

Congestion Control

Outline

- Queuing Discipline
- Reacting to Congestion
- Avoiding Congestion

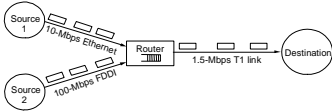
Spring 2000

CS 461

1

Issues

- Two sides of the same coin
 - pre-allocate resources so as to avoid congestion
 - control congestion if (and when) is occurs



- Two points of implementation
 - hosts at the edges of the network (transport protocol)
 - routers inside the network (queuing discipline)
- Underlying service model
 - best-effort (assume for now)
 - multiple *qualities of service* (later)

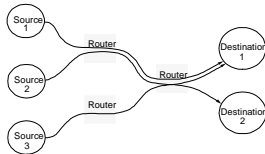
Spring 2000

CS 461

2

Framework

- Connectionless flows
 - sequence of packets sent between source/destination pair
 - maintain *soft state* at the routers



- Taxonomy
 - router-centric versus host-centric
 - reservation-based versus feedback-based
 - window-based versus rate-based

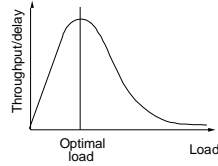
Spring 2000

CS 461

3

Evaluation

- Fairness
- Power (ratio of throughput to delay)



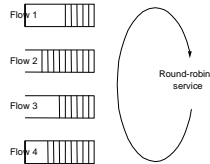
Spring 2000

CS 461

4

Queuing Discipline

- First-In-First-Out (FIFO)
 - does not discriminate between traffic sources
- Fair Queuing (FQ)
 - explicitly segregates traffic based on flows
 - ensures no flow captures more than its share of capacity
 - variation: weighted fair queuing (WFQ)
- Problem?



Spring 2000

CS 461

5

FQ Algorithm

- Suppose clock ticks each time a bit is transmitted
- Let P_i denote the length of packet i
- Let S_i denote the time when start to transmit packet i
- Let F_i denote the time when finish transmitting packet i
- $F_i = S_i + P_i$
- When does router start transmitting packet i ?
 - if before router finished packet $i - 1$ from this flow, then immediately after last bit of $i - 1$ (F_{i-1})
 - if no current packets for this flow, then start transmitting when arrives (call this A_i)
- Thus: $F_i = \text{MAX}(F_{i-1}, A_i) + P_i$

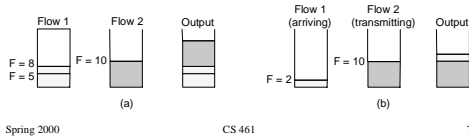
Spring 2000

CS 461

6

FQ Algorithm (cont)

- For multiple flows
 - calculate F_i for each packet that arrives on each flow
 - treat all F_i 's as timestamps
 - next packet to transmit is one with lowest timestamp
- Not perfect: can't preempt current packet
- Example



TCP Congestion Control

- Idea
 - assumes best-effort network (FIFO or FQ routers) each source determines network capacity for itself
 - uses implicit feedback
 - ACKs pace transmission (*self-clocking*)
- Challenge
 - determining the available capacity in the first place
 - adjusting to changes in the available capacity

Spring 2000 CS 461 8

Additive Increase/Multiplicative Decrease

- Objective: adjust to changes in the available capacity
- New state variable per connection: **CongestionWindow**
 - limits how much data source has in transit

$$\text{MaxWin} = \text{MIN}(\text{CongestionWindow}, \text{AdvertisedWindow})$$

$$\text{EffWin} = \text{MaxWin} - (\text{LastByteSent} - \text{LastByteAcked})$$

- Idea:
 - increase **CongestionWindow** when congestion goes down
 - decrease **CongestionWindow** when congestion goes up

Spring 2000 CS 461 9

AIMD (cont)

- Question: how does the source determine whether or not the network is congested?
- Answer: a timeout occurs
 - timeout signals that a packet was lost
 - packets are seldom lost due to transmission error
 - lost packet implies congestion

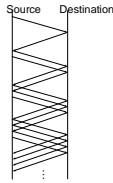
Spring 2000

CS 461

10

AIMD (cont)

- Algorithm
 - increment **CongestionWindow** by one packet per RTT (*linear increase*)
 - divide **CongestionWindow** by two whenever a timeout occurs (*multiplicative decrease*)
- In practice: increment a little for each ACK
 - Increment** = $(MSS * MSS) / \text{CongestionWindow}$
 - CongestionWindow** += **Increment**



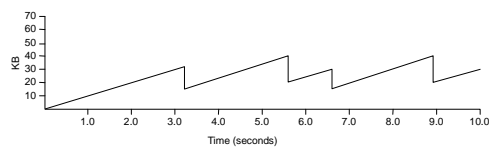
Spring 2000

CS 461

11

AIMD (cont)

- Trace: sawtooth behavior



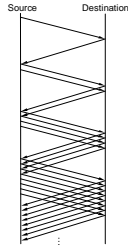
Spring 2000

CS 461

12

Slow Start

- Objective: determine the available capacity in the first
- Idea:
 - begin with `CongestionWindow = 1` packet
 - double `CongestionWindow` each RTT (increment by 1 packet for each ACK)



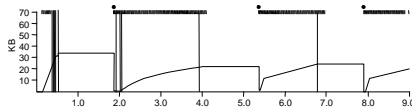
Spring 2000

CS 461

13

Slow Start (cont)

- Exponential growth, but slower than all at once
- Used...
 - when first starting connection
 - when connection goes dead waiting for timeout
- Trace



- Problem: lose up to half a `CongestionWindow`'s worth of data

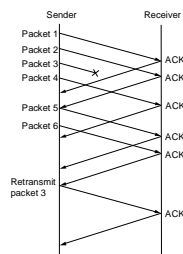
Spring 2000

CS 461

14

Fast Retransmit and Fast Recovery

- Problem: coarse-grain TCP timeouts lead to idle periods
- Fast retransmit: use duplicate ACKs to trigger retransmission



Spring 2000

CS 461

15

Results

- Fast recovery
 - skip the slow start phase
 - go directly to half the last successful **CongestionWindow (ssthresh)**

Spring 2000 CS 461 16

Congestion Avoidance

- TCP's strategy
 - control congestion once it happens
 - repeatedly increase load in an effort to find the point at which congestion occurs, and then back off
- Alternative strategy
 - predict when congestion is about to happen
 - reduce rate before packets start being discarded
 - call this congestion *avoidance*, instead of congestion *control*
- Two possibilities
 - router-centric: DECbit and RED Gateways
 - host-centric: TCP Vegas

Spring 2000 CS 461 17

DECbit

- Add binary congestion bit to each packet header
- Router
 - monitors average queue length over last busy+idle cycle

- set congestion bit if average queue length > 1
- attempts to balance throughput against delay

Spring 2000 CS 461 18

End Hosts

- Destination echoes bit back to source
- Source records how many packets resulted in set bit
- If less than 50% of last window's worth had bit set
 - increase `CongestionWindow` by 1 packet
- If 50% or more of last window's worth had bit set
 - decrease `CongestionWindow` by 0.875 times

Spring 2000

CS 461

19

Random Early Detection (RED)

- Notification is implicit
 - just drop the packet (TCP will timeout)
 - could make explicit by marking the packet
- Early random drop
 - rather than wait for queue to become full, drop each arriving packet with some *drop probability* whenever the queue length exceeds some *drop level*

Spring 2000

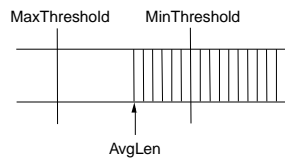
CS 461

20

RED Details

- Compute average queue length
$$\text{AvgLen} = (1 - \text{Weight}) * \text{AvgLen} + \text{Weight} * \text{SampleLen}$$
$$0 < \text{Weight} < 1 \text{ (usually } 0.002)$$

`SampleLen` is queue length each time a packet arrives



Spring 2000

CS 461

21

RED Details (cont)

- Two queue length thresholds

```
if AvgLen <= MinThreshold then
  enqueue the packet
if MinThreshold < AvgLen < MaxThreshold then
  calculate probability P
  drop arriving packet with probability P
if MaxThreshold <= AvgLen then
  drop arriving packet
```

Spring 2000

CS 461

22

RED Details (cont)

- Computing probability P

```
TempP = MaxP * (AvgLen - MinThreshold) /
        (MaxThreshold - MinThreshold)
P = TempP / (1 - count * TempP)
```

- Drop Probability Curve



Spring 2000

CS 461

23

Tuning RED

- Probability of dropping a particular flow's packet(s) is roughly proportional to the share of the bandwidth that flow is currently getting
- MaxP** is typically set to 0.02, meaning that when the average queue size is halfway between the two thresholds, the gateway drops roughly one out of 50 packets.
- If traffic is bursty, then **MinThreshold** should be sufficiently large to allow link utilization to be maintained at an acceptably high level
- Difference between two thresholds should be larger than the typical increase in the calculated average queue length in one RTT; setting **MaxThreshold** to twice **MinThreshold** is reasonable for traffic on today's Internet
- Penalty Box for Offenders

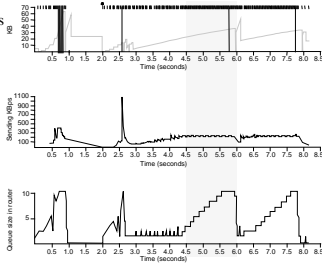
Spring 2000

CS 461

24

TCP Vegas

- Idea: source watches for some sign that router's queue is building up and congestion will happen too; e.g.,
 - RTT grows
 - sending rate flattens



Spring 2000

CS 461

25

Algorithm

- Let **BaseRTT** be the minimum of all measured RTTs (commonly the RTT of the first packet)
- If not overflowing the connection, then
 - $\text{ExpectedRate} = \text{CongestionWindow} / \text{BaseRTT}$
- Source calculates sending rate (**ActualRate**) once per RTT
- Source compares **ActualRate** with **ExpectedRate**

```

Diff = ExpectedRate - ActualRate
if Diff <  $\alpha$ 
    increase CongestionWindow linearly
else if Diff >  $\beta$ 
    decrease CongestionWindow linearly
else
    leave CongestionWindow unchanged
    
```

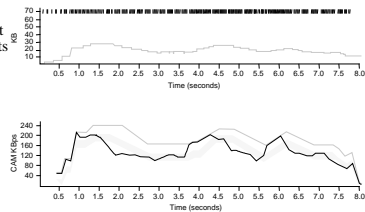
Spring 2000

CS 461

26

Algorithm (cont)

- Parameters
 - $\alpha = 1$ packet
 - $\beta = 3$ packets



- Even faster retransmit
 - keep fine-grained timestamps for each packet
 - check for timeout on first duplicate ACK

Spring 2000

CS 461

27
