**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

→ at time instance $t$
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

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

\[ Q(t) \]

\[ \text{unimodal or bell-shaped} \]

\[ \text{what is load } Q(t) \text{ in wireless LAN?} \]
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
→ e.g., RealPlayer, iTunes, VoD (video-on-demand), Internet radio

“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
  $\rightarrow$ causes 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:

\[
\begin{align*}
\text{Sender} & \quad \lambda(t) \quad \text{Buffer} \quad \gamma \\
\emptyset & \quad Q^* \\
\end{align*}
\]

- \( Q(t) \): current buffer level
- \( Q^* \): desired buffer level
- \( \gamma \): throughput—fixed playback rate
  \( \rightarrow \) e.g., 24 frames-per-second (fps) for movies

Goal: keep \( Q(t) \approx Q^* \) by adjusting \( \lambda(t) \)
  
  \( \rightarrow \) don’t buffer too much: resource wastage
  
  \( \rightarrow \) don’t buffer too little: cannot hide congestion
How does load $Q(t)$ vary?
→ 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
  → no. of packets
  → $Q(t + 1) = \max\{0, Q(t) + \lambda(t) - \gamma(t)\}$

• missing item?
Other applications.

Ex. 1: Router congestion control

→ active queue management (AQM)

• receiver is a router/switch

• $Q^*$ is desired buffer occupancy/delay at router
  → too much buffering: bufferbloat (Jim Getty)

• router throttles sender(s) to maintain $Q^*$
  → router sends control packets to senders
  → 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:

Video Quality: Miss vs. Hit

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

→ producer/consumer rate mismatch problem

Note: producer/consumer problem in OS

→ focus on orderly access of shared data structure

→ mutual exclusion

→ e.g., use of counting semaphores

→ necessary but insufficient
What is the goal?

\[\rightarrow \text{ achieve } Q(t) = Q^*\]

\[\rightarrow \text{ or close to it: } |Q(t) - Q^*| < \varepsilon\]

Basic idea:

• if \(Q(t) = Q^*\) do nothing

• if \(Q(t) < Q^*\) increase \(\lambda(t)\)
  \[\rightarrow \text{ too little in the buffer}\]

• if \(Q(t) > Q^*\) decrease \(\lambda(t)\)
  \[\rightarrow \text{ too much in the buffer}\]

Rule of thumb: called control law

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

\[\rightarrow \text{ called feedback control}\]

\[\rightarrow \text{ also closed-loop control}\]
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:

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

→ already know in which direction

Desired state of the system:

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

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

→ system is in equilibrium or steady-state

Starting state:

→ empty buffer and nothing is being sent

→ think of iTunes, Netflix, Spotify, etc.

i.e., $Q(t) = 0$ and $\lambda(t) = 0$
Time evolution (or dynamics): track $Q(t)$ and $\lambda(t)$
Congestion control methods: A, B, C and D

Method A:

- if $Q(t) = Q^*$ then $\lambda(t + 1) \leftarrow \lambda(t)$
- if $Q(t) < Q^*$ then $\lambda(t + 1) \leftarrow \lambda(t) + a$
- if $Q(t) > Q^*$ then $\lambda(t + 1) \leftarrow \lambda(t) - a$

where $a > 0$ is a fixed parameter

\[ \rightarrow \text{ called linear increase/linear decrease} \]

Question: how well does it work?

Example:

- $Q(0) = 0$
- $\lambda(0) = 0$
- $Q^* = 100$
- $\gamma = 10$
- $a = 1$
With \( a = 0.5 \):
With $a = 3$:
With $a = 6$: 

![Load Evolution Graph](image1)

![Lambda Evolution Graph](image2)
Remarks:

• Method A isn’t that great no matter what \( a \) value is used
  \[ \rightarrow \text{keeps oscillating} \]

• Actually: would lead to unbounded oscillation if not for physical restriction \( \lambda(t) \geq 0 \) and \( Q(t) \geq 0 \)
  \[ \rightarrow \text{i.e., bottoms out} \]
  \[ \rightarrow \text{easily seen: start from non-zero buffer} \]
  \[ \rightarrow \text{e.g., } Q(0) = 110 \]
With $a = 1$, $Q(0) = 110$, $\lambda(0) = 11$: 
Method B:

- if $Q(t) = Q^*$ then $\lambda(t + 1) \leftarrow \lambda(t)$
- if $Q(t) < Q^*$ then $\lambda(t + 1) \leftarrow \lambda(t) + a$
- if $Q(t) > Q^*$ then $\lambda(t + 1) \leftarrow \delta \times \lambda(t)$

where $a > 0$ and $0 < \delta < 1$ are fixed parameters

Note: only decrease part differs from Method A.

$\longrightarrow$ linear increase with slope $a$
$\longrightarrow$ exponential decrease with backoff factor $\delta$
$\longrightarrow$ e.g., binary backoff in case $\delta = 1/2$

Similar to Ethernet and WLAN backoff

$\longrightarrow$ question: does it work?
With $a = 1, \delta = 1/2$: 
With $a = 3$, $\delta = 1/2$: 
With $a = 1, \delta = 1/4$:  

![Graph showing Load Evolution and Lambda Evolution over time.](image-url)
With $a = 1, \delta = 3/4$: 
Note:

- Method B oscillates around target
  → does not converge
- Superior to Method A: unbounded oscillation
  → doesn’t hit “rock bottom”
  → due to asymmetry in increase vs. decrease policy
  → we observe “sawtooth” pattern
- Method B is used by TCP
  → linear increase/exponential decrease
  → additive increase/multiplicative decrease (AIMD)

Question: can we do better?

→ what information have we not made use of?
Method C:

\[ \lambda(t + 1) \leftarrow \lambda(t) + \varepsilon(Q^* - Q(t)) \]

where \( \varepsilon > 0 \) is a fixed parameter

Tries to adjust magnitude of change as a function of the gap \( Q^* - Q(t) \)

\[ \rightarrow \text{if gap is big, change by a lot} \]

\[ \rightarrow \text{if gap is small, change by a little} \]

Thus:

\[ \bullet \text{if } Q^* - Q(t) > 0, \text{ increase } \lambda(t) \text{ proportional to gap} \]

\[ \bullet \text{if } Q^* - Q(t) < 0, \text{ decrease } \lambda(t) \text{ proportional to gap} \]

Trying to be more clever…

\[ \rightarrow \text{bottom line: is it any good?} \]
With $\varepsilon = 0.1$:
With $\varepsilon = 0.5$: 
Answer: no

→ control law looks good on the surface

→ but looks can be deceiving

Time to try something new

→ any (crazy) ideas?
Method D:

\[
\lambda(t+1) \leftarrow \lambda(t) + \varepsilon(Q^* - Q(t)) - \beta(\lambda(t) - \gamma)
\]

where \( \varepsilon > 0 \) and \( \beta > 0 \) are fixed parameters

\( \rightarrow \) odd looking modification to Method C

\( \rightarrow \) additional term \(-\beta(\lambda(t) - \gamma)\)

\( \rightarrow \) what’s going on?

\( \rightarrow \) does it work?
With $\varepsilon = 0.2$ and $\beta = 0.5$: 
With $\varepsilon = 0.5$ and $\beta = 1.1$:
With $\varepsilon = 0.1$ and $\beta = 1.0$:
Remarks:

- Method D has desired behavior
- Is superior to Methods A, B, and C
- No unbounded oscillation
- In fact, dampening and convergence to desired state
  \[ \rightarrow \text{converges to target operating point } (Q^*, \gamma) \]
  \[ \rightarrow \text{called asymptotically stable} \]
  \[ \rightarrow \text{why?} \]