Router-Aided Congestion Control

Dina Katabi

nms.csail.mit.edu/~dina
Readings For the Last Two Lectures

From Peterson and Davie

- Reading for last lecture (i.e., TCP):
  - Read section 5 and focus on 5.2.1-5.2.6 (included)

- Reading for this lecture
  - Read 6.2-6.4 (included)
Outline

- No help from routers
  - DropTail
  - Problems with DropTail

- What can routers do?

- Examples
  - RED
  - XCP
  - Fair Queuing
  - Hop-by-hop congestion control
TCP + DropTail

- Most routers use DropTail

- Problems:
  - Large average queues
  - Bursty losses
  - Synchronization
  - No protection from misbehaving flows
Large Average Queues

- Queues are needed to absorb transient bursts
  - Yet avg. queue should stay low
- TCP fills the queue until a drop
  - Keeps avg. queue length high 😞
Traffic from a single source comes in a burst

- Packets are dropped in bursts
- Too many packets dropped from same cwnd
- Source can't use 3 dupacks to recover and has to wait for timeout ⇒ less efficient
Synchronization

- All losses happen together when the queue overflows
- Sources decrease together ⇒ potential underutilization
- Sources increase together until overflowing queue
- Cycle repeats...

Mainly caused by drops concentration at time of overflow
No Flow Protection

- No protection against misbehaving flows
- What happens if 1 UDP sending at 10 Mb/s shares a 10 Mb/s link with 30 TCPs?
How can routers help?

- Send an early and more explicit congestion feedback
  - Drops occur when queue overflows, but routers can send feedback earlier when queue starts forming
  - Spread the drops and desynchronize the senders

- Tell senders how much to increase or decrease
  - Routers can measure congestion by comparing input traffic to capacity and looking at the queue size

- Can provide protection against misbehaving senders
  - Routers can drop the packets of sources which send more traffic

- Scheduling: decide which packets in the queue go first
  - Give flows different quality of service (QoS)
Pors & Cons of Routers’ Participation in Cong. Cont.

**Pros**
- Who owns the resource should control it (why trust the senders)
- Can do many useful things that we couldn’t do otherwise:
  - better efficiency and fairness, QoS, protection, …

**Cons**
- Increase the complexity of the routers
- Deployment is harder
  - It is hard to change the routers (convince ISPs)
  - Many schemes need to change a large number of routers together for the scheme to be effective
1. Random Early Detection (RED)
RED’s objectives

- Operates at congestion avoidance
  - Keeps avg. queue low (low delay)
  - Safer
- Attempts to solve the synchronization and bursty drops problems
RED Algorithm

- Define 2 vars: min_thresh and max_thresh
  - Try to keep the queue between these two vars

- How? when a packet arrives:
  - Compute $Q_{avg} = w \times Q + (1-w) \times Q_{avg}; \ 0 < w < 1$
  - Drop packet with probability $p = f(Q_{avg})$
What Does RED Achieve?

- Less bias against bursty flows
  - Why?
- Smaller average queue size
  - Why?
- Reduces likelihood of bursty drops
  - Why?
- Reduces likelihood of synchronization
  - Why?
What Doesn’t RED Achieve?

- No flow protection
  - Remember the UDP-TCP problem
- RED introduces its own problems
  - Hard to set parameters

8 Sources

32 Sources
ECN: Explicit Congestion Notification

- Mark packet instead of dropping it. Receiver has to relay this mark to the sender
  - Explicit feedback

- Advantages
  - In wireless environments drops are mainly caused by errors rather than congestion (ECN may help if widely deployed)
  - No wasting of resources on packets that will be dropped

- Cons
  - Deployment
  - No significant improvement in wired environments
2. Fair Queuing
Fair Queuing

- **Objective**
  - Focuses on fairness
  - Protect a source from other sources that send a huge amount of traffic (fairness)
How does it work?

- Each Source has a separate queue

- To be really fair, you would like to do round robin between the queues and at each time send one bit from each queue
  - However we can’t divide traffic to bits, we have to work with packets, and packets are not the same size

- For each packet compute the finishing time if the system sent bit by bit round robin
  - Imagine a clock that ticks each time a bit from all active flows is sent (i.e., each round);
  - \( F_i = \text{Max} (F_{i-1}, A_i) + P_i \)
    - \( F_i \) is finishing time; \( P_i \) is length; \( A_i \) is the arriving time

- Send packets according to their finishing time
How does it work?

- Shorter packets are sent first
- Longer packets are not pre-empted
  - E.g., if the grey packet has started transmission before the two white packets arrive, then it finishes before them
- FQ is fair but not very scalable
  - Per flow state, and per-flow queuing
3. **XCP: eXplicit Control Protocol**
Problem

- Link bandwidth is increasing exponentially
- But, TCP can’t maintain a very large throughput and can’t benefit from the large increase in the per-flow bandwidth
For example,

- A TCP connection with a packet size 1500 byts and a steady-state throughput of 10Gb/s and RTT=100ms requires:
  - An average cwnd of 80,000 packets
  - 1 drop every 5,000,000,000 packets or more; I.e., one drop every 1 and 2/3 hours.

This is unrealistic!
Problem is caused by TCP’s Lack of Fast Response

- TCP increases by 1 packet/RTT; takes forever in high-bandwidth environments
- Slow-start doesn’t help much:
  - When a TCP starts, it increases exponentially
    ⇒ Too many drops
    ⇒ Flows halve cwnd
    ⇒ Flows ramp up by 1 packet/RTT, taking forever to grab the large spare bandwidth
What prevents TCP from increasing quickly?

- TCP Congestion feedback is binary (i.e., drop or no-drop) and **indifferent to the degree of congestion**
  - Source doesn’t know whether spare bandwidth is 1 packet or 1000,000 packets. Increasing too quickly might cause congestion collapse.
  - Also, binary feedback causes oscillations

**Solution**

- Make increase proportional to spare bandwidth and decrease proportional to congestion
  - ⇒ Need more explicit congestion feedback from routers
XCP:

Decouple Congestion Control from Fairness

Coupled in TCP because a *single* mechanism, 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
XCP: An explicit Control Protocol

1. Congestion Controller
2. Fairness Controller
How does XCP Work?

- Round Trip Time
- Congestion Window
  - Feedback = + 0.1 packet
How does XCP Work?

- **Round Trip Time**
- **Congestion Window**
- **Feedback = - 0.3 packet**
How does XCP Work?

Congestion Window = Congestion Window + Feedback

Note, every router can overwrite the feedback to decrease it but can never increase it

XCP extends ECN
How Does an XCP Router Compute the Feedback?

**Congestion Controller**

**Goal:** Matches input traffic to link capacity & drains the queue

Looks at aggregate traffic & queue

**Algorithm:**
Aggregate traffic changes by $\Delta$
$\Delta \sim \text{Spare Bandwidth}$
$\Delta \sim \text{- Queue Size}$

So, $\Delta = \alpha d_{avg} \text{Spare} - \beta \text{Queue}$

---

**Fairness Controller**

**Goal:** Divides $\Delta$ between flows to converge to fairness

Looks at a flow’s state in Congestion Header

**Algorithm:**
If $\Delta > 0 \Rightarrow$ Divide $\Delta$ so that flows increase their throughput equally
If $\Delta < 0 \Rightarrow$ Divide $\Delta$ between flows proportionally to their current rates

---

**MIMD**

**AIMD**
Reflections on XCP

- **Pros**
  - More efficient than TCP
    - Why?
  - Fairer than TCP
    - Why?
  - Can deal with large per-flow bandwidth well
    - Why?

- **Cons**
  - More work at routers
  - Deployment requires changing senders and routers
  - No flow protection (also true about TCP)
4. Hop-by-Hop Congestion Control
End2End vs. Hop-by-Hop

DropTail, RED and XCP use end2end cong. cont.
To make this a congestion control protocol, we need to specify how R1 and R2 slow down so that:
1) Bottleneck is not congested
2) fair to flows sharing R1 and R2
Evaluation of Hop-by-hop

- **Pros**
  - Does not involve links downstream a bottleneck
  - Works with short flows (i.e., messages)
  - Faster
  - Does it protect flows from misbehavior?

- **Cons**
  - Queue buildup at each hop → large delay
  - Deployment