Internet Congestion Control

II. Router Participation in Congestion Control

Dina Katabi

Office hours by appointments

dk@mit.edu
http://www.ana.lcs.mit.edu/dina

Outline

- No help from routers
  - DropTail
  - Problems with DropTail
- What can routers do?
- Examples
  - RED
  - XCP
  - Hop-by-hop congestion control

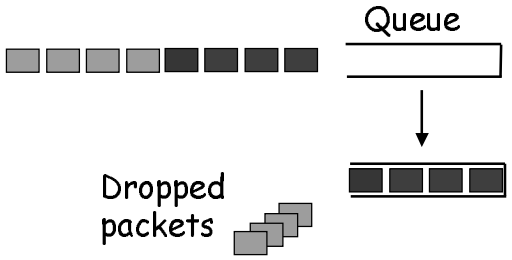
TCP + DropTail

- 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 ☹

Bursty Losses



- 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

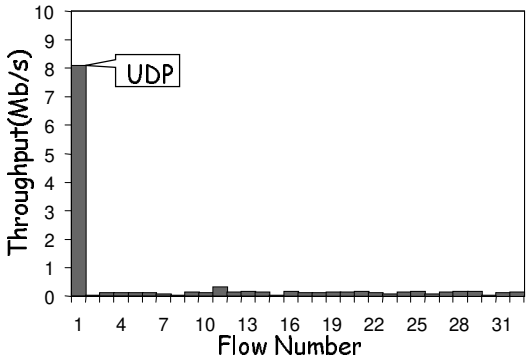
Synchronization

- 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
  - Spreads 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 and the queue size
- Can provide protection
  - Routers can drop the packets of misbehaving sources
- Scheduling: decide which packets in the queue go first
  - Give flows different quality of service (QoS)

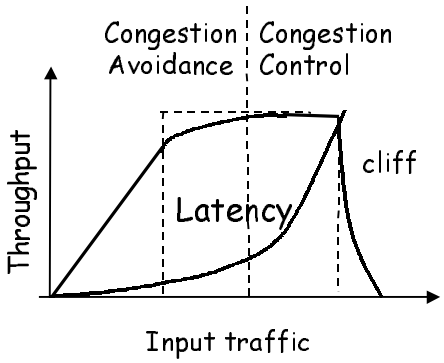
Pors & Cons of Routers' Participation in Cong. Cont.

- 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
  - Increases 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)

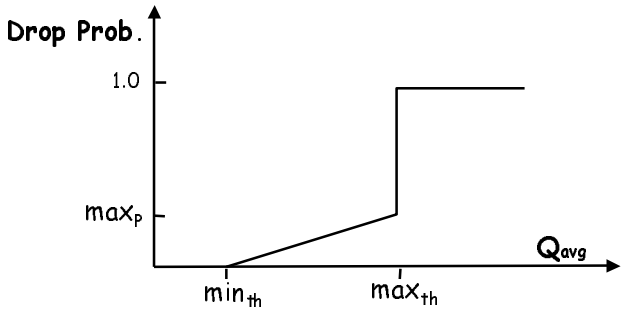
RED's objectives

- Operate at congestion avoidance
  - Keeps avg. queue low (low delay)
  - Safer
- Attempt 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 $p=f(Q_{avg})$

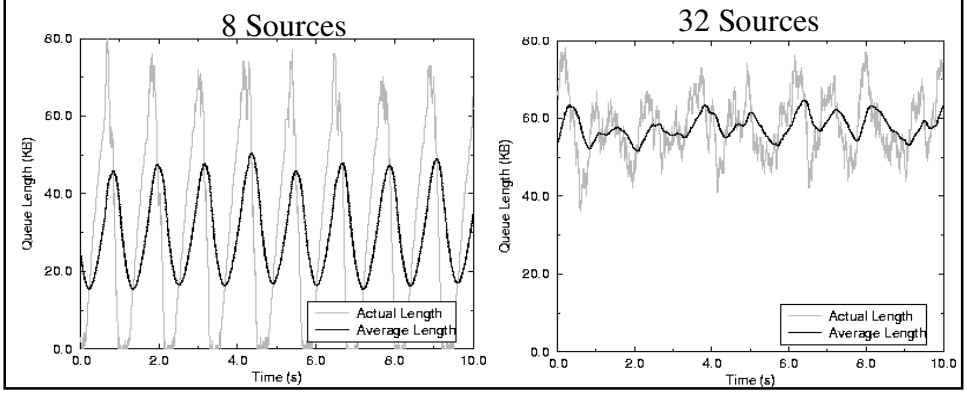


What Does RED Achieve?

- Less bias against bursty flows
  - Why?
- Smaller average queue size
  - Why?
- Reduces likelihood of synchronization
  - Why?
- Reduces likelyhood of bursty drops
  - Why?

What Doesn't RED Achieve?

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



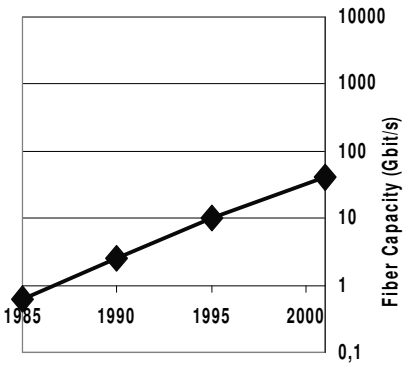
ECN: Explicit Congestion Notification

- 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. 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 bandiwdth



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 feedback from routers
- Still have a problem because multiplicative-increase and multiplicative-decrease (MIMD) doesn't converge to fairness

Proposed Solution:

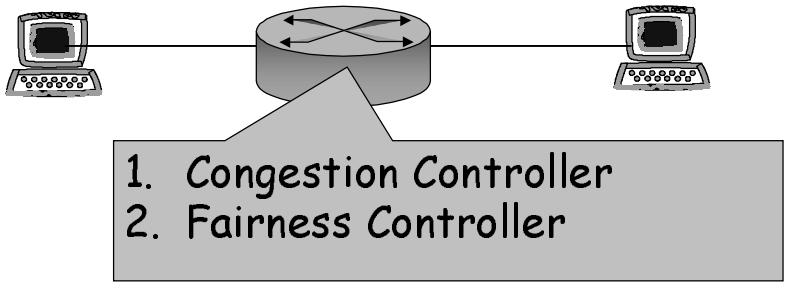
Decouple Congestion Control from Fairness

Coupled in TCP because a *single* mechanism, AIMD, controls both

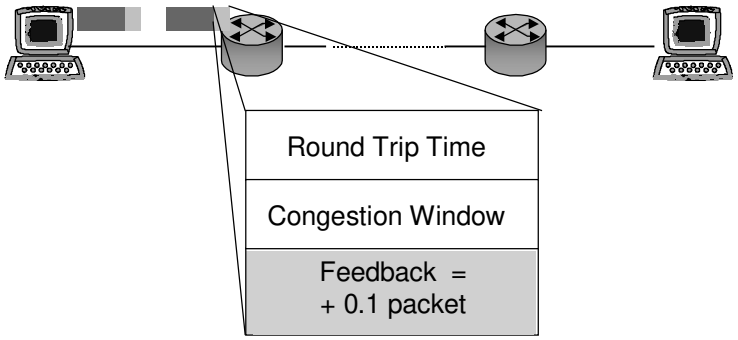
How does decoupling solve the problem?

1. To control congestion: use MIMD which shows fast response
2. To control fairness: use AIMD which converges to fairness

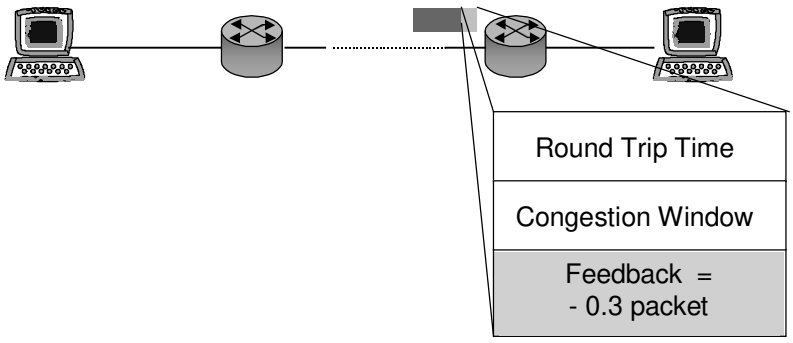
XCP: An eXplicit Control Protocol



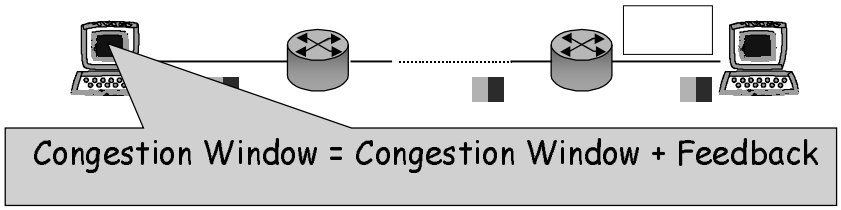
How does XCP Work?



How does XCP Work?



How does XCP Work?



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

MIMD

Algorithm:

Aggregate traffic changes by Δ

$\Delta \sim$ Spare Bandwidth

$\Delta \sim -$ Queue Size

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

Fairness Controller

Goal: Divides Δ between flows to converge to fairness

Looks at a flow's state in Congestion Header

AIMD

Algorithm:

If $\Delta > 0 \Rightarrow$ Divide Δ so that flows increase their throughput equally

If $\Delta < 0 \Rightarrow$ Divide Δ between flows proportionally to their current rates

A few cool things

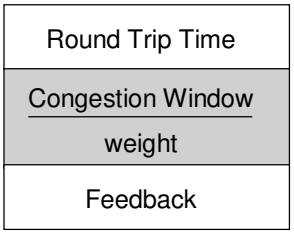
- Can choose constant values for α and β and prove convergence to optimal utilization and small queues for any link bandwidth, delay, number of sources
  - Why one can't do that with TCP?
- By putting the flows' state in the congestion header routers are maintained simple
  - For example, the fairness controller make all flows increase equally without per-flow state at routers
    - Naïve way needs to keep per-flow state
    - How can XCP do it without per-flow state?

Also Provides Differential BW

Differential Bandwidth Allocation

- Each user has a weight
- Users congested at the same link obtain throughputs proportional to their respective weights

- Implementation:
  - Substitute the congestion window field by congestion window/weight
  - Works because router would see the flow as multiple flows (e.g., if weight is 2, flow would look like 2 flows)

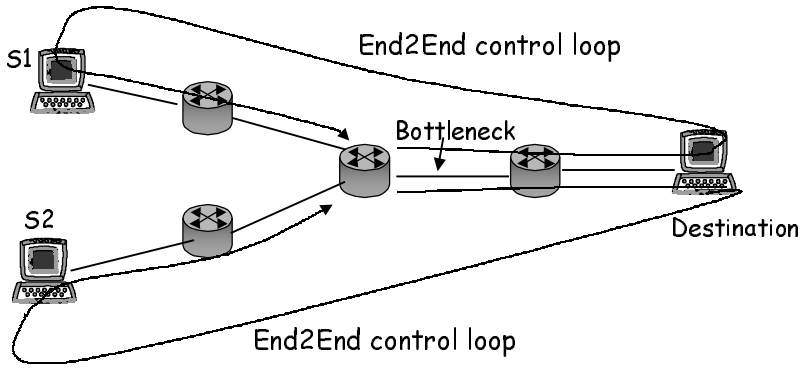


Reflections on XCP

- Pros
  - Better efficiency than TCP
    - Why?
  - Better fairness than TCP
    - Why?
  - Can deal with high per-flow bandwidth well
    - Why?
- Cons
  - More work at routers (a few instructions and no memory access)
  - Deployment requires changing both senders and routers
  - No flow protection

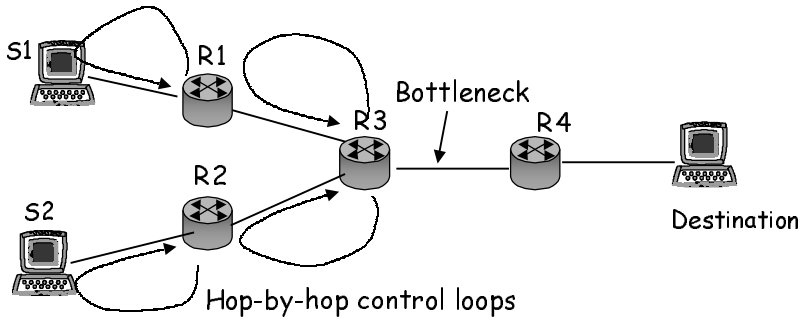
3. Hop-by-Hop Congestion Control

End2End vs. Hop-by-Hop



DropTail, RED and XCP use end2end cong. cont.

End2End vs. Hop-by-Hop



To make this a cong. cont. 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
  - Router complexity?
  - Deployment