Plan Ahead

Week 4:

- **Reliable data delivery**
- **TCP**

Week 5:

- **Congestion control**
- **Midterm**

Week 6:

- Network layer: Internet Protocol (IP), NAT

Week 7:

- Routing protocols
- Routing in the Internet

Week 8:

- Broadcast and multicast routing
- Data link layer, Ethernet

Week 9:

- Hubs and Switches
- wireless networking

Week 10:

- Mobile networking
- Review

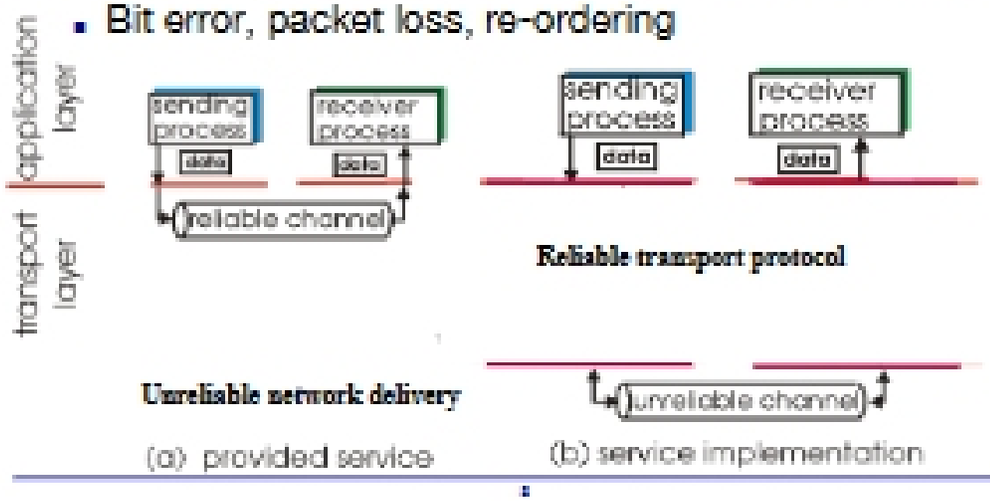
What we learned last week

- DNS: made of 4 pieces

  (1)Name space, (2)authoritative servers, (3)caching resolvers, and (4) the access protocol
- Transport protocol functions
  - Multiplexity and demultiplexing among processes in the same computer
  - For UDP: nothing more
    - Offers unreliable, best-effort delivery
  - For TCP
    - Reliable delivery
    - Flow control
    - Congestion control

How to Achieve Reliable Data Transfer

- consider one-way data transfer (control info will flow in both directions)
- What can go wrong in the unreliable channel?
  - Bit error, packet loss, re-ordering



Error Detection and Recovery

- Bit errors
  - Error detection: data checksum
  - Error recovery: send negative-ACK to sender to retransmit
- re-ordering
  - Error detection: by sequence number
  - Error recovery: receiver can re-sequence the data
- Packet loss
  - Loss detection: alarm timer (at sender end)
  - Loss recovery: retransmit
- Correct reception of data: receiver sends an ACK

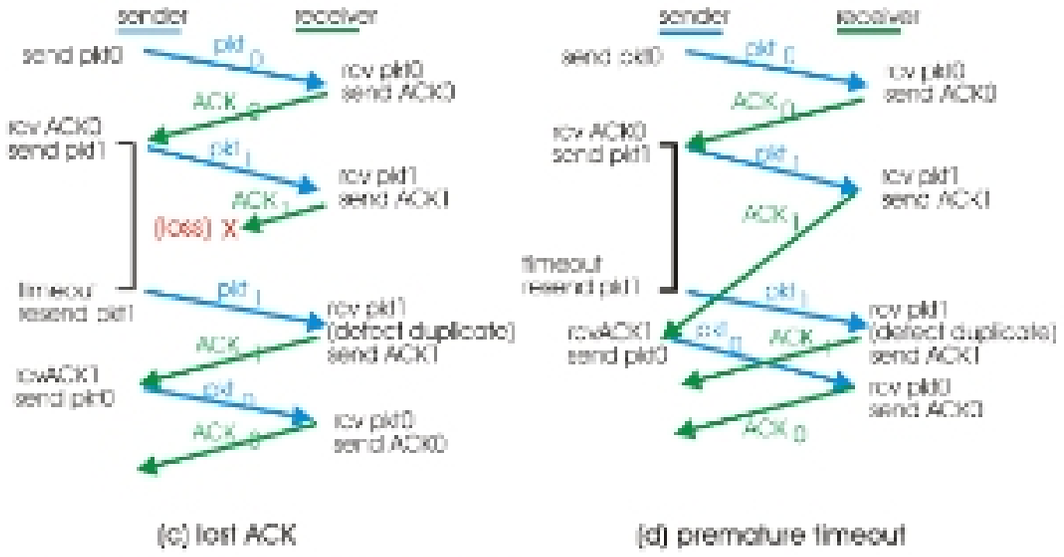
Three basic components in reliable data delivery by retransmission

- sequence #: used to uniquely identify individual packets
- Acknowledgment (ACK): reception report sent by receiver to the sender
- Retransmission timer set by the sender for the already sent, yet to be ack'ed packet
  - Retransmit the packet when timer expires

rdt3.0: channel with bit errors & packet loss

- Sender waits "reasonable" amount of time for ACK from receiver
  - Set up a retransmission timer
- When timeout: retransmits
- In case the packet (or ACK) just delayed but not lost
  - Retransmitted packet will be a duplicate
  - Sequence number can detect this

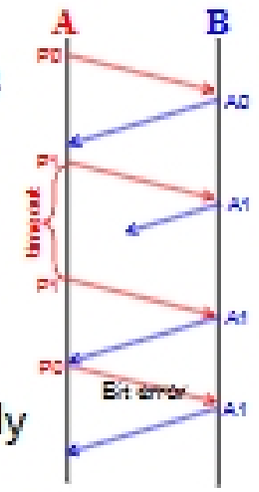
rdt3.0 in action



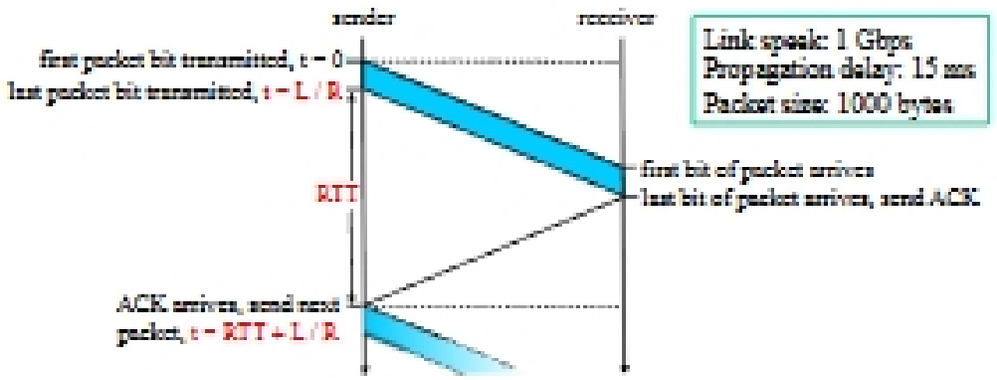
Rdt3.0: a stop-and-wait protocol

Stop-and-Wait

- Sender sends one packet, then waits for receiver's ACK
- Still need a sequence number
  - Consider the case of retransmission
- Speed up recovery: when a received packet is in error, receiver sends an ACK with the seq# of the last correctly received packet
  - When sender sees a duplicate ACK: take same action as NAK: *retransmit current pkt*
- This protocol functions correctly, but suffers from performance problems



rdt3.0: stop-and-wait operation

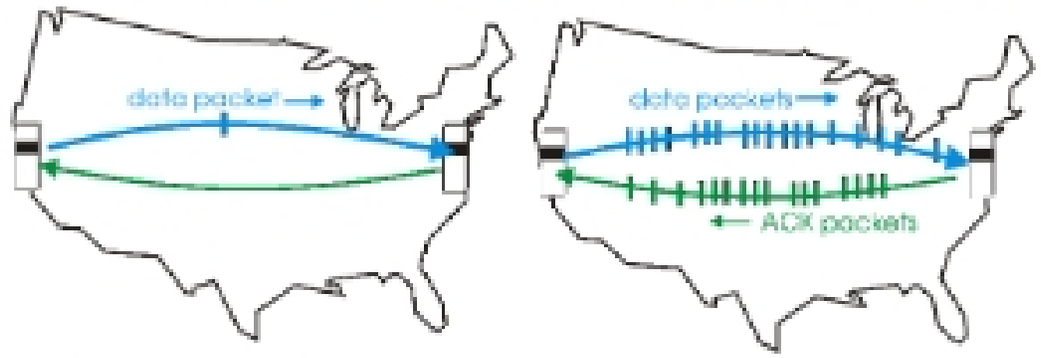


$$d_{trans} = \frac{L}{R} = \frac{8000 \text{bits}}{10^9 \text{bps}} = 8\,\mu s$$

$$U_{sender} = \frac{d_{trans}}{RTT + d_{trans}} = \frac{0.008ms}{30.008ms} = 0.00027$$

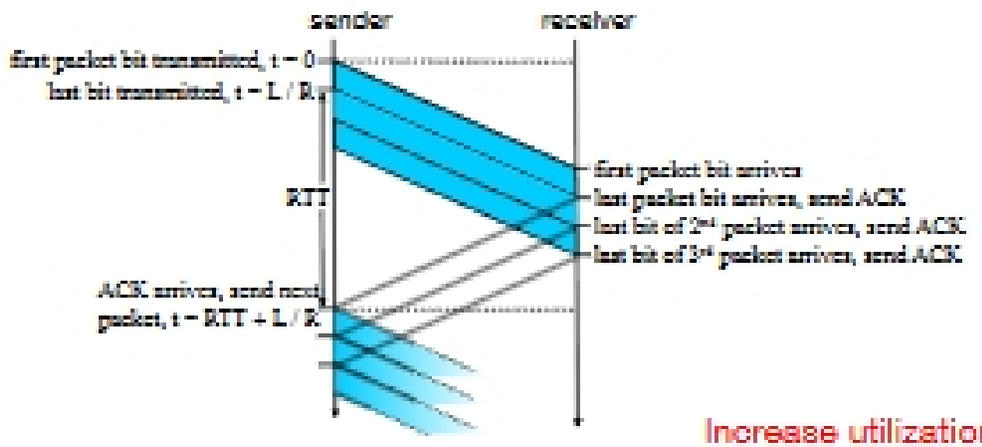
Pipelining transmission

- Allowing multiple, yet-to-be-acknowledged, packets to be "in-flight"
  - range of sequence numbers must increase
  - Need packet buffering at sender (and receiver too)



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

Pipelining: increased utilization



Increase utilization by a factor of 3!

$$U_{sender} = \frac{3 \times \frac{L}{R}}{RTT + \frac{L}{R}} = \frac{0.024}{30.008} = 0.0008$$

What if some packets get lost?

(1) Go-Back-N (GBN)

- Sender can send up to N unack'ed packets
  - N = Flow control window size
- Receiver only sends cumulative acks
  - Doesn't ack packet if there's any gap
- Sender has timer for oldest unack'ed packet
  - If timer expires, retransmit ***all*** unack'ed packets