

CSE 3214: Computer Network Protocols and Applications –Transport Layer (Part 2)

Dr. Peter Lian, Professor
Department of Computer Science and Engineering
York University
Email: peterlian@cse.yorku.ca
Office: 1012C Lassonde Building
Course website: [http://wiki.cse.yorku.ca/
course_archive/2012-13/W/3214](http://wiki.cse.yorku.ca/course_archive/2012-13/W/3214)

Port numbers

- ❖ Each port number is a 16-bit number ranging from 0-65535
- ❖ Well-known port numbers: 0-1,023
 - They are reserved for use by well-known application protocols, such as HTTP (80), FTP (21)
 - Given in RFC 1700 and updated at URL:
<http://www.iana.org>
- ❖ Registered port numbers: 1,024 – 49,151
 - Typically used by known but not standardized applications and are accessible by any user on an OS
- ❖ Dynamic port numbers: 49,152 – 65,535
 - They can be used for any purpose without registration

Transport Layer 3-16

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

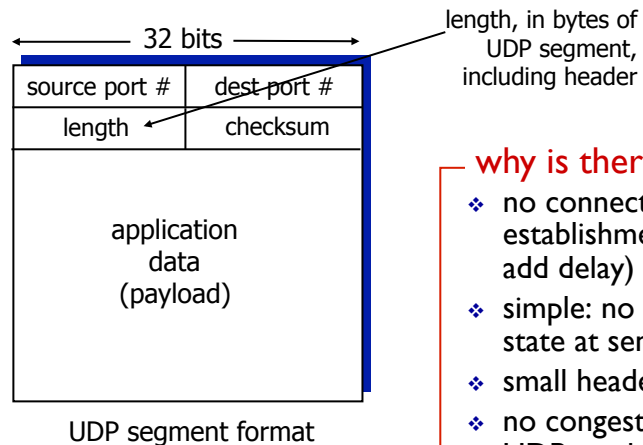
Transport Layer 3-17

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
- ❖ UDP use:
 - streaming multimedia apps (loss tolerant, rate sensitive)
 - DNS
 - SNMP
- ❖ reliable transfer over UDP:
 - add reliability at application layer
 - application-specific error recovery!

Transport Layer 3-18

UDP: segment header



why is there a UDP?

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

Transport Layer 3-19

UDP checksum

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

sender:

- ❖ treat segment contents, including header fields, as sequence of 16-bit integers
- ❖ checksum: addition (one'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 3-20

Internet checksum: example

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
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

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

Transport Layer 3-21

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

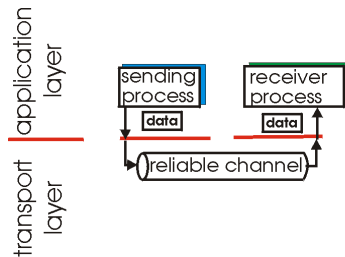
3.6 principles of congestion control

3.7 TCP congestion control

Transport Layer 3-22

Principles of reliable data transfer

- ❖ important in application, transport, link layers
 - top-10 list of important networking topics!



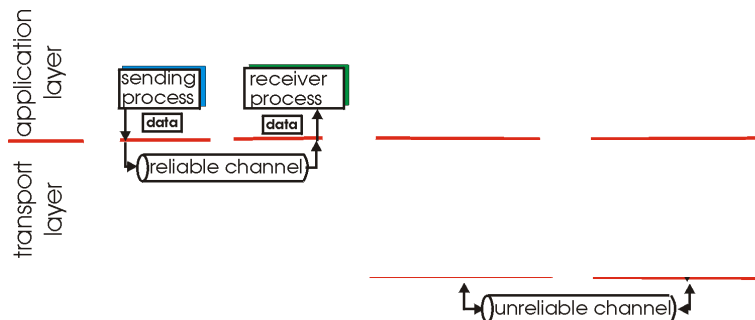
(a) provided service

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

Transport Layer 3-23

Principles of reliable data transfer

- ❖ important in application, transport, link layers
 - top-10 list of important networking topics!



(a) provided service

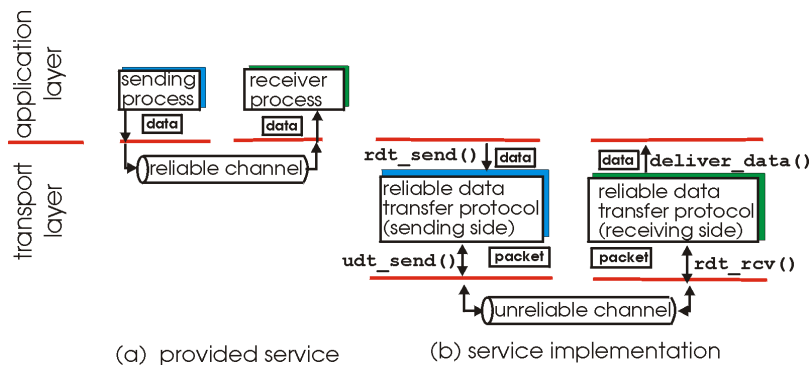
(b) service implementation

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

Transport Layer 3-24

Principles of reliable data transfer

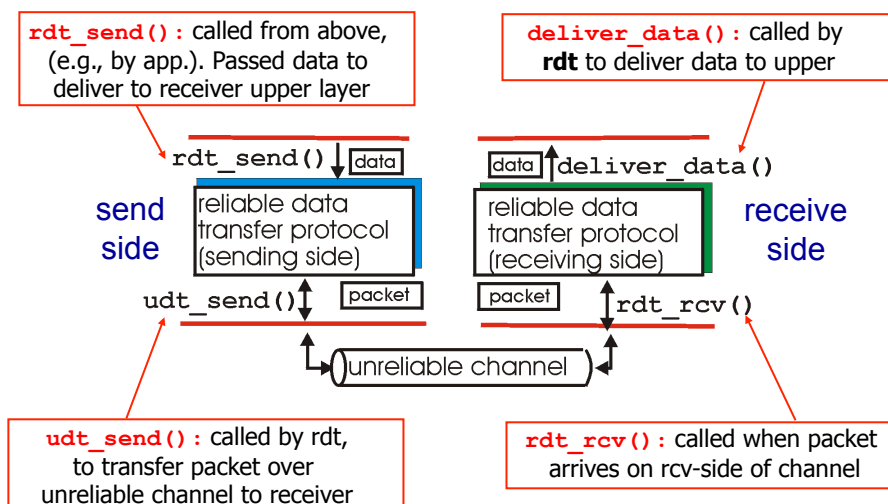
- ❖ important in application, transport, link layers
 - top-10 list of important networking topics!



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

Transport Layer 3-25

Reliable data transfer: getting started

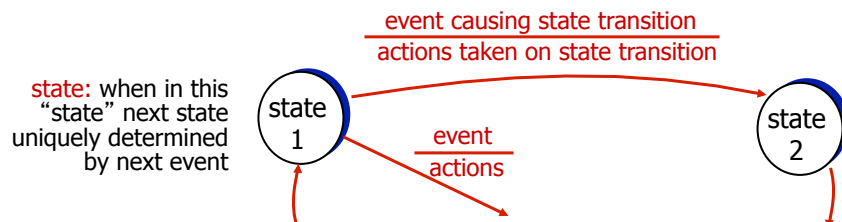


Transport Layer 3-26

Reliable data transfer: getting started

we'll:

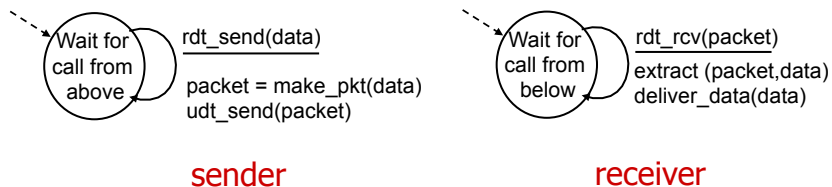
- ❖ incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- ❖ consider only unidirectional data transfer
 - but control info will flow on both directions!
- ❖ use finite state machines (FSM) to specify sender, receiver



Transport Layer 3-27

rdt I.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 reads data from underlying channel



Transport Layer 3-28

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:

*How do humans recover from “errors”
during conversation?*

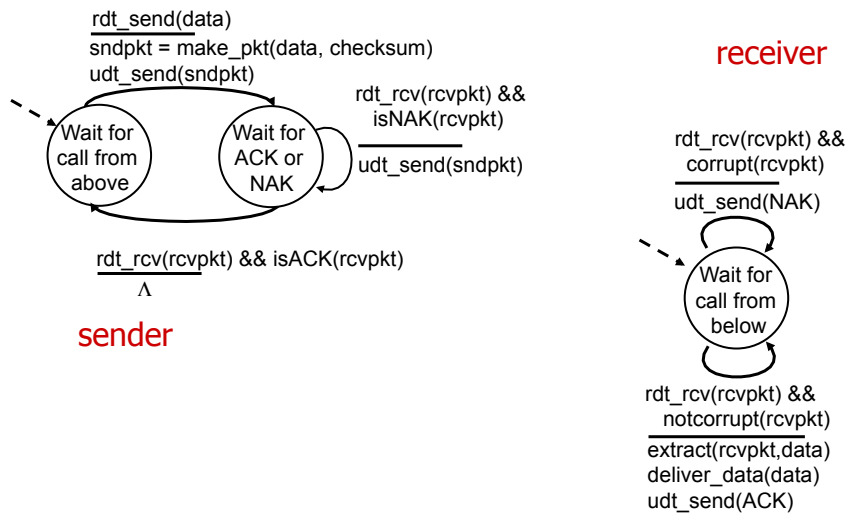
Transport Layer 3-29

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
 - feedback: control msgs (ACK,NAK) from receiver to sender

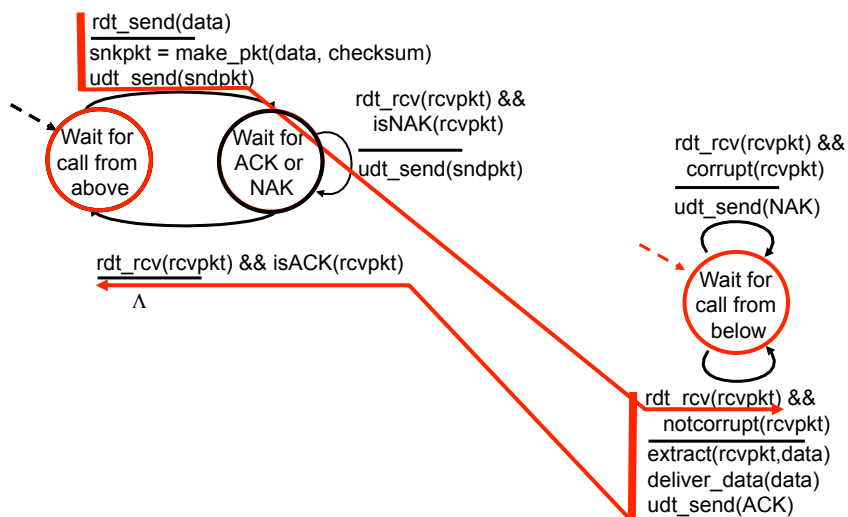
Transport Layer 3-30

rdt2.0: FSM specification



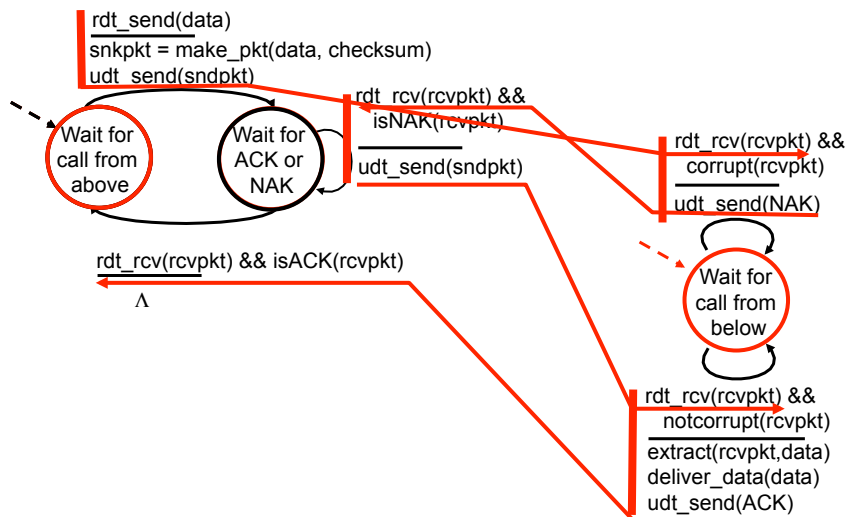
Transport Layer 3-31

rdt2.0: operation with no errors



Transport Layer 3-32

rdt2.0: error scenario



Transport Layer 3-33

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:

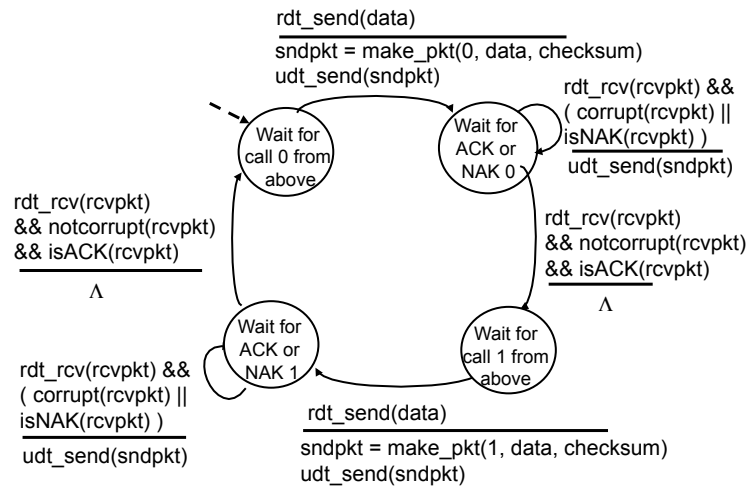
- ❖ sender retransmits current pkt if ACK/NAK corrupted
- ❖ sender adds *sequence number* to each pkt
- ❖ receiver discards (doesn't deliver up) duplicate pkt

stop and wait

sender sends one packet,
then waits for receiver
response

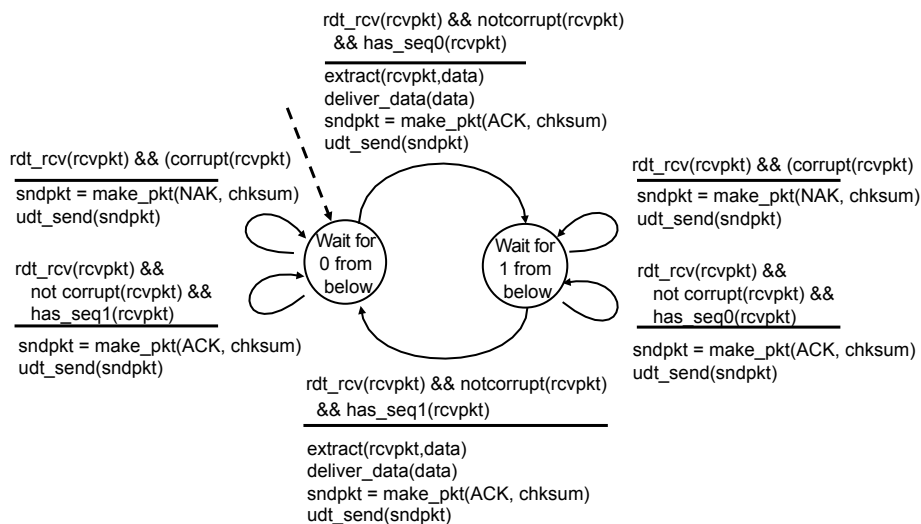
Transport Layer 3-34

rdt2.1: sender, handles garbled ACK/NAKs



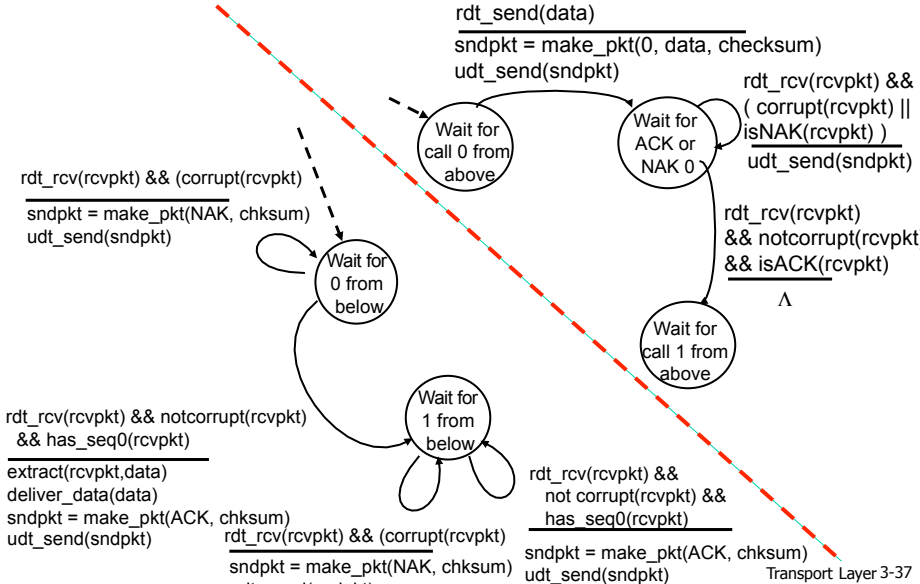
Transport Layer 3-35

rdt2.1: receiver, handles garbled ACK/NAKs



Transport Layer 3-36

rdt2.l: sender, handles garbled ACK/NAKs



rdt2.1: discussion

sender:

- ❖ seq # added to pkt
- ❖ two seq. #'s (0,1) will suffice.
- ❖ must check if received ACK/NAK corrupted
- ❖ twice as many states
 - state must “remember” whether “expected” pkt should have seq # of 0 or 1

receiver:

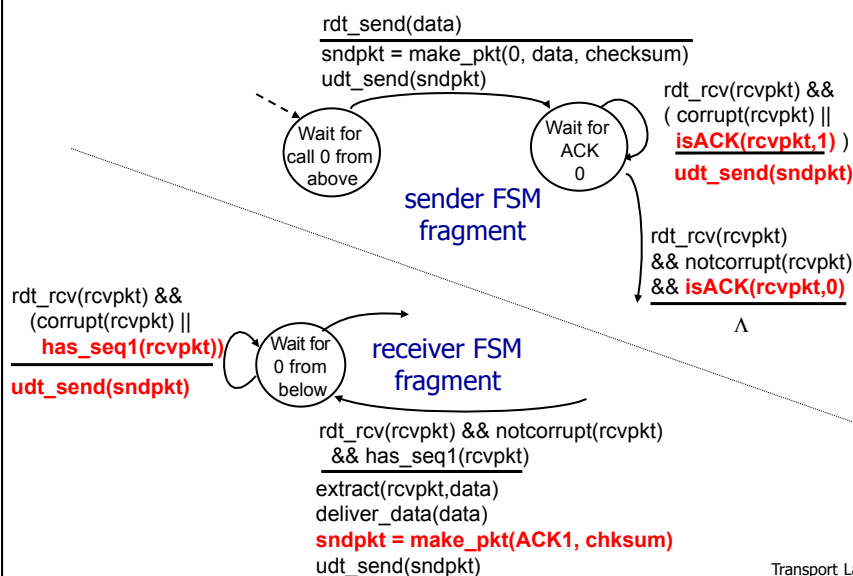
- ❖ must check if received packet is duplicate
 - state indicates whether 0 or 1 is expected packet seq #

rdt2.2: a NAK-free protocol

- ❖ same functionality as rdt2.1, using ACKs only
- ❖ instead of NAK, receiver sends ACK for last packet received OK, i.e. using ACK0 as NAK for a packet with sequence # 1, or using ACK1 as NAK for a packet with sequence # 0
 - receiver must *explicitly* include seq # of packet being ACKed
- ❖ duplicate ACK at sender results in same action as NAK: *retransmit current packet*

Transport Layer 3-39

rdt2.2: sender, receiver fragments



Transport Layer 3-40