Now we address the problem of congestion control. We'll see that much of the (great) complexity in TCP, and the factors limiting the performance of a TCP connection, are due to congestion control.

Transport Layer Protocols & Services

Outline

- Fundamental transport layer services
  - Multiplexing/Demultiplexing
  - Error detection
  - Reliable data delivery
  - Pipelining
  - Flow control
  - Congestion control

- Internet transport protocols
  - UDP
  - TCP
Congestion control is definitely a top-10 networking problem. Unlike in flow control, routers do not tell the sender what limit on window to use — the sender has to figure that out himself, especially when other connections are sharing router buffers and capacity.

Congestion control works only because all senders (more-or-less) participate in the process. Note that here, traffic intensity = 2*λ_in / λ_out.

### The Causes and Effects of Congestion

#### Scenario 1: Two equal-rate senders share a single link

- Two sources send as fast as possible to two receivers across a shared link with capacity $R$
  - Data is delivered to the application at the receiver at rate $\lambda_{out}$
- Packets queue at the router
  - Assume the router has infinite storage capacity
    (Thus no packets are lost and there are no retransmissions)

Congestion control vs. Flow control

- In flow control the sender adjusts its transmission rate so as not to overwhelm the receiver
  - One source is sending data too fast for a receiver to handle
- In congestion control the sender(s) adjust their transmission rate so as not to overwhelm routers in the network
  - Many sources independently work to avoid sending too much data too fast for the network to handle
- Symptoms of congestion:
  - Lost packets (buffer overflow at routers)
  - Long delays (queuing in router buffers)
X-axis on throughput graph can also be interpreted in terms of traffic intensity — traffic intensity = 1, when \( \lambda_{in} = R/2 \).

If both hosts send at the same rate then no matter how fast they send, each host's throughput cannot exceed \( R/2 \).

This assumes fair queuing in the router and fair access to any shared media.

Impact of congestion:

— Is \( \lambda_{in} \geq R/2 \) good? The outbound link is fully utilized.

— In this case a queue is growing without bound and hence the delay is growing exponentially.

— Note that even if the input rates sum to exactly \( R \), the length of the queue is now more slippery.

— Throughput here is now more slippery.

— Throughput of a connection is different on each side of the router.

— Offered load = data generation rate + retransmission rate

— Eg, if 10% loss rate, real useful throughput is only 90%!

— What is \( \lambda_{out} \)?
What can cause premature retransmissions?
- small value of timeout
- a large reordering event

In the case of premature retransmissions each packet is transmitted twice.

These curves are just representative. Think of them as upper and lower bound for performance.

What's the maximum delay now experienced by a packet?
- For a packet that is not lost, it depends on the maximum queue length (and the speed of the outbound link).
- For a packet that is lost (once), it depends on how long it takes to...
Another view of the same scenario.

This makes it easier to see how it is that sources can inject more traffic than the capacity of the router's outbound link.

— The link connecting the host to the router (the LAN) has higher capacity than the links between the routers (the WAN).

This is a plot of one source's input and goodput.

**Congestion collapse:**
- Even though link utilization is 100%, goodput is zero.
- What is the maximum delay now experienced by a packet?
- For a packet that is not lost it depends on the maximum queue length (and the speed of the outbound link).
- For a packet that is lost (once) it depends on how long it takes to signal the loss and the delay experienced by the retransmitted packet.

— All the links are fully utilized but no data is delivered to applications!
The Causes and Effects of Congestion

Summary

- Uncontrolled, congestion can lead to dropped packets
  - This means that bandwidth used for delivering packets to the point of congestion was wasted

- In the limit it can lead to network collapse
  - The network is fully busy but no work gets done

Costs of congestion:
- More work (retransmissions) must be done for a given level of goodput.
- Unneeded retransmissions waste link bandwidth and router buffer space.

How to avoid congestion?

How to deal with congestion?
- The Internet today uses pure end-to-end congestion control but is evolving towards router-assisted methods.

Approaches to Congestion Control
End-to-end v. Hop-by-hop

- End-to-end congestion control
  - End-systems receive no feedback from network
  - Congestion inferred by observing loss and/or delay

- Hop-by-hop congestion control
  - Routers provide feedback to end systems
    - Network determines an explicit rate that a sender should transmit at (e.g., ATM, XCP)
    - Network signals congestion by setting a bit in a packet’s header (e.g., SNA, DECbit, TCP/IP ECN, ATM)
At present, Internet switches and routers play no role in congestion control.

Congestion control is done purely on the end-systems.

Senders adjust transmission rate based on feedback from receivers.

Here, you limit the max send window not just by flow-control, but also congestion-control.

TCP's sliding window is called the congestion window.

The congestion window limits the maximum transmission rate of a connection.

For a window of size $w$ (in segments),

$$\text{if } RTT > w \times \frac{MSS}{R},$$

$$\text{throughput} = \frac{w \times MSS}{RTT} \text{ bytes/sec}.$$

Otherwise,

$$\text{throughput} = R \text{ bytes/sec}.$$

Note that the maximum rate does not depend on $R$!

If the connection cannot fully use the capacity of the link then the transmission speed does not matter.

Performance is limited by the propagation delay ($RTT$).

Thus for most connections, increasing the speed of the link doesn't improve performance!

Per-segment delay is reduced but since transmission speed typically doesn't effect propagation delay, so throughput remains unchanged.

What's the moral here? The window size can't be a constant.

When would the speed of the link effect throughput?

$\text{If } w \times \frac{MSS}{R} < RTT,$

$\text{then the maximum rate at which a TCP connection can transmit data is}$

$$\frac{w \times MSS}{RTT} \text{ bytes/sec}.$$
TCP Congestion Control

Lecture 3e, TCP Congestion Control

In TCP the sender's window is not static. It grows and shrinks adaptively as network conditions change.

Ideally each TCP connection wants to transmit as fast as possible without incurring any loss. Connections want to operate with the largest possible cwin.

cwin is increased until loss occurs.

Threshold is an estimate of the window size that results in throughput that is close to the capacity of the network.

Slow-start takes the window to the threshold very quickly.

Then congestion avoidance takes over and slowly probes around the threshold to see if a larger window is tolerable.

When loss occurs the network is assumed to be saturated.

Initially the congestion window is equal to 1 MSS bytes. Slow start continues until either:

- Loss is detected,
- The slow start/congestion-avoidance threshold is reached.

In reality, the initial threshold is set high enough (e.g., 1 MB (nearly 700 MSS-sized segments) in FreeBSD) that it is never reached.

Packet loss takes the connection out of the initial slow start phase.

Slow start begins again with a threshold equal to half of the congestion window — a decent estimate of the sustainable capacity of the network.

(Slow start means the connections start slow (with 1 MSS) but accelerate rapidly.)
TCP Congestion Control

**Congestion avoidance**

- Increase congestion window by 1 segment each RTT, decrease by a factor of 2 when packet loss is detected
  - "Additive Increase, Multiplicative Decrease" (AICM)

```
/* slowstart is over;
    congWin > threshold
*/
until (loss event) {
    whenever congWin segments ACKed:
        congWin++
/* loss event timeout */
threshold = congWin/2
congWin = 1 MSS perform slowstart
```

**TCP Congestion Control**

**Slow-start vs. Congestion avoidance**

- The threshold is an estimate of a "safe" level of throughput that is sustainable in the network
  - The threshold specifies a throughput that was sustainable in the recent past
- Slow-start quickly increases throughput to this threshold
- Congestion avoidance slows probes for additional available bandwidth beyond the threshold

Assume \( RTT > \frac{w \times MSS}{R} \)
The assumption is that the initial threshold (1 MB) is never reached. Loss will trigger a restart of slow-start with a significantly smaller (more rational) threshold.

How does reducing the threshold reduce the rate?

- The connection backs off to ($cwnd = 1 \frac{MSS}{RTT}$) and then quickly ramps up to $ssthresh \times \frac{MSS}{RTT}$ (1/2 of the pre-loss transmission rate).

- The point is that $cwnd$ determines rate.

Other variants:
- TCP SACK (Selective ACKs):
  - In Fast Recovery, the sender is able to detect multiple losses within a window (and tell which segments were lost).
  - This is an issue with large windows.
  - This can be done with just cumulative ACKs.
- RCP NewReno:
  - Used in non-SACK TCPs.
  - Allows a sender to recover from more than 1 loss during Fast Recovery (but slower than SACK).