`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

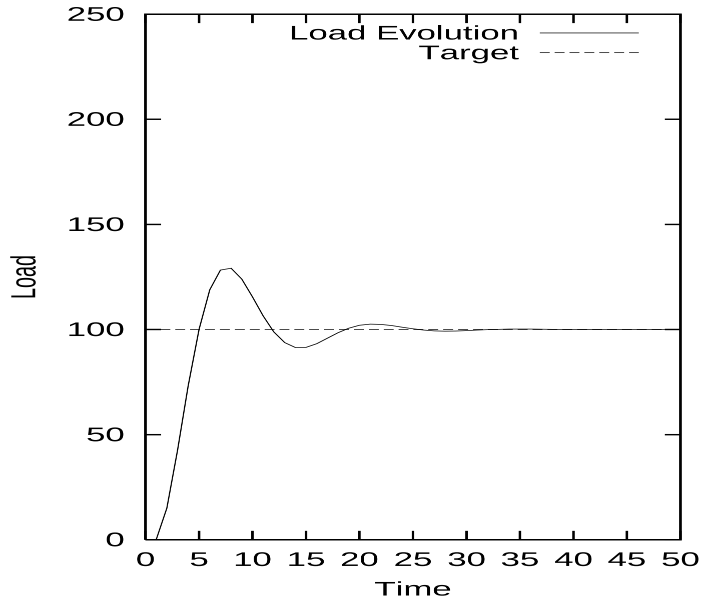
$\longrightarrow$ odd looking modification to `Method C`

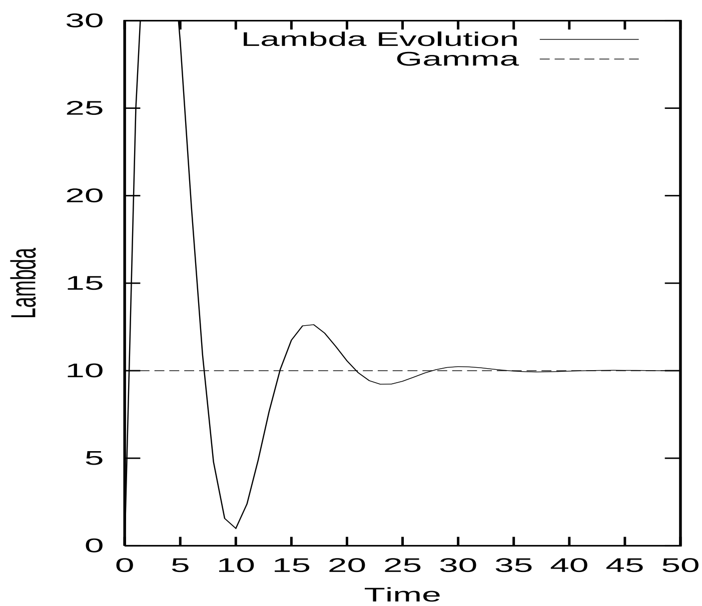
$\longrightarrow$ additional term $-\beta(\lambda(t) - \gamma)$

$\longrightarrow$ what's going on?

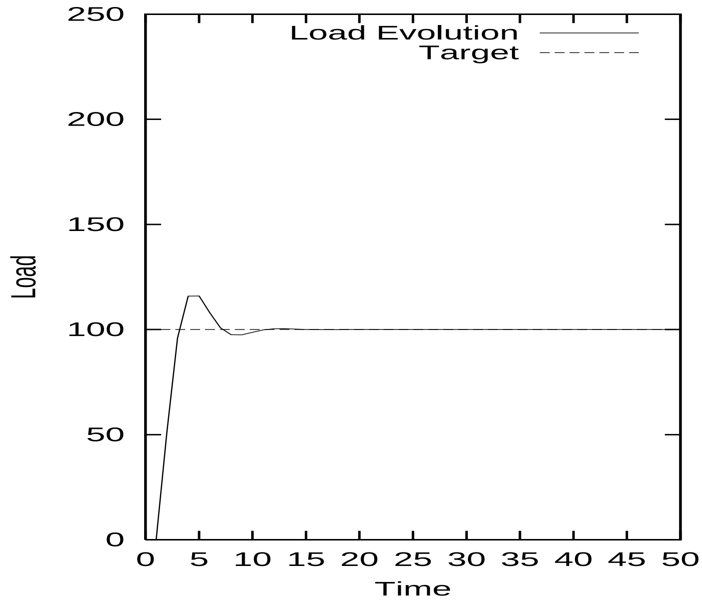
$\longrightarrow$ does it work?

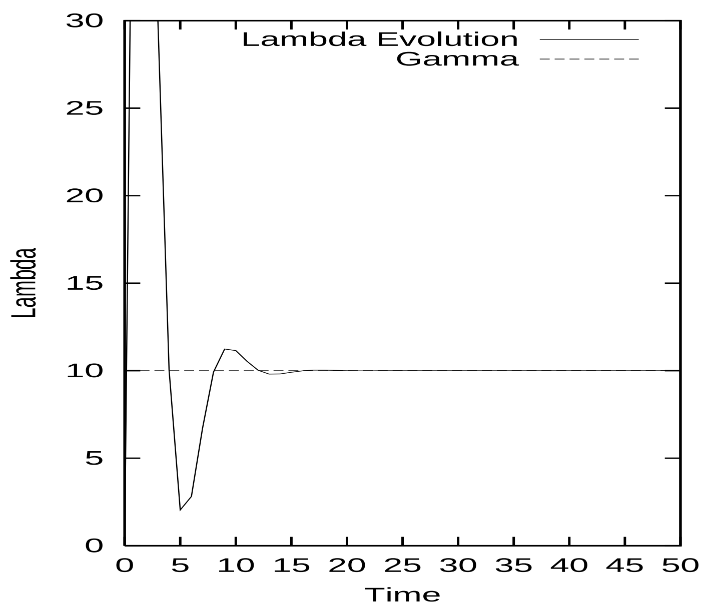
With $\varepsilon = 0.2$ and $\beta = 0.5$:



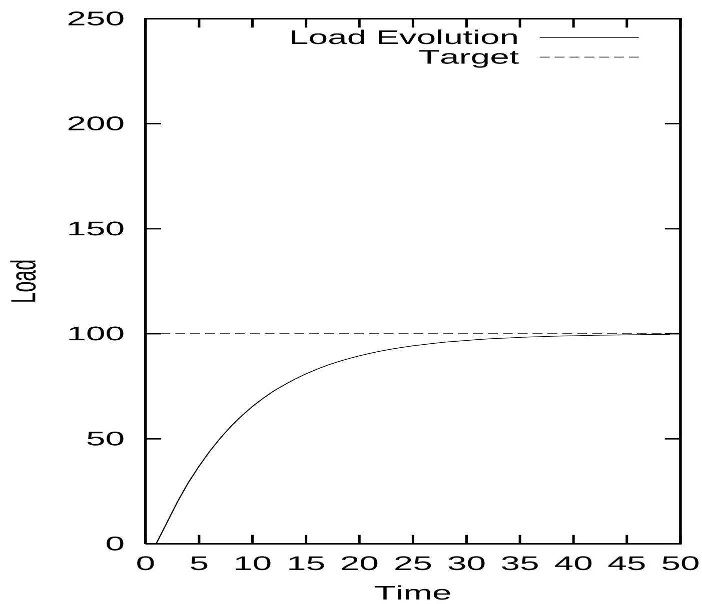


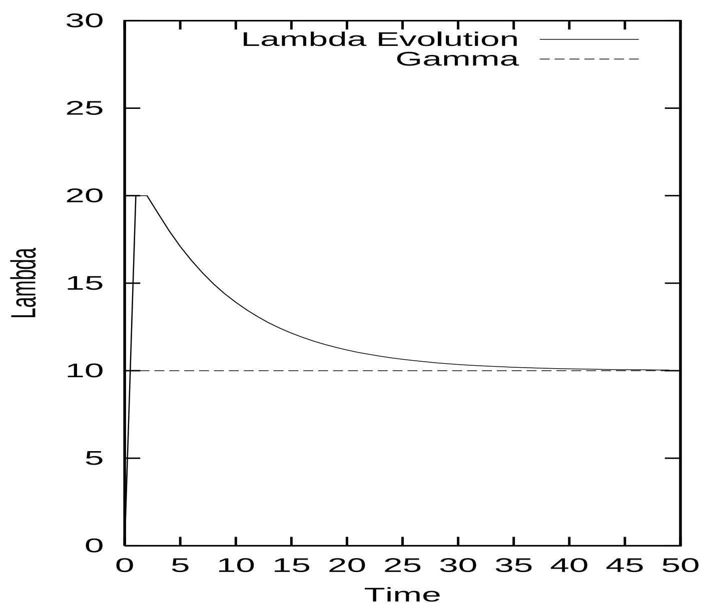
With $\varepsilon = 0.5$ and $\beta = 1.1$:





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

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

  $\rightarrow$ called asymptotically stable

  $\rightarrow$ why?

TCP congestion control

Recall:

$$\texttt{EffectiveWindow} = \texttt{MaxWindow} - (\texttt{LastByteSent} - \texttt{LastByteAcked})$$

where

$$\texttt{MaxWindow} = \min\{\, \texttt{AdvertisedWindow}, \texttt{CongestionWindow} \,\}$$

Key question: how to set `CongestionWindow` which, in turn, affects ARQ's sending rate?

$\longrightarrow$ linear increase/exponential decrease

$\longrightarrow$ AIMD

$\longrightarrow$ method B

TCP congestion control components:

(i) Congestion avoidance

$\longrightarrow$ linear increase/exponential decrease

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

As in Method B, increase `CongestionWindow` linearly, but decrease exponentially

Upon receiving ACK:

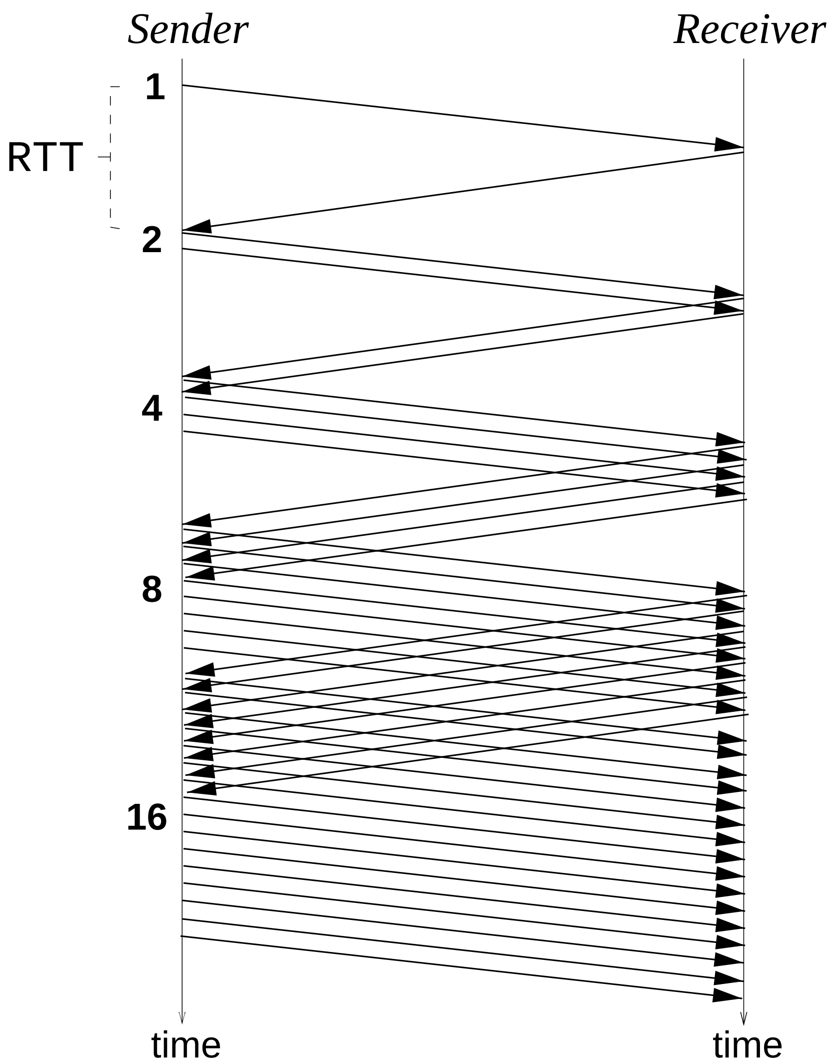
$$\texttt{CongestionWindow} \leftarrow \texttt{CongestionWindow} + 1$$

Upon timeout:

$$\texttt{CongestionWindow} \leftarrow \texttt{CongestionWindow} \, / \, 2$$

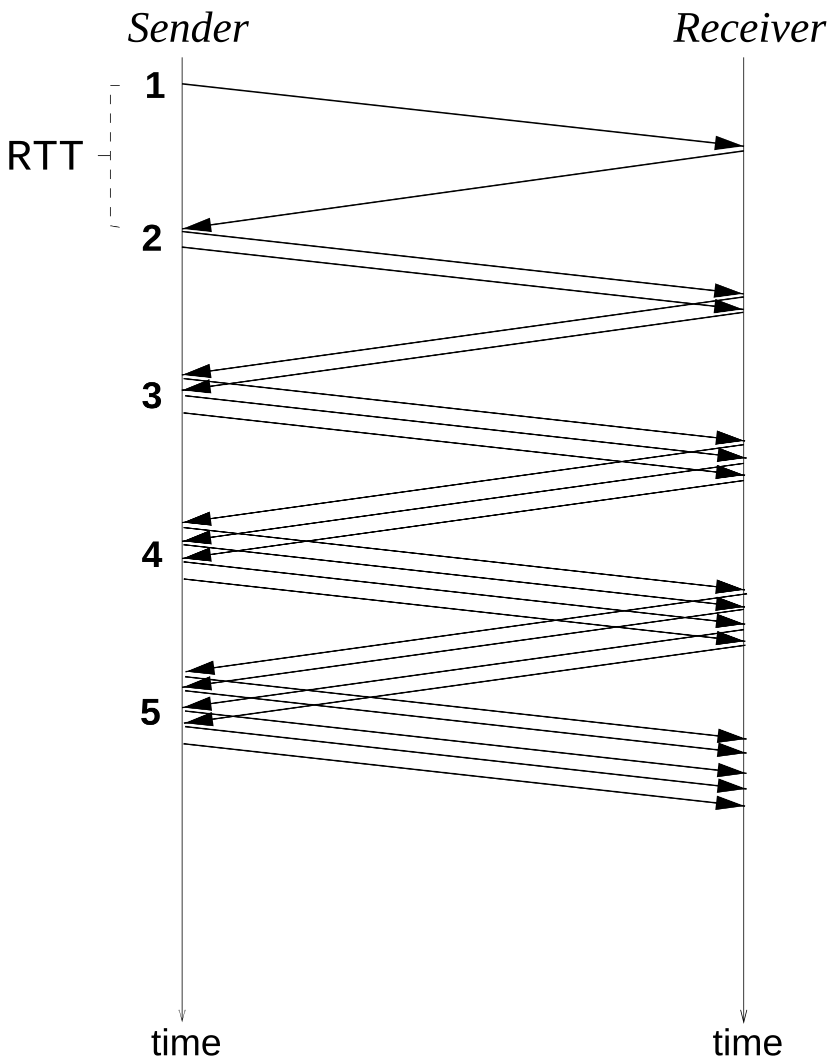
But is it correct...

"Linear increase" time diagram:



$\longrightarrow$ results in exponential increase

What we want:



$\longrightarrow$ increase by 1 every window

Thus, linear increase update:

$$\texttt{CongestionWindow} \leftarrow \texttt{CongestionWindow} + (1 \, / \, \texttt{CongestionWindow})$$

Upon timeout and exponential backoff,

$$\texttt{SlowStartThreshold} \leftarrow \texttt{CongestionWindow} \, / \, 2$$

(ii) Slow Start

Reset `CongestionWindow` to 1

Perform exponential increase

$$\texttt{CongestionWindow} \leftarrow \texttt{CongestionWindow} + 1$$

- Until timeout at start of connection

  $\rightarrow$ rapidly probe for available bandwidth

- Until `CongestionWindow` hits `SlowStartThreshold` following Congestion Avoidance

  $\rightarrow$ rapidly climb to safe level

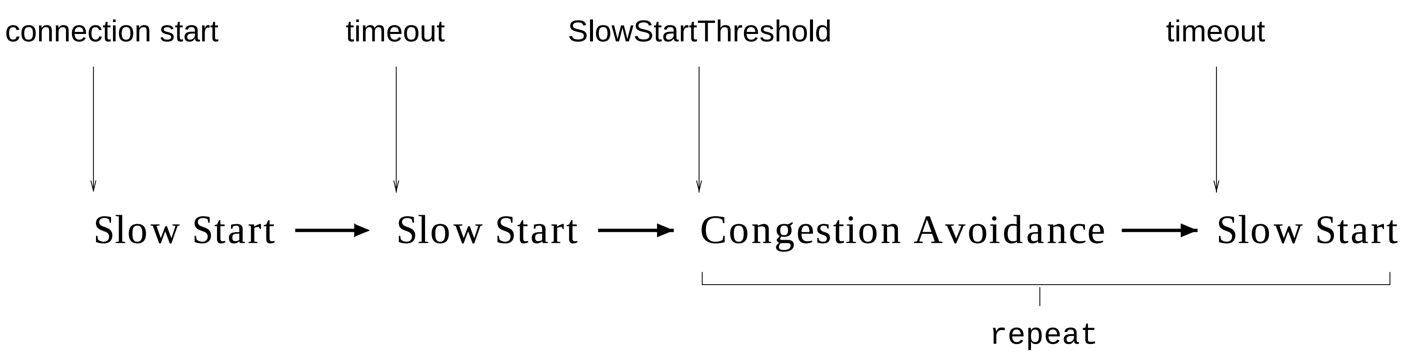
$\longrightarrow$ "slow" is a misnomer

$\longrightarrow$ exponential increase is super-fast

Basic dynamics:

⟶ after connection set-up

⟶ before connection tear-down



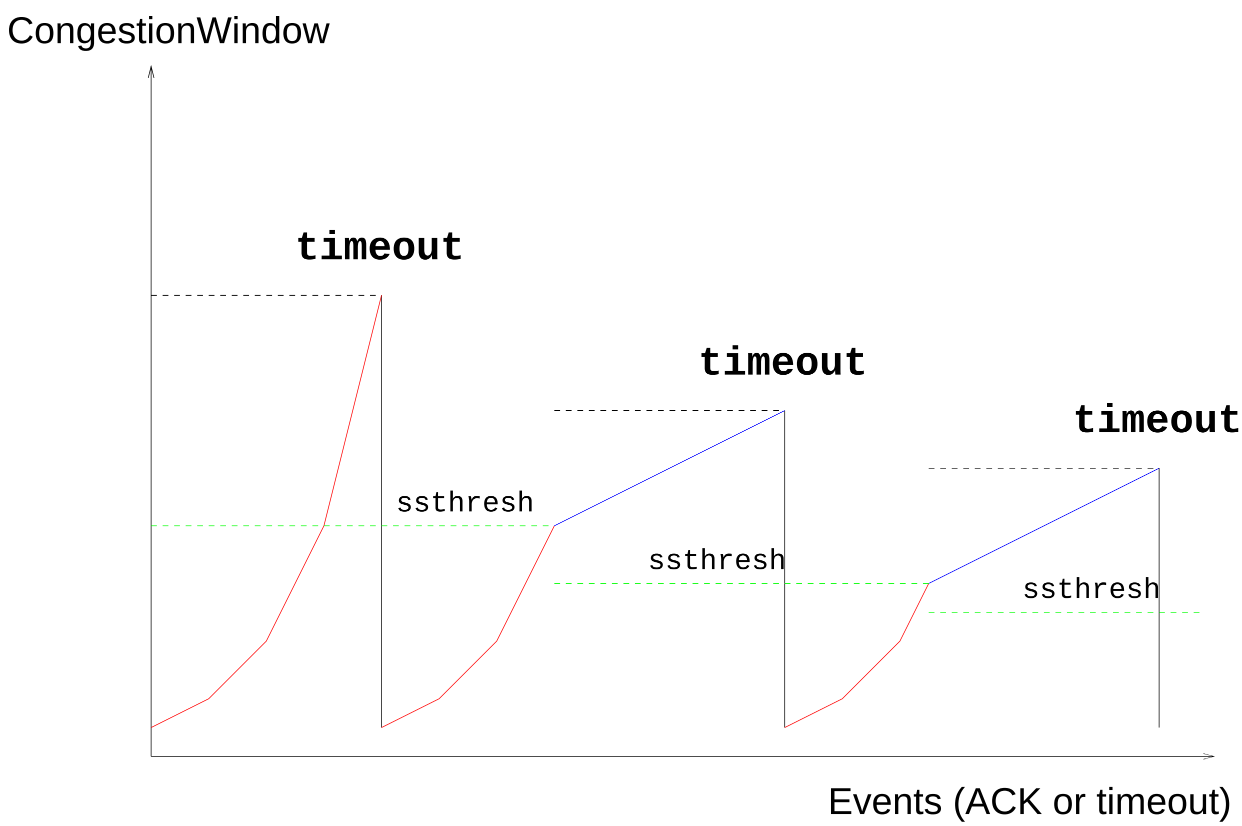
⟶ most TCP transfers are small

⟶ small files "dominate" Internet TCP connections

⟶ most TCP flows don't escape **`Slow Start`**

`CongestionWindow` evolution:

⟶ relevant for larger flows



(iii) Exponential timer backoff

$$\texttt{TimeOut} \leftarrow 2 \cdot \texttt{TimeOut} \qquad \text{if retransmit}$$

(iv) Fast Retransmit

Upon receiving three duplicate ACKs:

- Transmit next expected segment

  $\rightarrow$ segment indicated by ACK value

- Perform exponential backoff and commence Slow Start

  $\longrightarrow$ three duplicate ACKs: likely segment is lost

  $\longrightarrow$ react before timeout occurs

TCP Tahoe: features (i)-(iv)

(v) Fast Recovery

Upon Fast Retransmit:

- Skip Slow Start and commence Congestion Avoidance

  $\rightarrow$ dup ACKs: likely spurious loss

- Insert "inflationary" phase just before Congestion Avoidance

Given sawtooth behavior of TCP's linear increase/exponential backoff:

Why use exponential backoff and not Method D?

- For multimedia streaming (e.g., pseudo real-time), AIMD (Method B) is not appropriate

  $\rightarrow$ use Method D

- For unimodal case—throughput decreases when system load is excessive—story is more complicated

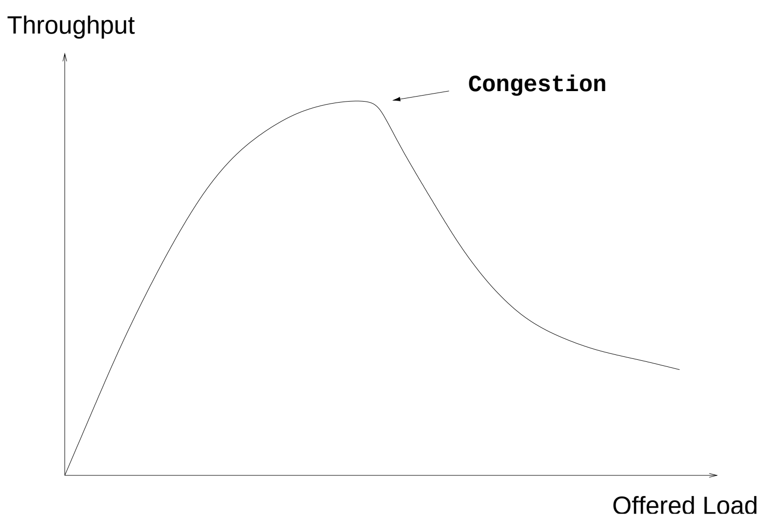
  $\rightarrow$ asymmetry in control law needed for stability

Congestion control and selfishness

$\longrightarrow$ to be or not to be selfish ...

$\longrightarrow$ John von Neumann, John Nash, ...

Ex.: "tragedy of commons," Garrett Hardin, '68



- if everyone acts selfishly, no one wins

  $\rightarrow$ in fact, everyone loses

- can this be prevented?

Ex.: Prisoner's Dilemma game

$\longrightarrow$ formalized by Tucker in 1950

$\longrightarrow$ "cold war"

- both cooperate (i.e., stay mum): 1 year each
- both selfish (i.e., rat on the other): 5 years each
- one cooperative/one selfish: 9 vs. 0 years

		Bob	
		C	N
Alice	C	5, 5	1, 9
	N	9, 1	3, 3

$\longrightarrow$ payoff matrix

$\longrightarrow$ what would "rational" prisoners do?

When cast as congestion control game:

Alice \ Bob	C	N
C	5, 5	1, 9
N	9, 1	3, 3

$\longrightarrow$ Alice and Bob share network bandwidth

$\longrightarrow$ (a, b): throughput (Mbps) achieved by Alice/Bob

$\longrightarrow$ upon congestion: back off or escalate?

$\longrightarrow$ equivalent to Prisoner's dilemma

Rational: in the sense of seeking selfish gain

$\longrightarrow$ both choose strategy "N"

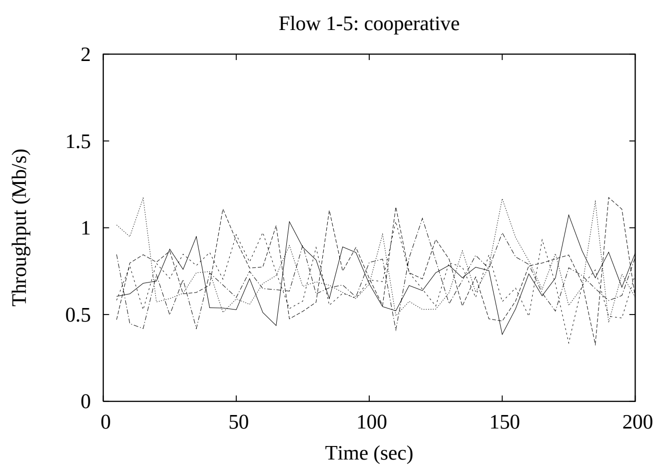
$\longrightarrow$ called Nash equilibrium

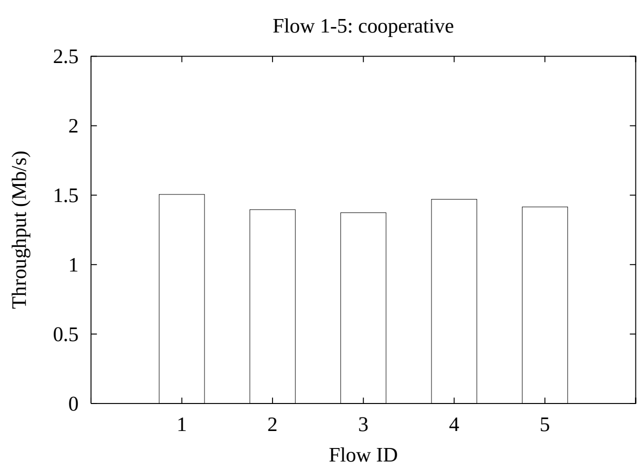
$\longrightarrow$ note: stable state

$\longrightarrow$ why: strategy "N" dominates strategy "C"

5 regular (cooperative) TCP flows:

$\longrightarrow$ share 11 Mbps WLAN bottleneck link





4 regular (cooperative) TCP flows and 1 noncooperative TCP flow:

$\longrightarrow$ same benchmark set-up