CS 268: Lecture 7
(Beyond TCP Congestion Control)

Ion Stoica
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720-1776

(Based on slides from R. Stallings, M. Handley and D. Katabi)

Outline

- TCP-Friendly Rate Control (TFRC)
  - ATM Congestion Control
  - eXplicit Control Protocol

TCP-Friendly

- Any alternative congestion control scheme needs to coexist with TCP in FIFO queues in the best-effort Internet, or be protected from TCP in some manner.

- To co-exist with TCP, it must impose the same long-term load on the network:
  - No greater long-term throughput as a function of packet loss and delay so TCP doesn’t suffer
  - Not significantly less long-term throughput or it’s not too useful

TFRC: General Idea

Use a model of TCP’s throughout as a function of the loss rate and RTT directly in a congestion control algorithm.

- If transmission rate is higher than that given by the model, reduce the transmission rate to the model’s rate.
- Otherwise increase the transmission rate.

TCP Modelling: The "Steady State" Model

The model: Packet size \( b \) bytes, round-trip time \( t \) secs, no queue.
- A packet is dropped each time the window reaches \( W \) packets.
- TCP’s congestion window:

  \[
  W, \frac{W}{2} + 1, \frac{W}{2} + 2, ..., W, \frac{W}{2} ...
  \]

  - The maximum sending rate in packets per roundtrip time: \( W \)
  - The maximum sending rate in bytes/sec: \( W B / R \)
  - The average sending rate \( T \):
  - \( T = (3/4)W B / R \)
  - The packet drop rate \( p \):
  - \( p = \frac{1}{8W^2} \)
  - The result:

An Improved "Steady State" Model

A pretty good improved model of TCP Reno, including timeouts, from Padhye et al, Sigcomm 1998:

\[
T = \frac{s}{R \sqrt{\frac{2\pi}{3} + t_{RTO} \left(3 \sqrt{\frac{3\pi}{8}} \right) p(1 + 32p^2)}}
\]

\( T \): sending rate in bytes/second
\( R \): round trip time
\( p \): fraction of packets lost
\( t_{RTO} \): TCP retransmission timeout

Would be better to have a model of TCP SACK, but the differences aren’t critical.
TFRC Details

- The devil’s in the details
  - How to measure the loss rate?
  - How to respond to persistent congestion?
  - How to use RTT and prevent oscillatory behavior?
- Not as simple as first thought

TFRC Performance (Simulation)

TFRC Performance (Experimental)

Outline

- TCP-Friendly Rate Control (TFRC)
  > ATM Congestion Control
  > eXplicit Control Protocol

ATM Congestion Control

- Credit Based
  - Sender is given “credit” for number of octets or packets it may send before it must stop and wait for additional credit.
- Rate Based
  - Sender may transmit at a rate up to some limit.
  - Rate can be reduced by control message.

Case study: ATM ABR congestion control

ABR: available bit rate:
- ‘elastic service’
- if sender’s path “underloaded”:
  - Sender should use available bandwidth
- if sender’s path congested:
  - Sender throttled to minimum guaranteed rate

RM (resource management) cells:
- Sent by sender, interleaved with data cells
- Bits in RM cell set by switches (“network-assisted”)
- NI bit: no increase in rate (mild congestion)
- CI bit: congestion indication
- RM cells returned to sender by receiver, with bits intact
**Explicit Case Study: ATM ABR Congestion Control**

- Two-byte ER (explicit rate) field in RM cell
  - Congested switch may lower ER value in cell
  - Sender’s send rate thus minimum supportable rate on path
- EFCI bit in data cells: set to 1 in congested switch
  - If data cell preceding RM cell has EFCI set, sender sets CI bit in returned RM cell

**ABR Cell Rate Feedback Rules**

- if CI == 1
  - Reduce ACR to a value >= MCR
- else if NI == 0
  - Increase ACR to a value <= PCR

- if ACR > ER
  - set ACR = max(ER, MCR)

**Outline**

- TCP-Friendly Rate Control (TFRC)
- ATM Congestion Control
  - eXplicit Control Protocol

**TCP Congestion Control Performs Poorly as Bandwidth or Delay Increases**

- Shown analytically in [Low01] and via simulations
- Because TCP lacks fast response
  - Spare bandwidth is available → TCP increases by 1 pkt/RTT even if spare bandwidth is huge
  - When a TCP starts, it increases exponentially → Too many drops → Flows ramp up by 1 pkt/RTT, taking forever to grab the large bandwidth

**Solution: Decouple Congestion Control from Fairness**

- High Utilization; Small Queues; Few Drops
- Bandwidth Allocation Policy

**Coupled because a single mechanism controls both**

- **Example:** In TCP, Additive-Increase Multiplicative-Decrease (AIMD) controls both

**How does decoupling solve the problem?**

1. **To control congestion:** use MIMD which shows fast response
2. **To control fairness:** use AIMD which converges to fairness
Characteristics of XCP Solution

1. Improved Congestion Control (in high bandwidth-delay & conventional environments):
   - Small queues
   - Almost no drops
2. Improved Fairness
3. Scalable (no per-flow state)
4. Flexible bandwidth allocation: min-max fairness, proportional fairness, differential bandwidth allocation,…

How does XCP Work?

XCP: An eXplicit Control Protocol

1. Congestion Controller
2. Fairness Controller

How Does an XCP Router Compute the Feedback?

- **MIMD**
  - Algorithm:
  - Aggregate traffic changes by $\Delta$
  - $\Delta \sim$ Spare Bandwidth
  - $\Delta \sim$ Queue Size
  - $\Delta = \alpha d_{avg}$ Spare $- \beta$ Queue
- **AIMD**
  - Algorithm:
  - If $\Delta > 0$ => Divide $\Delta$ equally between flows
  - If $\Delta < 0$ => Divide $\Delta$ between flows proportionally to their current rates
Getting the devil out of the details ...

System converges to optimal utilization (i.e., stable) for any link bandwidth, delay, number of sources if:

\[ 0 < \alpha < \frac{\pi}{4N^2} \quad \text{and} \quad \beta = \alpha \sqrt{2} \]

No Parameter Tuning

Fairness Controller

Algorithm:
- If \( \alpha > 0 \) Divide \( \Delta \) equally between flows
- If \( \alpha < 0 \) Divide \( \Delta \) between flows proportionally to their current rates

Need to estimate number of flows \( N \)

\[ N \approx \sum_{i} \frac{T \times (Cwnd_{i} / RTT_{i})}{(N - 1)} \]

No Per-Flow State

XCP Remains Efficient as Bandwidth or Delay Increases

Utilization as a function of Bandwidth

Utilization as a function of Delay

XCP Remains Efficient as Bandwidth or Delay Increases

Utilization as a function of Bandwidth

Utilization as a function of Delay

XCP shows faster response than TCP

XCP Deals Well with Short Web-Like Flows

Average Utilization

Average Queue

Drops

Arrivals of Short Flows/sec
XCP is Fairer than TCP

Same RTT

Different RTT

Avg. Throughput

Flow ID

(RTT is 40 ms ——— 330 ms)