Flavors of TCP Congestion Control

- **TCP Tahoe** (1988, FreeBSD 4.3 Tahoe)
  - Slow Start
  - Congestion Avoidance
  - Fast Retransmit
- **TCP Reno** (1990, FreeBSD 4.3 Reno)
  - Fast Recovery
- **New Reno** (1996)
- **SACK** (1996)
- **RED** (Floyd and Jacobson, 1993)
- **TCP Vegas** (S. Low, L. Peterson, L. Wang, 2000)

Flavors of TCP Congestion Control

- **TCP Tahoe** (1988, FreeBSD 4.3 Tahoe)
  - Slow Start
  - Congestion Avoidance
  - Fast Retransmit
- **TCP Reno** (1990, FreeBSD 4.3 Reno)
  - Fast Recovery
- **New Reno** (1996)
- **SACK** (1996)
- **RED** (Floyd and Jacobson, 1993)
- **TCP Vegas** (S. Low, L. Peterson, L. Wang, 2000)

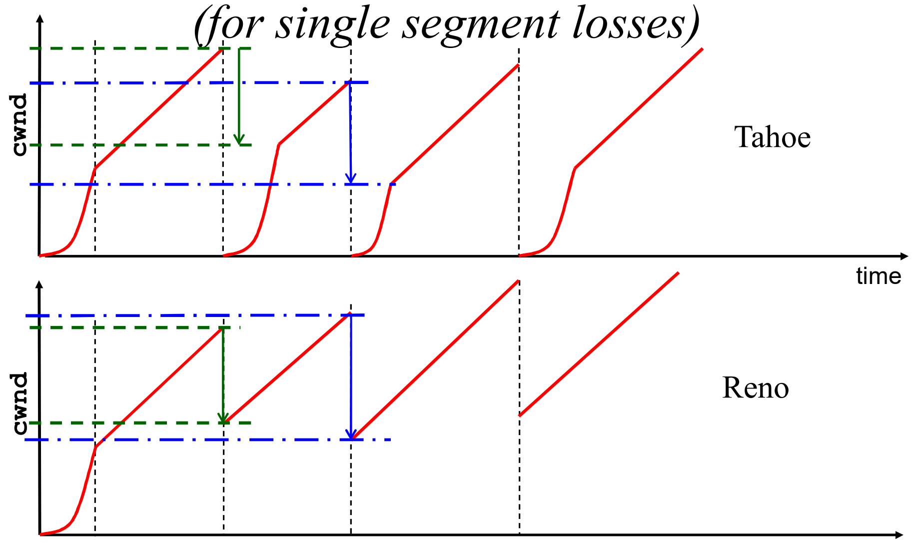
TCP Reno

- Duplicate ACKs:
  - Fast retransmit
  - Fast recovery

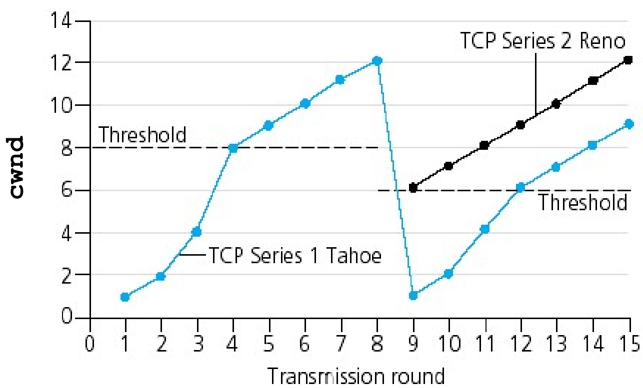
  → Fast Recovery avoids slow start
- Timeout:
  - Retransmit
  - Slow Start
- TCP Reno improves upon TCP Tahoe when a single packet is dropped in a round-trip time.
- Fast recovery avoids slow start after a fast retransmit
- Intuition: Duplicate ACKs indicate that data is getting through
- On packet loss detected by three duplicate ACKs:
  - **`ssthresh = cwnd/2`**
  - **`cwnd = ssthresh`**

  enter congestion avoidance

TCP Tahoe and TCP Reno (for single segment losses)



TCP CC



SACK

- SACK = Selective ACKnowledgment
- A Problem: Reno (and New Reno) retransmit at most 1 lost packet per round trip time
- Selective acknowledgments: The receiver can acknowledge non-continuous blocks of data (e.g., SACK 0-1023, 1024-2047)
- Multiple blocks can be sent in a single segment.
- TCP SACK:
  - Enters fast recovery upon 3 duplicate ACKs
  - Sender keeps track of SACKs and infers if segments are lost. Sender retransmits the next segment from the list of segments that are deemed lost.

Congestion Avoidance

- TCP's strategy
  - control congestion once it happens
  - repeatedly increase load in an effort to find the point at which congestion occurs and then back off
- Alternative strategy
  - predict when congestion is <u>about to happen</u>
  - reduce rate before packets start being discarded
  - call this ***congestion avoidance***, instead of ***congestion control***
- Two possibilities
  - host-centric: TCP Vegas
  - router-centric: DECbit and RED Gateways

Congestion Avoidance in TCP (Intro to TCP Vegas)

- Source watches for some sign that router's queue is building up and congestion will happen; e.g.,
  - **`RTT`** grows (e.g., if the current **`RTT`** > average of min and max **`RTT`** → degreases **`CongestionWindow`** by 1/8)
  - sending rate flattens; every **`RTT`** the window is increased by 1 and throughput compared (throughput = (#outstanding bytes)/**`RTT`**)
  - TCP Vegas is similar to this last strategy, but compares the measured and expected rates.
  - Still uses multiplicative decrease when congestion occurs (packets are dropped.

TCP Vegas Algorithm

- Let **`BaseRTT`** be the minimum of all measured **`RTT`**s (commonly the RTT of the first packet)
- If not overflowing the connection, then

  `ExpectRate = CongestionWindow/BaseRTT`
- **`CongestionWindow`** = number of bytes in transit
- Source calculates sending rate (**`ActualRate`**) once per **`RTT`**
- Source compares **`ActualRate`** with **`ExpectRate`**

  `Diff = ExpectRate – ActualRate` (note that **`Diff`** ≥ 0)
- Larger **`Diff`** implies more congestion in the network

```
if Diff < α                                      (α < β)
    increase CongestionWindow linearly in next RTT
else if Diff > β
    decrease CongestionWindow linearly in next RTT
else                                             (α < Diff < β)
    leave CongestionWindow unchanged
```

Challenges of Internet CC

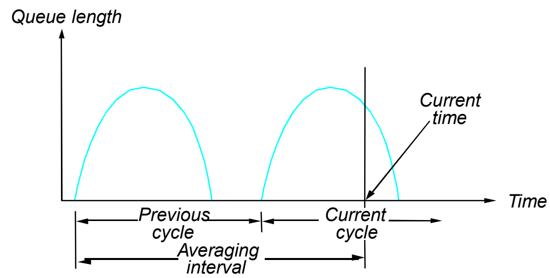
- The main culprit: bursty traffic!
- Main difficulty: RTT may be long (~100msec)
- Full queues
  - Routers are forced to have large queues to maintain high utilizations
  - TCP detects congestion from loss
    - Forces network to have long standing queues in steady-state
    - Remember: the mechanism responds to congestion, rather than prevents it from occuring
- Lock-out problem
  - Drop-tail routers treat bursty traffic poorly
  - Traffic gets synchronized easily

The DECbit Scheme

- This was used in an early computer network architecture by DEC – it is not used in the Internet today
  - A similar idea in the Internet is Early Congestion Notification
- Basic ideas:
  - On congestion, router sets congestion indication (CI) bit in packets
  - Receiver relays bit to sender
  - Sender adjusts sending rate
- Key design questions:
  - When to set CI bit?
  - How does sender respond to CI?

DECbit

- Router
  - monitors average queue length over last “busy + idle” cycle and the current busy cycle.



  - set congestion bit if average queue length > 1

DECBit

❖ End Hosts:
  - ➢ Destination echoes CI bit back to source
  - ➢ Source records how many packets resulted in set bit during the past congestion window interval
  - ➢ If less than 50% of last window’s worth of packets had bit set → increase ***CongestionWindow*** by 1 packet
  - ➢ If 50% or more of last window’s worth of packets had bit set → decrease ***CongestionWindow*** by 0.875 times

❖ Discussion
  - ➢ Relatively easy to implement
  - ➢ No per-connection state
  - ➢ Stable (additive increase / multiplicative decrease)
  - ➢ Assumes cooperative sources
  - ➢ Conservative window increase policy

RED: Random Early Detection

- RED Design Objectives
  - Keep throughput high and delay low
  - Accommodate bursts
  - Queue size should reflect ability to accept bursts rather than steady-state queuing
  - Improve TCP performance with minimal hardware changes
- Alternate Solutions to the Lock-out Problem
  - Random drop
    - Packet arriving when queue is full causes some random packet to be dropped
  - Drop front
    - On full queue, drop packet at head of queue
  - Random drop and drop front solve the lock-out problem but not the full-queues problem

RED: Random Early Detection

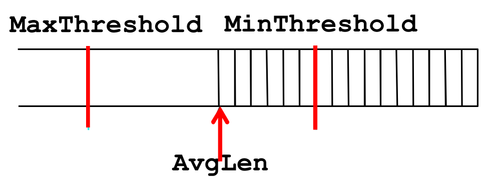
- Full queues problem: Drop packets before queue becomes full (early drop)
- Intuition: notify senders of imminent congestion
- A "possible" simplistic scheme:
  - If qlen > drop level, drop each new packet with fixed probability p (fixed level, fixed probability of drop)
    - Will not control congestion in a gradual way
    - May not respond well to temporal traffic variations
    - Does not control misbehaving users
    - Not friendly to bursty traffic

RED Details

- 1. Compute average queue length

```
AvgLen = (1 - w_q) * AvgLen + w_q * SampleLen
    0 < w_q < 1
```

  - `SampleLen` is queue length each time a packet arrives
  - w_q is a time constant of the above filter; should be such that changes to queue length over time scale of < RTT are filtered out
  - Typical $w_q = 0.002$

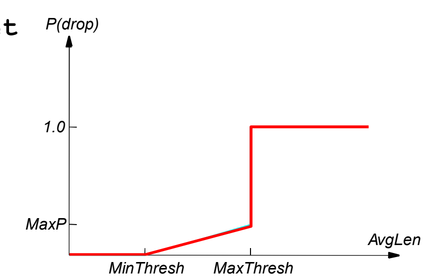


RED Details

- 2. The two queue length thresholds

```
if AvgLen <= MinThreshold then
    queue the packet
if MinThreshold < AvgLen < MaxThreshold then
    calculate probability P
    drop arriving packet with probability P
if MaxThreshold <= AvgLen then
    drop arriving packet
```

Drop Probability curve



RED Details

- 3. Computing probability P

```
TempP = MaxP * (AvgLen - MinThreshold)/
      (MaxThreshold - MinThreshold)

P = TempP/(1 - count * TempP)
```

  - The first equation sizes the probability of dropping, while the second equation increases the probability with the received packets.
  - Marking only based on **`TempP`** can lead to clustered marking
  - The above procedure better spreads the marked packets.
  - Example: **`MaxP=0.02; count=0, AvgLen; P=0.01;`** 1% dropping probability. If **`count=50`** with no packet drop, **`P=0.02.`** If **`count=99`** with no packet drop, **`P=1.0.`**

Tuning RED

- The probability of dropping a particular flow's packet(s) is roughly proportional to the share of the bandwidth that the flow is currently getting.
- **`MaxP`** is typically set to 0.02, meaning that when the average queue size is halfway between the two thresholds, the router drops roughly one out of 50 packets.
- If traffic is bursty, then **`MinThreshold`** should be sufficiently large to allow link utilization to be maintained at an acceptably high level
- Difference between the two thresholds should be larger than the typical increase in the calculated average queue length in one **`RTT`**; setting **`MaxThreshold`** to twice **`MinThreshold`** is reasonable for traffic on today's Internet
- Penalty for offenders

Explicit Congestion Notification

- Similar, in principle, to *DECBit*
- Proposed for standard TCP
- Recall that:
  - TCP uses packet losses to detect congestion
  - Wasteful and unnecessary
- ECN (RFC 2481)
  - Routers mark packets instead of dropping them
  - Receiver returns marks to sender in ACK packets
  - Sender adjusts its window accordingly
- Two bits in IP header (TOS field) to implement
  - ECT: ECN-capable transport (set to 1)
  - CE: congestion experienced (set to 1)
- Source responds to a set ECN bit as a dropped packet.

