CS 336 Lecture 32– 5/6/98

Announcements

- Simulator project REALLY due Monday.

Congestion Control

- Last time, I briefly reviewed the need for congestion control.
  - When routers/switches get overloaded, they drop packets.
  - If the hosts whose packets get dropped simply react by sending duplicates, the congestion is likely to increase (in a vicious cycle).

  and the difference between congestion control and flow control.

- Congestion control is a resource allocation problem. We are allocating the transmission and buffering capacity of network switches to handle the flow of packets from end stations.

- Like many allocation problems, there are two basic approaches:
  - Avoidance/Prevention — come up with allocation policies that make congestion impossible
    * This is a reasonable approach in a network based on virtual circuits. In such a network, when one machine makes a connection to another, all intermediate switches can reserve capacity for the connection (or reject the connection if the resources are not available).
  - Detection and Recovery — include mechanisms that allow the network to detect the onset of congestion and reduce network usage in response.
    * This approach is more appropriate in the TCP/IP world where there are no explicit connections at the lowest level (IP).

- The prevention approach tends to waste resources on average since many connections will not continuously used the capacity allocated to them.
  - To avoid congestion, each router will need to reserve enough capacity for the sum of the maximum bandwidths of all outstanding connections even though most connections will transmit below their maximums most of the time.

- The recovery approach will waste resources during the transient periods where the network does become congested.

- Given our loyalty to TCP/IP, we will consider the detection/recovery mechanisms built into TCP for congestion control.

Congestion control in TCP

1. The first thing to note about his topic is that is might be a bit surprising that congestion control is part of TCP.
   - If congestion does occur it will impact all IP traffic, not just TCP traffic.
   - Congestion can be caused by any source of IP traffic (UDP for example), not just TCP.

2. The logic behind implementing congestion control techniques within TCP seems to have two components.
   - A definitely not "first principles" reason — TCP is the source of the vast majority of IP traffic.
   - A somewhat deeper justification —
     - Congestion control is about allocating the bandwidth available within each of the paths through the network. This requires some notion of end-to-end flow of packets. If each packet sent is viewed as an independent event, there is no notion of "data rate" or bandwidth requirements.
     - While IP and UDP have no explicit notion of a connection, TCP provides one.

3. First, we should observe that TCP already tends to have one good property as far as congestion control is concerned. The limits imposed by end-to-end flow control tend to smooth out the data rate when large quantities of data are transmitted.
   - Recall that for flow control the amount of data a sender can transmit before receiving an ACK is limited by the window size chosen by the receiver.
   - Imagine a machine connected to a high speed network (100MB ethernet) sending packets through the Internet to a remote host. Assume the actual bandwidth the Internet can provide over this path is 50KB.
   - Initially, the sender is likely to fire off packets at something close to 100MB/sec.
   - Once the sender reaches this limit, the rate at which new packets are sent is limited by the arrival rate of acks.

- Since, if there isn't too much congestion, the arrival rate of acks has a lot to do with the arrival rate of the packets being acked, at steady state, the transmission rate will tend to the rate at which packets are being delivered, i.e. 50KB.
- This is known as self-clocking.

4. Recall from our discussion of sliding window protocols at the data link layer that if errors are highly unlikely, the perfect buffer size is the size that allows us just enough leeway to send packets just a bit longer than the time it takes for an Ack to return for the first packet sent.
   - If we have less buffer space than this, we will have to periodically pause and wait for acks.
   - In the standard data link protocol, if we have more buffer space, the sender will accept more packets from the higher level application, but it won't be able to send any faster.
5. Consider what happens at the transport layer if the flow-control window size allows us to send more data than the system can actually deliver in the time required for an ack to return.
   - The key issue here is that the transmission rate on our outgoing port may be higher than that of the underlying network.
   - When the sender first begins transmission, it will be able to send at a rate higher than that at which packets can actually be delivered. When these packets reach the bottleneck in the path (the network/router that is actually limiting the data rate), the will end up buffered and queued at the router.
   - Eventually, the sender will reach the window size limit and be forced to slow down its transmission to match the arrival rate of ACKs. When this occurs, it will add packets to the network at the same rate at which the bottleneck can process packets.
   - As a result, it will add packets to the queues at the bottleneck just fast enough to prevent the bottleneck from reducing its queue lengths.
   - If instead, the sender somehow knew enough to send packets at just the rate the bottleneck could handle, the overall rate of the connection would be the same, but amount of data the bottleneck would need to buffer would be reduced to one packet.
   - Alternately, if the receiver somehow knew enough to set the window size just big enough to handle one round-trip time worth of data, the bottleneck would be able to empty the initial build-up before the first ack allowed the sender to start the steady flow.
6. This illustrates where and how congestion arises:
   - If nodes send faster than some links in the network can handle their data, queues build up.
   - As more connections become active, the throughput for a given connection at a router may decrease causing additional queuing.
   - At some point, routers run out of room for buffers and begin discarding packets.
7. It also suggest that it might be completely avoidable:
   - If all connections limited their data rate so that no router had to buffer more than one packet...

   but this is misleading:
   - When the load on a router lightens up temporarily, we would like to have packets on hand to soak up the available bandwidth. If the packets are all sitting back at the source being send out at what was just the right rate, opportunity may be lost.
8. So, the trick is to overburden the routers somewhat but not too much.
9. The other trick is that end-hosts can infer than "too much" has occurred by monitoring "lost" packets:
   - It turns out that most of the Internet is reliable enough that the main reasons packets get lost is that they are discarded by overloaded routers.
   - Therefore, it is somewhat reasonable to conclude that congestion has occured if a packet's timeout expires.
10. TCP uses this simple feedback to limit congestion by placing an additional "congestion window" on its transmissions and adjusting this window's size based on when timeouts occur.
    - The limit actually placed on outstanding unacknowledged packets will be the smaller of the flow-control window specified by the receiver and the congestion-control window determined by the sender.
11. At the steady state, TCP employs something like Ethernet's exponential backoff algorithm:

- When a packet times out, TCP assumes congestion has set it and halves the size of the congestion control window to reduce the load on the network.
- If this were the only adjustment made to the congestion control window, it would eventually shrink to nothing. so in addition:
  (a) TCP places a lower bound of one packet on the window.
  (b) TCP incrementally adds 1 packet's worth to the congestion window size every time a full window worth of packets is acked.
  (c) These policies are known as multiplicative decrease /additive increase.

12. If you think carefully about what actually happens when a packet gets lost, you will see that we still have a problem. Basically, the whole connection goes dead so we lose our self-clocking property.
    - When one ack is lost, following packets can't be acked because the acks are cummulative.
    - By the time the retransmitted packet arrives, a full window worth of unackable packets may have been received.
    - The retransmitted packet is therefore likely to produce an ACK for a full window's worth of packets which the sender will happily transmit as fast as its outgoing port will allow which is likely to exceed the capacity of some poor router down steam which will throw away more packets.
13. TCP solves this with an approach called "slow start". This approach is used (with slight difference) both when a connection is first established and after a packet is lost.
    - In the case of a timeout, the old window size (divided by two) is saved. In either case the congestion window is set to 1.
    - During the "slow start" period, each time an ack is received, the congestion window is increased by 1. That is, each time a full window worth of packets is correctly acked, the window size is doubled.
    - In the case that slow start was brought on by a timeout, it stops if the window size reaches the saved window size from before the timeout.
    - Otherwise, it continues until a timeout occurs.
14. The mechanisms I have described so far represent a (somewhat imcomplete) description of most current TCP implementations.
15. If you think about them for a bit, you should at least conclude that Berners-Lee wasn't thinking much about how TCP works when the web protocol HTTP was developed.
16. There are some odd tricks routers can perform to help schemes like this work better.
17. One approach is called "Random Early Detection" gateways.
    - The idea is to have routers randomly throw away packets with a probability that increases as the routers available buffer space decreases.
    - The idea is to "trick" the TCP senders whose packets get lost to slow down before the router is overwhelmed and forced to drop many packets.
    - This is an interesting example of "non-critical state". That is, while the IP philosophy would inhibit any scheme in which the routers kept track of information about connections that would destroy the connections if the router failed, RED routers just play a helpful but non-essential role in congestion control.
18. Another issue with which routers can help is fairness.
    - If you think about how TCP handles congestion it might strike you that if someone sends lots of data using UDP, they could convince all the TCP users to back off by just hogging the network and end up benefitting.
    - One can combat this by using a technique called fair queueing in the routers.
      - The basic idea is to allocate each connection passing through a router an equal percentage of the router's capacity.
      - Roughly speaking, this is done by taking turns forwarding packets for connections in a round-robin fashion.
      - At the very least, this is another example of non-critical state.
      - In fact, given the nature of IP, it is not clear how a router would even identify connections. Any reasonable heuristic, however, should be usable since the function is not essential.