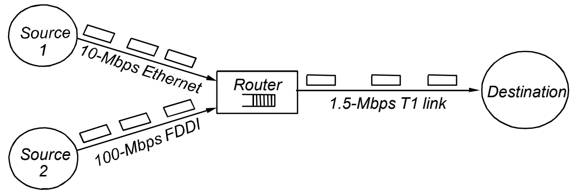
 Copyright ©by Zygmunt J. Haas, 2017

What is congestion?

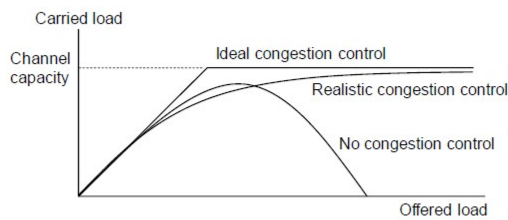
- Increase in network load results in decrease of useful work done
  - Different sources compete for resources inside network
  - Why is it a problem?
    - Sources are unaware of current state of resource
    - Sources are unaware of each other
    - In many situations, this will result in decrease in throughput (congestion collapse)



 Copyright ©by Zygmunt J. Haas, 2017

What is congestion?

- *Resource Allocation:* the process through which the network trying to meet the (competing) demands of the application, allocates the network resources, such as bandwidth, buffer space, etc.
- *Congestion Control:* the process by which the network act to prevent or responds to network overload conditions.
- The problem with congestion control is that these operations need to be perform on multiple network layers.



 Copyright ©by Zygmunt J. Haas, 2017

What is congestion?

- Another problem (challenge) is that these operations need to be "fair" to all network users. Why?
- *Flow Control* vs. *Congestion Control* - know the difference
- Congestion is primarily an issue in packet-switched network.
- Indeed, congestion can be avoided by pre-allocation of resources (such as is done in connection-oriented networks, such as virtual circuit switching).
- The notion of a "bottleneck" router.
- The notion of a *flow* used for resource allocation.
- The notion of a *soft state,* as between connectionless and connection-oriented operation.

 Copyright ©by Zygmunt J. Haas, 2017

Issues in Congestion Control

- How to deal with congestion?
  - pre-allocate resources so as to avoid congestion (avoidance)
  - control congestion if (and when) it occurs (control)
- Underlying service model
  - best-effort data delivery
- Router-centric vs. Host-centric
  - hosts at the edges of the network (transport protocol)
  - routers inside the network (queuing discipline)
- Reservation-based vs. Feedback-based
  - Reservation-based is router-centric
  - Feedback-based can be *explicit* (typically router-centric) or *implicit* (typically host-centric)

Copyright ©by Zygmunt J. Haas, 2017

Issues in Congestion Control

- Window-based vs. Rate-based
  - Similar mechanisms for flow-control and congestion-control
  - TCP congestion control is window-based
  - Rate-based is used in reservation-based systems supporting different QoS.
- Evaluation Criteria
  - Delay vs. Throughput tradeoff:
  - Higher utilization → larger throughput
  - Higher utilization → larger delay
- Evaluation Criterion for efficient use of resources (power index)
  - $Power = \frac{(Throughput)^{\alpha}}{Delay}; \quad 0 < \alpha < 1$
- Evaluation Criterion for fairness (Jain's fairness index)
  - $f(x_1, x_2, \dots x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} (x_i)^2}; \quad 0 \le f \le 1$

Copyright ©by Zygmunt J. Haas, 2017

TCP Congestion Control

- Idea
  - assumes best-effort network (FIFO or FQ routers)
  - each source determines network capacity for itself
  - uses implicit feedback
  - ACKs pace transmission (*self-clocking*)
- Challenge
  - determining the available capacity in the first place
  - adjusting to changes in the available capacity
- TCP sender is in one of two states:
  - slow start OR congestion avoidance

 Copyright ©by Zygmunt J. Haas, 2017

TCP Congestion Control

- Three components of implementation

  Original TCP (TCP Tahoe)
  - 1. Slow Start

    2. Additive Increase/Multiplicative Decrease (AIMD)

    3. Fast Retransmit
- TCP Reno
  - 3. Fast Recovery
- TCP Vegas
  - Introduces Congestion Avoidance
- Objective: adjust to changes in the available capacity

 Copyright ©by Zygmunt J. Haas, 2017

TCP Congestion Control

- New state variables per connection: **`CongestionWindow`** and (slow start) **`threshold`**
  - limits how much data source has in transit

```
MaxWindow = MIN(CongestionWindow,
                AdvertisedWindow)
EffectiveWindow = MaxWindow -
               (LastByteSent - LastByteAcked)
```

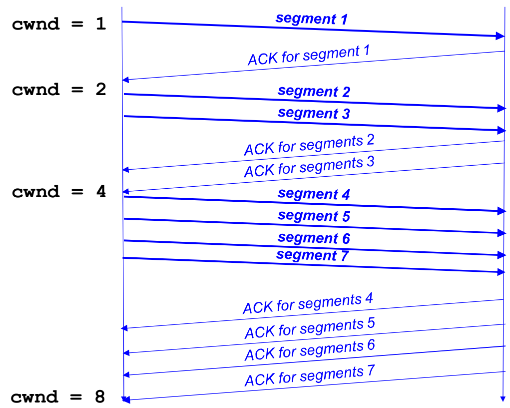
- I.e., TCP is allowed to transmit no faster than either the network or the destination can support.

Slow Start Mechanism

- Initial value: **`Set cwnd = 1`**
  - Note: Unit is a MSS. TCP actually is based on bytes and increments by 1 MSS (maximum segment size)
- The receiver sends an acknowledgement (ACK) for each segment
  - Note: Generally, a TCP receiver sends an ACK for every other segment.
- Each time an ACK is received by the sender, the congestion window is increased by 1 segment: **`cwnd = cwnd + 1`**
  - If an ACK acknowledges two segments, **`cwnd`** is still increased by only 1 segment.
  - Even if ACK acknowledges a segment that is smaller than MSS bytes long, **`cwnd`** is still increased by 1.
- Does Slow Start increment slowly? Not really. In fact, the increase of **`cwnd`** is exponential (Why?)

Slow Start Example

- The congestion window size grows very rapidly
  - For every ACK, we increase **cwnd** by 1 irrespective of the number of segments ACK'ed
- TCP slows down the increase of **cwnd** when **cwnd > ssthresh**
- Thus, **cwnd** is doubled every RTT



Congestion Avoidance via AIMD

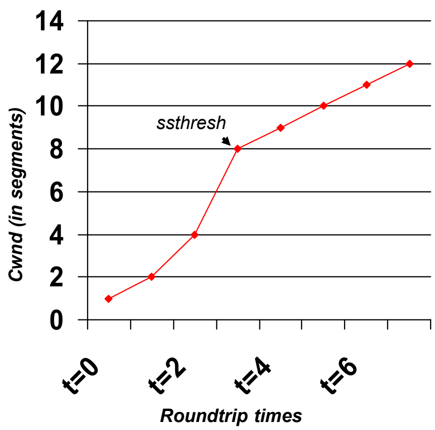
- Congestion avoidance phase is started if **cwnd** has reached the slow-start threshold value
- If **cwnd >= ssthresh** then each time an ACK is received, increment **cwnd** as follows:
  - **cwnd = cwnd + 1/cwnd**
- So **cwnd** is increased by one only if all **cwnd** segments have been acknowledged.

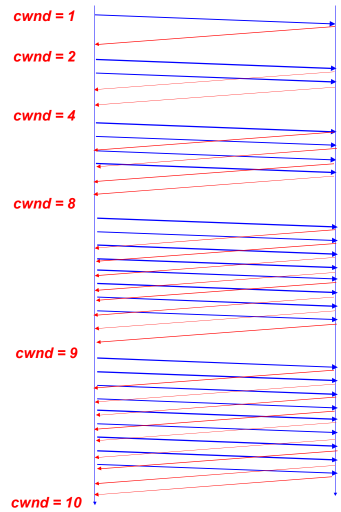
Increment = MSSx(MSS/CongestionWindow)
CongestionWindow+=Increment

- Thus, **cwnd** is increased by 1 every RTT.

Example of Slow Start/Congestion Avoidance

Assume that **ssthresh** = 8





 Copyright ©by Zygmunt J. Haas, 2017

Responses to Congestion

- TCP assumes there is congestion if it detects a packet loss
- A TCP sender can detect lost packets via:
    - Expiration of a retransmission timer
    - Receipt of a duplicate ACK
- TCP interprets a Timeout as a binary congestion signal. When a timeout occurs, the sender performs:
  - **cwnd** is reset to one:

    cwnd = 1
  - **ssthresh** is set to half the current size of the congestion window:

    ssthresh = cwnd / 2
  - and slow-start phase is entered

 Copyright ©by Zygmunt J. Haas, 2017

Summary of TCP congestion control

Initially:
  cwnd = 1;
  ssthresh =
    advertised window size;
New Ack received:
  if (cwnd < ssthresh)
    /* Slow Start*/
    cwnd = cwnd + 1;
  else
    /* Cong. Avoidance */
    cwnd = cwnd + 1/cwnd;
Timeout:
  /* Multiplicative decrease */
  ssthresh = cwnd/2;
  cwnd = 1;

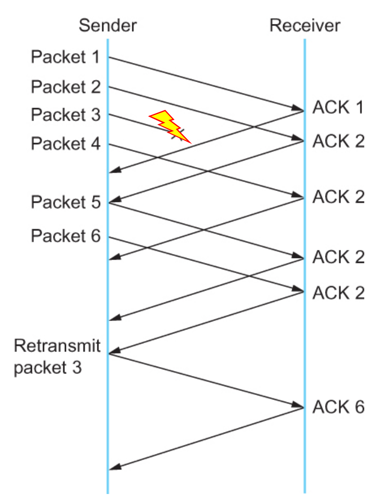
 Copyright ©by Zygmunt J. Haas, 2017

Fast Retransmit

- If three or more duplicate ACKs are received in a row, the TCP sender believes that a segment has been lost.
- Then TCP performs a retransmission of what seems to be the missing segment, without waiting for a timeout to happen.
- Enter slow start:

  `ssthresh = cwnd/2`

  `cwnd = 1`



 Copyright ©by Zygmunt J. Haas, 2017

Flavors of TCP Congestion Control

- **TCP Tahoe** (1988, FreeBSD 4.3 Tahoe)
  - Slow Start
  - Congestion Avoidance
  - Fast Retransmit
- **TCP Reno** (1990, FreeBSD 4.3 Reno)
  - Fast Recovery
- **New Reno** (1996)
- **SACK** (1996)
- **RED** (Floyd and Jacobson 1993)

Copyright ©by Zygmunt J. Haas, 2017

TCP Reno

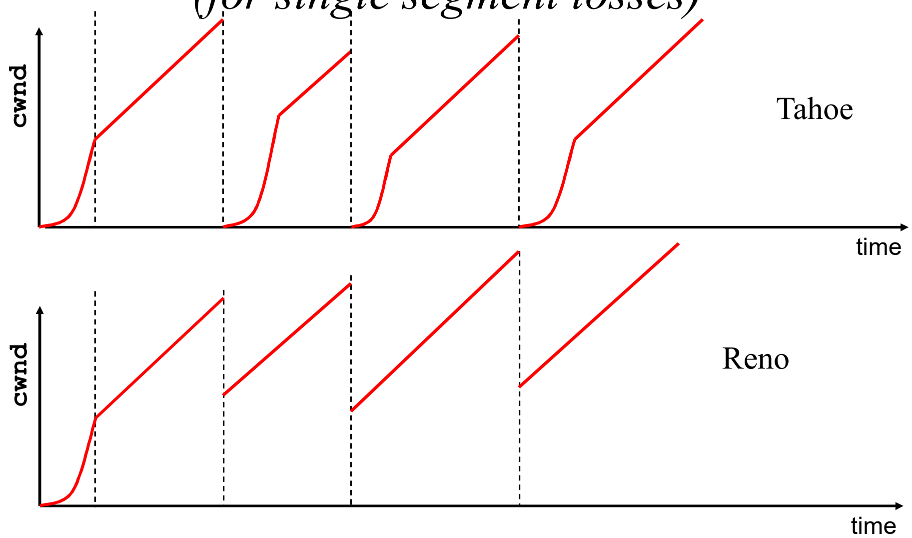
- Duplicate ACKs:
  - Fast retransmit
  - Fast recovery

  → Fast Recovery avoids slow start
- Timeout:
  - Retransmit
  - Slow Start
- TCP Reno improves upon TCP Tahoe when a single packet is dropped in a round-trip time.
- Fast recovery avoids slow start after a fast retransmit
- Intuition: Duplicate ACKs indicate that data is getting through
- On packet loss detected by three duplicate ACKs:
  - **`ssthresh = cwnd/2`**
  - **`cwnd = ssthresh`**

  enter congestion avoidance

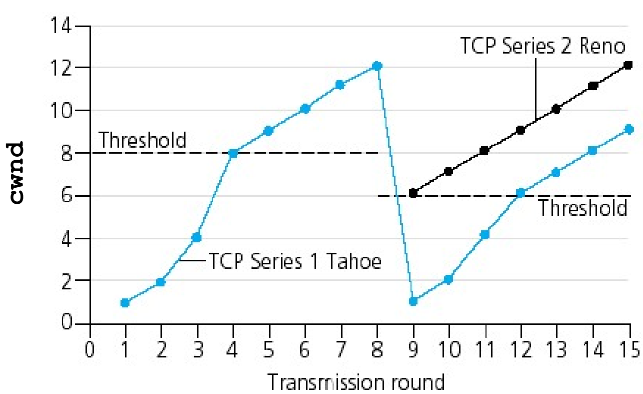
Copyright ©by Zygmunt J. Haas, 2017

TCP Tahoe and TCP Reno (for single segment losses)



Copyright ©by Zygmunt J. Haas, 2017

TCP CC



Copyright ©by Zygmunt J. Haas, 2017

Congestion Avoidance

- TCP's strategy
  - control congestion once it happens
  - repeatedly increase load in an effort to find the point at which congestion occurs and then back off
- Alternative strategy
  - predict when congestion is about to happen
  - reduce rate before packets start being discarded
  - call this *congestion avoidance*, instead of *congestion control*
- Two possibilities
  - host-centric: TCP Vegas
  - router-centric: DECbit and RED Gateways

Copyright ©by Zygmunt J. Haas, 2017

Algorithm

- Let **`BaseRTT`** be the minimum of all measured RTTs (commonly the RTT of the first packet)
- If not overflowing the connection, then

  `ExpectRate = CongestionWindow/BaseRTT`
- Source calculates sending rate (**`ActualRate`**) once per RTT
- Source compares **`ActualRate`** with **`ExpectRate`**

```
Diff = ExpectRate - ActualRate
if Diff < α
    increase CongestionWindow linearly
else if Diff > β
    decrease CongestionWindow linearly
else
    leave CongestionWindow unchanged
```


Copyright ©by Zygmunt J. Haas, 2017