

# Announcement



- Project I is delayed for another week
- Senior oral presentation confirmation

# Last Time



- Non-persistent HTTP vs. Persistent HTTP
- Cookies
- Web caches, conditional GET
- FTP: control connection and data connection
- Email system: user agents, mail servers, and SMTP
- DNS: root servers, TLD servers, authoritative servers, and local servers
- P2P: file distribution



# Interview Question (MediaTek)



- What happens when a user clicks on a hyperlink
  - e.g., [www.cs.sfsu.edu](http://www.cs.sfsu.edu)
  - We do not consider web caches here
- DNS
  - The browser extracts the hostname from the URL and passes it to the client side of DNS
  - The DNS client contacts local DNS server, which may need to contact root servers, TLD servers, and authoritative servers
- HTTP
  - TCP Three-way handshake
  - HTTP request
  - HTTP response

# Chapter 3: Transport Layer

## our goals:

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

# 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

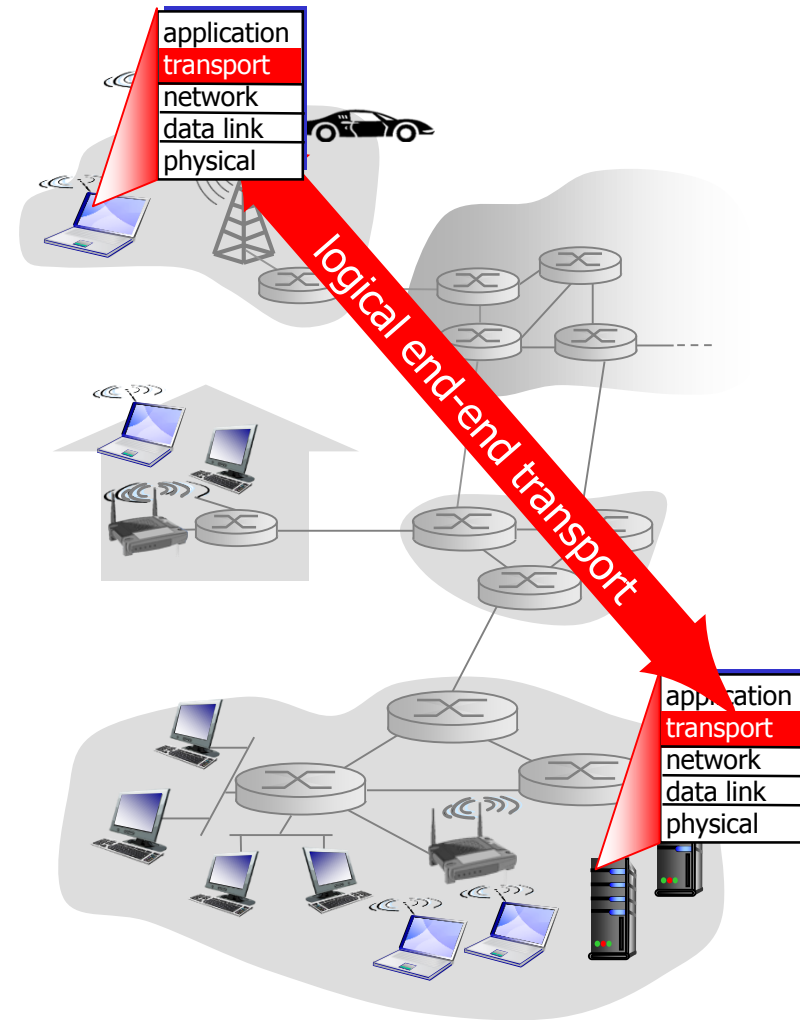
- segment structure
- reliable data transfer
- flow control
- connection management

## 3.6 principles of congestion control

## 3.7 TCP congestion control

# Transport 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 vs. network layer

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

# Internet transport-layer protocols

- ❖ unreliable, unordered delivery: UDP
  - multiplexing and demultiplexing
  - error-correction code
- ❖ reliable, in-order delivery (TCP)
  - multiplexing and demultiplexing
  - error-correction code
  - reliable data transfer
  - congestion control
  - flow control
- ❖ services not available:
  - delay guarantees
  - bandwidth guarantees



# 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

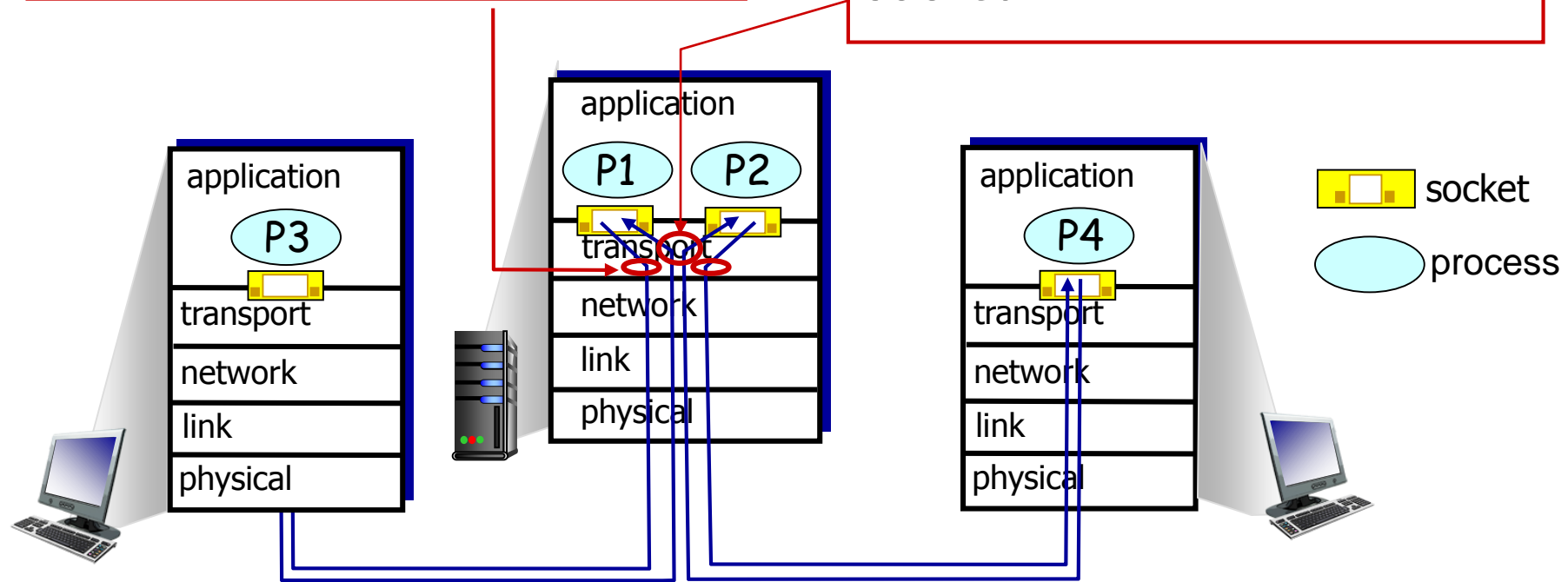
3.6 principles of congestion control

3.7 TCP congestion control

# Multiplexing/demultiplexing

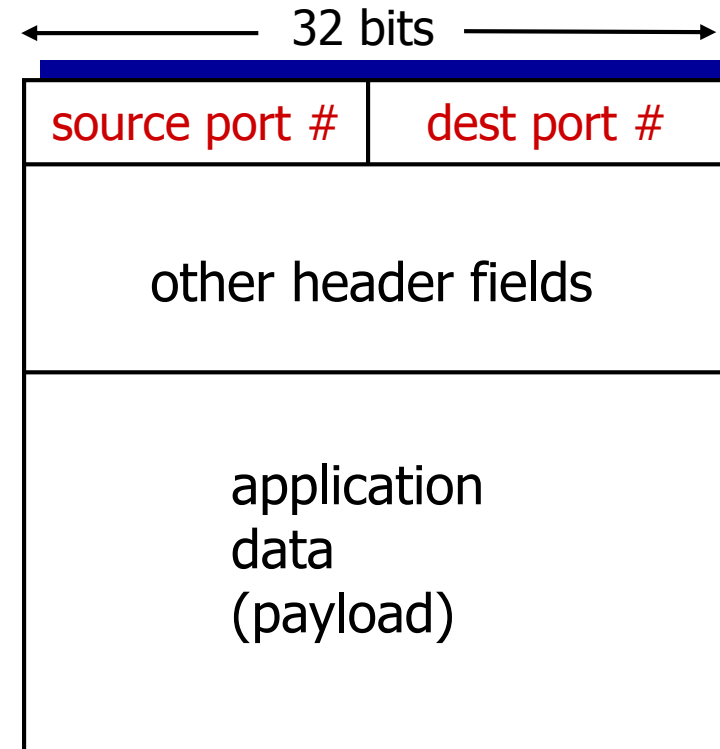
*multiplexing at sender:*  
handle data from multiple sockets, add transport header (later used for demultiplexing)

*demultiplexing at receiver:*  
use header info to deliver received segments to correct socket



# How demultiplexing works

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



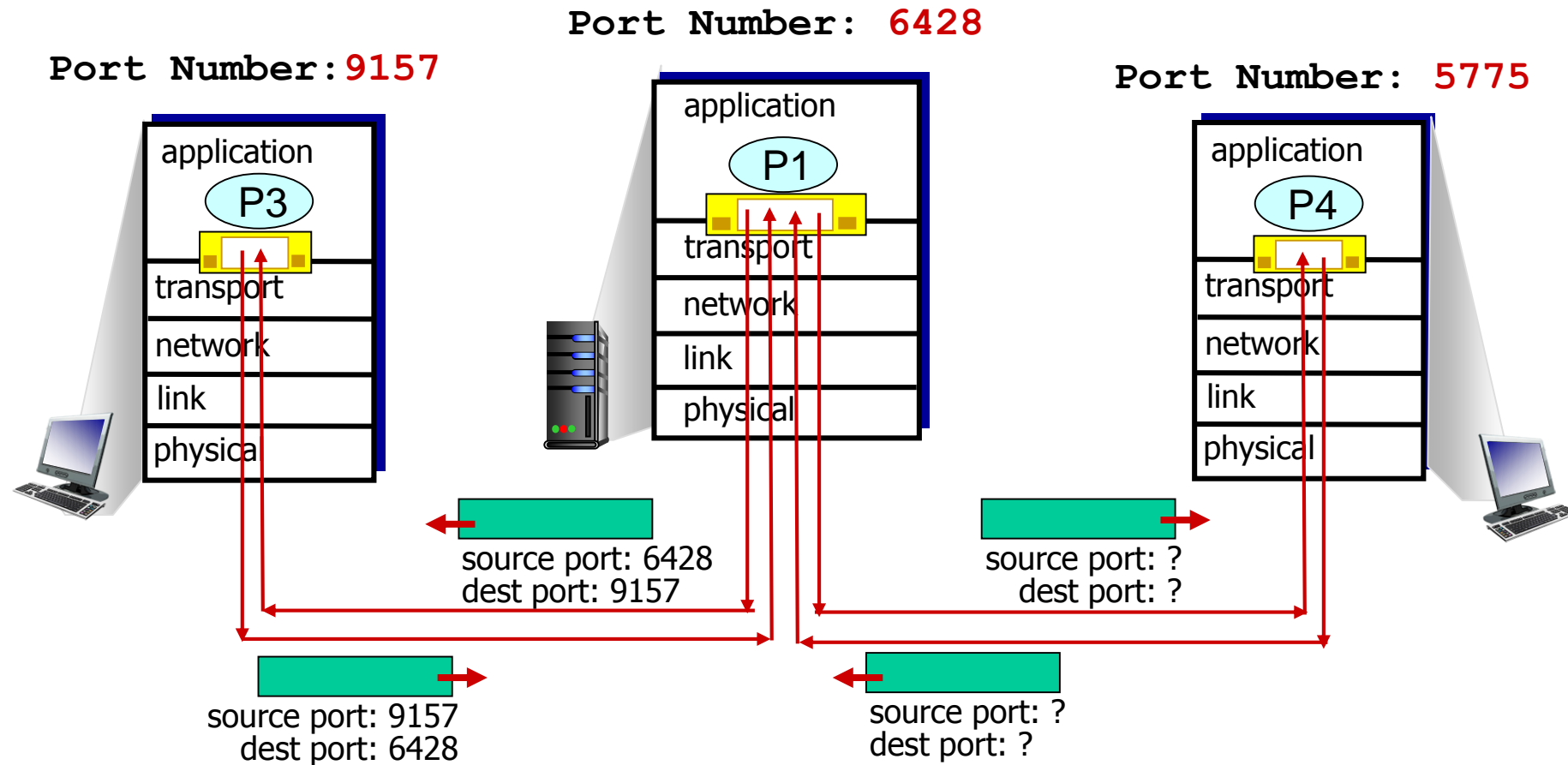
TCP/UDP segment format

# Connectionless demultiplexing

- ❖ when creating datagram to send into UDP socket, must specify
  - destination IP address
  - destination port #
- ❖ when host receives UDP segment:
  - checks destination port # in segment
  - directs UDP segment to socket with that port #

→ IP datagrams with *same dest. port #*, but different source IP addresses and/or source port numbers will be directed to *same socket* at dest

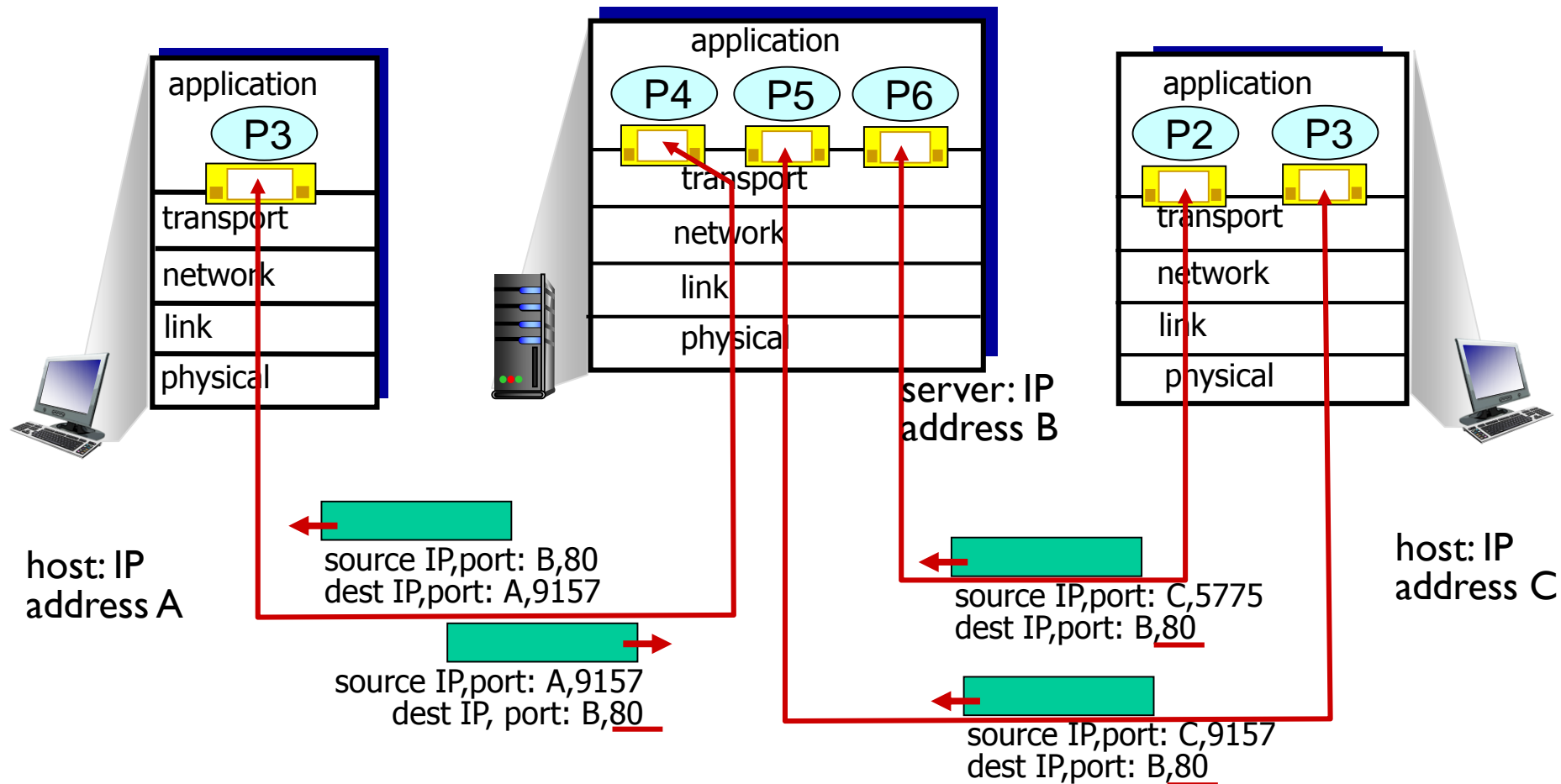
# Connectionless demux: example



# Connection-oriented demux

- ❖ TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- ❖ demux: receiver 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

# Connection-oriented demux: example



three segments, all destined to IP address: B,  
dest port: 80 are demultiplexed to *different* sockets

# 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

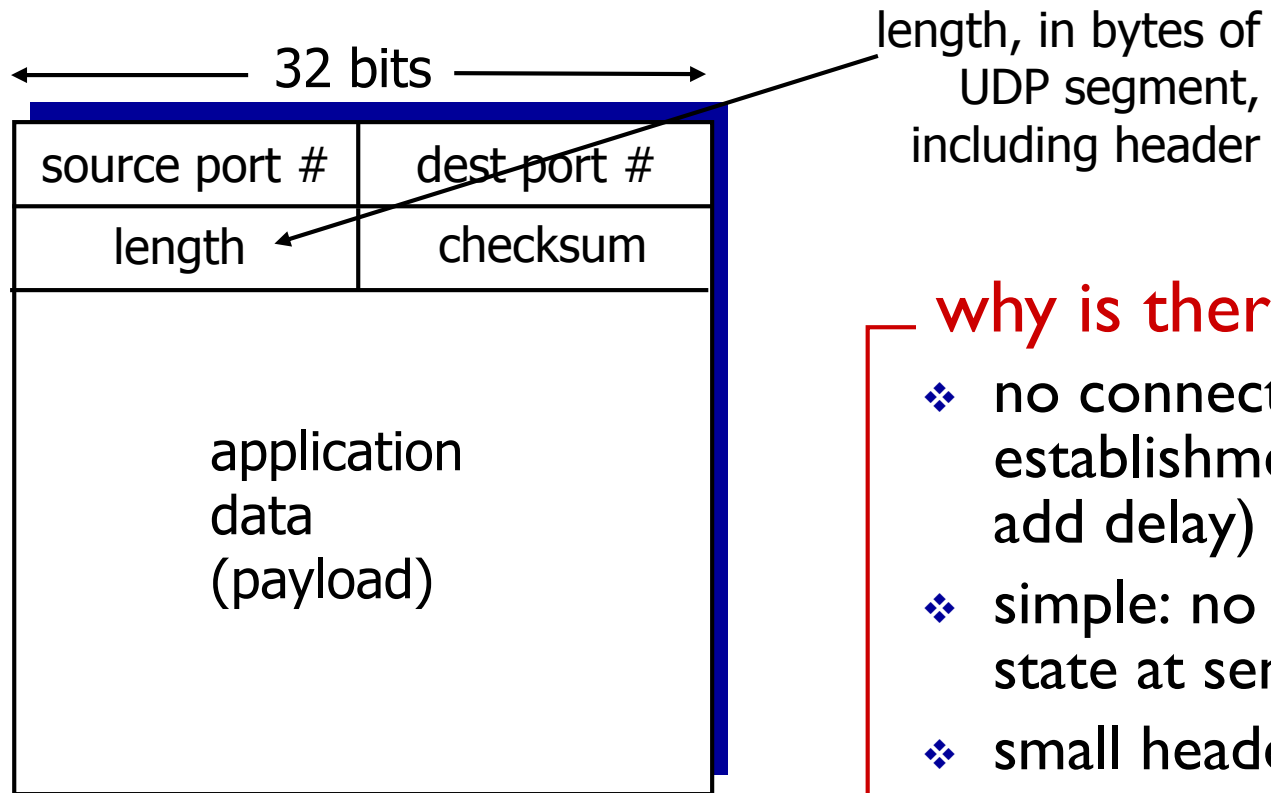
3.7 TCP congestion control



# 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
- ❖ reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

# UDP: segment header



UDP segment format

## — 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

# 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*  
....

# 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
<hr/>																
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1
<hr/>																
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

# 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

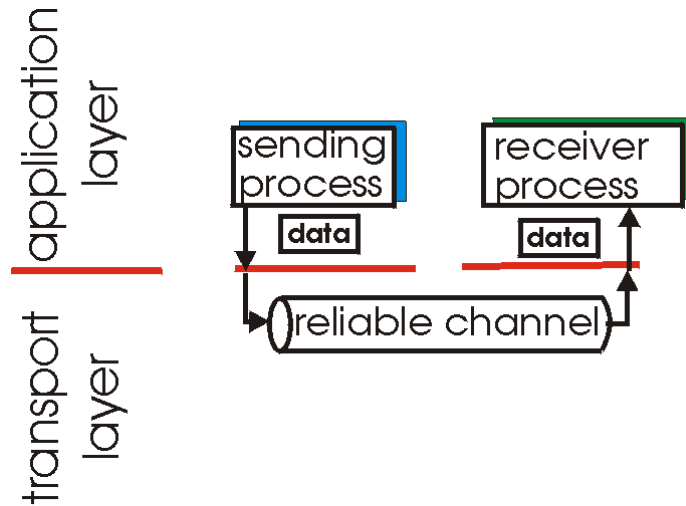
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

# Principles of reliable data transfer

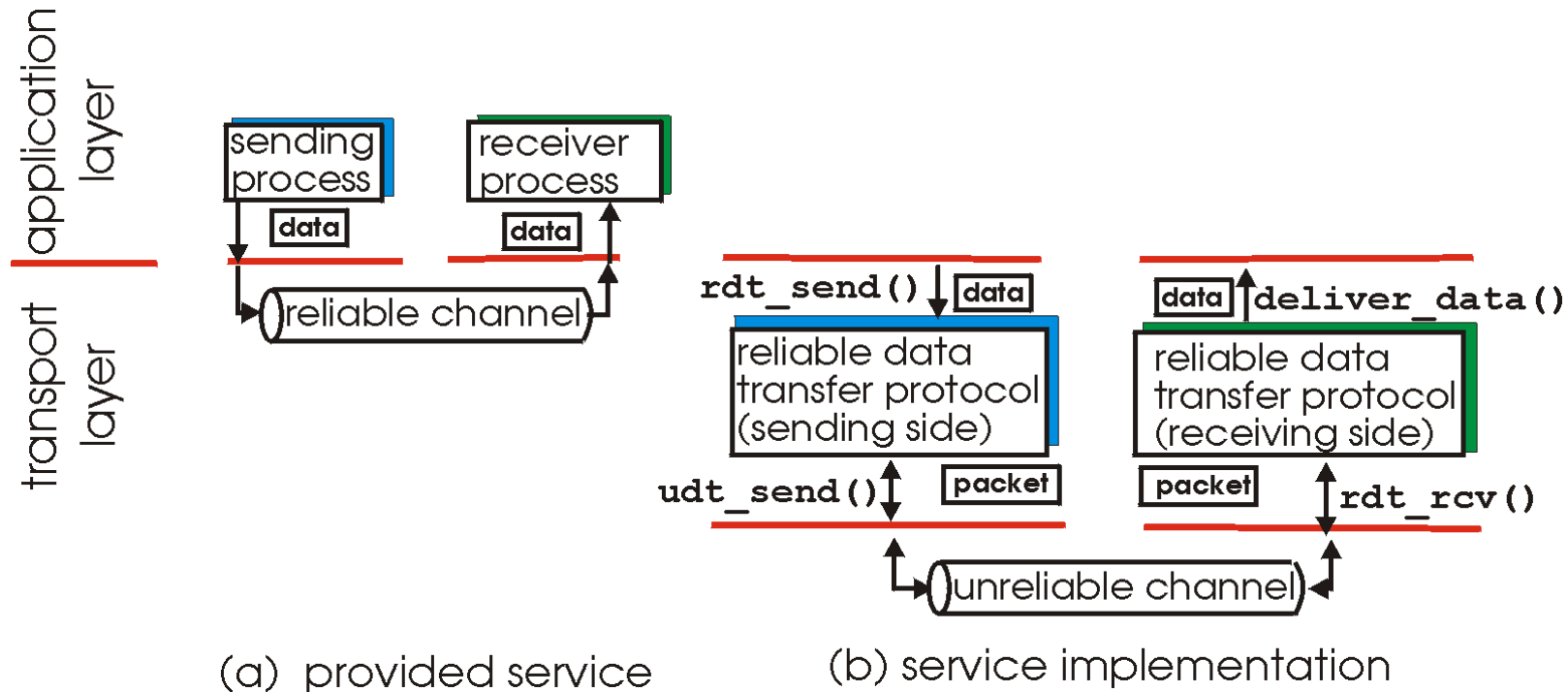
- ❖ important in application, transport, link layers
  - top-10 list of important networking topics!



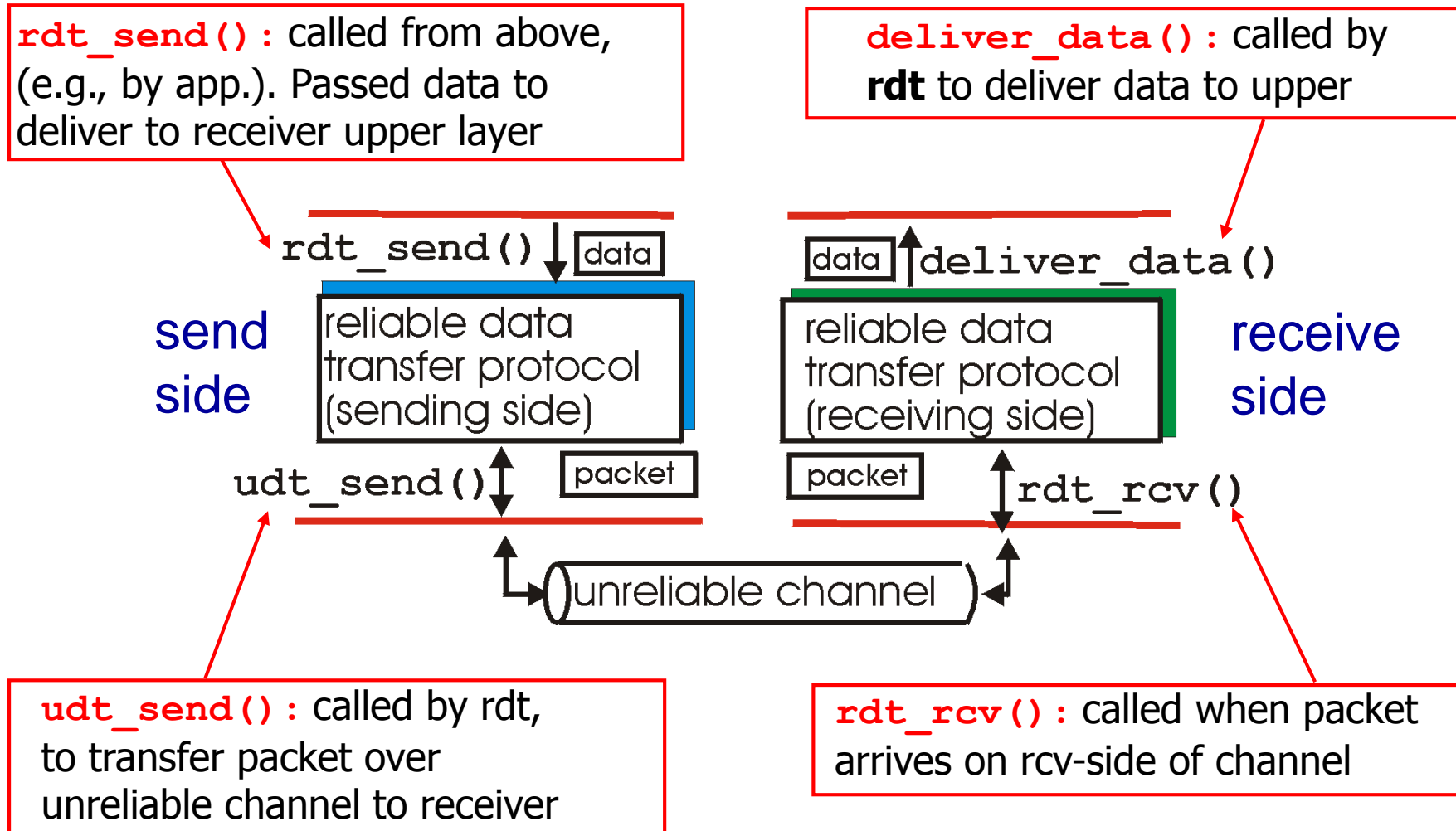
(a) provided service

# Principles of reliable data transfer

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



# Reliable data transfer: getting started

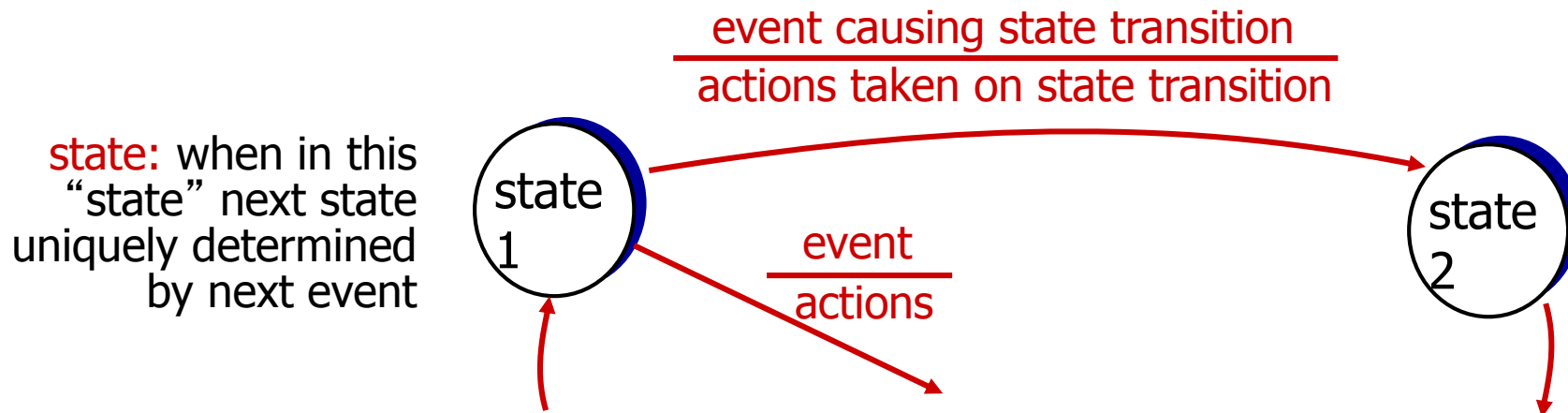




# Reliable data transfer: getting started

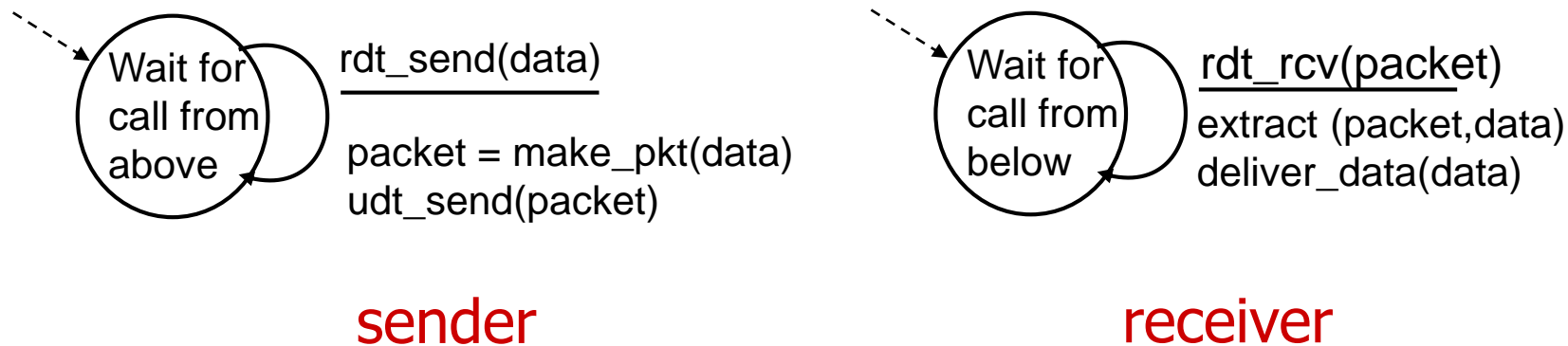
we'll:

- ❖ 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



# 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 reads data from underlying channel



## rdt2.0: channel with bit errors

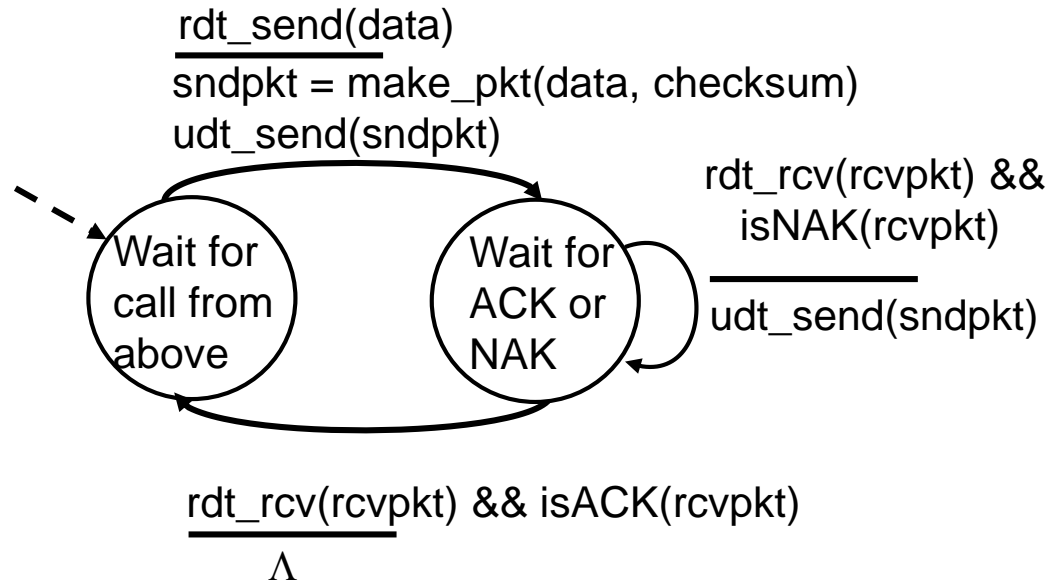
- ❖ underlying channel may flip bits in packet
  - checksum to detect bit errors
- ❖ *the question: how to recover from errors:*

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

# 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
  - retransmission

# rdt2.0: FSM specification

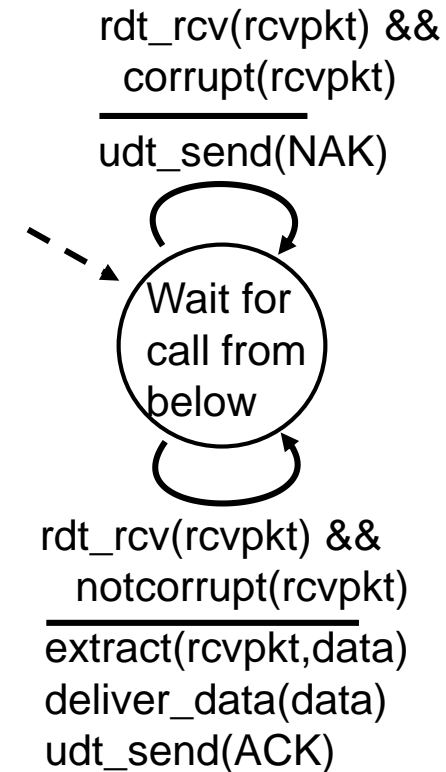


sender

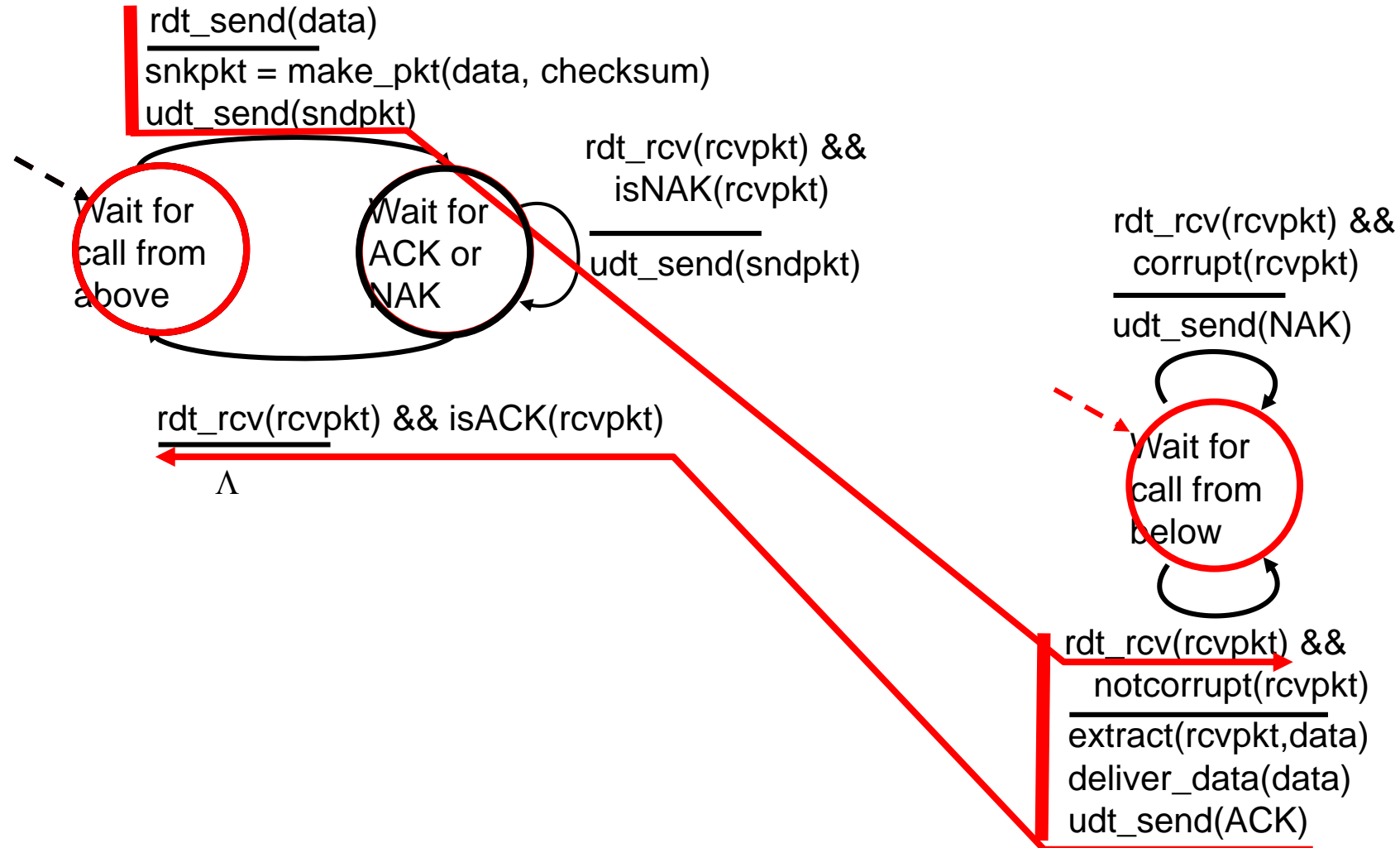
**stop and wait**

sender sends one packet,  
then waits for receiver  
response

receiver



# rdt2.0: operation with no errors



# rdt2.0: error scenario

