Congestion Control (cont'd)

Recap

- Objectives of congestion control
  - Avoid congestion collapse
  - High throughput, low delay, and fairness (e.g., max-min)

Recap

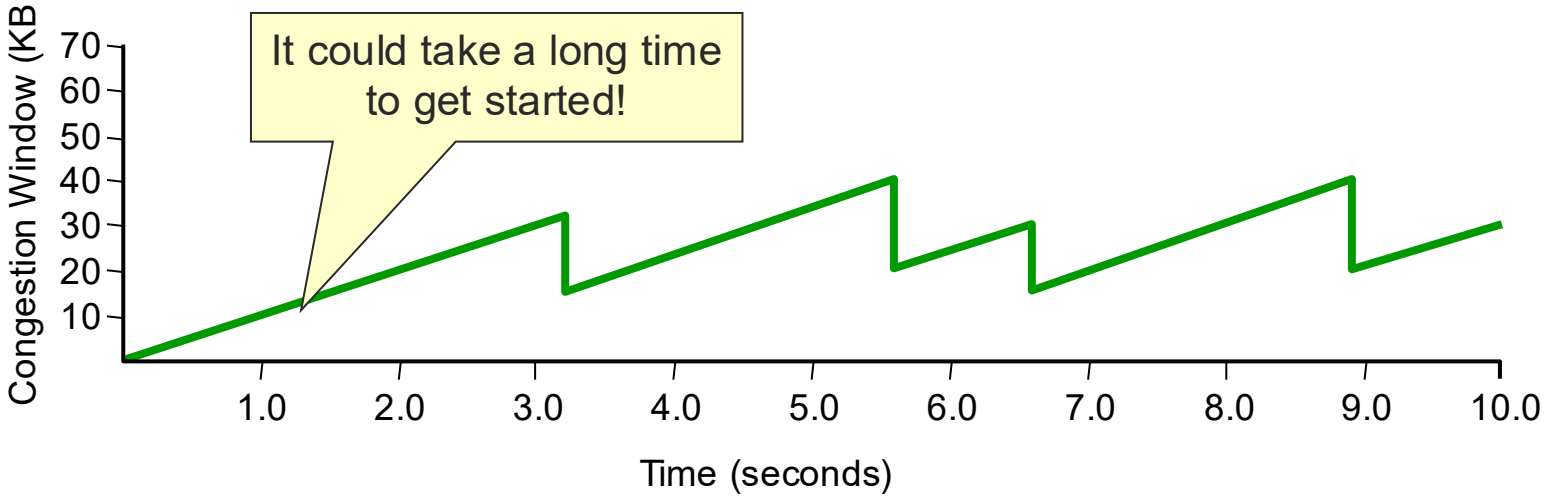
- Objectives of congestion control
  - Avoid congestion collapse
  - High throughput, low delay, and fairness (e.g., max-min)
- Router-centric: Queuing disciplines
  - Control forwarding and dropping policies
  - FIFO with tail drop, Fair Queuing
- Host-centric: TCP congestion control
  - React to congestion signals: loss/timeout in classic TCP
  - Congestion window and AIMD

Learning Objectives

- Host-centric: TCP Congestion Control (cont'd)
  - **TCP Slow Start, Congestion Avoidance**
  - Fast Retransmit, Fast Recovery
  - Beyond loss-based congestion control
- Quality of Service

TCP Start Up Behavior

- How should TCP start sending data?
  - AIMD is good for channels operating at capacity
  - AIMD can take a long time to ramp up to full capacity from scratch



TCP Start Up Behavior

- How should TCP start sending data?
  - AIMD is good for channels operating at capacity
  - AIMD can take a long time to ramp up to full capacity from scratch
  - *TCP Slow Start*
    - Begins with a small window size (start slowly)
      - Slow compared with sending a whole window immediately in original TCP
    - Increases the window exponentially (fast!)

TCP Slow Start: Initialization of Congestion Window

- Congestion window should start small
  - Avoid congestion due to new connections
- Start at 1 MSS,
  - Initially, CWND is 1 MSS
  - Initial sending rate is MSS/RTT
- Reset to 1 MSS with each timeout

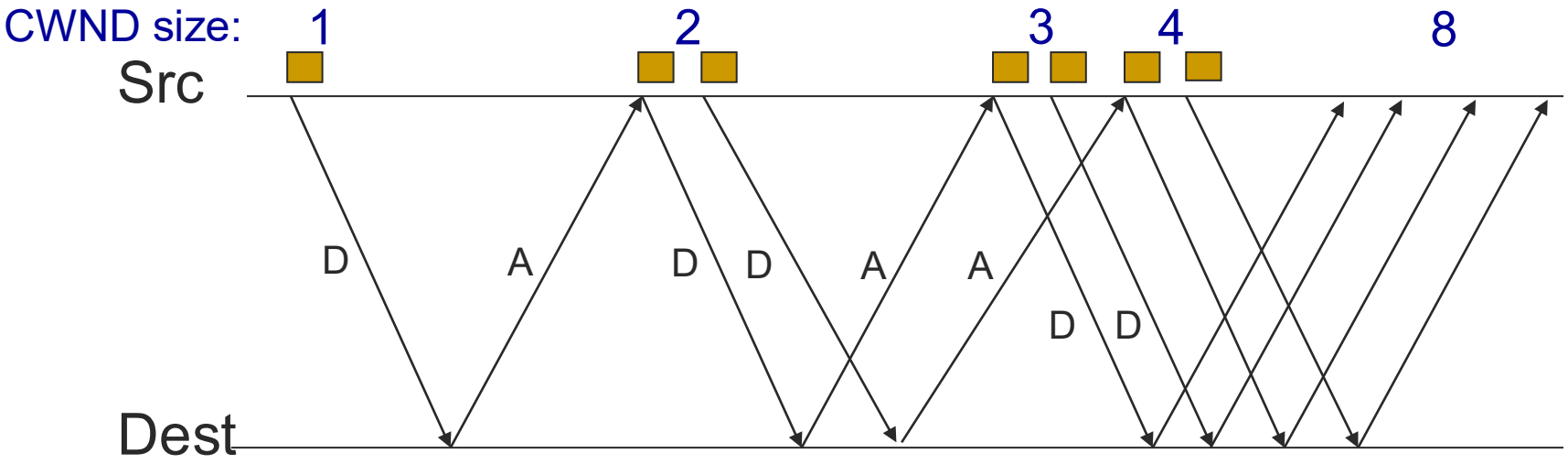
TCP Slow Start: Growth of Congestion Window

- Start slow but then grow fast
  - Sender starts at a slow rate
  - Increase the rate exponentially
  - Until the first loss event
- Linear growth could be pretty wasteful
  - Might be much less than the actual bandwidth
  - Linear increase takes a long time to accelerate

Slow Start: Summary

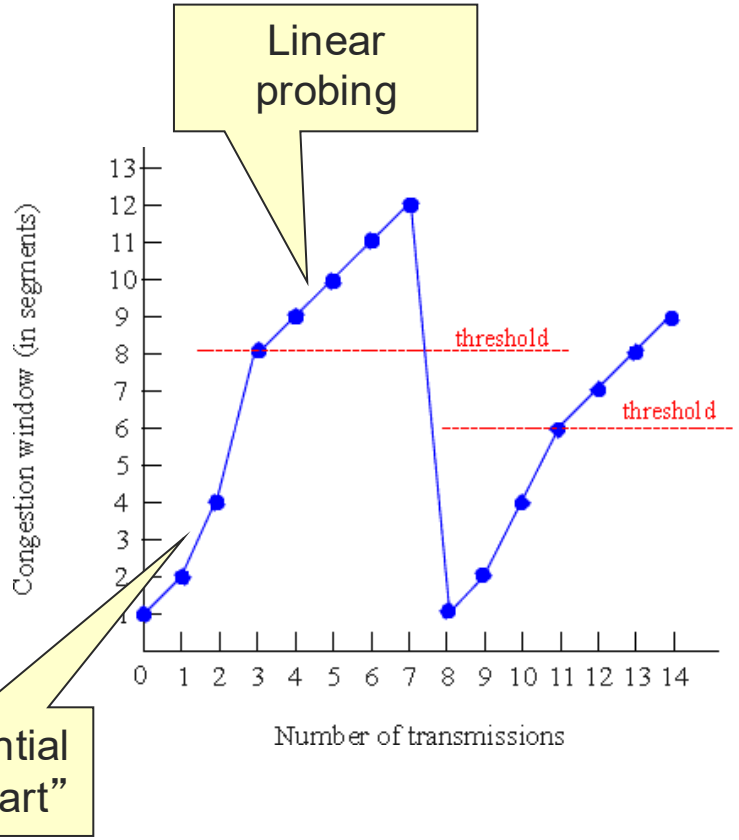
- Objective
  - Determine initial available capacity
- Implementation
  - Begin with **`CongestionWindow`** = 1 packet
  - Double **`CongestionWindow`** each RTT
    - Increment by 1 packet for each ACK
  - Continue increasing until timeout (loss)
    - Reset to 1 MSS and return to slow start

Slow Start Example



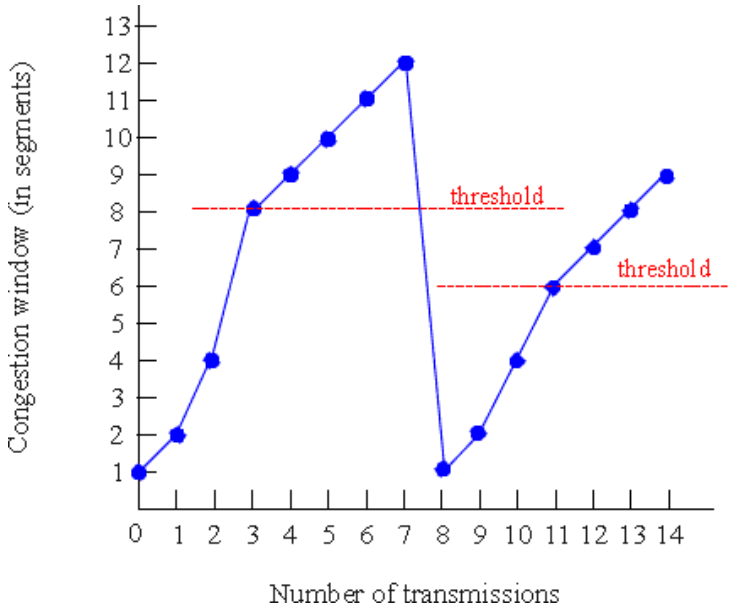
From Slow Start to Congestion Avoidance

- How long should the exponential increase from slow start continue?
  - Use **`CongestionThreshold`** as target window size
  - Estimates network capacity
  - When **`CongestionWindow`** reaches **`CongestionThreshold`** switch to additive increase
  - Set to 1/2 of current window on timeout
  - Initially set to a large value



From Slow Start to Congestion Avoidance

- Example: initially,
  - **`CongestionThreshold = 8`**
  - **`CongestionWindow = 1`**
- Slow start until transmission 3
- Additive increase until transmission 7
- Timeout after transmission 7
  - **`CongestionWindow`** currently 12
  - Set **`Congestionthreshold = CongestionWindow/2`**
  - Set **`CongestionWindow = 1`**



Learning Objectives

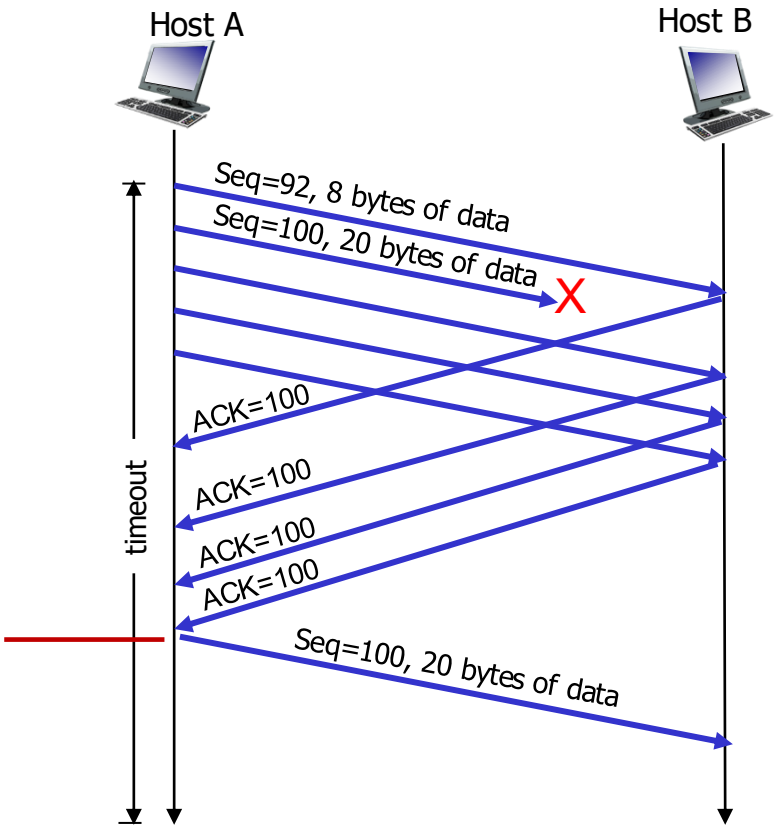
- Host-centric: TCP Congestion Control (cont'd)
  - TCP Slow Start, Congestion Avoidance
  - **Fast Retransmit, Fast Recovery**
  - Beyond loss-based congestion control
- Quality of Service

Retransmission Revisited

- Problem
  - TCP timeouts lead to long idle periods
- Fast retransmit
  - Use duplicate ACKs to trigger retransmission
- When duplicate ACKs received
  - Resend lost segment immediately
  - Do not wait for timeout
  - In practice, retransmit on 3rd duplicate (triple dup ACK)

TCP Fast Retransmit

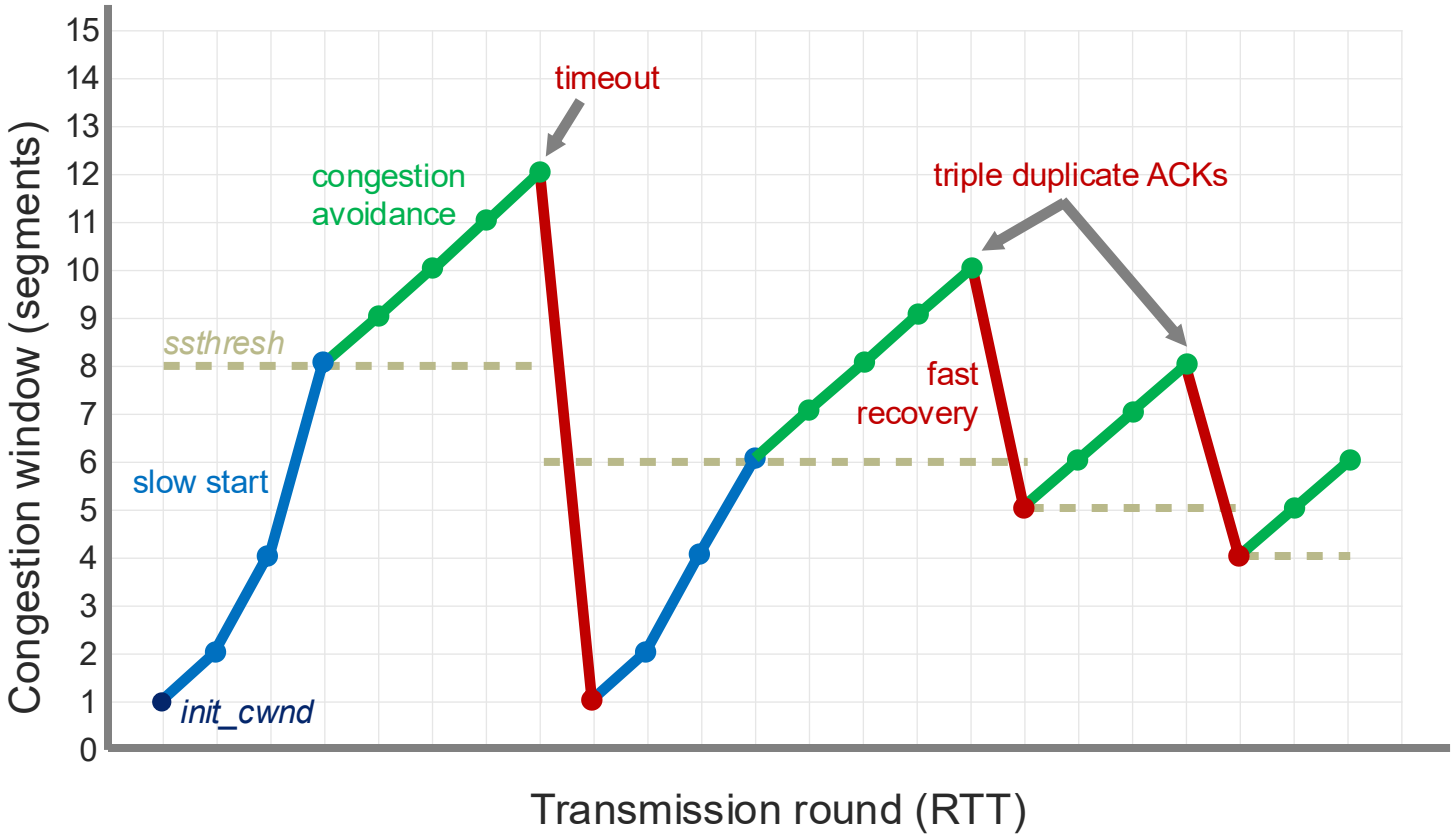
Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – retransmit it!



TCP Fast Recovery

- Fast recovery
  - When TCP fast retransmit occurs, skip slow start
  - Congestion window becomes 1/2 previous
  - Start additive increase immediately
    - Restoring AIMD

TCP Congestion Window Dynamics Summary



Learning Objectives

- Host-centric: TCP Congestion Control (cont'd)
  - TCP Slow Start, Congestion Avoidance
  - Fast Retransmit, Fast Recovery
  - **Beyond loss-based congestion control**
- Quality of Service

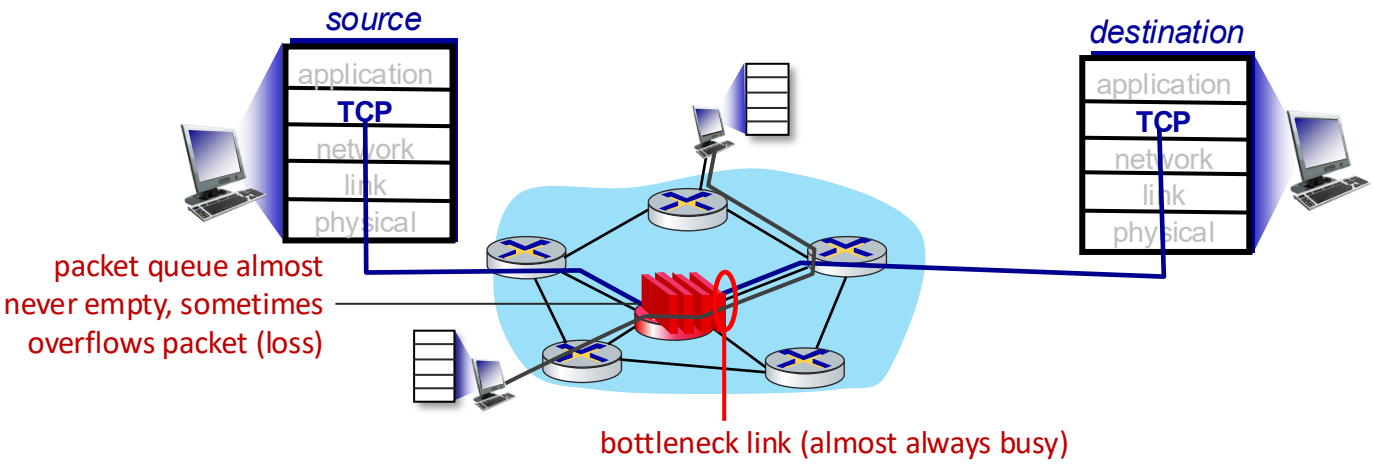
Recap: Detecting Congestion

- How can a TCP sender determine that the network is congested?
- Network could tell it
  - ICMP Source Quench: during times of overload the signal itself could be dropped (and add to congestion)!
  - ECN: Explicit Congestion Notification
- Packet delays go up (knee of load-delay curve)
  - Tricky: noisy signal (delay often varies considerably)
- Packet loss
  - Fail-safe signal that TCP already has to detect
  - Complication: non-congestive loss

Is loss really a good congestion signal?

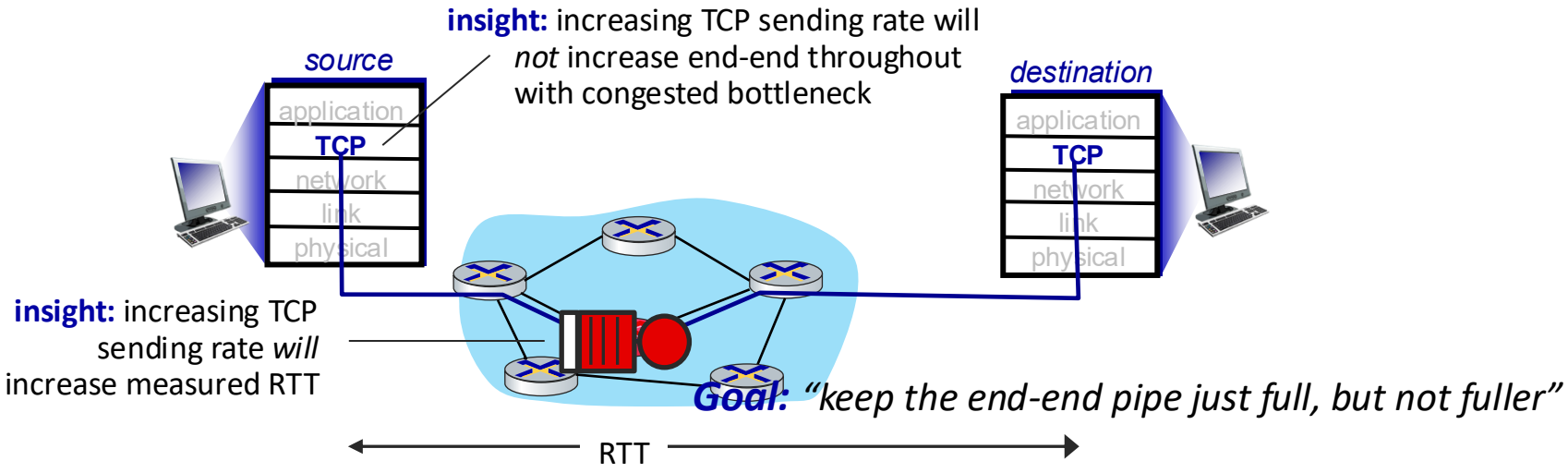
The congested "bottleneck link"

- Classic TCP increases the sending rate until packet loss occurs at some router's output: the *bottleneck link*



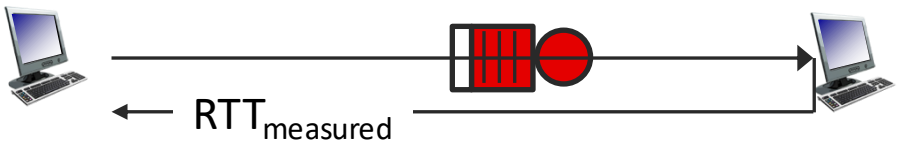
The congested "bottleneck link"

- Classic TCP increases the sending rate until packet loss occurs at some router's output: the *bottleneck link*
- Understanding congestion: useful to focus on congested bottleneck link



Delay-based TCP congestion Control

Keep bottleneck link busy transmitting, but avoid high delays/buffering



$$\text{measured throughput} = \frac{\text{\# bytes sent in last RTT interval}}{RTT_{measured}}$$

Delay-based approach:

- RTT_{min}: minimum observed RTT (uncongested path)
- uncongested throughput with congestion window `cwnd` is `cwnd`/RTT_{min}

```
if measured throughput "very close" to uncongested throughput
        increase cwnd linearly          /* since path not congested */
else if measured throughput "far below" uncongested throughout
        decrease cwnd linearly          /* since path is congested */
```

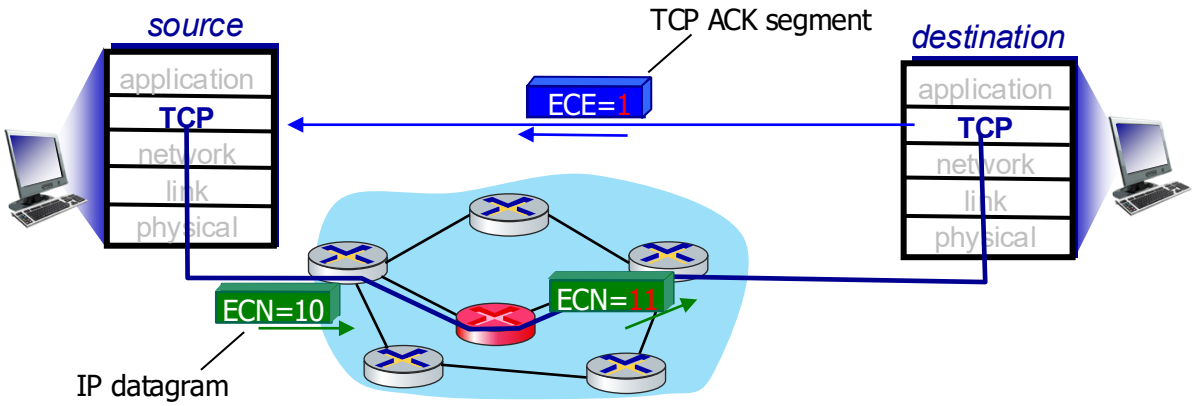
Delay-based TCP congestion Control

- Congestion control without inducing/forcing loss
- Maximizing throughout (“keeping the just pipe full… ”) while keeping delay low (“…but not fuller”)
- A number of TCPs take a delay-based approach
  - TCP Vegas (1994)
    - Limited deployment in production
  - TCP BBR (2016, developed by Google)
    - BBR is replacing Cubic (2006, loss-based, default in Linux)

Explicit Congestion Notification (ECN)

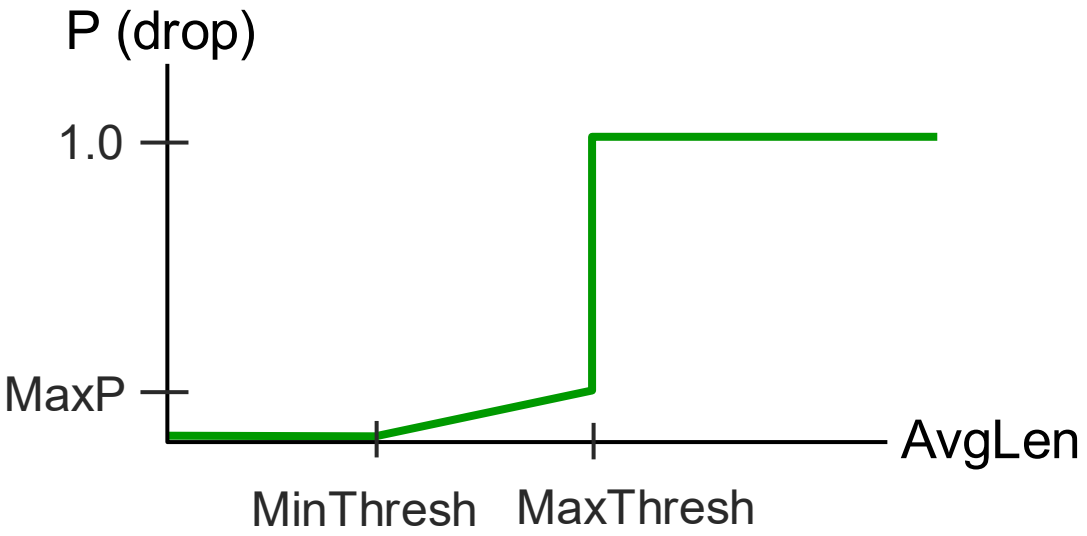
TCP may implement *network-assisted* congestion control:

- 2 bits in IP header (ToS) marked *by network router* to indicate congestion
  - *policy* to determine marking chosen by network operator
- congestion indication (ECN=11 in IP header) carried to destination
- destination sets ECE bit (in TCP header) on ACK segment to notify sender of congestion
- sender cuts cwnd by half and sets CWR bit (in TCP header) on the next segment



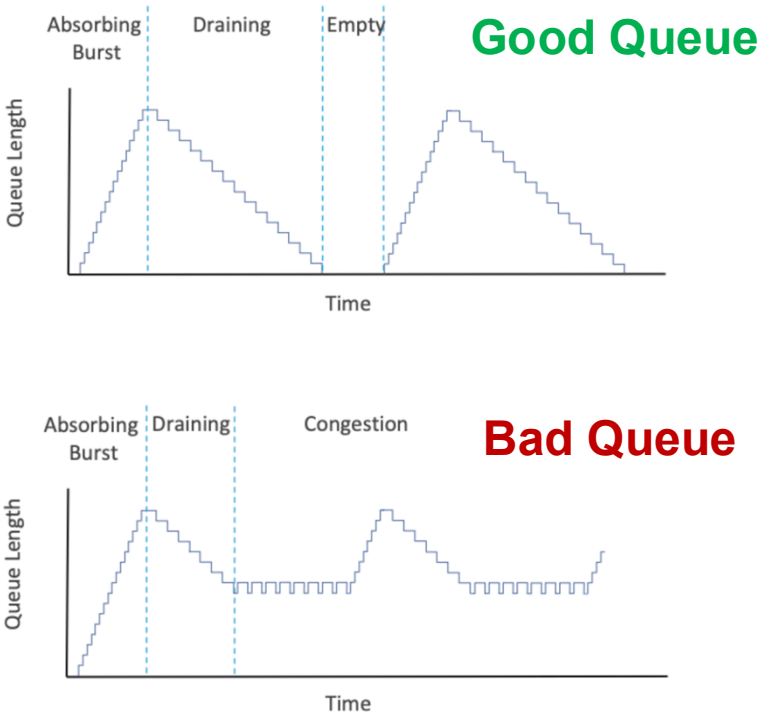
Active Queue Management (AQM)

- Random early detection (RED)
  - Compute average queue length on router (AvgLen)
  - Generally: drops more packets as AvgLen increases
  - Can be adapted to mark ECN bits
  - Never widely deployed



Active Queue Management (AQM)

- Controlled Delay (CoDel)
  - Computes *sojourn time* (≈ queue delay) for every packet leaving the queue
  - Maintains the minimum sojourn time in the most recent *interval* (default: 100 ms)
  - Begins dropping if minimum sojourn time > *target* (default: 5 ms)
  - Can be adapted to mark ECN bits
  - Implemented in Linux, incl. OpenWRT



Learning Objectives

- Host-centric: TCP Congestion Control (cont'd)
  - TCP Slow Start, Congestion Avoidance
  - Fast Retransmit, Fast Recovery
  - Beyond loss-based congestion control
- **Quality of Service**

Quality of Service

- Broader area
  - Related to reservation-based congestion control
  - Related to applications
- How “good” are late data and low-throughput channels?
- It depends on the application. Do you care if...
  - Your email takes 30 min to reach your friend?
  - You have to spend 30 min to make a cheaper plane reservation on the web?
  - Your call to 911 takes 30 min to go through your nifty new IP phone service?

Application Requirements

- Internet currently provides one single class of "best-effort" service
  - No assurances about delivery
- High speed networks have enabled new applications
  - Require "deliver on time" assurances from the network
  - Real-time applications
    - Sensitive to the timeliness of their data
    - Voice
    - Video
    - Industrial control

Timely Delivery

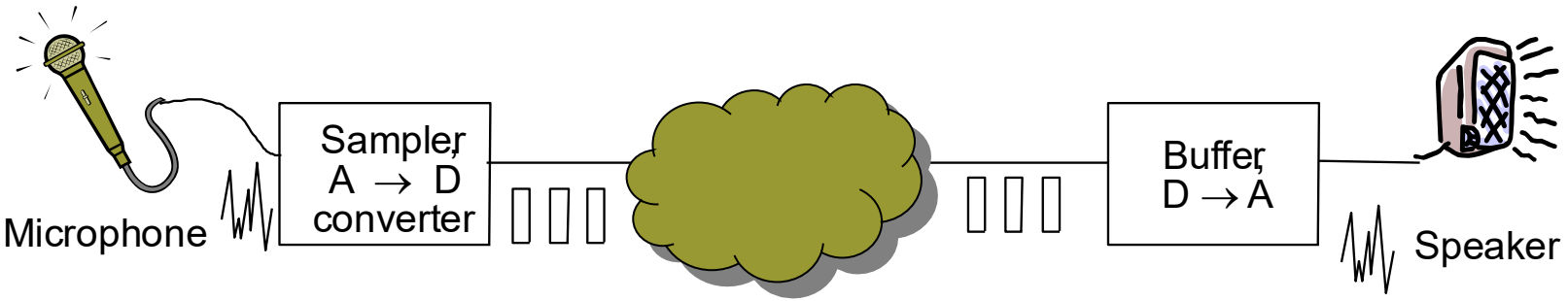
- How to achieve timely delivery
  - When actual RTT is small relative to acceptable delay
    - Retransmit
  - When base RTT (no queuing delay) is large relative to acceptable delay
    - Impossible
  - Otherwise possible, but not through retransmission

Timely Delivery

- Within the 48-state U.S
  - Base RTT (no queueing delay) peaks around 75 ms
  - Actual RTT is often 10-100 ms
- Humans notice about 50 ms delay for voice
  - Support delay preferences in the network; called quality of service, or QoS

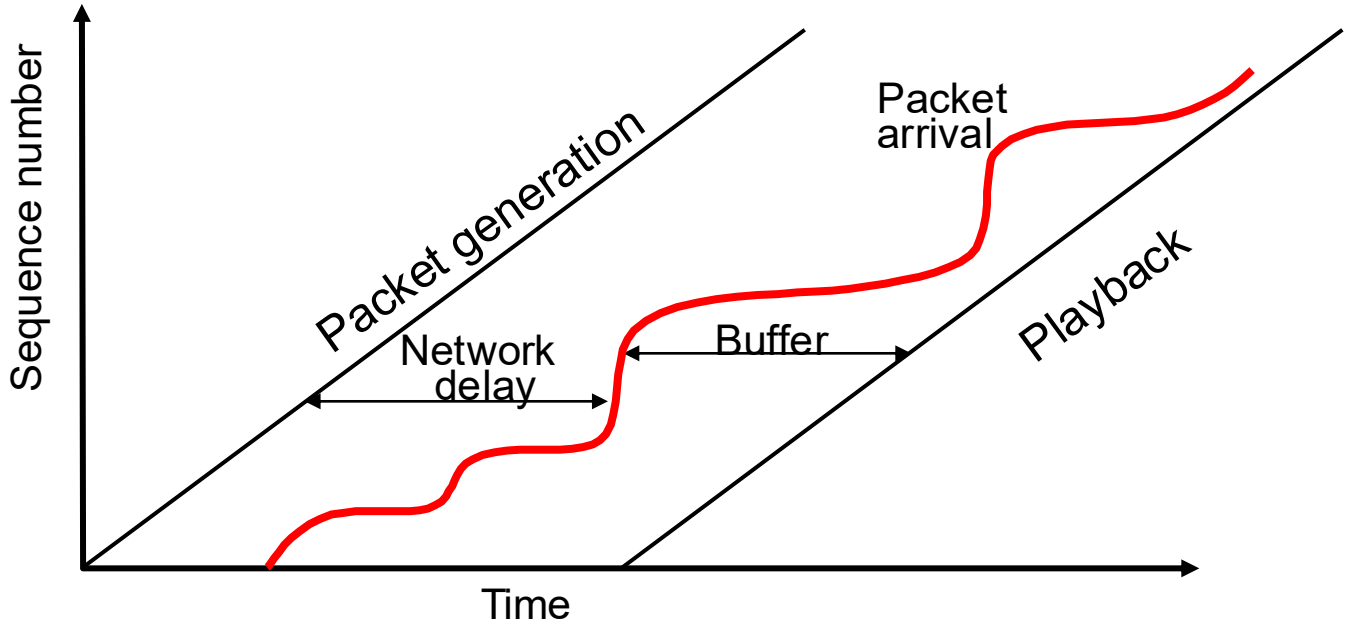
Real-time Applications

- Two types or applications
  - Hard real-time
  - Elastic (soft real-time)
- Example real-time application requirements — audio
  - Sample voice once every 125μs
  - Each packet has a playback time
  - Packets experience variable delay in network
  - Add constant offset to playback time – playback point

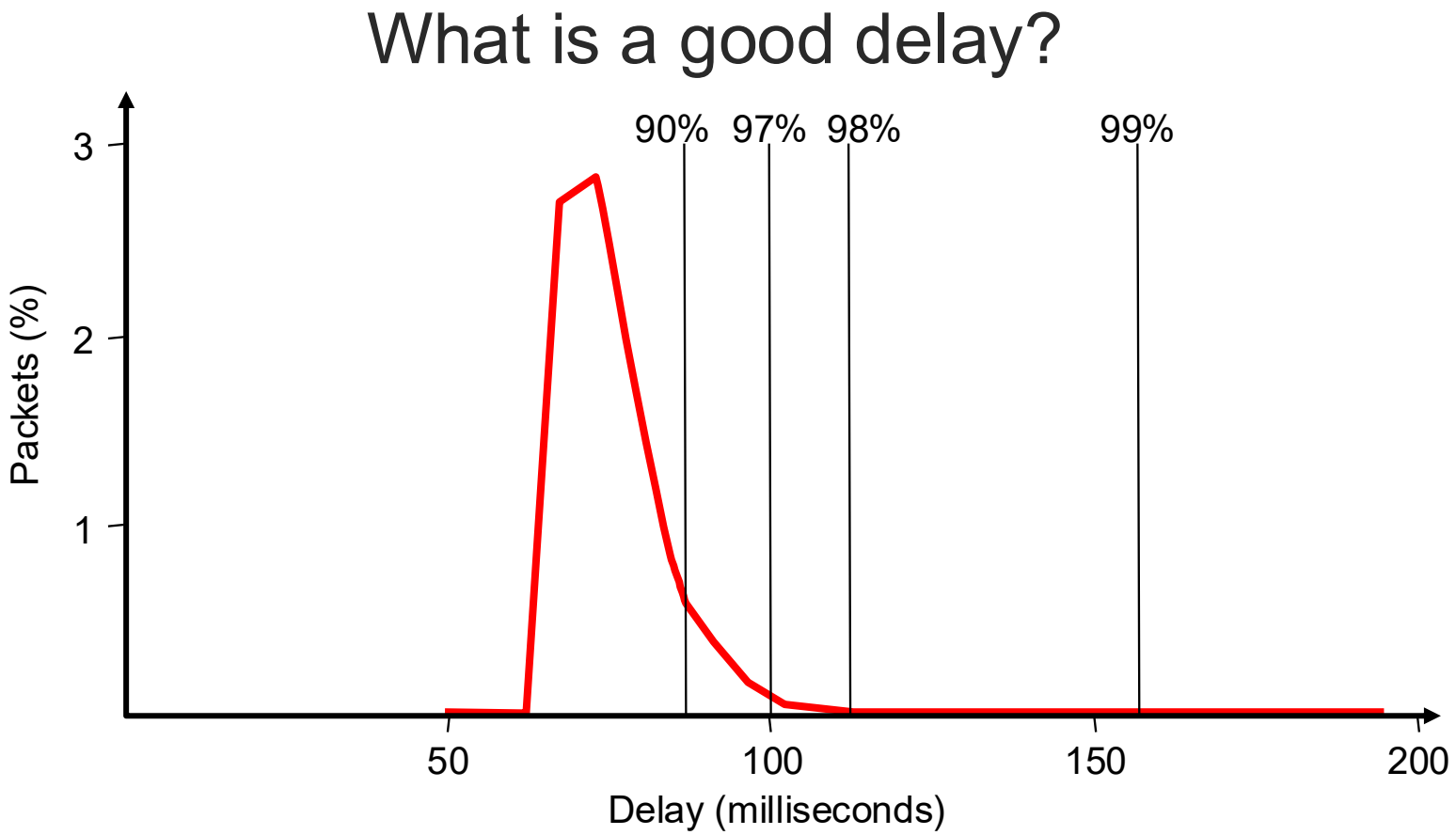


Real-Time Applications

- Playback Buffer



Delay Distribution



Quality of Service Approaches

- Approach : Admission control
  - Flow tells the network what it wants
  - Network decides if flow can be admitted
- Fine-grained
  - Provide QoS to individual applications or flows
    - Example: Resource Reservation Protocol (RSVP)
- Coarse-grained
  - Provide QoS to large classes of data or aggregated flows
    - Example: Differentiated Services (DIFFSERV)

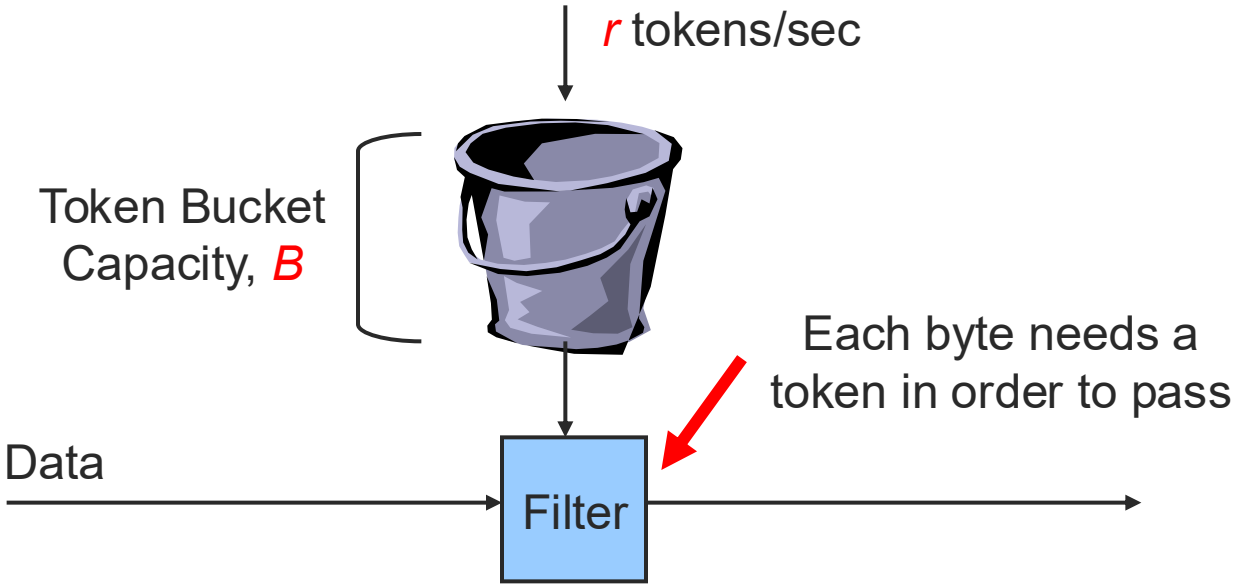
Mechanisms

- Flow specification
  - Tell the network what the flow wants
- Admission control
  - Network decides if it can handle flow
- Reservation
  - Enable admission control
- Packet classification
  - Map packets to flows
- Scheduling
  - Forwarding policy

Characterizing a Flow

- Describe flow's traffic characterization
  - Average bandwidth + burstiness: token bucket filter
  - Token fill rate: *r*
  - Bucket size: *B*
- Use
  - Must have *n* tokens to send *n* bytes
  - Start with no tokens
  - Accumulate tokens at rate of *r* per second
  - Can accumulate no more than *B* tokens

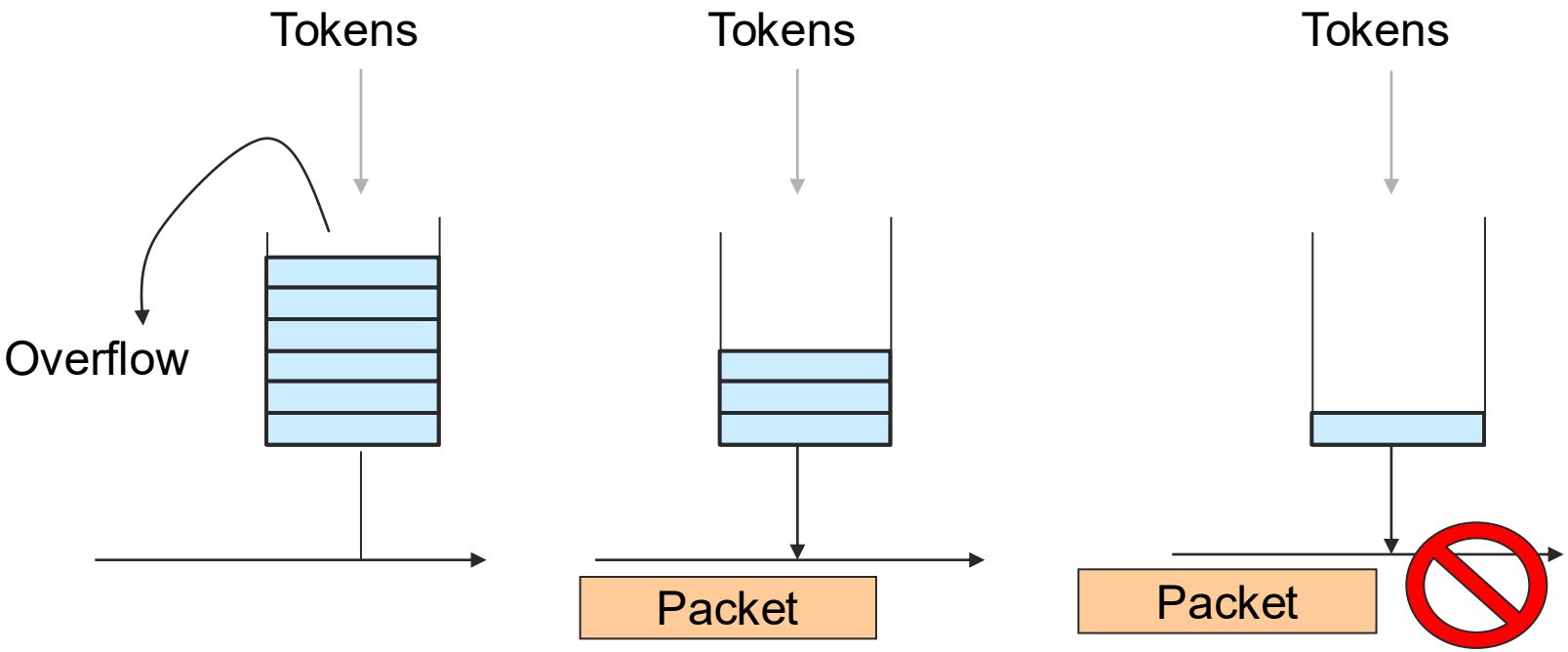
Token Bucket Filters



Dropping Filter: drops packets if token is not available

Buffered Filter: buffers data until tokens become available

Token Bucket Operation



Buffer tokens up to capacity of bucket

Enough tokens → packet goes through, tokens removed

Not enough tokens → wait for tokens to accumulate

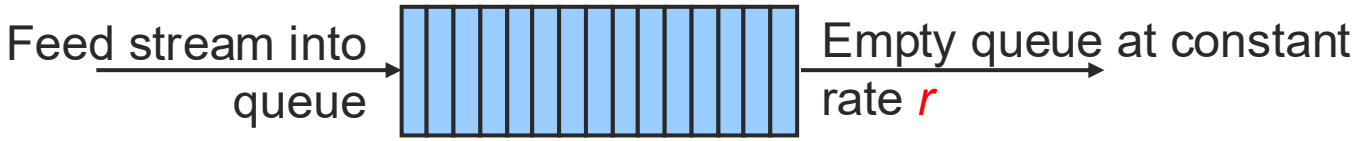
Token Bucket Filters

- Given a finite length data stream, will it be affected by a token bucket filter?

Not if during every time interval, the number of bytes is less than or equal to $B + rt$, where t is the length of the interval

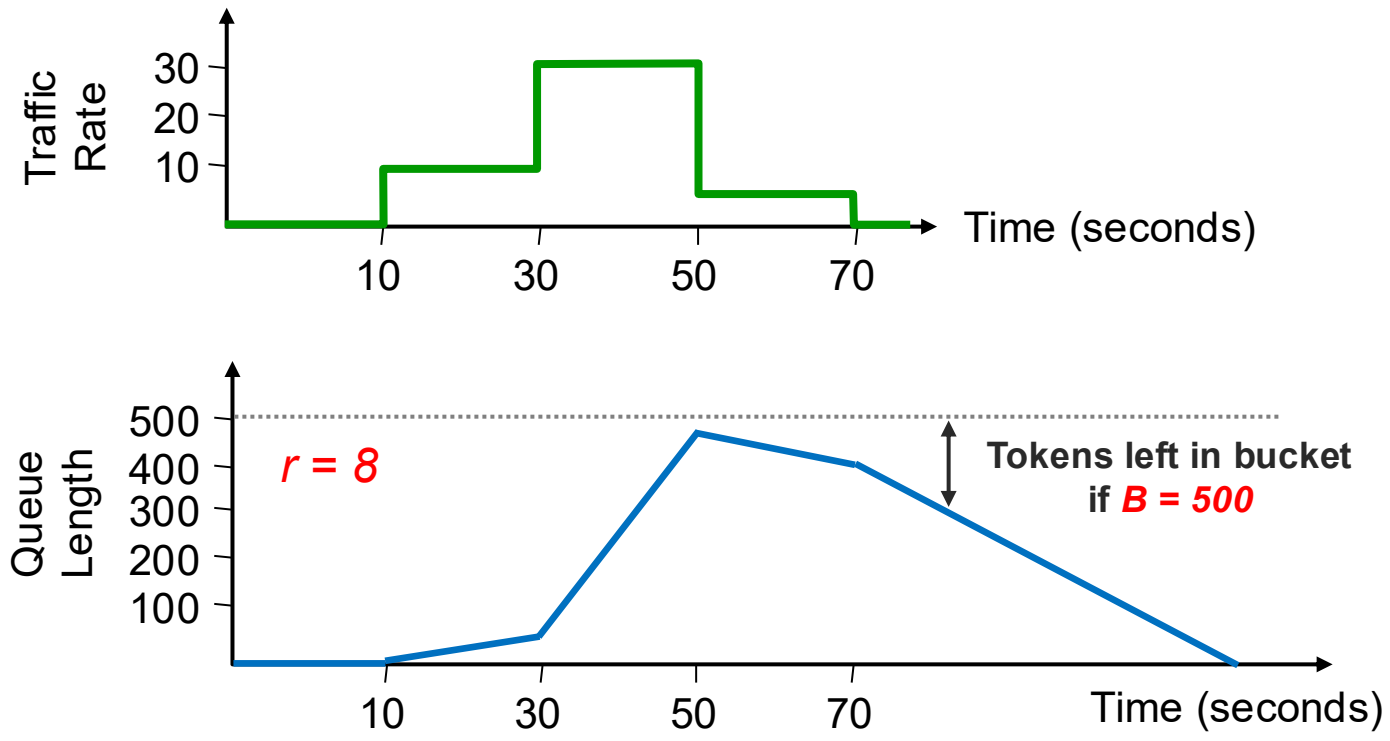
Token Bucket Filters

- Given a token rate *r* and a finite data trace, how can the minimum token bucket size *B* be found such that the filter has no effect?



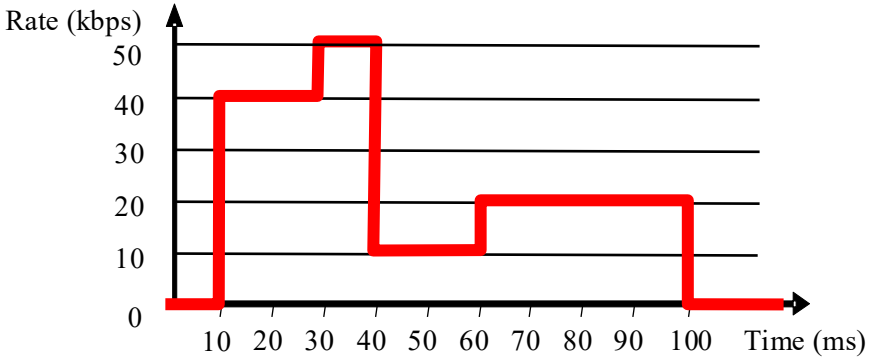
- Simply observe the maximum buffer size. Why?
  - Buffer tracks backlog of arriving data
  - Token bucket tracks unused service capacity
  - If the buffer is truncated to size *B*, then the number of empty buffer positions is equivalent to the number of tokens in an *(r, B)* token bucket filter

Token Bucket Filters



Token Bucket Filters

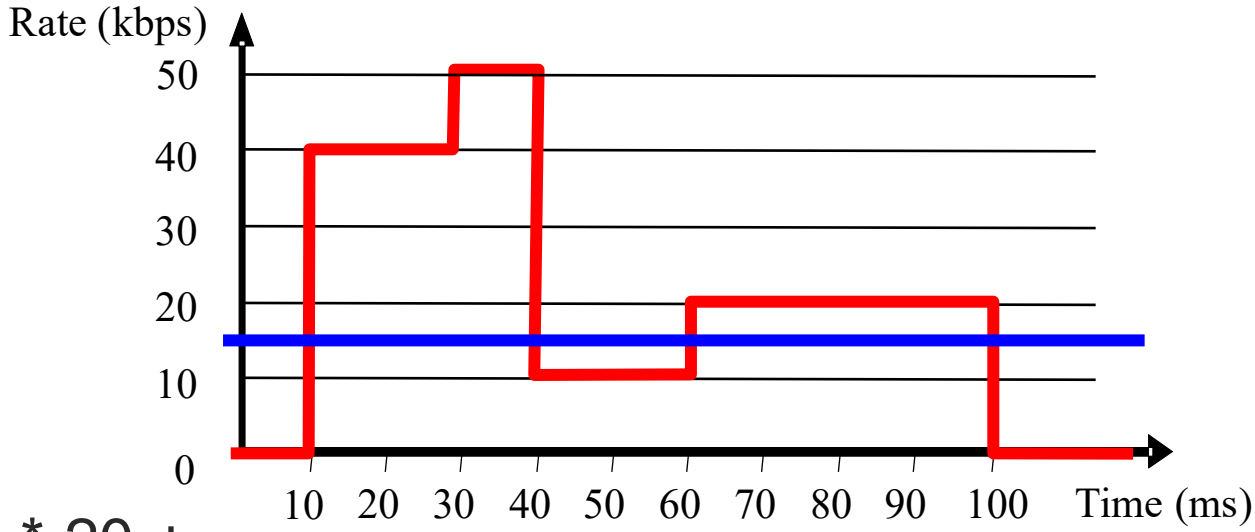
- *r = 15* kbps



- What is the minimum size of *B* required so that the filter lets the stream pass with no loss or delay?

Token Bucket Filters

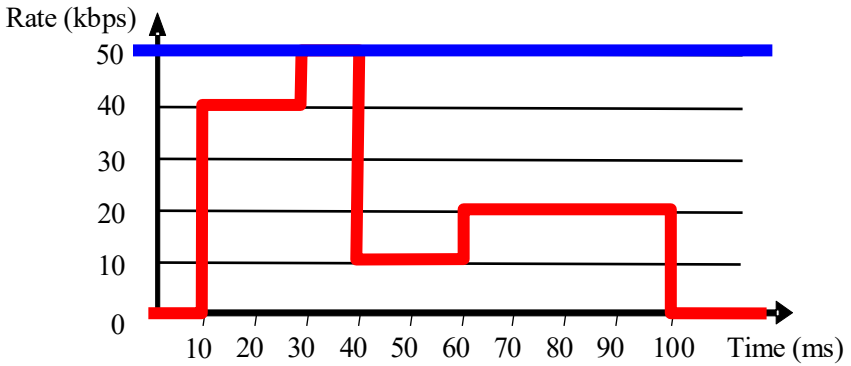
- *r = 15* kbps



- Min *B* =
  (40 – 15) * 20 +
  (50 – 15) * 10 –
  (15 – 10) * 20 +
  (20 – 15) * 40
  = 950 bits

Token Bucket Filters

- What is the minimum B as a function of r ≥ 0



- If r ≥ 50
  B = 0
- If 50 > r ≥ 40
  Min B = (50 – r) * 10
- If 40 > r ≥ 20
  Min B = (40 – r) * 20 + (50 – r) * 10
- If 20 > r ≥ 10
  Min B = (40 – r) * 20 + (50 – r) * 10 – (r – 10) * 20 + (20 – r) * 40
- If 10 > r ≥ 0
  Min B = (40 – r) * 20 + (50 – r) * 10 + (10 – r) * 20 + (20 – r) * 40

Traffic Shaping / Rate Limiting: Token Bucket vs. Leaky Bucket

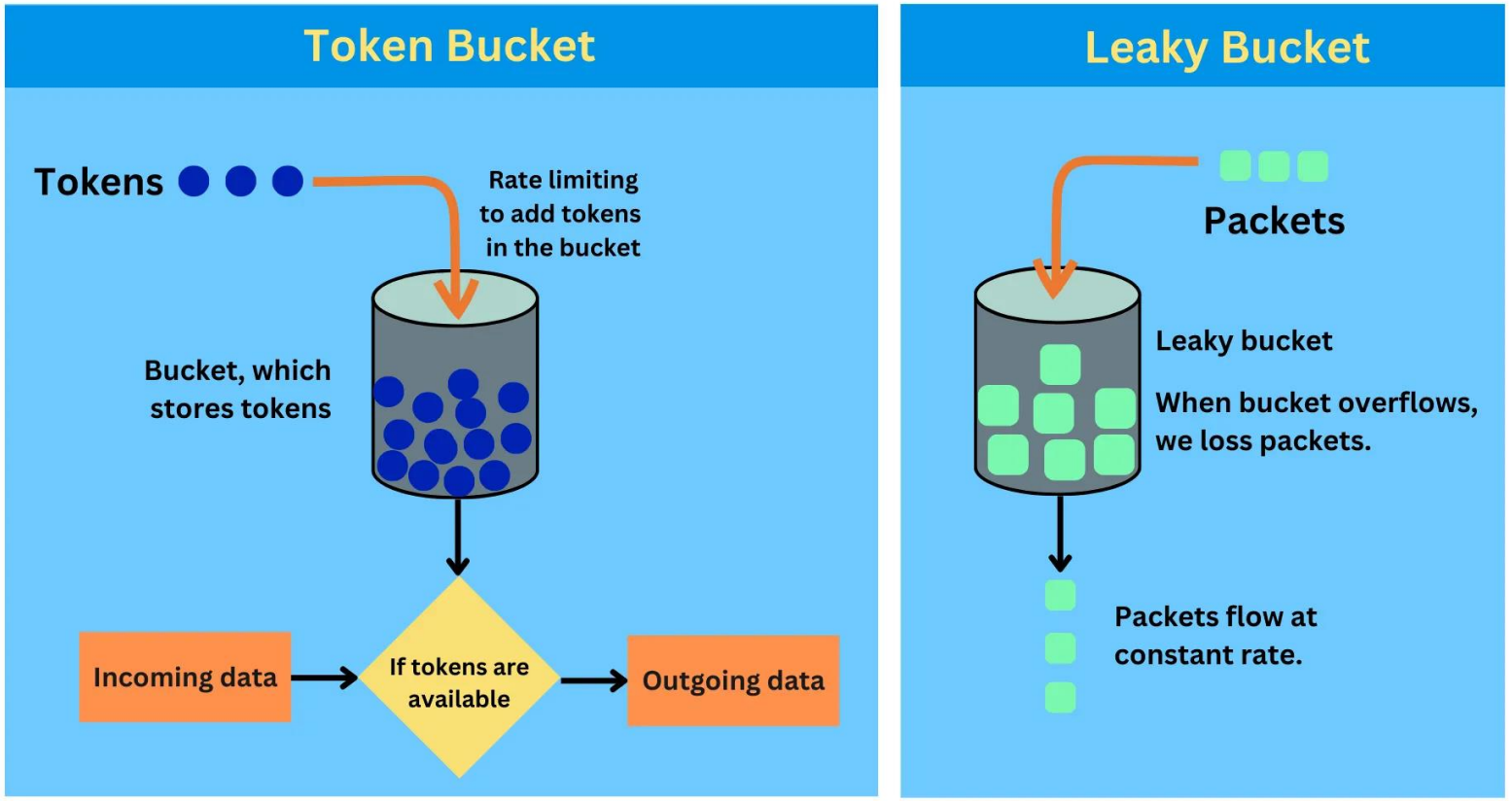


Image source: blog post

Differentiated Services

- Goal
  - Scalability through the use of only a small number of service classes
    - Two classes
      - Regular and premium (i.e., first class and bulk mail)
    - Diffserv
      - Proposes 6 bits of IP ToS field (2^6 = 64 classes)
- Questions
  - Who is allowed to set the premium bit?
    - Typically an ISP
    - Should we allow an individual customer or application?
  - How do routers react to such a classification?
    - IETF has specified per-hop behavior

Differentiated Services

- Expedited forwarding
  - Per-hop behavior, minimal delay and loss
  - Need to strictly limit the load of traffic receiving expedited forwarding
    - Give strict priority
    - Use weighted fair queueing (WFQ) and assign sufficiently large weights for traffic receiving expedited forwarding

Differentiated Services

- Assured forwarding
  - Per-hop behavior
  - Like RED but with “in” and “out” packets (RIO)
    - in-profile, conforming traffic vs. out-of-profile, excess traffic
  - Does not reorder packets
- Profile meters at the edges of ISP networks could mark packets as “in” or “out”