Congestion Control

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

$\longrightarrow$ excessive traffic can cause congestion

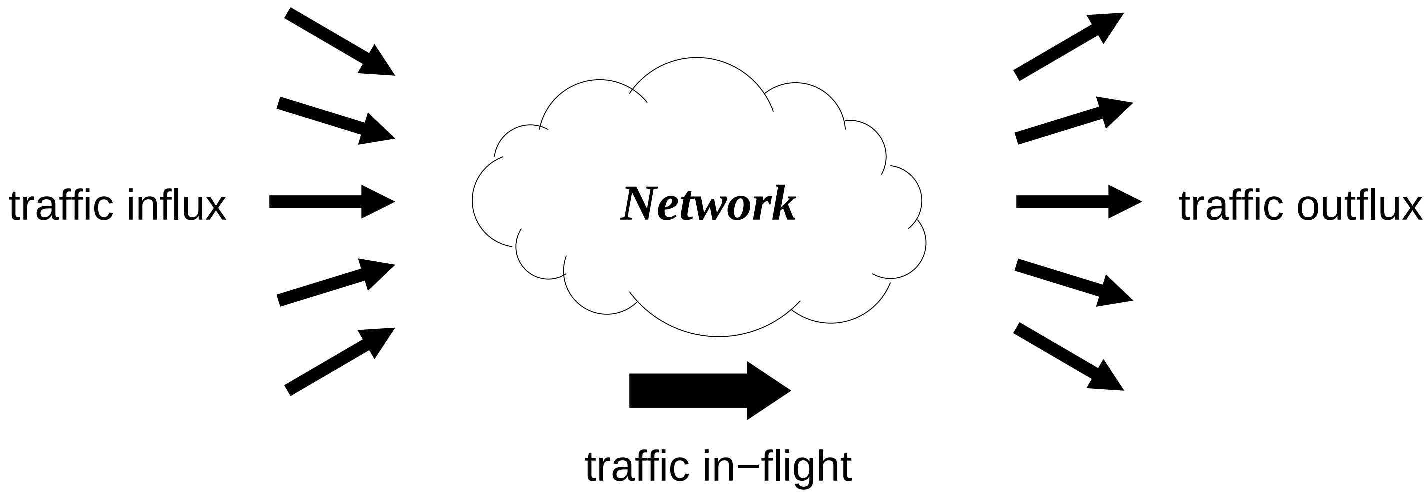
Problem: regulate traffic influx such that congestion does not occur

$\longrightarrow$ congestion control

Need to understand:

- What is congestion?
- How do we prevent or manage it?

Traffic influx/outflux picture:



- traffic influx: $\lambda(t)$ "offered load"

  $\rightarrow$ rate: bps (or pps) at time t

- traffic outflux: $\gamma(t)$ "throughput"

  $\rightarrow$ rate: bps (or pps) at time t

- traffic in-flight: $Q(t)$

  $\rightarrow$ volume: total packets in transit at time t

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

$\longrightarrow$ all 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

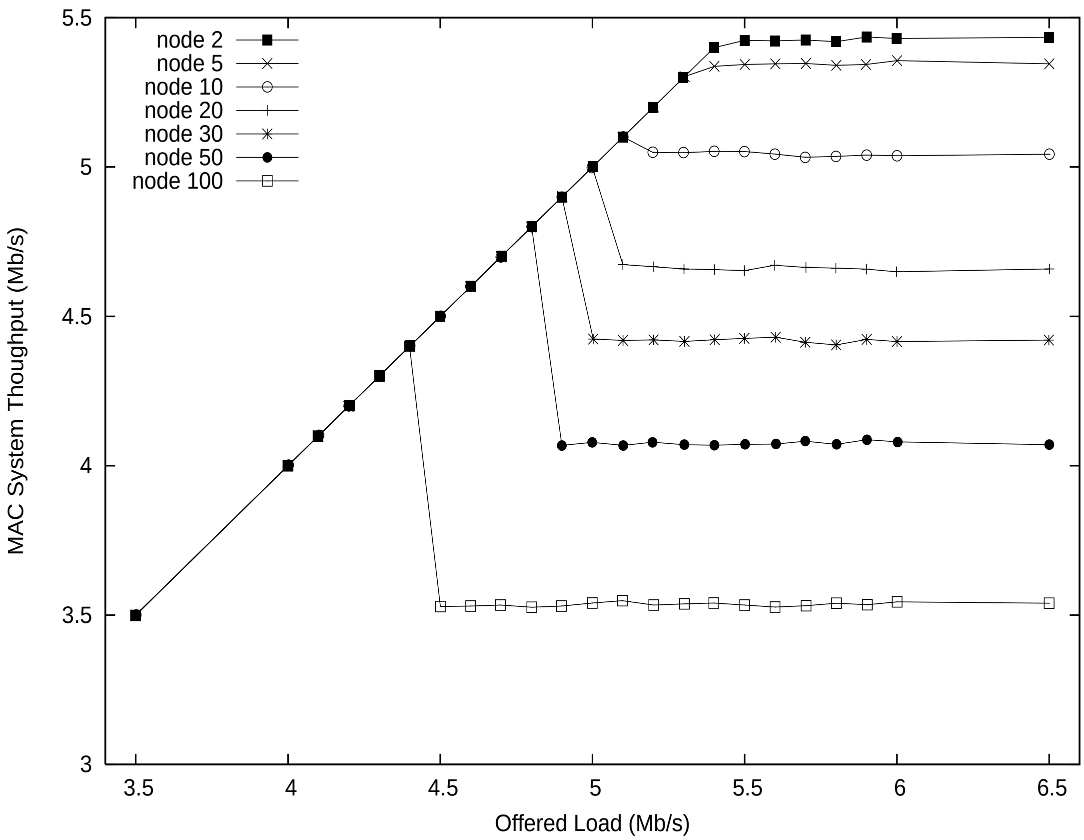
$\longrightarrow$ "congestion?"

faucet.com

Thermostat . . .

802.11b WLAN:

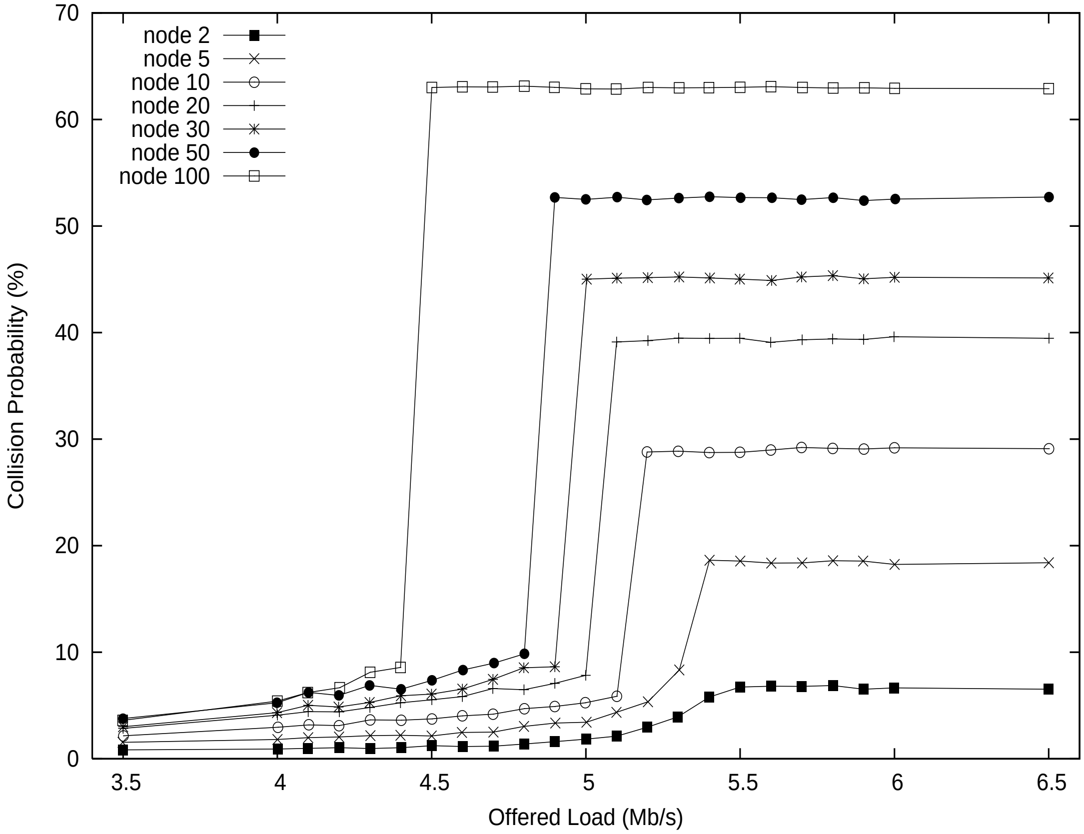
• Throughput



$\longrightarrow$ unimodal or bell-shaped

802.11b WLAN:

- Collision



$\longrightarrow$ underlying cause of unimodal throughput

What we can regulate or control:

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

Ex.:

- Faucet knob in water sink
- Temperature needle in thermostat
- Cars entering onto highway
- Traffic sent by UDP or TCP

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

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 (delivered to applications)
- $\lambda(t) - \gamma(t)$: net influx
- $Q(t)$ cannot be negative
- Missing item

Goal: Want to keep system in "good" state

Ex.: If $\lambda(t) > \gamma(t)$ for all time then

$$Q(t) \rightarrow \infty \quad \text{as} \quad t \rightarrow \infty$$

$\longrightarrow$ water level in sink grows and grows

$\longrightarrow$ water sink has finite "buffer" capacity, overflows

$\longrightarrow$ want to keep water level stable; how?

Control actions:

- If water level is too high, close faucet
- If water level is too low, open faucet

$\longrightarrow$ feedback control

Pseudo Real-Time Multimedia Streaming:

$\longrightarrow$ e.g., RealPlayer, Rhapsody, Internet radio

$\longrightarrow$ "pseudo" because prefetching trick

$\longrightarrow$ name of the game: prevent empty buffer

Method:

- Prefetch X seconds worth of data (e.g., 5 seconds)

  $\rightarrow$ audio/video frames

- Initial delayed playback

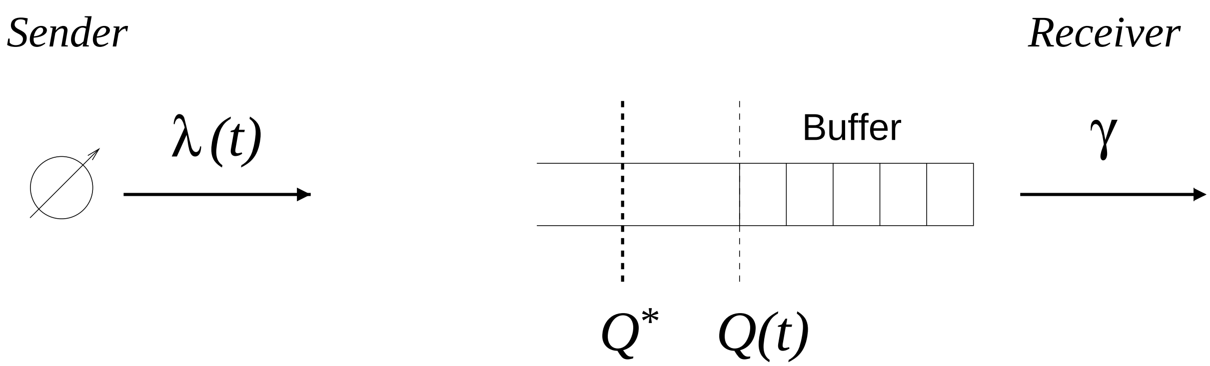
  $\rightarrow$ penalty: pseudo real-time

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

  $\rightarrow$ buffering allows hiding of spurious congestion

  $\rightarrow$ continuous playback experience

Pseudo real-time traffic control:



- $Q(t)$: current buffer level
- Q^*: desired buffer level
- γ: 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

$\longrightarrow$ don't buffer too little

Basic idea:

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

$\longrightarrow$ control law

$\longrightarrow$ thermostat control (same as water faucet)

Protocol implementation:

- Control action undertaken at sender
- Receiver needs to inform sender of Q^* and $Q(t)$

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

Other applications:

Router congestion control

$\longrightarrow$ active queue management (AQM)

- Receiver is viewed as router
- Q^* is viewed as desired buffer occupancy and delay
- Router throttles sender(s) to maintain Q^*

$\longrightarrow$ feedback: ECN (explicit congestion notification)

$\longrightarrow$ two bits in IPv4 TOS field

$\longrightarrow$ supported in most routers, not turned on

Also proposed to throttle denial-of-service attack traffic

$\longrightarrow$ called push-back

$\longrightarrow$ good guy vs. bad guy problem

Key question in feedback traffic control: how much to increase or decrease $\lambda(t)$?

$\longrightarrow$ "control problem"

$\longrightarrow$ job of traffic protocols (including TCP)

What is the desired state of the system?

$\longrightarrow$ operating point

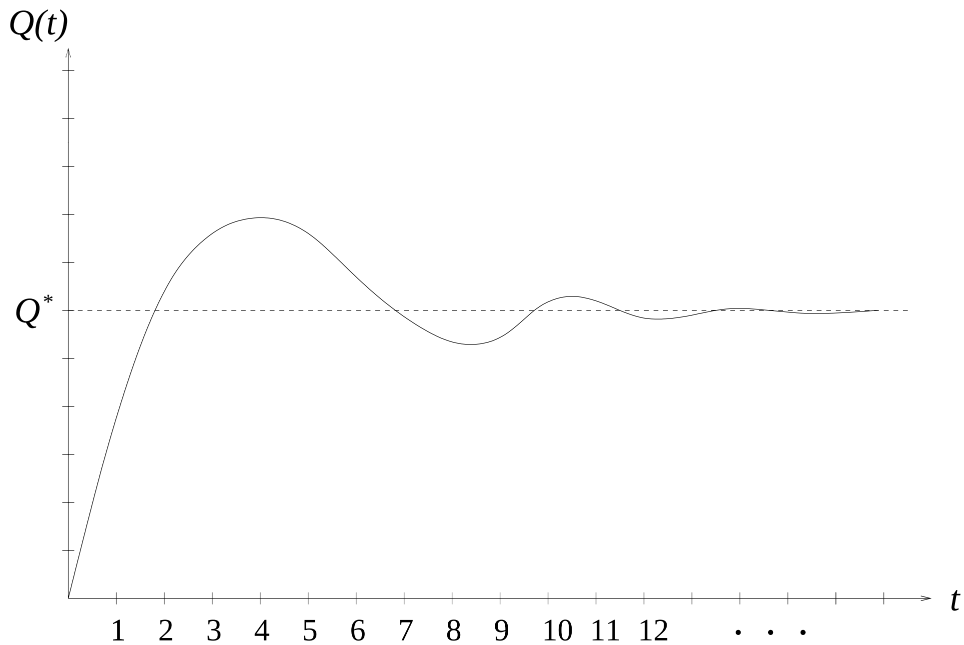
$$Q(t) = Q^* \text{ and } \lambda(t) = \gamma$$

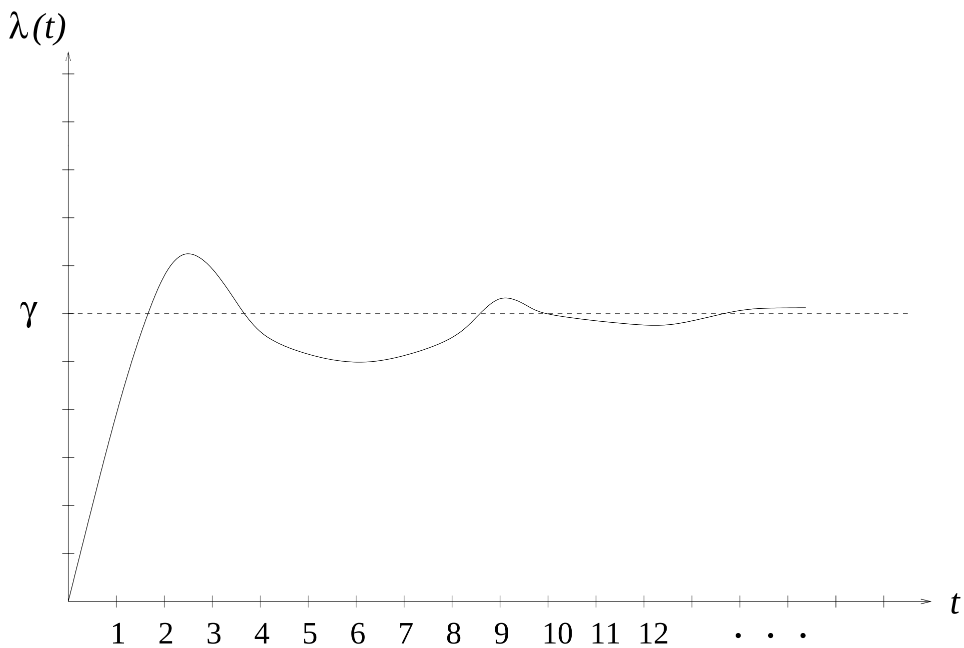
Where do we start from?

$\longrightarrow$ empty buffer and no sending rate at start

$$Q(t) = 0 \text{ and } \lambda(t) = 0$$

Time evolution (or dynamics):





`Method A`:

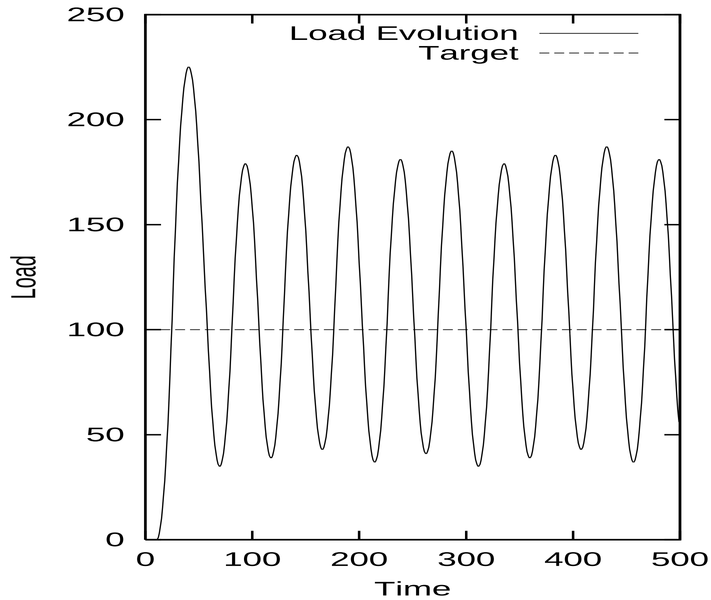
- if $Q(t) = Q^*$ then $\lambda(t+1) \leftarrow \lambda(t)$
- if $Q(t) < Q^*$ then $\lambda(t+1) \leftarrow \lambda(t) + a$
- if $Q(t) > Q^*$ then $\lambda(t+1) \leftarrow \lambda(t) - a$

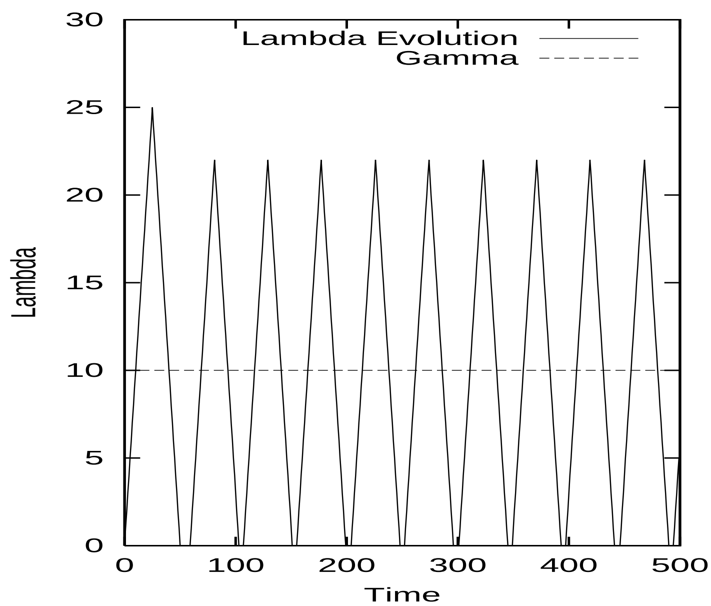
where $a > 0$ is a fixed parameter

Question: does it work?

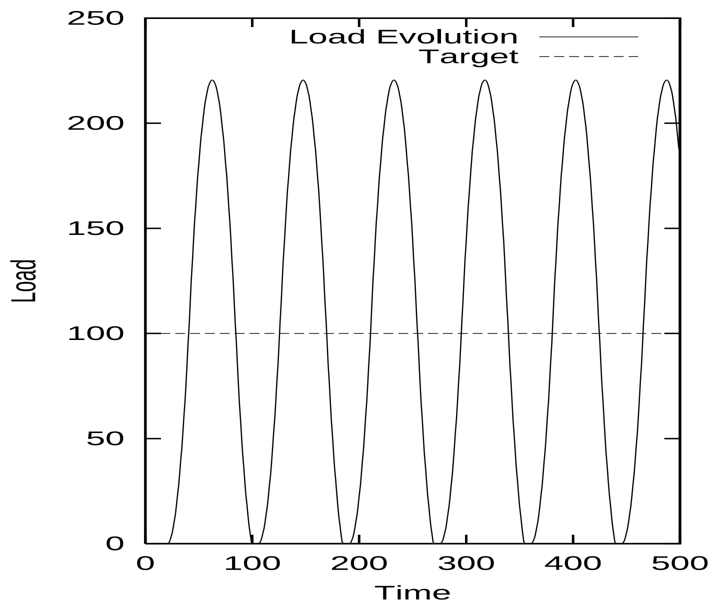
Example:

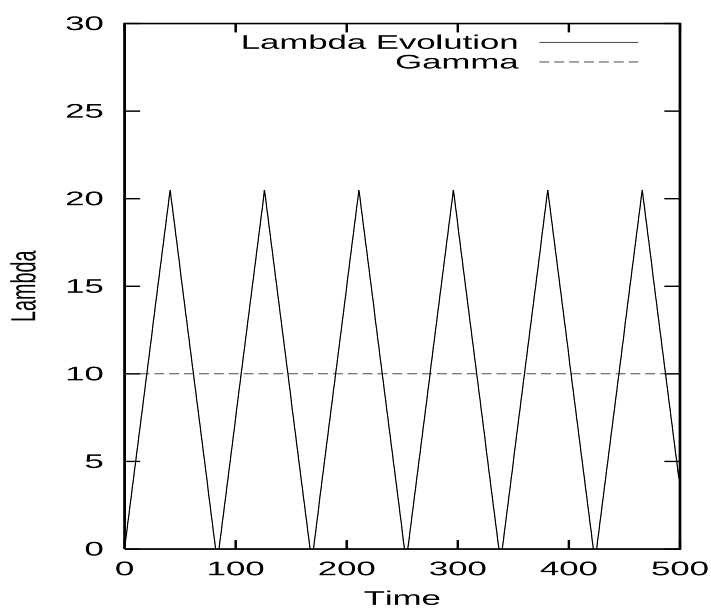
- $Q^* = 100$
- $\gamma = 10$
- $Q(0) = 0$
- $\lambda(0) = 0$
- $a = 1$


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

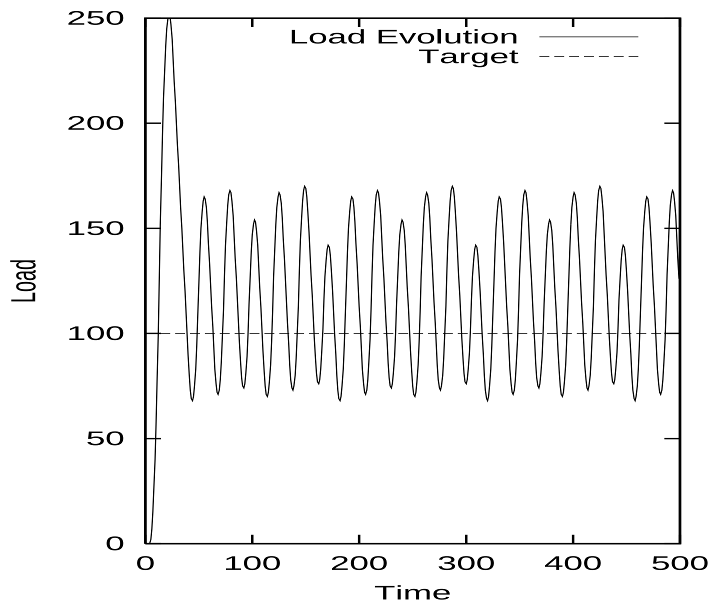

Lambda Evolution
Gamma
Lambda
Time
30
25
20
15
10
5
0
0
100
200
300
400
500

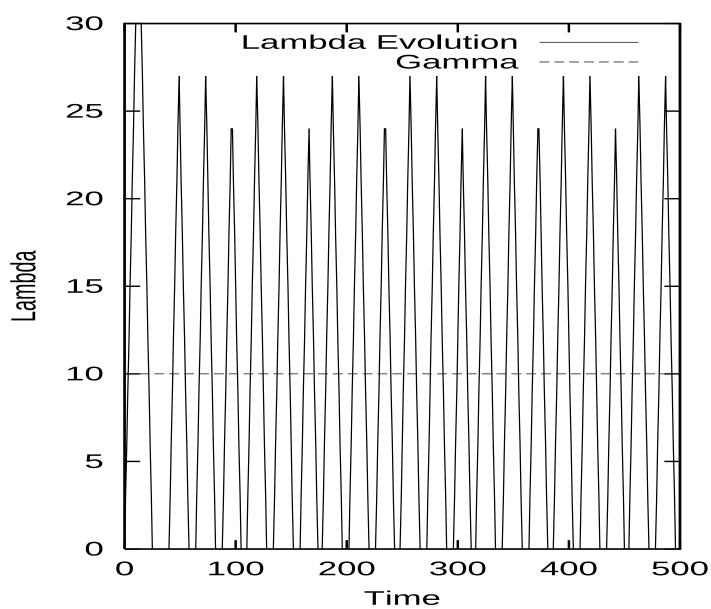
With $a = 0.5$:


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

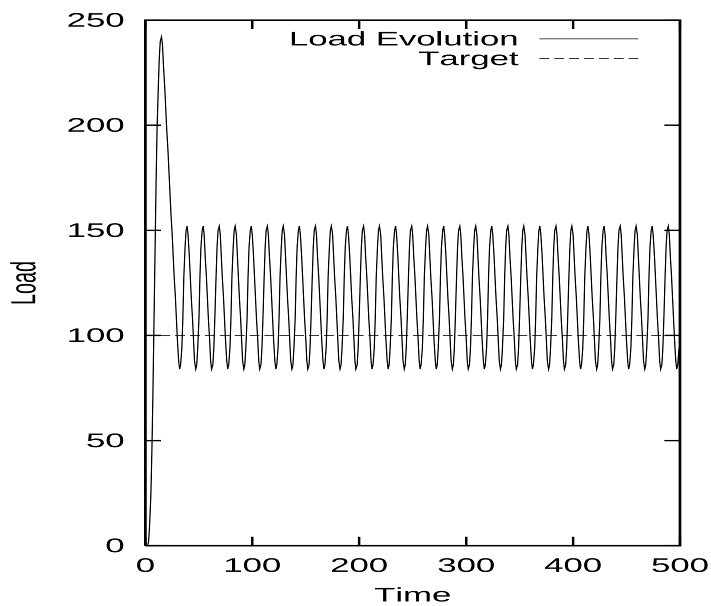

Lambda Evolution
Gamma
Lambda
Time
30
25
20
15
10
5
0
0
100
200
300
400
500

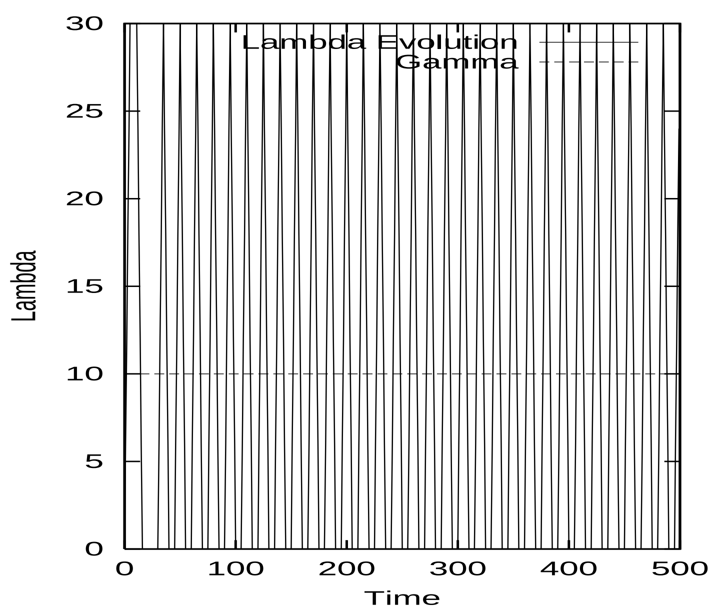
With $a = 3$:


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
250
200
150
100
50
0
0
100
200
300
400
500

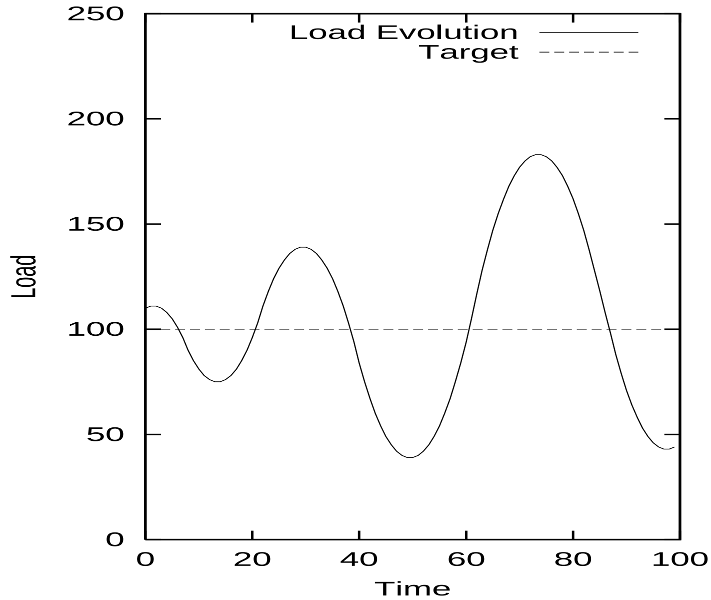

Lambda Evolution
Gamma
Lambda
Time
30
25
20
15
10
5
0
0
100
200
300
400
500

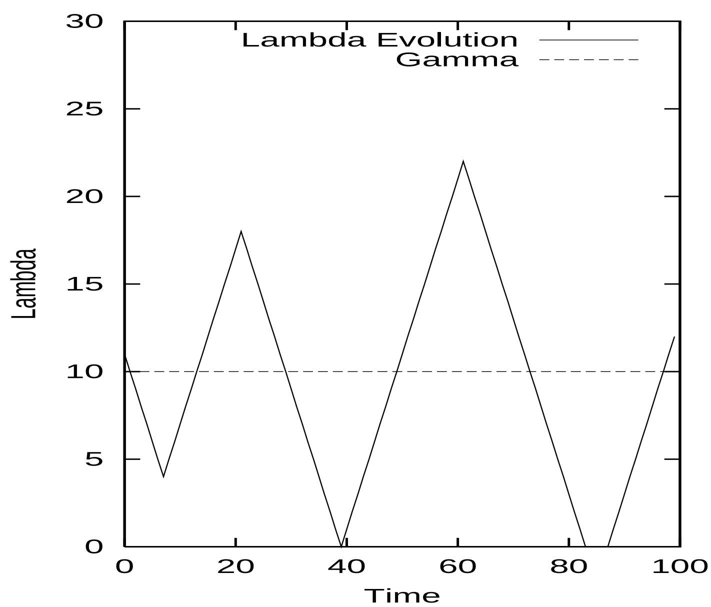
Remarks:

- Method A isn't that great no matter what a value is used
- Actually: would lead to unbounded oscillation if not for physical restriction $\lambda(t) \geq 0$ and $Q(t) \geq 0$

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

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

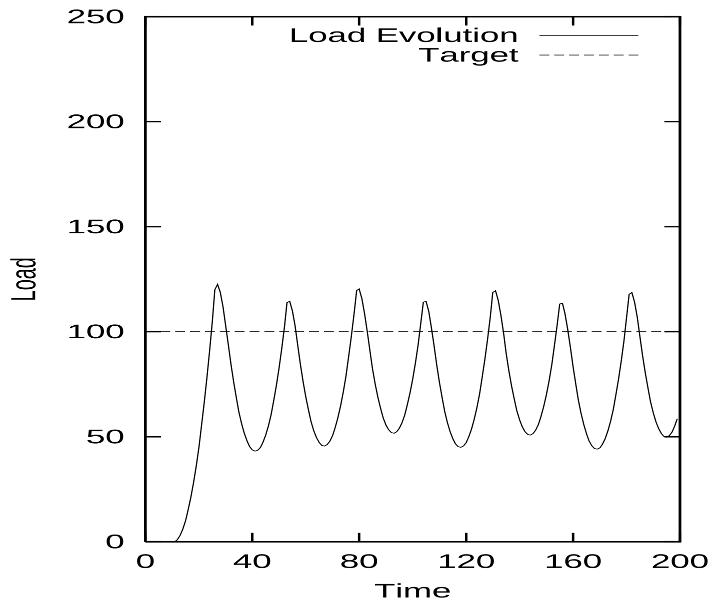
where $a > 0$ and $0 < \delta < 1$ are fixed parameters

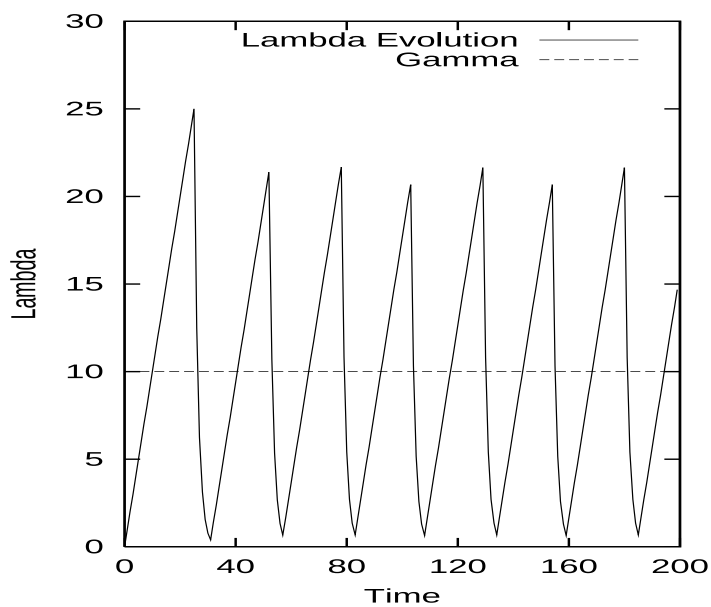
$\longrightarrow$ linear increase with slope a

$\longrightarrow$ exponential decrease with backoff factor δ

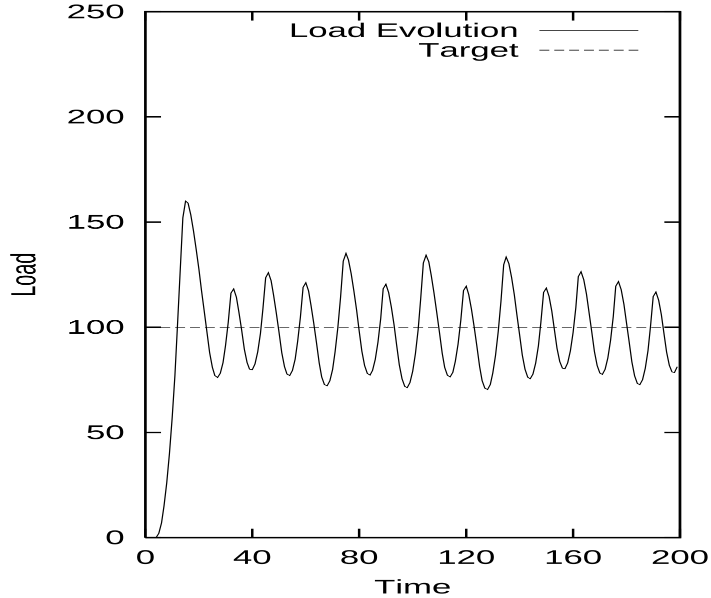
$\longrightarrow$ e.g., binary backoff in case $\delta = 1/2$

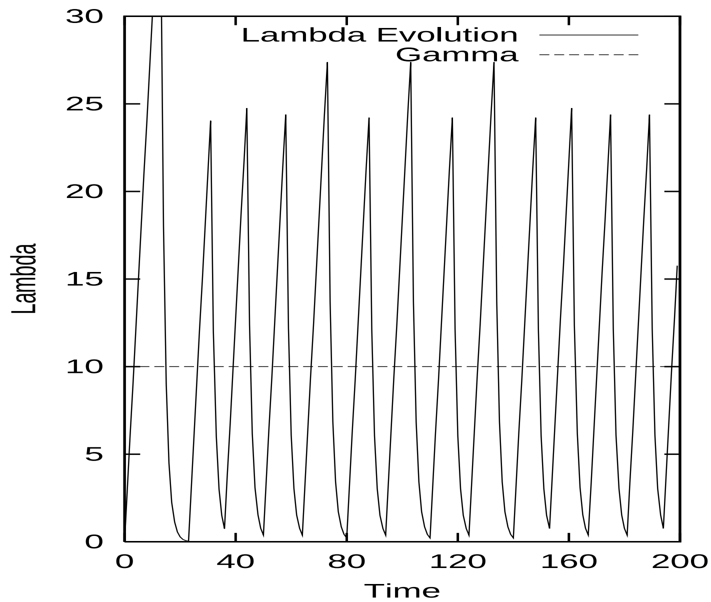
With $a = 1$, $\delta = 1/2$:



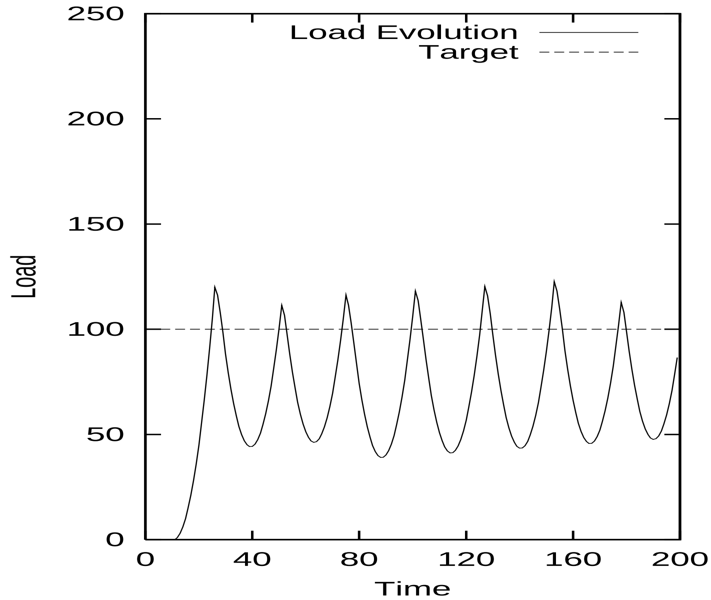


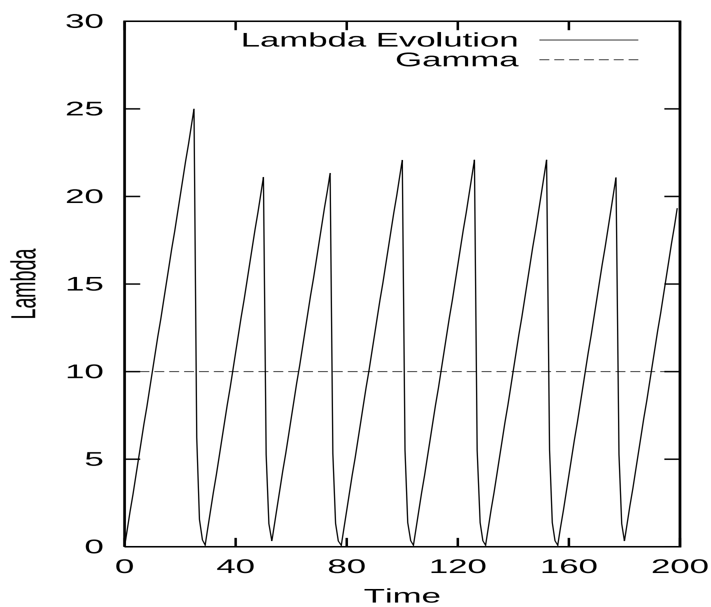
With $a = 3$, $\delta = 1/2$:



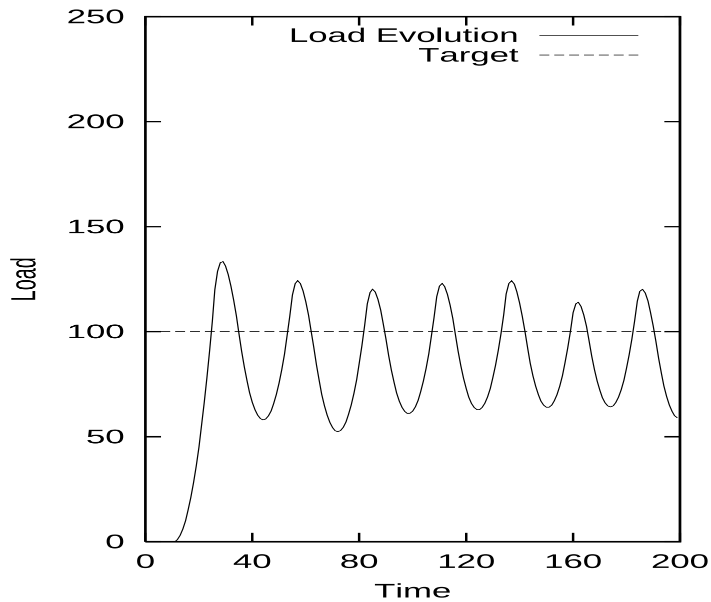


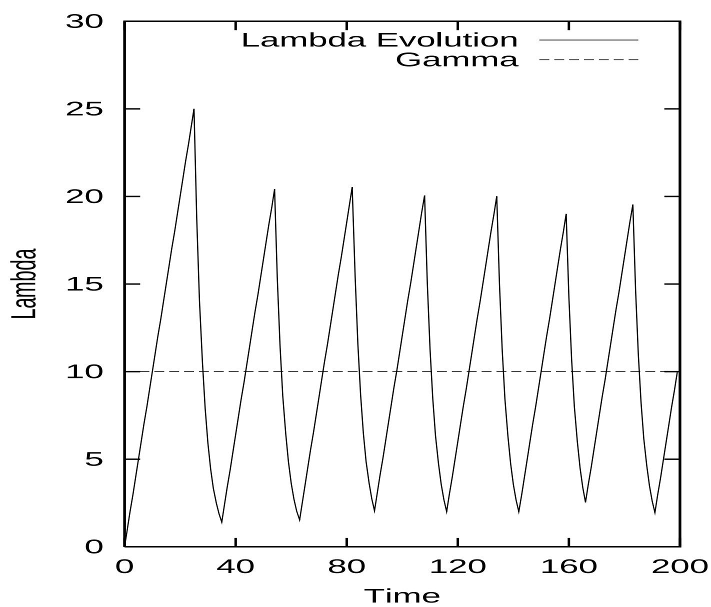
With $a = 1$, $\delta = 1/4$:





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

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

  $\longrightarrow$ "sawtooth" pattern
- Method B is used by TCP

  $\longrightarrow$ linear increase/exponential decrease

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

Question: Can we do better?

`Method C`:

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

where $\varepsilon > 0$ is a fixed parameter

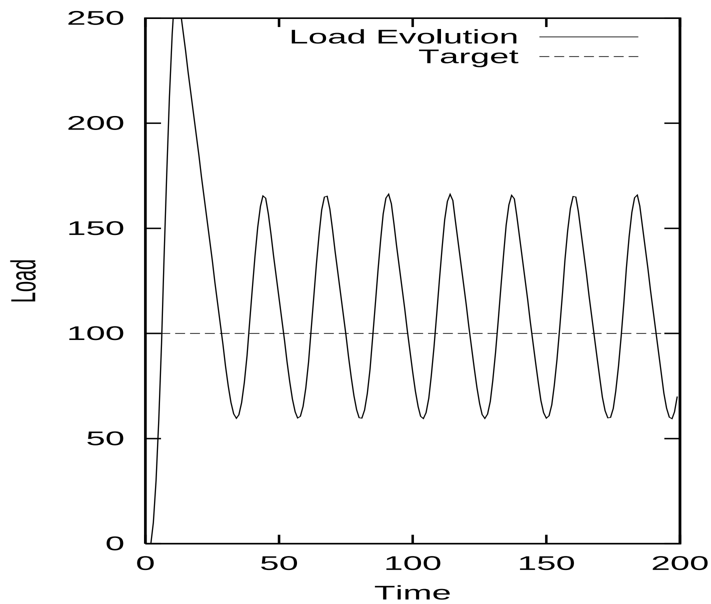
Tries to adjust magnitude of change as a function of gap $Q^* - Q(t)$

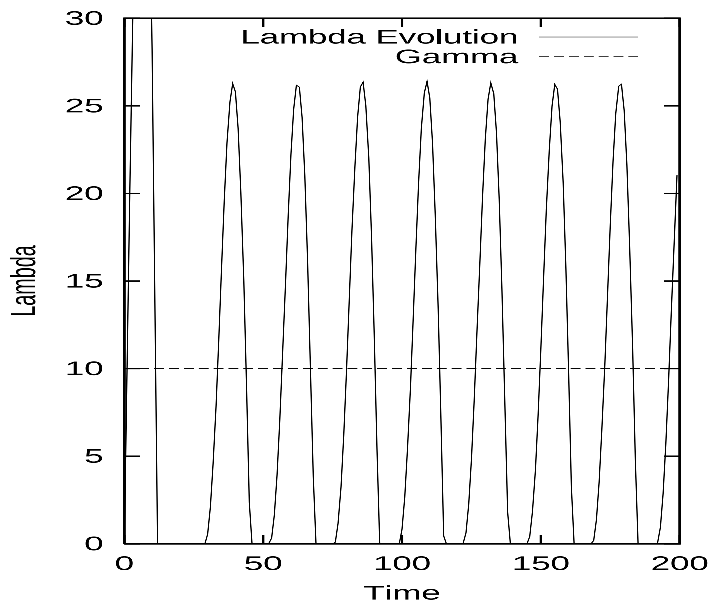
- if $Q^* - Q(t) > 0$, increase $\lambda(t)$ proportional to gap
- if $Q^* - Q(t) < 0$, decrease $\lambda(t)$ proportional to gap

Trying to be more clever...

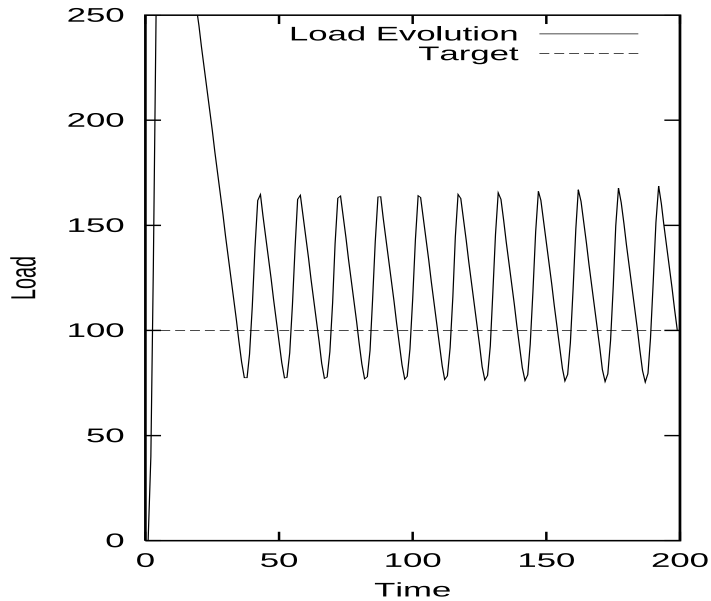
$\longrightarrow$ bottom line: is it any good?

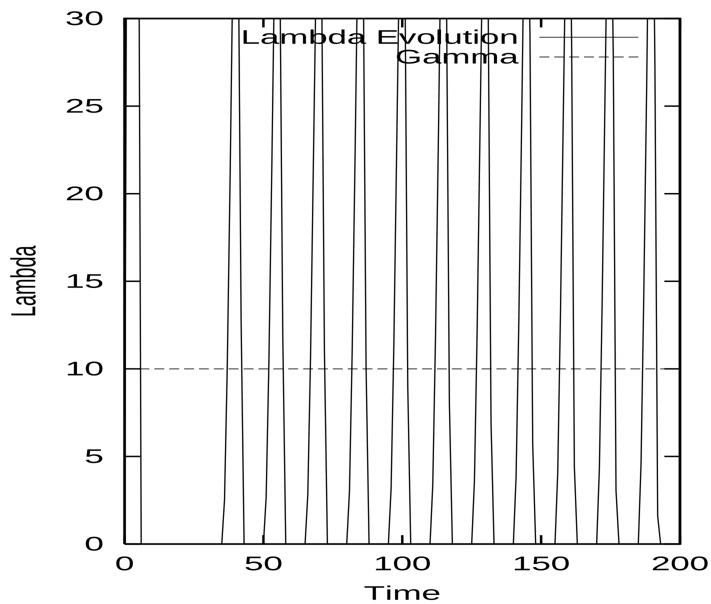
With $\varepsilon = 0.1$:





With $\varepsilon = 0.5$:





Answer: No.

Time to try something strange.

`Method D`:

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

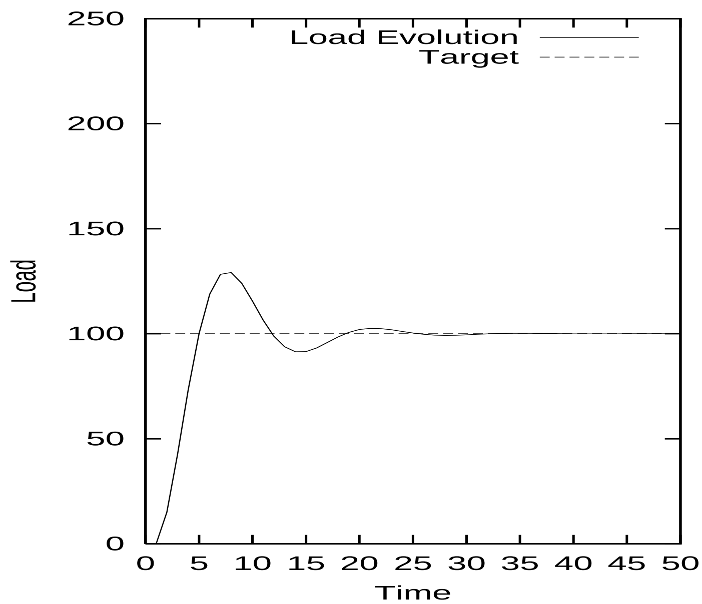
where $\varepsilon > 0$ and $\beta > 0$ are fixed parameters

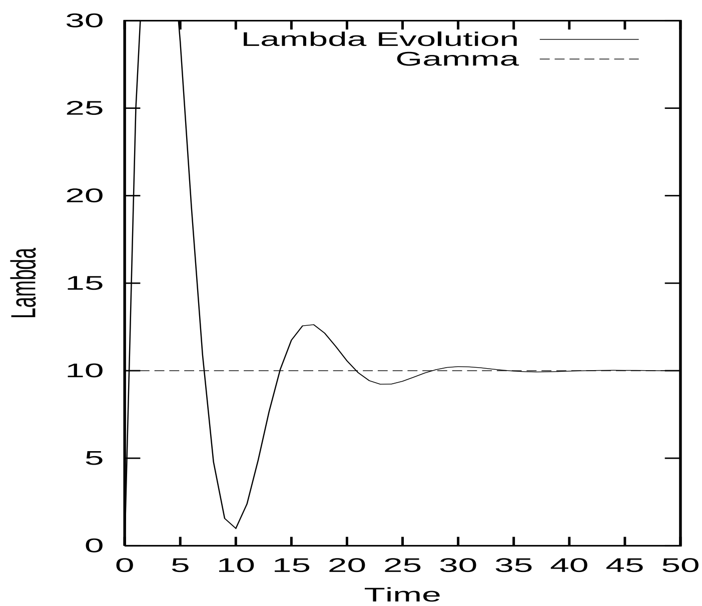
$\longrightarrow$ kind of like Method C

$\longrightarrow$ what's going on?

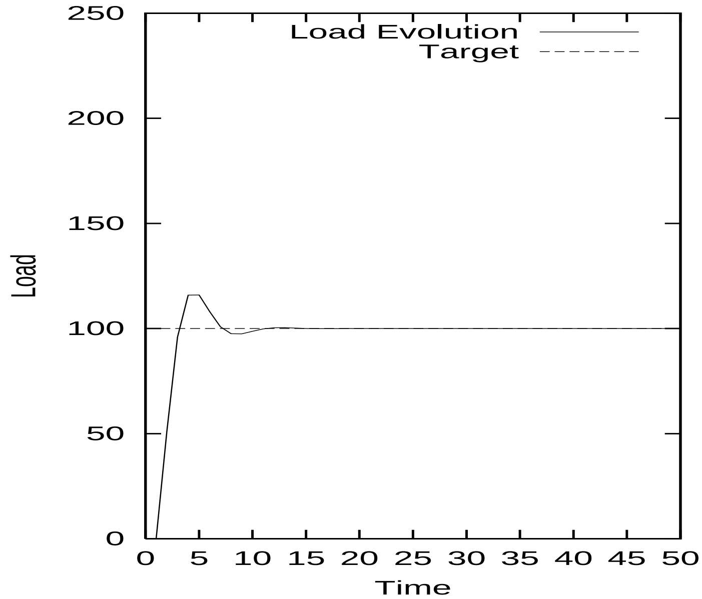
Sanity check: at desired operating point $Q(t) = Q^*$ and $\lambda(t) = \gamma$, what will happen under Method D?

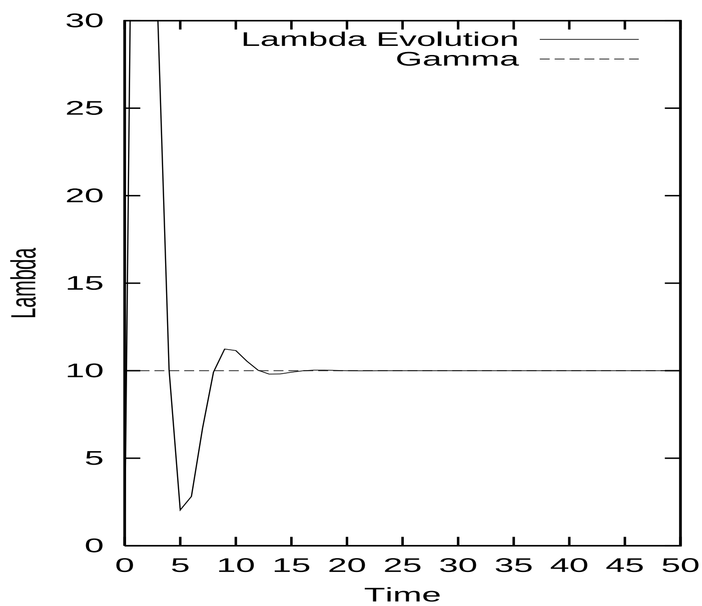
With $\varepsilon = 0.2$ and $\beta = 0.5$:



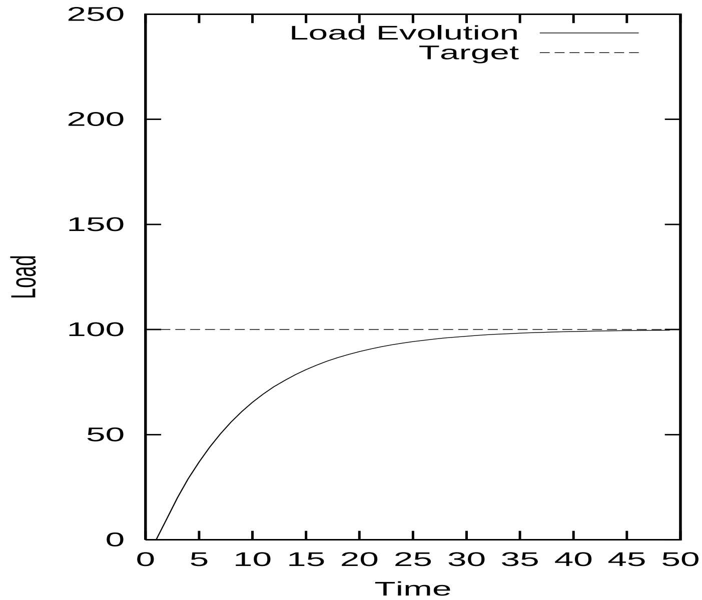


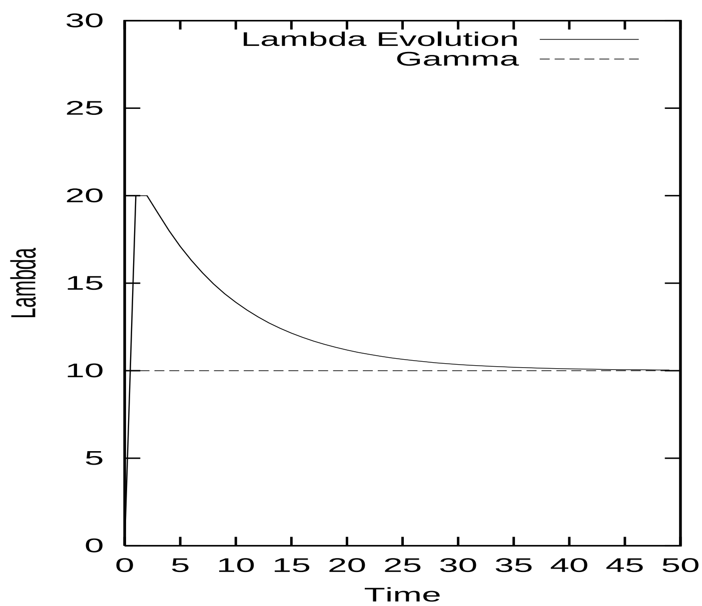
With $\varepsilon = 0.5$ and $\beta = 1.1$:





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 operating point

  $\rightarrow$ asymptotic stability

  $\rightarrow$ oscillations die down

Why does it work?

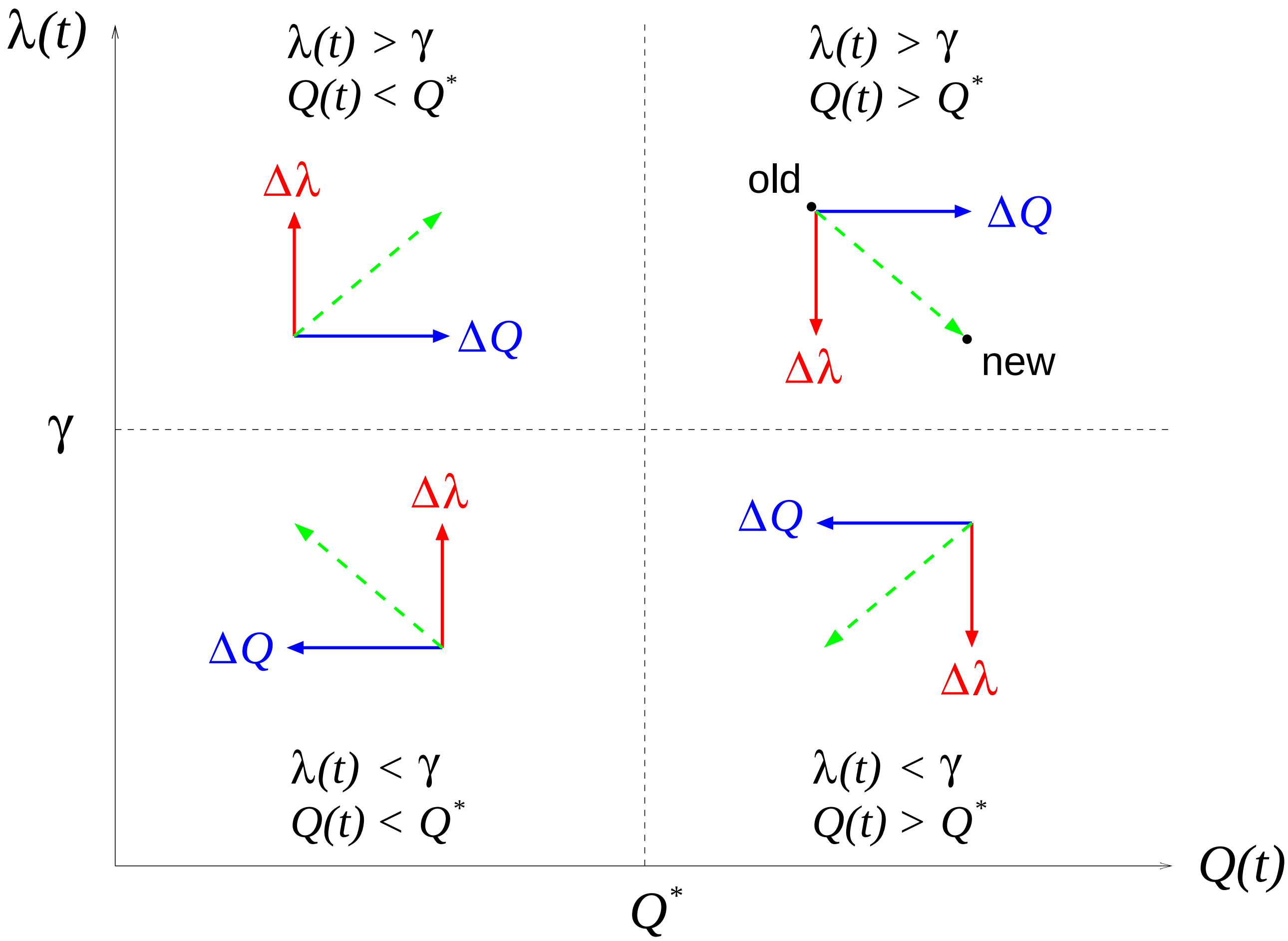
What is the role of the $-\beta(\lambda(t) - \gamma)$ term in the control law

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

Need to look beneath the hood ...

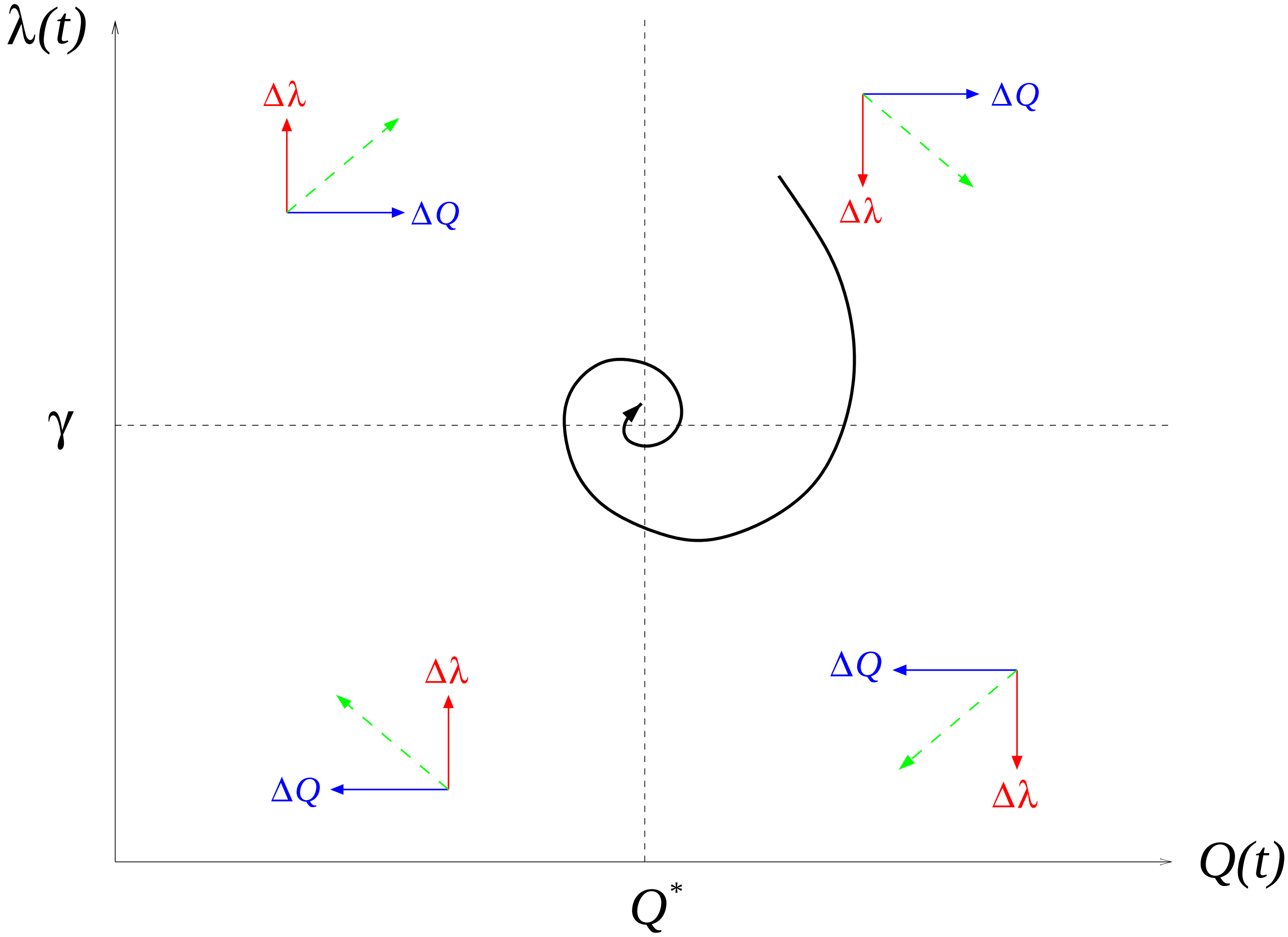
Visualize action in 2-D $(Q(t), \lambda(t))$-space:

$\longrightarrow$ phase space



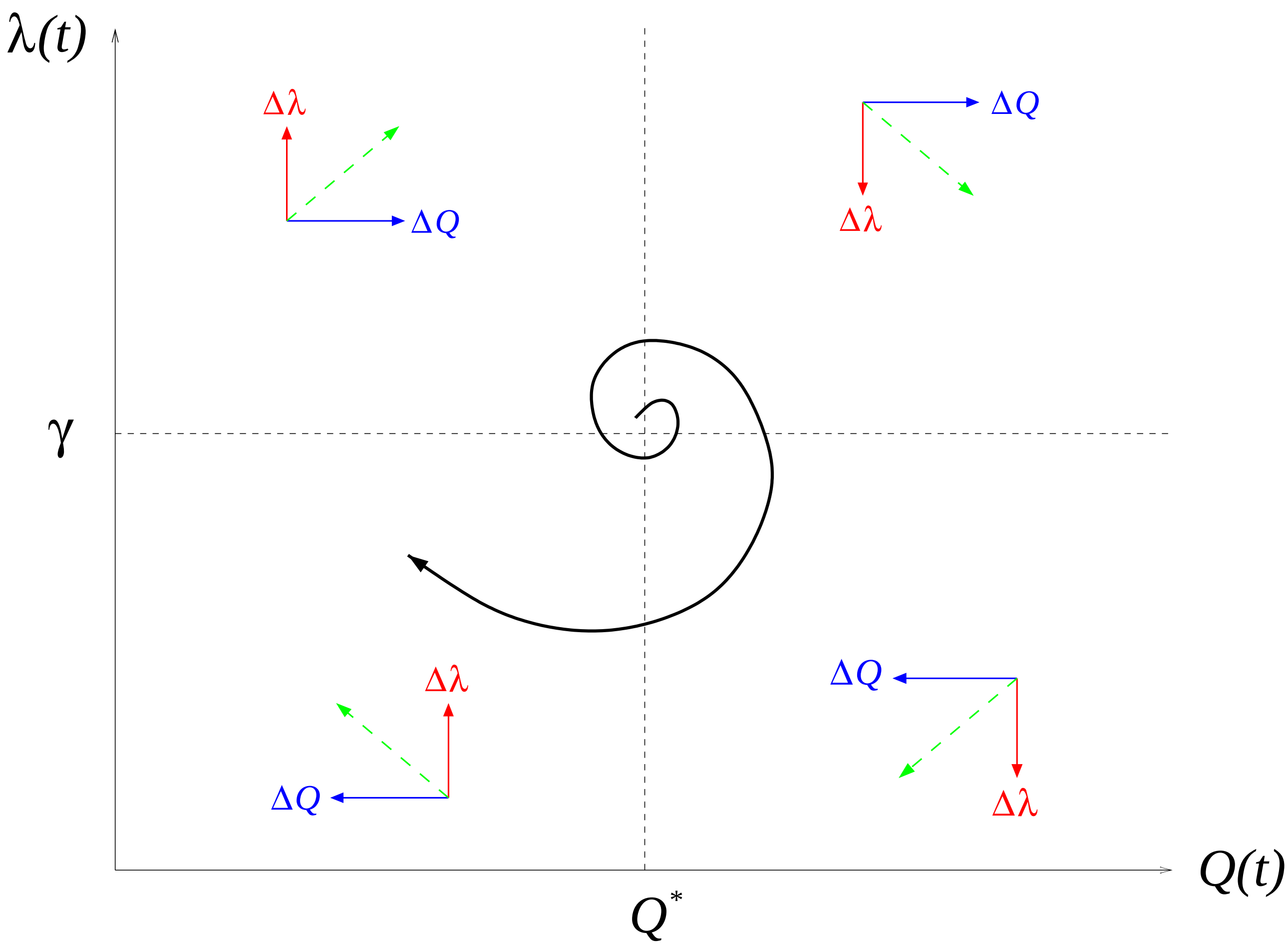
Convergent trajectory:

$\longrightarrow$ asymptotically stable & optimal



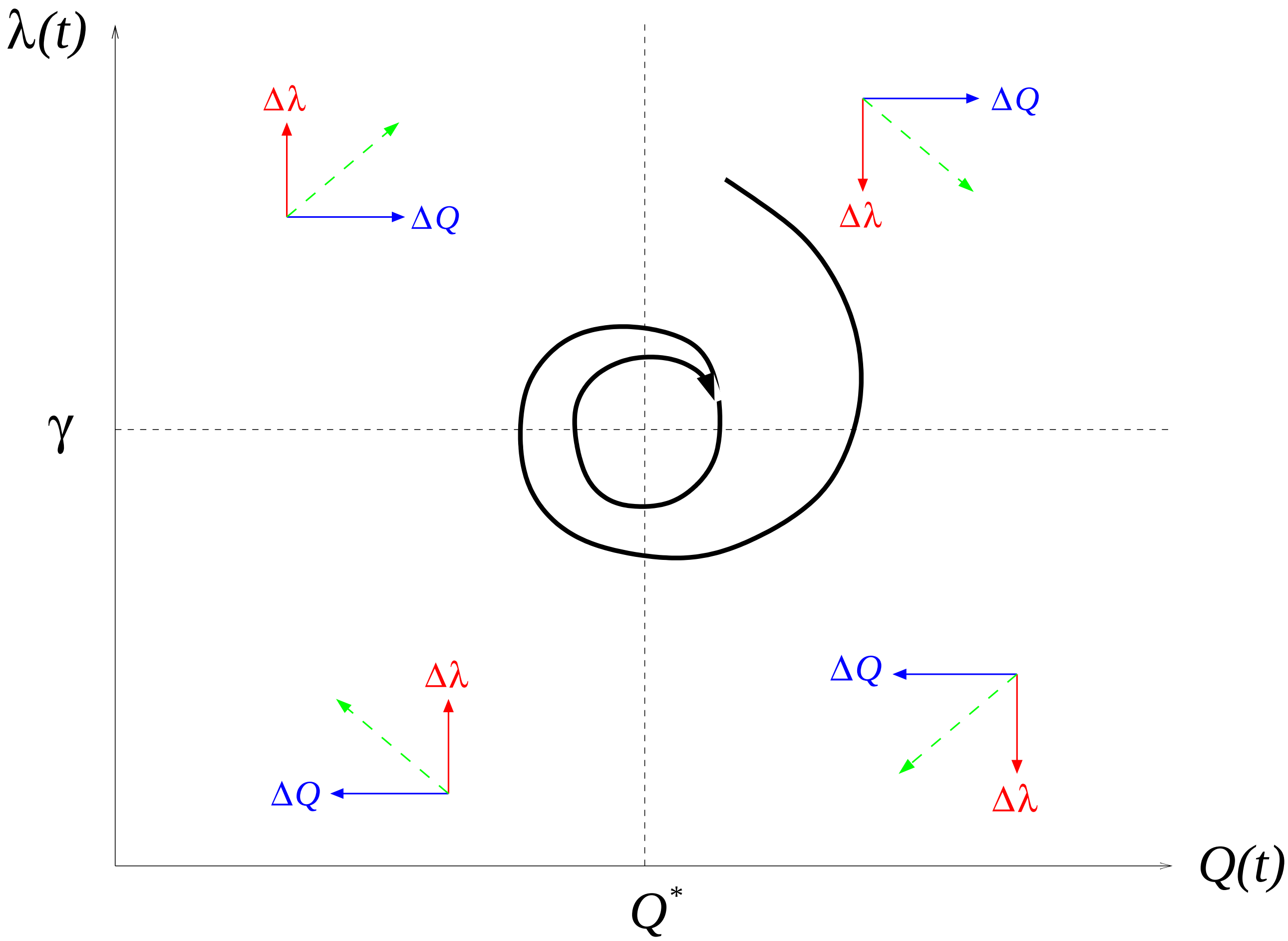
Divergent trajectory:

$\longrightarrow$ unstable



Stable (but not asymptotically so) trajectory:

$\longrightarrow$ limit cycle



Which case arises depends on the specifics of protocol actions.

For example:

- Methods A and C: divergent
- Method B: stable (but not asymptotically)

  $\rightarrow$ TCP
- Method D: asymptotically stable & optimal

  $\rightarrow$ "optimal control"

Why does Method D work?!

$\longrightarrow$ 'cause it does

For longer explanation, come to graduate school...

$\longrightarrow$ not just any graduate school