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?

\[ \rightarrow \text{linear increase/exponential decrease} \]

\[ \rightarrow \text{AIMD} \]
TCP congestion control components:

(i) Congestion avoidance

\[ \rightarrow \text{linear increase/exponential decrease} \]

\[ \rightarrow \text{additive increase/exponential decrease (AIMD)} \]

As in Method B, increase \texttt{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:

\[ \text{results in exponential increase} \]
What we want:

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

\(\text{RTT} \quad \rightarrow \quad \text{time} \quad \uparrow \quad \text{time} \quad \uparrow \quad \text{time}\)

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

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

Upon timeout and exponential backoff,

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

Reset \texttt{CongestionWindow} to 1

Perform exponential increase

\[
\text{CongestionWindow} \leftarrow \text{CongestionWindow} + 1
\]

- Until timeout at start of connection
  
  \rightarrow \text{rapidly probe for available bandwidth}

- Until \text{CongestionWindow} hits \texttt{SlowStartThreshold}
  following Congestion Avoidance

  \rightarrow \text{rapidly climb to safe level}

\rightarrow \text{“slow” is a misnomer}

\rightarrow \text{exponential increase is super-fast}
Basic dynamics:

→ after connection set-up

→ before connection tear-down

connection start → timeout → SlowStartThreshold → timeout

Slow Start → Slow Start → Congestion Avoidance → Slow Start

repeat
CongestionWindow evolution:

Events (ACK or timeout)
(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
  \[\rightarrow\] 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
  → 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

- formalized by Tucker in 1950
- “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

<table>
<thead>
<tr>
<th></th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
</tr>
<tr>
<td>Alice</td>
<td>5, 5</td>
</tr>
<tr>
<td></td>
<td>9, 1</td>
</tr>
</tbody>
</table>

- payoff matrix
- what would “rational” prisoners do?
When cast as congestion control game:

<table>
<thead>
<tr>
<th></th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>5, 5</td>
</tr>
<tr>
<td>N</td>
<td>9, 1</td>
</tr>
<tr>
<td>N</td>
<td>3, 3</td>
</tr>
</tbody>
</table>

→ Alice and Bob share network bandwidth

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

→ upon congestion: back off or escalate?

→ equivalent to Prisoner’s dilemma
Rational: in the sense of seeking selfish gain

$\rightarrow$ both choose strategy “N”

$\rightarrow$ called Nash equilibrium

$\rightarrow$ why: strategy “N” dominates strategy “C”

Dominance: suppose Alice chooses “C”; from Bob’s perspective, choosing “N” yields 9 Mbps whereas “C” yields only 5 Mbps. Similarly if Alice were to choose “N.”

$\rightarrow$ for Bob: “N” dominates “C”

$\rightarrow$ a “no brainer” for Bob

$\rightarrow$ by symmetry, the same logic applies to Alice
Ex.: von Neumann argued for first-strike policy based on this reasoning.

- → luckily “MAD” prevailed
- → MAD: mutually assured destruction
- → sometimes “delay” is good!
In a selfish environment, the system tends to converge to a Nash equilibrium.

A Nash equilibrium is a system state where no player has an incentive to make a **unilateral** move.

- → unilateral: only one player makes a move
- → e.g.: (N,C) is not a Nash equilibrium
- → Bob gains by switching from “C” to “N”
- → Bob’s payoff increases from 1 to 3
- → Nash equilibrium is a stable state: impasse
5 regular (cooperative) TCP flows:

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

→ same benchmark set-up
Remarks:

- a Nash equilibrium need not exist
  → system subject to oscillation
  → circular “chain reaction”

- Nash’s main result (game theory): finite noncooperative games with **mixed** strategies—choose action probabilistically—always possess equilibrium
  → vs. **pure** strategy (more in tune with reality)
  → pure strategy games: hard problem

- congestion pricing
  → penalize those who congest: e.g., usage pricing
  → in the States: flat pricing (dominant)
  → not skimpy like the rest of the world!
• repeated/evolutionary games
  → e.g.: iterated Prisoner’s Dilemma
  → rob bank/get caught, again and again . . .
  → what should the prisoners do then?
  → “grim trigger” policy: don’t forgive
  → “tit-for-tat” policy: conditionally forgive
  → both are optimal (in a certain sense)
  → most relevant for “greedy” TCP