TCP & Congestion Control

MẠNG MÁY TÍNH NÂNG CAO

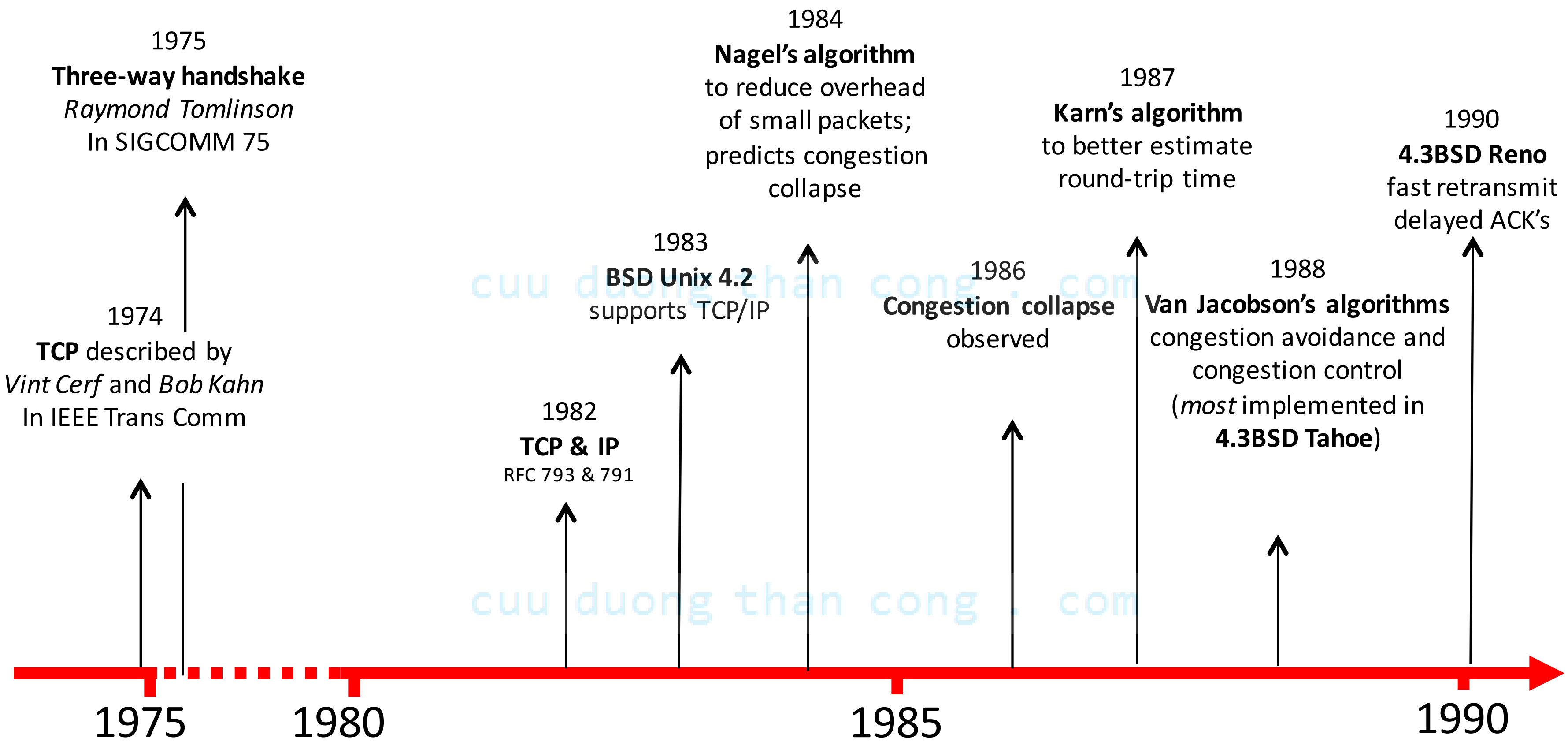
Tháng 09/2015

Introduction to TCP

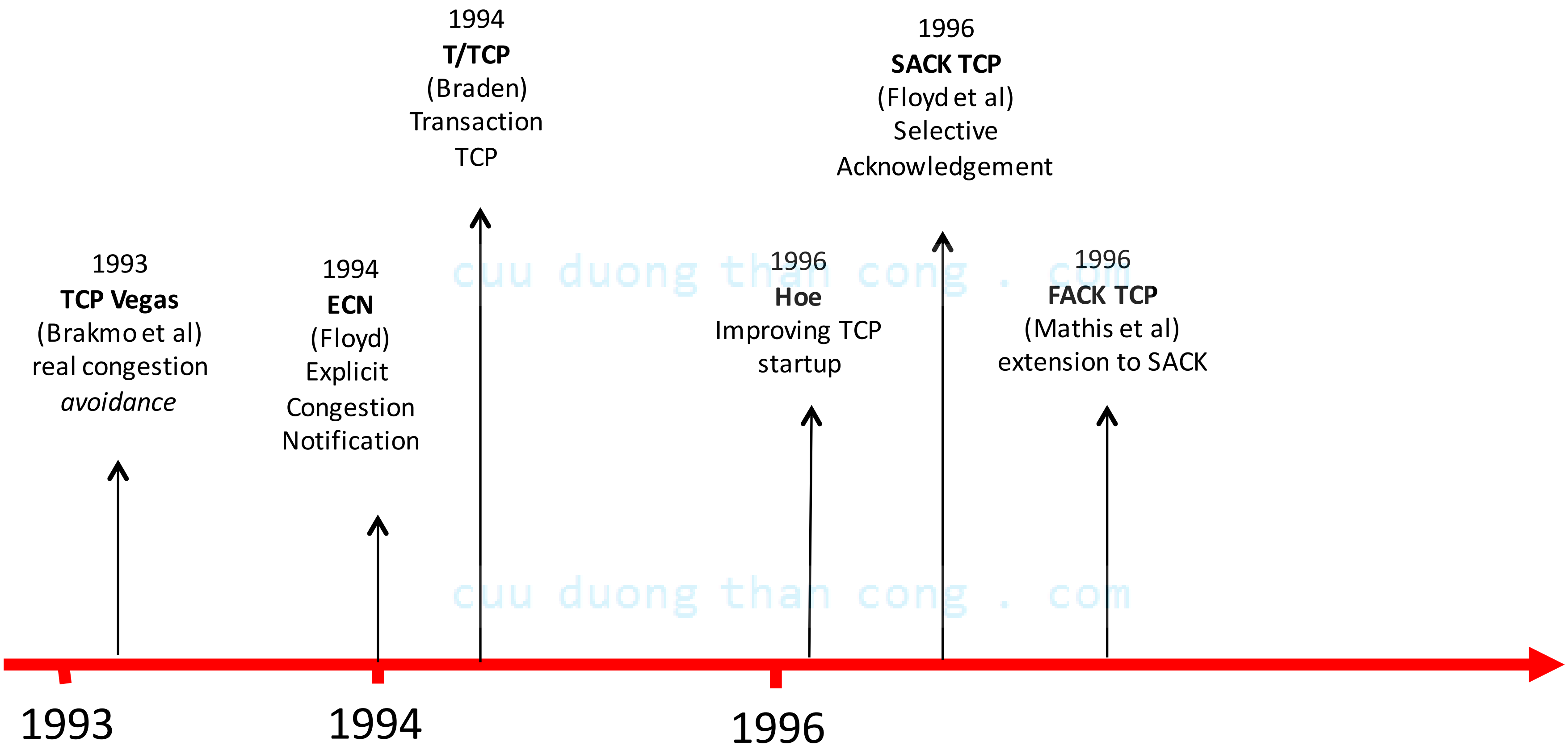
- Communication abstraction:
  - Reliable
  - Ordered
  - Point-to-point
  - Byte-stream
  - Full duplex
  - Flow and congestion controlled

- Sliding window with cumulative acks
  - Ack field contains last in-order packet received
  - Duplicate acks sent when out-of-order packet received

Evolution of TCP



TCP Through the 1990s



Flow Control vs. Congestion Control

- Flow control
  - Keeping *one fast sender* from overwhelming *a slow receiver*
- Congestion control
  - Keep a *set of senders* from overloading the *network*

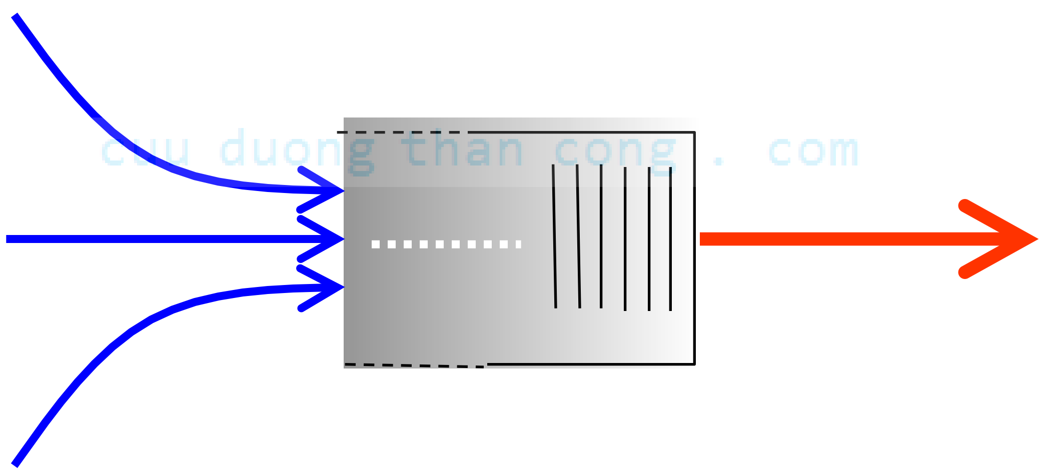
- Different concepts, but similar mechanisms
  - TCP flow control: receiver window
  - TCP congestion control: congestion window
  - TCP window: min{congestion window, receiver window}

Three Key Features of Internet

- Packet switching
  - A given source may have enough capacity to send data
  - … and yet the packets may encounter an overloaded link
- Connectionless flows
  - No notions of connections inside the network
  - … and no advance reservation of network resources
  - Still, you can view related packets as a group ("flow")
  - … e.g., the packets in the same TCP transfer
- Best-effort service
  - No guarantees for packet delivery or delay
  - No preferential treatment for certain packets

Congestion is Unavoidable

- Two packets arrive at the same time
  - The node can only transmit one
  - … and either buffer or drop the other
- If many packets arrive in a short period of time
  - The node cannot keep up with the arriving traffic
  - … and the buffer may eventually overflow



Why prevent congestion ?

- Congestion is bad for the overall performance in the network.
  - Excessive delays can be caused.
  - Retransmissions may result due to dropped packets
    - Waste of capacity and resources.
  - Note: Main reason for lost packets in the Internet is due to congestion -- errors are rare.

The Congestion Window

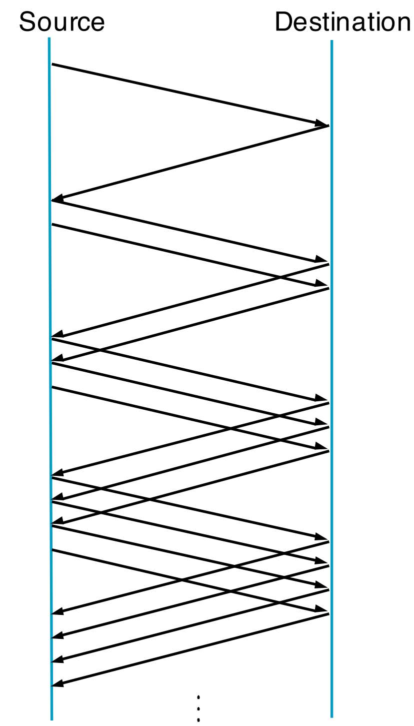
- In order to deal with congestion, a new state variable called "CongestionWindow" is maintained by the source.
  - Limits the amount of data that it has in transit at a given time.
- MaxWindow = Min(Advertised Window, CongestionWindow)
- EffectiveWindow = MaxWindow - (LastByteSent - LastByteAcked).
- TCP sends no faster than what the slowest component -- the network or the destination host --can accommodate.

Managing the Congestion Window

- Decrease window when TCP perceives high congestion.
- Increase window when TCP knows that there is not much congestion.
- How ? Since increased congestion is more catastrophic, reduce it more aggressively.
- Increase is additive, decrease is multiplicative -- called the Additive Increase/Multiplicative Decrease (AIMD) behavior of TCP.

AIMD details

- Each time congestion occurs - the congestion window is halved.
  - Example, if current window is 16 segments and a time-out occurs (implies packet loss), reduce the window to 8.
  - Finally window may be reduced to 1 segment.
- Window is not allowed to fall below 1 segment (MSS).
- For each congestion window worth of packets that has been sent out successfully (an ACK is received), increase the congestion window by the size of a (one) segment.

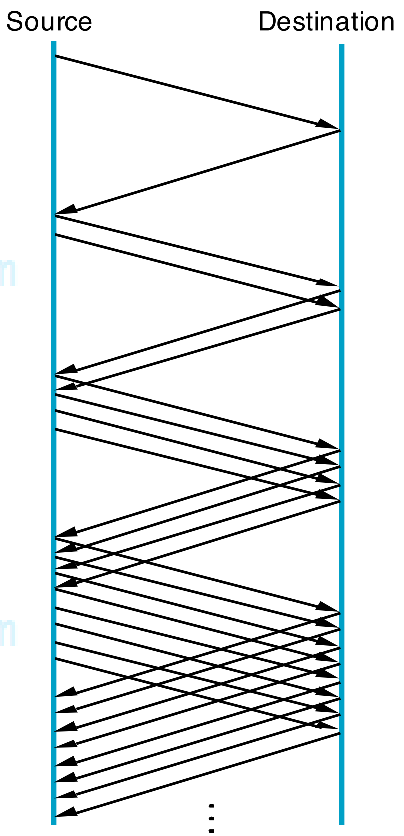


TCP Slow Start

- Additive Increase is good when source is operating at near close to the capacity of the network.
  - Too long to ramp up when it starts from scratch.
  - Slow start --> increase congestion window rapidly at cold start.
- Slow start allows for exponential growth in the beginning.

  E.g. Initially CW =1, if ACK received, CW = 2.

  If 2 ACKs are now received, CW = 4. If 4 ACKs are now received, CW =8 and so on.
- Note that upon experiencing packet loss, multiplicative decrease takes over.



Where does AIMD come in now ?

- Slow start is used to increase the rate to a "target window size" prior to AIMD taking over.
- What is this target window size ?
- In addition, we now have to do book keeping for two windows -- the congestion window and the "target congestion window" where Slow start ends and AIMD begins.

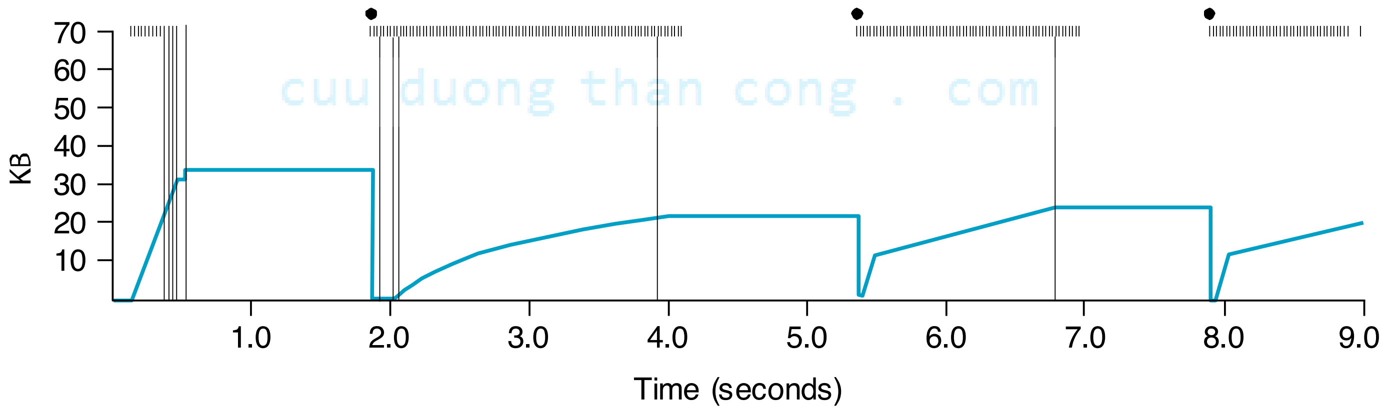
The Congestion Threshold

- Initially no target window -- when a packet loss occurs, divide the current CW by 2 (due to multiplicative decrease) -- this now becomes the target window.
- Define this to be the “Congestion Threshold”.
- Reduce actual CW to 1.
- Use Slow Start to ramp up to the Congestion Threshold (or simply threshold). Once this is reached use AIMD.

Summary: TCP Tahoe

- Thus:
  - When CW is below the threshold, CW grows exponentially
  - When it is above the threshold, CW grows linearly.
  - Upon time-out, set "new" threshold to half of current CW and the CW is reset to 1.

  - This version of TCP is called "TCP Tahoe".

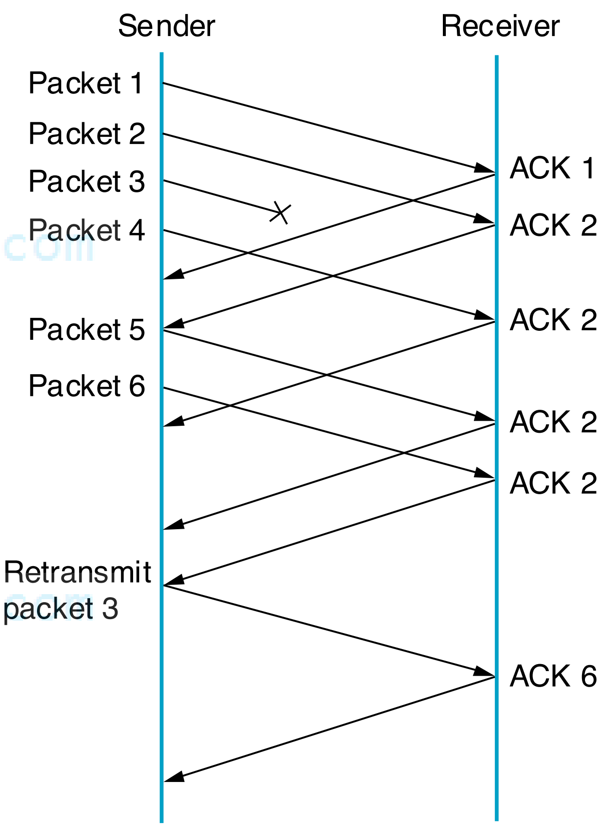


Fast Retransmit

- 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

Duplicate ACKs

- When a duplicate ACK is seen by the sender, it infers that the other side must have received a packet out of order.
  - Delays on different paths could be different -- thus, the missing packets may be delivered.
  - So wait for "some" number of duplicate ACKs before resending data.
  - This number is usually 3.



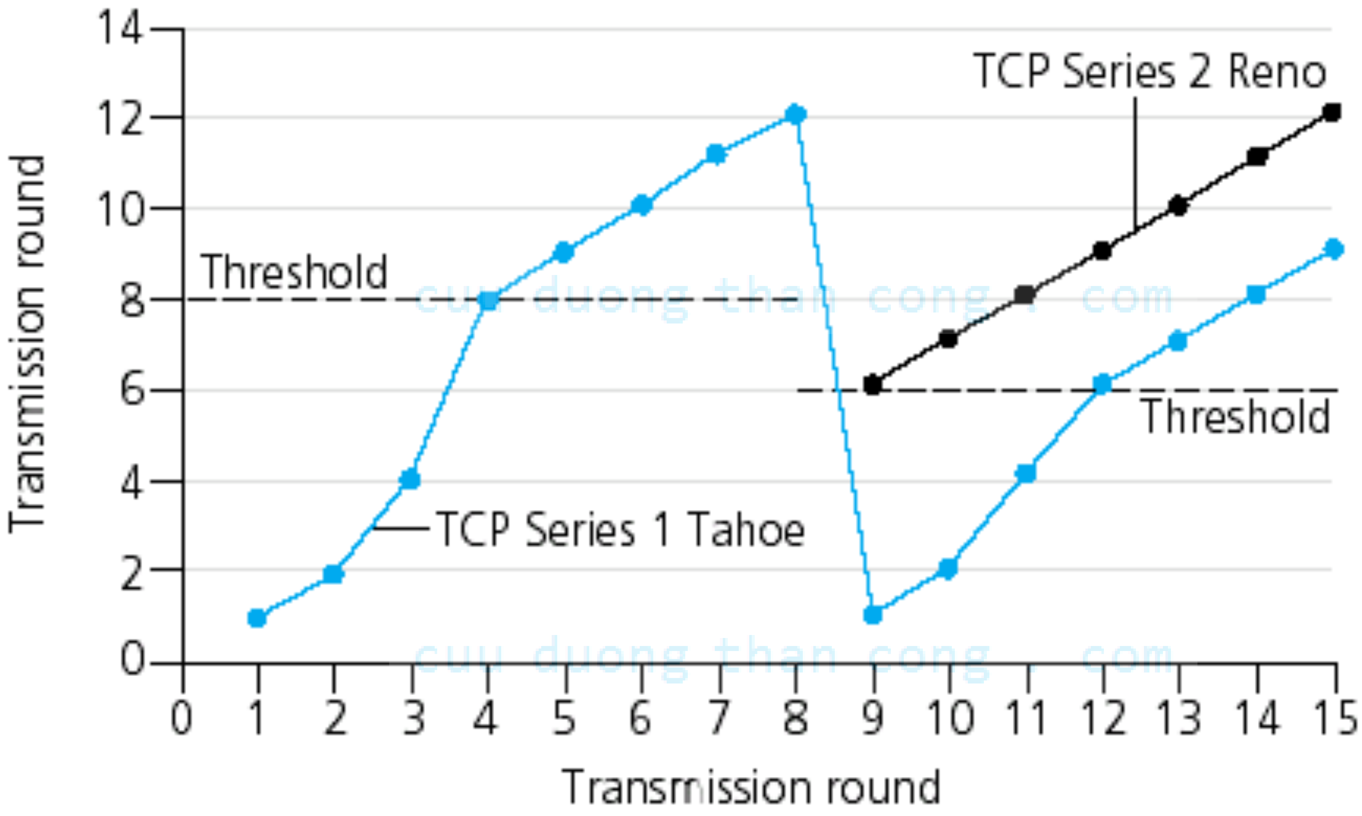
Fast Recovery

- When the fast retransmit mechanism signals congestion, the sender, instead of returning to Slow Start uses a pure AIMD.
  - Simply reduces the congestion window by half and resumes additive increase.
- Thus, recovery is faster -- this is called Fast Recovery.

TCP Reno

- The version of TCP wherein fast retransmit and fast recovery are added in addition to previous congestion control mechanisms is called TCP Reno.
  - Has other features -- header compression (if ACKs are being received regularly,omit some fields of TCP header).
  - Delayed ACKs -- ACK only every other segment.

Summary - TCP Congestion Control



Summary: TCP Congestion Control

- when **`cwnd < ssthresh`**, sender in slow-start phase, window grows exponentially.
- when **`cwnd >= ssthresh`**, sender is in congestion-avoidance phase, window grows linearly.
- when triple duplicate ACK occurs, **`ssthresh`** set to **`cwnd/2`**, **`cwnd`** set to ~ **`ssthresh`**
- when timeout occurs, **`ssthresh`** set to **`cwnd/2`**, **`cwnd`** set to 1 MSS.

Other flavors

- TCP NewReno
- TCP Vegas
- SACK TCP
- FACK TCP

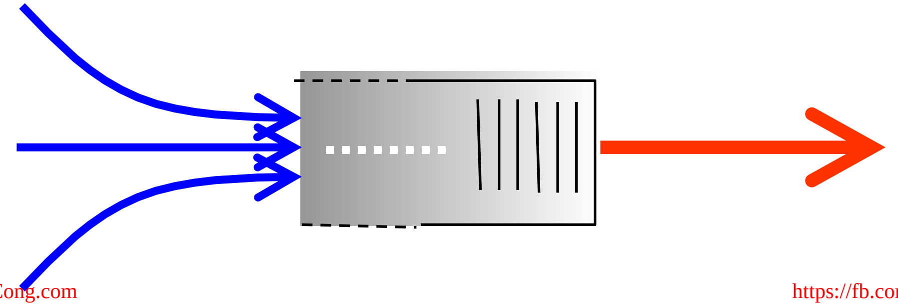
Queuing Mechanisms

Random Early Detection (RED)

Explicit Congestion Notification (ECN)

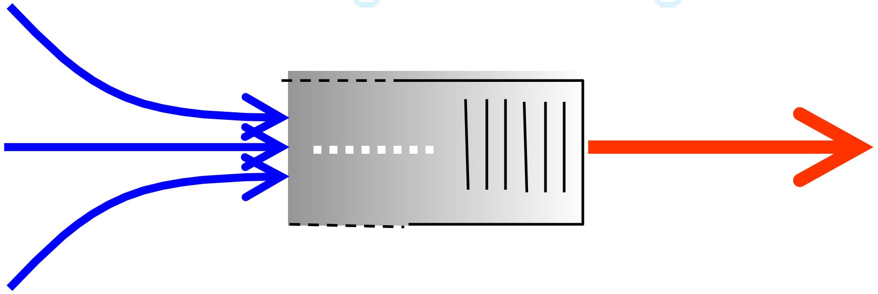
Bursty Loss From Drop-Tail Queuing

- TCP depends on packet loss
  - Packet loss is the indication of congestion
  - In fact, TCP *drives* the network into packet loss
  - … by continuing to increase the sending rate
- Drop-tail queuing leads to *bursty* loss
  - When a link becomes congested…
  - … many arriving packets encounter a full queue
  - And, as a result, many flows divide sending rate in half
  - … and, many individual flows lose multiple packets



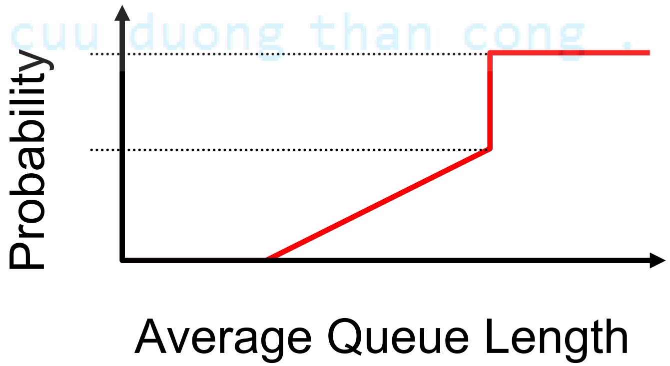
Slow Feedback from Drop Tail

- Feedback comes when buffer is completely full
  - … even though the buffer has been filling for a while
- Plus, the filling buffer is increasing RTT
  - … and the variance in the RTT
- Might be better to give early feedback
  - Get one or two flows to slow down, not all of them
  - Get these flows to slow down before it is too late



Random Early Detection (RED)

- Basic idea of RED
  - Router notices that the queue is getting backlogged
  - … and randomly drops packets to signal congestion
- Packet drop probability
  - Drop probability increases as queue length increases
  - If buffer is below some level, don't drop anything
  - … otherwise, set drop probability as function of queue



Properties of RED

- Drops packets before queue is full
  - In the hope of reducing the rates of some flows
- Drops packet in proportion to each flow's rate
  - High-rate flows have more packets
  - … and, hence, a higher chance of being selected
- Drops are spaced out in time
  - Which should help desynchronize the TCP senders
- Tolerant of burstiness in the traffic
  - By basing the decisions on *average* queue length

More RED Details

- With RED, two thresholds are maintained -- the MinThreshold and MaxThreshold.
- If AvgLen <= MinThreshold queue packet
- If AvgLen >= MaxThreshold drop arriving packet.
- If MinThreshold <= AvgLen <= MaxThreshold, then, calculate a drop probabilty P (as we will see) and drop the arriving packet with the probability P.

Problems With RED

- Hard to get the tunable parameters just right
  - How early to start dropping packets?
  - What slope for the increase in drop probability?
  - What time scale for averaging the queue length?
- Sometimes RED helps but sometimes not
  - If the parameters aren't set right, RED doesn't help
  - And it is hard to know how to set the parameters
- RED is implemented in practice
  - But, often not used due to the challenges of tuning right
- Many variations
  - With cute names like "Blue" and "FRED"… ☺

Explicit Congestion Notification

- Early dropping of packets
  - Good: gives early feedback
  - Bad: has to drop the packet to give the feedback
- Explicit Congestion Notification
  - Router marks the packet with an ECN bit
  - … and sending host interprets as a sign of congestion
- Surmounting the challenges
  - Must be supported by the end hosts and the routers
  - Requires two bits in the IP header (one for the ECN mark, and one to indicate the ECN capability)
  - Solution: borrow two of the Type-Of-Service bits in the IPv4 packet header

Questions

- ❑ Compare TCP Tahoe, Reno, NewReno, Vegas, SACK
- ❑ Pros and Cons ?