CPE348: Introduction to Computer Networks

Lecture #17: Chapter 6

Jianqing Liu
Assistant Professor of Electrical and Computer Engineering, University of Alabama in Huntsville

jianqing.liu@uah.edu
http://jianqingliu.net

TCP Congestion Control Overview

- Early on, we talked about:
  - TCP sliding window protocol;
  - TCP adaptive transmission algorithms.
- In this lecture, we will talk about:
  - TCP congestion control protocols
    - Additive Increase/Multiplicative Decrease
    - Slow Start
    - Fast Retransmit and Fast Recovery

TCP Congestion Control – history

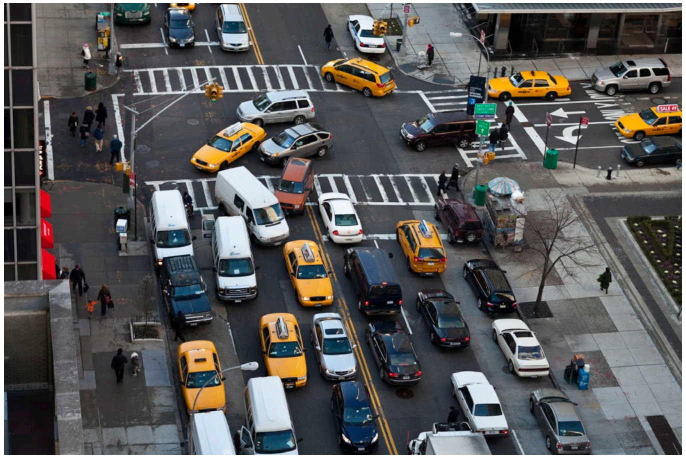
- It was introduced into the Internet in the late 1980s by Van Jacobson, ~ 8 yrs after the TCP/IP had become operational.
- Immediately preceding this time, the Internet was suffering from congestion collapse —
  - hosts would send their packets into the Internet as fast as the advertised window would allow
  - congestion would occur at some router, causing pkts drop
  - hosts would time out and retransmit their packets, resulting in even more congestion

TCP Congestion Control – history

- Basic idea
  - Source determines how much capacity is available in the network,
    - it uses the arrival of an ACK as a signal that one of its packets has left the network,
    - source can then insert a new packet into the network.
  - By using ACKs to pace the transmission of packets, TCP is said to be *self-clocking.*

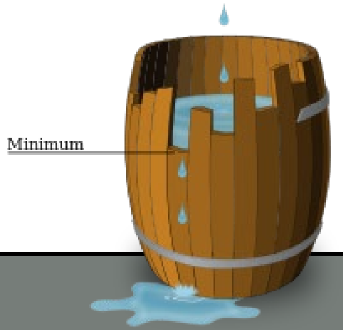
TCP Congestion Control

- Three popular TCP congestion control protocols
  - Additive Increase/Multiplicative Decrease
  - Slow Start
  - Fast Retransmit and Fast Recovery



Additive Increase Multiplicative Decrease

- Source maintains a new variable CongestionWindow
- TCP's effective window is revised as follows:
  - **MaxWindow = MIN(CongestionWindow, AdvertisedWindow)**
  - **EffectiveWindow = MaxWindow − (LastByteSent − LastByteAcked).**
- That is, MaxWindow replaces AdvertisedWindow;
- Source is allowed to send no faster than the slowest component—the network or the destination host—can accommodate.



Additive Increase Multiplicative Decrease

- Question is how source comes to learn an appropriate value for CongestionWindow.
  - First, source sets it as a default value based its perception to the network.
  - Then,
    - Decrease the CongestionWindow when the level of congestion goes up and
    - Increase the CongestionWindow when the level of congestion goes down.
    - Taken together, the mechanism is commonly called *additive increase/multiplicative decrease (AIMD)*

Additive Increase Multiplicative Decrease

- Th next question is how source comes to learn congestion occurs.
  - Observation: when packets are not delivered, timeout occurs
  - Then, source interprets timeouts as a sign of congestion and reduces the rate at which it is transmitting.

    - When a timeout occurs, the source sets CongestionWindow to half of its previous value – multiplicative decrease

    - When no timeout occurs, the source increments CongestionWindow by **1** – additive increase.

Additive Increase Multiplicative Decrease

- For example
  - CongestionWindow is defined in terms of bytes;
  - Then,
    - Firstly, CongestionWindow is currently set to 16 packets. If a loss is detected, CongestionWindow is set to 8 packets.
    - Additional losses cause CongestionWindow to be reduced to 4, then 2, and finally to 1 packet.
    - CongestionWindow is not allowed to fall below the size of a single packet, or in TCP terminology, the *maximum segment size (MSS).*

Additive Increase Multiplicative Decrease

- Another example

Congestion Window =1

Source must receive 1 ACK

Congestion Window =2

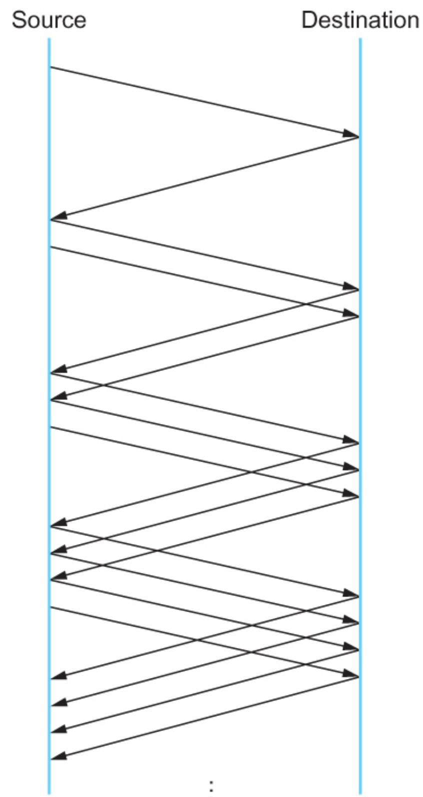
Source must receive 2 ACKs

Congestion Window =3

Source must receive 3 ACKs

Congestion Window =4

Source must receive 4 ACKs



Packets in transit during additive increase, with one packet being added each RTT.

Slow Start

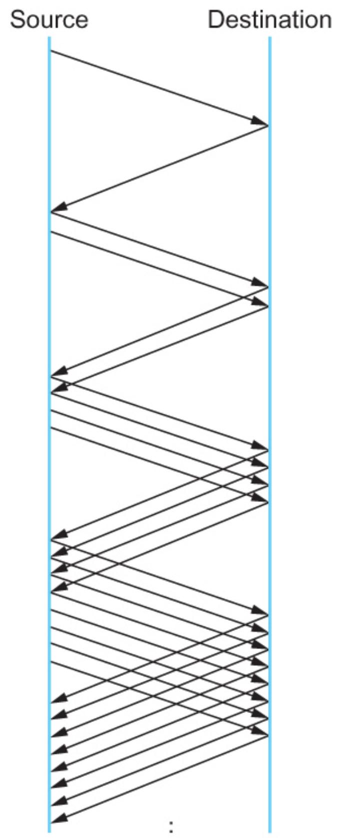
- Drawback of AIMD: too conservative! It takes too long to ramp up a connection starting from scratch.

- TCP therefore provides a second mechanism, ironically called *slow start,*
  - Increase the CongestionWindow rapidly from a cold start when the CongestionWindow size starts at 1;
  - Slow start effectively increases the congestion window exponentially, rather than linearly (additive increase).

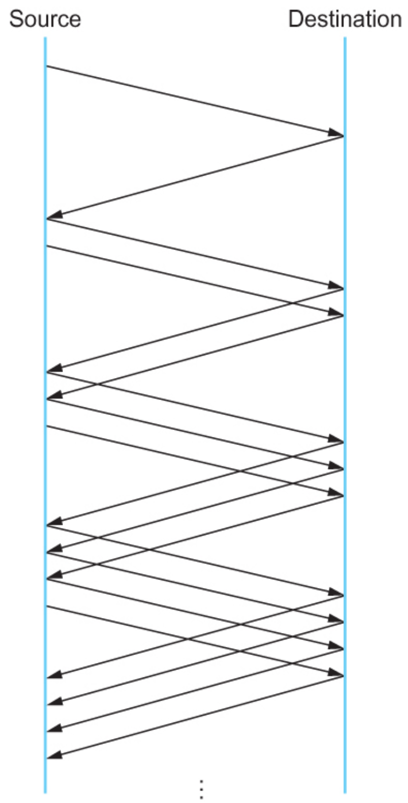
Slow Start

- For example
  - Source starts out by setting CongestionWindow to 1 packet.
  - When the ACK for this packet arrives, source adds 1 to CongestionWindow and then sends 2 packets.
  - Upon receiving the corresponding 2 ACKs, source increments CongestionWindow by 2 and next sends 4 packets.
  - The result is that source doubles the number of packets it has in transit every RTT.

Slow Start



Packets in transit during slow start.



Packets in transit during additive increase

Slow Start

- There are two situations in which slow start is favorable:
  - The first is at the very beginning of a connection:

    - Slow start continues to double CongestionWindow each RTT

    - After a packet is lost, a timeout causes multiplicative decrease to divide CongestionWindow by 2.

  - The second happens when the connection goes dead while waiting for a timeout to occur:

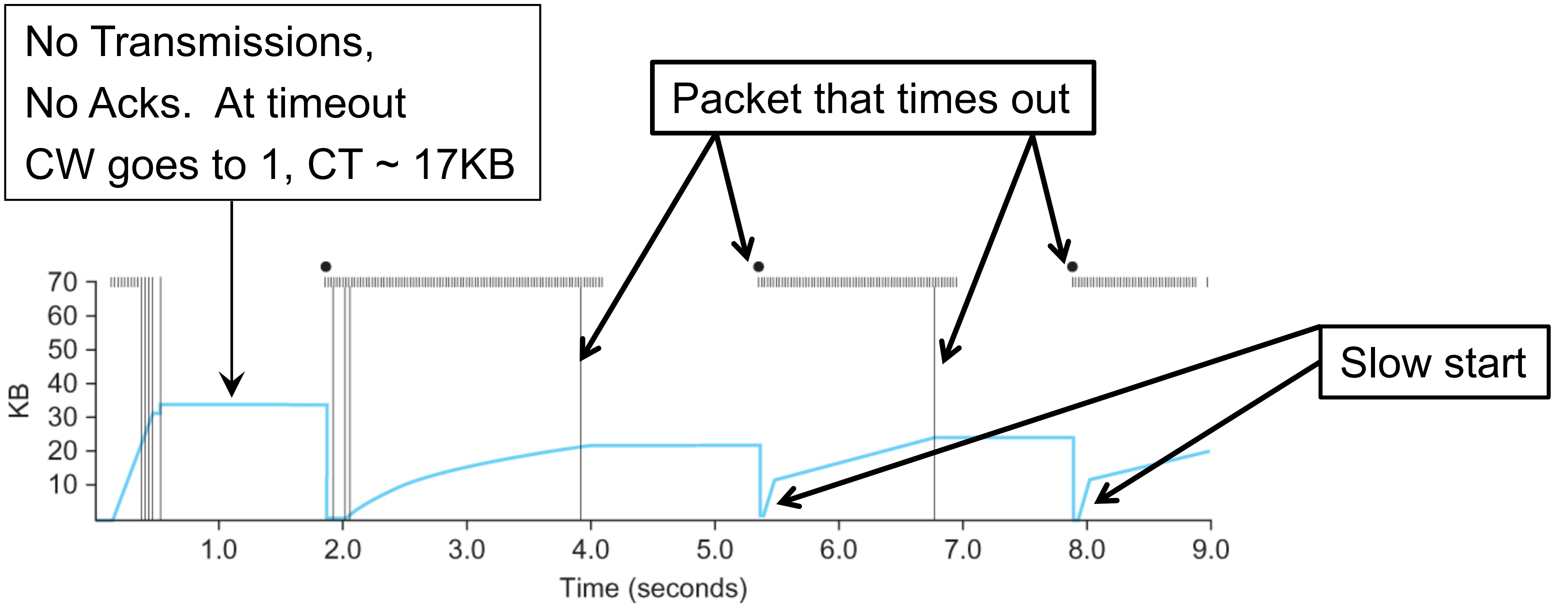
    - After communications for a while, source has a useful value of CongestionWindow, which is called CongestionThreshold;

    - Use CongestionThreshold as the "target" CongestionWindow.

    - Slow start is used to rapidly increase the sending rate up to this value, and then additive increase is used beyond this point. **(Combination! Fine-resolution!)**

Slow Start

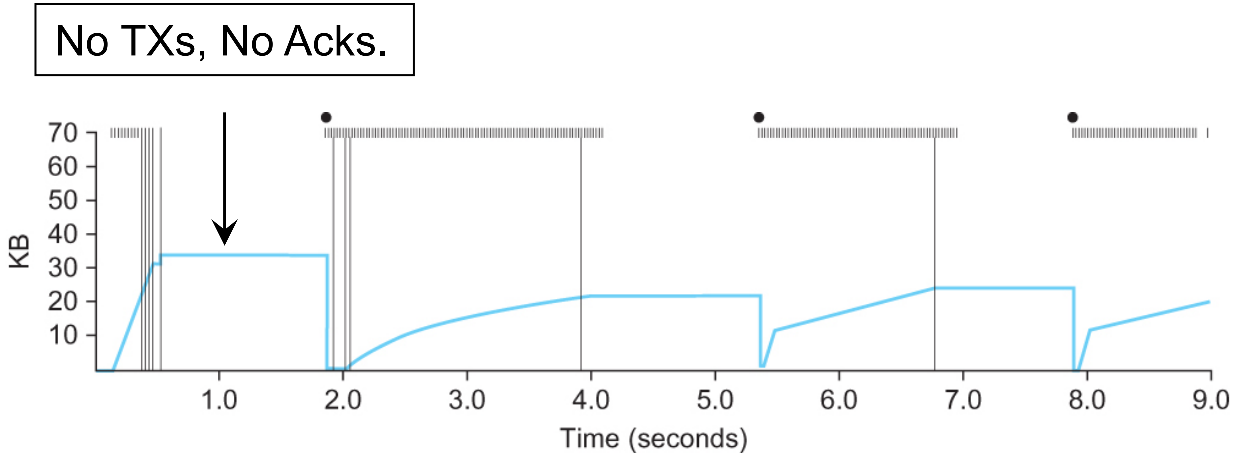
- Cont' – Example for timeout occurs



Behavior of TCP congestion control. Colored line = value of CongestionWindow over time; solid bullets at top of graph = timeouts; hash marks at top of graph = time when each packet is transmitted; vertical bars = time when a packet that was eventually retransmitted was first transmitted.

Fast Retransmit and Fast Recovery

- Issue of slow start + AIMD: TCP timeouts led to long periods of time during which the connection went dead while waiting for a timer to expire.



  - A new mechanism called fast retransmit was added to TCP.
  - It is a heuristic that sometimes triggers the retransmission of a dropped packet sooner than the timeout.

Fast Retransmit and Fast Recovery

- Basic idea
  - Receiver responds with an ACK every time a packet arrives - even if it has already been ACKed
  - When an out of order packet is received, TCP resends the same ACK as last time
  - TCP cannot ACK the data contained in an out of order packet

Receiver Assisted Approach

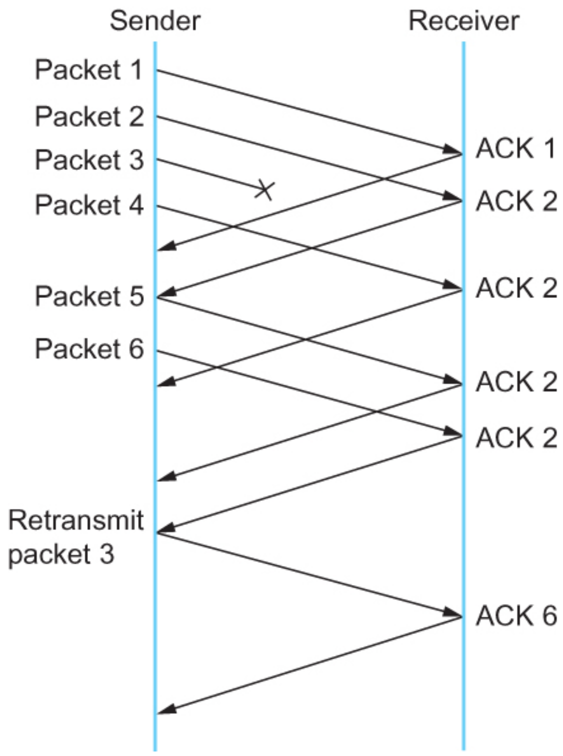
Fast Retransmit and Fast Recovery

- Basic idea – cont'
  - When the source sees a duplicate ACK,
    - It knows that the other side must have received a packet out of order,
    - Suggests that an earlier packet might have been lost.
  - In practice, TCP waits until it has seen three duplicate ACKs before retransmitting the packet.

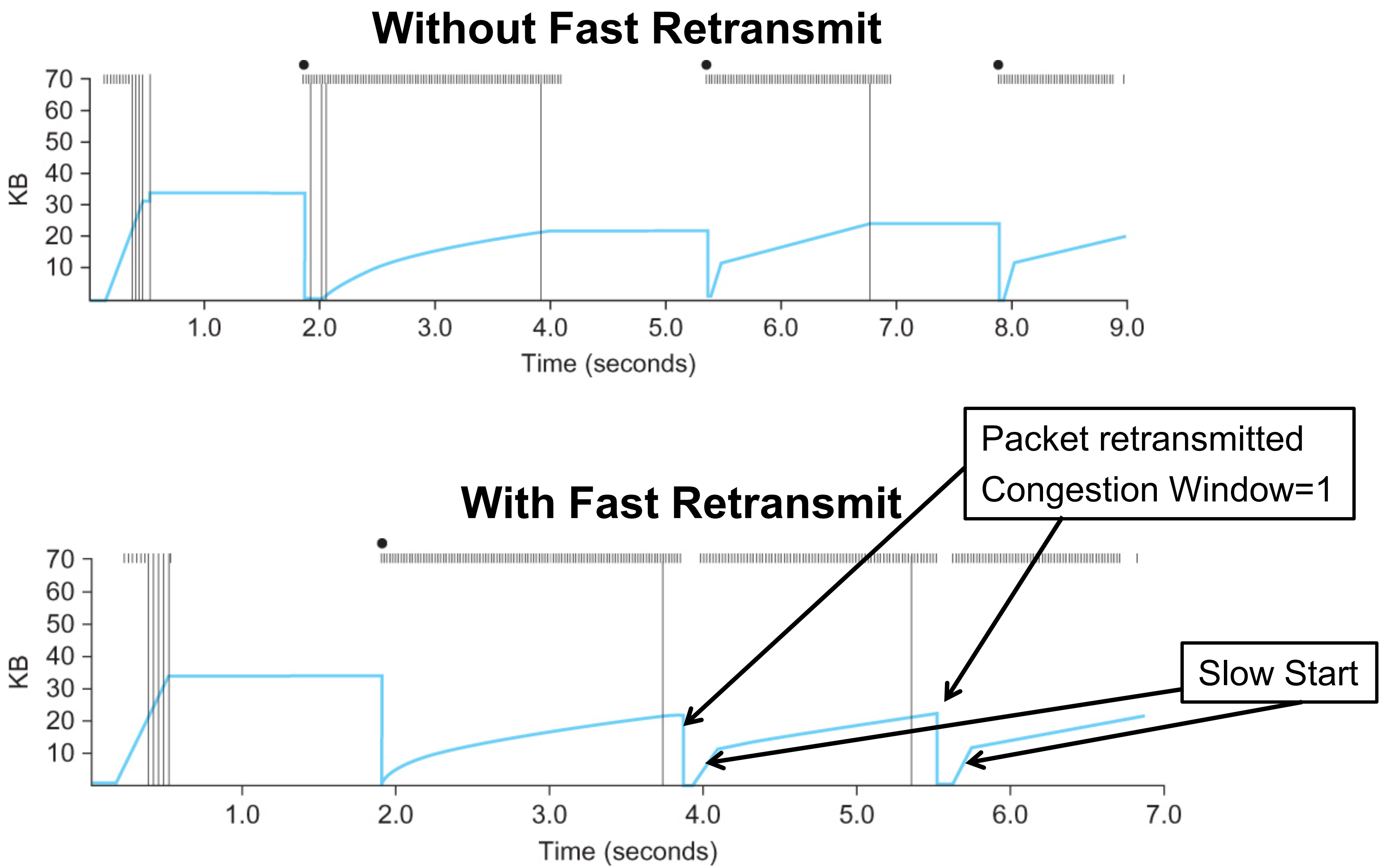
    - in case the packet is delayed instead of lost

Fast Retransmit and Fast Recovery

- Example



Fast Retransmit and Fast Recovery



Fast Retransmit and Fast Recovery

- Fast Recovery
  - When the fast retransmit mechanism signals congestion,
    - NOT drop the CongestionWindow back to 1 packet and run slow start,
    - it is possible to use the ACKs that are still in the pipe to clock the sending of packets.
  - This mechanism, which is called fast recovery.

TCP Congestion Control Summary

- In this lecture, we will talked about:
  - TCP congestion control protocols;
    - Additive Increase/Multiplicative Decrease
    - Slow Start
    - Fast Retransmit and Fast Recovery
  - The (dis-)advantages of them;
  - You are expected to draw the tx diagram given a specific TCP congestion control protocol;