TCP congestion control:

Recall:

\[
\text{EffectiveWindow} = \text{MaxWindow} - (\text{LastByteSent} - \text{LastByteAcked})
\]

where

\[
\text{MaxWindow} = \min\{\text{AdvertisedWindow}, \text{CongestionWindow}\}
\]

Key question: how to set \text{CongestionWindow} which, in turn, affects ARQ’s sending rate?

\[\rightarrow\] linear increase/exponential decrease

\[\rightarrow\] AIMD

\[\rightarrow\] method B
TCP congestion control components:

(i) Congestion avoidance

→ linear increase/exponential decrease

→ additive increase/exponential decrease (AIMD)

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

Upon receiving ACK:

`CongestionWindow ← CongestionWindow + 1`

Upon timeout:

`CongestionWindow ← CongestionWindow / 2`

But is it correct...
'Linear increase' time diagram:

\[ \text{Sender} \quad \text{Receiver} \]

\[ \begin{array}{c|c}
\text{RTT} & \text{time} \\
1 & 1 \\
2 & 2 \\
4 & 4 \\
8 & 8 \\
16 & 16 \\
\end{array} \]

\[ \longrightarrow \] results in exponential increase
What we want:

\[ \text{RTT time} \rightarrow \text{increase by 1 every window} \]
Thus, linear increase update:

\[
\text{CongestionWindow} \leftarrow \text{CongestionWindow} + \left( \frac{1}{\text{CongestionWindow}} \right)
\]

Upon timeout and exponential backoff,

\[
\text{SlowStartThreshold} \leftarrow \text{CongestionWindow} / 2
\]
(ii) Slow Start

Reset $CongestionWindow$ to 1

Perform exponential increase

$$CongestionWindow \leftarrow CongestionWindow + 1$$

- 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

$\rightarrow$ “slow” is a misnomer

$\rightarrow$ 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:

$\rightarrow$ relevant for larger flows
(iii) Exponential timer backoff

\[ \text{TimeOut} \leftarrow 2 \cdot \text{TimeOut} \quad \text{if retransmit} \]

(iv) Fast Retransmit

Upon receiving three duplicate ACKs:

- Transmit next expected segment
  \[ \rightarrow \text{segment indicated by ACK value} \]
- Perform exponential backoff and commence Slow Start
  \[ \rightarrow \text{three duplicate ACKs: likely segment is lost} \]
  \[ \rightarrow \text{react before timeout occurs} \]

TCP Tahoe: features (i)-(iv)
(v) Fast Recovery

Upon Fast Retransmit:

- Skip Slow Start and commence Congestion Avoidance
  → 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
  → use Method D

- For unimodal case—throughput decreases when system load is excessive—story is more complicated
  → asymmetry in control law needed for stability
Congestion control and selfishness:

→ to be or not to be selfish . . .
→ John von Neumann, John Nash, . . .

Ex.: “tragedy of commons,” Garrett Hardin, ’68

- if everyone acts selfishly, no one wins
  → in fact, everyone loses
- can this be prevented?
Ex.: Prisoner’s Dilemma game

$\rightarrow$ formalized by Tucker in 1950

$\rightarrow$ "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

$\begin{array}{c|cc}
\text{Alice} & \text{Bob} \\
\hline
\text{C} & 1,1 & 9,0 \\
\text{N} & 0,9 & 5,5 \\
\end{array}$

$\rightarrow$ payoff matrix

$\rightarrow$ what would “rational” prisoners do?
When cast as congestion control game:

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>5,5</td>
</tr>
<tr>
<td>N</td>
<td>9,0</td>
</tr>
</tbody>
</table>

Alice and Bob share network bandwidth
→ \((a, b)\): throughput (Mbps) achieved by Alice/Bob
→ large is desirable

Upon congestion: back off or escalate?
→ equivalent to Prisoner’s dilemma
Rational: in the sense of seeking selfish gain
→ both choose strategy “N”
→ called Nash equilibrium
→ steady-state or stable fixed-point

Reason:
→ whatever choice the other player makes, “N” yields better payoff over “C”
→ i.e., strategy “N” dominates strategy “C”

In some systems, selfish behavior results in system optimal outcome
→ theoretical foundation of Adam Smith’s “invisible hand”
→ in general, not the case
→ cooperation is better but can it be enforced?
Impact in networks:

→ 5 regular (cooperative) TCP flows
→ share 11 Mbps WLAN bottleneck link
4 regular (cooperative) TCP flows and 1 noncooperative TCP flow:

→ starts behaving selfishly at time 100s
Potential danger for:

→ unfairness

→ overall system performance

Is it being exploited in today’s Internet?

→ no one knows

→ technical implementation issues

→ e.g., interoperability with legacy protocols

→ e.g., shooting oneself in the foot