Lecture 4: Sliding Window Wrapup
UDP/TCP Protocols

4.0 Main Points

Review of reliable message delivery
RTT estimation
UDP
TCP

4.1 Review of reliable message delivery

What if message delivery is unreliable?

Stop and wait: send one packet at a time
Receiver acks each packet as it arrives
Sender retransmits if no ack within a timeout.

How do we disambiguate resends?

Include sequence # in each packet; need only 1 bit for stop and wait

How do we keep the pipe full?

Sliding window; send multiple packets without waiting for ack

What if we lose a packet?

Two alternatives:

Go back N: receiver acks "got up through k" packet.
On timeout, sender restarts from k+1.

Selective ack: receiver sends back ack for each packet in window
(Implementation: bit vector)
On timeout, sender resends only missing packet

Can we shortcut the timeout?

Negative ack: assume packets usually arrive in order.
If receiver gets packet k, then immediately send nack for any missing packets, to tell sender to retry immediately.

Similarly, if sender gets acks out of order, can **infer** that packet was lost

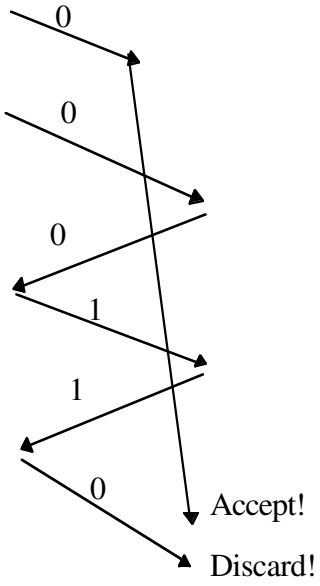
How big should the send/receive windows be?

To fill the pipe: window size = round trip delay * bit rate / packet size

How many sequence numbers do we need for sliding window?

Sender window <= (max seq # + 1) / 2

What if packets can be delivered out of order?



Solutions?

Assume network has bound on how much packet can be delayed.
Never reuse sequence numbers within delay bound.

What happens on reboot?
Assume boot takes max packet delay?

Epoch number: increment high order bits of sequence number on each boot.

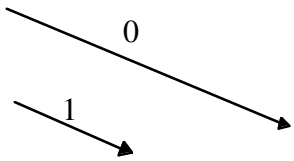
What if link is very lossy?
End to end retransmission will still work, even with go back N!

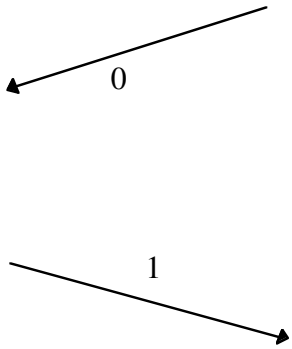
Hop by hop retransmission, as performance optimization, not for correctness.

4.2 RTT estimation

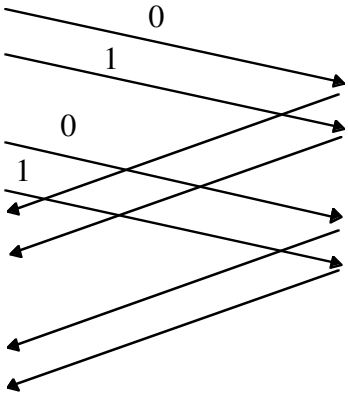
How do we determine timeout?

What if the timeout is too big? Low utilization



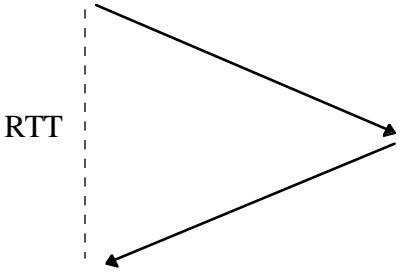


What if timeout is too small? Unnecessary retransmits and redundant acks.



Want transport code to handle any kind of network, from local area Ethernet to wide area Internet. Round trip times may vary by several orders of magnitude.

Solution? Set timeout based on estimate of round trip time, measured by looking at how long acks take to return.



What if RTT changes? (due to rerouting or congestion)

Average samples together over time.

a. Average all samples

RTT = (1/N) * Sum X(i)

b. Windowed average : consider last N samples

c. Exponentially weighted moving average

RTT(k) = Sum X(i) * a^(k-i)

RTT(k) ~ a * X(k) + (1-a) * RTT(k-1)

What if RTT measurements are noisy?

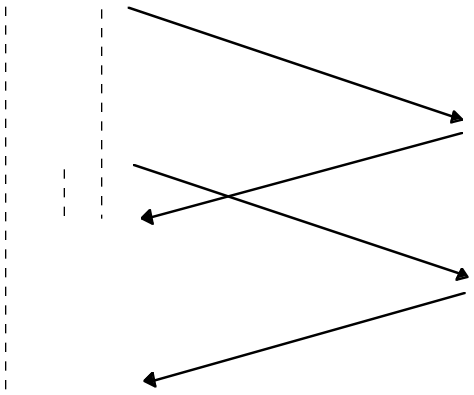
Timeout = estimated RTT?

Timeout = 2 * est RTT?

Timeout = est RTT + k * est variance in RTT

Example of self-tuning system: adapts to changes in underlying hardware.

Retransmission ambiguity: can't distinguish first ack from retransmitted ack



a. Karn-Partridge: ignore RTT estimates for retransmitted packets

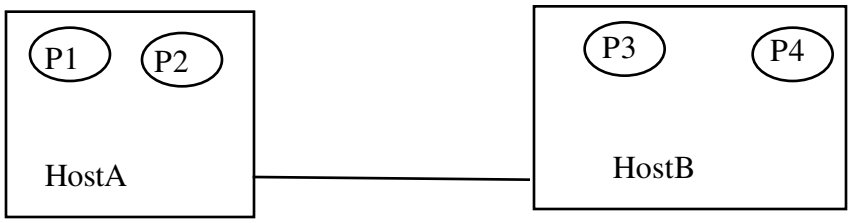
and double timer on every retransmission (until get a successful estimate)

b. add sequence numbers to retransmissions (e.g., retry #1, retry #2)

c. add timestamp to every send; ack returns timestamp

4.3 UDP -- User Datagram Protocol

Process to process communication on top of IP.



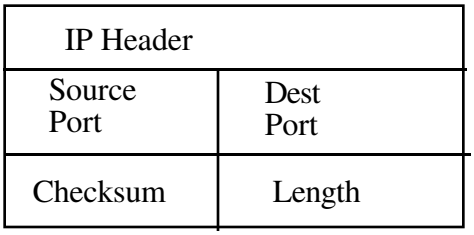
So far, assume host to host communication.

How do processes communcate, e.g., P1 talk to P3, and P2 to P4?
http client (netscape) process talks to http server (httpd) *process*

UDP is one approach:

unreliable message delivery
send to "port" (mailbox) on destination machine
no retransmissions
no sequencing
=> no persistent state

Assumes higher level protocol (e.g., RPC) provides retransmissions, etc.



IP header includes IP address of source, destination machine
Checksum is error detecting code to prevent corrupted packets from being delivered to the application

Target process reads from destination port, replies to source port.

4.4 TCP (Transmission Control Protocol)

Reliable byte stream between user processes
Uses sliding window, go back N, RTT estimation

Connection setup -- buffer sizes and initial sequence #s.