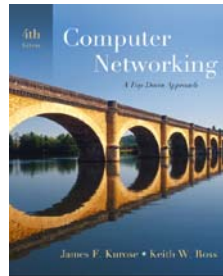


Chapter 3 Transport Layer



*Computer Networking:
A Top Down Approach*
4th edition.
Jim Kurose, Keith Ross
Addison-Wesley, July
2007.

Transport Layer 3-1

Chapter 3: Transport Layer

Our goals:

- understand principles behind transport layer services:
 - multiplexing/demultiplexing
 - reliable data transfer
 - flow control
 - congestion control
- learn about transport layer protocols in the Internet:
 - UDP: connectionless transport
 - TCP: connection-oriented transport
 - TCP congestion control

Transport Layer 3-2

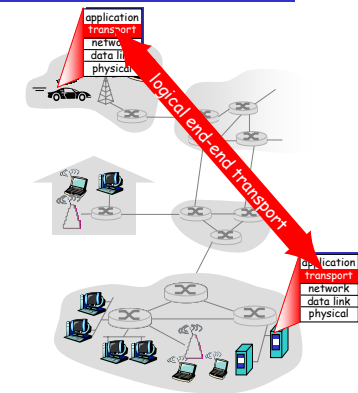
Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
 - segment structure
 - reliable data transfer
 - flow control
 - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Transport Layer 3-3

Transport services and protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
 - send side: breaks app messages into **segments**, passes to network layer
 - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
 - Internet: TCP and UDP



Transport Layer 3-4

Transport vs. network layer

- **network layer:** logical communication between hosts
- **transport layer:** logical communication between processes
 - relies on, enhances, network layer services

Household analogy:

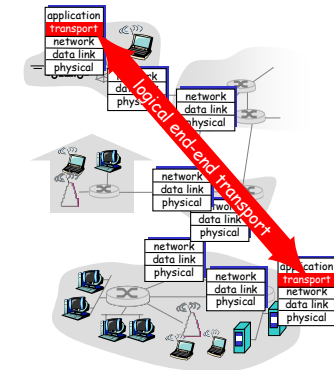
12 kids sending letters to 12 kids

- processes = kids
- app messages = letters in envelopes
- hosts = houses
- transport protocol = Ann and Bill
- network-layer protocol = postal service

Transport Layer 3-5

Internet transport-layer protocols

- reliable, in-order delivery (TCP)
 - congestion control
 - flow control
 - connection setup
- unreliable, unordered delivery: UDP
 - no-frills extension of "best-effort" IP
- services not available:
 - delay guarantees
 - bandwidth guarantees



Transport Layer 3-6

Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
 - segment structure
 - reliable data transfer
 - flow control
 - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

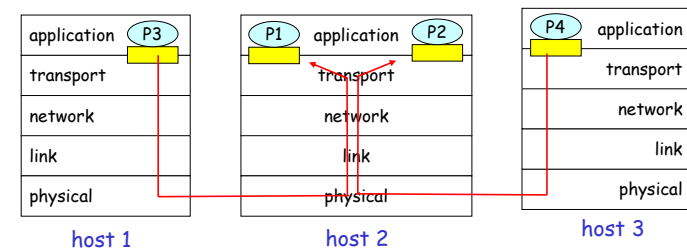
Transport Layer 3-7

Multiplexing/demultiplexing

Demultiplexing at rcv host:
delivering received segments to correct socket

Multiplexing at send host:
gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)

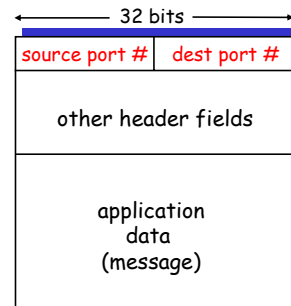
■ = socket ○ = process



Transport Layer 3-8

How demultiplexing works

- host receives IP datagrams
 - each datagram has source IP address, destination IP address
 - each datagram carries 1 transport-layer segment
 - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



Transport Layer 3-9

Connectionless demultiplexing

- Create sockets with port numbers:

```
DatagramSocket mySocket1 = new DatagramSocket(12534);
```

```
DatagramSocket mySocket2 = new DatagramSocket(12535);
```

- UDP socket identified by two-tuple:

(dest IP address, dest port number)

- When host receives UDP segment:

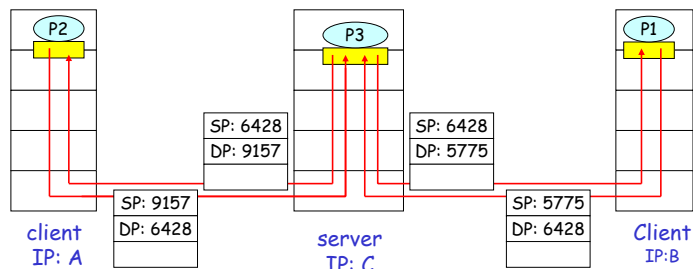
- checks destination port number in segment
- directs UDP segment to socket with that port number

- IP datagrams with different source IP addresses and/or source port numbers directed to same socket

Transport Layer 3-10

Connectionless demux (cont)

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```



SP provides "return address"

Transport Layer 3-11

Connection-oriented demux

- TCP socket identified by 4-tuple:

- source IP address
- source port number
- dest IP address
- dest port number

- recv host uses all four values to direct segment to appropriate socket

- Server host may support many simultaneous TCP sockets:

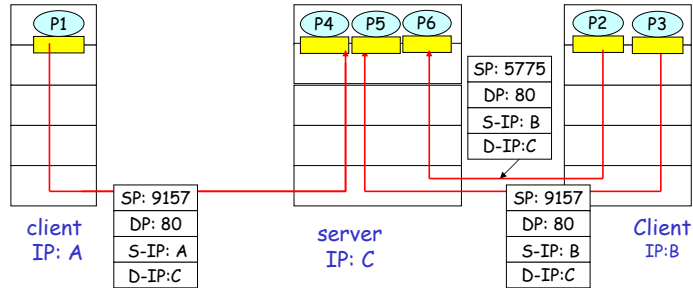
- each socket identified by its own 4-tuple

- Web servers have different sockets for each connecting client

- non-persistent HTTP will have different socket for each request

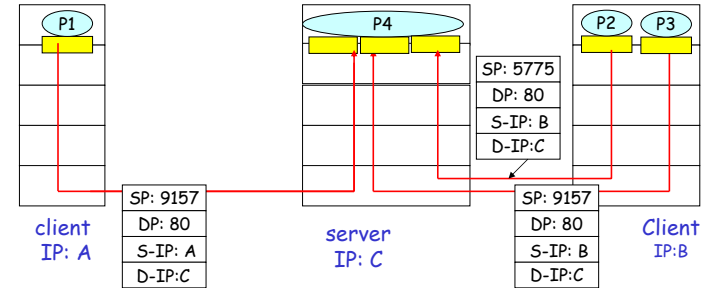
Transport Layer 3-12

Connection-oriented demux (cont)



Transport Layer 3-13

Connection-oriented demux: Threaded Web Server



Transport Layer 3-14

Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 **Connectionless transport: UDP**
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
 - segment structure
 - reliable data transfer
 - flow control
 - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Transport Layer 3-15

UDP: User Datagram Protocol [RFC 768]

- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
 - lost
 - delivered out of order to app
- **connectionless:**
 - no handshaking between UDP sender, receiver
 - each UDP segment handled independently of others

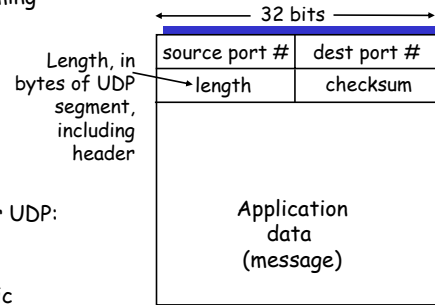
Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

Transport Layer 3-16

UDP: more

- often used for streaming multimedia apps
 - loss tolerant
 - rate sensitive
- other UDP uses
 - DNS
 - SNMP
- reliable transfer over UDP: add reliability at application layer
 - application-specific error recovery!



UDP segment format

Transport Layer 3-17

UDP checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

Sender:

- treat segment contents as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
 - NO - error detected
 - YES - no error detected. *But maybe errors nonetheless? More later*

Transport Layer 3-18

Internet Checksum Example

- Note
 - When adding numbers, a carryout from the most significant bit needs to be added to the result
- Example: add two 16-bit integers

```

      1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
      1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
      -----
wraparound 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
              |-----|
sum          1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
checksum    0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1
    
```

Transport Layer 3-19

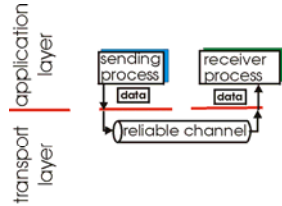
Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
 - segment structure
 - reliable data transfer
 - flow control
 - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Transport Layer 3-20

Principles of Reliable data transfer

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



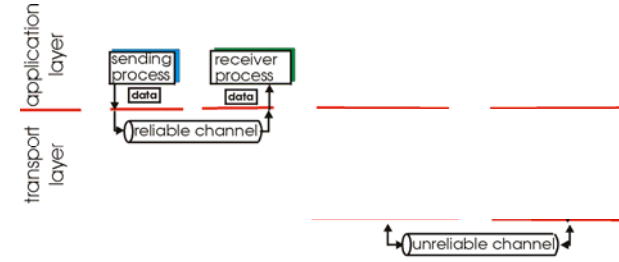
(a) provided service

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

Transport Layer 3-21

Principles of Reliable data transfer

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



(a) provided service

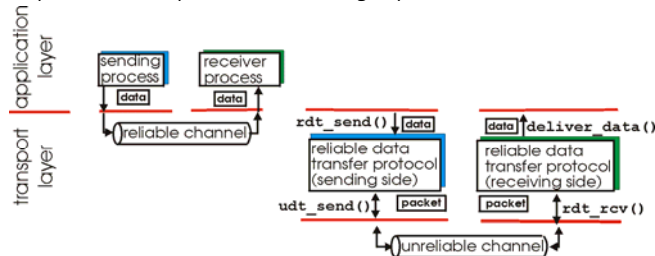
(b) service implementation

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

Transport Layer 3-22

Principles of Reliable data transfer

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



(a) provided service

(b) service implementation

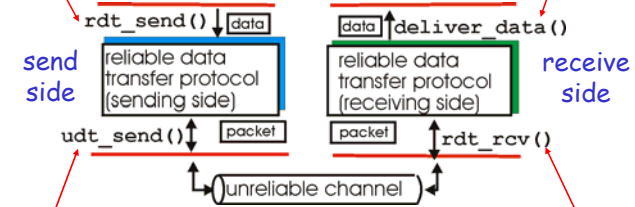
- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Transport Layer 3-23

Reliable data transfer: getting started

rdt_send(): called from above, (e.g., by app.). Passed data to deliver to receiver upper layer

deliver_data(): called by rdt to deliver data to upper



udt_send(): called by rdt, to transfer packet over unreliable channel to receiver

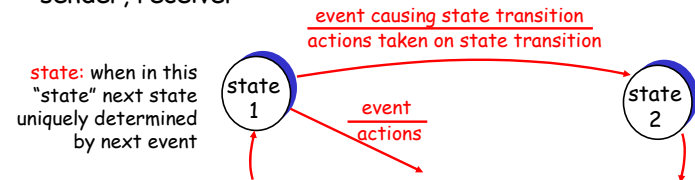
rdt_rcv(): called when packet arrives on rcv-side of channel

Transport Layer 3-24

Reliable data transfer: getting started

We'll:

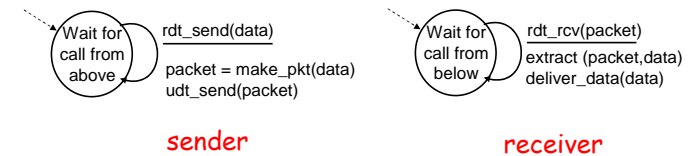
- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only unidirectional data transfer
 - but control info will flow on both directions!
- use finite state machines (FSM) to specify sender, receiver



Transport Layer 3-25

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



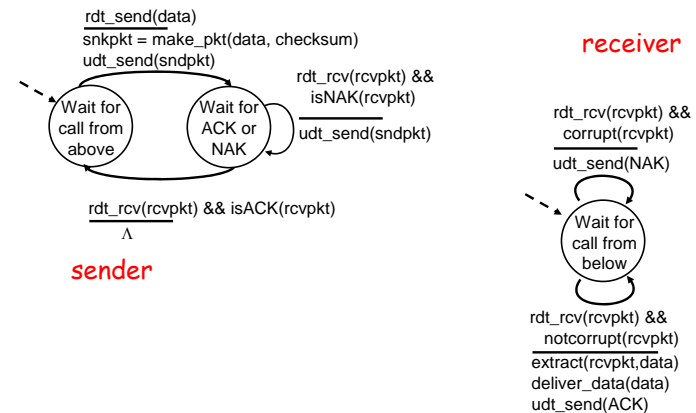
Transport Layer 3-26

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

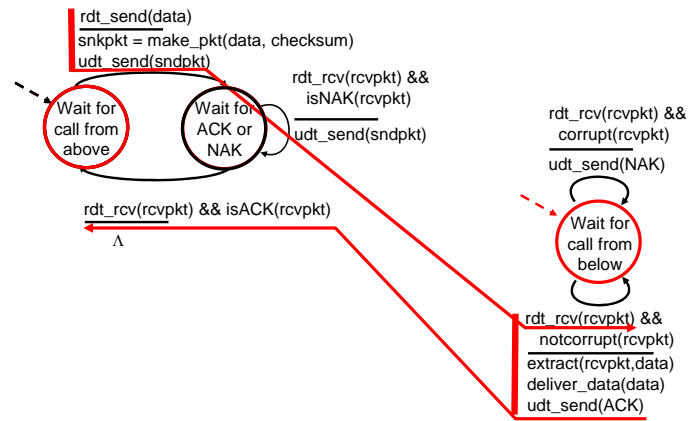
Transport Layer 3-27

rdt2.0: FSM specification



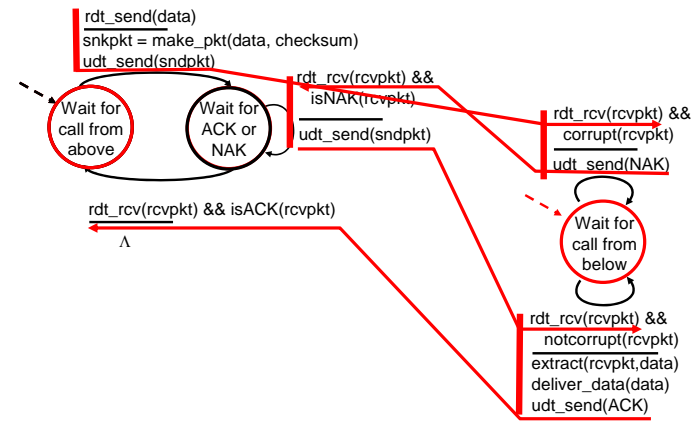
Transport Layer 3-28

rdt2.0: operation with no errors



Transport Layer 3-29

rdt2.0: error scenario



Transport Layer 3-30

rdt2.0 has a fatal flaw!

What happens if ACK/NAK corrupted?

- sender doesn't know what happened at receiver!
- can't just retransmit: possible duplicate

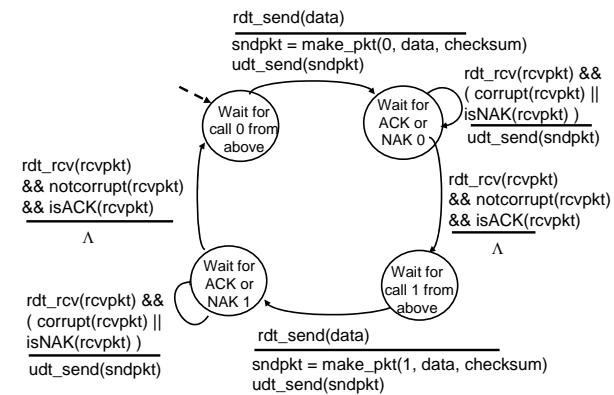
Handling duplicates:

- 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

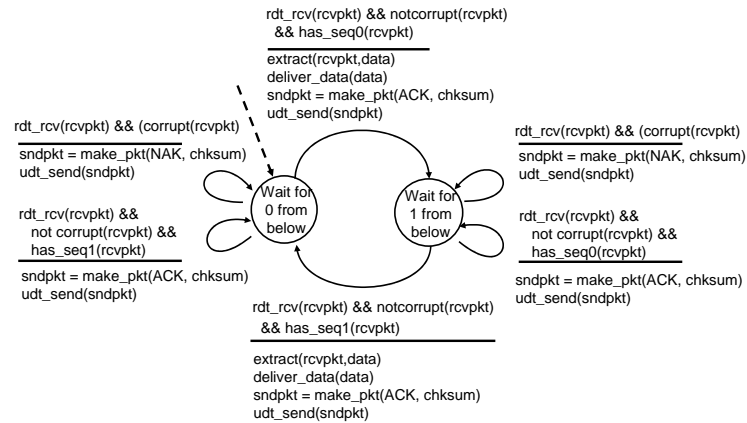
Transport Layer 3-31

rdt2.1: sender, handles garbled ACK/NAKs



Transport Layer 3-32

rdt2.1: receiver, handles garbled ACK/NAKs



Transport Layer 3-33

rdt2.1: 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

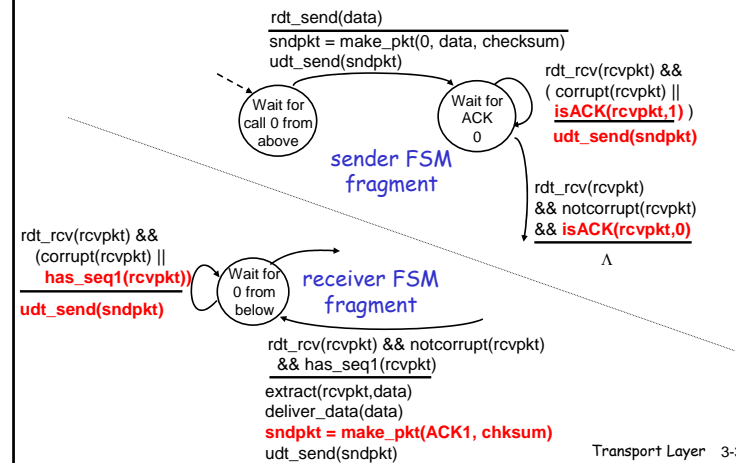
Transport Layer 3-34

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*

Transport Layer 3-35

rdt2.2: sender, receiver fragments



Transport Layer 3-36

rdt3.0: channels with errors *and* loss

New assumption:

underlying channel can also lose packets (data or ACKs)

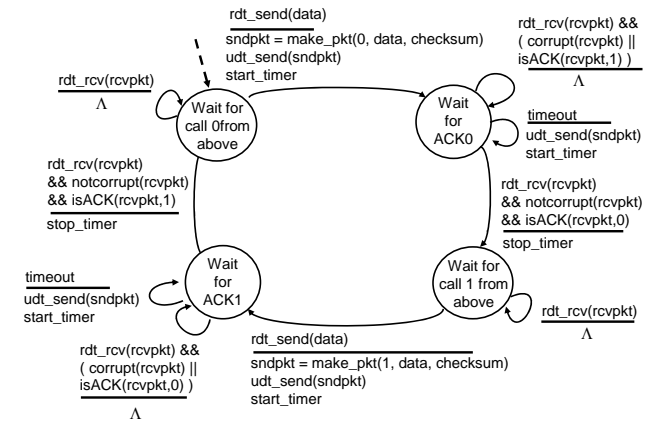
- checksum, seq. #, ACKs, retransmissions will be of help, but not enough

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

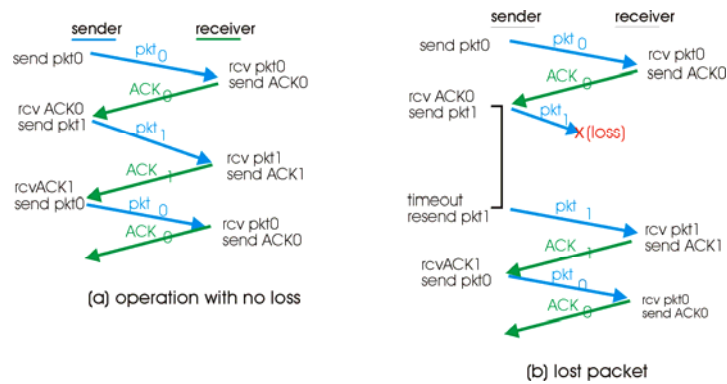
Transport Layer 3-37

rdt3.0 sender



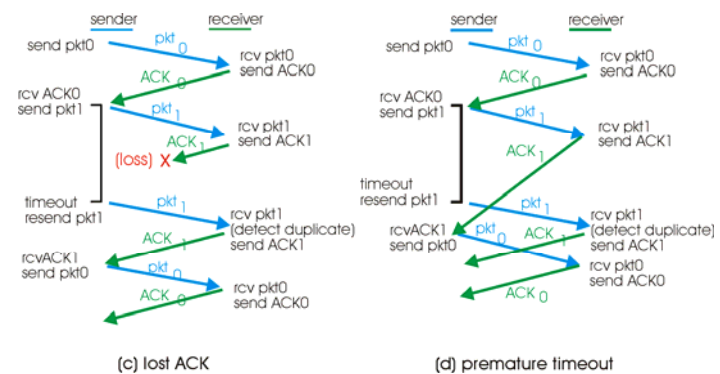
Transport Layer 3-38

rdt3.0 in action



Transport Layer 3-39

rdt3.0 in action



Transport Layer 3-40

Performance of rdt3.0

- rdt3.0 works, but performance stinks
- example: 1 Gbps link, 15 ms e-e prop. delay, 1KB packet:

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

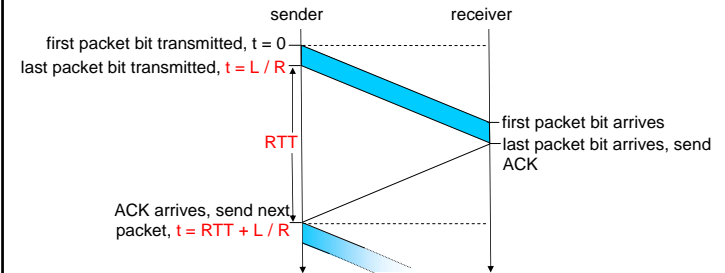
- U_{sender} : **utilization** - fraction of time sender busy sending

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

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

Transport Layer 3-41

rdt3.0: stop-and-wait operation



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

Transport Layer 3-42