Congestion Control

Phenomenon: when too much traffic enters into system, performance degrades

⟶ excessive traffic can cause congestion

Problem: regulate traffic influx such that congestion does not occur

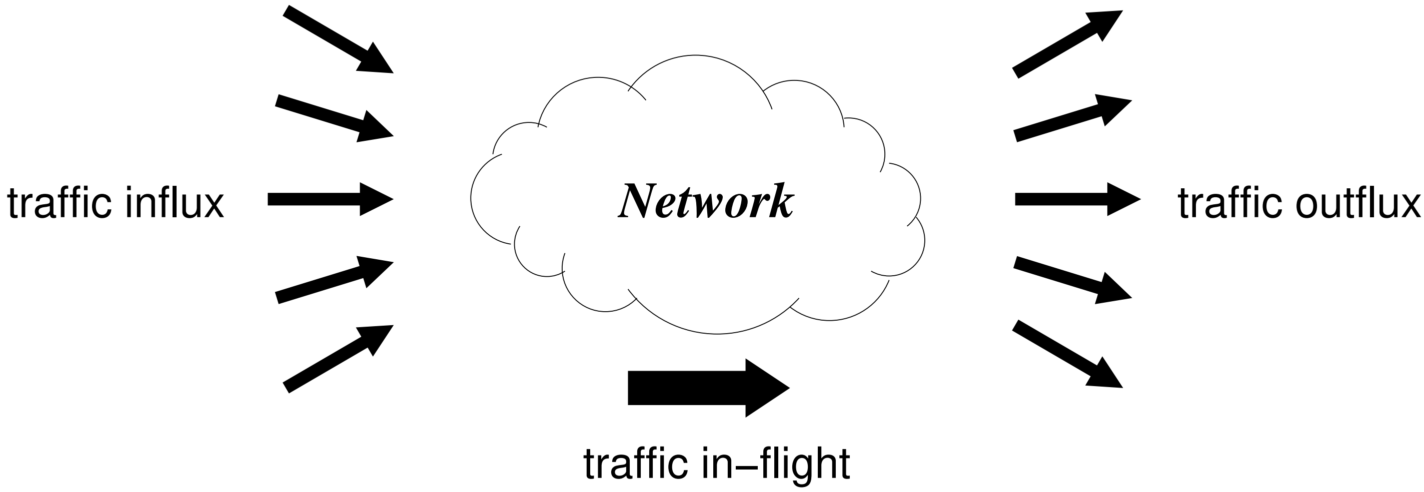
⟶ not too fast, not too slow

⟶ congestion control

⟶ first question: what is congestion?

Viewpoint: 3 components

→ (1) traffic coming in, (2) in transit, (3) going out



At time instance t:

- traffic influx: $\lambda(t)$ "offered load" (bps)
- traffic outflux: $\gamma(t)$ "throughput" (bps)
- traffic in-flight: $Q(t)$ "load" (volume, i.e., no. of packets)

Examples:

Highway system:

- traffic influx: no. of cars entering highway per second
- traffic outflux: no. of cars exiting highway per second
- traffic in-flight: no. of cars traveling on highway

$\longrightarrow$ at time instance t

California Dept. of Transportation (Caltrans)

Water faucet and sink:

- traffic influx: water influx per second
- traffic outflux: water outflux per second
- traffic in-flight: water level in sink

→ not good if sink overflows

faucet.com

Many examples: heating/cooling system with thermostat . . .

What is the meaning of congestion?

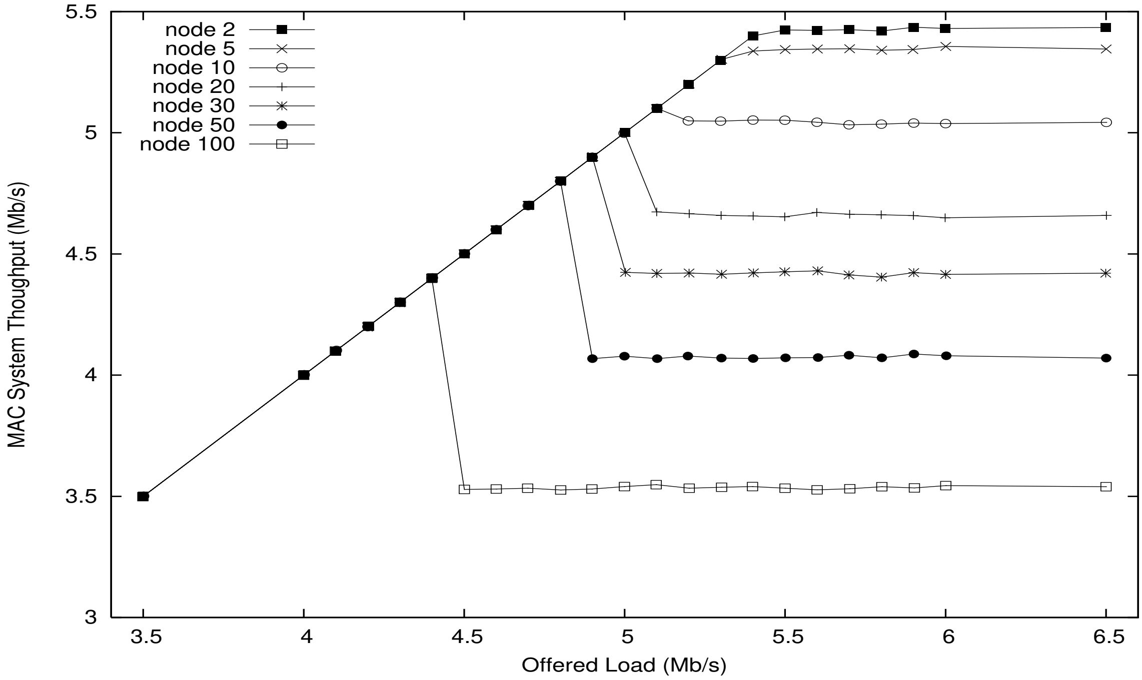
→ when sending too fast, throughput starts to go down

In the water faucet/sink example: is there congestion?

What about highway system?

Example: 802.11b WLAN:

- Throughput



$\longrightarrow$ unimodal or bell-shaped

$\longrightarrow$ what is load $Q(t)$ in wireless LAN?

What we can control:

→ traffic influx rate $\lambda(t)$

→ no power over anything else

Congestion control: how to regulate influx rate $\lambda(t)$—not too fast, not too slow—so that throughput $\gamma(t)$ is maximized

→ many applications

→ TCP congestion control

→ multimedia video/audio streaming

Pseudo Real-Time Multimedia Streaming:

Examples: streaming client/server apps

→ real-time vs. pseudo-real-time

"Pseudo" because of prefetching trick

→ application is given headstart before playback

→ fill & prevent client buffer from becoming empty

Main steps:

- prefetch X seconds worth of audio/video data

  $\rightarrow$ initial playback delay

- keep fetching audio/video data such that X seconds worth of future data resides in receiver's buffer

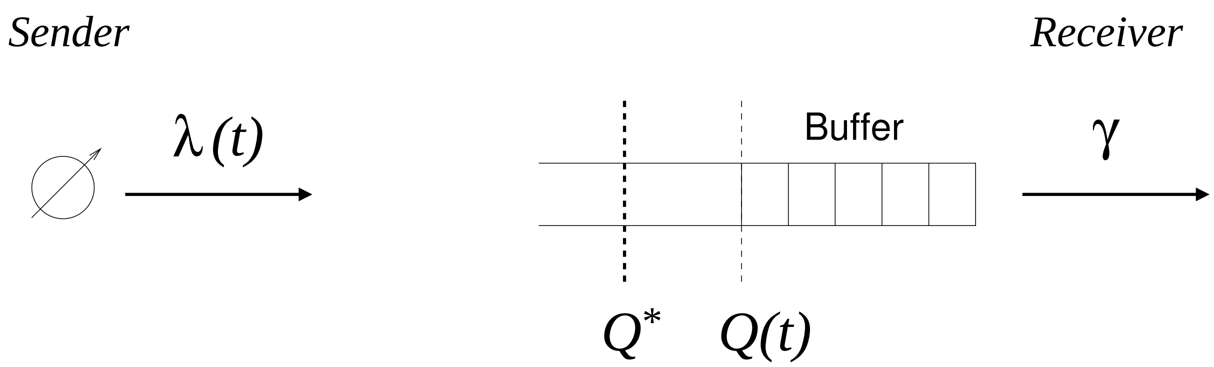
  $\rightarrow$ protects against, and hides, spurious congestion

  $\rightarrow$ don't keep more than X

  $\rightarrow$ potential for wasting resources: bandwidth, memory, CPU

If streaming is done well, user experiences continuous playback without quality disruptions

Pseudo real-time application architecture:



- $Q(t)$: current buffer level
- Q^*: desired buffer level
- γ: throughput—fixed playback rate

  $\rightarrow$ e.g., 24 frames-per-second (fps) for movies

Goal: keep $Q(t) \approx Q^*$ by adjusting $\lambda(t)$

$\longrightarrow$ don't buffer too much: resource wastage

$\longrightarrow$ don't buffer too little: cannot hide congestion

How does load $Q(t)$ vary?

$\rightarrow$ obeys simple rule

Compare two time instances t and $t+1$.

At time $t+1$:

$$Q(t+1) = Q(t) + \lambda(t) - \gamma(t)$$

- $Q(t)$: what was there to begin with
- $\lambda(t)$: what newly arrived
- $\gamma(t)$: what newly exited
- $\lambda(t) - \gamma(t)$: net influx (positive or negative)
- note: $Q(t)$ cannot be negative by its meaning

  $\rightarrow$ no. of packets

  $\rightarrow Q(t+1) = \max\{0, Q(t) + \lambda(t) - \gamma(t)\}$
- missing item?

Other applications.

Ex. 1: Router congestion control

$\longrightarrow$ active queue management (AQM)

- receiver is a router/switch
- Q^* is desired buffer occupancy/delay at router

  $\rightarrow$ too much buffering: bufferbloat (Jim Getty)
- router throttles sender(s) to maintain Q^*

  $\rightarrow$ router sends control packets to senders

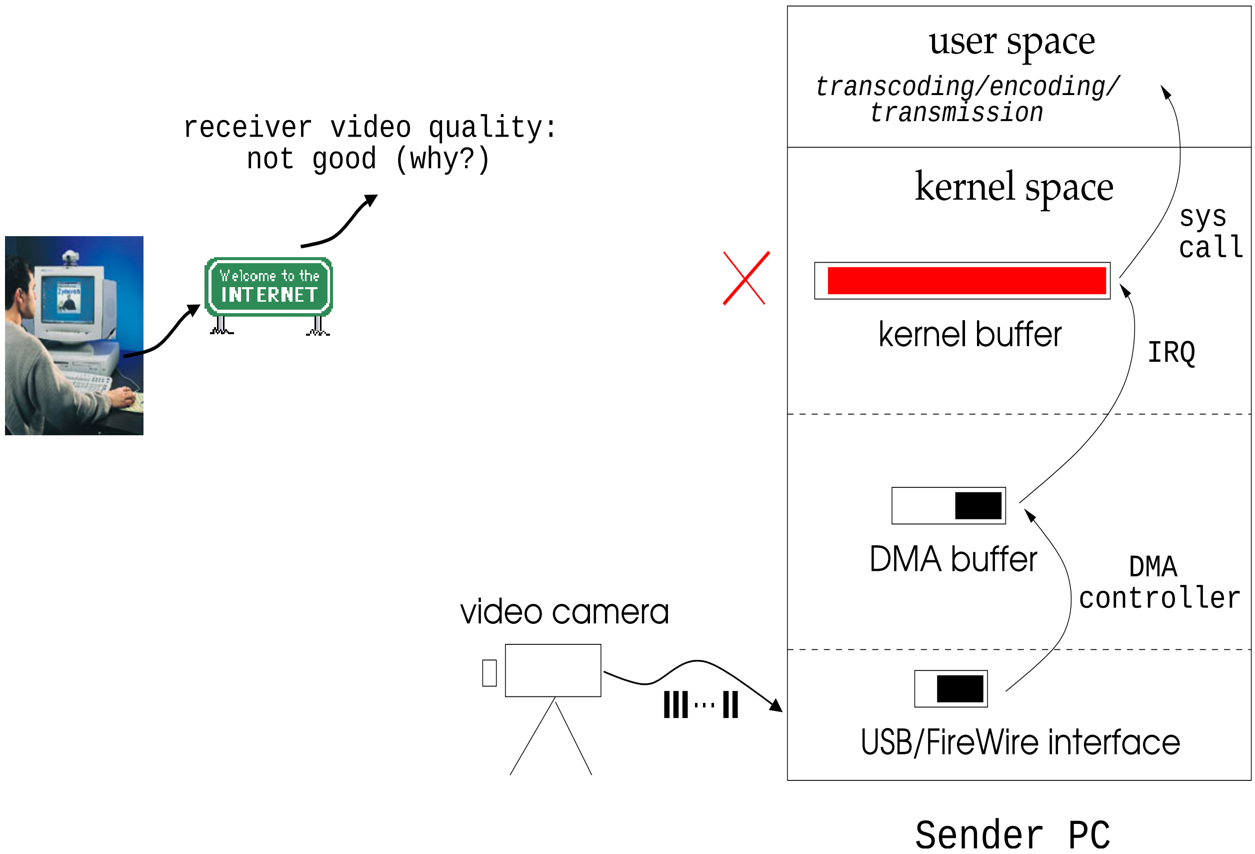
  $\rightarrow$ instruction: slow down, go faster, stay put

Ex. 2: Desktop videoconferencing

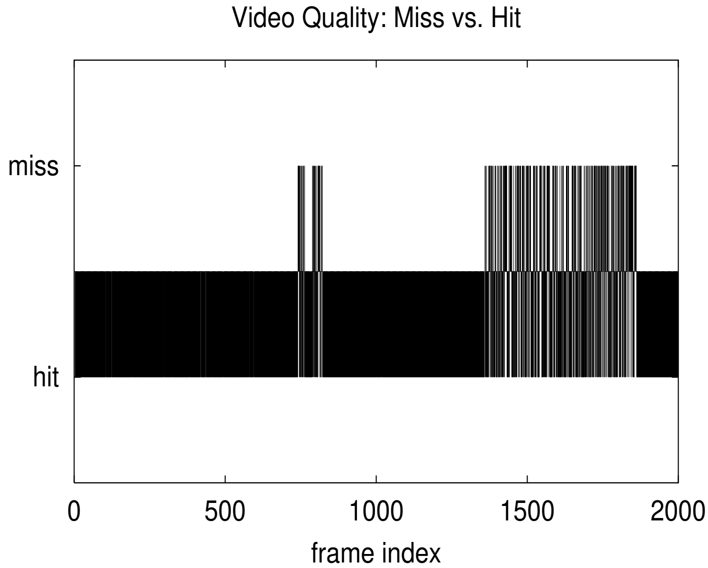
→ e.g., AOL, MSN, Skype, Yahoo

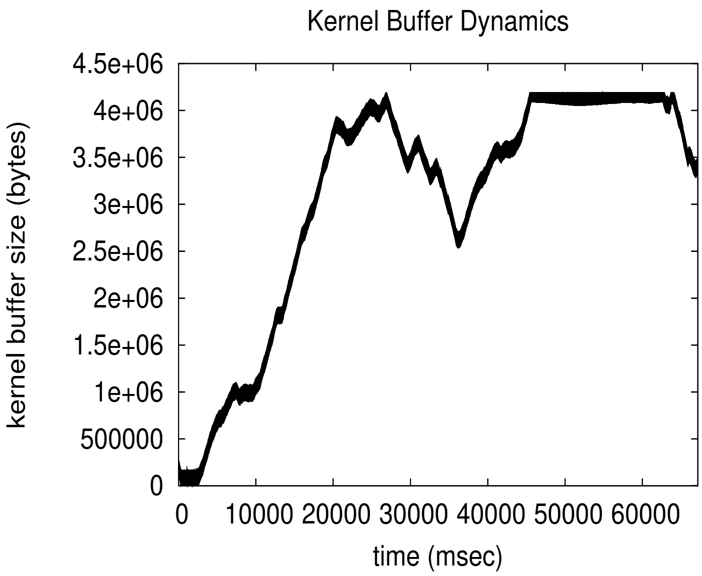
→ video quality may not be good: why?

→ common misconception: sole culprit is network



Performance consequences:





Thus: pseudo real-time multimedia streaming application of congestion control

$\longrightarrow$ producer/consumer rate mismatch problem

Note: producer/consumer problem in OS

$\longrightarrow$ focus on orderly access of shared data structure

$\longrightarrow$ mutual exclusion

$\longrightarrow$ e.g., use of counting semaphores

$\longrightarrow$ necessary but insufficient

What is the goal:

$\longrightarrow$ achieve $Q(t) = Q^*$

$\longrightarrow$ or close to it: $|Q(t) - Q^*| < \varepsilon$

Basic idea:

- if $Q(t) = Q^*$ do nothing
- if $Q(t) < Q^*$ increase $\lambda(t)$

  $\rightarrow$ too little in the buffer
- if $Q(t) > Q^*$ decrease $\lambda(t)$

  $\rightarrow$ too much in the buffer

Rule of thumb: called control law

Since state of receiver buffer must be conveyed to sender who adjusts $\lambda(t)$:

$\longrightarrow$ called feedback control

$\longrightarrow$ also closed-loop control

Network protocol implementation:

→ design choices

- control action undertaken at sender

  → smart sender/dump receiver

  → preferred mode of Internet protocols

  → when might the opposite be better?

- receiver informs sender of Q^* and $Q(t)$

  → feedback could just be gap $Q^* - Q(t)$

  → or simply up/down binary indication

Key question in feedback congestion control:

$\longrightarrow$ **how much** to increase/decrease $\lambda(t)$

Desired state of the system:

$$Q(t) = Q^* \text{ and } \lambda(t) = \gamma$$

$\longrightarrow$ why is $\lambda(t) = \gamma$ needed?

$\longrightarrow$ system is in equilibrium or steady-state

Starting state:

$\longrightarrow$ empty buffer and nothing is being sent

$\longrightarrow$ think of iTunes, Netflix, Spotify, etc.

$$\text{i.e.,} \quad Q(t) = 0 \text{ and } \lambda(t) = 0$$

Time evolution (or dynamics): track $Q(t)$ and $\lambda(t)$