Congestion Control

INF3190

Foreleser: Carsten Griwodz
Email: griff@ifi.uio.no
Congestion and Flow Control

Opposite objectives
- **End-system**
  - Optimize its own throughput
  - Possibly at the expense of other end-systems
- **Network**
  - Optimize overall throughput

Two different problems
- Receiver capacity
- Network capacity

Cannot be distinguished easily at all places
But should be differentiated
Congestion

- 2 problem areas
  - Receiver capacity
    - Approached by flow control
  - Network capacity
    - Approached by congestion control

- Possible approach to avoid both bottlenecks
  - Receiver capacity: “actual window”, credit window
  - Network capacity: “congestion window”
  - Valid send window = min(actual window, congestion window)

- Terms
  - Traffic
    - All packets from all sources
  - Traffic class
    - All packets from all sources with a common distinguishing property, e.g. priority
**Congestion**

- **Traffic**
  - All traffic from all sources

- **Traffic class**
  - All packets from all sources with a common distinguishing property, e.g. priority

- **Persistent congestion**
  - Router stays congested for a long time
  - Excessive traffic offered

- **Transient congestion**
  - Congestion occurs for a while
  - Router is temporarily overloaded
  - Often due to burstiness
  - **Burstiness**
    - Average rate $r$
    - Burst size $b$ (#packets that appear at the same time)
    - Token bucket model
Congestion

- **Traffic**
  - All traffic from all sources

- **Traffic class**
  - All packets from all sources with a common distinguishing property, e.g. priority

- **Persistent congestion**
  - Router stays congested for a long time
  - Excessive traffic offered

- **Transient congestion**
  - Congestion occurs for a while
  - Router is temporarily overloaded
  - Often due to *burstiness*
Reasons for congestions

- Two senders, two receivers
- One router, infinite buffers
- No retransmissions

- Very long delays in case of congestion
- Maximum utilization of the networks
Reasons for congestions

- Two senders, two receivers
- One router, finite buffers
- Retransmission of lost packets

\[ \lambda_{in} \] Data rate sent by the application

\[ \lambda_{in}' \] Higher data rate sent by the transport layer including retransmissions
Reasons for congestions

- Always $\lambda_{in} = \lambda_{out}$
- Perfect retransmission only in case of loss $\lambda_{in}' > \lambda_{out}$
- Retransmission of delayed (but not lost) packets increases $\lambda_{in}'$ above the perfect value, without increasing $\lambda_{out}$

“Cost of congestion”:
- More work (retransmissions) for a desired throughput
- Useless retransmissions, some links transmit several copies of the same packet
Reasons for congestions

- Always $\lambda_{in} = \lambda_{out}$
- Perfect retransmission only in case of loss $\lambda'_{in} > \lambda_{out}$
- Retransmission of delayed (but not lost) packets increases $\lambda'_{in}$ above the perfect value, without increasing $\lambda_{out}$

“Cost of congestion”:
- More work (retransmissions) for a desired throughput
- Useless retransmissions, some links transmit several copies of the same packet
Reasons for congestions

- No congestion / persistent congestion
  - Constant bitrate (non-bursty) traffic
  - Bursty traffic

- Transient congestion
  - \( \lambda_{in} = \lambda_{in} \)
  - \( \lambda_{in} \rightarrow \infty \)

\[
\lambda_{in} = \lambda_{in} \quad \lambda_{out} = \frac{C}{2}
\]

\[
\lambda_{out} = \frac{C}{2} \quad \lambda_{in} = \lambda_{in}
\]
Reasons for congestions

- 4 senders
- Paths with several hops
- Bursty traffic
- Timeout and retransmissions

Q: What happens when $\lambda_{in}$ and $\lambda_{in}'$ grow?
Reasons for congestions

- Another cost of congestion:
  - When packets are dropped, all network capacity that is consumed for transporting them so far is wasted.
Congestion

- Why is that so?
  - asd

Make a figure at the blackboard that shows how many streams irregular traffic can lead to increased burstiness

Add the point about network calculus to the slides at this point

The token bucket spec must be known before this point
Congestion

- Reasons for congestion, among others
  - Incoming traffic overloads outgoing lines
  - Router too slow for routing algorithms
  - Too little buffer space in router

- When too much traffic is offered
  - Congestion sets in
  - Packets are dropped
  - Packets that are not dropped have a long latency
  - **Application performance** degrades sharply

Congestion tends to amplify itself
- Network layer: unreliable service
  - Router simply drops packet due to congestion
- Transport layer: reliable service
  - Packet is retransmitted

\[ \text{Congestion} \Rightarrow \text{More delays at end-systems} \]
\[ \Rightarrow \text{Retransmissions} \]
\[ \Rightarrow \text{Additional traffic} \]
Approaches to congestion control
Congestion Control

- Strategies
  - Increase capacity
  - Decrease traffic
Congestion Control

- Strategies

- End-to-end congestion control
  - no explicit feedback from the network
  - congestion is detected by observed packet loss and delay
  - this is the approach of basic (regular) TCP

- Network-assisted congestion control
  - router give feedback to end systems
  - choke packets (SNA, DECbit, ICMP, TCP/IP ECN, ATM)
  - explicit send rate (ATM, XCP)
Congestion Control

- **Strategies**
  - **Repair**
    - when congestion is noticed
    - explicit feedback (packets are sent from the point of congestion)
    - implicit feedback (source assumes that congestion occurred due to other effects)
    - Methods: drop packets, choke packets, hop-by-hop choke packets, fair queuing, ...
  - **Avoid**
    - before congestion happens
    - initiate countermeasures at the sender
    - initiate countermeasures at the receiver
    - Methods: leaky bucket, token bucket, isarithmic congestion control, reservation, ...
Repair

Principle
- No resource reservation
- Necessary steps
  - Congestion detected
  - Introduce appropriate procedures for reduction
Repair by Packet dropping

- **Principle**
  - At each intermediate system
  - Queue length is tested
  - Incoming packet is dropped if it cannot be buffered
    - We may not wait until the queue is entirely full

- **To provide**
  - Unreliable service
    - No preparations necessary
  - Reliable service
    - Buffer packet until reception has been acknowledged
Repair by Packet dropping

Assigning buffers to queues at output lines

1. Fair distribution of buffers per output line
   - Packet may be dropped although there are free buffers
2. Minimal number (usually 1) of buffers per output line
   - Sequences to same output line ("bursts") lead to drops
3. Dynamic buffer assignment
   - React badly to load shifting loads
Repair by Packet dropping

4. Content-related dropping: relevance

- Relevance of data connection as a whole or every packet from one end system to another end system
  - Examples
    - Favor IPv6 packets with flow id 0x4b5 over all others
    - Favor packets of TCP connection (65.246.255.51,80,129.240.69.49,53051) over all others

- Relevance of a traffic class
  - Examples
    - Favor ICMP packets over IP packets
    - Favor HTTP traffic (all TCP packets with source port 80) over FTP traffic
    - Favor packets from 65.246.0.0/16 over all others
Repair by Packet dropping

- Properties of packet dropping
  - Very simple

- But
  - Retransmitted packets waste bandwidth
  - Packet has to be sent $\frac{1}{1 - p}$ times before it is accepted
    - ($p$ ... probability that packet will be dropped)

- Optimization necessary to reduce the waste of bandwidth
  - Dropping packets that have not gotten that far yet
    - e.g. Choke packets
Repair by Choke Packets

- **Principle**
  - Reduce traffic during congestion by telling source to slow down

- **Procedure for router**
  - Each outgoing line has one variable
    - Utilization $u$ ( $0 \leq u \leq 1$ )
      - Calculating $u$: Router checks the line usage $f$ periodically ($f$ is 0 or 1)
        - $u = a \times u + (1 - a) \times f$
        - $0 \leq a \leq 1$ determines to what extent "history" is taken into account
  - $u >$ threshold: line changes to condition "warning"
    - Send choke packet to source (indicating destination)
    - Tag packet (to avoid further choke packets from down stream router) & forward it
Repair by Choke Packets

- **Principle**
  - Reduce traffic during congestion by telling source to slow down

- **Procedure for source**
  - Source receives the choke packet
    - Reduces the data traffic to the destination in question by $x_1\%$
  - Source recognizes 2 phases
    (gate time so that the algorithm can take effect)
    - Ignore: source ignores further Choke packets until timeout
    - Listen: source listens if more Choke packets are arriving
      - yes: further reduction by $x_2\%$; go to Ignore phase
      - no: increase the data traffic
Repair by Choke Packets

- Hop-by-Hop Choke Packets
- Principle
  - Reaction to Choke packets already at router (not only at end system)

Plain Choke packets

A heavy flow is established
Congestion is noticed at D
A Choke packet is sent to A
The flow is reduced at A
The flow is reduced at D

Hop-by-hop Choke packets

A heavy flow is established
Congestion is noticed at D
A Choke packet is sent to A
The flow is reduced at F
The flow is reduced at D
Repair by Choke Packets

- **End-to-end variation**
  - $u > \text{threshold}$: line changes to condition "warning"
    - Procedure for router
      - do not send choke packet to source (indicating destination)
      - tag packet (to avoid further choke packets from downstream router) & forward it
    - Procedure at receiver
      - send choke packet to sender

- **Other variations**
  - Varying choke packets depending on state of congestion
    - Warning
    - acute warning
  - For $u$ instead of utilization
    - queue length
    - ....
Repair by Choke Packets

- Properties
  - Effective procedure
  - But:
    - possibly many choke packets in the network
      - even if Choke bits may be included in the data at the senders to minimize reflux
    - end systems can (but do not have to) adjust the traffic
    - choke packets take time to reach source
      - transient congestion may have passed when the source reacts
    - oscillations
      - several end systems reduce speed because of choke packets
      - seeing no more choke packets, all increase speed again
Repair with Fair Queuing

- **Background**
  - End-system adapting to traffic (e.g. by Choke-Packet algorithm) should not be disadvantaged

- **Principle**
  - On each outgoing line each end-system receives its own queue
  - Packet sending based on Round-Robin (always one packet of each queue (sender))

- **Enhancement "Fair Queuing"**
  - Adapt Round-Robin
  - But weighting is not taken into account

- **Enhancement "Weighted Fair Queuing"**
  - Favoring (statistically) certain traffic
  - Criteria variants
    - In relation to VPs (virtual paths)
    - Service specific (individual quality of service)
    - etc.

Find one of the control papers that are based on a matrix that logs traffic info for all flows, GREEN, BLUE ...
Transport Layer

Congestion Avoidance
Avoidance

- **Principle**
  - Appropriate communication system behavior and design

- **Policies at various layers can affect congestion**
  - **Data link layer**
    - Flow control
    - Acknowledgements
    - Error treatment / retransmission / FEC
  - **Network layer**
    - Datagram (more complex) vs. virtual circuit (more procedures available)
    - Packet queueing and scheduling in router
    - Packet dropping in router (including packet lifetime)
    - Selected route
  - **Transport layer**
    - Basically the same as for the data link layer
    - But some issues are harder (determining timeout interval)
Avoidance by Traffic Shaping

- **Motivation**
  - Congestion is often caused by bursts
  - Bursts are relieved by smoothing the traffic (at the price of a delay)

```
Original packet arrival

Peak rate

Smoothed stream
```

- **Procedure**
  - Negotiate the traffic contract beforehand (e.g., flow specification)
  - The traffic is shaped by sender
    - Average rate
    - Burstiness
  - Applied
    - In ATM
    - In the Internet (“DiffServ” – Differentiated Services)
Token Bucket

- **Principle**
  - Permit a certain amount of data to flow off for a certain amount of time
  - Controlled by "tokens"
  - Number of tokens limited
  - Number of queued packets limited

- **Implementation**
  - Add tokens periodically
    - Until maximum has been reached
  - Remove token
    - Depending on the length of the packet (byte counter)

- **Comparison**
  - Leaky Bucket
    - Max. constant rate (at any point in time)
  - Token Bucket
    - Permits a limited burst
Token Bucket

- Principle
  - Permit a certain amount of data to flow off for a certain amount of time
  - Controlled by "tokens"
  - Number of tokens limited
  - Number of queued packets limited

- Implementation
  - Add tokens periodically
    - Until maximum has been reached
  - Remove token
    - Depending on the length of the packet (byte counter)

Leaky Bucket
- Max. constant rate (at any point in time)

Token Bucket
- Permits a limited burst packet burst
Token Bucket

- **Principle**
  - Permit a certain amount of data to flow off for a certain amount of time
  - Controlled by "tokens"
  - Number of tokens limited
  - Number of queued packets limited

- **Implementation**
  - Add tokens periodically
    - Until maximum has been reached
  - Remove token
    - Depending on the length of the packet (byte counter)
Avoidance by Reservation: Admission Control

- **Principle**
  - Prerequisite: virtual circuits
  - Reserving the necessary resources (incl. buffers) during connect
  - If buffer or other resources not available
    - Alternative path
    - Desired connection refused

- **Example**
  - Network layer may adjust routing based on congestion
  - When the actual connect occurs
Avoidance by Reservation: Admission Control

- Sender oriented
  - Sender (initiates reservation)
    - Must know target addresses (participants)
    - Not scalable
    - Good security
Avoidance by Reservation: Admission Control

- **Receiver oriented**
  - Receive (initiates reservation)
    - Needs advertisement before reservation
    - Must know "flow" addresses
  - **Sender**
    - Need not to know receivers
    - More scalable
    - Insecure

![Diagram](image-url)
Avoidance by Reservation: Admission Control

- Combination?

- Start sender oriented reservation
Avoidance by Buffer Reservation

- **Principle**
  - Buffer reservation

- **Implementation variant: Stop-and-Wait protocol**
  - one buffer per router and connection (simplex, VC=virtual circuit)

- **Implementation variant: Sliding Window protocol**
  - $m$ buffers per router and (simplex-) connection

- **Properties**
  - congestion not possible
  - buffers remain reserved
    - even if there is no data transmission for some periods
  - usually only with applications that require low delay & high bandwidth
Avoidance: combined approaches

- **Controlled load**
  - Traffic in the controlled load class experiences the network as empty
  - Traffic class for the IntServ/RSVP reservation mechanism

- **Approach**
  - Allocate *few* buffers for this class on each router
  - Use admission control for these few buffers
    - Reservation is in packets/second (or Token Bucket specification)
    - Router knows its transmission speed
    - Router knows the number of packets it can store
  - Strictly prioritize traffic in a controlled load class

- **Effect**
  - Controlled load traffic is hardly ever dropped
  - Overtakes other traffic
Avoidance: combined approaches

- **Expedited forwarding**
  - Very similar to controlled load
  - A differentiated services PHB (per-hop-behavior) for the DiffServ mechanisms

- **Approach**
  - Set aside *few* buffers for this class on each router
  - Police the traffic
    - Shape or mark the traffic
    - Only at senders, or at some routers
  - Strictly prioritize traffic in a controlled load class

**Effect**
Shapers drop excessive traffic
EF traffic is hardly ever dropped
Overtakes other traffic

<table>
<thead>
<tr>
<th>Version</th>
<th>IHL</th>
<th>DS</th>
<th>Total length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identification</td>
<td>DM</td>
<td>Fragment offset</td>
<td></td>
</tr>
<tr>
<td>Time to live</td>
<td>Protocol</td>
<td>Header checksum</td>
<td></td>
</tr>
<tr>
<td>Source address</td>
<td>Destination Address</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 0 1 1 1 0 0 0
Transport Layer

Congestion Avoidance: TCP
TCP Congestion Control

- TCP limit sending rate as a function of perceived network congestion
  - little traffic – increase sending rate
  - much traffic – reduce sending rate

- Congestion algorithm has three major “components”:
  - additive-increase, multiplicative-decrease (AIMD)
  - slow-start
  - reaction to timeout events
TCP Congestion Control

- Testing for available bandwidth
  - Ideally: send as fast as possible ($cwnd$ as large as possible) without loss
  - Increase congestion window until you have loss
  - If loss, reduce congestion window, try increasing again

- Two phases
  - Slow start
  - Congestion avoidance

- Important variables
  - $cwnd$ (congestion window)
  - $ssthresh$ – defines the threshold where the transition from slow start to congestion avoidance is made
TCP Congestion Control

- End-to-end control (no support from the network layer)
- Send rate is limited by the size of a congestion window, $cwnd$, that is measured in bytes

- $w$ segments, each of size MSS, can be sent in each RTT:

$$\text{throughput} = \frac{w \times MSS}{RTT} \text{ bytes/sec}$$
TCP Congestion Control

Initially, $cwnd$ is 1 MSS (message segment size)

Then, the size increases by 1 for each received ACK (until threshold $ssthresh$ is reached or an ACK is missing)
TCP Congestion Control

Normally, the threshold is 65 K

Losing a packet (TCP Tahoe):
☑ ssthresh drops to half cwnd
☑ cwnd back to 1

Losing a single packet (TCP Reno):
☑ ssthresh drops to halve cwnd
☑ cwnd back to new threshold
TCP Congestion Control

**Slow Start**
TCP will always return to a slow start when a packet loss is detected by timeout (instead of duplicate ACKs). That means that it starts from scratch with only one segment per RTT, then 2, then 4, etc.

**Congestion Avoidance Phase**
One more segments sent after 1 RTT without loss in congestion avoidance phase

**Additive Increase**
Performed when loss is detected in slow-phase and in congestion avoidance phase

**Multiplicative Decrease**
Performed when loss is detected in slow-phase and in congestion avoidance phase

TCP will always return to a slow start when a packet loss is detected by timeout (instead of duplicate ACKs). That means that it starts from scratch with only one segment per RTT, then 2, then 4, etc.
TCP Fairness

- **Goal of fairness**
  - When N TCP streams share a bottleneck, each TCP stream should receive an $n^{th}$ of the bottleneck bandwidth

- **more realistic demand**
  - When N TCP streams with the same RTT and loss rate share a bottleneck, and they are infinitely long, each TCP stream receives an $n^{th}$ of the bottleneck bandwidth

- **but the approximation is in many cases good**
TCP Friendliness: The definition of good Internet behavior

A TCP connection’s throughput is bounded

- $w_{\text{max}}$ - maximum retransmission window size
- RTT - round-trip time

**Congestion windows size changes**

- AIMD algorithm
- additive increase, multiple decrease

The TCP send rate limit is

$$R_s = \frac{w_{\text{max}}}{RTT}$$

In case of **loss** in an RTT:

$$w = \beta \cdot w, \beta = \frac{1}{2}$$

In case of **no loss**:

$$w = w + \alpha, \alpha = 1$$

**TCP is said to be fair**

- Streams that share a path will reach an equal share

That’s not generally true

- Bigger RTT
  - higher loss probability per RTT
  - slower recovery
- Disadvantage for long-distance traffic
TCP Friendliness:
The definition of good Internet behavior

- A protocol is TCP-friendly if
  - **Colloquial:** It long-term average throughput is not bigger than TCP’s
  - **Formal:** Its arrival rate is at most some constant over the square root of the packet loss rate

- Thus, *if the rule is not violated* ...
  - … the AIMD algorithm with $\alpha \neq 1/2$ and $\beta \neq 1$ is still TCP-friendly
  - … TCP-friendly protocols may
    - probe for available bandwidth faster than TCP
    - adapt to bandwidth changes more slowly than TCP
    - use different equations or statistics, i.e., not AIMD
    - not use slow start (i.e., don’t start with $w=0$)
TCP Fairness

- In which way is TCP fair?
  - Two competing streams
    - Additive increase adds 1 MSS per RTT until loss occurs
    - Multiplicative decrease reduces throughput proportionally
TCP Delay Modeling

- Q: how much time does the delivery of an object from a web server take after the request has been sent?
  - Creation of the TCP connection
  - Data transfer delay

- Notation and assumptions
  - Assume a link between client and server with rate R
  - Assume constant cwnd, w segments, fixed-size segments
  - O: object size (bits)
  - No retransmissions

- To situations must be considered
  - \( W \times \text{MSS}/R > \text{RTT} + \text{MSS}/R \)
    - ACK for the first segment in the window returns before data that fills the whole window has been sent
  - \( W \times \text{MSS}/R < \text{RTT} + \text{MSS}/R \)
    - Wait for an ACK after having sent all the data that a window allows
TCP slides from Olav

\[ S = \text{MSS} \]
TCP delay modeling: Slow Start

- Assume that $cwnd$ grows as in Slow Start
- Delay of an object of size $O$ is

$$Latency = 2 \cdot RTT + \frac{O}{R} + P \cdot \left[ RTT + \frac{MSS}{R} \right] - (2^P - 1) \cdot \frac{MSS}{R}$$

  - Where $P$ is the number of times TCP stops at the sender

$$P = \min\{Q, K - 1\}$$

  - Where $Q$ is the number of times the sender will stop of $O$ is infinitely large
  - Where $K$ is the number of windows that are needed to cover the object completely
Transport Layer

Congestion Avoidance: RED and ECN
Random Early Detection (RED)

- Random Early Detection (Discard/Drop) (RED) uses active queue management

- Drops packet in an intermediate node based on average queue length exceeding a threshold
  - TCP receiver reports loss in ACK
  - sender applies MD

- Why?
  - if not, many TCPs loose packets at the same time
  - many TCP streams probe again at the same time
  - oscillating problems
Random Early Detection (RED)

- Information flow is implicit
  - just drop one packet (TCP will retransmit it and back off)
  - can be made explicit by marking packets

- Random early discard of packets
  - instead of waiting until the queue is full, drop some packets considering the probability (drop probability) when the queue length exceeds a certain level (drop level)

- RED: details
  - compute the average queue length
  - \[ \text{AvgLen} = (1-\text{Weight}) \times \text{AvgLen} + \text{Weight} \times \text{SampleLen} \]
  - \(0 < \text{Weight} < 1\) (usually 0.002)
  - SampleLen is the queue length every time a packet arrives
Two thresholds for queue length

- If \( \text{AvgLen} \leq \text{MinThreshold} \):
  - Enqueue the packet
- If \( \text{MinThreshold} < \text{AvgLen} < \text{MaxThreshold} \):
  - Compute probability \( P \)
  - Drop packet with probability \( P \)
- If \( \text{MaxThreshold} \leq \text{AvgLen} \):
  - Drop packet
Random Early Detection (RED)

- Probability $P$
  - Floating
  - Function of $\text{AvgLen}$ and how long since the previous drop
  - Variable count keeps the number of new packets that have been added to the queue (not dropped) while $\text{AvgLen}$ was between the two thresholds

\[
\begin{align*}
\text{TempP} &= \text{MaxP} \times \frac{\text{AvgLen} - \text{MinThreshold}}{\text{MaxThreshold} - \text{MinThreshold}} \\
P &= 1 - \frac{\text{count} \times \text{TempP}}{\text{TempP}}
\end{align*}
\]
Random Early Detection (RED)

- **Notes**
  - probability for dropping a packet of a given flow approx. proportional to the size of the flow
  
  - $\text{MaxP}$ will usually be 0.02. That means that when the average queue length is in the middle of the thresholds, roughly 1 in 50 packets will be dropped
  
  - when the traffic is "bursty", $\text{MinThreshold}$ should be high enough to allow an acceptable utilization of the links
  
  - the difference between the thresholds should be bigger than the typical increase in the computed queue length in one RTT for today's Internet traffic it is reasonable to set $\text{MaxThreshold} = 2 \times \text{MinThreshold}$
Early Congestion Notification (ECN) - RFC 2481

- an end-to-end congestion avoidance mechanism
- implemented in routers and supported by end-systems
- not multimedia-specific, but very TCP-specific
- two IP header bits used
  - ECT - ECN Capable Transport, set by sender
  - CE - Congestion Experienced, may be set by router

Extends RED

- if packet has ECT bit set
  - ECN node sets CE bit
  - TCP receiver sets ECN bit in ACK
  - sender applies multiple decrease (AIMD)
- else
  - Act like RED
Early Congestion Notification (ECN)

- **Effects**
  - Congestion is not oscillating - RED & ECN
  - ECN-packets are never lost on uncongested links
  - Receiving an ECN mark means
    - TCP window decrease
    - No packet loss
    - No retransmission
Transport Layer

Congestion Avoidance: TCP Vegas
TCP Vegas

- Idea
  - Source is looking for signs that a router is close to congestion
  - By checking whether
    - The RTT is growing implying that other traffic leads to a growth of the router queue
    - The distance between the received ACKs grows implying that the flow itself is too fast
TCP Vegas

Algorithm

- Let $\text{BaseRTT}$ be the minimum of all measured RTTs (usually the RTT of the first packet).
- If we don’t congest the connection, we can assume
  \[ \text{ExpectedRate} = \frac{cwnd}{\text{BaseRTT}} \]
- The source computes the actual sending rate ($\text{ActualRate}$) once per RTT.
- The source compares $\text{ActualRate}$ to $\text{ExpectedRate}$
  \[
  \text{Diff} = \text{ExpectedRate} - \text{ActualRate}
  \]
  if $\text{Diff} < \alpha$
  increase $cwnd$ linearly
  if $\text{Diff} > \beta$
  decrease $cwnd$ linearly
  Else
  leave $cwnd$ unchanged
TCP Vegas simulation results: from http://www.cs.arizona.edu/projects/protocols/
Summary

Congestion control
- reason for congestion
- congestion repair
- congestion avoidance
- TCP congestion control
- Variants of TCP congestion control