Reliable Transport Fundamentals

Daniel Zappala

CS 460 Computer Networking
Brigham Young University

How do you reliably send data across an unreliable network?

Components

- positive acknowledgements (ACKs) or negative acknowledgements (NACKs)
- timers
- selective or cumulative ACKs

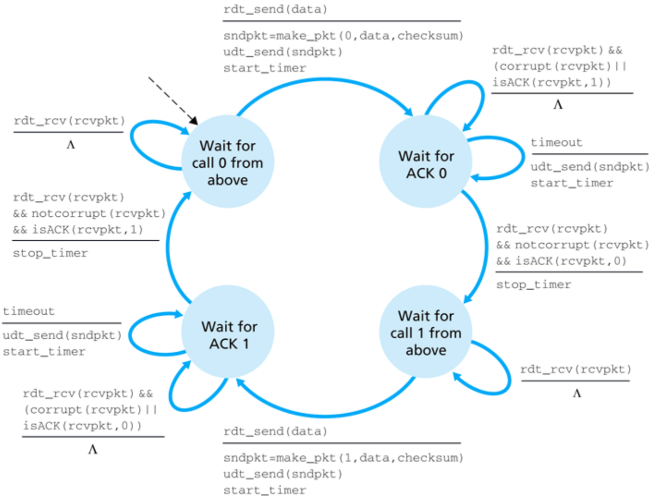
$\Rightarrow$ in the context of three different protocols

Stop-and-Wait

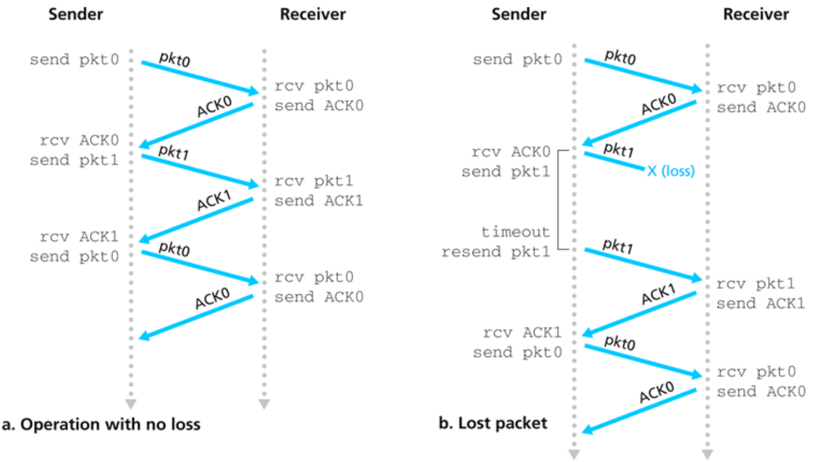
RDT Roadmap

- RDT 1.0: the network is reliable
  - a transport protocol can be viewed as a finite state machine
- RDT 2.0: the network can cause bit errors
  - need a checksum to detect errors
  - send an ACK or NACK
  - retransmit data upon receiving a NACK
- RDT 2.1: ACKs, NACKs can be corrupted
  - retransmit when ACK or NACK is corrupted
  - need sequence numbers to detect duplicates – if an ACK is corrupted you're re-sending data that the receiver already has
- RDT 2.2: eliminate NACKs
- RDT 3.0: network can also lose packets
  - need a timer in case packet or ACK lost
  - retransmit if timer expires before ACK received

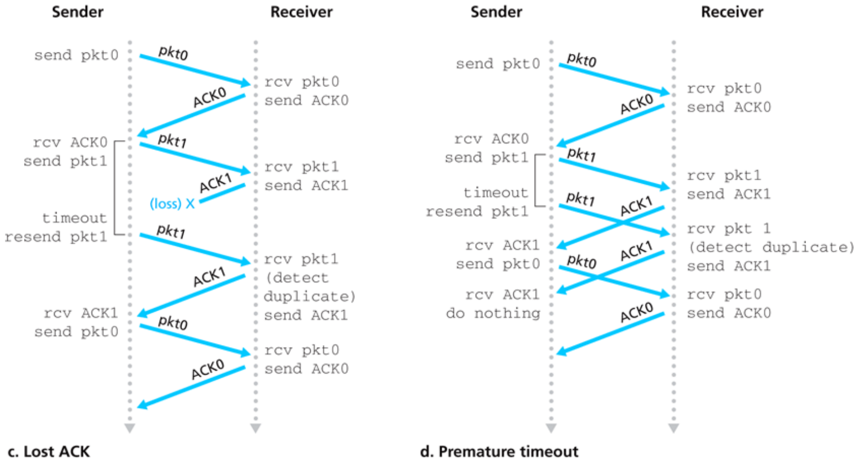
RDT 3.0 Sender



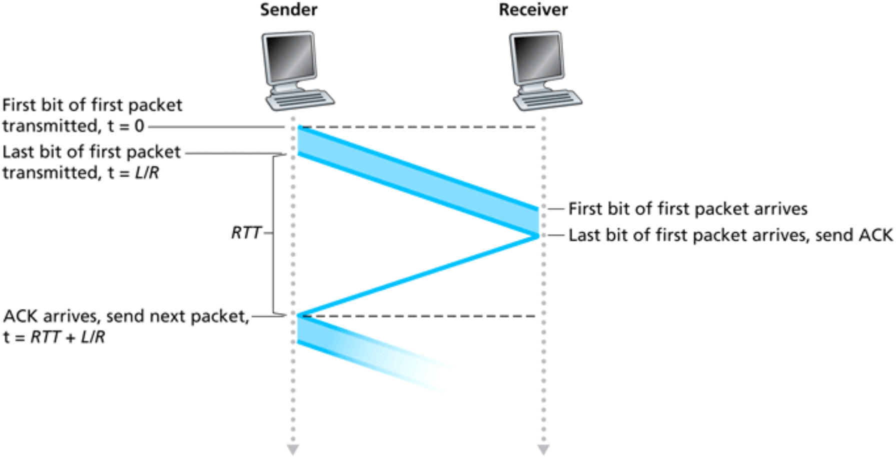
RDT 3.0 Packet Trace



RDT 3.0 Packet Trace



RDT 3.0 = Stop-and-Wait



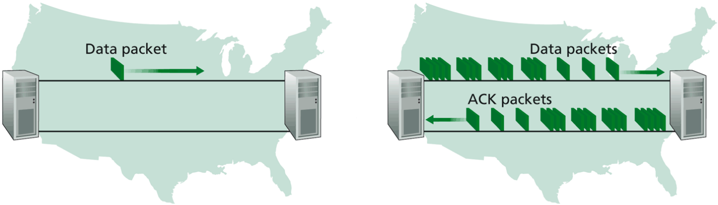
Stop-and-Wait Performance

- example: 1 Gbps link, 15 ms propagation delay, 1000 byte packet
- calculate U_{sender}: utilization : fraction of time sender is busy sending
  - $T_{transmit} = \frac{L}{R} = \frac{8kb}{10^9 bps} = 8\mu s$
  - $U_{sender} = \frac{L/R}{RTT+L/R} = \frac{0.008}{30.008} = 0.00027 = .027\%$
- 1000 bits every 30 ms = 33 kbps over a 1 Gbps link
- performance is lousy

$\Rightarrow$ transport protocol limits use of physical resources

Go-Back-N

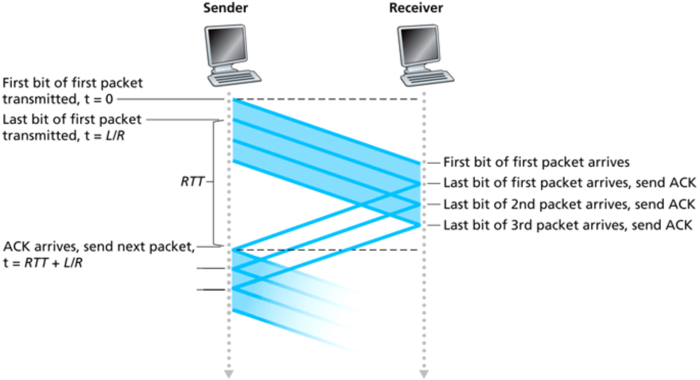
Pipelining



a. A stop-and-wait protocol in operation b. A pipelined protocol in operation

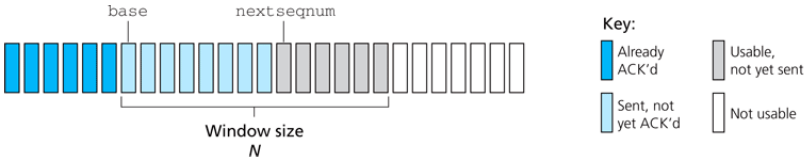
- send multiple packets at a time, wait for ACKs
  - must increase sequence number space
  - need buffering at sender and receiver

Increased Utilization



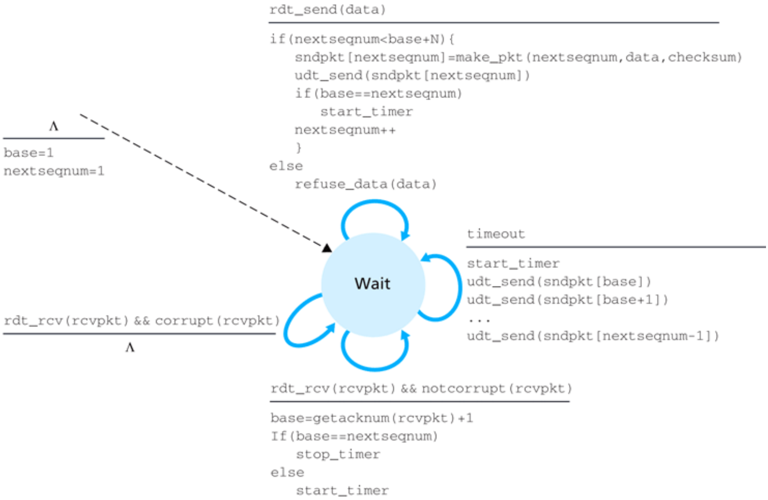
- send 1000 packets at a time
- $U_{sender} = \frac{1000 * L/R}{RTT + L/R} = \frac{8}{38} = 0.21 = 21\%$

Go-Back-N Overview

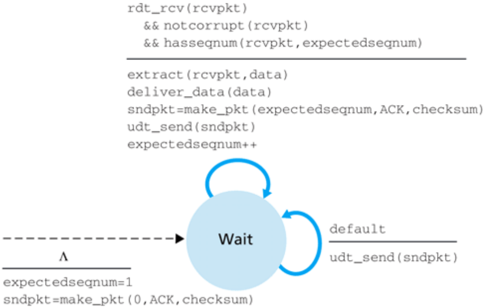


- sender keeps a *window* of packets
  - window represents a series of consecutive sequence numbers
  - window size N : number of un-ACKed packets allowed
- cumulative ACKs
  - $ACK(n)$: acks packets up to and including n
  - sender may receive duplicate acks
- go back n
  - sender keeps a timer for each packet
  - $timeout(n)$: retransmit packet n and all higher packets
  - **no receiver buffering!**

Go-Back-N Sender FSM

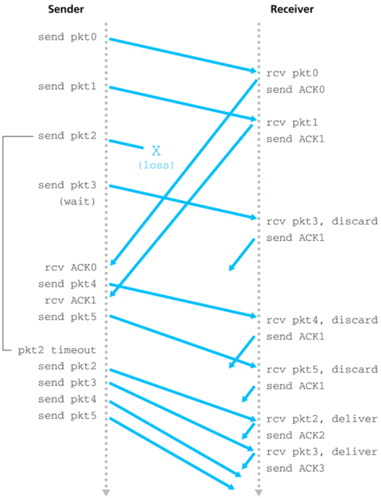


Go-Back-N Receiver



- cumulative ACK
  - always send ACK for in-order packet with highest sequence number
  - may generate duplicate ACKs
  - only state is expected sequence number
- **out-of-order packets are discarded : no buffering**

Go-Back-N Packet Trace



- warning: this will lead to congestion collapse
- bad reaction to congestion: send more packets!
- good reaction to congestion: slow down
- more on this when we visit TCP

Selective Repeat

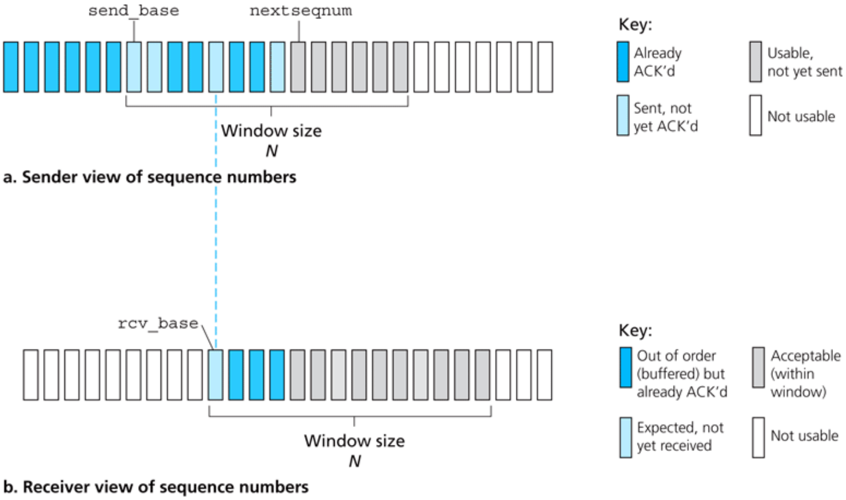
Selective Repeat Overview

- sender keeps a *window* of packets
  - window represents a series of consecutive sequence numbers
  - window size N : number of un-ACKed packets allowed

$\Rightarrow$ same as Go-Back-N

- selective ACKs
  - $ACK(n)$: ACKS only sequence number n
- selective repeat
  - sender keeps a timer for each packet
  - $timeout(n)$: retransmit packet n only
  - **receiver must buffer out-of-order packets!**

Selective Repeat Sender and Receiver Windows



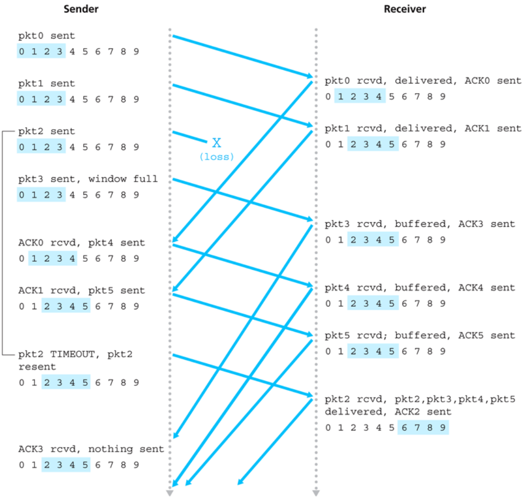
Selective Repeat Sender

```
def data():
    if next available sequence number in window:
        send packet

def timeout(n):
    resend packet n
    restart timer

def ACK(n):
    if n not in [sendbase, sendbase + N]
        return
    mark packet n as received
    if n smallest un-ACKed packet:
        advance sendbase to next un-ACKed sequence number
    if buffered packets can be sent:
        send packets
```

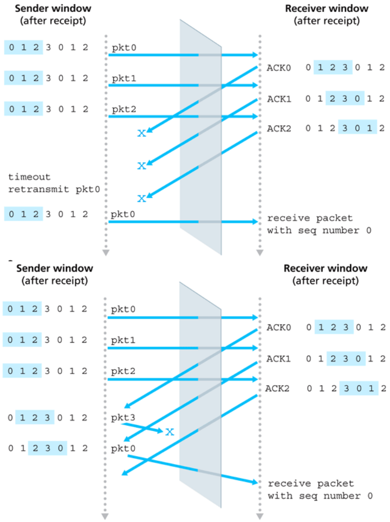
Selective Repeat Receiver

```
def packet():
    if n in [rcvbase, rcvbase + N − 1]:
        send ACK(n)
        if packet is out of order:
            buffer packet
        else:
            deliver all in−order packets
            advance rcvbase to next not−yet−received packet
    else if n in [rcvbase − N, rcvbase − 1]:
        send ACK(n)
    else:
        return
```

Selective Repeat Packet Trace

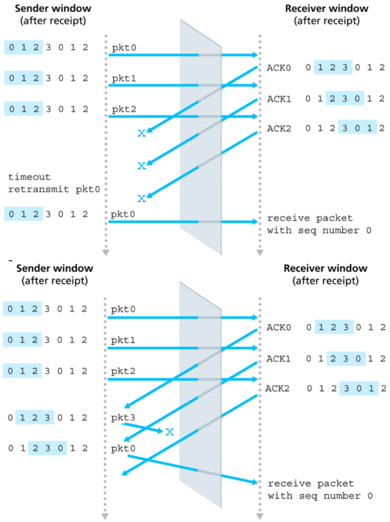


Selective Repeat Window and Sequence Number Sizes



- example
  - 2-bit sequence number space
  - window size = 3
- receiver can't tell difference between old and new packet 0
- how large should sequence space be?

Selective Repeat Window and Sequence Number Sizes



- example
  - 2-bit sequence number space
  - window size $= 3$
- receiver can't tell difference between old and new packet 0
- how large should sequence space be?
- sequence number size $>= 2*$ window size
- RFC 1323: TCP sequence space large enough for handling duplicates 3 minutes later

TCP does not use Stop and Wait, Go-Back-N, or Selective Repeat

TCP

- this lecture shows you various design options
- the following lecture will explain how TCP implements reliability