3F4: Data Transmission

Handout 12:

Transmission Control Protocol (TCP)

Congestion Control: TCP-Reno

Ioannis Kontoyiannis

Signal Processing and Communications Lab
Department of Engineering
i.kontoyiannis@eng.cam.ac.uk

Lent Term 2019

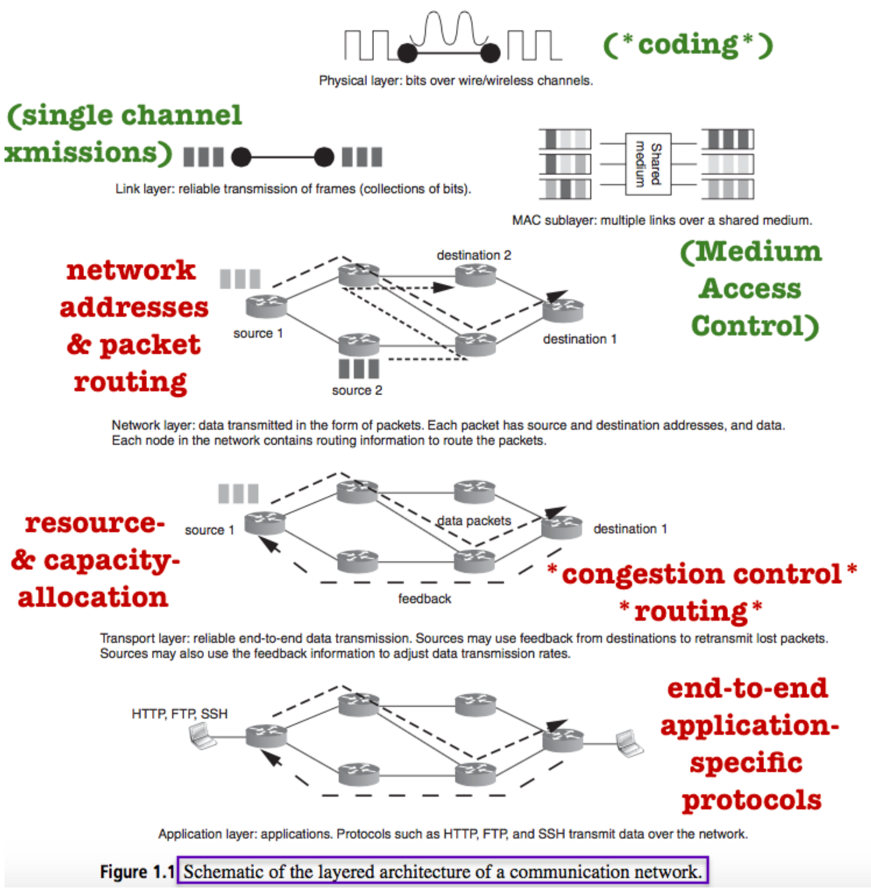


Image from: Srikant and Ying "Communication networks: an optimization, control, and stochastic networks perspective." Cambridge University Press, 2013

Transmission Control Protocol (TCP)

- End-to-end transport protocol with many parts
- Used by http (web), smtp (email), telnet,
  ftp (file transfer), chat, . . .

TCP

1. Connection establishment
2. Connection maintenance
   - Reliability
   - **Congestion control**
   - Flow control
   - Sequencing
3. Connection termination

TCP historical evolution

1974 TCP first described (Vint Cerf, Bob Kahn)

1975 Three-way handshake (Ray Tomlinson)

1981 TCP/IP (Internet Engineering Task Force Standard)

1983 BSD Unix 4.2 supports TCP/IP

1984 Nagel's algorithm predicts congestion collapse

1986 First internet **congestion collapse** observed!
Link Lawrence Berkeley Lab to UC Berkeley
400 yards, 3 hops, throughput 32 Kbps
Dropped to 40 bps: factor of $\approx$ 1000 drop!

1988 Jacobson's algorithms: TCP-Tahoe
slow start, congestion avoidance

1990 **TCP-Reno**

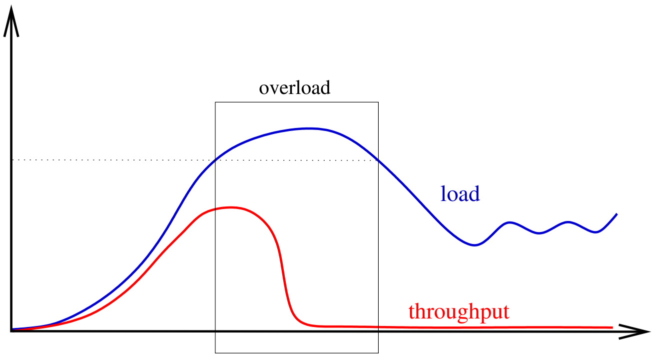
1993 TCP-Vegas

1996 TCP-NewReno

Congestion

- *"too many sources sending too much data too fast for the* **network** *to handle"*
- manifests as **packet loss** (buffer overflow at routers) or **large delay** (long queues at router buffers)
- a very serious issue

Effects of congestion:



- Packet loss
- Retransmission
- Reduced throughput
- Congestion collapse due to
  - Unnecessary retransmissions
  - Undelivered packets
- Congestion may continue after the overload!

Congestion control goals

- High network utilization
- Maximize throughput = total # of bits end-end
- Avoid congestion collapse
- Fair bandwidth sharing
  - Give different sessions equal share
  - Max-min fairness: Maximize the minimum rate session

TCP approach to congestion control

- no explicit feedback from network: closed-loop, end-to-end congestion control
- congestion inferred from end-system observed packet loss and delay
- works well so far: the bandwidth of the Internet has increased by > 5 orders of magnitude

TCP-Reno

One of the most common congestion control algorithms used in the Internet today

It consists of several important elements that are part of almost all common congestion control algorithms

adaptive window flow control

slow start phase

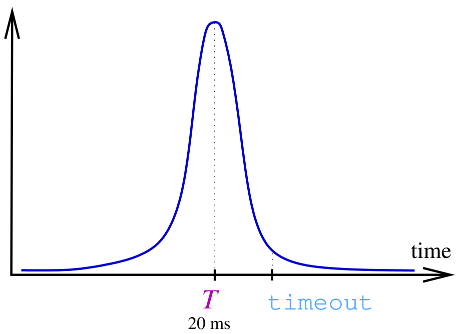
congestion avoidance phase

fast retransmit

fast recovery

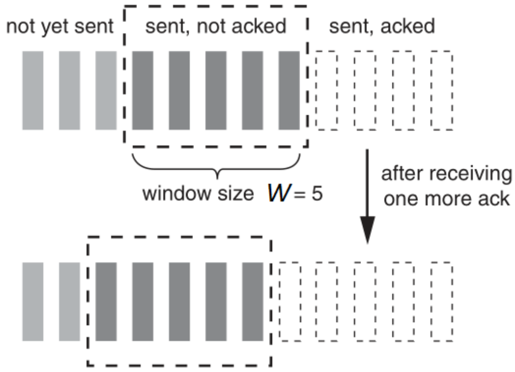
TCP-Reno: Preliminaries

- Source sends packets to receiver
- Receiver responds with an acknowledgment `ack` every time a packet is received
- Let T denote the RTT: $T =$ round-trip time between packet being sent and `ack` received
- Make repeated measurements of time from packet transmission until `ack` receipt
- Compute empirical mean μ and standard deviation σ
- Set $T = \mu$
- Set `timeout` $= \mu +$ (a few)σ



TCP-Reno: Window flow control and ack's

- Source maintains a window containing a varying number of W packets
- Every time an ack is received, window shifts by 1 position
- An ack contains the index of the next packet the receiver expects. E.g., if it's received packets 1,2, upon receiving packet 3 the next ack asks for packet 4
- If then packet 5 is received next ack asks for packet 4 again
- This is called a 'duplicate acknowledgment' or dupack



TCP-Reno: Initialization

Initialize:

- Set window size W =1
- Estimate RTT T
- Select slow start threshold value ssthresh for W

Basic TCP-Reno operation:

- Send all W packets currently in the window
- Wait T seconds for the corresponding ack's
- Update $W \mapsto W'$ and ssthresh
- Send next W' packets in the new window

Transmission rate:

- At time t, the rate $R(t) = \frac{W(t)}{T}$ packets/second

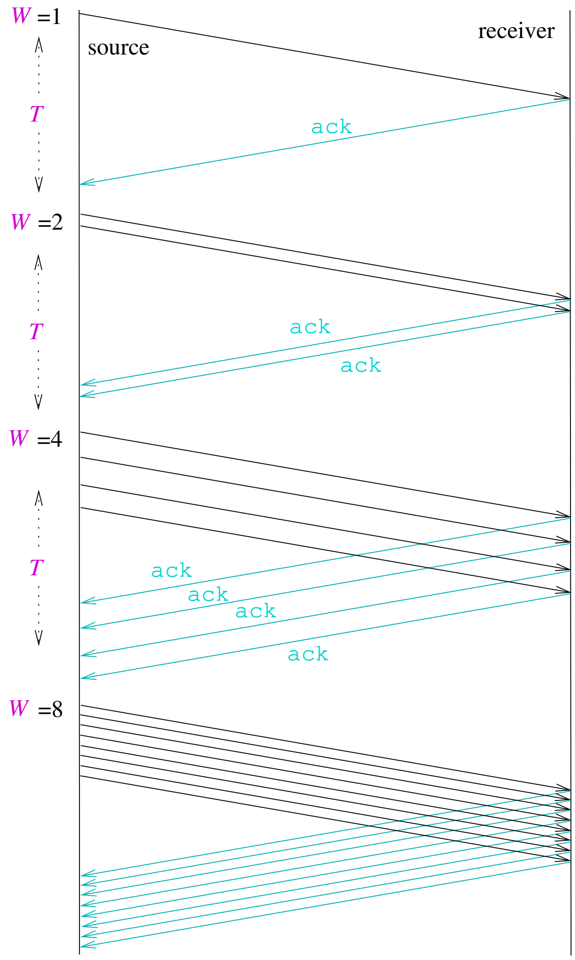
TCP-Reno: Slow start phase

Idea: Increase W by 1
with each ack

While all is going well:

- Send $W = 1$ packet
  receive ack, set $W = 2$
- Send $W = 2$ packets
  receive 2 ack's, set $W = 4$
- Send $W = 4$ packets
  receive 4 ack's, set $W = 8$
- Continue until
  $W >$ ssthresh

When $W >$ ssthresh
enter congestion avoidance phase



TCP-Reno: Congestion avoidance phase

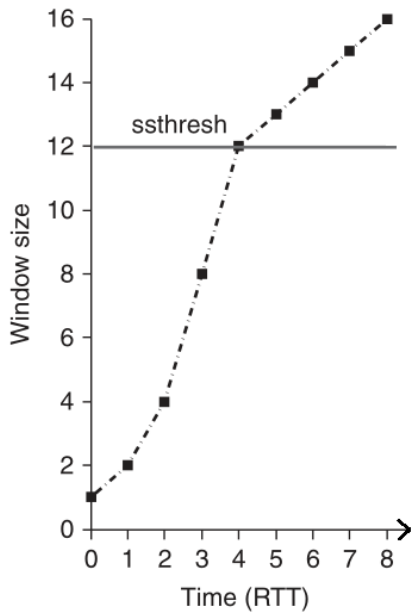
Idea: Increase W
by $1/W$ with each ack

While all is going well:

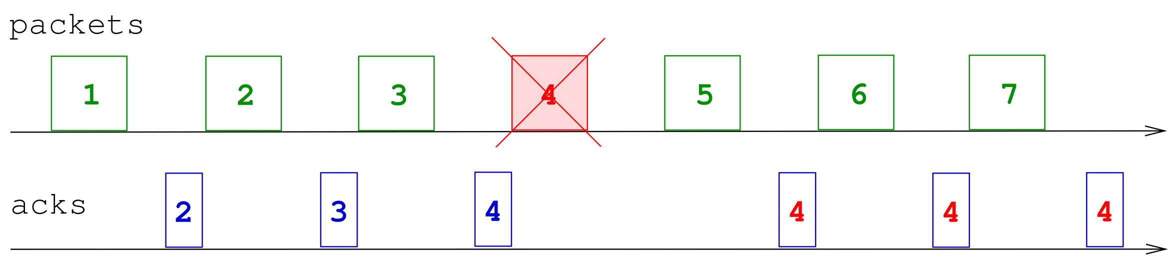
- Send W packets
  receive W ack's
  set $W \mapsto W + 1$
- Continue until
  something goes wrong

What can go wrong?

- **Delay** = three dupack's
- **Loss** = timeout



TCP-Reno: Three `dupack`'s ⇒ 'Fast Retransmit'



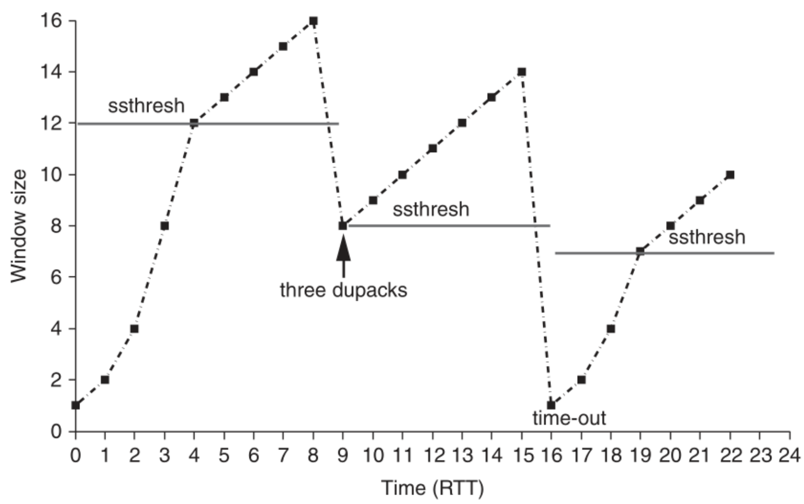
Transmitter detects **delay**:

- If sender receives 3 `dupack`'s for the same data it assumes that packet was irreparably delayed
- Resend missing packet before `timeout` (hence 'fast retransmit')
- Change the value of `ssthresh` to $W/2$
- Set the window size $W \mapsto W/2$
- Continue as before

TCP-Reno: `timeout` ⇒ 'Fast Recovery'

Transmitter detects **loss**:

- If sender receives no `ack`'s for any of the packets sent from the window by `timeout`, it assumes they are permanently lost
- Change the value of `ssthresh` to $W/2$
- Set window size $W = 1$
- Return to slow start phase: Start by resending lost packets



TCP-Reno: Observations

Revisit five TCP-Reno elements:

- Variable **window size** regulates transmission speed
- **Slow start**: Multiplicative (i.e., exponential) rate increase until `ssthresh`/T packets/second
- **Congestion avoidance**: Additive (i.e., linear) rate increase until delay/loss detected
- **Fast retransmit** when delay detected
- **Fast recovery** when `timeout`/loss detected

Observe:

- Loss (`timeout`) is treated as a more serious issue than delay (3 `dupack's`)
- In both cases, multiplicative rate decrease
- In the long run, TCP-Reno spends almost all of the time in the congestion avoidance phase
- Additive Increase Multiplicative Decrease (AIMD) adaptation

TCP-Reno: Equilibrium rate

What is the algorithm's
long-term average transmission rate $R(t)$?

- Recall rate $R(t) = W(t)/T$
- Assume in CA phase (ignore SS)
- Consider a continuous-time approximate model
- Let $q(t)$ = loss rate at time t

In the long run, how does $R(t)$ depend on $q(t)$?

Strategy:

- Compute the rate of change $W'(t)$ of the window size
- Assume that, at equilibrium, $R'(t) = W'(t)/T = 0$
- Solve $R'(t) = 0$

TCP-Reno: Continuous-time model

- rate at which `ack`'s are received is $R(t-T)(1-q(t))$
- each `ack` increases $W(t)$ by $1/W(t)$
- packet loss rate is $R(t-T)q(t)$
- each loss event decreases $W(t)$ by a factor $\beta = 1/2$

Therefore:

$$W'(t) = \frac{R(t-T)(1-q(t))}{W(t)} - \beta R(t-T)q(t)W(t)$$

i.e.,

$$R'(t) = \frac{R(t-T)(1-q(t))}{T^2 R(t)} - \beta R(t-T)q(t)R(t)$$

and solving $R'(t) = 0$:

TCP square-root law

$$R(t) = \frac{1}{T\sqrt{\beta}}\sqrt{\frac{1-q(t)}{q(t)}}$$

TCP-Reno in equilibrium: Conclusions

$$R(t) = \frac{1}{T\sqrt{\beta}}\sqrt{\frac{1-q(t)}{q(t)}}$$

- Rate $R(t) \propto 1/\text{RTT}$
- When the loss probability $q(t)$ is small the rate $R(t) \propto 1/\sqrt{q(t)}$
- Results extensively verified in empirical observations E.g., 3 groups of 10 sources each, 3ms $\leq$ RTT $\leq$ 7ms, link capacity = 64Mbps