Data Communications & Networks

Session 6 – Main Theme
Reliable Data Transfer

Dr. Jean-Claude Franchitti

New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Agenda

What is the class about?

- Course description and syllabus:
  - » http://www.nyu.edu/classes/jcf/g22.2262-001/
  - » http://www.cs.nyu.edu/courses/spring10/G22.2262-001/index.html
- Textbooks:
  - » ***Computer Networking: A Top-Down Approach (5th Edition)***

James F. Kurose, Keith W. Ross
Addison Wesley
ISBN-10: 0136079679, ISBN-13: 978-0136079675, 5th Edition (03/09)

Course Overview

- Computer Networks and the Internet
- Application Layer
- Fundamental Data Structures: queues, ring buffers, finite state machines
- Data Encoding and Transmission
- Local Area Networks and Data Link Control
- Wireless Communications
- Packet Switching
- OSI and Internet Protocol Architecture
- Congestion Control and Flow Control Methods
- Internet Protocols (IP, ARP, UDP, TCP)
- Network (packet) Routing Algorithms (OSPF, Distance Vector)
- IP Multicast
- Sockets

Reliable Data Transfer Session in Brief

- Principles of Reliable Data Transfer
- Reliable Data Transfer: Getting Started
- Reliable Data Transfer: Operational Details
- Other Reliable Data Transfer Protocols
- Conclusion

Icons / Metaphors


Information


Common Realization

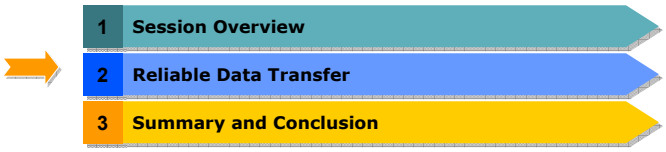

Knowledge/Competency Pattern


Governance


Alignment


Solution Approach

Agenda

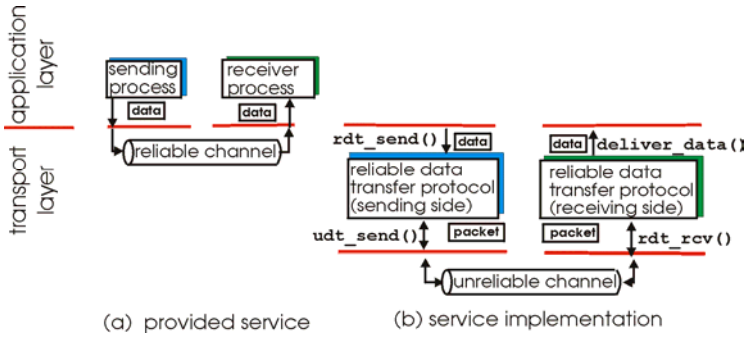


Reliable Data Transfer Session in Brief

- Principles of Reliable Data Transfer
- Reliable Data Transfer: Getting Started
- Reliable Data Transfer: Operational Details
- Other Reliable Data Transfer Protocols

Principles of Reliable Data Transfer

- Important in app., transport, link layers
- Top-10 list of important networking topics!

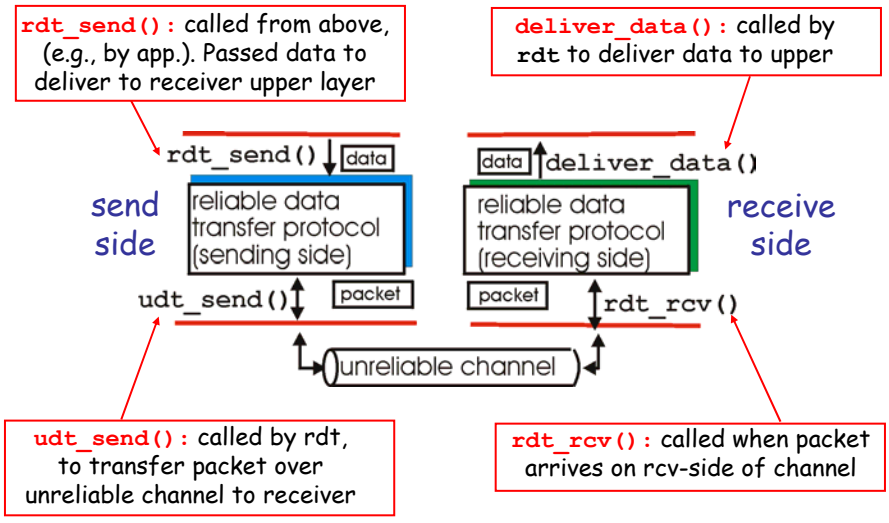


- Characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Reliable Data Transfer Session in Brief

- Principles of Reliable Data Transfer
- Reliable Data Transfer: Getting Started
- Reliable Data Transfer: Operational Details
- Other Reliable Data Transfer Protocols

Reliable Data Transfer: Getting Started

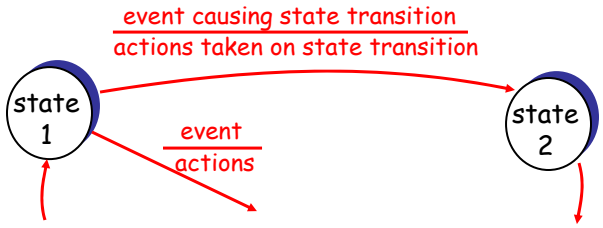


Reliable Data Transfer: Getting Started

We'll:

- Incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- Consider only unidirectional data transfer
  - But control info will flow on both directions!
- Use finite state machines (FSM) to specify sender, receiver

state: when in this "state" next state uniquely determined by next event

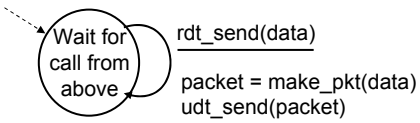


Reliable Data Transfer Session in Brief

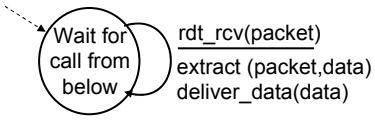
- Principles of Reliable Data Transfer
- Reliable Data Transfer: Getting Started
- Reliable Data Transfer: Operational Details
- Other Reliable Data Transfer Protocols

Rdt1.0 - Reliable Transfer Over a Reliable Channel

- Underlying channel perfectly reliable
  - No bit errors
  - No loss of packets
- Separate FSMs for sender, receiver:
  - Sender sends data into underlying channel
  - Receiver read data from underlying channel



sender

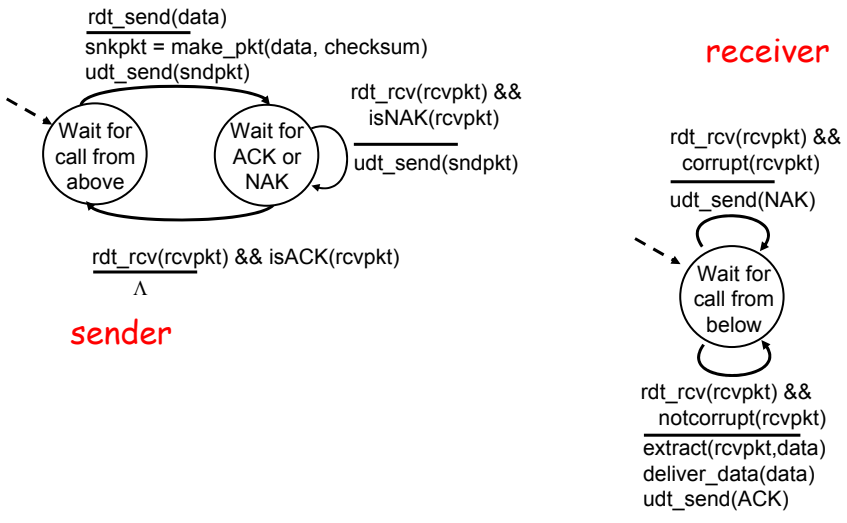


receiver

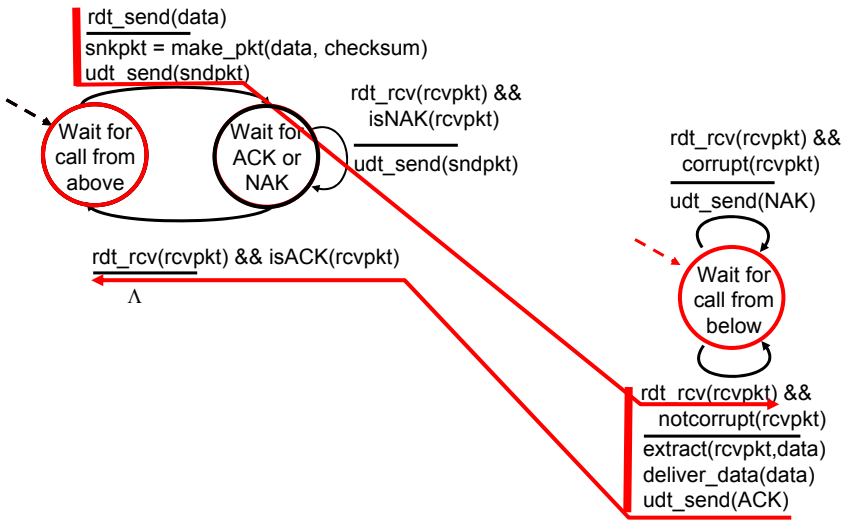
Rdt2.0: Channel with Bit Errors

- Underlying channel may flip bits in packet
  - Checksum to detect bit errors
- *The* question: how to recover from errors:
  - *Acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *Negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - Sender retransmits pkt on receipt of NAK
- New mechanisms in **rdt2.0** (beyond **rdt1.0**):
  - Error detection
  - Receiver feedback: control msgs (ACK,NAK) rcvr->sender

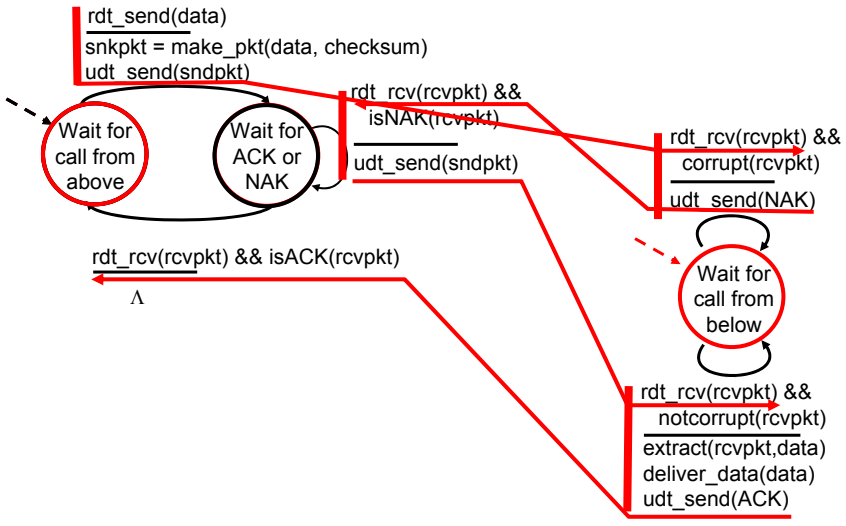
Rdt2.0: FSM Specification



Rdt2.0: Operation with No Error



Rdt2.0: Error Scenario



Rdt 2.0 Has a Fatal Flaw!

What happens if ACK/NAK corrupted?

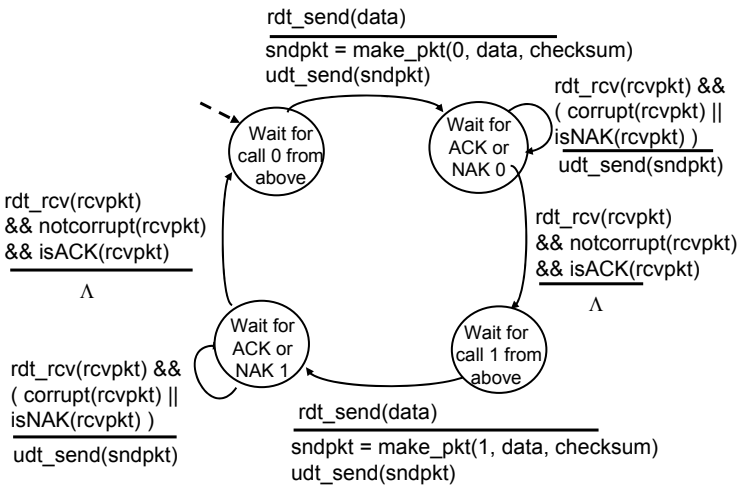
- Sender doesn't know what happened at receiver!
- Can't just retransmit: possible duplicate

Handling duplicates:

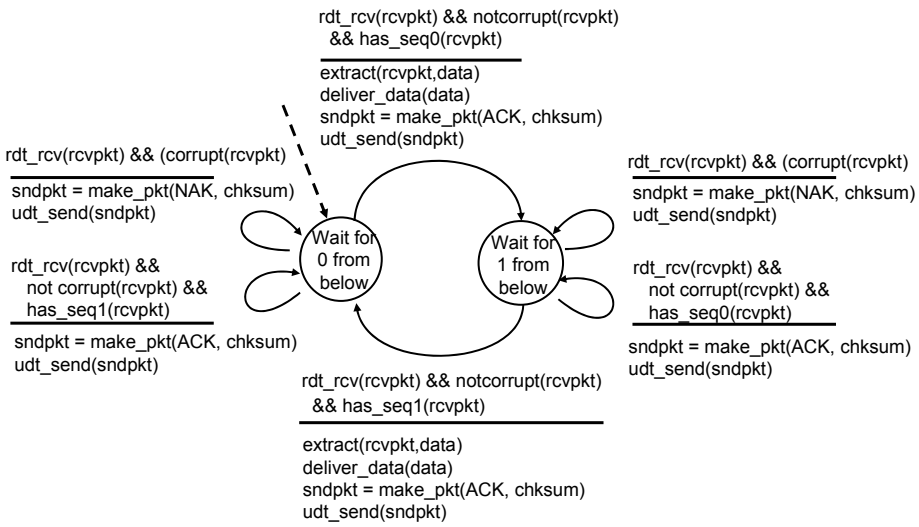
- Sender retransmits current pkt if ACK/NAK garbled
- Sender adds *sequence number* to each pkt
- Receiver discards (doesn't deliver up) duplicate pkt

stop and wait
Sender sends one packet, then waits for receiver response

Rdt2.1: Sender Handles Garbled ACK/NAKs



Rdt2.1: Receiver Handles Garbled ACK/NAKs



Rdt 2.1: Discussion

Sender:

- Seq # added to pkt
- Two seq. #'s (0,1) will suffice. Why?
- Must check if received ACK/NAK corrupted
- Twice as many states
  - » state must "remember" whether "current" pkt has 0 or 1 seq. #

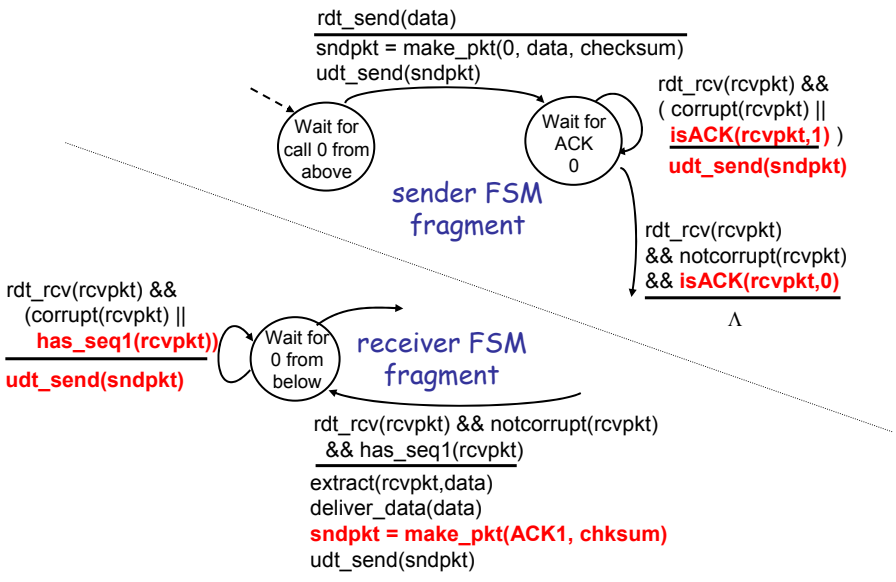
Receiver:

- Must check if received packet is duplicate
  - » State indicates whether 0 or 1 is expected pkt seq #
- Note: receiver can *not* know if its last ACK/NAK received OK at sender

Rdt2.2: A NAK-Free Protocol

- Same functionality as rdt2.1, using ACKs only
- Instead of NAK, receiver sends ACK for last pkt received OK
  - Receiver must *explicitly* include seq # of pkt being ACKed
- Duplicate ACK at sender results in same action as NAK: *retransmit current pkt*

Rdt2.2: Sender, Receiver Fragments



Rdt 3.0: Channels with Errors and Loss

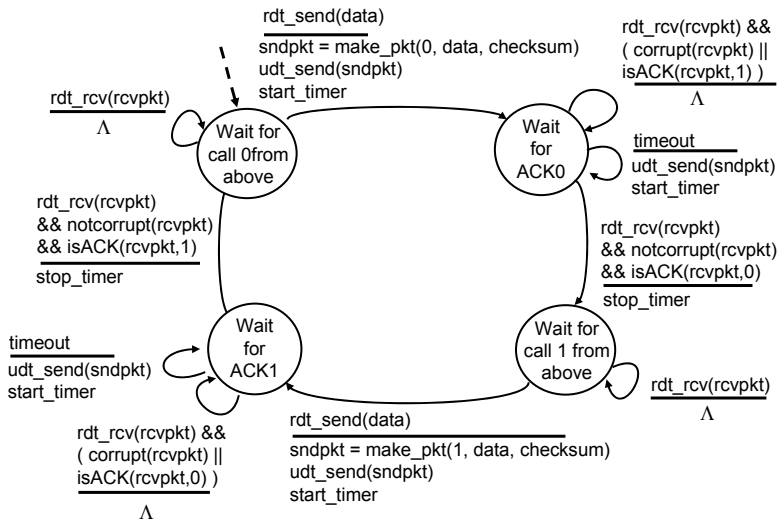
New Assumption:

- Underlying channel can also lose packets (data or ACKs)
  - » checksum, seq. #, ACKs, retransmissions will be of help, but not enough

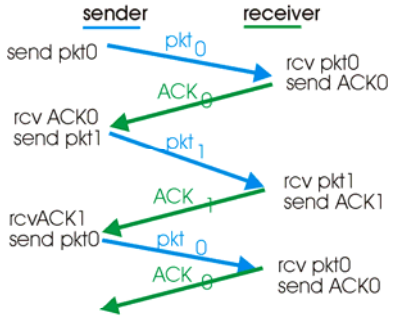
Approach: sender waits "reasonable" amount of time for ACK

- Retransmits if no ACK received in this time
- If pkt (or ACK) just delayed (not lost):
  - » Retransmission will be duplicate, but use of seq. #'s already handles this
  - » Receiver must specify seq # of pkt being ACKed
- Requires countdown timer

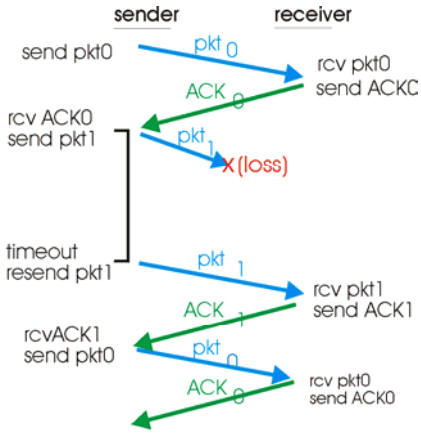
Rdt3.0: Sender



Rdt3.0 in Action

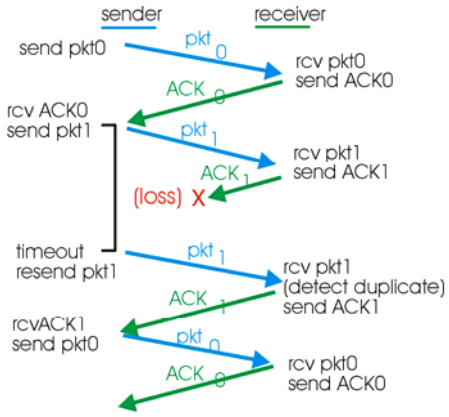


(a) operation with no loss

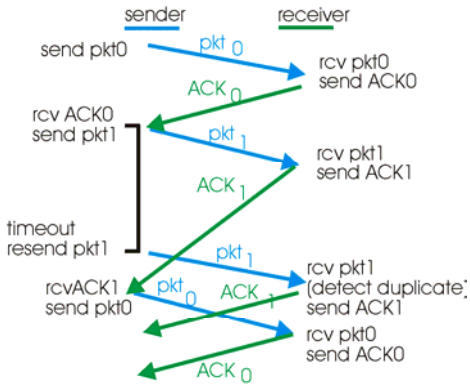


(b) lost packet

Rdt3.0 in Action



(c) lost ACK



(d) premature timeout

Performance of Rdt 3.0

- Rdt3.0 works, but performance stinks
- Example: 1 Gbps link, 15 ms e-e prop. delay, 1KB packet:

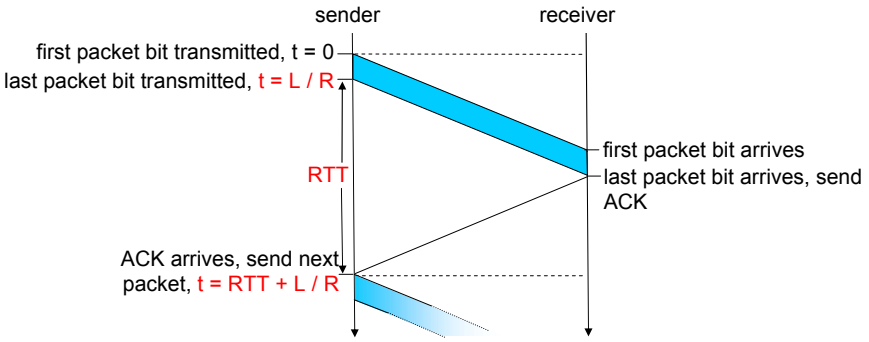
$$T_{transmit} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8kb/pkt}{10^{**}9 \text{ b/sec}} = 8 \text{ microsec}$$

  - » U_{sender}: utilization – fraction of time sender busy sending

$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027 \text{ microsec}$$

  - » 1KB pkt every 30 msec -> 33kB/sec thruput over 1 Gbps link
  - » network protocol limits use of physical resources!

Rdt 3.0: Stop-and-Wait Operation



$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027 \text{ microsec}$$

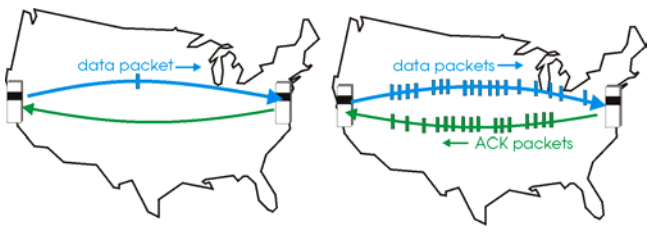
Reliable Data Transfer Session in Brief

- Principles of Reliable Data Transfer
- Reliable Data Transfer: Getting Started
- Reliable Data Transfer: Operational Details
- Other Reliable Data Transfer Protocols
- Conclusion

Pipelined Protocols

Pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts

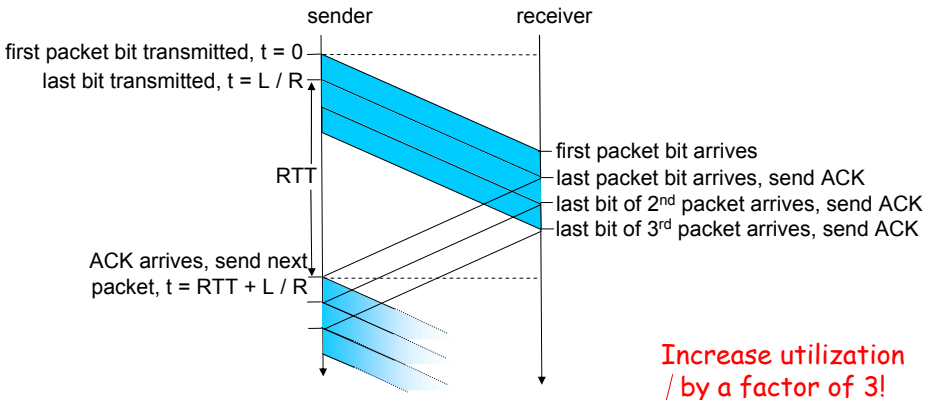
- Range of sequence numbers must be increased
- Buffering at sender and/or receiver



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

- Two generic forms of pipelined protocols: *go-Back-N, selective repeat*

Pipelining: Increased Utilization



Increase utilization by a factor of 3!

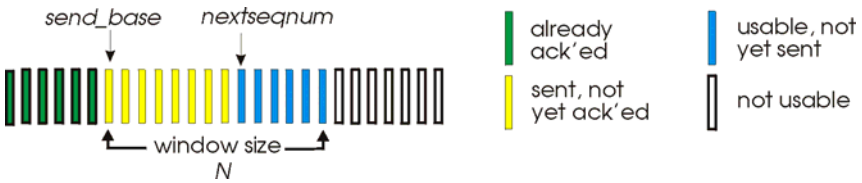
$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

microsecon

Go-Back-N

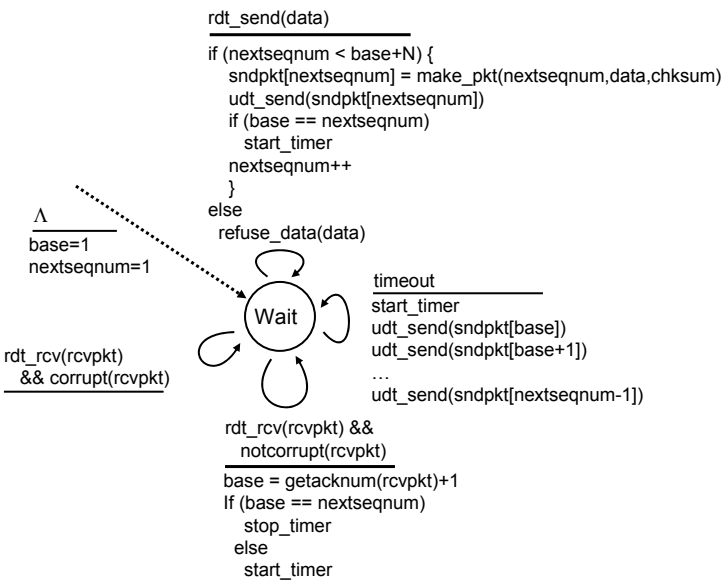
Sender:

- k-bit seq # in pkt header
- "window" of up to N, consecutive unack'ed pkts allowed

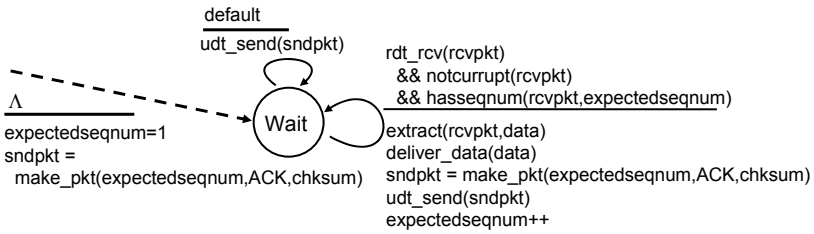


- ACK(n): ACKs all pkts up to, including seq # n - "cumulative ACK"
  - May receive duplicate ACKs (see receiver)
- Timer for each in-flight pkt
- *timeout(n):* retransmit pkt n and all higher seq # pkts in window

GBN: Sender Extended FSM

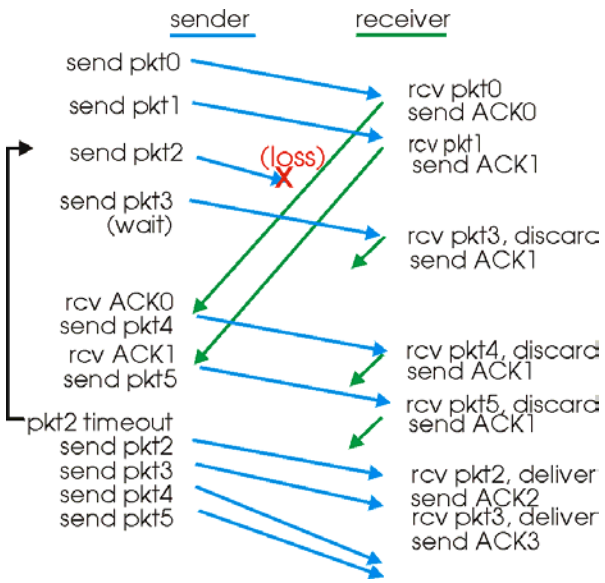


GBN: Receiver Extended FSM



- ACK-only: always send ACK for correctly-received pkt with highest *in-order* seq #
  - May generate duplicate ACKs
  - Need only remember **expectedseqnum**
- Out-of-order pkt:
  - Discard (don't buffer) -> no receiver buffering!
  - Re-ACK pkt with highest in-order seq #

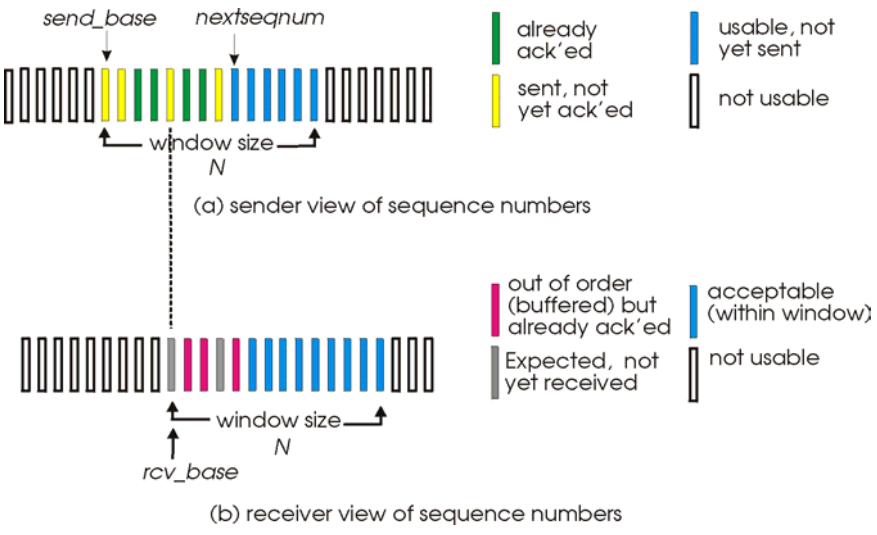
GBN In Action



Selective Repeat

- Receiver *individually* acknowledges all correctly received pkts
  - Buffers pkts, as needed, for eventual in-order delivery to upper layer
- Sender only resends pkts for which ACK not received
  - Sender timer for each unACKed pkt
- Sender window
  - N consecutive seq #'s
  - Again limits seq #s of sent, unACKed pkts

Selective Repeat: Sender, Receiver Windows



Selective Repeat

sender

Data from above:

- If next available seq # in window, send pkt

Timeout(n):

- Resend pkt n, restart timer

ACK(n) in [sendbase,sendbase+N]:

- Mark pkt n as received
- If n smallest unACKed pkt, advance window base to next unACKed seq #

receiver

Pkt n in [rcvbase, rcvbase+N-1]

- Send ACK(n)
- Out-of-order: buffer
- In-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

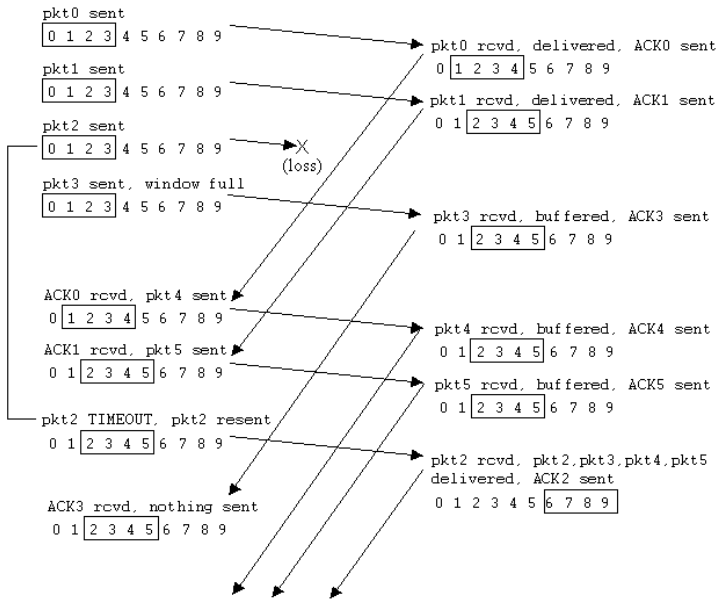
Pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

Otherwise:

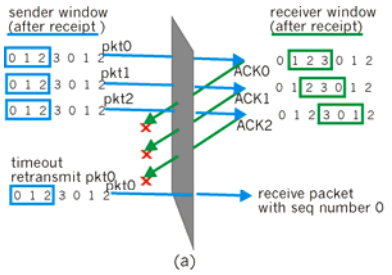
- Ignore

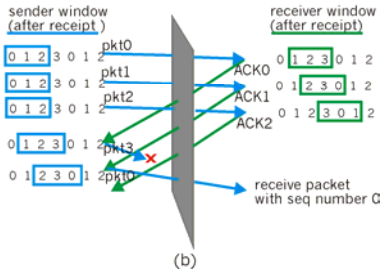
Selective Repeat in Action



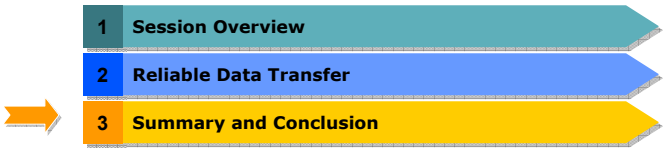
Selective Repeat: Dilemma

- Example:
  - Seq #'s: 0, 1, 2, 3
  - Window size=3
  - Receiver sees no difference in two scenarios!
  - Incorrectly passes duplicate data as new in (a)
- Q: what relationship between seq # size and window size?





Agenda



Summary

- Principles of Reliable Data Transfer
- Reliable Data Transfer: Getting Started
- Reliable Data Transfer: Operational Details
- Other Reliable Data Transfer Protocols

Assignments & Readings

- Readings

  - Chapter 3
- Assignment #4
  - Due March 25 2010

Next Session: Networks - Part I