CC2: Review of TCP’s CC

- **Goal:** Fair efficient sharing of links
- **Steps in TCP:** 3WH, Exchange, 1st ½ Close; 2nd ½ Close
- **Slow Start:**
  - Exp. Increase of Window to discover bw (W = W + 1 …)
  - TO → ssthresh = W/2
  - Repeat until W = ssthresh, then CA
- **Congestion Avoidance:**
  - AIMD: + 1 MSS/RTT (W = W + 1/W…); W/2 when 3DA
- **Timeout:** mean + 4 deviations
  - double when triggered, then W = 1 & SS
  - reset after new ACK or new packet
- **Flow Control:** min(RAW – OUT, W)

CC2: Cheating TCP

- **Some Methods:**
  - Increase faster than 1/RTT per RTT
  - Start SS with W > 1
  - Open many connections
- **Why Not?**
Cheating: Start SS with $W > 1$

- $x$ starts SS with $W = 4$
- $y$ starts SS with $W = 1$

Cheating: Open Many Connections

Assume
- $A$ starts 10 connections to $B$
- $D$ starts 1 connection to $E$
- Each connection gets about the same throughput

Then $A$ gets 10 times more throughput than $D$

Cheating: Why Not?

- Assume
  - $A$ starts 10 connections to $B$
  - $D$ starts 1 connection to $E$
  - Each connection gets about the same throughput

Then $A$ gets 10 times more throughput than $D$

CC2: Explicit Congestion Notification

- Standard TCP:
  - Losses needed to detect congestion
  - Wasteful and unnecessary
- ECN:
  - Routers mark packets instead of dropping them
  - Destination marks ACKs of marked packets
  - Source set $W = W/2$ when it sees mark
- Advantages:
  - No time wasted to retransmit
  - Link errors not confused with congestion
- Illustration
- Backward Compatibility

CC2: Explicit Congestion Notification

Illustration:

- Bit in Header indicates if hosts implement ECN
- If it does, router marks packet
- If it does not, router drops packet
CC2: Noisy Link

- Basic TCP assumption:
  - Losses indicate congestion
  - What is losses come from link errors?
  - Slowing down does not help; in fact it hurts

Illustration

Solutions

Noisy Link: Illustration

- Illustration

- Solutions

Noisy Link: Solutions

- Link Error Control
- ECN
- Link Layer Hint
- Various other schemes ...

Noisy Link: Solutions: Link Error Control

- Noisy link implements a retransmission protocol

Noisy Link: Solutions: Link Layer Hint

- Link Layer knows when an error occurred
- It indicates that event in next packet
- Destination reflects that fact in ACK
- Source retransmits but does not reduce W

Wireless router

Link error caused drop
**CC2: Virtual Queues**

- **Motivation:**
  - Detect impending congestion before it leads to a queue build-up
  - Reduce losses, delays, need for storage in routers

- **Mechanism:**
  - Construct a virtual queue that mimics the real queue when served with a fraction of the link rate

**Illustration**

- Virtual queue “detects” when link utilization exceeds 90%, even though the data queue is empty
- The router can use this indication to mark packets in ECN
- Data queues remain almost empty, links used at 90%

---

**CC2: RED**

Random Early Detection:
As queue builds up, drop or mark packets with increasing probability (before queue gets full)

- **Advantages:**
  - Avoids penalizing streams with large bursts
  - De-synchronizes the source behaviors

**Illustration**

- Calculate recent average of queue length: \( Q_{av}(n+1) = (1 - b)Q_{av}(n) + Q(n) \)
- Determine drop or mark probability \( p(Q_{av}) \):

**Note: Exponential Averaging**

\[ A(n+1) = (1 - b)A(n) + bX(n) \]

**Router Buffers: Rule of Thumb**

- Imagine that all connections on input port with rate \( R \) “burst” for RTT seconds (until stopped by RED)
- Router must store \( RxRTT \) for each port

**Example:**
- 40 Gbps throughput (sum of port rates)
- RTT = 200 ms (worst case?)
- Then storage = 8 Gbits = (about) 1 GByte

**Question:** Is this reasonable?
CC2: Unfairness

- Fact:
  - TCP favors connections with short RTT

- Cause:
  - Increase rate is 1 MSS/RTT, so that it is faster for connections with small RTT
  - Recall our discussion of "Cheating RTT"
  - It is quite possible for a connection to get only a few percent of its fair share

- Solutions:
  - Modify TCP to increase in proportion to RTT?
  - TCP Vegas (see next)

CC2: TCP Vegas

- Consider one queue:
  \[ \text{Assume both connections have the same backlog} \]
  Then they have the same throughput ...

- Vegas Algorithm:
  \[ Q = \text{OUT} - \text{rate}.\text{RTT} \]
  If \( Q > 3 \text{ MSS} \), then slow down; otherwise, speed up

- Problem: Reno clobbers Vegas (unlike in real life)

CC2: Fast Links

- Imagine a 10 Gbps link
- Assume MSS = 10kbits, RTT = 100 ms

- Slow Start:
  - After n RTT, window = \( 2^n - 1 \)10kbits, so that the rate is \( 2^{n-1}1000\text{kbps} \)
  - The rate reaches 10Gbps when \( 2^{n-1} = 10^9 \), i.e., when \( n = 17 \), which takes about 2 seconds

- CA:
  - If rate must increase by 5 Gbps after 3DA, this takes an increase of \( k \) MSS where \( k \cdot 10^9 \text{bits} = 5 \text{Gbps} \) \( \Rightarrow k = 5 \cdot 10^4 \), which takes forever

- Some proposed solutions: Vegas, probe

CC2: Fast Links - Probe

- Probe for bandwidth of slowest link
- Example: Packet Pair:
  \[ P/T = 5\text{Mbps} \]