

Congestion Control in TCP

Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

April 20, 2020

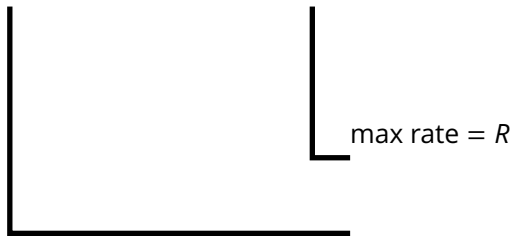
- Intro to congestion control
- Input rate vs. output throughput
- Congestion window
- “Congestion avoidance”
- “Slow start”
- “Fast recovery”

Understanding Congestion

- A router behaves a lot like a kitchen sink

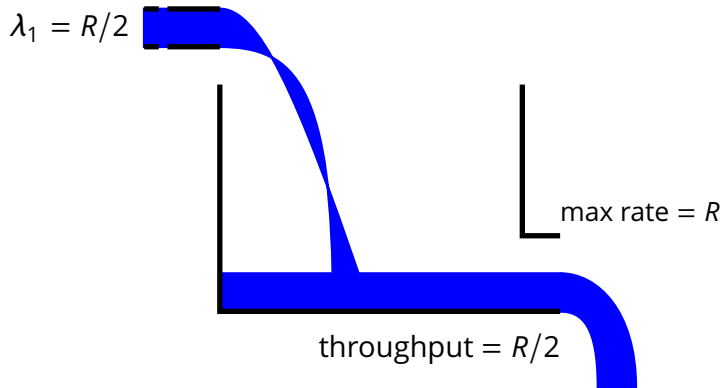
Understanding Congestion

- A router behaves a lot like a kitchen sink



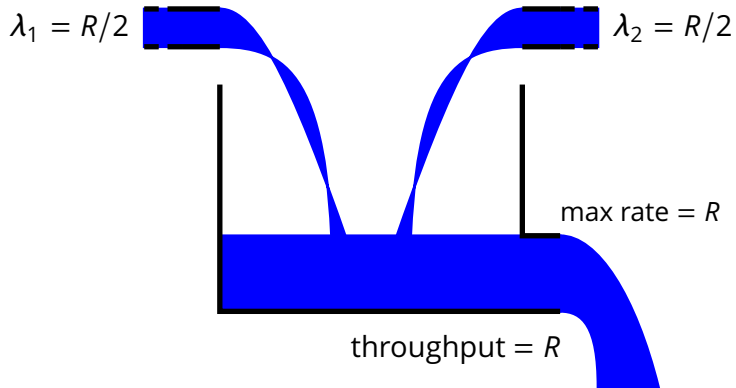
Understanding Congestion

- A router behaves a lot like a kitchen sink



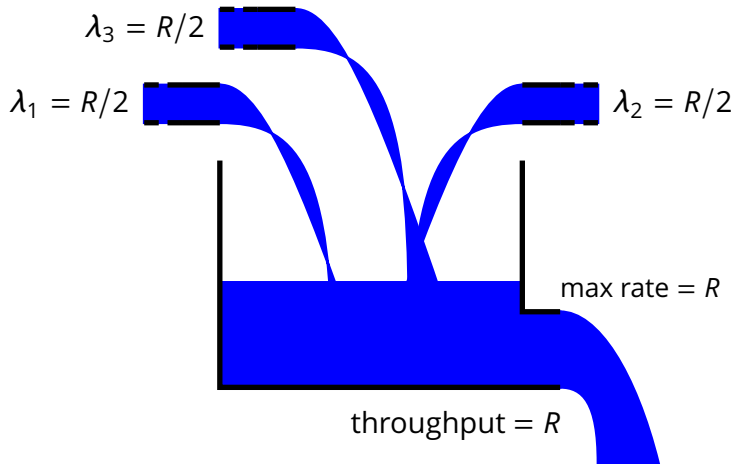
Understanding Congestion

- A router behaves a lot like a kitchen sink



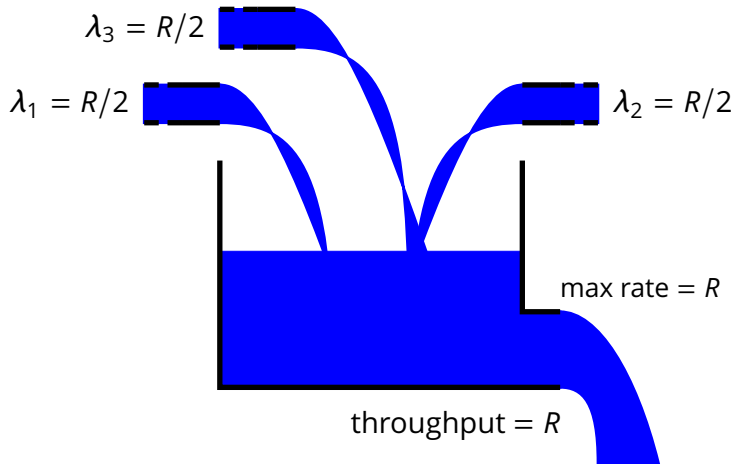
Understanding Congestion

- A router behaves a lot like a kitchen sink



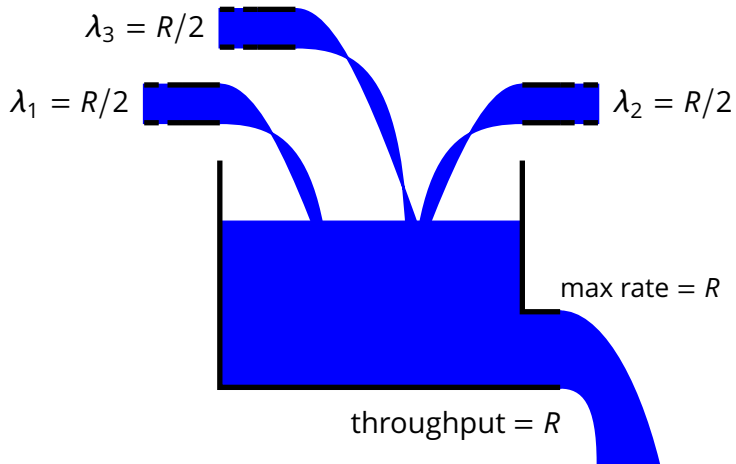
Understanding Congestion

- A router behaves a lot like a kitchen sink



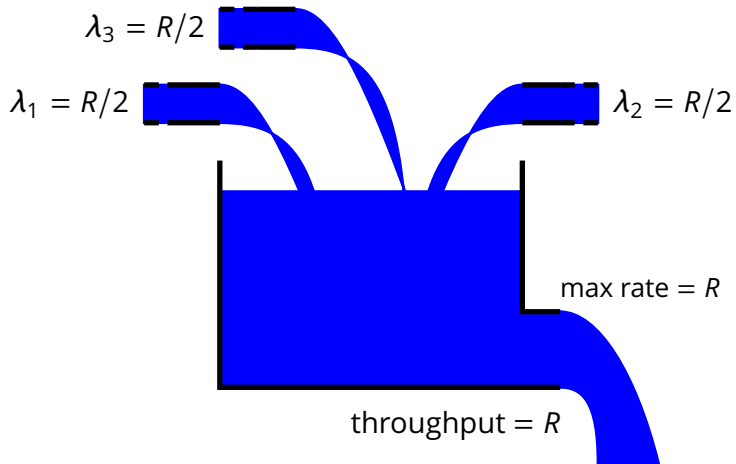
Understanding Congestion

- A router behaves a lot like a kitchen sink



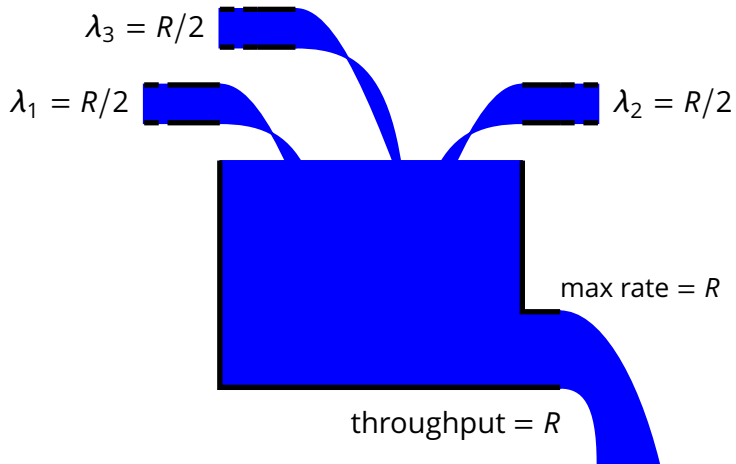
Understanding Congestion

- A router behaves a lot like a kitchen sink



Understanding Congestion

- A router behaves a lot like a kitchen sink

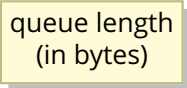


- Total latency is the sum of link latency, processing time, and the time that a packet spends in the input queue

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

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

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



queue length
(in bytes)

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

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

queue length
(in bytes)

Model: simplistic queueing model (D/D/1 queue)

- ▶ deterministic, constant inter-arrival times (MSS/λ)
- ▶ deterministic, constant service times (MSS/R)

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

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

queue length
(in bytes)

Model: simplistic queueing model (D/D/1 queue)

- ▶ deterministic, constant inter-arrival times (MSS/λ)
- ▶ deterministic, constant service times (MSS/R)

- **Stable flow:**

$$\lambda_{in} < R$$

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

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

queue length
(in bytes)

Model: simplistic queueing model (D/D/1 queue)

- ▶ deterministic, constant inter-arrival times (MSS/λ)
- ▶ deterministic, constant service times (MSS/R)

- **Stable flow:**

$$\lambda_{in} < R \quad \Rightarrow \quad |q| = 0 \quad \Rightarrow \quad d_q = 0$$

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

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

queue length
(in bytes)

Model: simplistic queueing model (D/D/1 queue)

- ▶ deterministic, constant inter-arrival times (MSS/λ)
- ▶ deterministic, constant service times (MSS/R)

- **Stable flow:**

$$\lambda_{in} < R \quad \Rightarrow \quad |q| = 0 \quad \Rightarrow \quad d_q = 0$$

- **Unstable flow:**

$$\lambda_{in} > R$$

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

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

queue length
(in bytes)

Model: simplistic queueing model (D/D/1 queue)

- ▶ deterministic, constant inter-arrival times (MSS/λ)
- ▶ deterministic, constant service times (MSS/R)

- **Stable flow:**

$$\lambda_{in} < R \quad \Rightarrow \quad |q| = 0 \quad \Rightarrow \quad d_q = 0$$

- **Unstable flow:**

$$\lambda_{in} > R \quad \Rightarrow \quad |q| = (\lambda_{in} - R)t$$

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

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

queue length
(in bytes)

Model: simplistic queueing model (D/D/1 queue)

- ▶ deterministic, constant inter-arrival times (MSS/λ)
- ▶ deterministic, constant service times (MSS/R)

- **Stable flow:**

$$\lambda_{in} < R \quad \Rightarrow \quad |q| = 0 \quad \Rightarrow \quad d_q = 0$$

- **Unstable flow:**

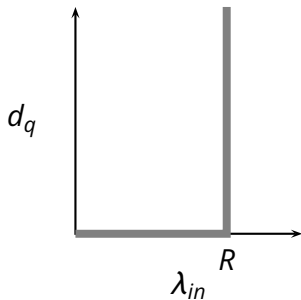
$$\lambda_{in} > R \quad \Rightarrow \quad |q| = (\lambda_{in} - R)t \quad \Rightarrow \quad d_q = \frac{\lambda_{in} - R}{R}t$$

- 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

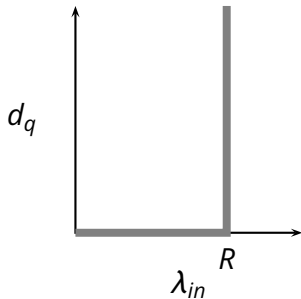
$$d_q = \begin{cases} 0 & \lambda_{in} < R \\ \frac{\lambda_{in} - R}{R} t & \lambda_{in} > R \end{cases}$$



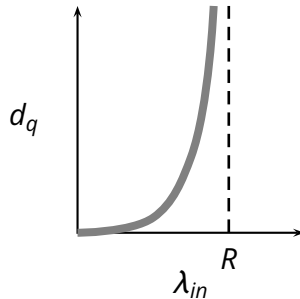
simplified model (D/D/1)

- Steady-state queuing delay

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



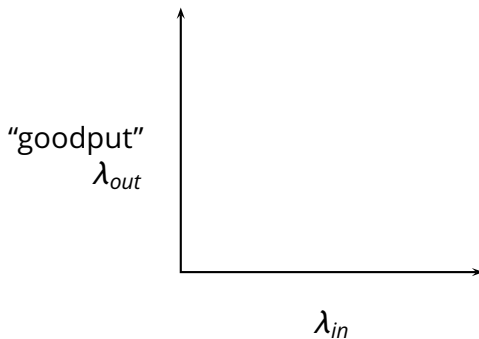
simplified model (D/D/1)



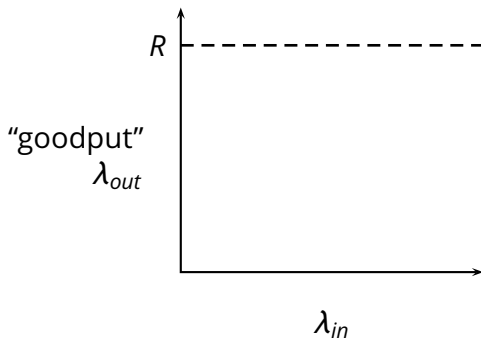
more realistic model (M/M/1)

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

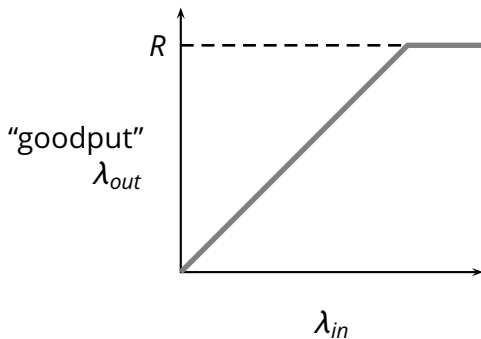
- **Conclusion:** as the input rate λ_{in} approaches the maximum throughput R , packets will experience very long delays



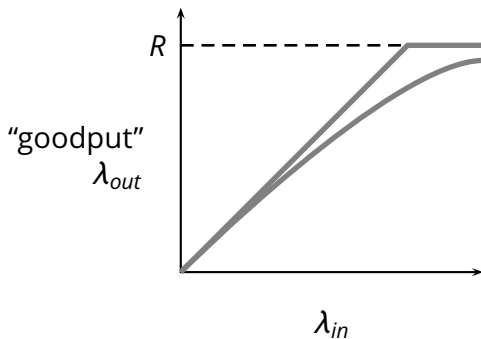
- **Conclusion:** as the input rate λ_{in} approaches the maximum throughput R , packets will experience very long delays



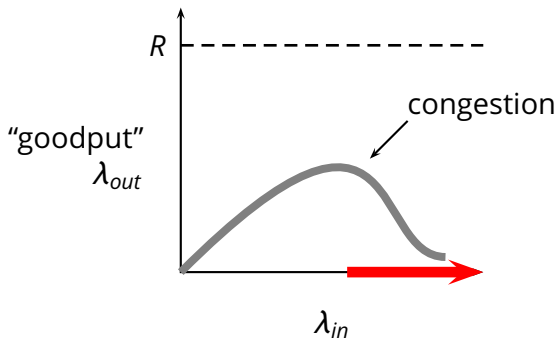
- **Conclusion:** as the input rate λ_{in} approaches the maximum throughput R , packets will experience very long delays



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

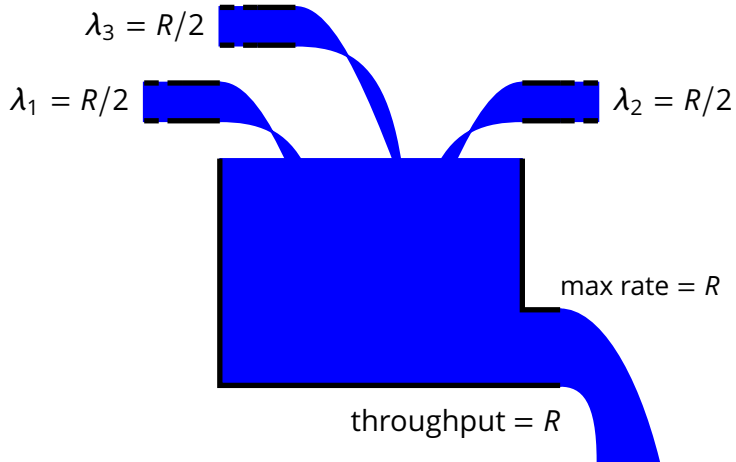


- **Conclusion:** as the input rate λ_{in} approaches the maximum throughput R , packets will experience very long delays

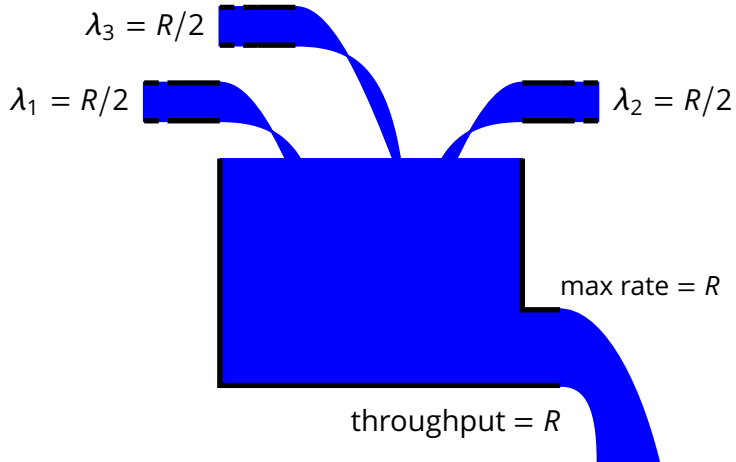


- More realistic assumptions and models
 - ▶ finite queue length (buffers) in routers
 - ▶ effects of retransmission overhead
 - ▶ full queues along multi-hops paths

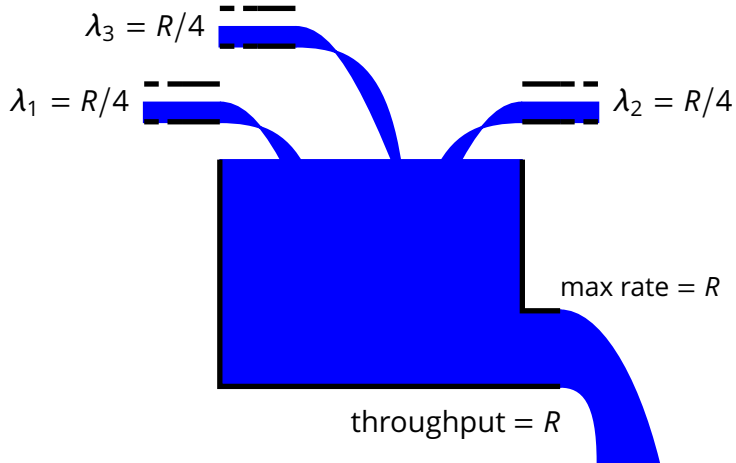
- What to do when the network is congested?



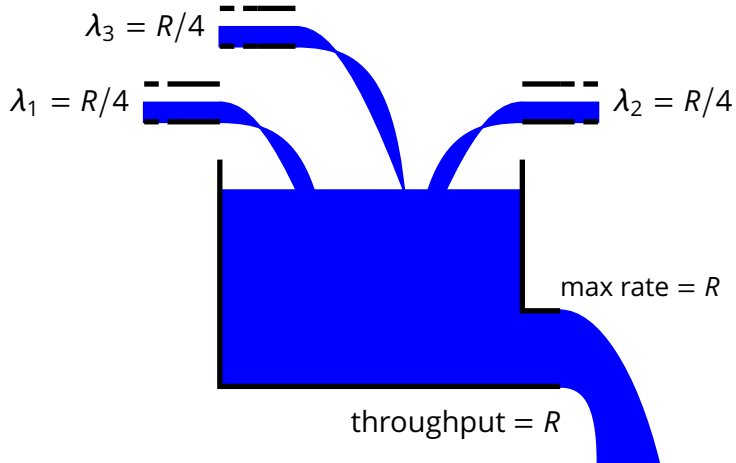
- What to do when the network is congested? **BACK OFF!**



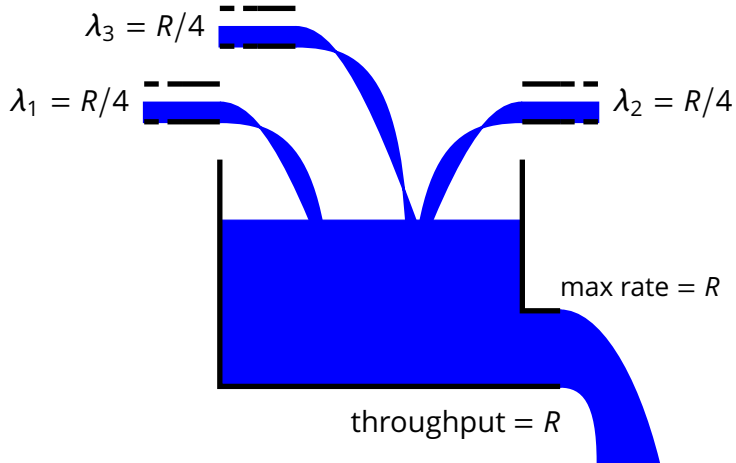
- What to do when the network is congested? **BACK OFF!**



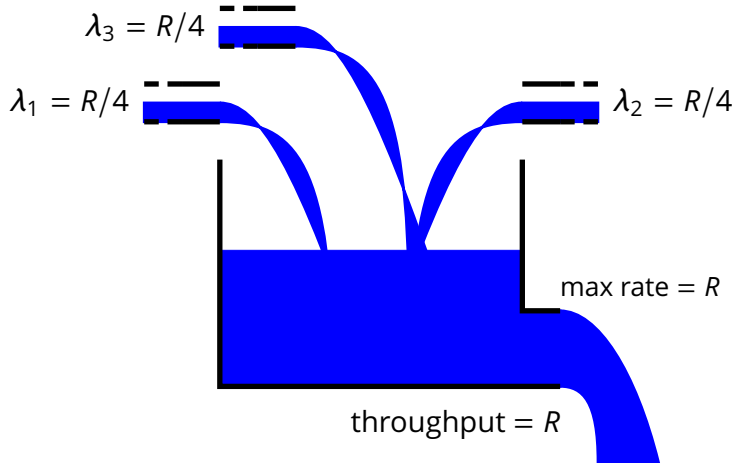
- What to do when the network is congested? **BACK OFF!**



- What to do when the network is congested? **BACK OFF!**



- What to do when the network is congested? **BACK OFF!**



Approach:

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

Approach:

- ***The sender limits its output rate according to the state of the network***

- ▶ The sender output rate becomes (part of) the input rate for the network (λ_{in})

Ingredients:

1. How does the sender ***measure the state of the network?***

- ▶ we need ***eyes*** to see the traffic ahead

Approach:

- ***The sender limits its output rate according to the state of the network***

- ▶ The sender output rate becomes (part of) the input rate for the network (λ_{in})

Ingredients:

1. How does the sender ***measure the state of the network?***

- ▶ we need ***eyes*** to see the traffic ahead

2. how does the sender ***set its output rate?***

- ▶ we need ***accelerator*** and ***brakes*** to speed up or slow down

Approach:

- ***The sender limits its output rate according to the state of the network***

- ▶ The sender output rate becomes (part of) the input rate for the network (λ_{in})

Ingredients:

1. How does the sender ***measure the state of the network?***

- ▶ we need ***eyes*** to see the traffic ahead

2. how does the sender ***set its output rate?***

- ▶ we need ***accelerator*** and ***brakes*** to speed up or slow down

3. how should the sender ***control its output rate?***

- ▶ we need a ***brain*** and we need to know ***how to drive!***

Detecting Congestion (Eyes)

Detecting Congestion (Eyes)

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

Detecting Congestion (Eyes)

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

Detecting Congestion (Eyes)

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

Congestion Window (Accelerator/Brakes)

- The sender maintains a *congestion window* W

Congestion Window (Accelerator/Brakes)

- 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 (Accelerator/Brakes)

- 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 (Accelerator/Brakes)

- 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

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

- The resulting maximum output rate is roughly

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

Congestion Control (Brain, Algorithm)

Congestion Control (Brain, Algorithm)

- *Additive-increase and multiplicative-decrease*

Congestion Control (Brain, Algorithm)

- *Additive-increase and multiplicative-decrease*
- *Slow start* (should be called *fast start*!)

Congestion Control (Brain, Algorithm)

- *Additive-increase and multiplicative-decrease*
- *Slow start* (should be called *fast 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 $RTT = 2d$. This process is called **congestion avoidance**

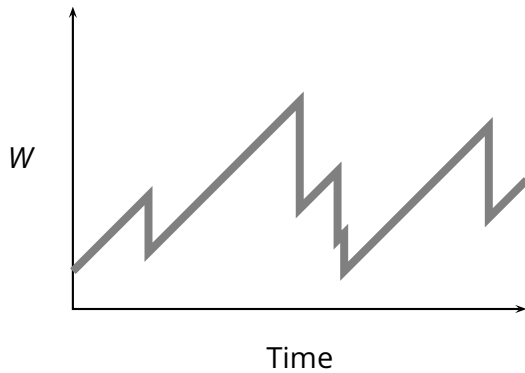
Additive-Increase/Multiplicative-Decrease

- **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 $RTT = 2d$. 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

- 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

- 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

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

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*

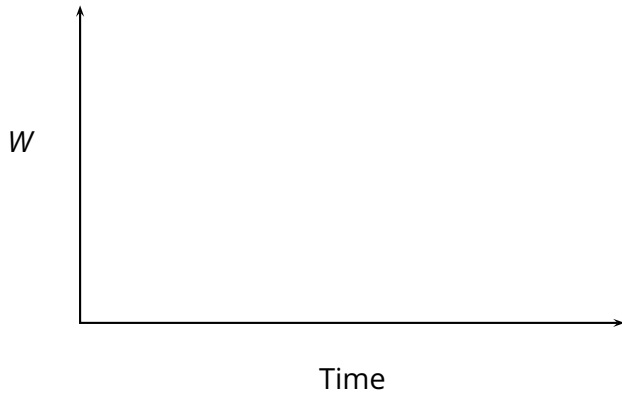
Assuming the current window size is $W = \bar{W}$

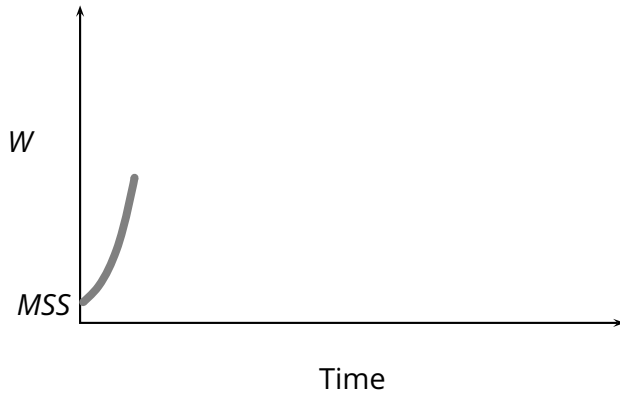
■ *Timeout*

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

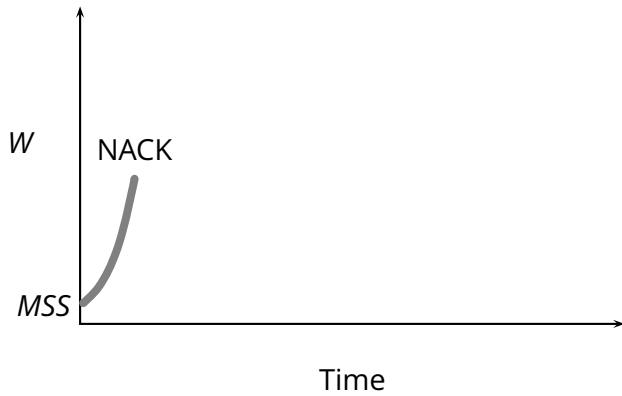
■ NACK (i.e., triple duplicate-ack)

- ▶ 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***

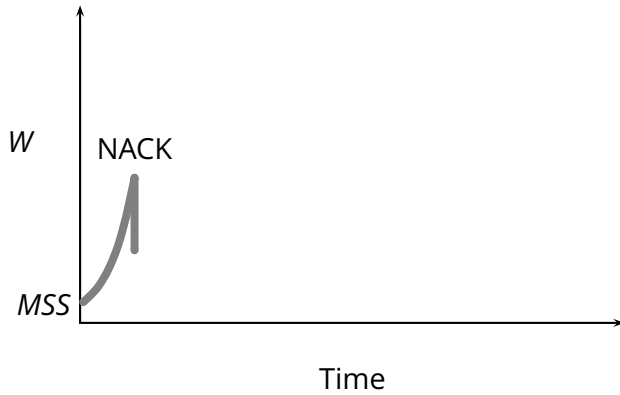


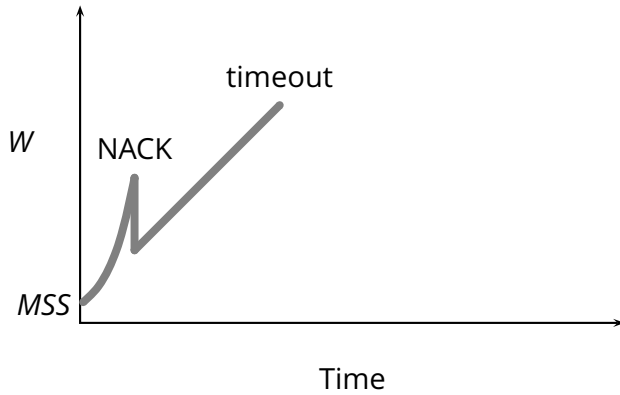


Sender Behavior

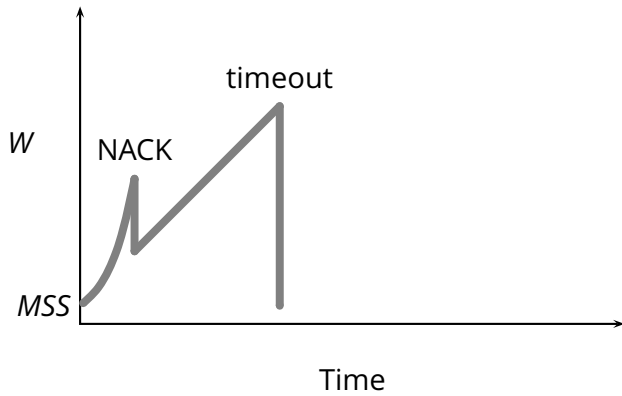


Sender Behavior

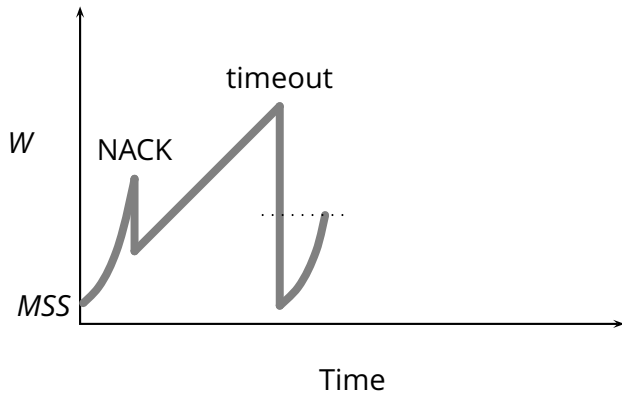




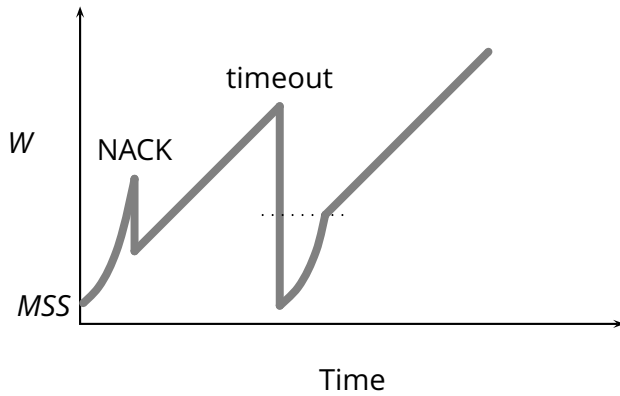
Sender Behavior



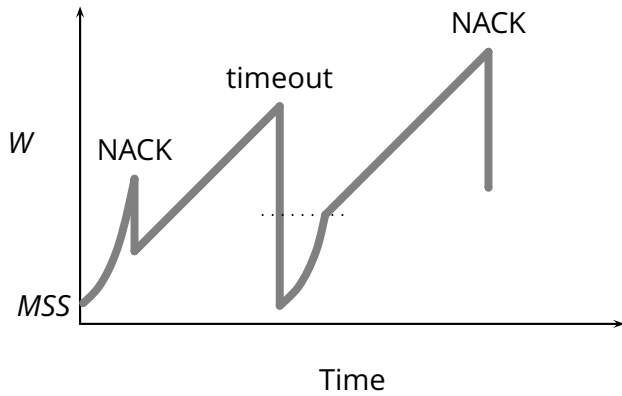
Sender Behavior



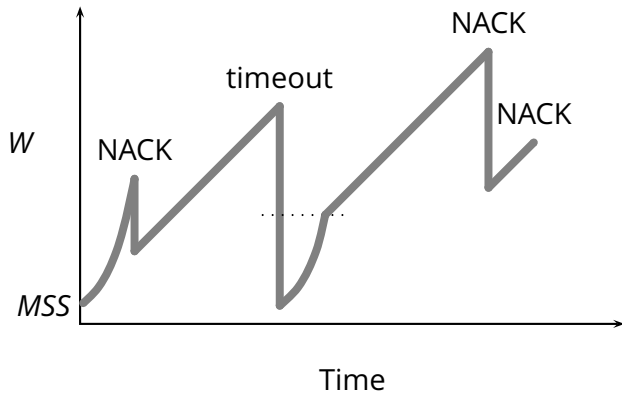
Sender Behavior



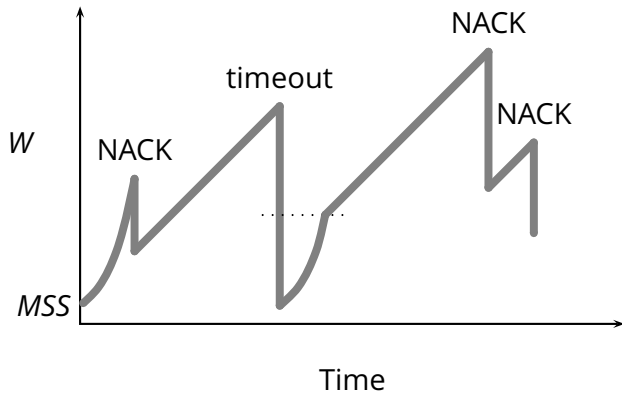
Sender Behavior



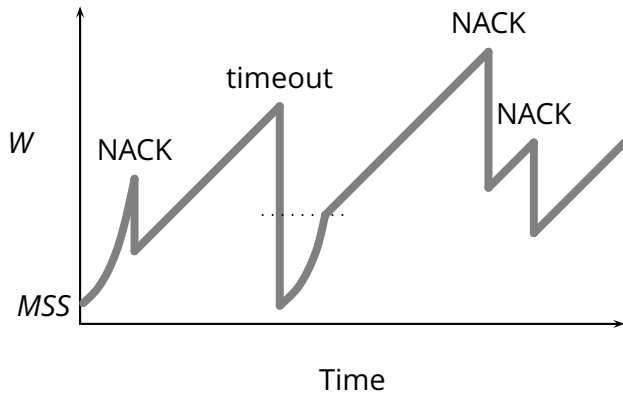
Sender Behavior



Sender Behavior



Sender Behavior



Sender Behavior

