

Congestion Control in TCP

Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

November 28, 2016

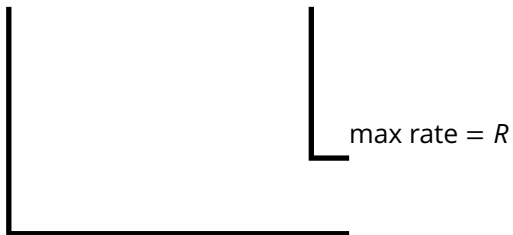
- Intro to congestion control
- Input rate vs. output throughput
- Congestion window
- “Congestion avoidance”
- “Slow start”
- “Fast recovery”

Understanding Congestion

- A router behaves a lot like a kitchen sink

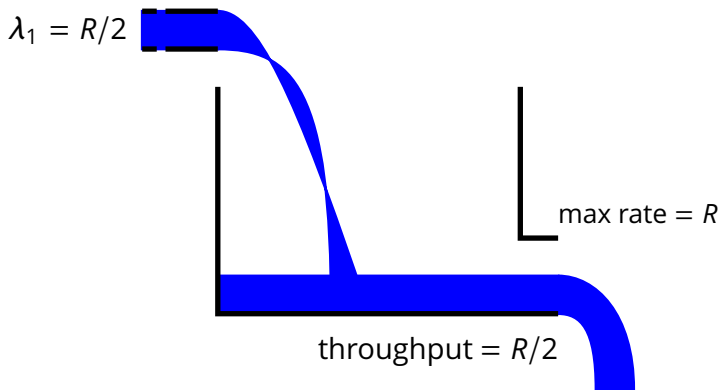
Understanding Congestion

- A router behaves a lot like a kitchen sink



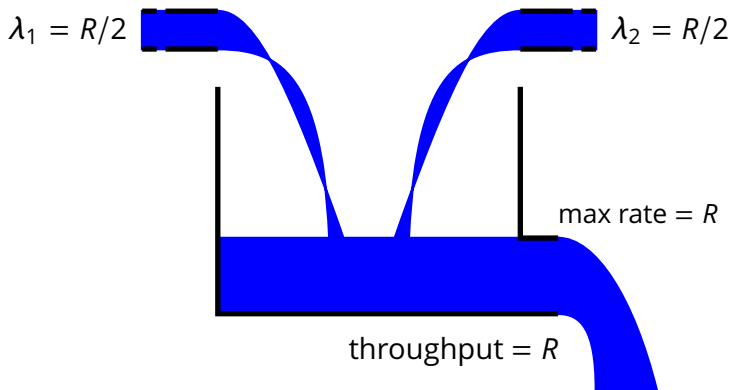
Understanding Congestion

- A router behaves a lot like a kitchen sink



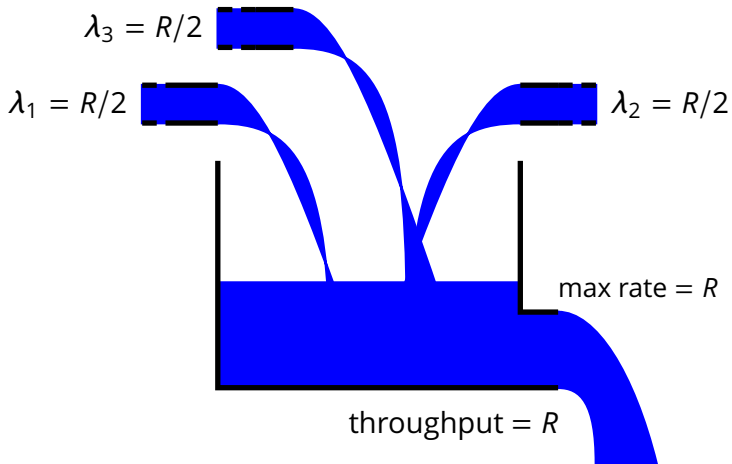
Understanding Congestion

- A router behaves a lot like a kitchen sink



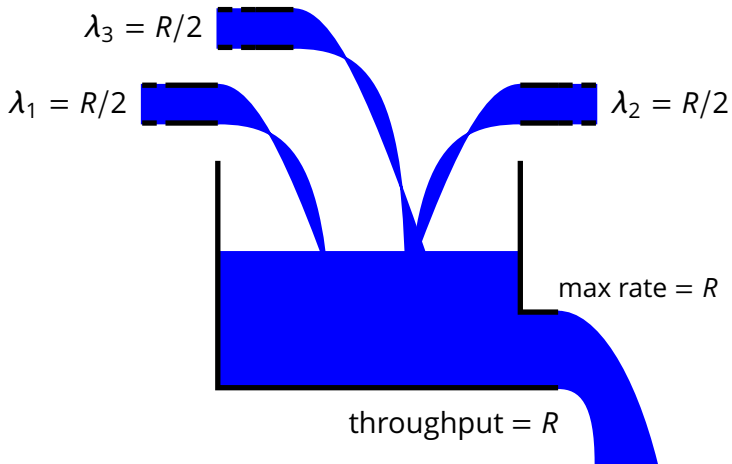
Understanding Congestion

- A router behaves a lot like a kitchen sink



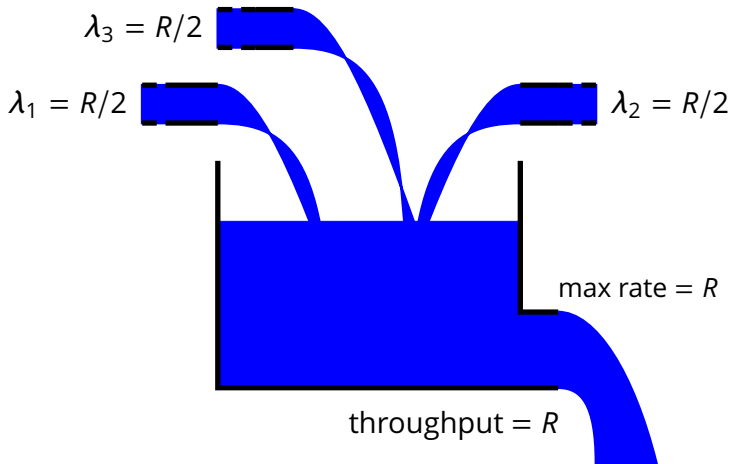
Understanding Congestion

- A router behaves a lot like a kitchen sink



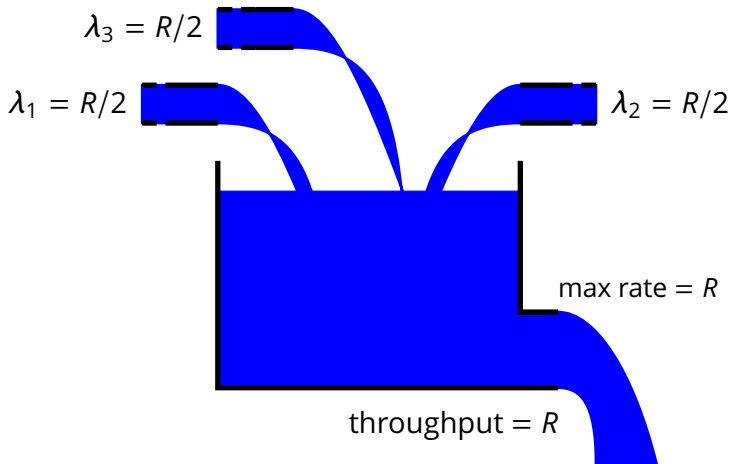
Understanding Congestion

- A router behaves a lot like a kitchen sink



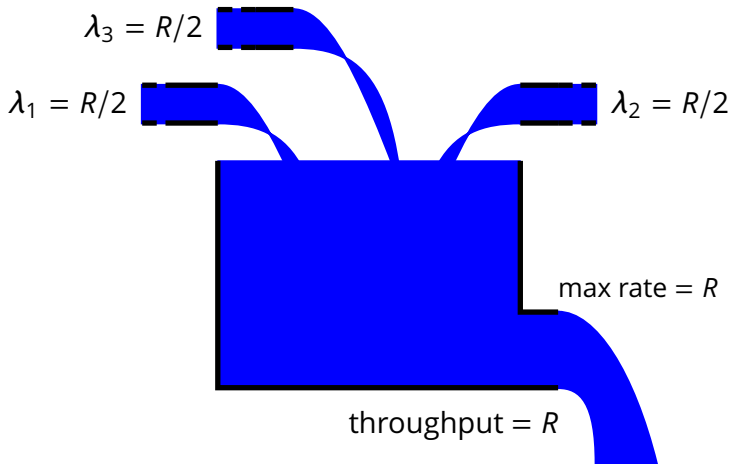
Understanding Congestion

- A router behaves a lot like a kitchen sink



Understanding Congestion

- A router behaves a lot like a kitchen sink



Queuing Delay

- Total latency is the sum of link latency, processing time, and the time that a packet spends in the input queue

$$L = d_{TX} + d_{CPU} + d_q \quad \text{where } d_q = |q|/R$$

- Total latency is the sum of link latency, processing time, and the time that a packet spends in the input queue

$$L = d_{TX} + d_{CPU} + d_q \quad \text{where } d_q = |q|/R$$

- **Ideal case:** constant input data rate

$$\lambda_{in} < R$$

In this case the $d_q = 0$, because $|q| = 0$ (ideal input flow)

- Total latency is the sum of link latency, processing time, and the time that a packet spends in the input queue

$$L = d_{TX} + d_{CPU} + d_q \quad \text{where } d_q = |q|/R$$

- **Ideal case:** constant input data rate

$$\lambda_{in} < R$$

In this case the $d_q = 0$, because $|q| = 0$ (ideal input flow)

- **Extreme case:** constant input data rate

$$\lambda_{in} > R$$

In this case $|q| = (\lambda_{in} - R)t$ and therefore

$$d_q = \frac{\lambda_{in} - R}{R}t$$

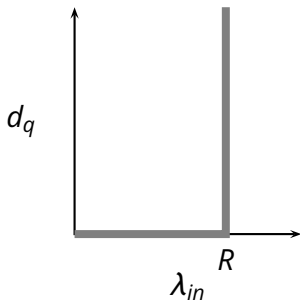
Queuing Delay

- Steady-state queuing delay

$$d_q = \begin{cases} 0 & \lambda_{in} < R \\ \frac{\lambda_{in}-R}{R}t & \lambda_{in} > R \end{cases}$$

- Steady-state queuing delay

$$d_q = \begin{cases} 0 & \lambda_{in} < R \\ \frac{\lambda_{in} - R}{R} t & \lambda_{in} > R \end{cases}$$

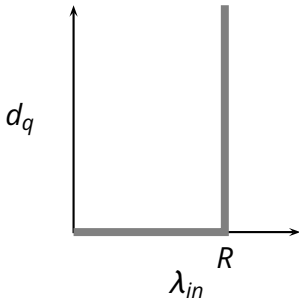


ideal input flow

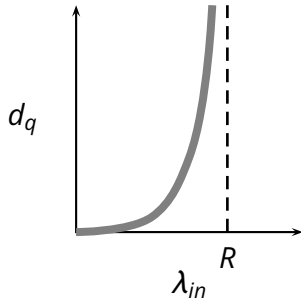
λ_{in} constant

■ Steady-state queuing delay

$$d_q = \begin{cases} 0 & \lambda_{in} < R \\ \frac{\lambda_{in}-R}{R}t & \lambda_{in} > R \end{cases}$$



ideal input flow
 λ_{in} constant



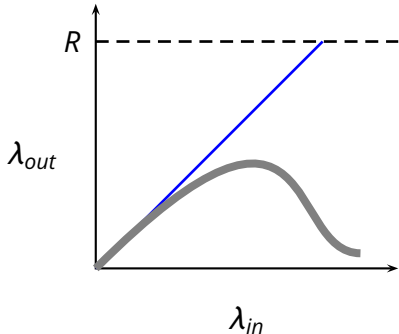
realistic input flow
 λ_{in} variable

Queuing Delay

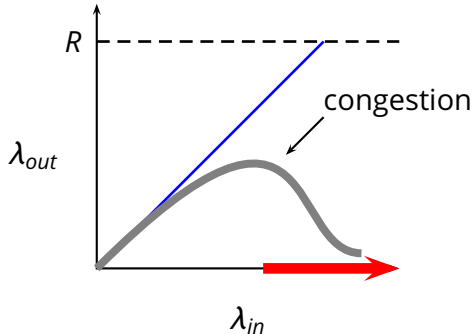
- *Conclusion:* as the input rate λ_{in} approaches the maximum throughput R , packets will experience very long delays

- *Conclusion:* as the input rate λ_{in} approaches the maximum throughput R , packets will experience very long delays
- More realistic assumptions and models
 - ▶ finite queue length (buffers) in routers
 - ▶ effects of retransmission overhead
 - ▶ full queues along multi-hops paths

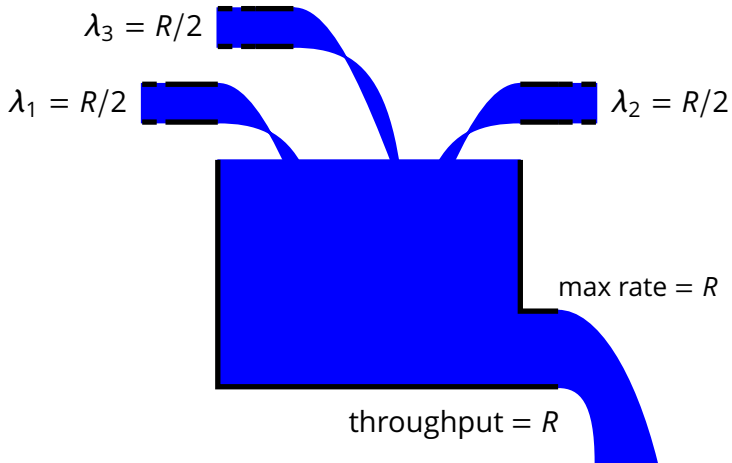
- *Conclusion:* as the input rate λ_{in} approaches the maximum throughput R , packets will experience very long delays
- More realistic assumptions and models
 - ▶ finite queue length (buffers) in routers
 - ▶ effects of retransmission overhead
 - ▶ full queues along multi-hops paths



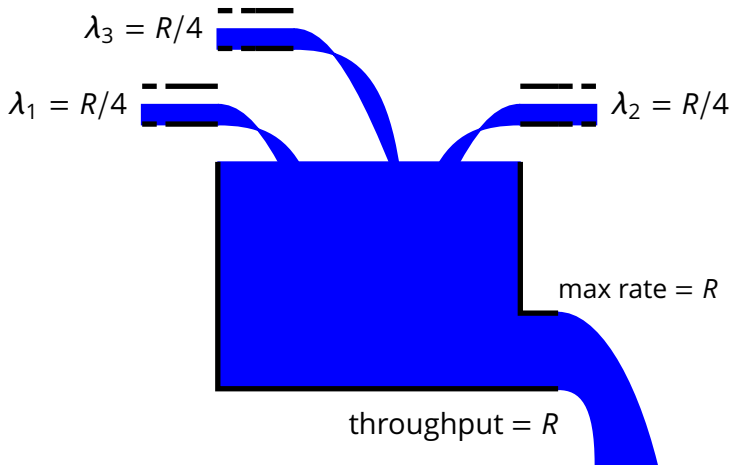
- *Conclusion:* as the input rate λ_{in} approaches the maximum throughput R , packets will experience very long delays
- More realistic assumptions and models
 - ▶ finite queue length (buffers) in routers
 - ▶ effects of retransmission overhead
 - ▶ full queues along multi-hops paths



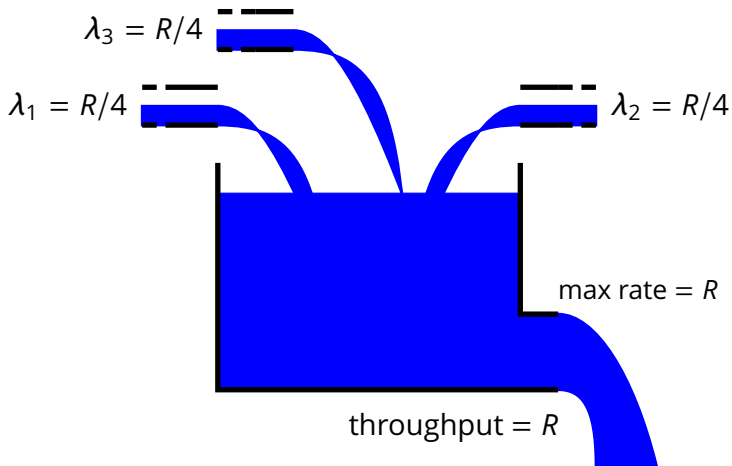
- What to do when the network is congested?



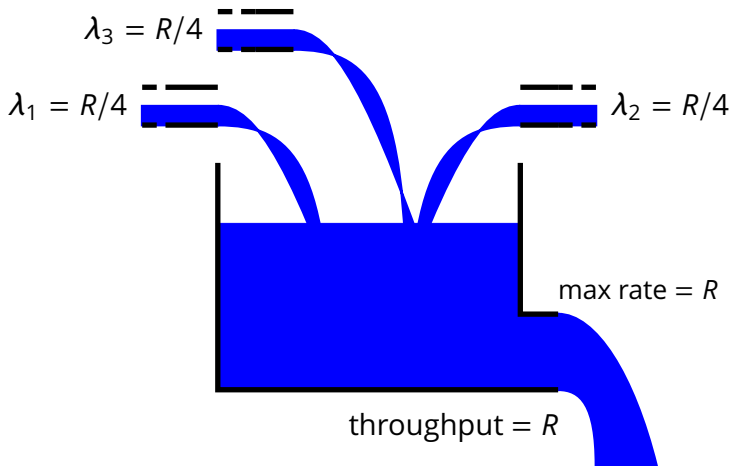
- What to do when the network is congested?



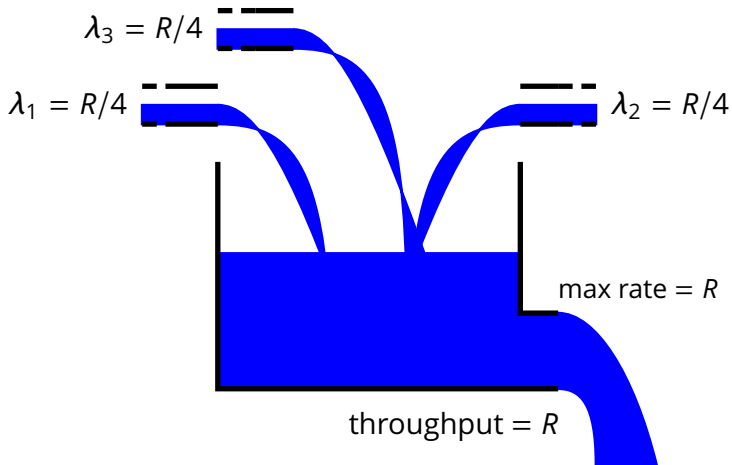
- What to do when the network is congested?



- What to do when the network is congested?



- What to do when the network is congested?



Congestion Control (in TCP)

Congestion Control (in TCP)

- Approach: ***the sender limits its output rate according to the status of the network***
 - ▶ the sender output rate becomes (part of) the input rate for the network (λ_{in})

Congestion Control (in TCP)

- Approach: ***the sender limits its output rate according to the status of the network***
 - ▶ the sender output rate becomes (part of) the input rate for the network (λ_{in})
- *Issues*

Congestion Control (in TCP)

- Approach: ***the sender limits its output rate according to the status of the network***
 - ▶ the sender output rate becomes (part of) the input rate for the network (λ_{in})

- *Issues*
 - ▶ how does the sender “measure” the status of the network?
 - ▶ i.e., how does the sender detect congestion?

Congestion Control (in TCP)

- Approach: ***the sender limits its output rate according to the status of the network***
 - ▶ the sender output rate becomes (part of) the input rate for the network (λ_{in})
- *Issues*
 - ▶ how does the sender “measure” the status of the network?
 - ▶ i.e., how does the sender detect congestion?
 - ▶ how does the sender effectively limit its output rate?

Congestion Control (in TCP)

- Approach: ***the sender limits its output rate according to the status of the network***
 - ▶ the sender output rate becomes (part of) the input rate for the network (λ_{in})

- *Issues*
 - ▶ how does the sender “measure” the status of the network?
 - ▶ i.e., how does the sender detect congestion?
 - ▶ how does the sender effectively limit its output rate?
 - ▶ how should the sender “modulate” its output rate?
 - ▶ i.e., what algorithm should the sender use to decrease or increase its output rate?

Detecting Congestion

Detecting Congestion

- If all traffic is correctly acknowledged, then the sender assumes (quite correctly) that there is no congestion

Detecting Congestion

- If all traffic is correctly acknowledged, then the sender assumes (quite correctly) that there is no congestion
- Congestion means that the queue of one or more routers between the sender and the receiver overflow
 - ▶ the visible effect is that some segments are dropped

Detecting Congestion

- If all traffic is correctly acknowledged, then the sender assumes (quite correctly) that there is no congestion
- Congestion means that the queue of one or more routers between the sender and the receiver overflow
 - ▶ the visible effect is that some segments are dropped
- Therefore the server assumes that the network is congested when it detects a segment loss
 - ▶ time out (i.e., no ACK)
 - ▶ multiple acknowledgements (i.e., NACK)

Congestion Window

- The sender maintains a *congestion window* W

Congestion Window

- The sender maintains a ***congestion window*** W
- The congestion window limits the amount of bytes that the sender pushes into the network before blocking waiting for acknowledgments

Congestion Window

- The sender maintains a ***congestion window*** W
- The congestion window limits the amount of bytes that the sender pushes into the network before blocking waiting for acknowledgments

$$LastByteSent - LastByteAked \leq W$$

where

$$W = \min (CongestionWindow, ReceiverWindow)$$

Congestion Window

- The sender maintains a **congestion window** W
- The congestion window limits the amount of bytes that the sender pushes into the network before blocking waiting for acknowledgments

$$LastByteSent - LastByteAcked \leq W$$

where

$$W = \min (CongestionWindow, ReceiverWindow)$$

- The resulting maximum output rate is roughly

$$\lambda = \frac{W}{2L}$$

Congestion Control

- How does TCP “modulate” its output rate?

Congestion Control

- How does TCP “modulate” its output rate?
- *Additive-increase and multiplicative-decrease*

Congestion Control

- How does TCP “modulate” its output rate?
- *Additive-increase and multiplicative-decrease*
- *Slow start*

Congestion Control

- How does TCP “modulate” its output rate?
- *Additive-increase and multiplicative-decrease*
- *Slow start*
- *Reaction to timeout events*

Additive-Increase/Multiplicative-Decrease

Additive-Increase/Multiplicative-Decrease

- ***How W is reduced:*** at every *loss* event, TCP halves the congestion window

Additive-Increase/Multiplicative-Decrease

- **How W is reduced:** at every *loss* event, TCP halves the congestion window
 - ▶ e.g., suppose the window size W is currently 20Kb, and a loss is detected
 - ▶ TCP reduces W to 10Kb

Additive-Increase/Multiplicative-Decrease

- **How W is reduced:** at every *loss* event, TCP halves the congestion window
 - ▶ e.g., suppose the window size W is currently 20Kb, and a loss is detected
 - ▶ TCP reduces W to 10Kb

- **How W is increased:** at every (good) acknowledgment, TCP increments W by $1MSS/W$, so as to increase W by MSS every round-trip time $2L$. This process is called **congestion avoidance**

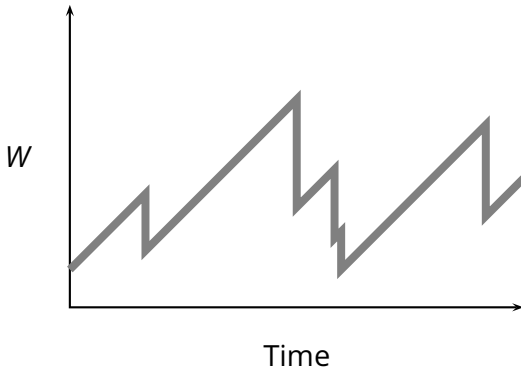
Additive-Increase/Multiplicative-Decrease

- **How W is reduced:** at every *loss* event, TCP halves the congestion window
 - ▶ e.g., suppose the window size W is currently 20Kb, and a loss is detected
 - ▶ TCP reduces W to 10Kb

- **How W is increased:** at every (good) acknowledgment, TCP increments W by $1MSS/W$, so as to increase W by MSS every round-trip time $2L$. This process is called **congestion avoidance**
 - ▶ e.g., suppose $W = 14600$ and $MSS = 1460$, then the sender increases W to 16060 after 10 acknowledgments

Additive-Increase/Multiplicative-Decrease

- Window size W over time



- What is the initial value of W ?

- What is the initial value of W ?
- The initial value of W is MSS , that is **1 segment**, which is quite low for modern networks

- What is the initial value of W ?
- The initial value of W is MSS , that is **1 segment**, which is quite low for modern networks
- To get quickly to a good throughput level, TCP increases its sending rate exponentially for its first growth phase, up to a ***slow-start threshold (ssthresh)***

- What is the initial value of W ?
- The initial value of W is MSS , that is **1 segment**, which is quite low for modern networks
- To get quickly to a good throughput level, TCP increases its sending rate exponentially for its first growth phase, up to a **slow-start threshold (*ssthresh*)**
- After the threshold, TCP proceeds with its linear push

- What is the initial value of W ?
- The initial value of W is MSS , that is **1 segment**, which is quite low for modern networks
- To get quickly to a good throughput level, TCP increases its sending rate exponentially for its first growth phase, up to a **slow-start threshold (*ssthresh*)**
- After the threshold, TCP proceeds with its linear push
- This process is called “slow start” because of the small initial value of W

Timeouts vs. NACKs

- As we know, three duplicate ACKs are interpreted as a NACK

Timeouts vs. NACKs

- As we know, three duplicate ACKs are interpreted as a NACK
- Both timeouts and NACKs signal a loss, but they say different things about the status of the network

Timeouts vs. NACKs

- As we know, three duplicate ACKs are interpreted as a NACK
- Both timeouts and NACKs signal a loss, but they say different things about the status of the network
- A *timeout indicates congestion*

Timeouts vs. NACKs

- As we know, three duplicate ACKs are interpreted as a NACK
- Both timeouts and NACKs signal a loss, but they say different things about the status of the network
- A *timeout indicates congestion*
- Three (duplicate) ACKs suggest that the network is still able to deliver segments along that path

Timeouts vs. NACKs

- As we know, three duplicate ACKs are interpreted as a NACK
- Both timeouts and NACKs signal a loss, but they say different things about the status of the network
- A ***timeout indicates congestion***
- Three (duplicate) ACKs suggest that the network is still able to deliver segments along that path
- So, TCP reacts differently to a timeout and to a triple duplicate ACKs

Timeouts vs. NACKs

Assuming the current window size is $W = \bar{W}$

Assuming the current window size is $W = \bar{W}$

■ *Timeout*

- ▶ go back to $W = MSS$
- ▶ set $ssthresh = \bar{W}/2$
- ▶ run *slow start* up to $W = ssthresh$
- ▶ then proceed with *congestion avoidance*

Assuming the current window size is $W = \bar{W}$

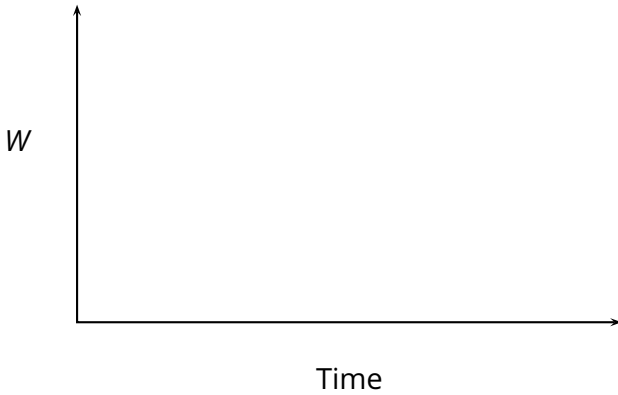
■ *Timeout*

- ▶ go back to $W = MSS$
- ▶ set $ssthresh = \bar{W}/2$
- ▶ run *slow start* up to $W = ssthresh$
- ▶ then proceed with *congestion avoidance*

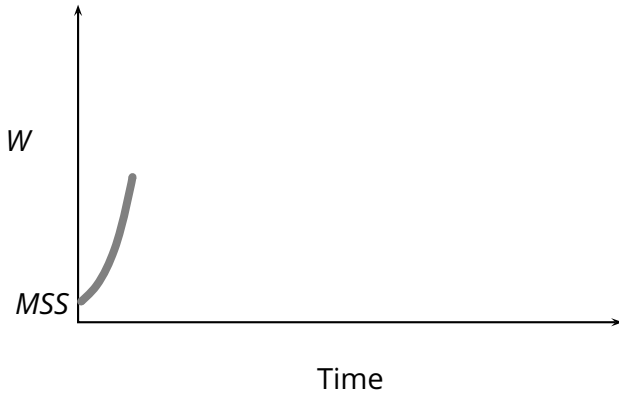
■ *NACK* (i.e., triple duplicate-ack)

- ▶ set $ssthresh = \bar{W}/2$
- ▶ cut W in half: $W = \bar{W}/2$
- ▶ run *congestion avoidance*, ramping up W linearly
- ▶ This is called ***fast recovery***

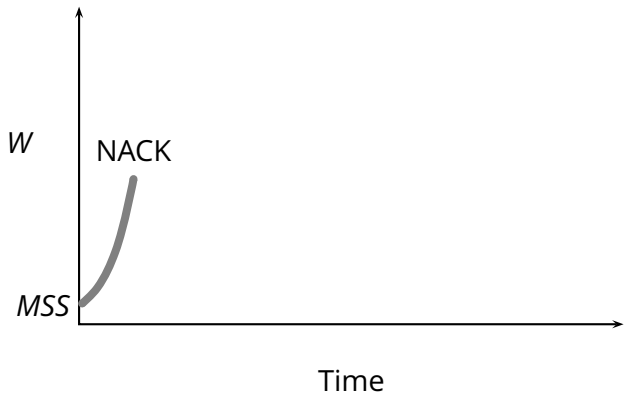
Sender Behavior



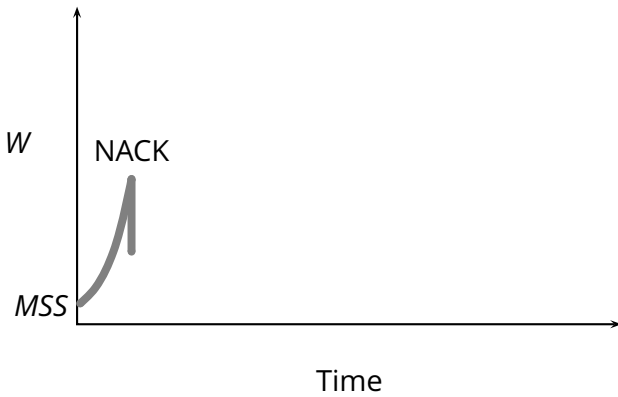
Sender Behavior



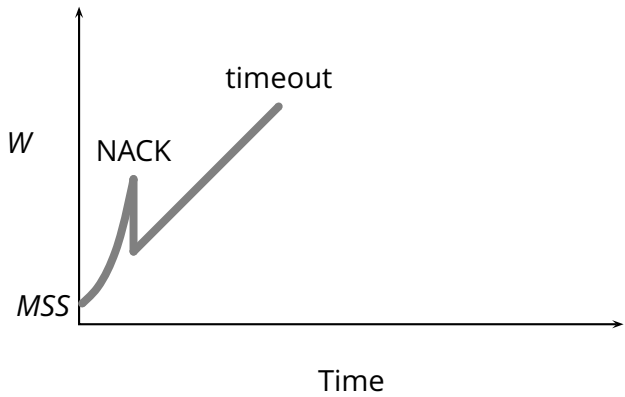
Sender Behavior



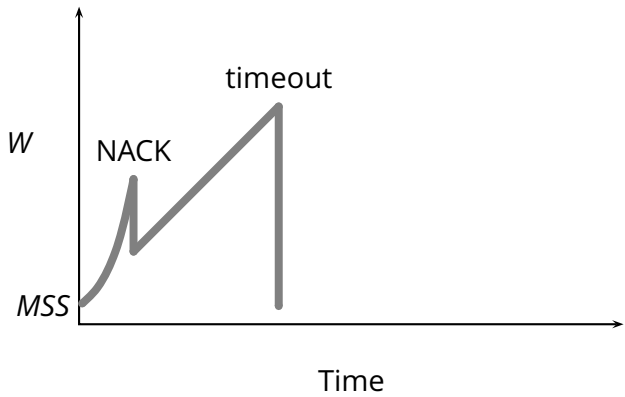
Sender Behavior



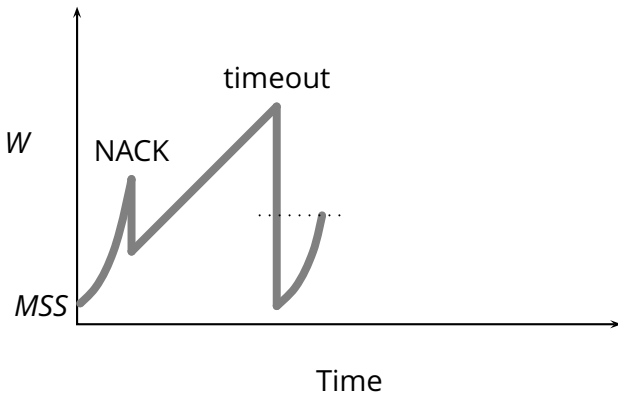
Sender Behavior



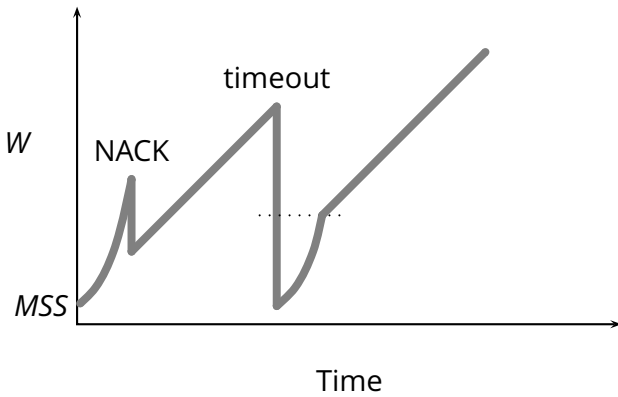
Sender Behavior



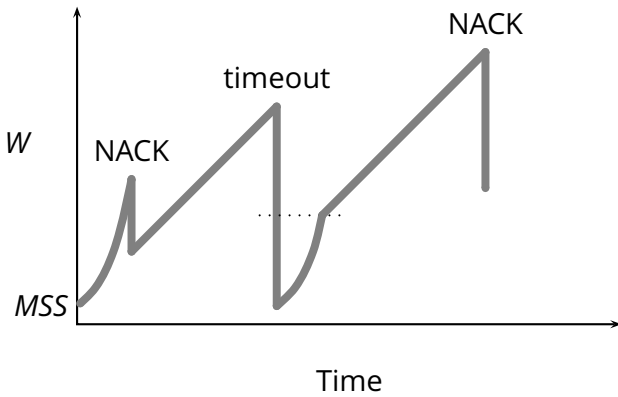
Sender Behavior



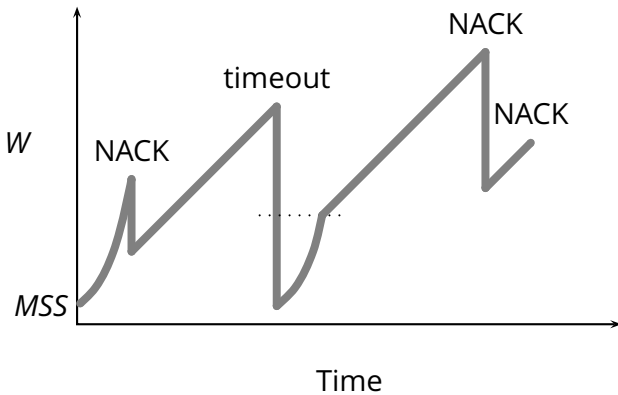
Sender Behavior



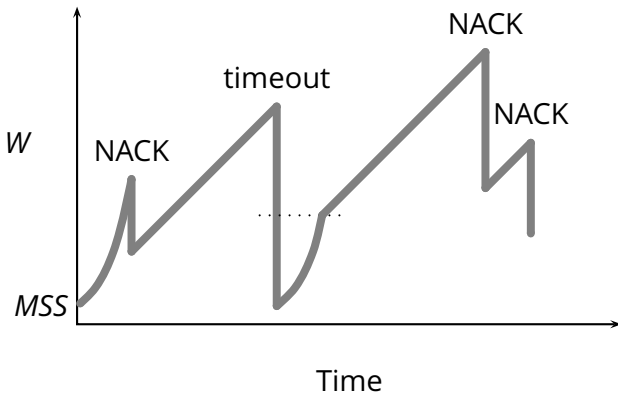
Sender Behavior



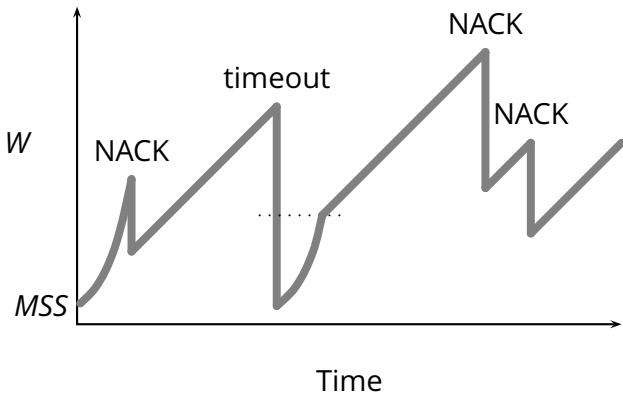
Sender Behavior



Sender Behavior



Sender Behavior



Sender Behavior

