15-441 Computer Networks

Congestion Control

Professor Hui Zhang

hzhang@cs.cmu.edu

Review of Sliding Window Protocol For Error Control

- **Sender window size**
  - Maximum number of outstanding bytes (octets) before ack
- **Receiver window size**
  - Maximum number of out-of-order bytes
- **Sequence number to uniquely identify bytes**
- **Sender window**
  - Valid set of outstanding bytes
- **Receiver window**
  - Valid set of bytes to be received
- **Timeout to detect loss**
- **Retransmission to recover from loss**

Choices of Ack

- Cumulative ack
- Selective ack
- Negative ack
- Tradeoffs?

Timeout Value Selection

- **Long timeout?**
- **Short timeout?**
- **Solution?**

Sliding Window for Flow Control

- What is flow control?

- How to use sliding window protocol to implement flow control?

Sliding Window for Congestion Control

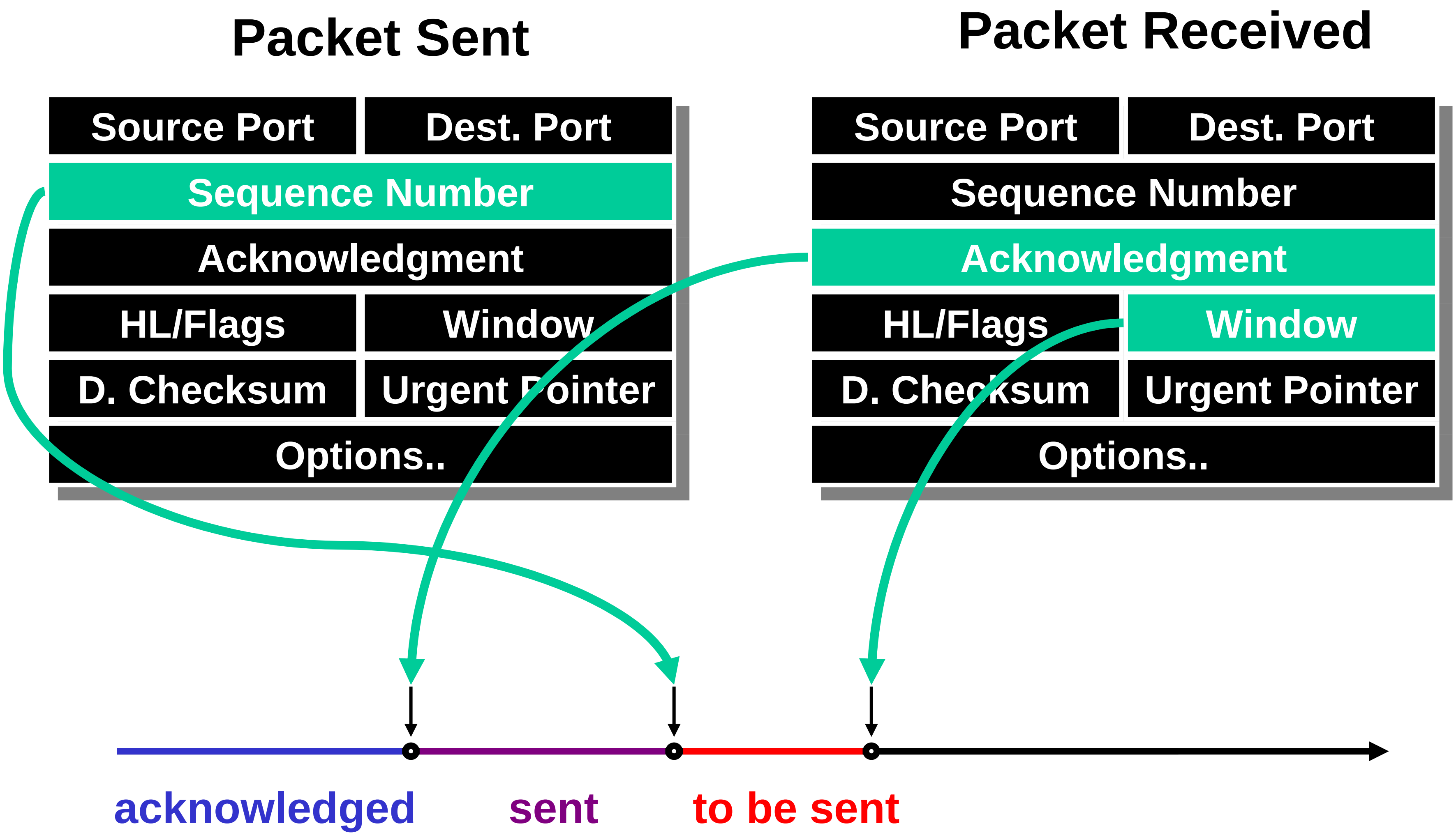
- What is congestion?

- How to implement congestion control using sliding window?

Review of TCP

- **Sliding window with cumulative acks**
  - Receiver can only return a single "ack" sequence number to the sender.
  - Acknowledges all bytes with a lower sequence number
  - Starting point for retransmission
  - Duplicate acks sent when out-of-order packet received
- **But: sender only retransmits a single packet.**
  - Reason???
- **Error control is based on byte sequences, not packets.**
  - Retransmitted packet can be different from the original lost packet – Why?

TCP Header Review

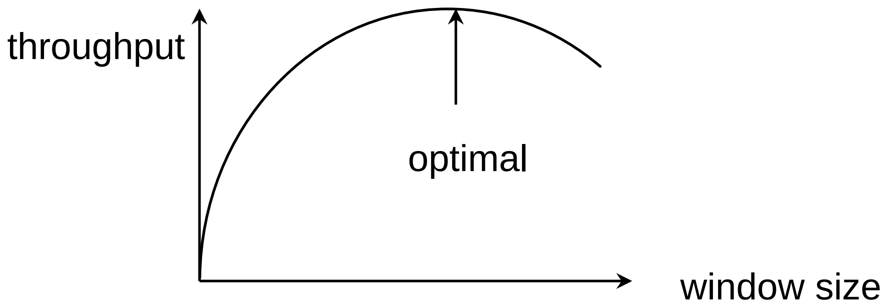


Hui Zhang

Importance of Window

- **Window mechanism implements both flow and congestion control**
- **Data can only be sent when the amount of outstanding data is less than the size of congestion window.**
  - The amount of outstanding data is increased on a “send” and decreased on “ack”
  - (last sent – last acked) < congestion window
- **Window limited by both congestion and buffering**
  - Sender’s maximum window = Min (advertised window, cwnd)

Window Size and Throughput



- **Larger the window size, higher the throughput**
  - Max Throughput = Window size /Round-trip Time
  - Need to worry about sequence number wrapping
- **Everyone uses large window size**
  - Too much traffic, router buffers overflow, packets dropped
  - End systems keep retransmitting the same packets
  - Nothing gets through
  - Congestion collapse!
- **How do you pick the window size?**

What is Steady State?

- **Packet conservation**
  - at equilibrium, inject packet into network only when one is removed
- **Self clocking**
  - Acknowledgement triggers the transmission of next packet

Reaching Steady State

- **How does TCP know what is a good initial rate to start with?**
  - Should work both for modem (14.4 kbps) and for OC-192 links (10 Gbps)
- **Quick initial phase to help get up to speed (slow start)**

TCP Congestion Control

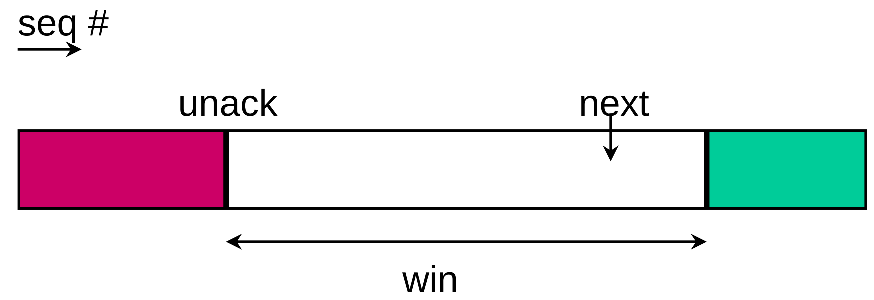
- **Initially, quickly increase the <u>congestion window</u> size until a packet is lost to get a rough estimate of the optimal congestion window size**
  - "Slow Start"
  - Exponential increase
- **Starting from the rough estimate, slowly increase the congestion window size to probe for additional available bandwidth**
  - "Congestion Avoidance"
  - Additive increase
- **Cut congestion window size aggressively if a timeout occurs**
  - Multiplicative decrease

TCP Congestion Control Pseudocode

```
Initially:
    cwnd = 1;
    ssthresh = infinite;
New ack received:
    if (cwnd < ssthresh)
        /* Slow Start*/
        cwnd = cwnd + 1;
    else
        /*Congestion Avoidance*/
        cwnd = cwnd + 1/cwnd;
Timeout:
    /* Multiplicative decrease */
    ssthresh = 0.5 * win;
    cwnd = 1;
```

```
while (next < unack + win)
    transmit next packet;

where   win = min(cwnd,
                  flow_win);
```

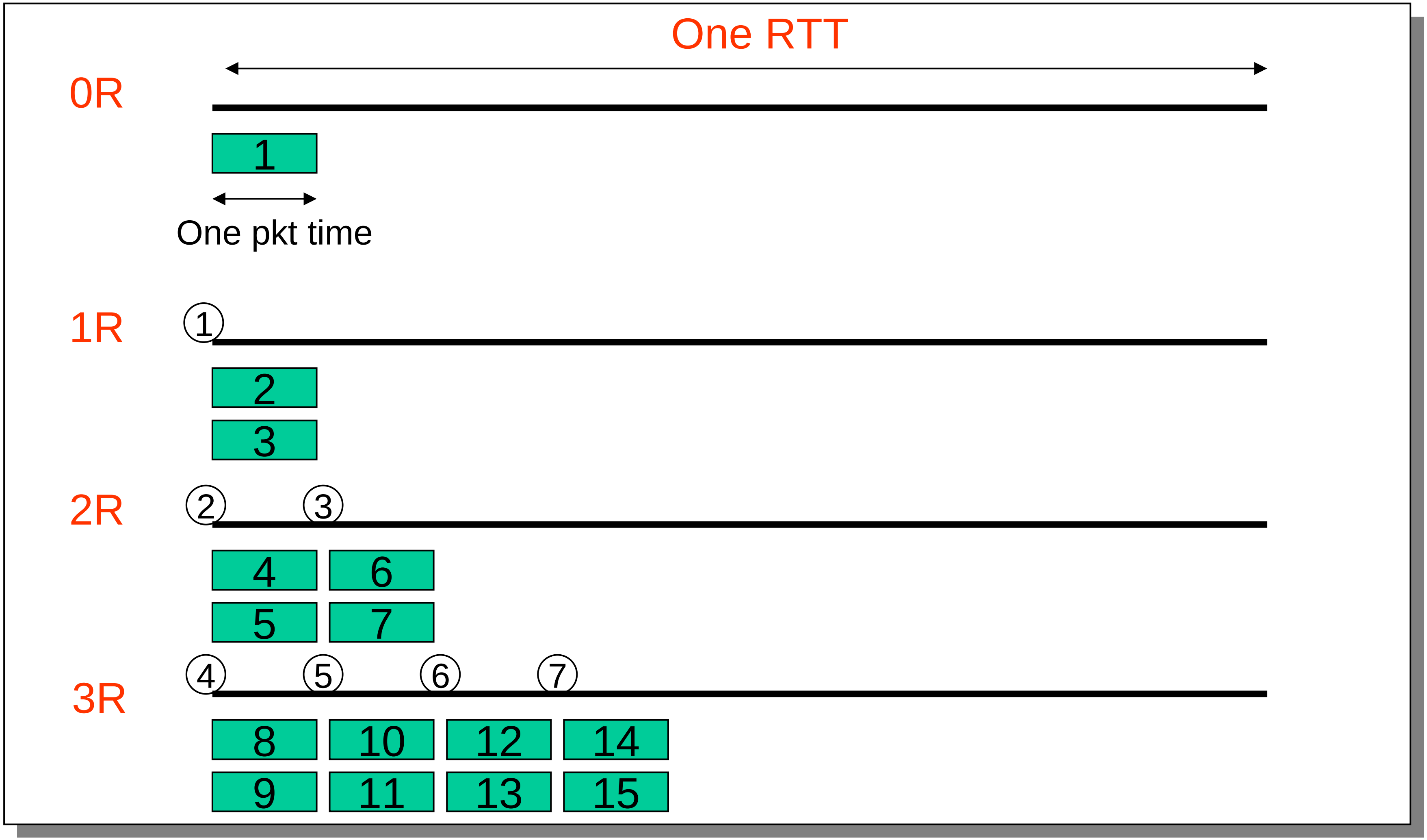


Slow Start

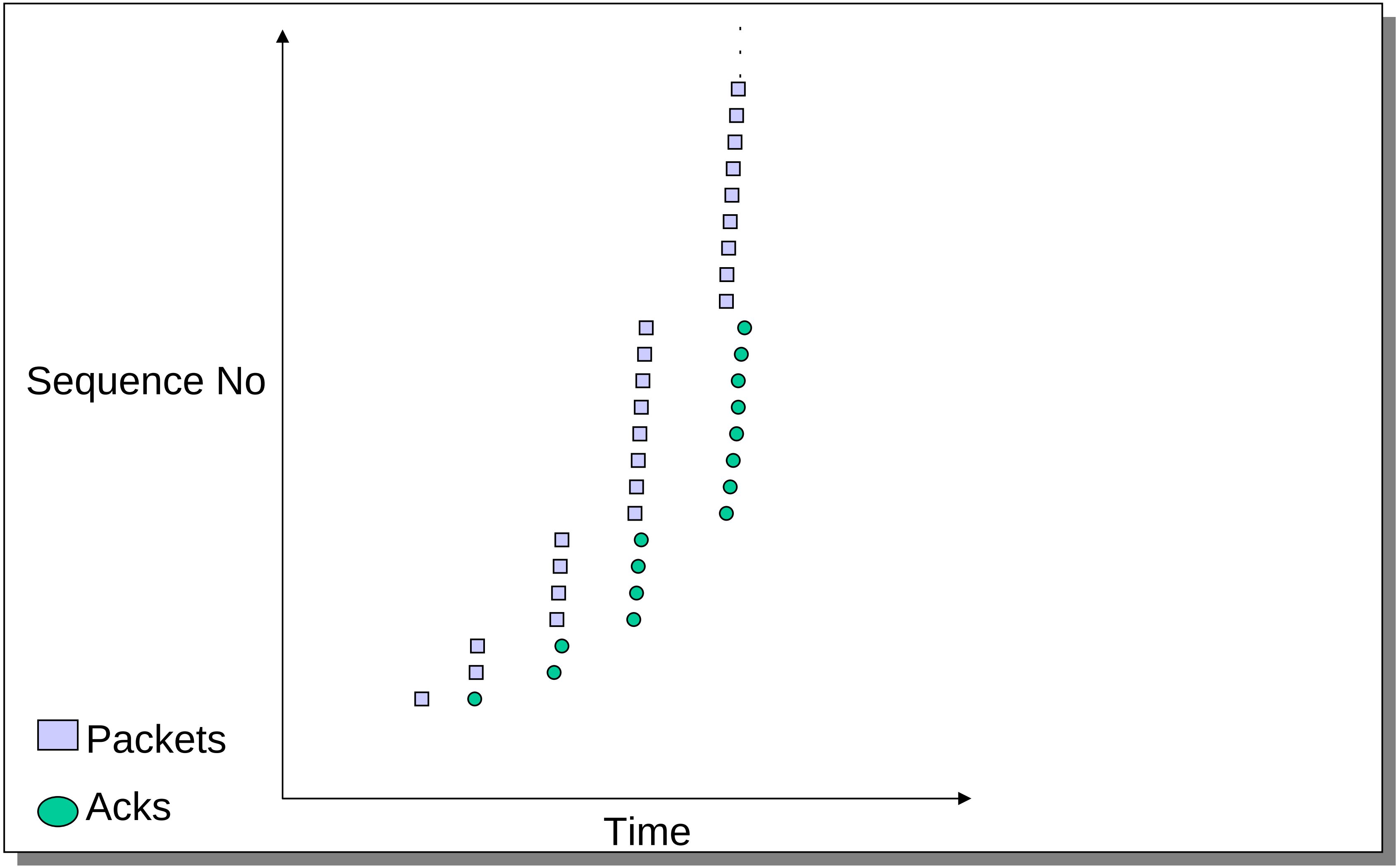
- **Whenever starting traffic on a new connection, or whenever increasing traffic after congestion was experienced:**
  - Set *cwnd* =1
  - Each time a segment is acknowledged increment *cwnd* by one (*cwnd*++).

- ***Does Slow Start increment slowly?* Not really. In fact, the increase of *cwnd* is exponential**

Slow Start Example



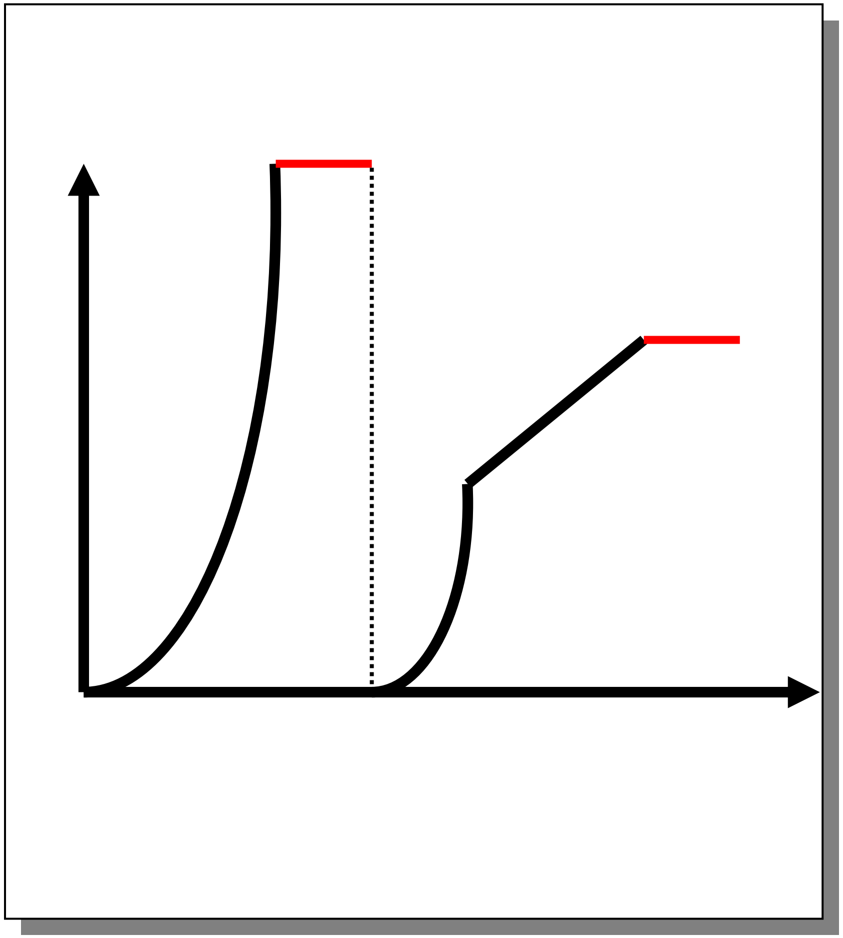
Slow Start Sequence Plot



Hui Zhang

Multiplicative Decrease

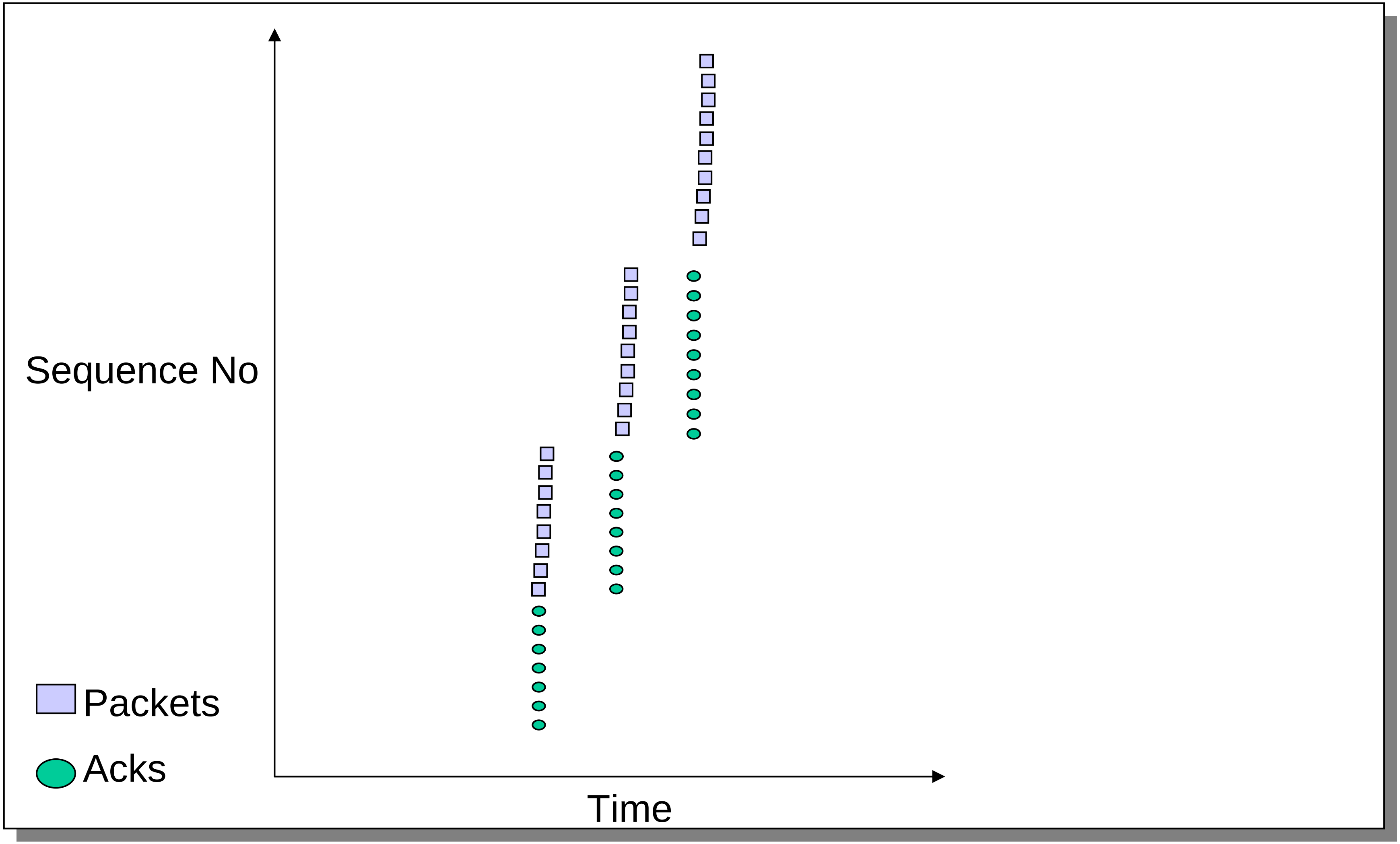
- What happens if we send too much?
- What does send do if there is a timeout?



Congestion Avoidance

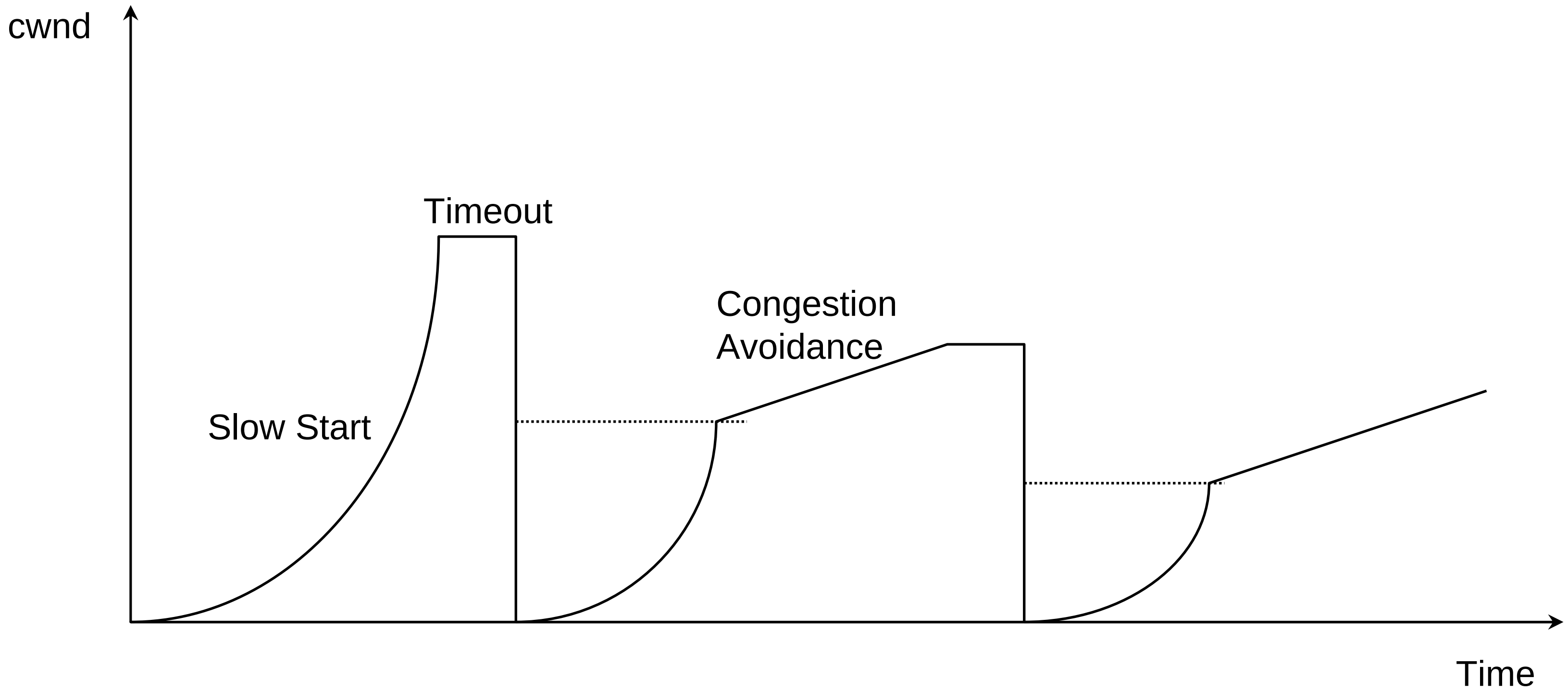
- **If loss occurs when cwnd = W**
  - Network can handle 0.5W ~ W segments
  - Set cwnd to 0.5W (multiplicative decrease)
- **Upon receiving ACK**
  - Increase cwnd by 1/cwnd
- **Implements AIMD**

Congestion Avoidance Sequence Plot



Hui Zhang

The big picture



Round-trip Time Estimation

- **Wait at least one RTT before retransmitting**
- **Importance of accurate RTT estimators:**
  - Low RTT → unneeded retransmissions
  - High RTT → poor throughput
- **RTT estimator must adapt to change in RTT**
  - But not too fast, or too slow!

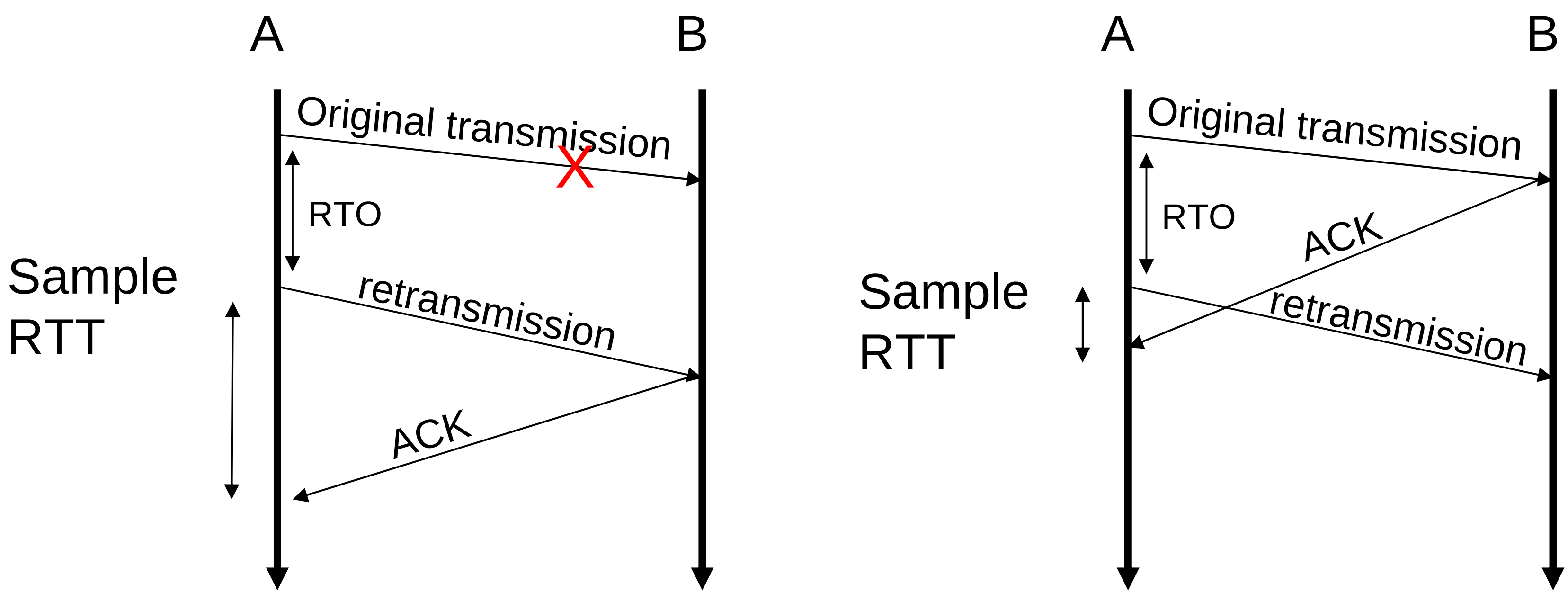
Initial Round-trip Estimator

- **Round trip times exponentially averaged:**
  - New RTT = α (old RTT) + (1 - α) (new sample)
  - Recommended value for α: 0.8 - 0.9
    - 0.875 for most TCP's
- **Retransmit timer set to β RTT, where β = 2**
  - Every time timer expires, RTO exponentially backed-off
  - Like Ethernet
- **Not good at preventing spurious timeouts**
  - Why?

Jacobson's Retransmission Timeout

- **Key observation:**
  - At high loads round trip variance is high
- **Solution:**
  - Base RTO on RTT and standard deviation
  - rttvar = χ * dev + (1- χ)rttvar
    - Dev = linear deviation
    - Inappropriately named – actually smoothed linear deviation

Retransmission Ambiguity



Karn's RTT Estimator

- **Accounts for retransmission ambiguity**
- **If a segment has been retransmitted:**
  - Don't count RTT sample on ACKs for this segment
  - Keep backed off time-out for next packet
  - Reuse RTT estimate only after one successful transmission

Timestamp Extension

- **Used to improve timeout mechanism by more accurate measurement of RTT**
- **When sending a packet, insert current timestamp into option**
  - 4 bytes for timestamp, 4 bytes for echo
- **Receiver echoes timestamp in ACK**
  - Actually will echo whatever is in timestamp
- **Removes retransmission ambiguity**
  - Can get RTT sample on any packet

Timer Granularity

- **Many TCP implementations set RTO in multiples of 200,500,1000ms**
- **Why?**
  - RTTs can vary quickly due to cross traffic
  - Make timers interrupts efficient
- **What is the implication?**

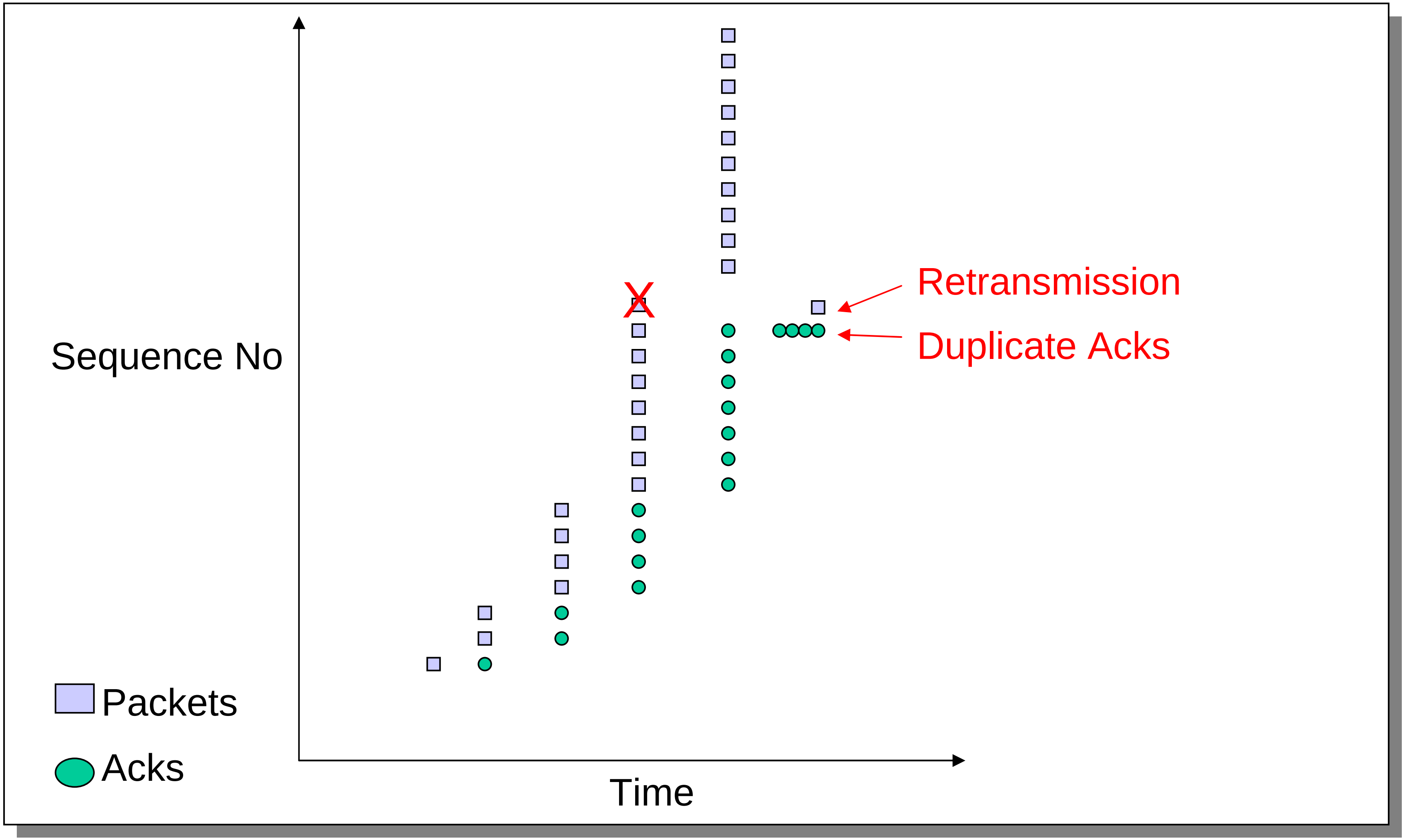
Avoiding Timeouts

- Current mechanism to detect packet loss is timeout
- Large timeout value slows down communication
- Alternative way of detecting loss?

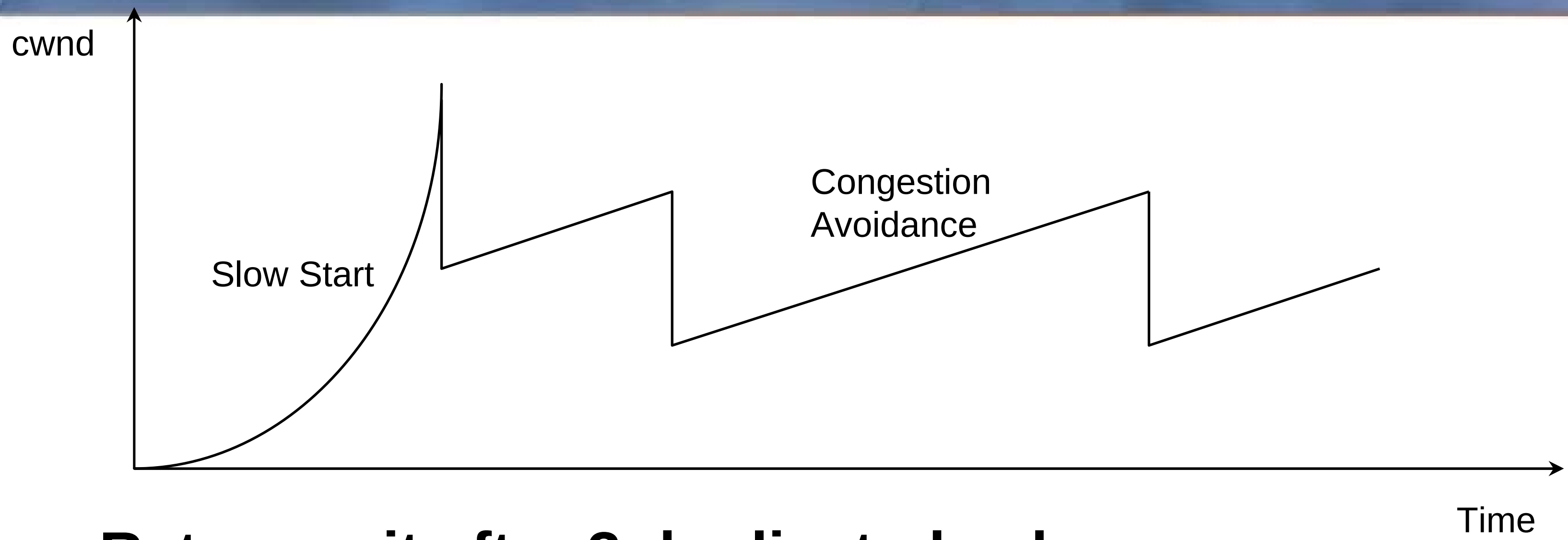
Fast Retransmit

- **What are duplicate acks (dupacks)?**
  - Repeated acks for the same sequence
- **When can duplicate acks occur?**
  - Loss
  - Packet re-ordering
  - Window update – advertisement of new flow control window
- **Assume re-ordering is infrequent and not of large magnitude**
  - Use receipt of 3 or more duplicate acks as indication of loss
  - Don’t wait for timeout to retransmit packet

Fast Retransmit

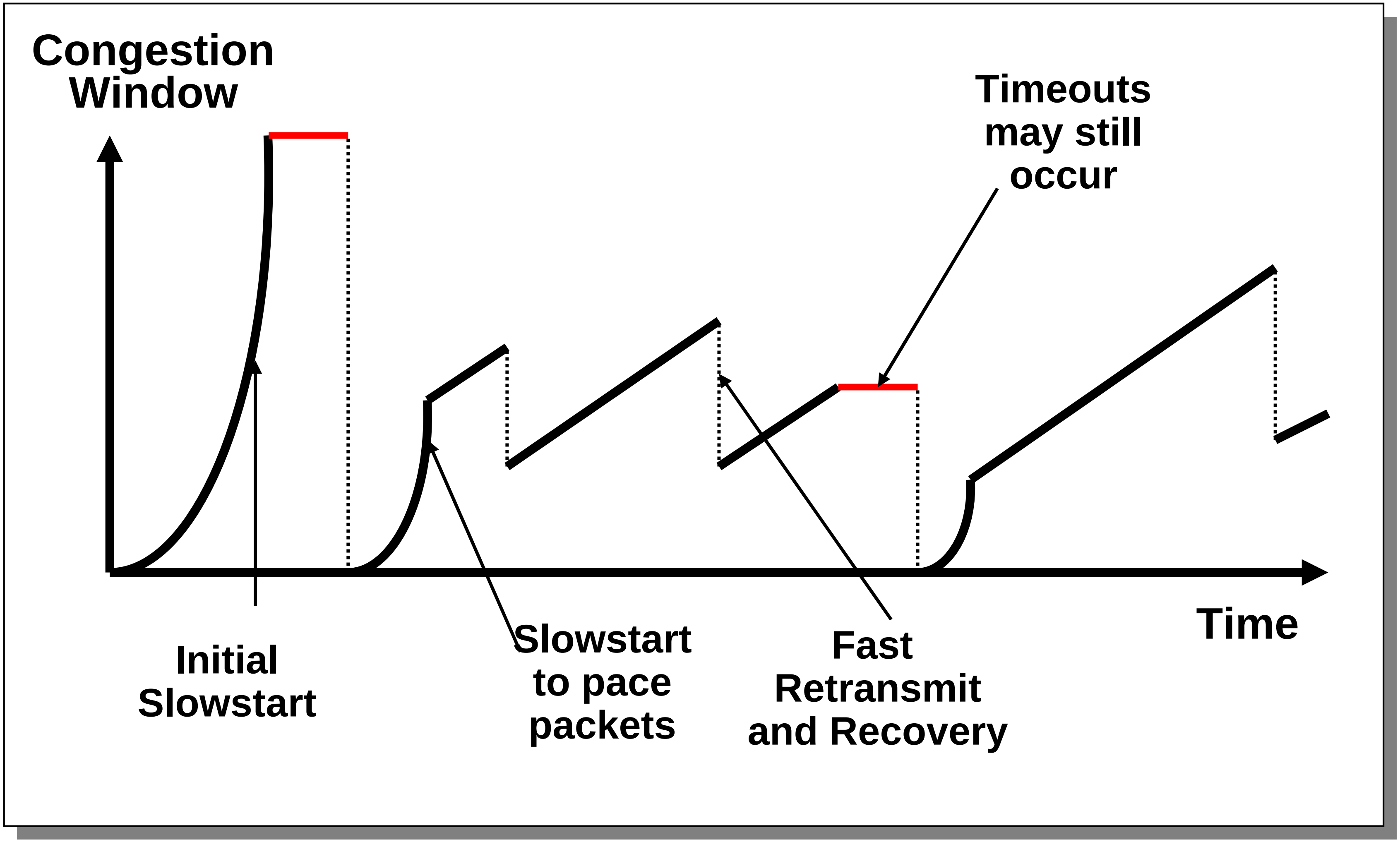


Fast Retransmit and Fast Recovery



- **Retransmit after 3 duplicated acks**
  - prevent expensive timeouts
- **No need to slow start again**
- **At steady state, cwnd oscillates around the optimal window size.**

TCP Saw Tooth Behavior



How to Improve?

- **Timeout value**
  - Explicitly signaling RTT (how?)
- **Loss as congestion signal**
  - Good signal?
  - Better signal?
  - Rely on accurate calculation of timeout
- **16 bits window size big enough?**
- **32 bits sequence number?**
- **Synchronization effect**
- **Fairness?**
- **Non-cooperative sources**