MIT 6.829: Computer Networks Fall 2017

Lecture 14: Congestion Avoidance and Control

Scribe: Jake Read, Dougie Kogut

1 Overview

This lecture is on the key problems in wired networks: routing and congestion control[1].

2 Network Congestion

Say we have a network with a bottleneck - see Fig. 2 and we assume all endpoints have 'infinite' demand. At $Router_1$, the buffer fills quickly to its maximum, and if the sources don't back off, eventually the buffer is overrun and we have dropped packets, as in Fig. 2.

As well, when the queue becomes large, packets see increased delay - as they have to 'wait' for messages ahead of them to be passed along.

Large, persistent queue's cause large delays and cause many drops. For the first time in 1988, this became a problem in TCP/IP - we had what were called 'congestion collapses'. The internet came to a complete stop.

TCP/IP uses timeouts, as delays increase, those timeouts fire too early - so messages that are received are *still* re-transmitted. This leads to exponential increase of network traffic. This is a common behaviour of networks - i.e. in traffic, in highly congested situations the capacity of the network drops below it's theoretical maximum capacity. As such, long delays and a large number of drops are indicators of a congested network.

3 Congestion Control

3.1 Intro

Congestion Control is any protocol or mechanism that is used to control the source's rate, in the interest of maintaining network efficiency and fairness.

In the aforementioned example, the network is considered efficient if the bottleneck utilization is at 100%. I.E. the network is operating at it's maximum capacity. In addition, we desire a small queue at each of the routers (to minimize packet delay) and low, or no, dropped packets.

Fairness is evaluated as how traffic, or 'bottleneck time' is distributed to different senders. I.E. in a 2MBps bottleneck, that two sources should get 1MBps each.

In the particular concept of MaxMin fairness, you look at the maximum capacity of the network, and you give sources the minimum of their fair share (equal divisions), or their demands. You then

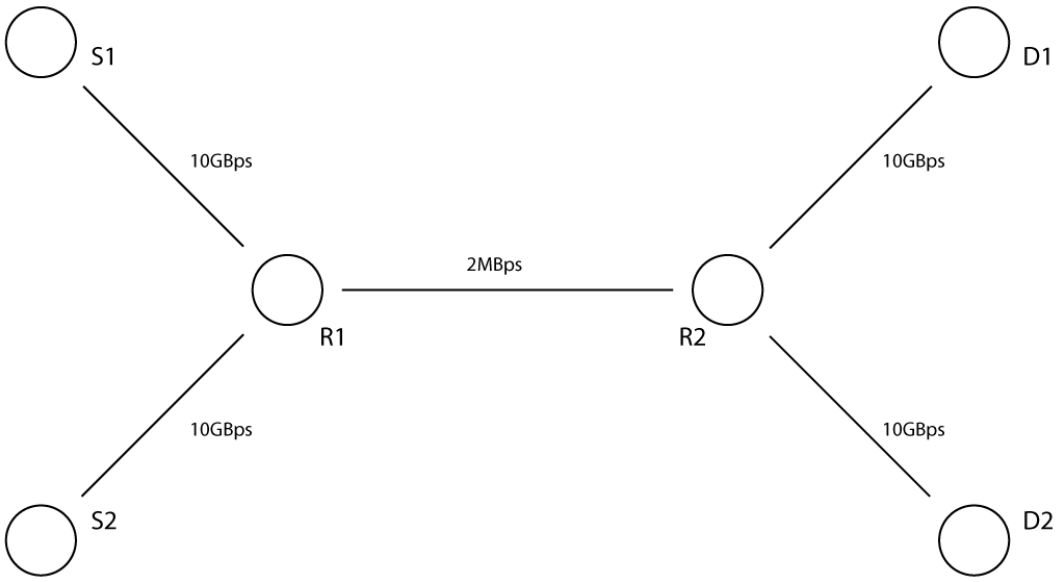

Fig. 1 Example Bottlenecked Network

distribute the rest of the 'leftover' speed (the difference between the equal division and the demand if the demand was less than the equal distribution) equally, and iterate.

3.2 Congestion Control Strategies

While congestion is only detected at a router - it can only be acted upon by a source. This means that congestion control must be executed in a distributed manner. The Router needs somehow to communicate to the sources that are sending traffic to them. However, routers want to be relatively dumb devices - we want to minimize computation time at the Router so as to allow fast packet

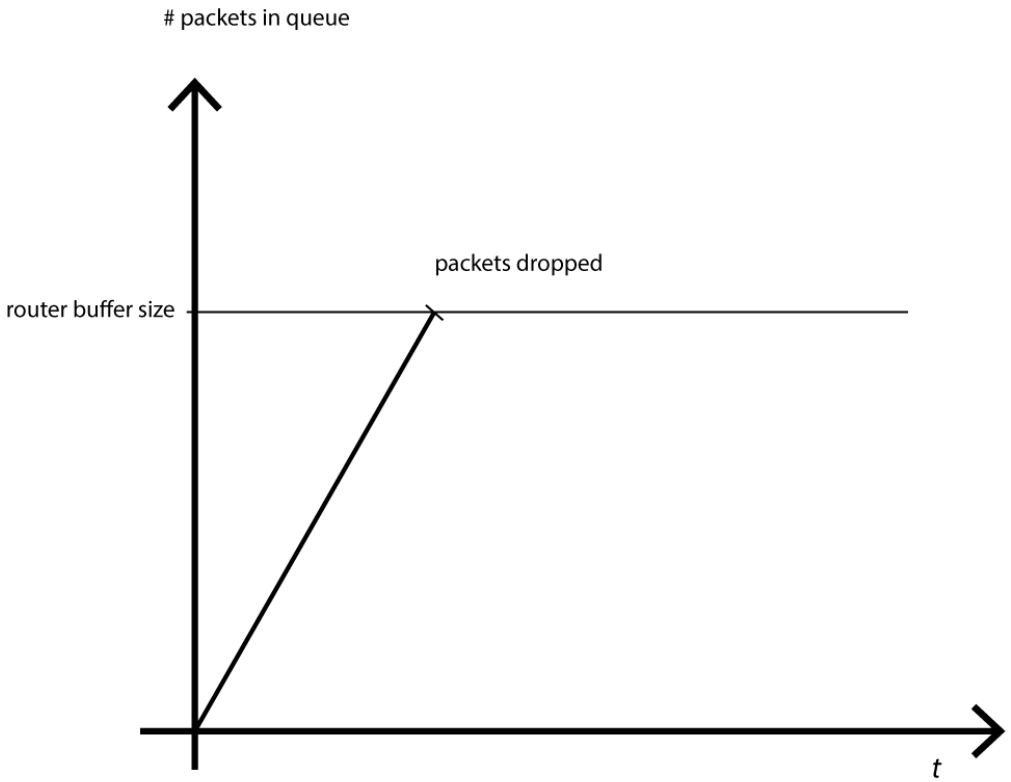

Fig. 2 Buffer / Queue Fillup

forwarding and avoid this computation time becoming the bottleneck instead of the link.

So, information about the problem is very distributed. In addition, routers only see packets - not units of flow. Simplicity 'is king' in these problems - elegant solutions.

3.3 End to End

E2E(End-To-end) congestion control is a critical insight on the internet's infrastructure - complexity is, as much as possible, pushed to 'the edges' of the network, rather than embeeded in routers, switches etc. In E2E congestion control, sources control rates.

TCP operates here - routers simply drop packets when their buffers are full, and TCP Source uses no-acks as a sign of congestion, and backs off.

3.4 Router Assisted Congestion Control

On the other end, (E2E / RACC is a 'spectrum') router helps by providing info or computing the desired rates.

This is only possible, really, where network size is limited and 'under the same administration'. Data centers, for instance, can afford to go further towards router assisted congestion control whereas the Internet at large cannot.

4 TCP Congestion Control

Basically, the source transmits, and slowly increases its transmission rate and when it sees a drop, it backs off. Simple!

TCP is a *window based* congestion control protocol. A *window* is a # of outstanding packets. I.E. TCP puts this # of packets 'on the line' and will not put any more on the line until it receives an ack. Fig. 3 is a fixed-size window. If the congestion window is fixed, what is the rate - in packets per second. $Throughput = cwnd/RTT$ where RTT is round trip time. Rount Trip Time is $RTT = t_{transmit} + t_{ackreceived}$ and critically includes propogation time for the message going forwards, and as well propogation for the message to return - AND the time packets and acks spend in queues.

A window protocol 'is clocked by the ack' - i.e. it's event based, you only transmit another packet when you receive an ack. This is kind of brilliant. See Fig. 4

However, there is no mechanism yet described to set the size of this congestion window. If we simply open up and rip 30 packets onto the network, we can quickly add congestion.

4.1 Additive Increase, Multiplicative Decrease (AIMD)

This is TCP's algorithm for setting the congestion window:

- Every RTT if no drop : cwnd = cwnd + 1 if drop : cwnd = cwnd / 2

In order to detect drops, timeouts around the size of RTT are used.

To improve the network's initial utilization, TCP uses a mechanism called Slow Start to jump start the initial bandwidth usage. Instead of incrementing by 1 packet at each RTT, TCP doubles the window size until a packet is lost. This is only used once at the beginning of a connection.

4.2 TCP - AIMD's Fairness and Efficiency

Let's discuss x_1 and x_2 as the windows of S_1 and S_2

Assume infinite demands.

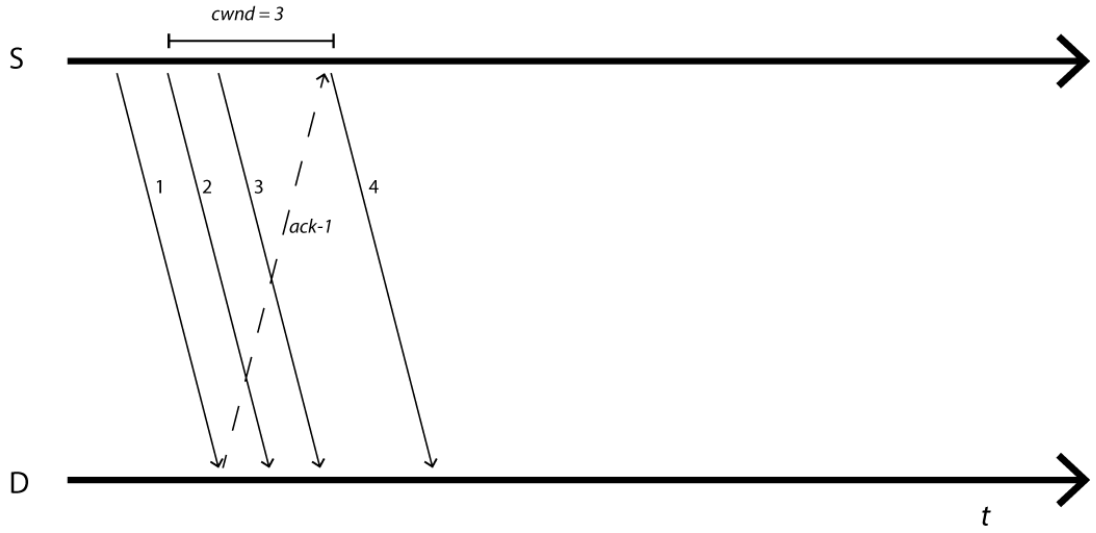

Fig. 3 Window-Based Rate Limiting

Anything on the fairness line is fair, anything on the efficiency line is efficient. We want to operate around the intersection of these lines.

Say we have P_1 - both packets are lost, and so both senders do $cwnd/2$. Now we have P_2. We go to P_3, P_4 where we are past capacity again, and again do $cwnd/2$, etc. Essentially we set up a system which oscillates around the fair / efficient point. Multiplicative decrease brings the point closer to the line of fairness, additive increase brings the rates up towards capacity, parallel along the line of fairness.

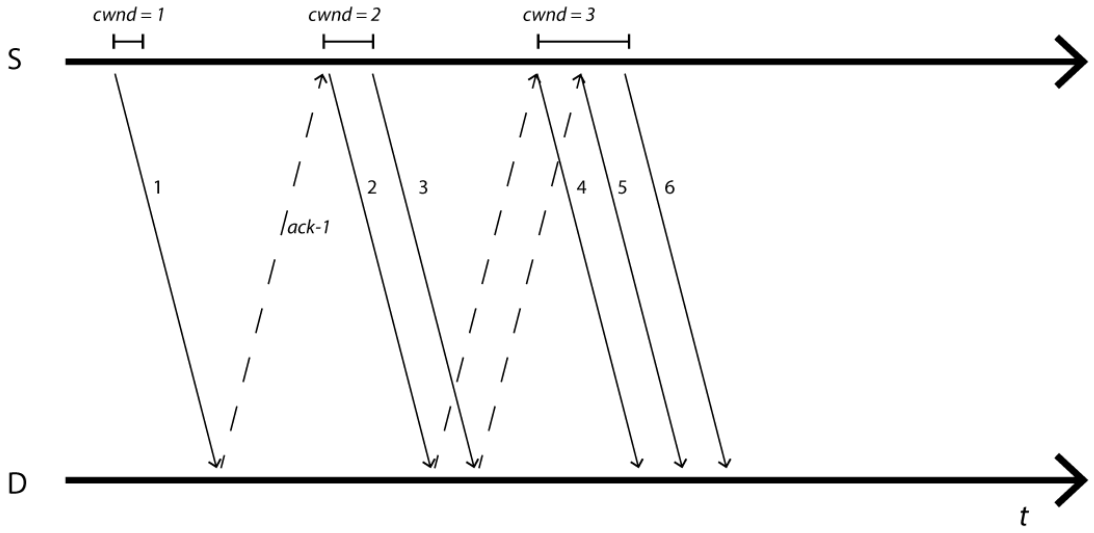

Fig. 4 Increasing Window on Acks

4.3 Remarks on TCP

- TCP Acks the 'next missing packet' - when receives packet 1, sends 'i am waiting for packet 2'. say packet 1 is lost. send packet 3, ack says 'waiting for packet 2'. sends packet 4, receives 'waiting for packet 2' - so retransmit packet 2, receives ack 'waiting for packet 5' - i.e. it 'catches up' . SO a duplicate ack indicates a loss (or at least an ACK that is taking an unusually long time to arrive). if two acks say 'waiting for packet 2' it is likely that packet 2 was lost.
- How to detect a loss:

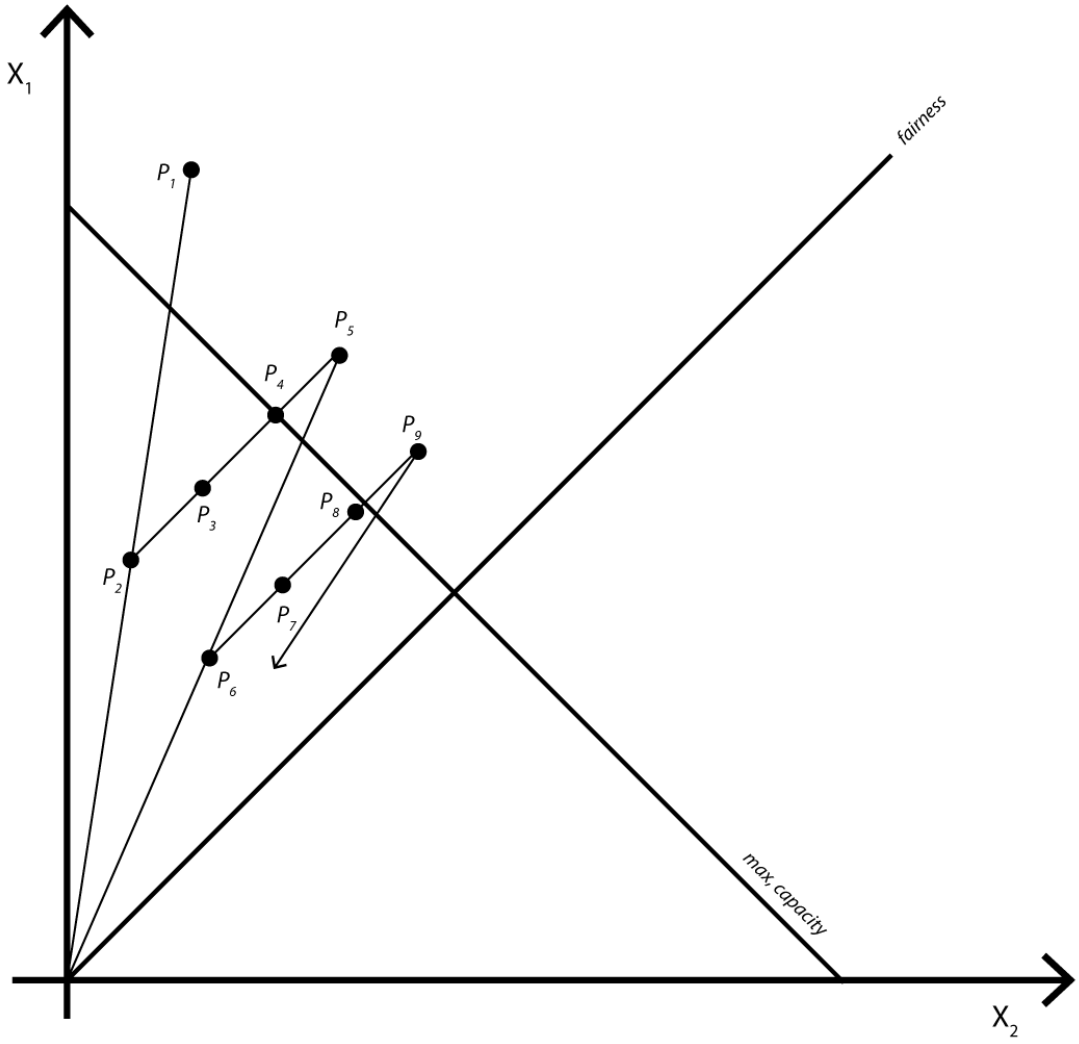

Fig. 5 Convergence on Fairness and Capacity

  - traditionally we use a timeout. the timer is set to some value which is reasonable larger than the RTT - we want to err on the side of conservatively large RTT to avoid retransmits.
  - however, a very large timeout means we may be sitting for a long time, waiting.
  - TCP uses 3 dup acks as a setting for the timer.

- On Startup, TCP uses 'slow start'. This has Exponential Increase - it is not slow at all. At the beginning, on every RTT $cwnd = w * cwnd$ until the first loss, then we go back to AIMD.
- The number of packets in the network is the Sum of the congestion window of the sources. Congestion occurs when the number of packets in the network exceeds the buffer size @ a

bottleneck. $\sum cwnd > bottleneck_{buffer} + bottleneck_{datarate} * bottleneck_{delay}$ this is bandwidth-delay product. The bandwidth-delay product is essentially the 'capacity' of the network to 'hold' packets.

5 Limitations of Droptail

We have a queue, when it is full we drop everything. We have a condition called synchronized drops. All sources will see drops at the same time, all will back off at the same time, all ramp at the same rate, see drops at the same time, back off at the same time. The system oscillates between being underutilized and completely congested and increases the average aggreagte efficiency of the network. If we spread the drops, we have a kind of RC filter on the drops - this is what RED does. RED is a paper we were not assigned. *welp*

5.1 RED-Q

$Q_{avg} = w * Q_{instant} + (1 - w) * Q_{avg}$

Drop incoming packets with probability P where $P = f(Q_{avg}$

Essentially, drop early - once we have a minimum average queue size. Turns out it's very hard to tune the values used in RED-Q.

In addition, the router has to do this calculation - as it has the queue.

References

[1] V. Jacobson. 1988. Congestion avoidance and control. In Symposium proceedings on Communications architectures and protocols (SIGCOMM '88), Vinton Cerf (Ed.). ACM, New York, NY, USA, 314-329. DOI=http://dx.doi.org/10.1145/52324.52356

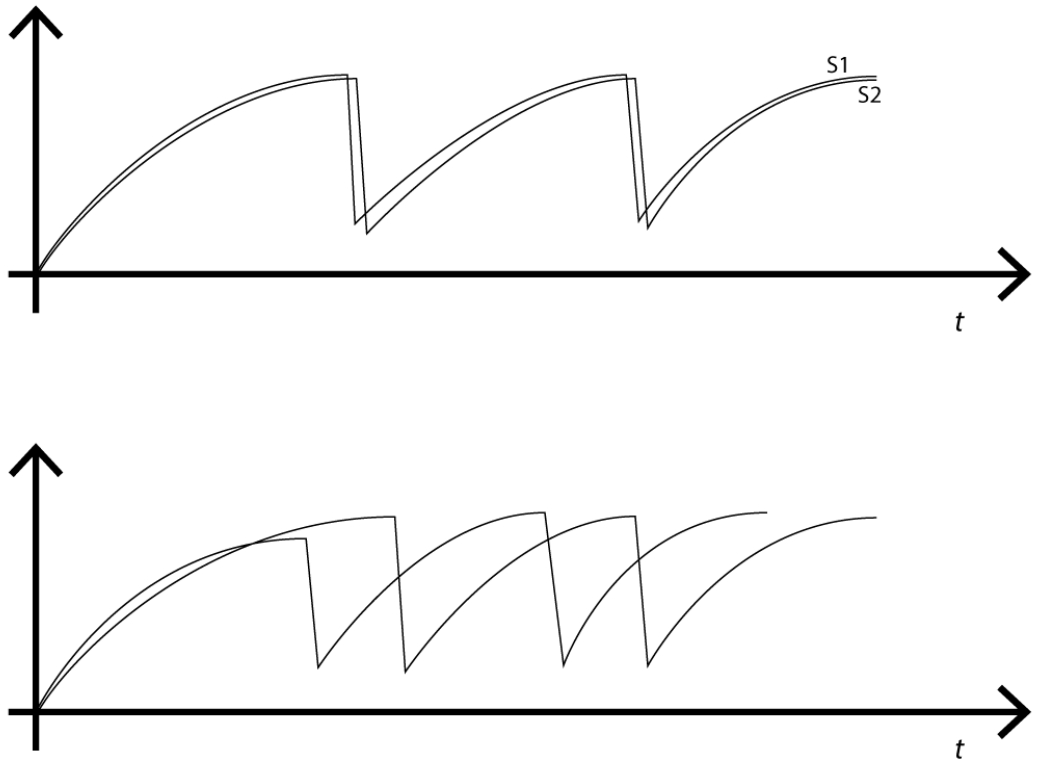

Fig. 6 Synced Drops vs Distributed Drops