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

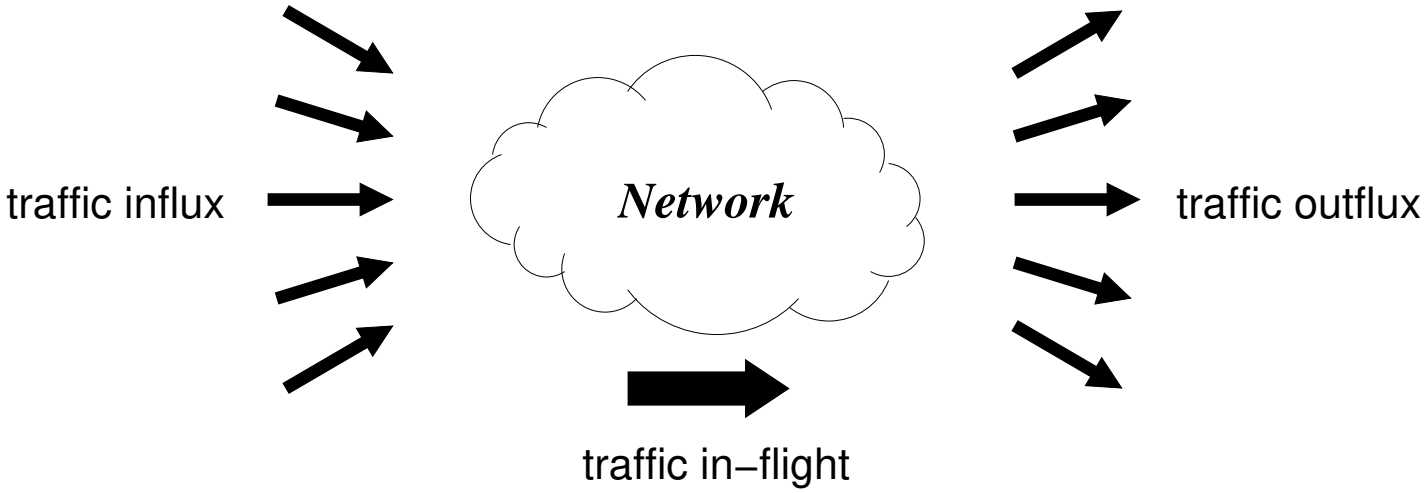
$\longrightarrow$ not too fast, not too slow

$\longrightarrow$ congestion control

$\longrightarrow$ first question: what is congestion?

Viewpoint: 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, car cruise control, ...

What is the meaning of congestion?

$\rightarrow$ when sending too fast, throughput starts to go down

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

What about highway system?

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

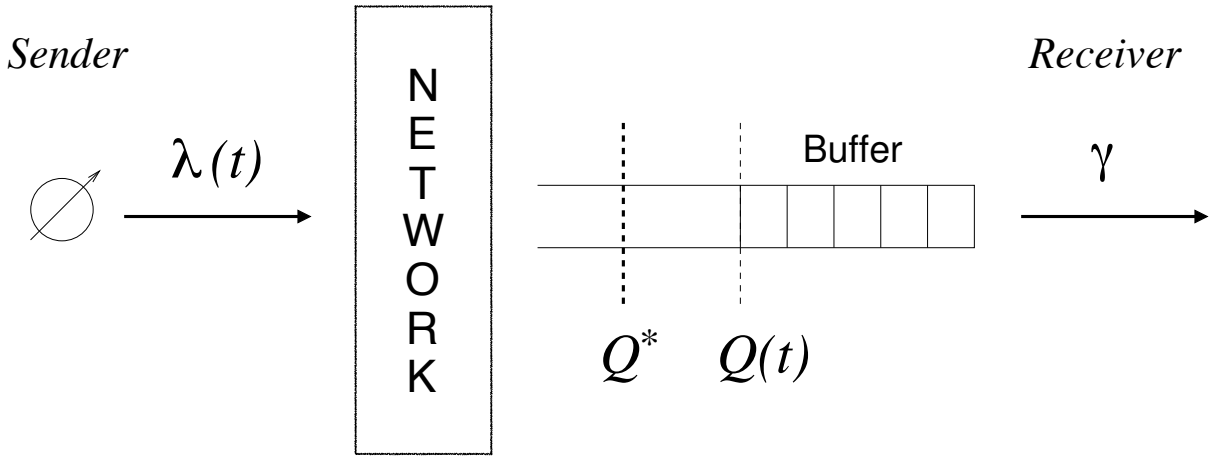
  $\rightarrow$ protects against spurious congestion

  $\rightarrow$ don't keep more than X

  $\rightarrow$ potential waste of 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
- router throttles sender(s) to maintain Q^*

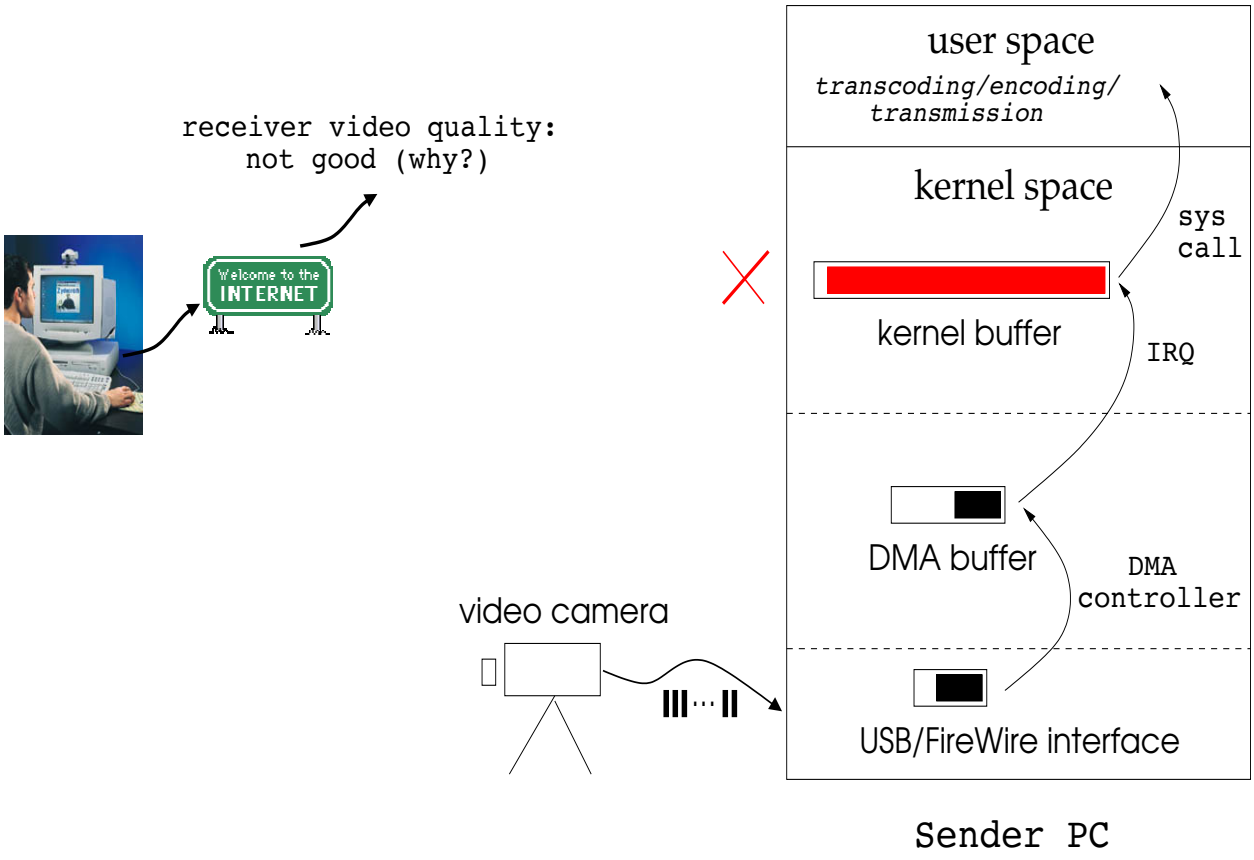
  $\rightarrow$ router sends control packets to senders

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

Ex. 2: Desktop videoconferencing

→ video quality may not be good: why?

→ common misconception: sole culprit is network



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:

$\rightarrow$ design choices

- control action undertaken at sender

  $\rightarrow$ smart sender/dump receiver

  $\rightarrow$ preferred mode of Internet protocols

  $\rightarrow$ when might the opposite be better?

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

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

  $\rightarrow$ or simply up/down binary indication

Key question in feedback congestion control:

$\longrightarrow$ how 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

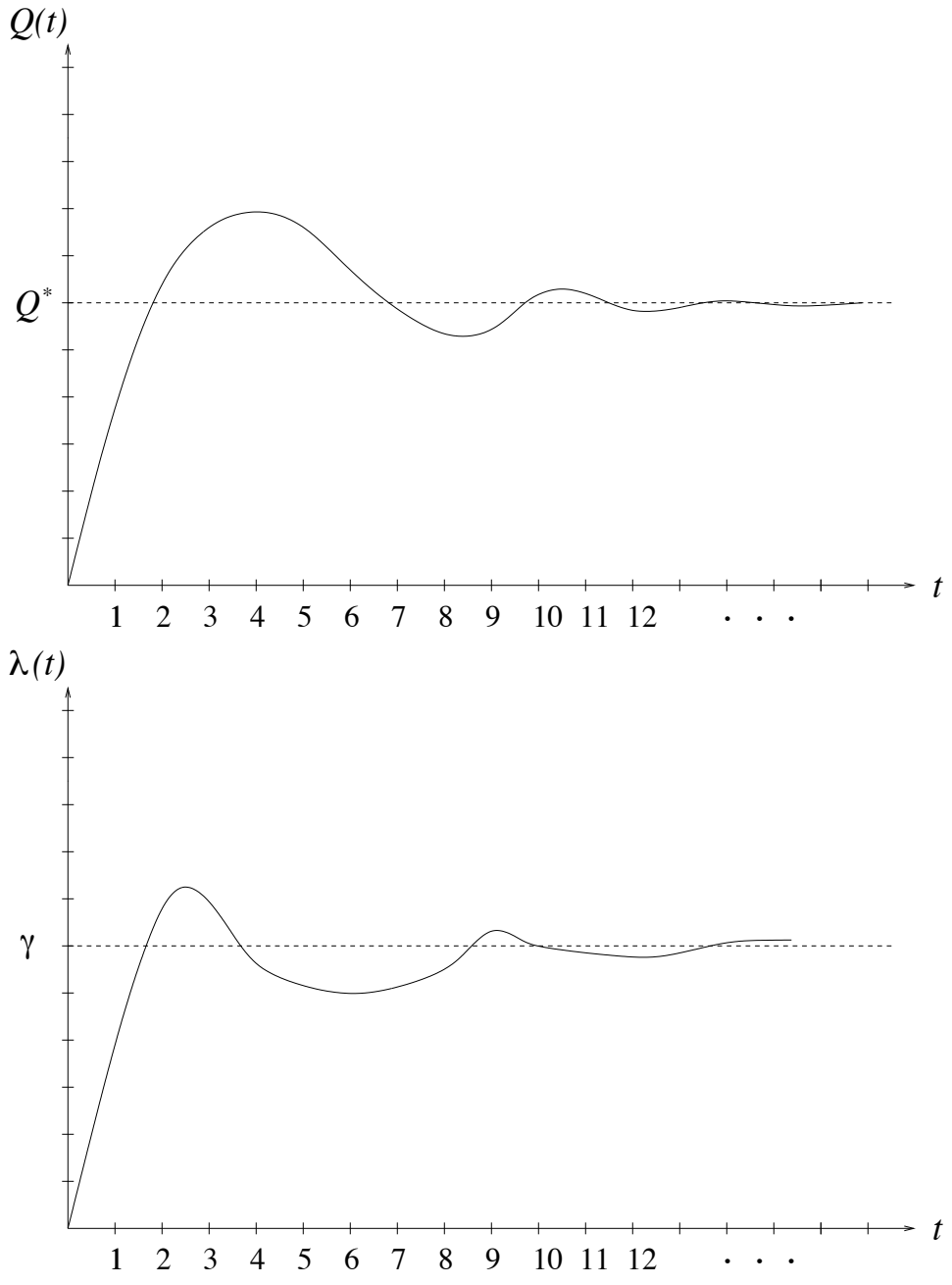
Starting state:

$\longrightarrow$ 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 $\lambda(t)$



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

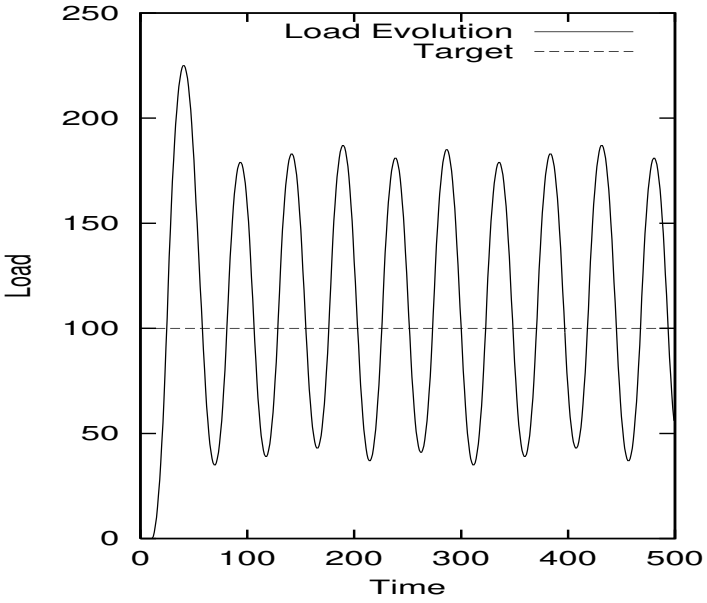
where $a > 0$ is a fixed parameter

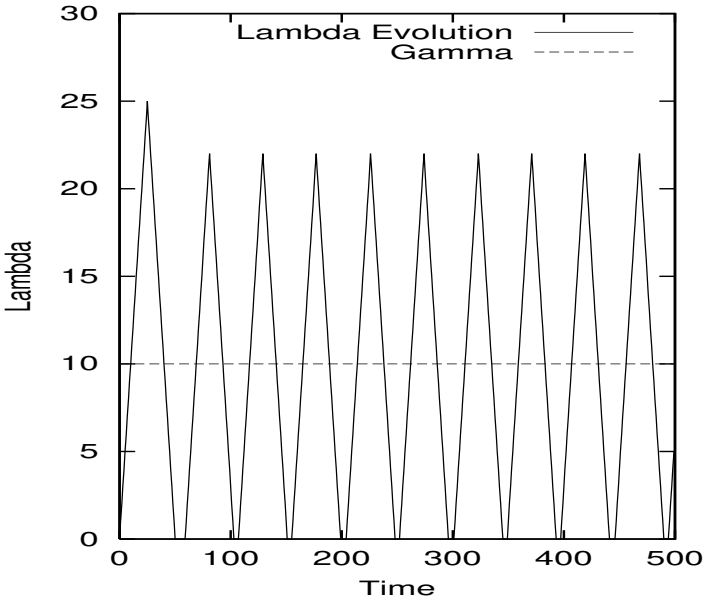
$\longrightarrow$ called linear increase/linear decrease

Question: how well does it work?

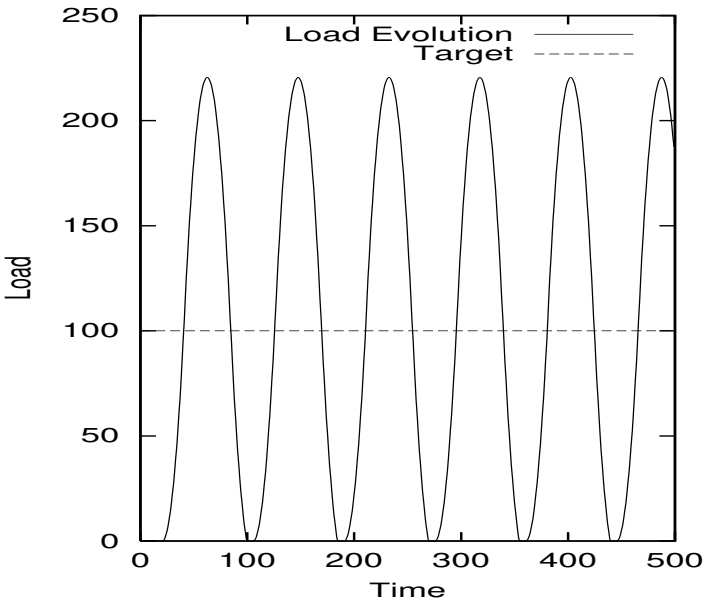
Example:

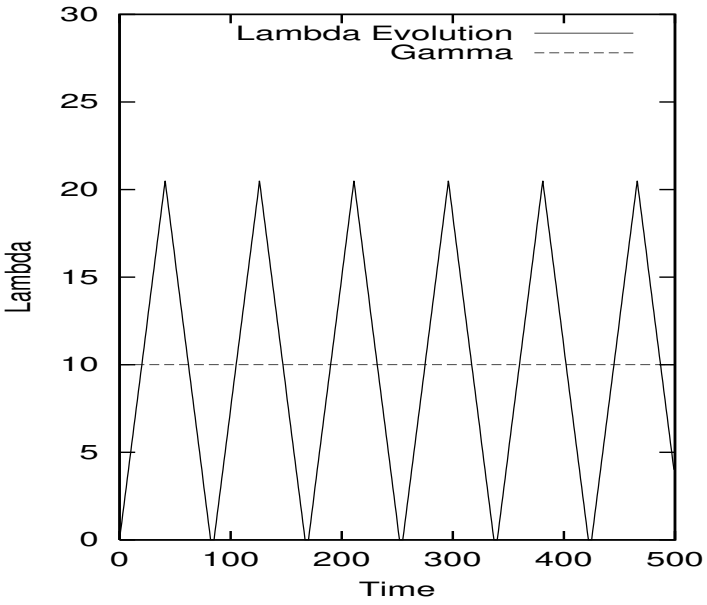
- $Q(0) = 0$
- $\lambda(0) = 0$
- $Q^* = 100$
- $\gamma = 10$
- $a = 1$


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

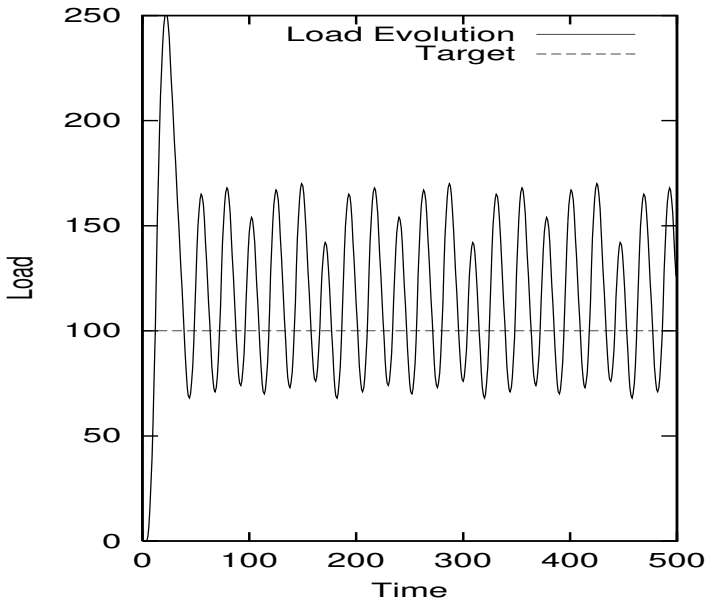

Lambda Evolution
Gamma
Lambda
Time
0
5
10
15
20
25
30
0
100
200
300
400
500

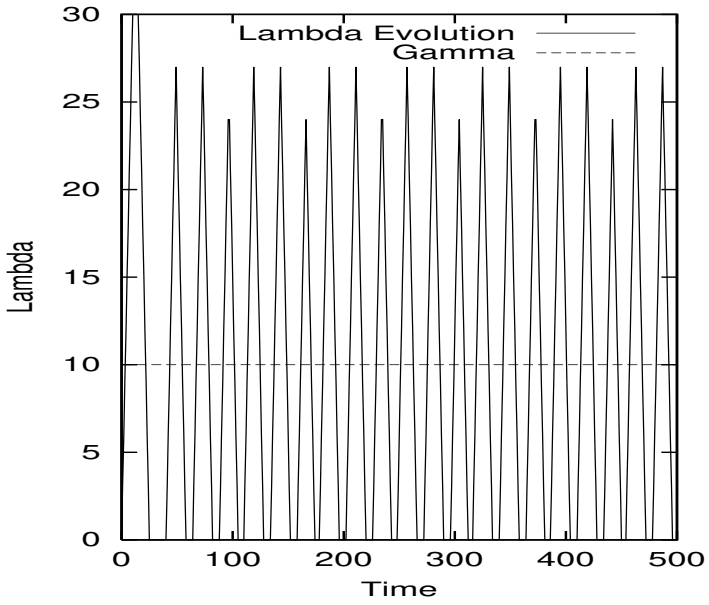
With $a = 0.5$:


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
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

Remarks:

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

  $\rightarrow$ keeps oscillating

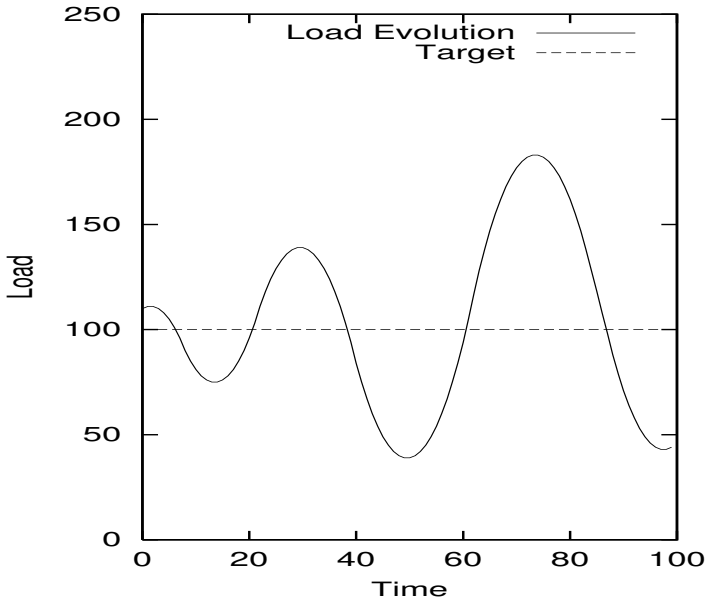
- Actually: would lead to unbounded oscillation if not for physical restriction $\lambda(t) \geq 0$ and $Q(t) \geq 0$

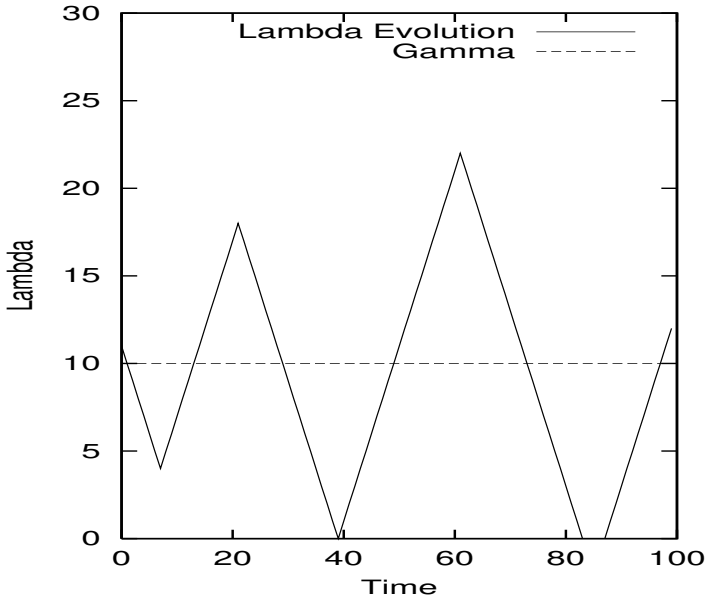
  $\longrightarrow$ i.e., bottoms out

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

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

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

$\longrightarrow$ linear increase with slope a

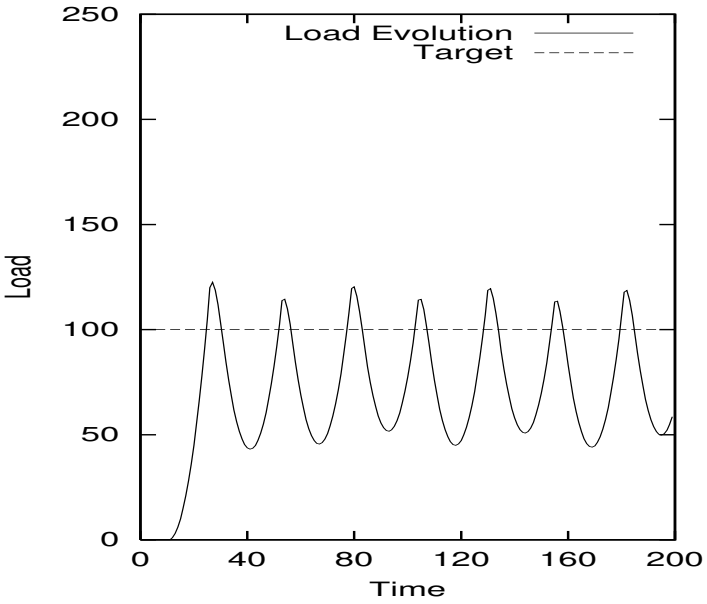
$\longrightarrow$ exponential decrease with backoff factor δ

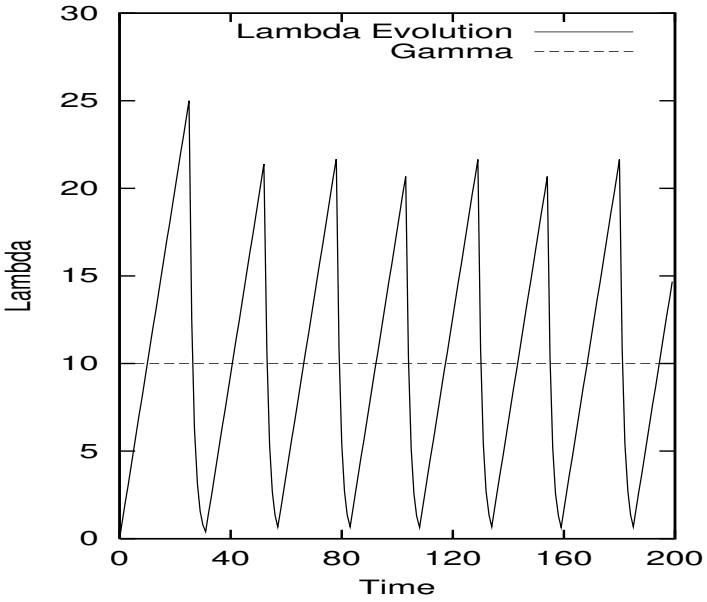
$\longrightarrow$ e.g., binary backoff in case $\delta = 1/2$

Similar to Ethernet and WLAN backoff

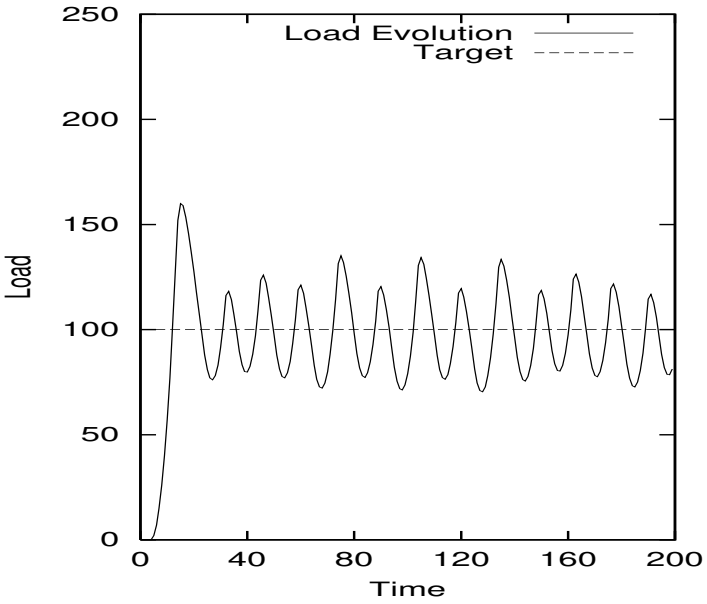
$\longrightarrow$ question: does it work?

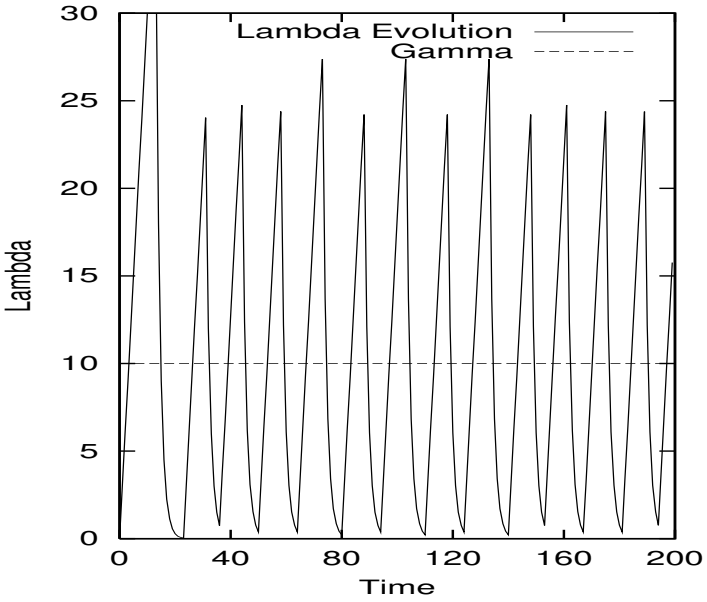
With $a = 1$, $\delta = 1/2$:



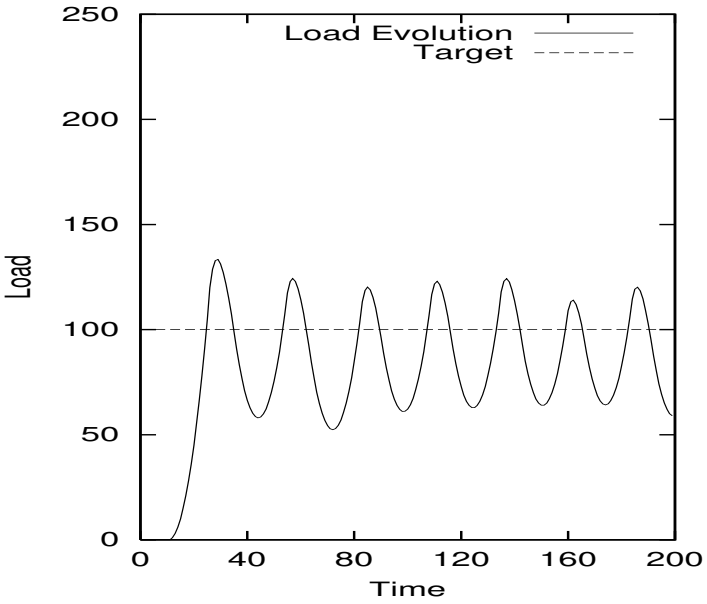


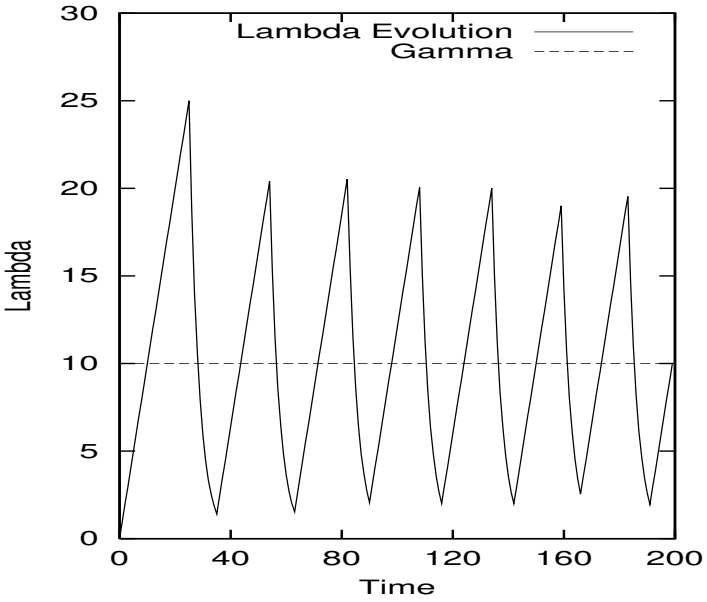
With $a = 3$, $\delta = 1/2$:





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





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)

  → will discuss separately under TCP congestion control

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 $Q^* - Q(t)$

$\rightarrow$ if gap is big, change by a lot

$\rightarrow$ if gap is small, change by a little

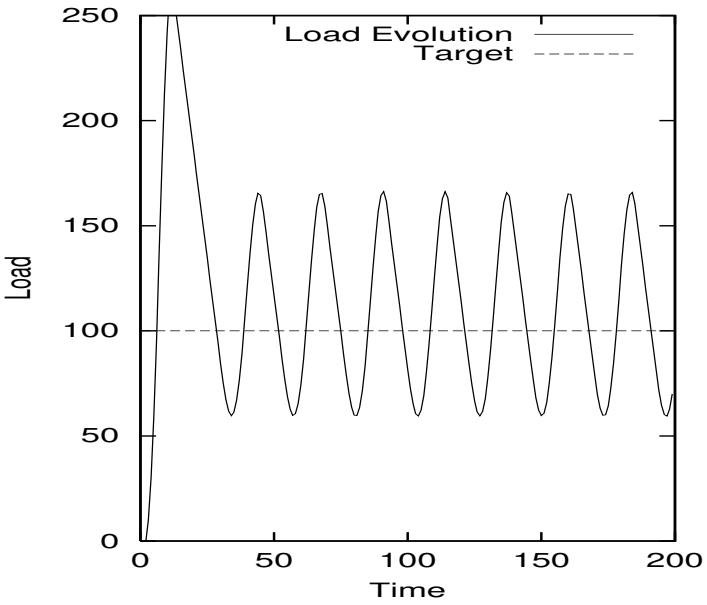
Thus:

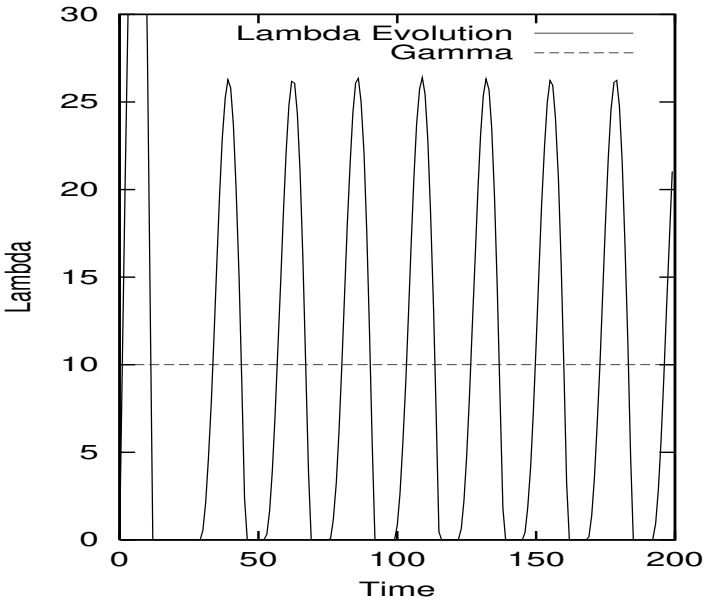
- if $Q^* - Q(t) > 0$, increase $\lambda(t)$ proportional to gap
- if $Q^* - Q(t) < 0$, decrease $\lambda(t)$ proportional to gap

Trying to be more clever...

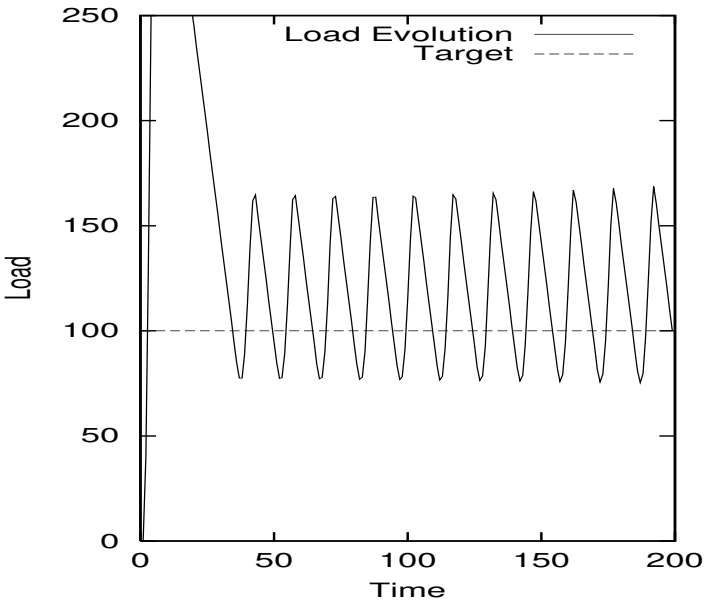
$\longrightarrow$ bottom line: is it any good?

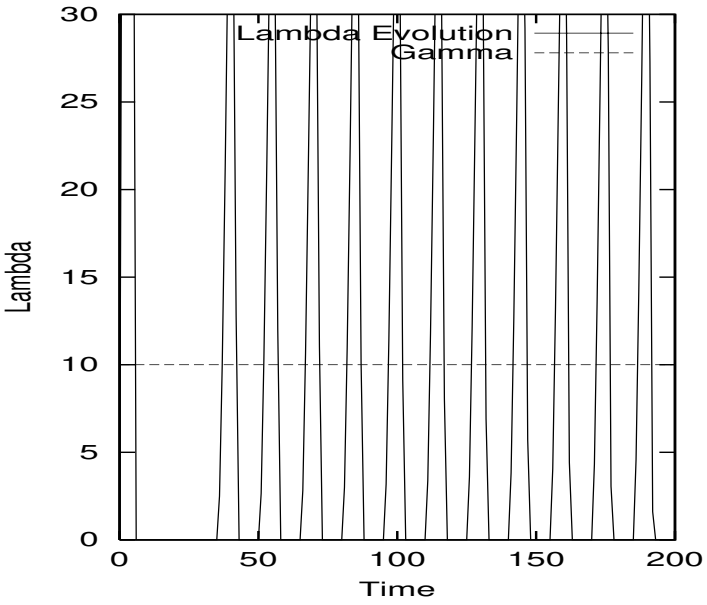
With $\varepsilon = 0.1$:





With $\varepsilon = 0.5$:





Answer: no

$\rightarrow$ control law looks good on the surface

$\rightarrow$ but looks can be deceiving

Time to try something different.

`Method D`:

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

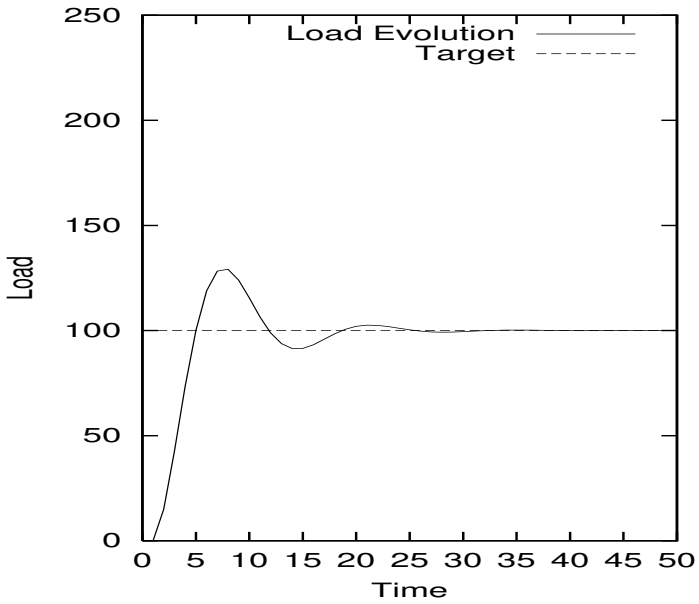
where $\varepsilon > 0$ and $\beta > 0$ are fixed parameters

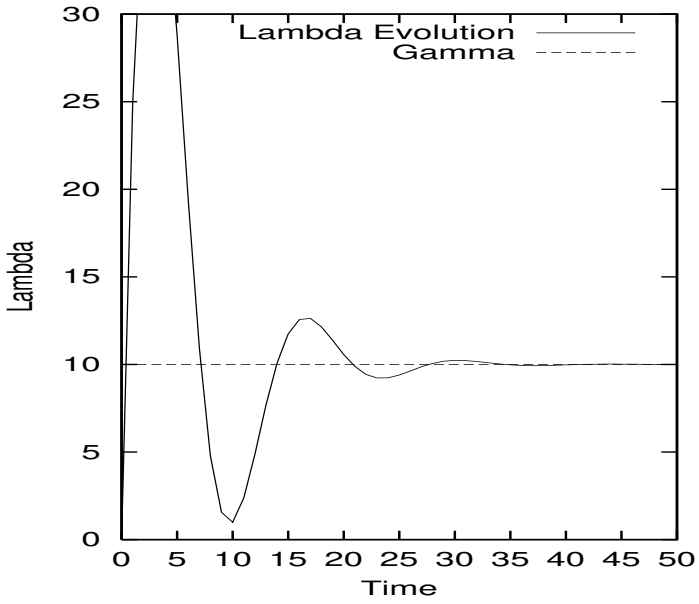
$\longrightarrow$ additional term $\beta(\gamma - \lambda(t))$

$\longrightarrow$ what's going on?

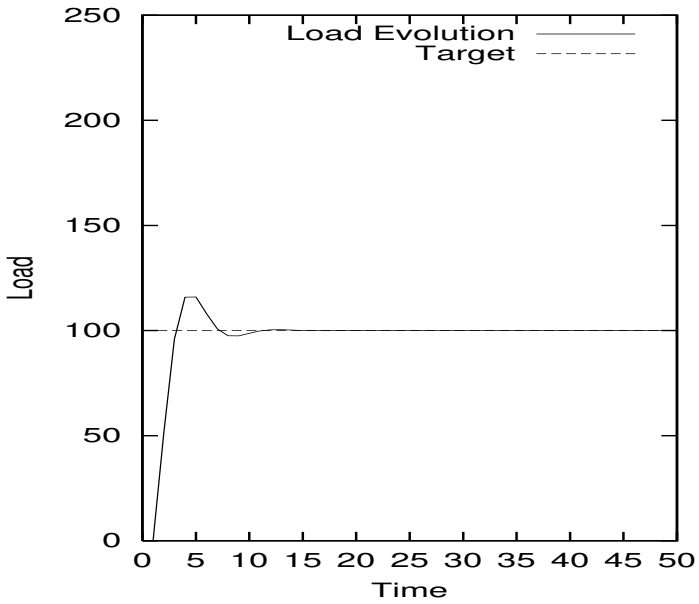
$\longrightarrow$ does it work?

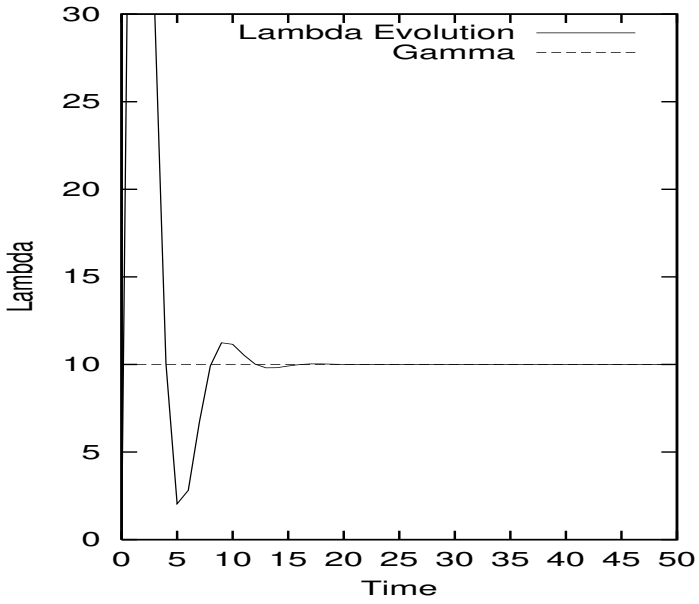
With $\varepsilon = 0.2$ and $\beta = 0.5$:



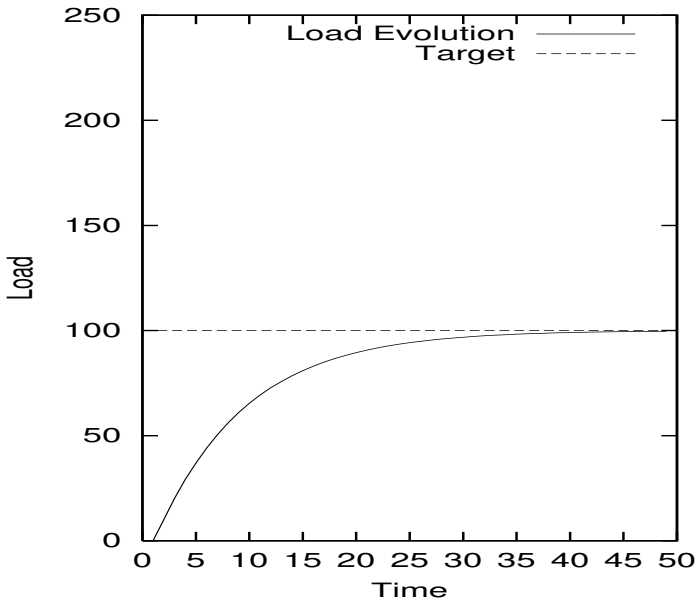


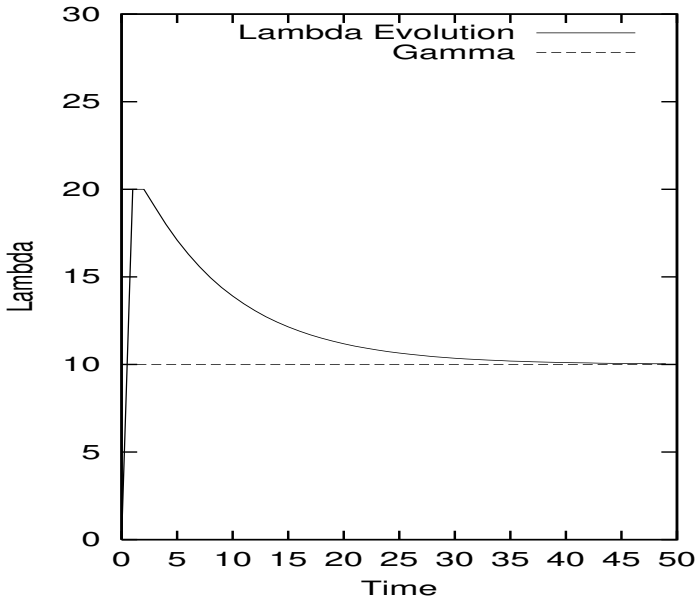
With $\varepsilon = 0.5$ and $\beta = 1.1$:





With $\varepsilon = 0.1$ and $\beta = 1.0$:





Remarks:

- Method D has desired behavior
- Superior to Methods A, B, and C,
- No unbounded oscillation
- Convergence to desired state

  $\rightarrow$ target operating point (Q^*, γ)

  $\rightarrow$ asymptotically stable

Related to PID control of physical systems.

$\rightarrow$ e.g., drones