Congestion Control

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

$\longrightarrow$ excessive traffic can cause congestion

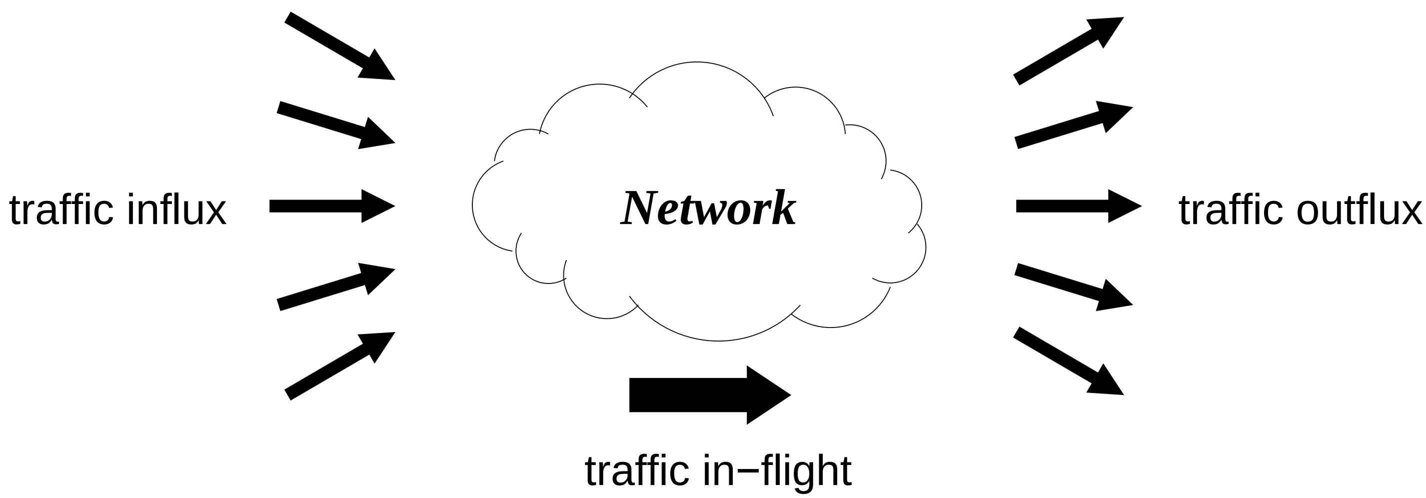
Problem: regulate traffic influx such that congestion does not occur

$\longrightarrow$ not too fast, not too slow

$\longrightarrow$ congestion control

$\longrightarrow$ first question: what is congestion?

Viewpoint: traffic coming in, in transit, going out



At time t:

- traffic influx: $\lambda(t)$ "offered load" (bps)
- traffic outflux: $\gamma(t)$ "throughput" (bps)
- traffic in-flight: $Q(t)$ "load"

  $\longrightarrow$ volume: total packets in transit (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

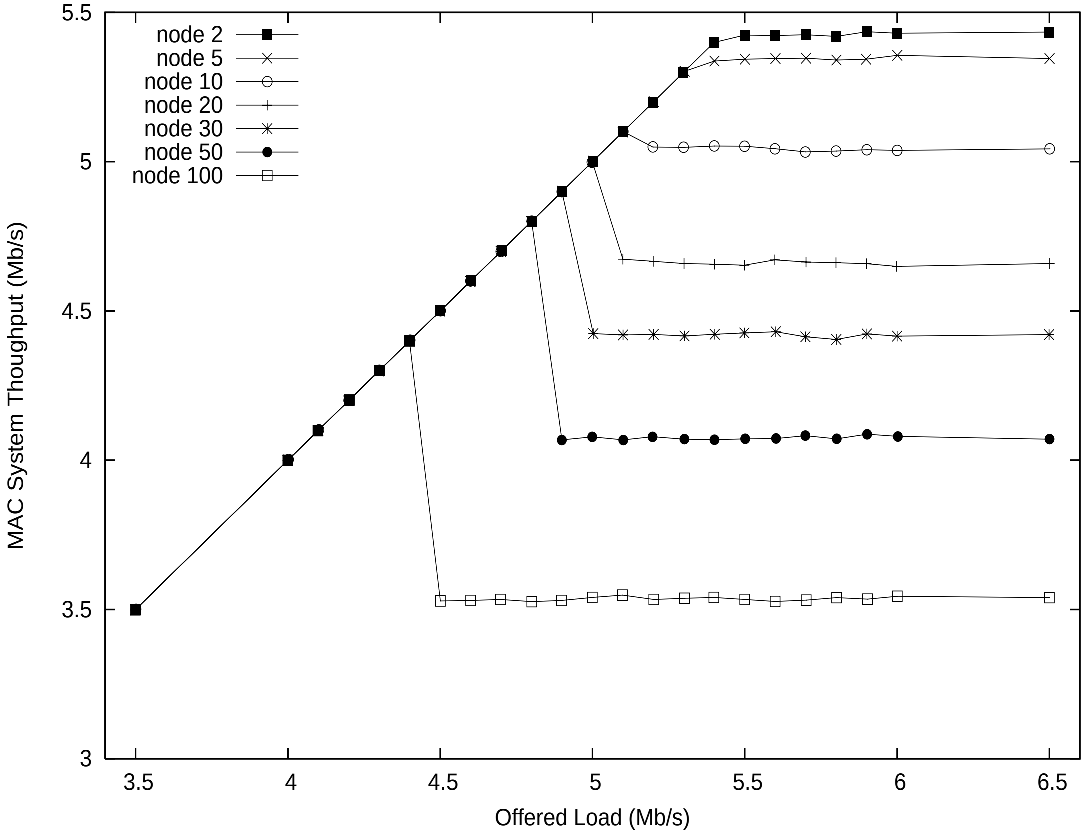
$\longrightarrow$ "congestion?"

faucet.com

Thermostat ...

802.11b WLAN:

- Throughput



$\longrightarrow$ unimodal or bell-shaped

$\longrightarrow$ recall: less pronounced in real systems

What we can control:

$\longrightarrow$ traffic influx rate $\lambda(t)$

$\longrightarrow$ no power over anything else

Ex.:

- Faucet knob in water sink
- Temperature needle in thermostat
- Cars entering onto highway: traffic light
- Packets entering the Internet

  $\rightarrow$ from web server, P2P server, PC, laptop/handheld

How does in-flight traffic or load $Q(t)$ vary?

$\longrightarrow$ 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 ($+$ or $-$)
- $Q(t)$ cannot be negative: no. of packets

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

Pseudo Real-Time Multimedia Streaming

⟶ e.g., RealPlayer, iTunes, Internet radio

⟶ "pseudo" because of prefetching trick

⟶ application is given headstart: few seconds

⟶ fill buffer & prevent from becoming empty

Steps involved:

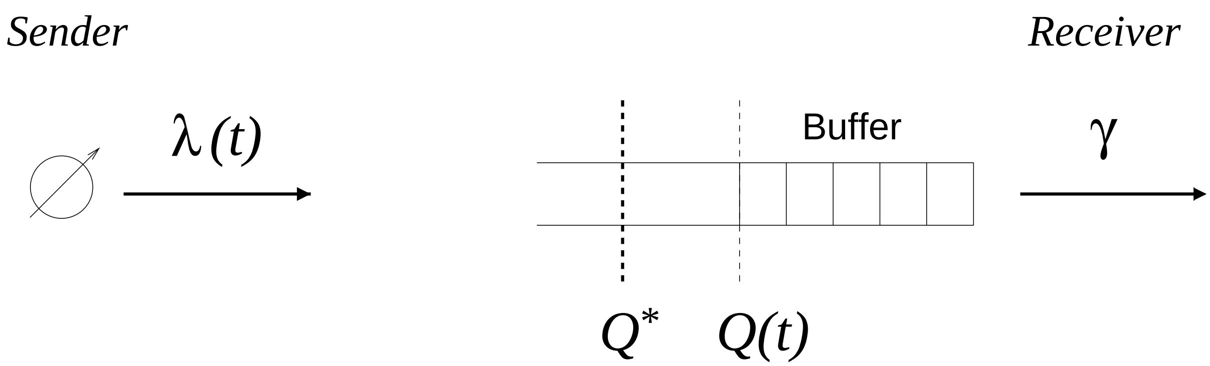
- prefetch X seconds worth of audio/video data
- causes initial delayed playback

  → e.g., couple of seconds delay after click
- keep fetching audio/video data such that X seconds worth of future data resides in receiver's buffer

  → hides spurious congestion

  → user: continuous playback experience

Pseudo real-time application architecture:



- $Q(t)$: current buffer level
- Q^*: desired buffer level
- γ: throughput, i.e., playback rate

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

Goal: vary $\lambda(t)$ such that $Q(t) \approx Q^*$

$\longrightarrow$ don't buffer too much: memory cost

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

Other applications:

$\longrightarrow$ pseudo real-time set-up is highly versatile

$\longrightarrow$ captures many scenarios

Ex. 1: Router congestion control

$\longrightarrow$ active queue management (AQM)

- receiver is a router/switch
- Q^* is desired buffer occupancy/delay at router
- router throttles sender(s) to maintain Q^*

$\longrightarrow$ send control packets to senders

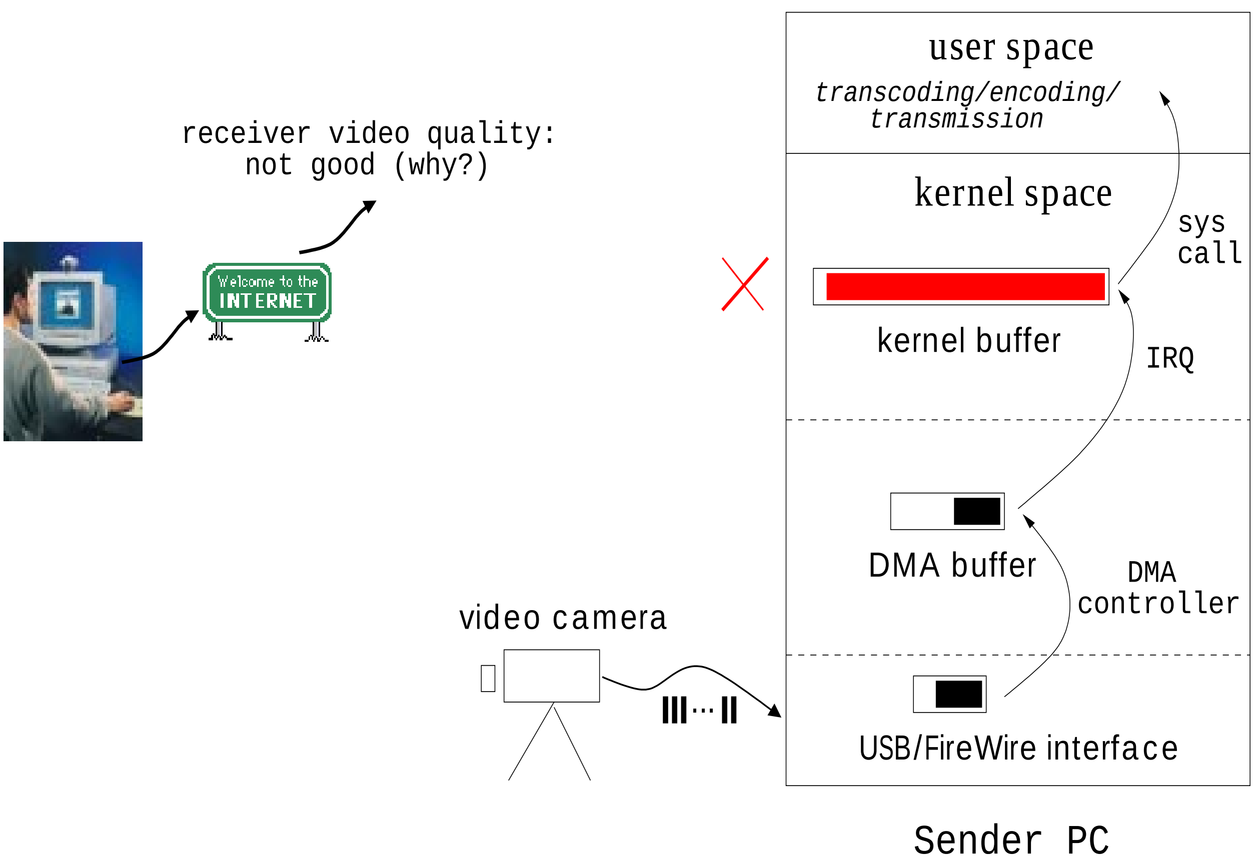
$\longrightarrow$ slow down, go faster, stay put

Ex. 2: Desktop videoconferencing

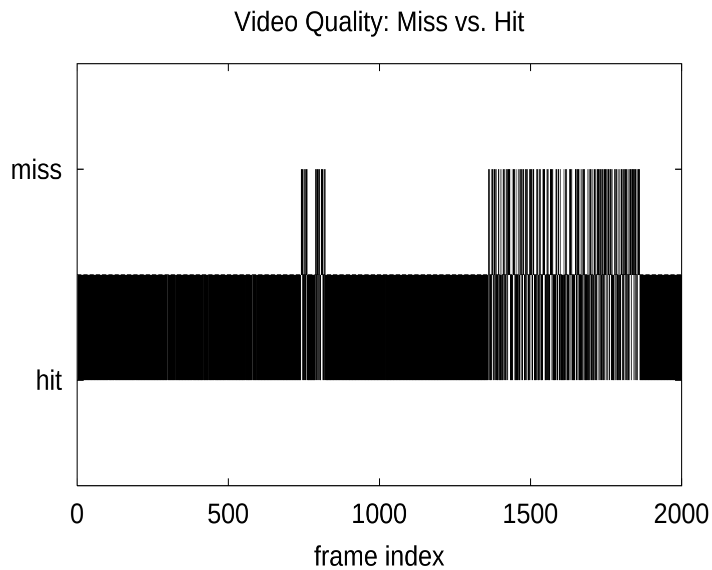
⟶ e.g., AOL, MSN, Skype, Yahoo

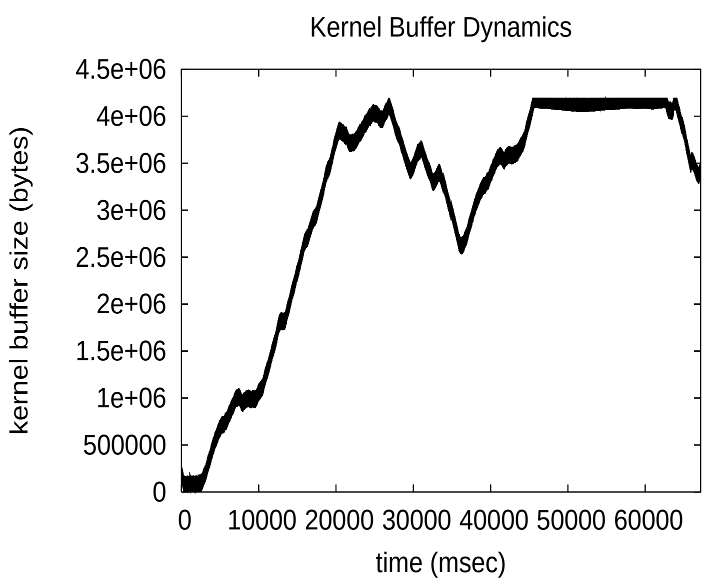
⟶ video quality is not good: why?

⟶ misconception: network is blamed



Performance consequences:





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

$\longrightarrow$ producer/consumer rate mismatch problem

$\longrightarrow$ also called "flow control"

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 $Q(t) = Q^*$

How to: basic idea

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

$\longrightarrow$ a rule of thumb

$\longrightarrow$ called "control law"

Network protocol implementation:

$\longrightarrow$ some design options available

- control action undertaken at sender

  $\rightarrow$ smart sender/dump receiver

  $\rightarrow$ preferred mode in many Internet protocols

  $\rightarrow$ when might the opposite be better?

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

  $\rightarrow$ feedback packet ("control signaling")

  $\rightarrow$ or simply $+/-$ indication (binary)

  $\rightarrow$ or actual gap $Q^* - Q(t)$

Receiver sends feedback to sender; sender takes action

$\longrightarrow$ called feedback control

$\longrightarrow$ or closed-loop control

Key question in feedback congestion control:

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

$\longrightarrow$ we already know in which direction

Desired state of the system:

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

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

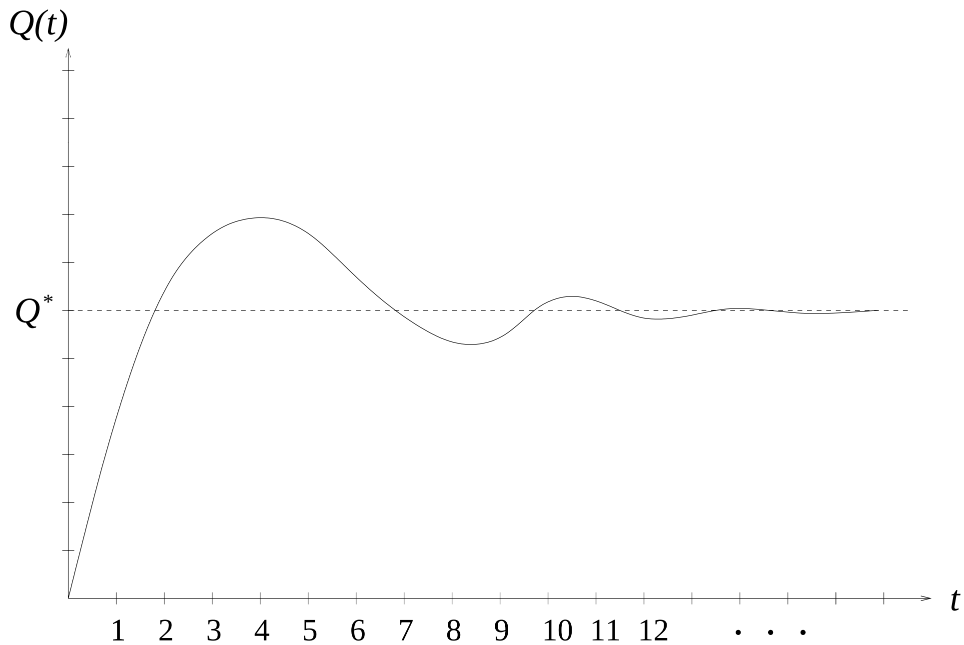
Starting state:

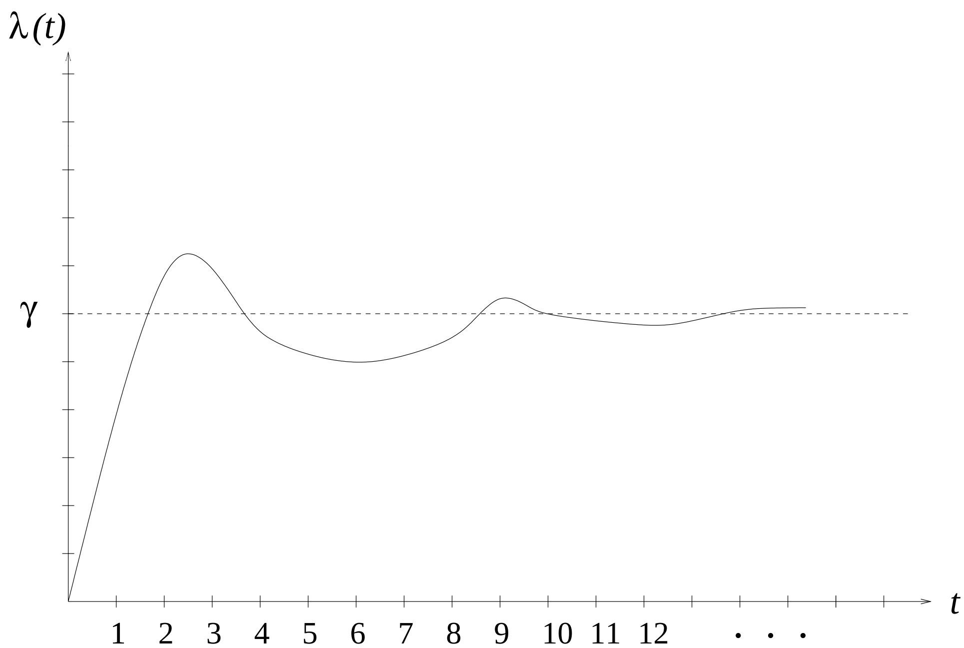
$\longrightarrow$ empty buffer and nothing is being sent

$\longrightarrow$ think of iTunes, Rhapsody, etc.

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

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 $\lambda(t+1) \leftarrow \lambda(t) - a$

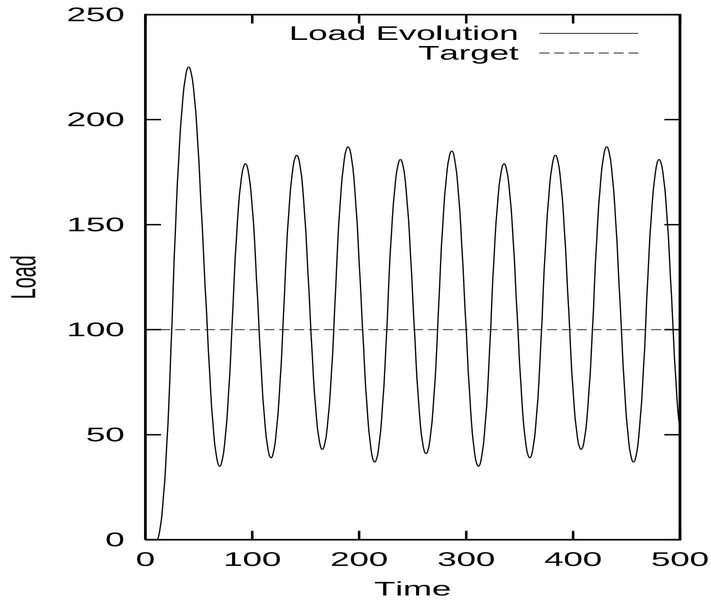
where $a > 0$ is a fixed parameter

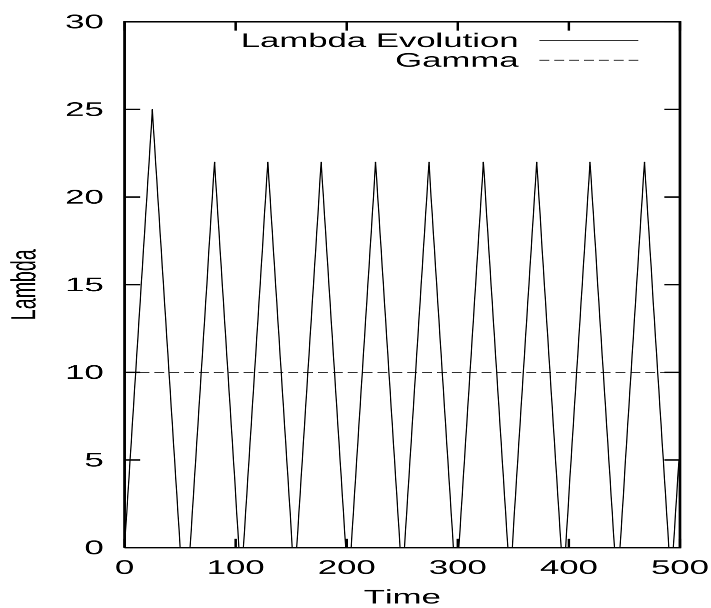
$\longrightarrow$ called linear increase/linear decrease

Question: how well does it work?

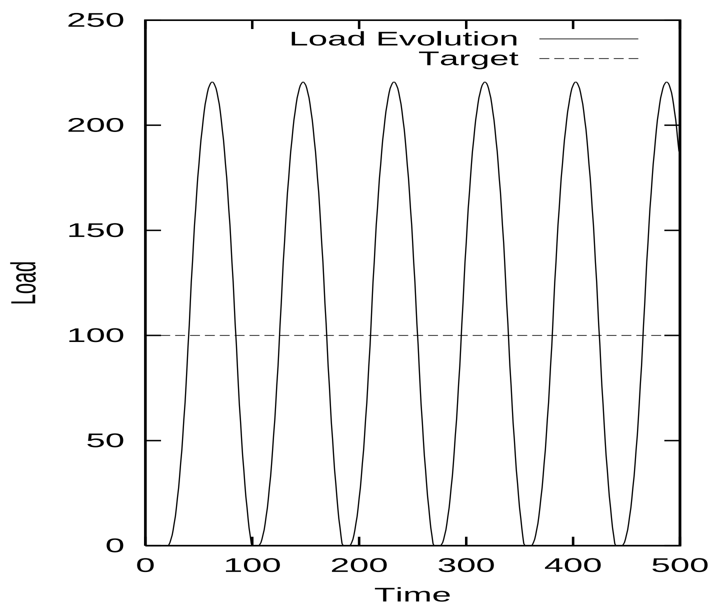
Example:

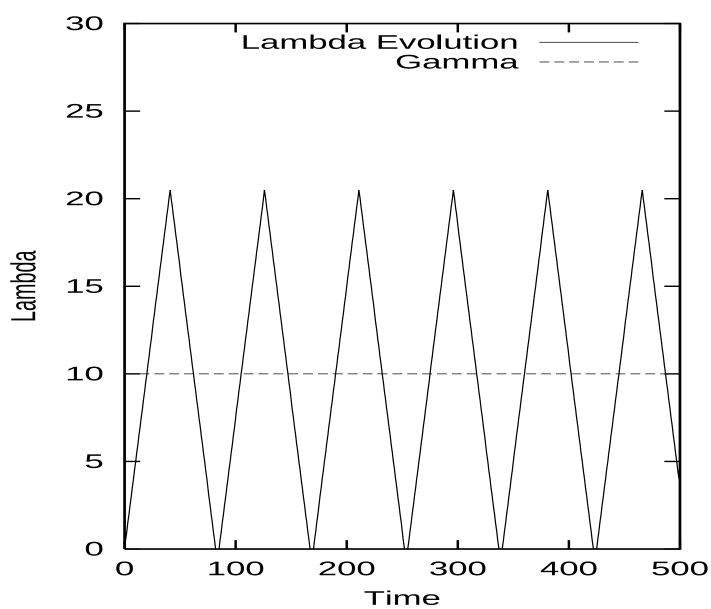
- $Q(0) = 0$
- $\lambda(0) = 0$
- $Q^* = 100$
- $\gamma = 10$
- $a = 1$


Load Evolution
Target
Load
Time
0
50
100
150
200
250
0
100
200
300
400
500

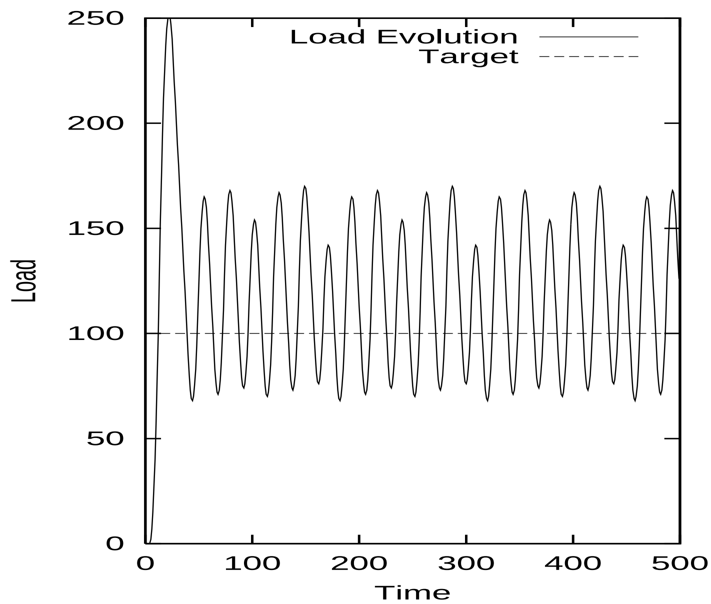

Lambda Evolution
Gamma
Lambda
Time
0
5
10
15
20
25
30
0
100
200
300
400
500

With $a = 0.5$:

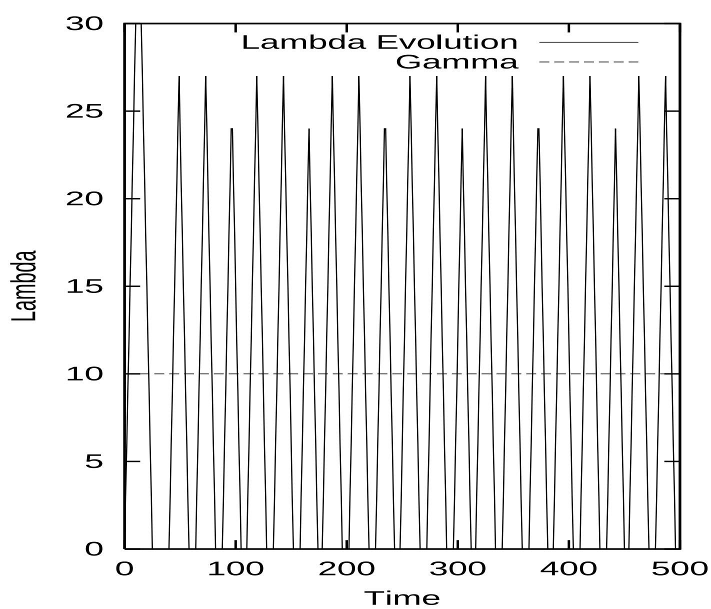

Load Evolution
Target
Load
Time
250
200
150
100
50
0
0
100
200
300
400
500


Lambda Evolution
Gamma
Lambda
Time
30
25
20
15
10
5
0
0
100
200
300
400
500

With $a = 3$:

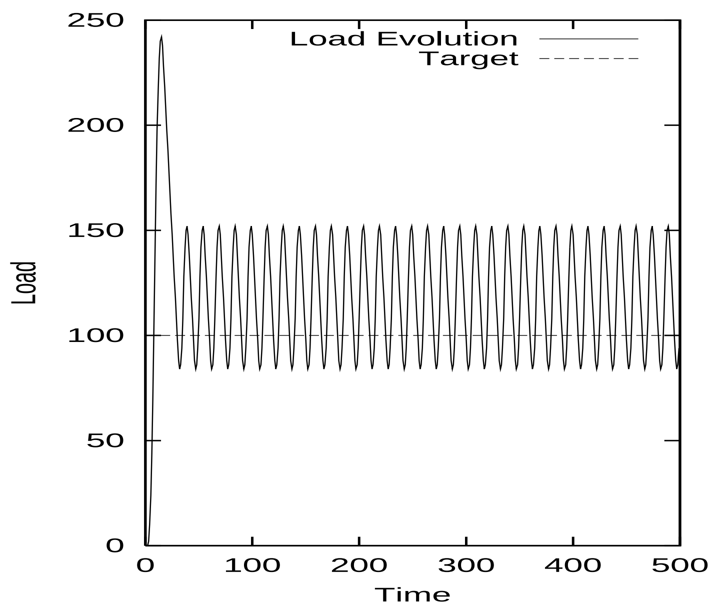


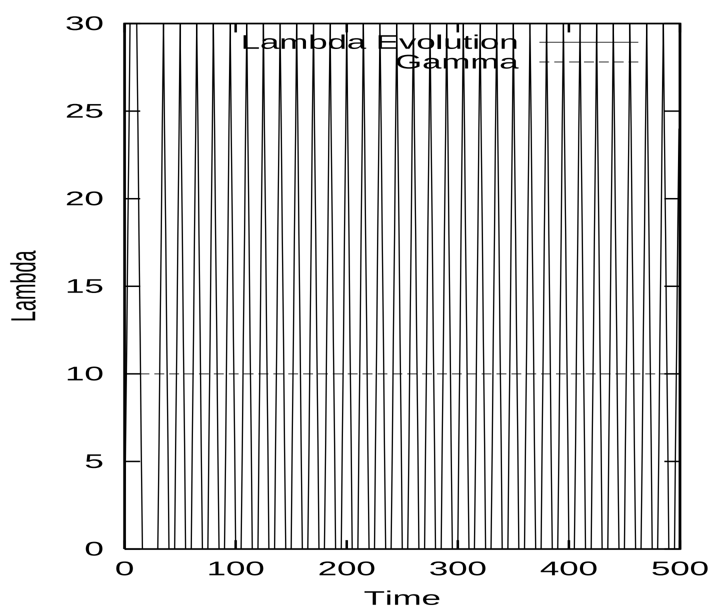
Load Evolution
Target
Load
Time
0
50
100
150
200
250
0
100
200
300
400
500



Lambda Evolution
Gamma
Lambda
Time
0
5
10
15
20
25
30
0
100
200
300
400
500

With $a = 6$:


Load Evolution
Target
Load
Time
0
50
100
150
200
250
0
100
200
300
400
500


Lambda Evolution
Gamma
Lambda
Time
0
5
10
15
20
25
30
0
100
200
300
400
500

Remarks:

- Method A isn't that great no matter what a value is used

  $\rightarrow$ keeps oscillating

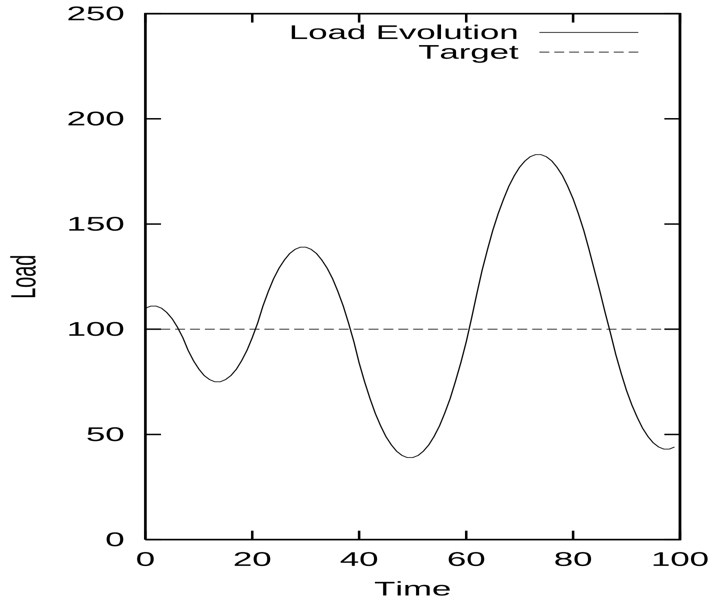
- Actually: would lead to unbounded oscillation if not for physical restriction $\lambda(t) \geq 0$ and $Q(t) \geq 0$

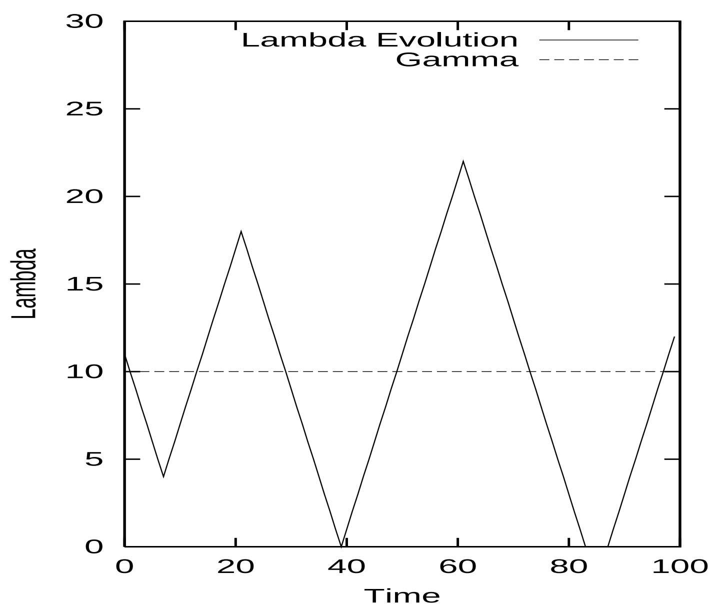
  $\longrightarrow$ i.e., bottoms out

  $\longrightarrow$ easily seen: start from non-zero buffer

  $\longrightarrow$ 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 $\lambda(t+1) \leftarrow \delta \cdot \lambda(t)$

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

Note: only decrease part differs from `Method A`.

$\longrightarrow$ linear increase with slope a

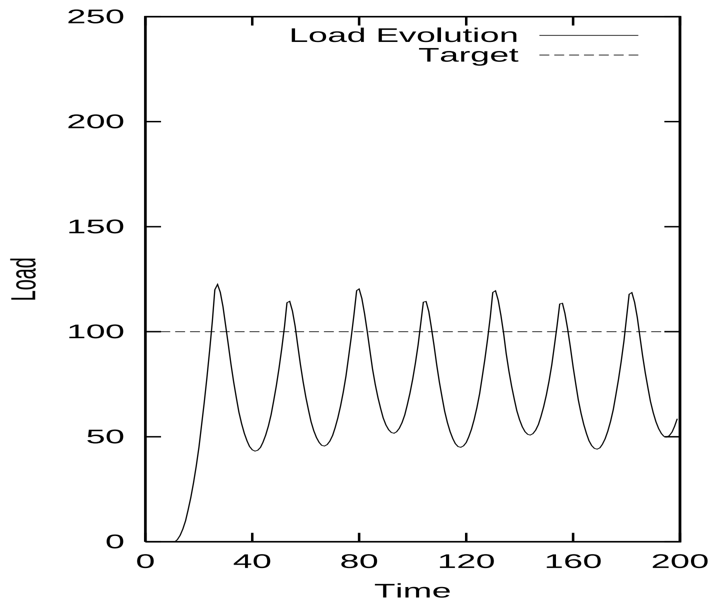
$\longrightarrow$ exponential decrease with backoff factor δ

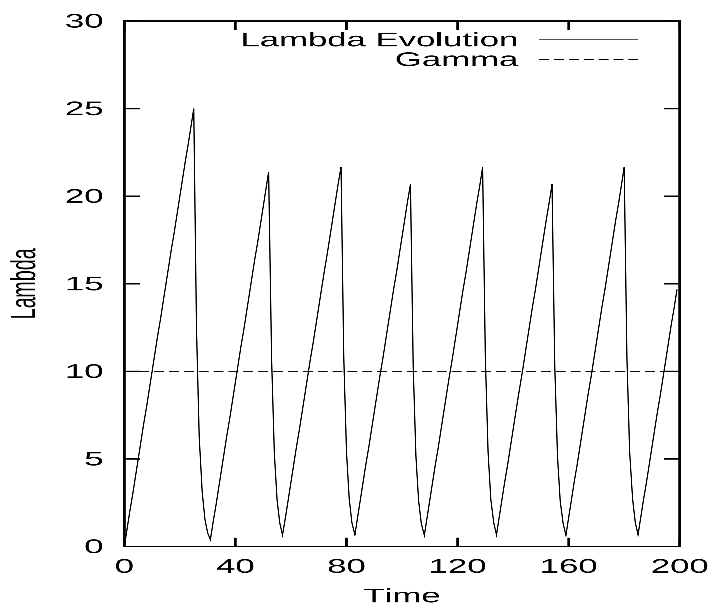
$\longrightarrow$ e.g., binary backoff in case $\delta = 1/2$

Similar to Ethernet and WLAN backoff

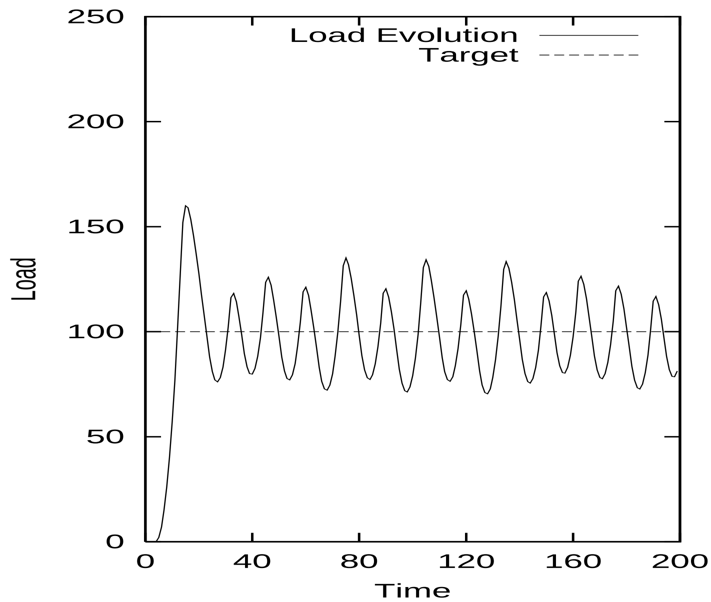
$\longrightarrow$ question: does it work?

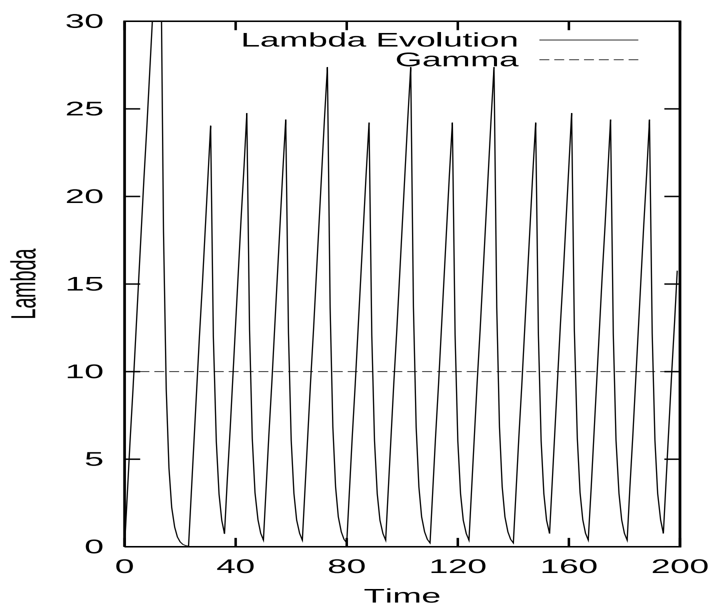
With $a = 1$, $\delta = 1/2$:



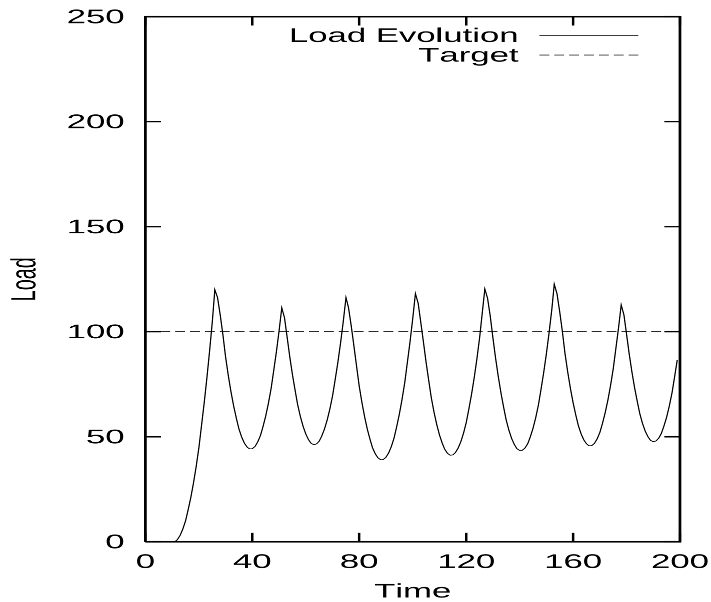


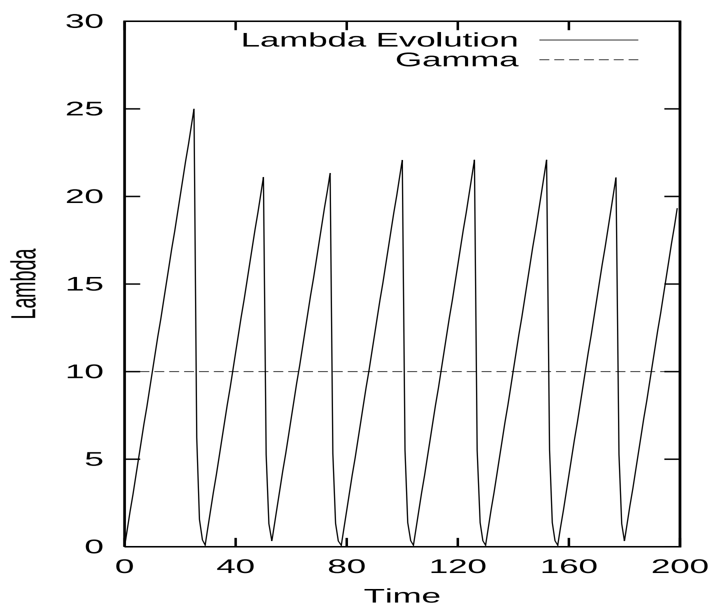
With $a = 3$, $\delta = 1/2$:



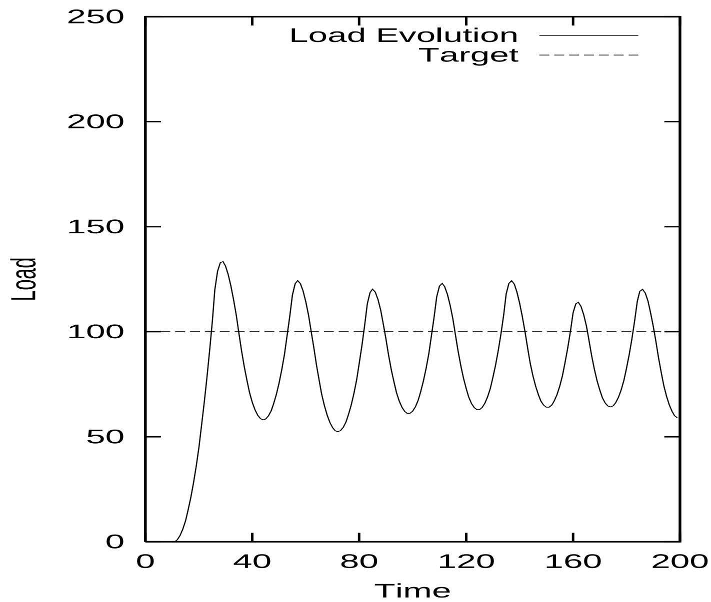


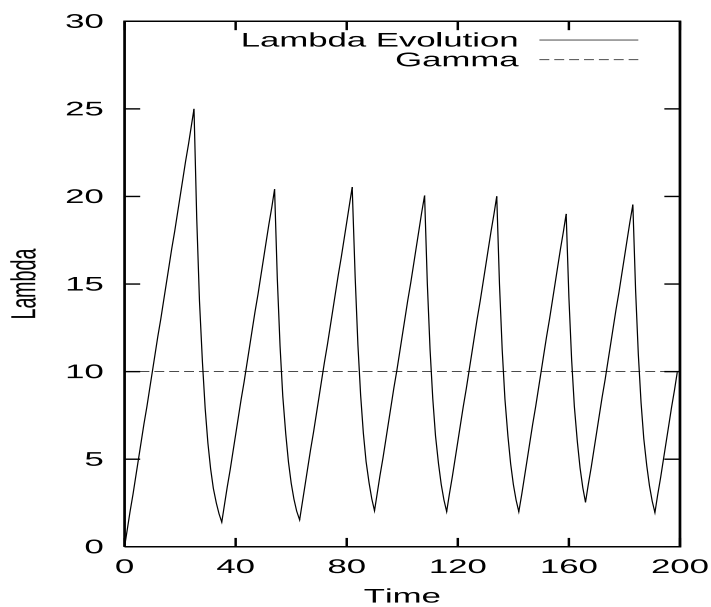
With $a = 1$, $\delta = 1/4$:





With $a = 1$, $\delta = 3/4$:





Note:

- Method B isn't that great either
- One advantage over Method A: doesn't lead to unbounded oscillation

  $\rightarrow$ note: doesn't hit "rock bottom"

  $\rightarrow$ due to asymmetry in increase vs. decrease policy

  $\rightarrow$ we observe "sawtooth" pattern
- Method B is used by TCP

  $\rightarrow$ linear increase/exponential decrease

  $\rightarrow$ additive increase/multiplicative decrease (AIMD)

Question: can we do better?

$\longrightarrow$ what "freebie" 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 $Q^* - Q(t)$

$\longrightarrow$ incorporate distance from target Q^*

$\longrightarrow$ before: just the sign (above/below)

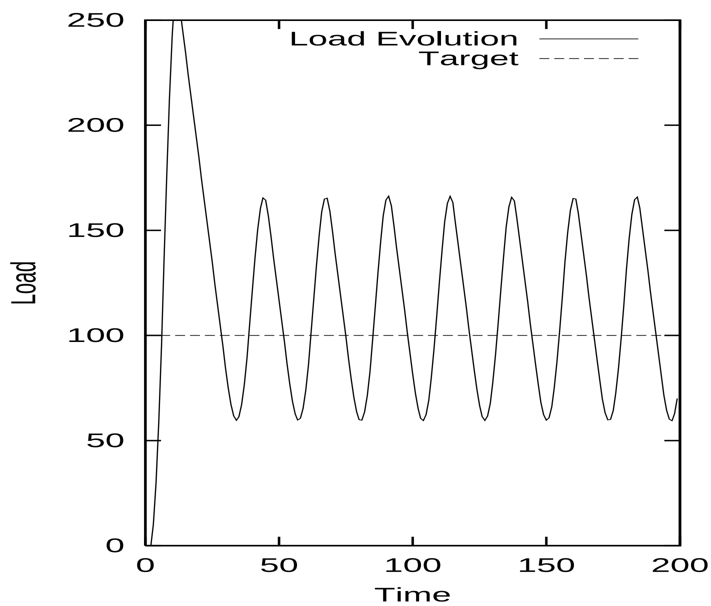
Thus:

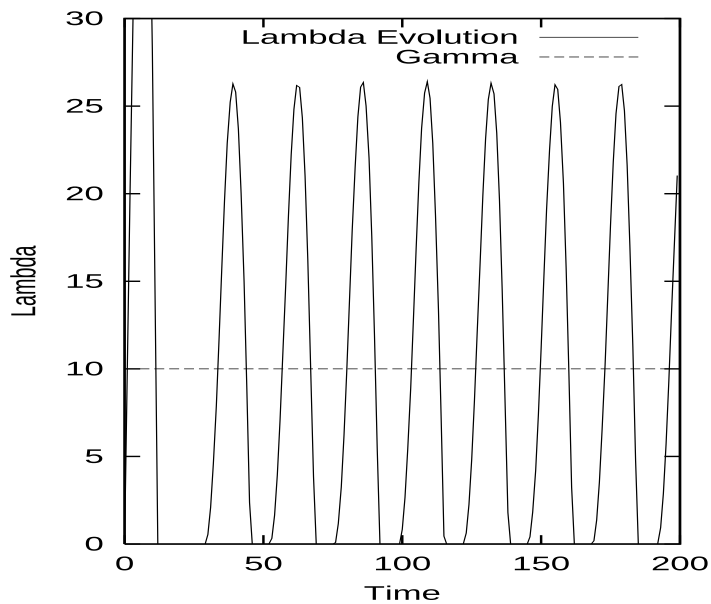
- if $Q^* - Q(t) > 0$, increase $\lambda(t)$ proportional to gap
- if $Q^* - Q(t) < 0$, decrease $\lambda(t)$ proportional to gap

Trying to be more clever…

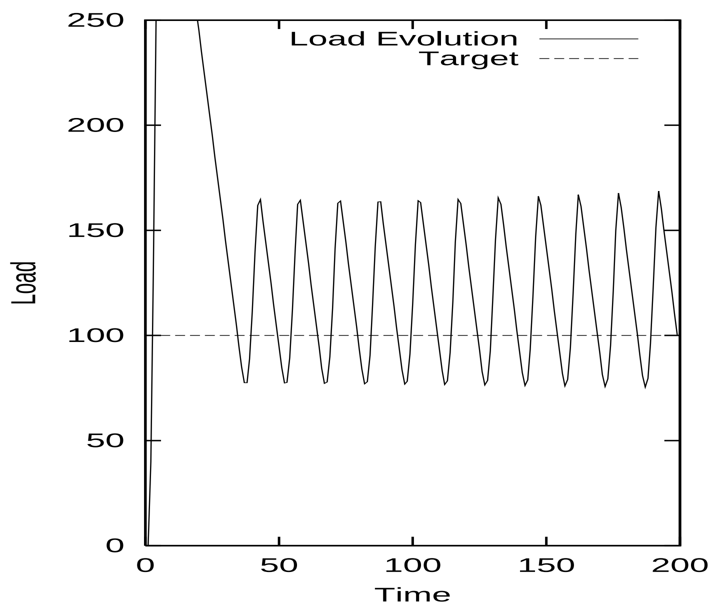
$\longrightarrow$ bottom line: is it any good?

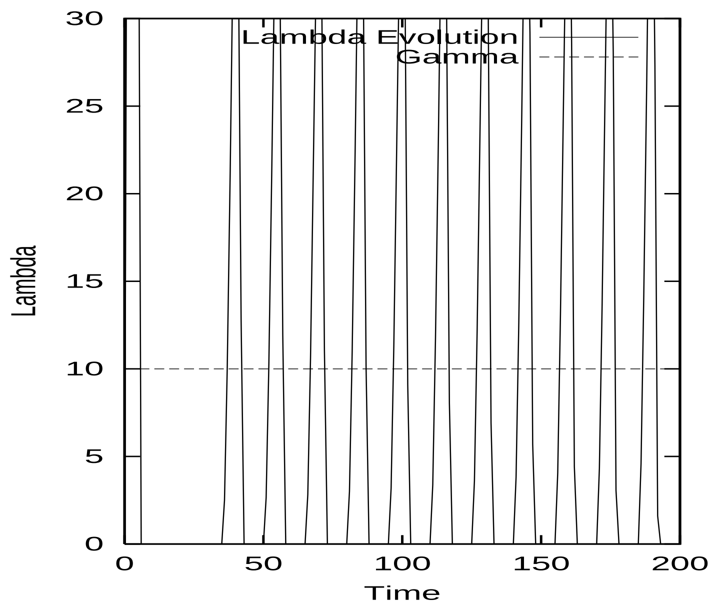
With $\varepsilon = 0.1$:





With $\varepsilon = 0.5$:





Answer: no

- $\longrightarrow$ looks good
- $\longrightarrow$ but looks can be deceiving

Time to try something strange

- $\longrightarrow$ any (crazy) ideas?
- $\longrightarrow$ good for course project

`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

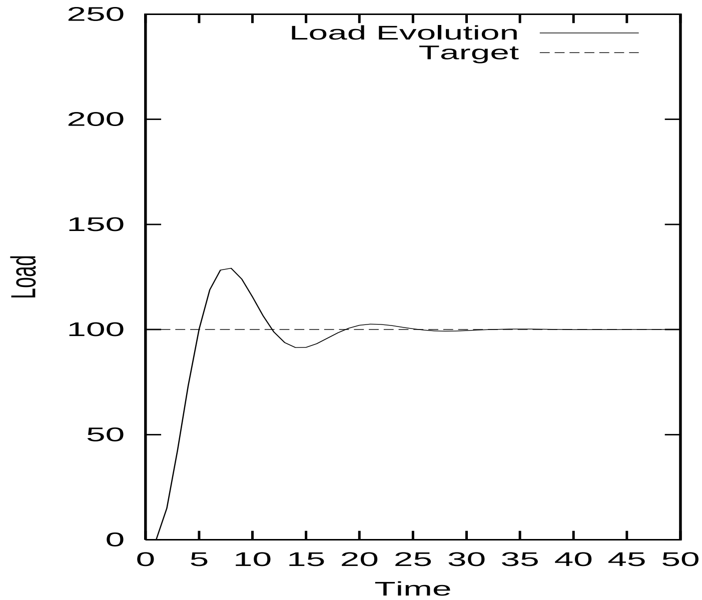
$\longrightarrow$ odd looking modification to `Method C`

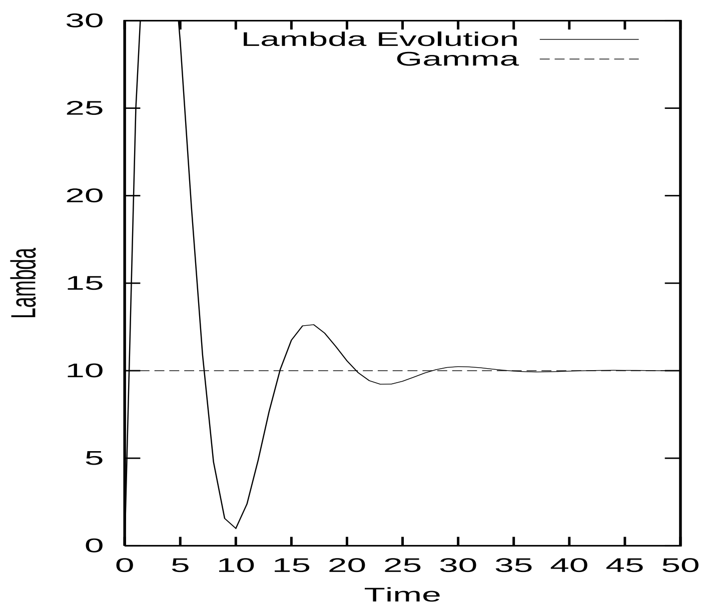
$\longrightarrow$ additional term $-\beta(\lambda(t) - \gamma)$

$\longrightarrow$ what's going on?

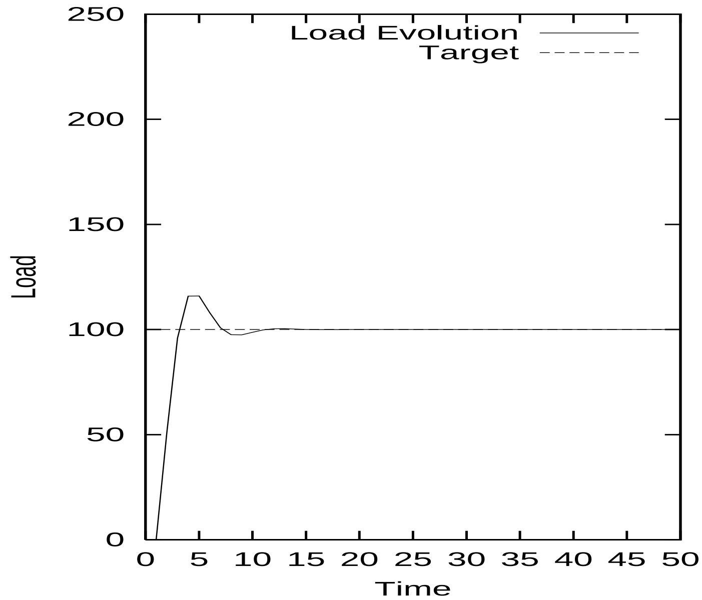
$\longrightarrow$ does it work?

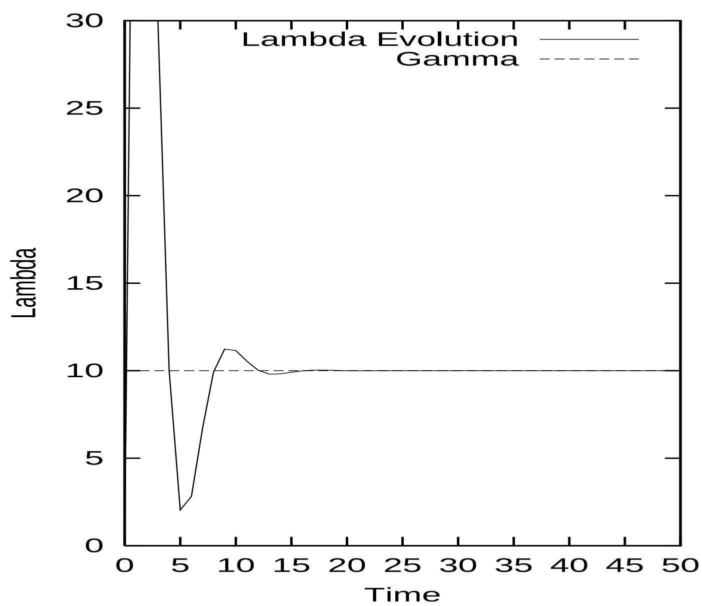
With $\varepsilon = 0.2$ and $\beta = 0.5$:



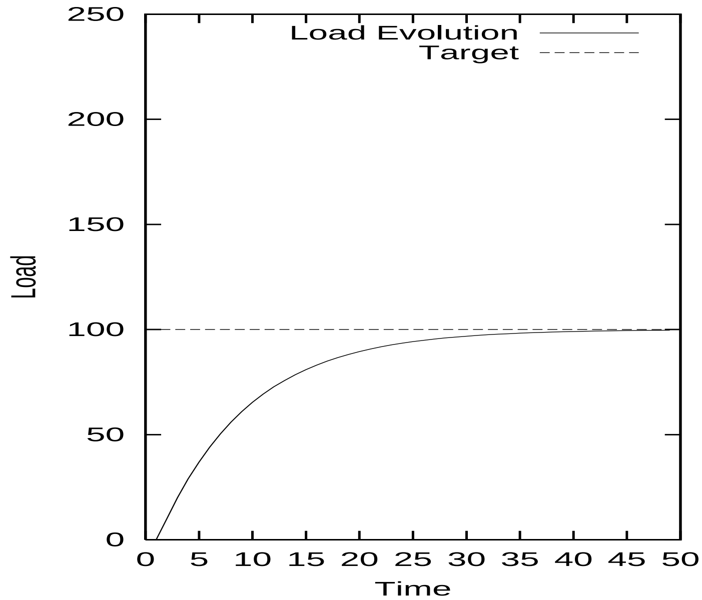


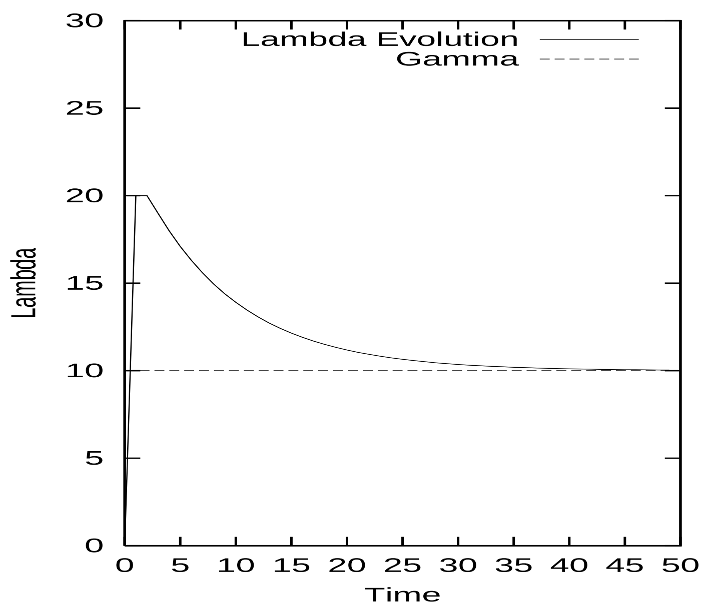
With $\varepsilon = 0.5$ and $\beta = 1.1$:





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

  $\longrightarrow$ converges to target operating point (Q^*, γ)

  $\longrightarrow$ called asymptotically stable

  $\longrightarrow$ why?