

## **Transport Layer Part 3: TCP Congestion Control**

These slides are created by Dr. Yih Huang of George Mason University. Students registered in Dr. Huang's courses at GMU can make a single machine-readable copy and print a single copy of each slide for their own reference, so long as each slide contains the copyright statement, and GMU facilities are not used to produce paper copies. Permission for any other use, either in machine-readable or printed form, must be obtained from the author in writing.

## **Network Congestion**

- ❑ When the network is congested (that is, there are more packets than routers can handle), packets are buffered at routers.
- ❑ When congestion worsens, routers run out of buffer space and have no choice but to drop some packets.
- ❑ In response, users of the network (precisely, TCP modules of the users) must reduce traffic volumes.

## Basic Ideas

- ❑ The assumption is that packet loss caused by transmission errors is rare and thus
  - **packet losses signal congestion**
- ❑ Signs of packet losses as seen by a sender
  - Timeouts
  - Any others ?

CS 656

3

## Responses

- ❑ With sliding window protocols,
  - A large window allows a sender to be aggressive in transmission
  - a small one forces it to stop-and-wait frequently and thus curbs traffic volume (bytes per sec).
- ❑ Window size is our major concern.

CS 656

4

## Slow Start: Introduction

- ❑ Old TCP starts a connection with the sender injecting its entire window to the network.
- ❑ If there are slow/congested links between the sender and receiver, this may create/exacerbate congestion.

- ❑ The slow start mechanism is later introduced so that the sender opens up its window incrementally, rather than starting with a full size window.
- ❑ Effectively, the sender of a new connection probes the capacity of the network.

## Slow Start: Algorithm

- ❑ The sender maintains an integer variable, **cwnd** (congestion window).
- ❑ When a connection is established,  $cwnd = \text{segsize}$  (segment size).
- ❑ The sender transmits up to the minimum of  $cwnd$  and **rwnd**, the window size advertised by receiver.
- ❑ Each time an ACK is received,  $cwnd += \text{segsize}$ .

CS 656

7

## Window Size Advertisement

- ❑ In TCP, the receiver also maintains a window to buffer inbound data
- ❑ When the receiver returns a (piggybacked) Ack to the sender, it uses the “Window Size” field in the TCP header to indicate the available space in bytes in its window.
- ❑ The sender adjusts its window size in order not to overflow the receiver window.

CS 656

8

## Discussion

- ❑ `cwnd` signifies the sender's assessment of network capacity.
- ❑ `rwnd` indicates the capacity of the receiver.
- ❑ The sender transmits at most the minimum of the two.

$$\text{WindowSize} = \min\{\text{cwnd}, \text{rwnd}\}$$

## How Slow Is Slow Start ?

- ❑ The sender starts by transmitting one segment and wait for its ACK.
- ❑ When the ACK is received, `cwnd` is increased to two segments, and two segments are then sent.
- ❑ When each of those two segments is acknowledged, `cwnd` is increased to four segments.
- ❑ Thus, `cwnd` can be doubled per round-trip time and increase *exponentially*.
  - not exactly slow !
- ❑ Either `cwnd` grows larger than `rwnd`, or the sender has overloaded the network, causing packet losses and timeouts.

## Handling Congestion

- ❑ In the face of congestion, the sender reduces traffic immediately by setting `cwnd` to one segment.
- ❑ It should then probe network capacity again.
  - Meaning to increase `cwnd` incrementally
- ❑ However, since we already experienced congestion, probing by slow start may be too aggressive.

- ❑ The solution is to allow `cwnd` to grow exponentially halfway and afterwards only *linearly* (grows roughly one segment per round-trip time).
- ❑ The “halfway” point is recorded in variable **`ssthresh`**, standing for slow start threshold.

## Algorithm

- Upon connection establishment:
  - `cwnd = segsize`
  - `ssthresh = 65535`
- When a timeout occurs,
  - `ssthresh = max{cwnd/2, segsize*2}`
  - `cwnd = segsize`
- When an Ack is received,
  - If `cwnd ≤ ssthresh`, `cwnd += segsize`
  - Otherwise,
    - `cwnd += segsize*segsize/cwnd`

CS 656

13

## Discussion

- Please note how the algorithm approximates linear growth by  
`cwnd += segsize*segsize/cwnd`
  
- Why the large initial value of `ssthresh` ?

CS 656

14

## Recall TCP ACKs

- Assume that all bytes up to 999 have been received and acked. Give the acks in response to segments
  - Seg(1000, 200)
  - Seg(1300, 100)
  - Seg(1400, 300)
  - Seg(1700, 100)
  - Seg(1200, 100)

CS 656

15

## Fast Retransmission

- When the receiver receive segments  $x-1$ ,  $x+1$ ,  $x+2$ ,  $x+3$ , ..., it acknowledges  $x$ ,  $x$ ,  $x$ ,  $x$ , ..., indicating its expecting a segment with seq #  $x$ .
- Duplicate ACKs of  $x$  are strong indications of the loss of  $x$ .
  - Why can't we be 100% certain ?

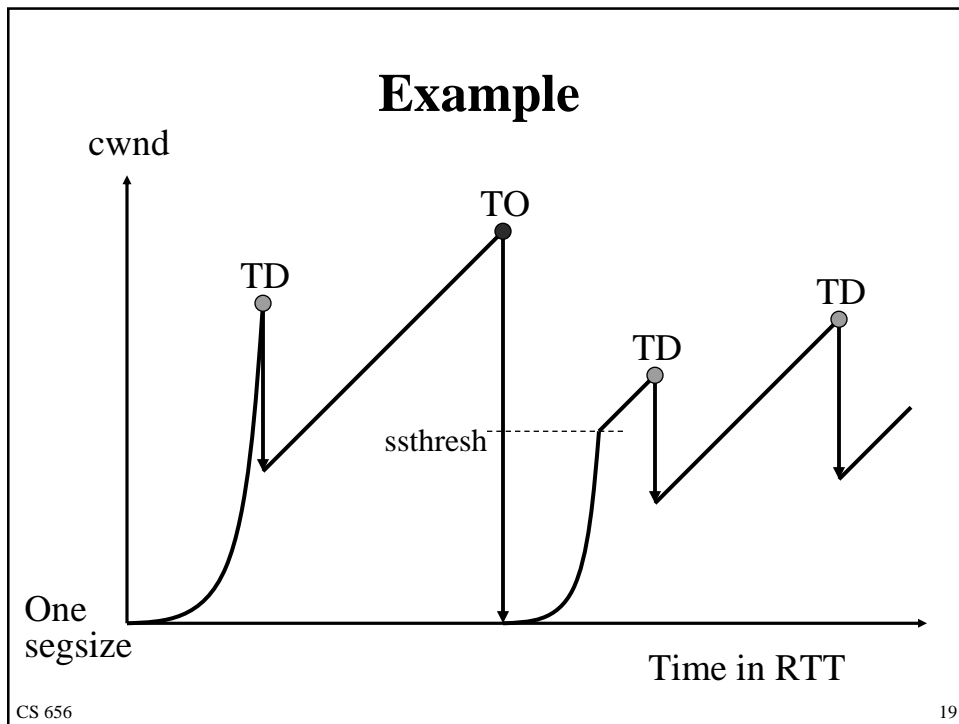
CS 656

16

- ❑ However, the network does not seem congested, for subsequent segments do get through.
- ❑ When the sender sees 3 duplicate Ack( $x$ ), it retransmits  $x$  immediately (not to await timeout); hence the name “fast” retransmission.
- ❑ Subsequently, we want to proceed cautiously but not as pessimistically as real congestion.

## Fast Recovery

- ❑ After fast retransmission of segment  $x$ , half the value of  $cwnd$ 
  - Notice that  $cwnd$  is now automatically greater than  $ssthresh$
- ❑ The result is to skip the slow-start phase; hence the name “fast” recovery.



- ### Some Remarks
- ❑ Fast retransmissions and recovery are optimizations.
  - ❑ They are not used in all implementations.
  - ❑ TCP has many flexibilities/loopholes that lead to different “flavors” of implementations.
    - All implementations are mutually compatible even though their behaviors differ from time to time.
- CS 656 20