Unit 22

Congestion Control (Part II)

Acknowledgments: These slides were originally developed by Prof. Jean Walrand for EE122. The past and current EE122 instructors including Profs. Kevin Fall, Abhay Parekh, Shyam Parekh, and Adam Wolisz have contributed to their evolution.
Quick Review of TCP’s CC
Cheating TCP
ECN
Noisy Links
Virtual Queues
RED
How Big are Router Buffers?
Unfairness
TCP Vegas
Fast Links
**CC2: Review of TCP’s CC**

- **Goal:** Fair efficient sharing of links
- **Steps in TCP:** 3WH, Exchange, 1<sup>st</sup> ½ Close; 2<sup>nd</sup> ½ Close
- **Slow Start:**
  - Exp. Increase of Window to discover bdw \( W = W + 1 \) ...
  - TO \( \to \) ssthresh = \( W/2 \)
  - Repeat until \( W = \text{ssthresh} \), then CA
- **Congestion Avoidance:**
  - AIMD: \( + 1 \text{ MSS/RTT} \) \( W = W + 1/W \) ; \( W/2 \) when 3DA
- **Timeout:** mean + 4 deviations
  - double when triggered, then \( W = 1 \) & SS
  - reset after new ACK (i.e., a new RTT measurement)
- **Flow Control:** \( \min\{\text{RAW, } W\} \)
CC2: Cheating TCP

Some Methods:
- **Increase faster** than 1/RTT per RTT
- **Start SS with** \( W > 1 \)
- **Open many connections**

Why Not?
Cheating: Increase Faster

x increases by 2 MSS/RTT per RTT
y increases by 1 MSS/RTT per RTT

Limit rates: $x = 2y$
Cheating: Increase Faster

C = 50

A → B
D → E

X

Y

0
10
20
30
40
50
60

1
28
55
82
109
136
163
190
217
244
271
298
325
352
379
406
433
460
487

EECS 122
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?

One shot: A has an incentive to "cheat"
Long Term: Users have an incentive not to cheat

<table>
<thead>
<tr>
<th></th>
<th>Increases by 1</th>
<th>Increases by 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>22, 22</td>
<td>A increases by 1</td>
<td>D increases by 1</td>
</tr>
<tr>
<td>35, 10</td>
<td>A increases by 5</td>
<td>D increases by 5</td>
</tr>
<tr>
<td>10, 35</td>
<td>B increases by 1</td>
<td>E increases by 1</td>
</tr>
<tr>
<td>15, 15</td>
<td>B increases by 5</td>
<td>E increases by 5</td>
</tr>
</tbody>
</table>

Too aggressive → Losses → Throughput falls
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:
CC2: Explicit Congestion Notification

Backward Compatibility:

- 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 if losses come from link errors?
  - Slowing down does not help; in fact it hurts

Illustration

Solutions
**Noisy Link: Illustration**

- No congestion → $x$ increases by one packet/RTT every RTT
- Congestion → decrease $x$ by factor 2

![Diagram with X and C = 50 pkts/RTT]
**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: ECN

- If all the routers implement ECN, then a link error is not confused with congestion.
- In that case, a 3DA would indicate a link error and the source should not reduce the window size.
- Not applicable today ....
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 $W$

Diagram:

```
               Wireless router
               n - 1
               
               n + 1

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**
**CC2: Virtual Queues: 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%.
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
CC2: RED: Illustration

- Calculate recent average of queue length: $Q_{av}$
  
  $Q_{av}(n+1) = (1 - b)Q_{av}(n) + bQ(n)$

- Determine drop or mark probability $p(Q_{av})$:

![Graph showing the relationship between $Q_{av}$ and drop probability](image)
Note: Exponential Averaging

\[ A(n+1) = (1 - b)A(n) + bX(n) \]
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?
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? Problem: Estimate of RTT is noisy
- TCP Vegas (see next)
CC2: TCP Vegas

- Don’t rely on packet losses to control windows
- Idea: Maintain enough window to obtain maximal throughput

Vegas Algorithm:
- Compute \( \text{Diff} = \text{Expected Rate} - \text{Actual Rate} \)
  where Expected Rate uses the minimum RTT observed
- If Diff is too small, increase the window size; and if Diff is too large, reduce the window size
- This is equivalent to attempt to control the backlog due to this connection within a range
- Problem: Reno clobbers Vegas (unlike in real life!!)
Imagine a 10 Gbps link
- Assume MSS = 10kbits, RTT = 100 ms
- Slow Start:
  - After n RTT (before the last update), window = $2^{n-1} \times 10\text{kbits}$, so that the rate is $2^{n-1} \times 100\text{kbps}$
  - The rate reaches 10Gbps when $2^{n-1} = 10^5$, 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 $k \times 10\text{kbits}/0.1 = 5\text{Gbps}$, i.e., $k = 5 \times 10^4$, which takes forever

Some proposed solutions: Vegas, probe
CC2: Fast Links - Probe

- Probe for bandwidth of slowest link
- Example: Packet Pair:

10Mbps 5Mbps 100Mbps 10Mbps

\[ \frac{P}{T} = 5\text{Mbps} \]