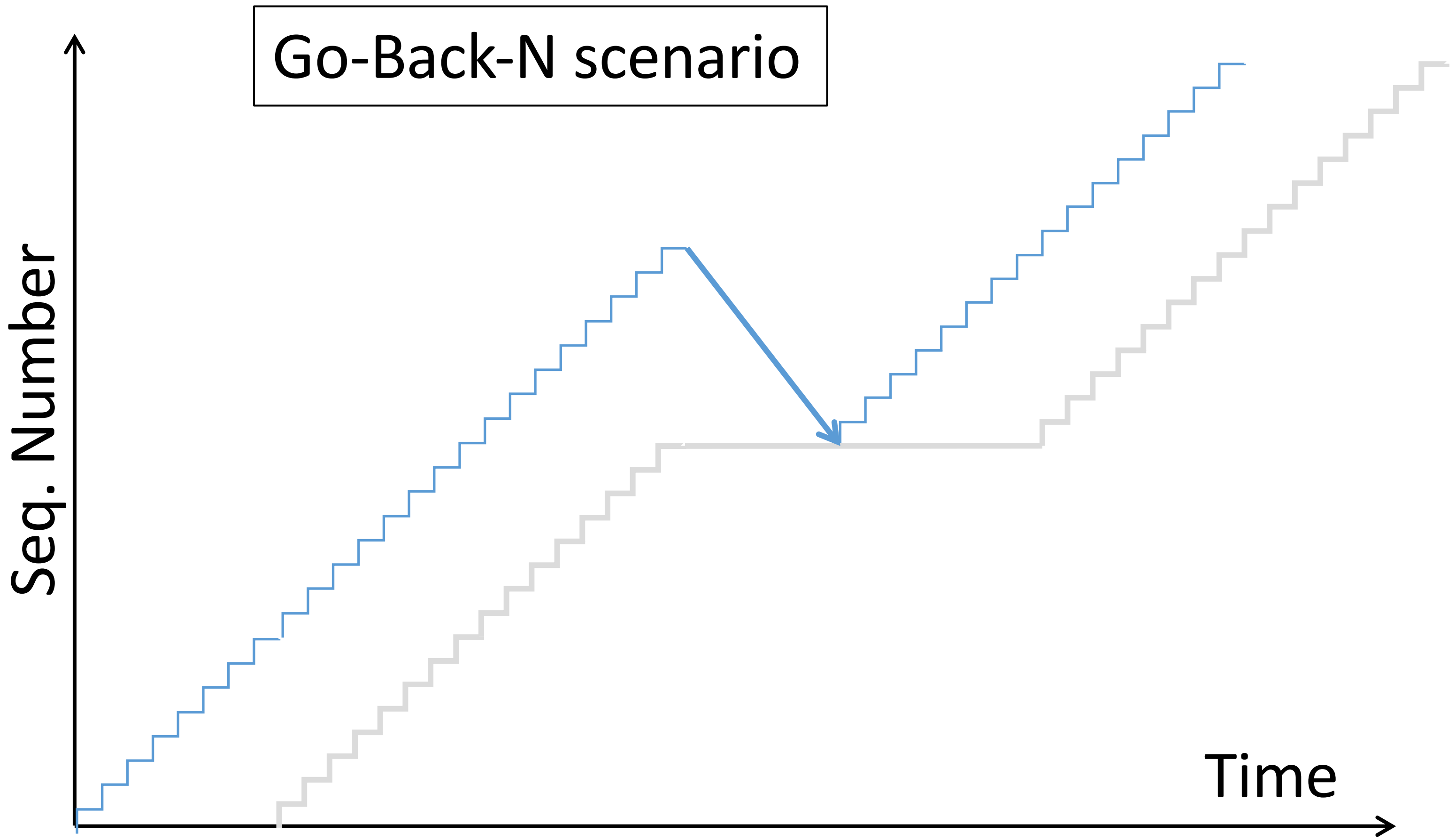
Typically implement seq. number with an N-bit counter that wraps around at 2^N—1

- E.g., N=8: …, 253, 254, 255, 0, 1, 2, 3, …

Sequence Time Plot



Sequence Time Plot (2)



Sequence Time Plot (3)