

Wednesday, Jan. 30

## TCP:

References:

TCP: <http://www.ietf.org/rfc/rfc793.txt>

Slow Start & other requirements: <http://www.ietf.org/rfc/rfc1122.txt>

TCP extensions: <http://www.ietf.org/rfc/rfc2001.txt>

## Network Comparisons

	Metrics	Switching	Reliability	Flow Control	Net. Deadlock	Protocol/App Deadlock
MRC	↑BW, low latency	Packet	Assumed	HW @ link level •Low overhead •Can block network	x, y Routing	No
IP	↑BW, uncontrolled latency	Packet	no	No/ICMP	Buffer/Drop	Buffering
TCP	↑BW, poor latency	Circuit (socket)	yes	src: flow window dest: sliding window	Buffer/Drop	Buffer/Drop

**Sliding windows:** gives flow control and reliability. Actually, these are two separate mechanisms coupled together. ACKs are sent with window advertisements.

### Other:

### Setup/Teardown:

Flow control  
Lost packet detection

### Other options:

NACKs  
T3E (3-D Torus network). Consulted on by the designer of MRC

### Tricks:

Piggyback ACKs with data if doing 2-way transfers.

Changing window size, i.e. how much pipelining should the network have?

- Congestion control & avoidance
- Slow start

In class, we also talked about ICMP as an IP mechanism for flow control. However, it turns out that the source-quench implementation is not described in the ICMP RFC (RFC 792). Looking back at the BSD code, the source-quench message is actually handled by TCP. I'm not sure how Linux et al. handle this, but I'd imagine it's some similar fashion. Basically, the network layer passes the message up to the transport layer, where TCP checks the error flag, and UDP does not.

TCP:

Sender sends data with sequence # and size

Receiver sends ACK with sequence # and window size

This gets us

1. flow control
2. lost packet detection  
    sender times out if no ack and retransmits

How do we “unglue” flow control and reliability?

Idea: use NACKs

Briefly covered the T3E network

“Return to sender” flow control—like a NACK

Forward coding: ELC

Error checking: CRC