

# CMPE 150/L : Introduction to Computer Networks

Chen Qian

Computer Engineering  
UCSC Baskin Engineering

Lecture 7

# Homework questions

- ❑ Available on course website
- ❑ **Please work on them**, but do not submit your answers. The answers will be posted later.

# 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

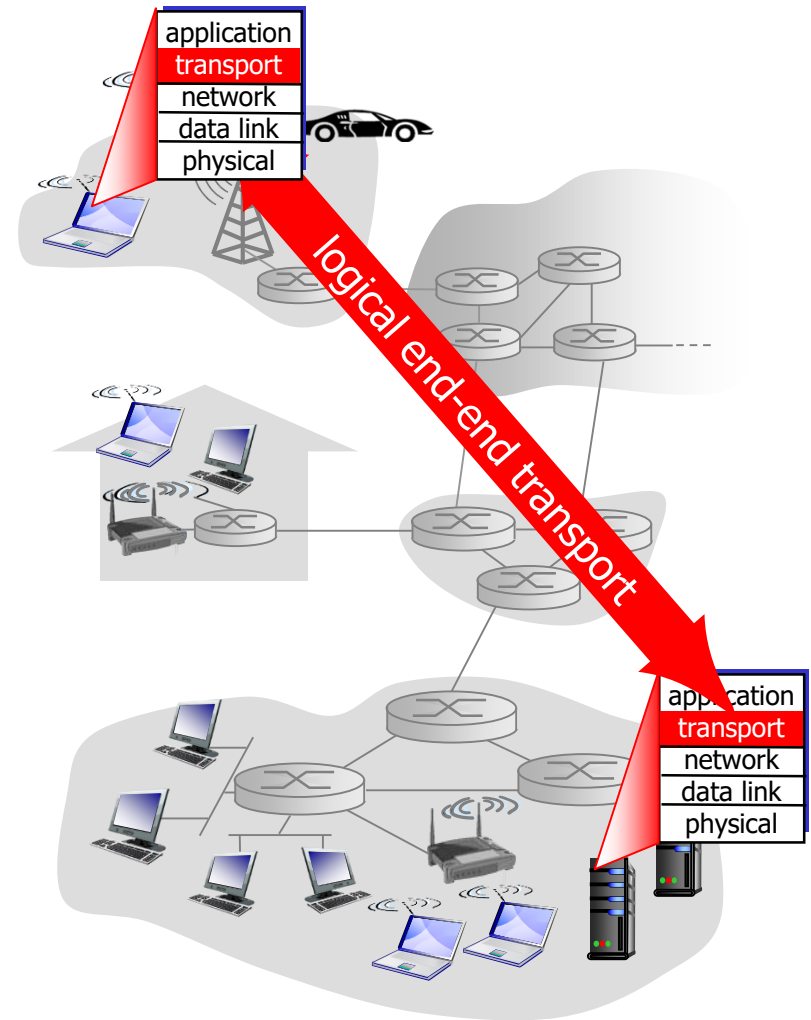
- segment structure
- reliable data transfer
- flow control
- connection management

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

## *household analogy:*

3 kids in Ann 's house sending letters to 3 kids in Bill 's house:

- ❖ hosts = houses
- ❖ processes = kids
- ❖ app messages = letters in envelopes
- ❖ transport protocol = Ann and Bill who demux to in-house siblings
- ❖ network-layer protocol = postal service

❖ Ann's home

- Ann
- Alice
- Amy

❖ Home address:

- 123, Santa Cruz

❖ Bill's home

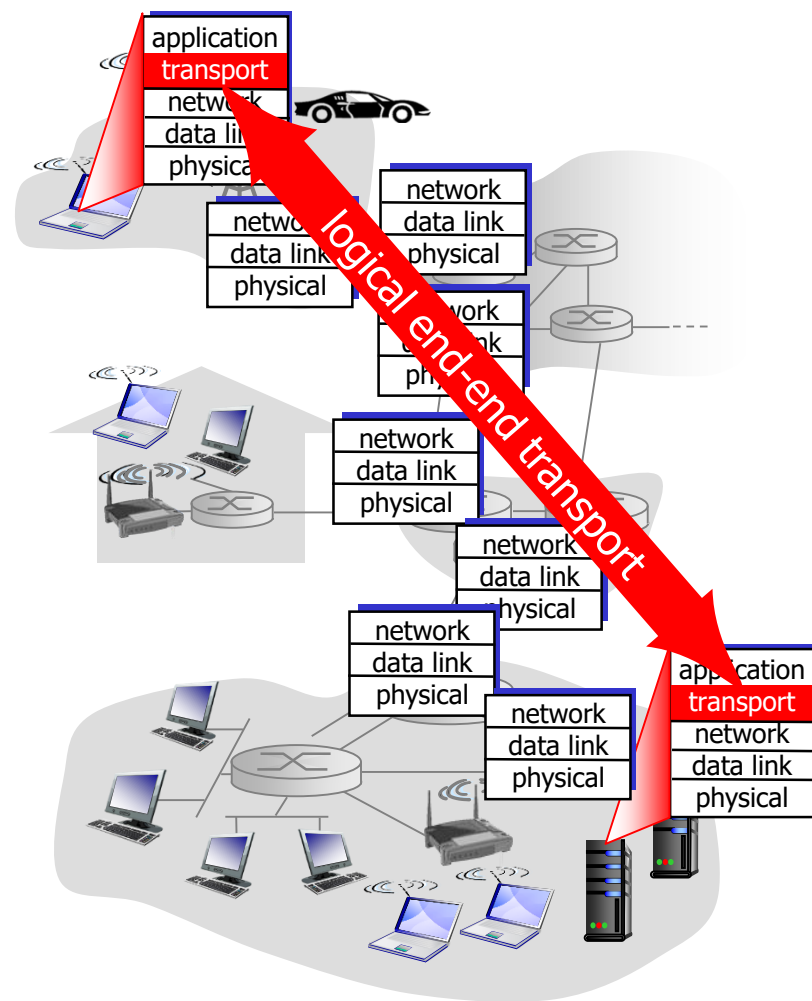
- Bill
- Bob
- Brent

❖ Home address :

- 456, Scotts Valley

# 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



# 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

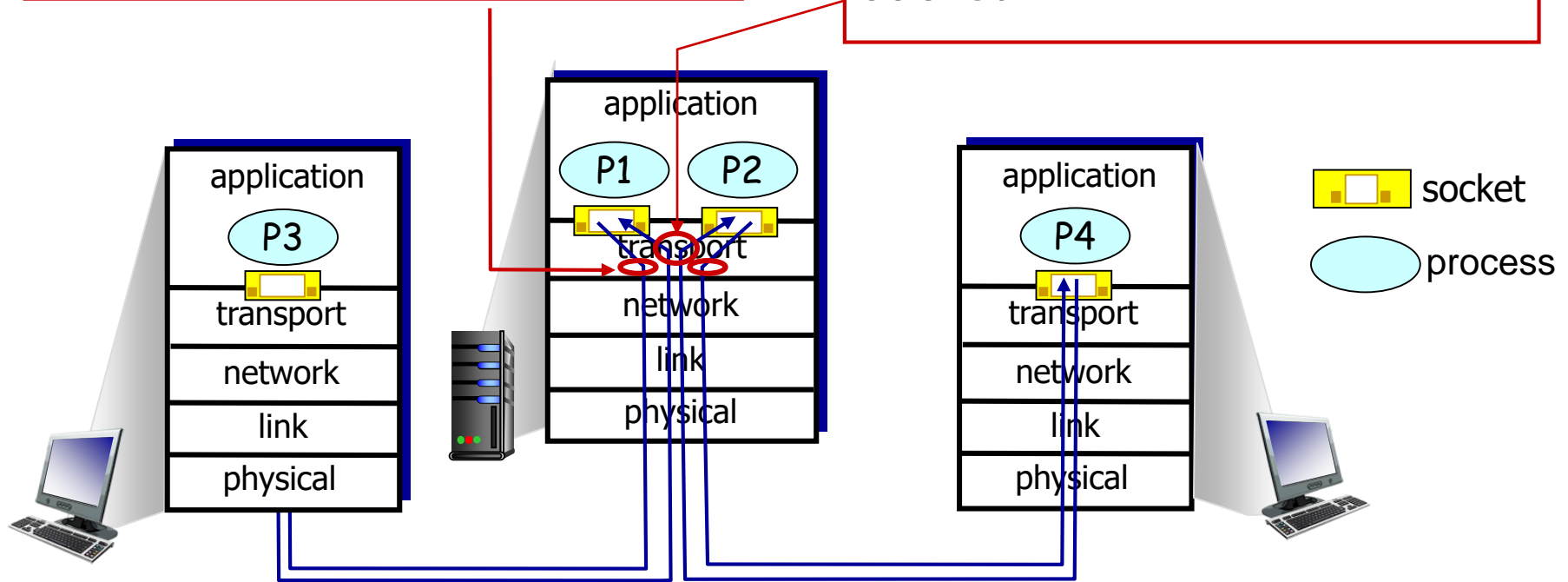
3.7 TCP congestion control



# Multiplexing/demultiplexing

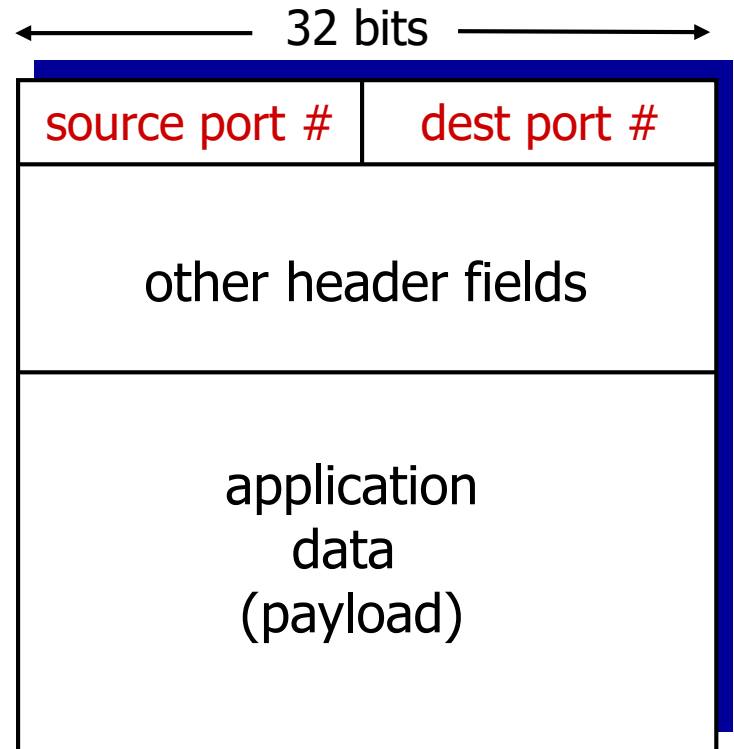
*multiplexing at sender:*  
handle data from multiple sockets, add transport header (later used for demultiplexing)

*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

- ❖ *recall*: created socket has host-local port #:

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

- ❖ *recall*: 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 #



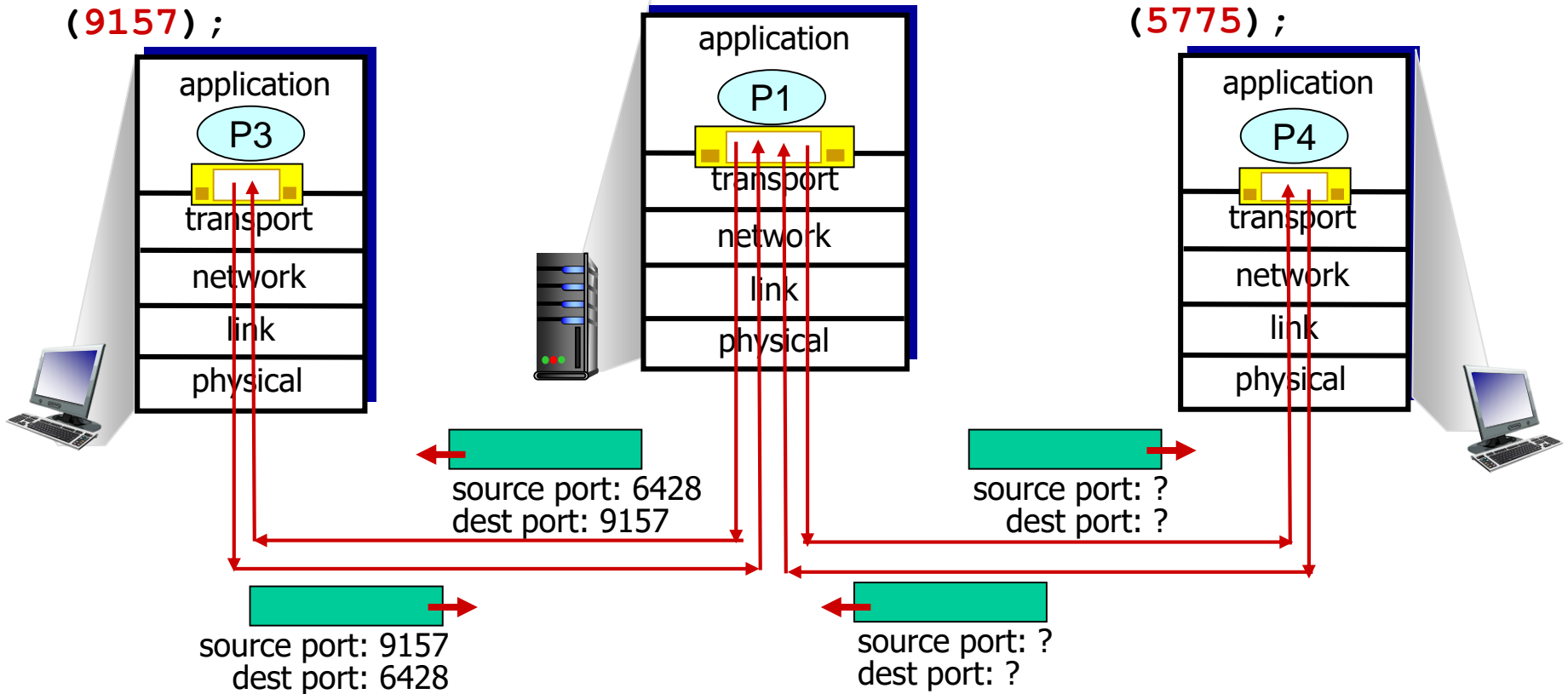
IP datagrams with *same dest. port #*, but different source IP addresses and/or source port numbers will be directed to *same socket* at dest

# Connectionless demux: example

```
DatagramSocket  
mySocket2 = new  
DatagramSocket  
(9157);
```

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

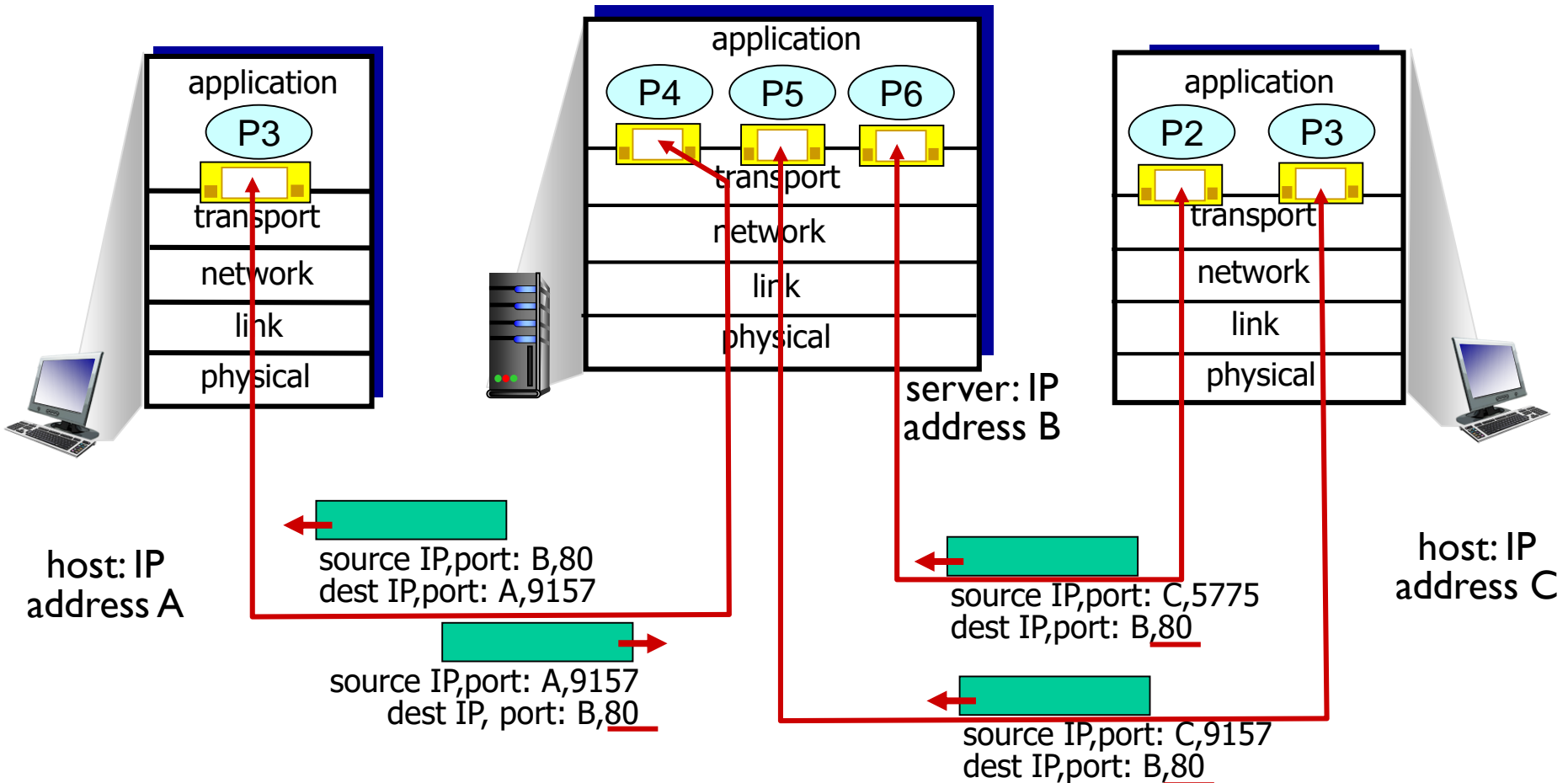
```
DatagramSocket  
mySocket1 = new  
DatagramSocket  
(5775);
```



# Connection-oriented demux

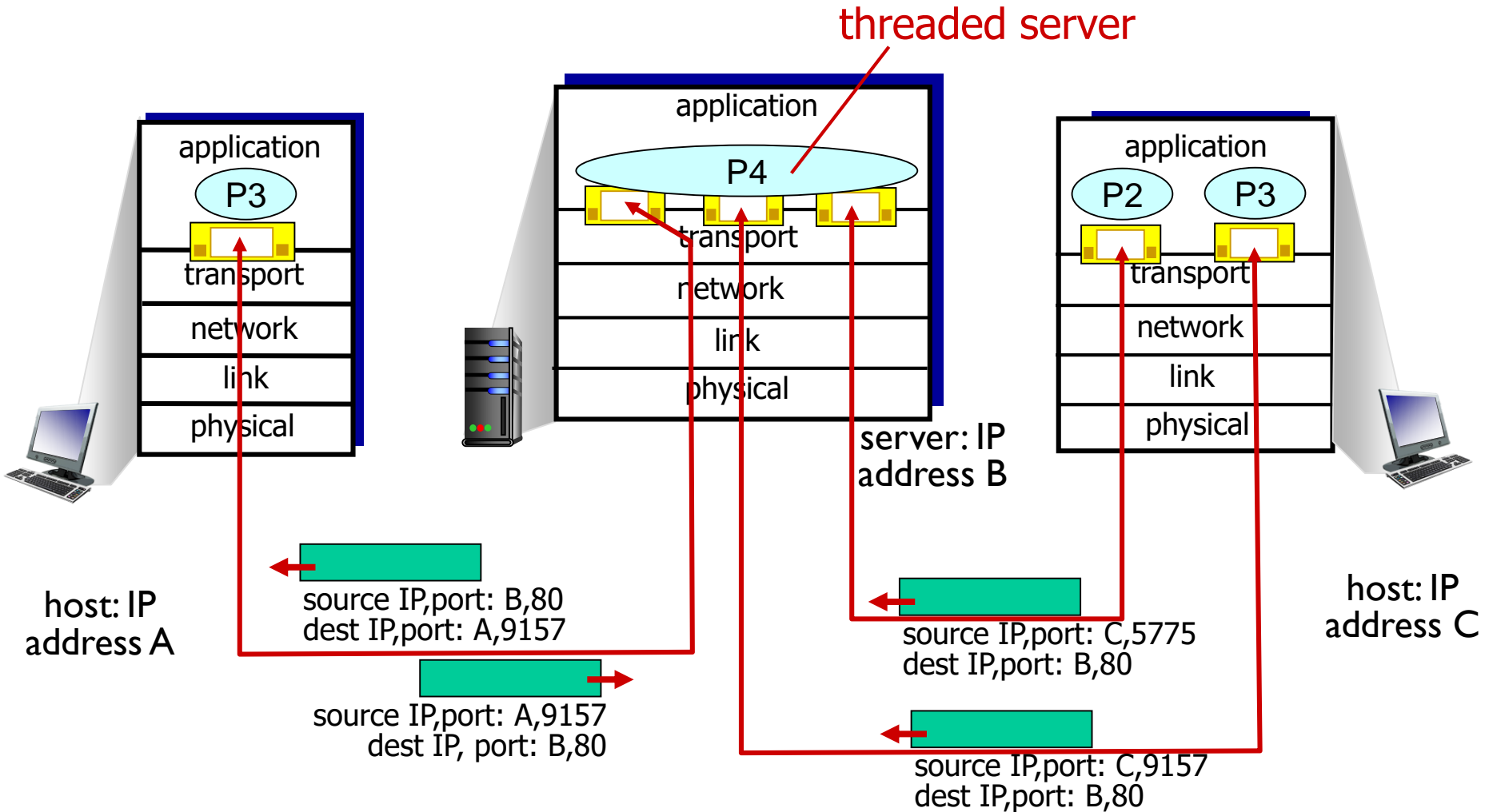
- ❖ TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- ❖ demux: receiver uses all four values to direct segment to appropriate socket
- ❖ server host may support many simultaneous TCP sockets:
  - each socket identified by its own 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

# Connection-oriented demux: example

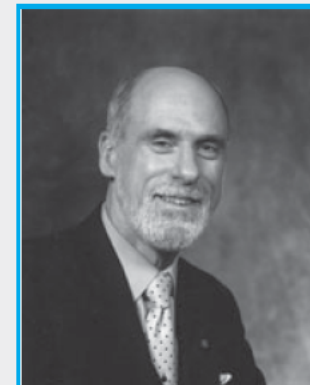


# Interview with Vinton Cerf

- ❖ One of the inventors of the TCP and IP protocols.
- ❖ <https://www.youtube.com/watch?v=ILiQnw0b-YQ> 0:00-6:00

## Vinton G. Cerf

Vinton G. Cerf is Vice President and Chief Internet Evangelist for Google. He served for over 16 years at MCI in various positions, ending up his tenure there as Senior Vice President for Technology Strategy. He is widely known as the co-designer of the TCP/IP protocols and the architecture of the Internet. During his time from 1976 to 1982 at the US Department of Defense Advanced Research Projects Agency (DARPA), he played a key role leading the development of Internet protocols, including TCP and IP.





# 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

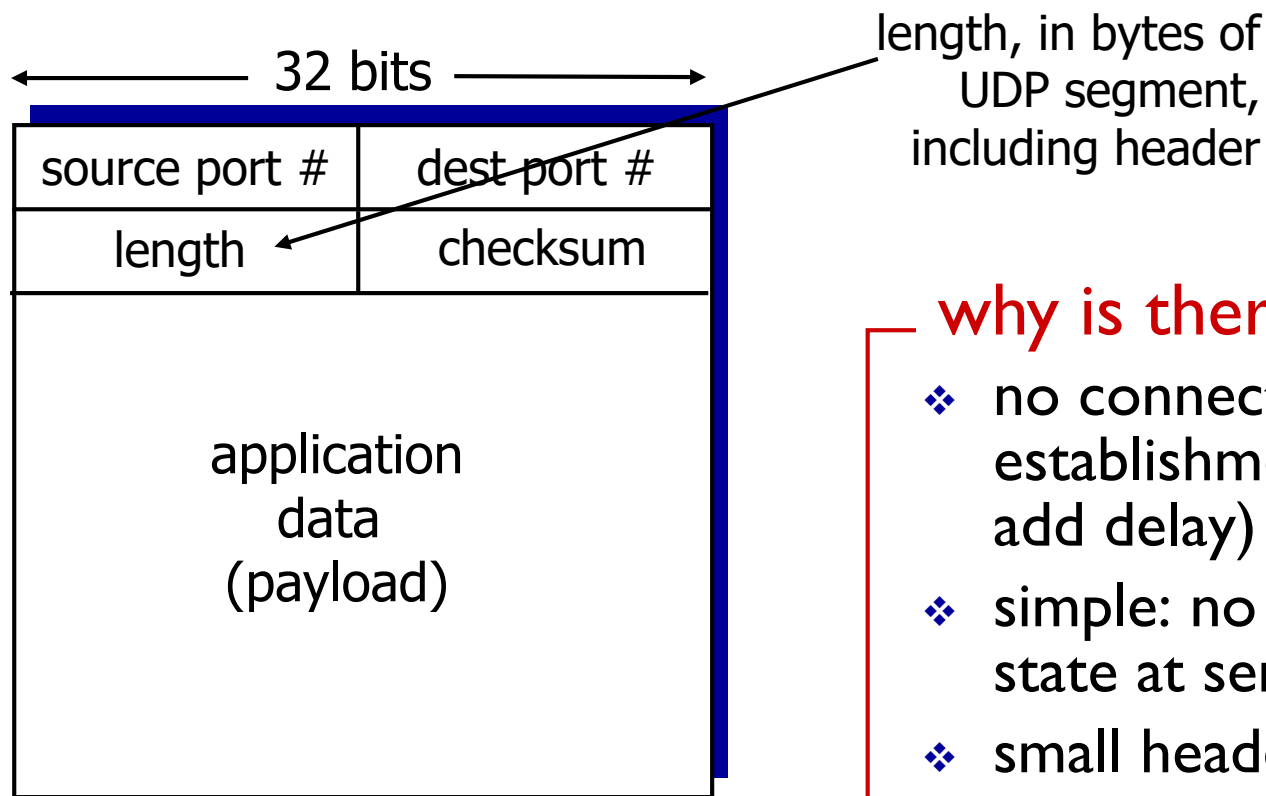
3.6 principles of congestion control

3.7 TCP congestion control

# UDP: User Datagram Protocol [RFC 768]

- ❖ “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
  - Simple Network Management Protocol (SNMP)

# 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	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	0	1	1

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

At the receiver, adding all words and checksum, the result should be all ones. If there is a 0, some error must happen.

- ❖ The previous checksum algorithm may not be able to detect even-number bit errors.
  - Odd-number bit errors are guaranteed to detect

# 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

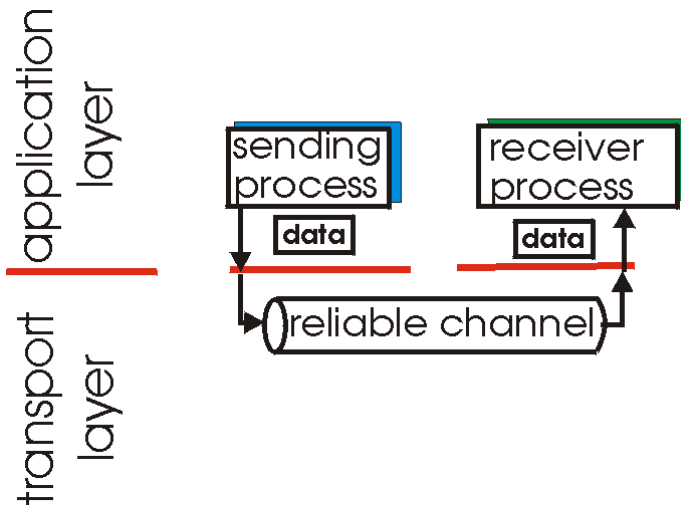
- segment structure
- reliable data transfer
- flow control
- connection management

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!



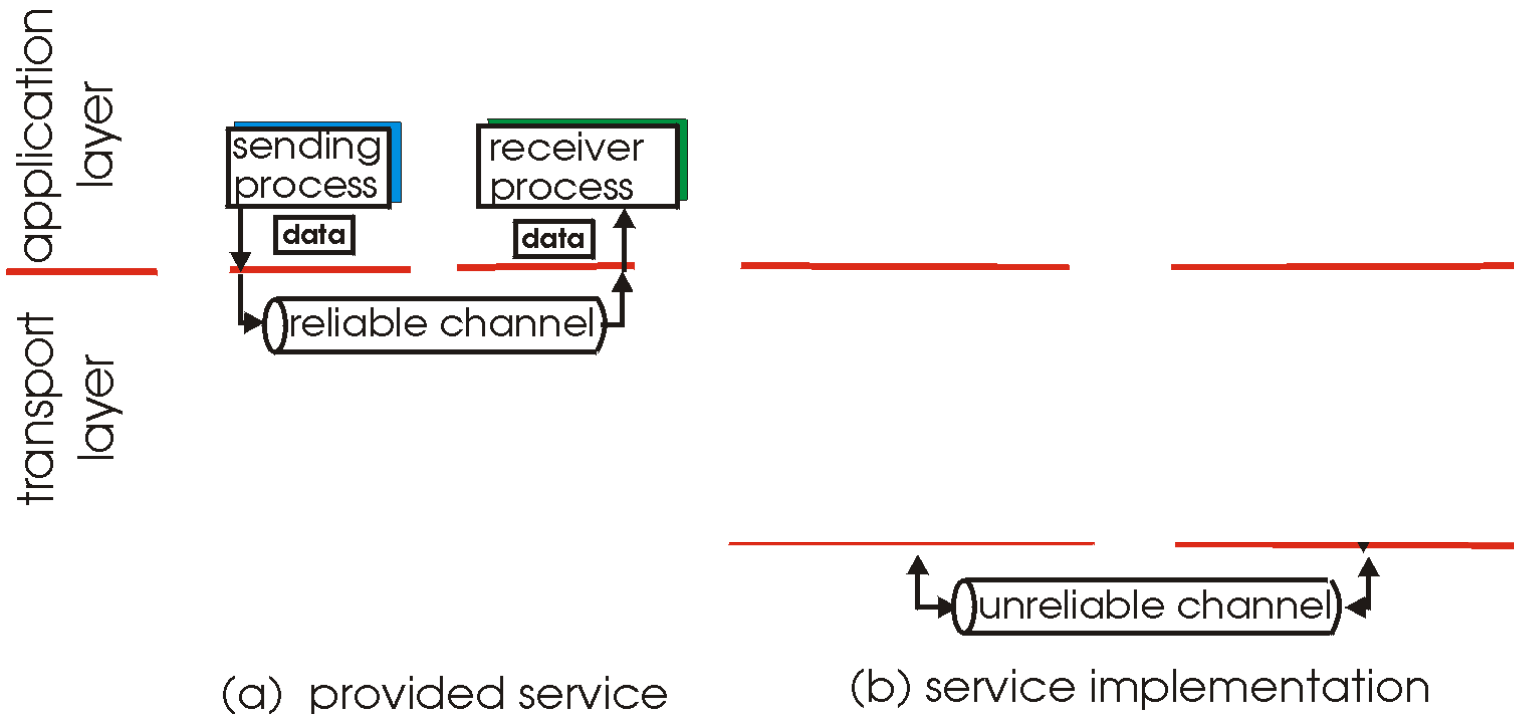
(a) provided service

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



# Principles of reliable data transfer

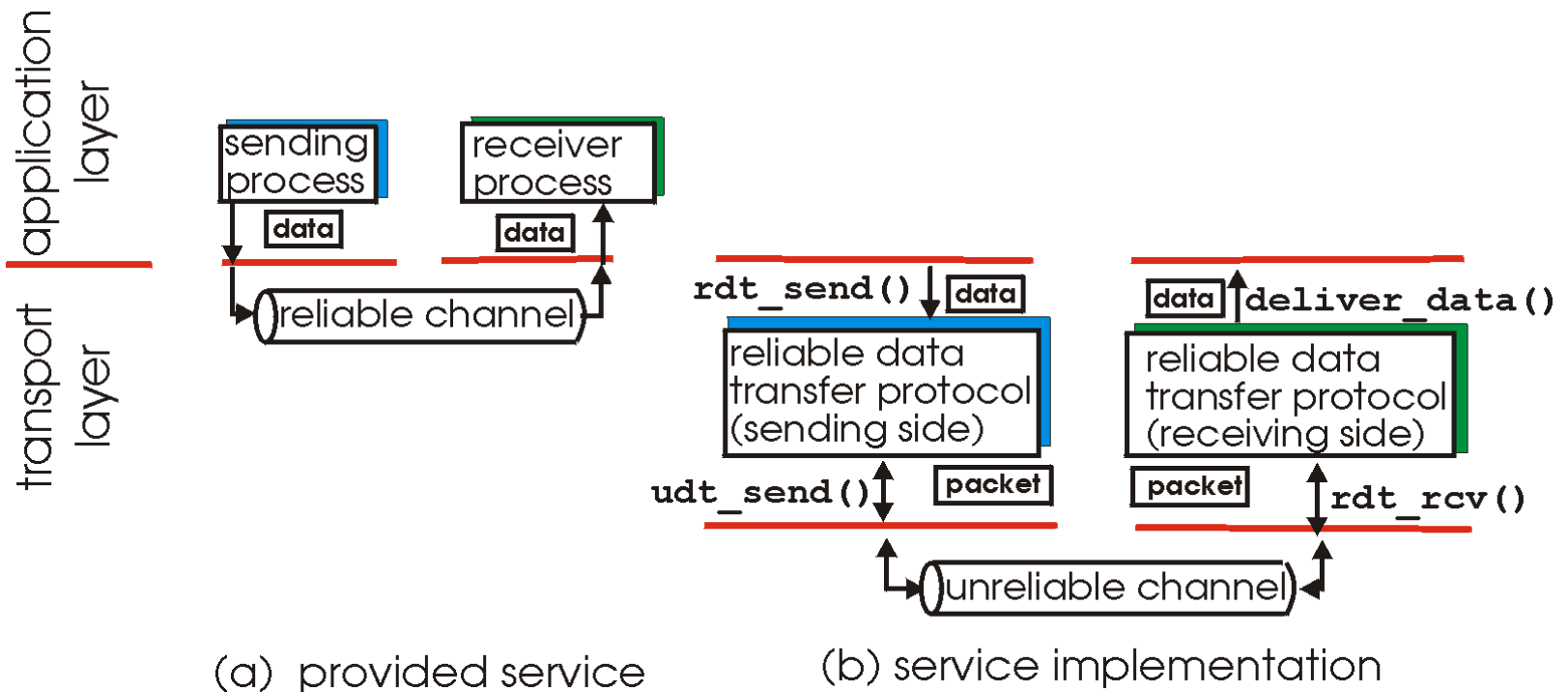
- ❖ important in application, transport, link layers
  - top-10 list of important networking topics!



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

# Principles of reliable data transfer

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

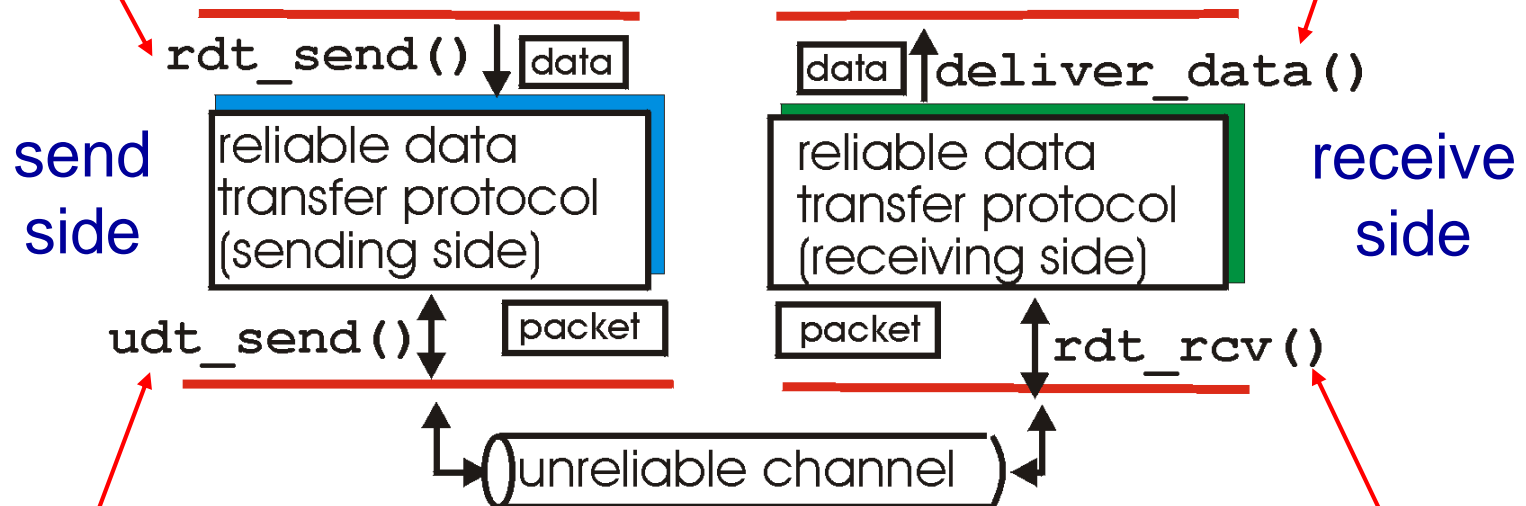


- ❖ characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

# 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

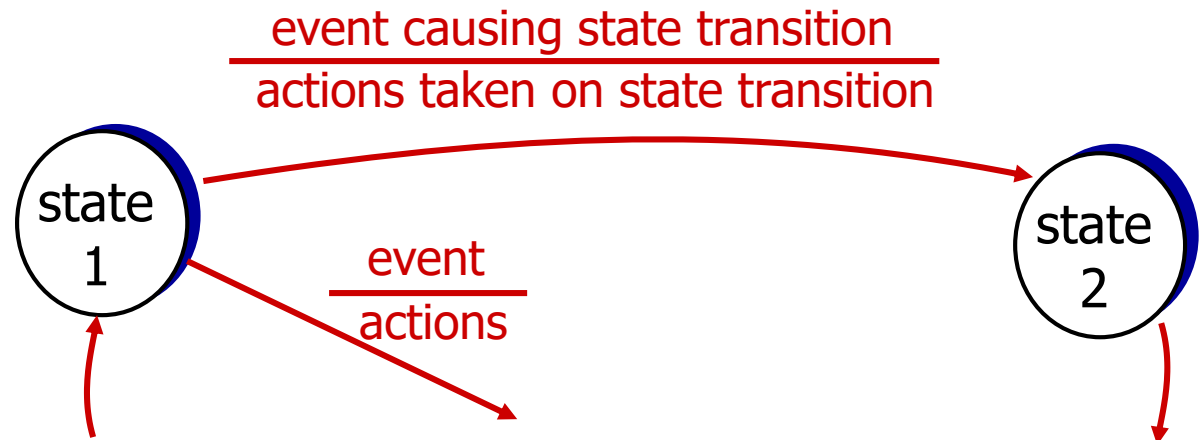
**rdt\_rcv()** : called when packet arrives on rcv-side of channel

# Reliable data transfer: getting started

we' ll:

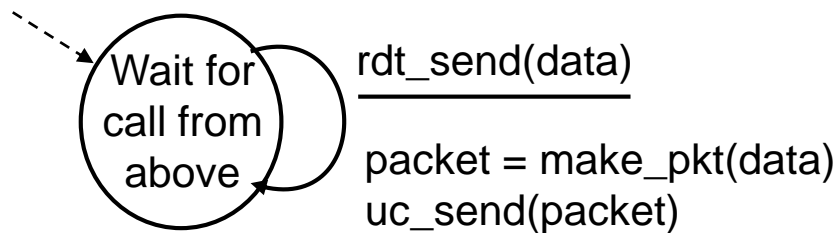
- ❖ incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- ❖ consider only unidirectional data transfer
  - but control info will flow on both directions!
- ❖ use finite state machines (FSM) to specify sender, receiver

**state:** when in this “state” next state uniquely determined by next event

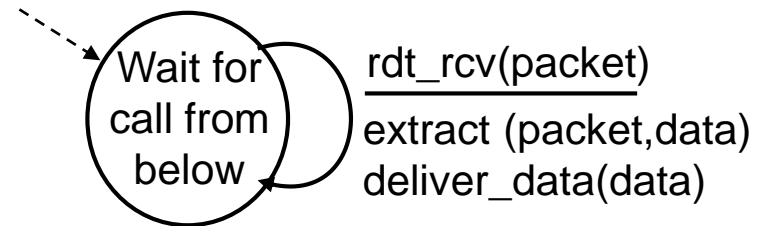


# rdt1.0: reliable transfer over a reliable channel

- ❖ 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



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:*

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

# rdt2.0: channel with bit errors

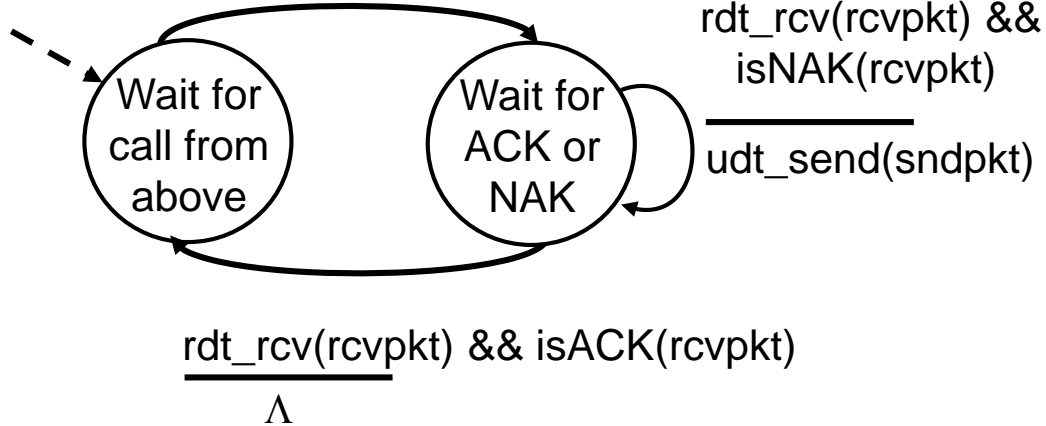
- ❖ underlying channel may flip bits in packet
  - checksum to detect bit errors
- ❖ *the question*: how to recover from errors:
  - *acknowledgements (ACKs)*: receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs)*: receiver explicitly tells sender that pkt had errors
  - 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

# rdt2.0: FSM specification

rdt\_send(data)

sndpkt = make\_pkt(data, checksum)

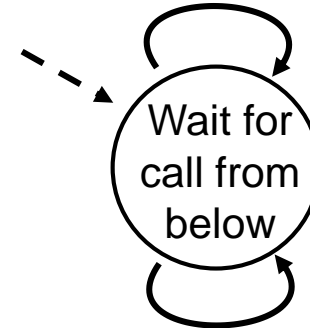
udt\_send(sndpkt)



sender

receiver

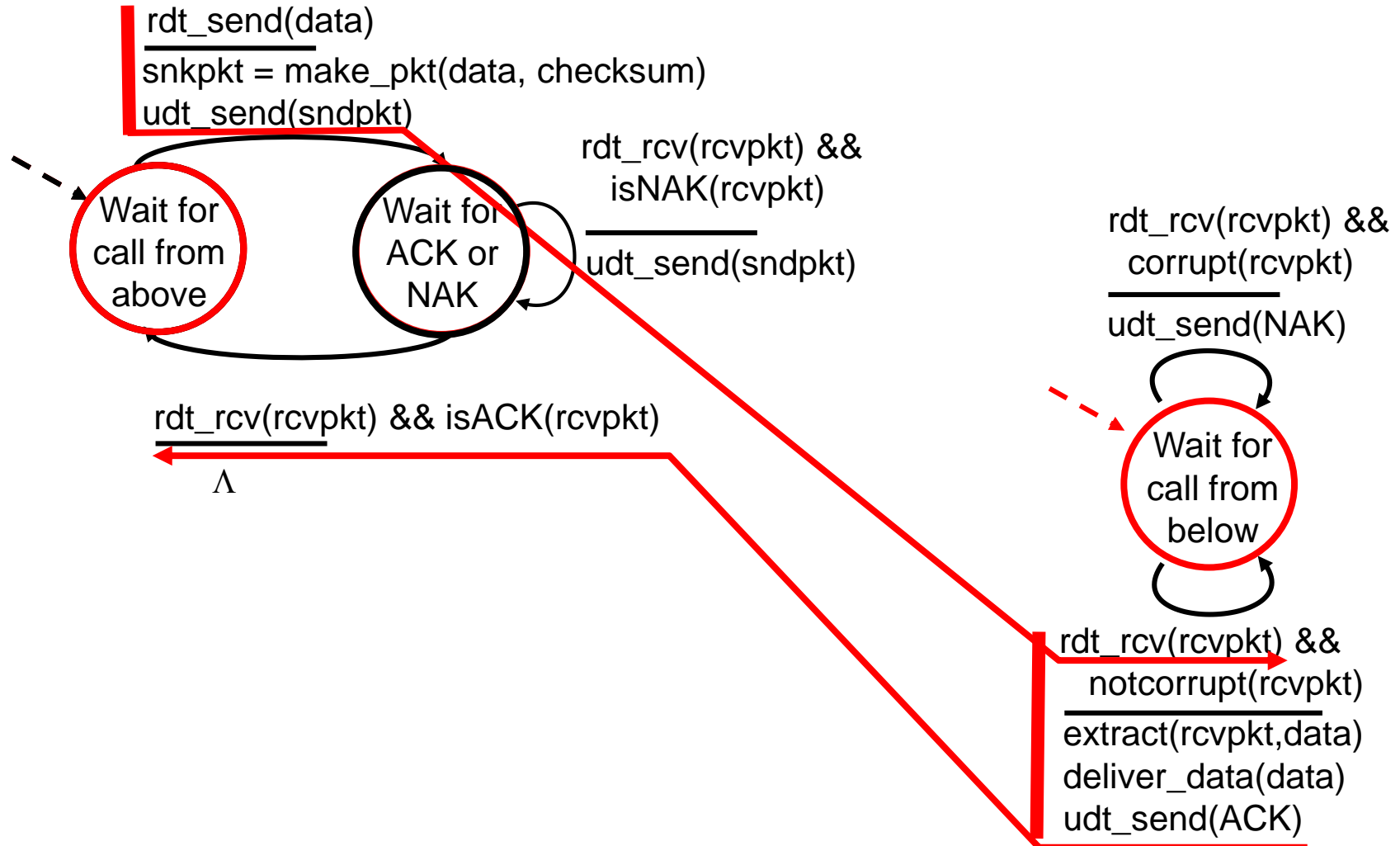
rdt\_rcv(rcvpkt) && corrupt(rcvpkt)  
udt\_send(NAK)



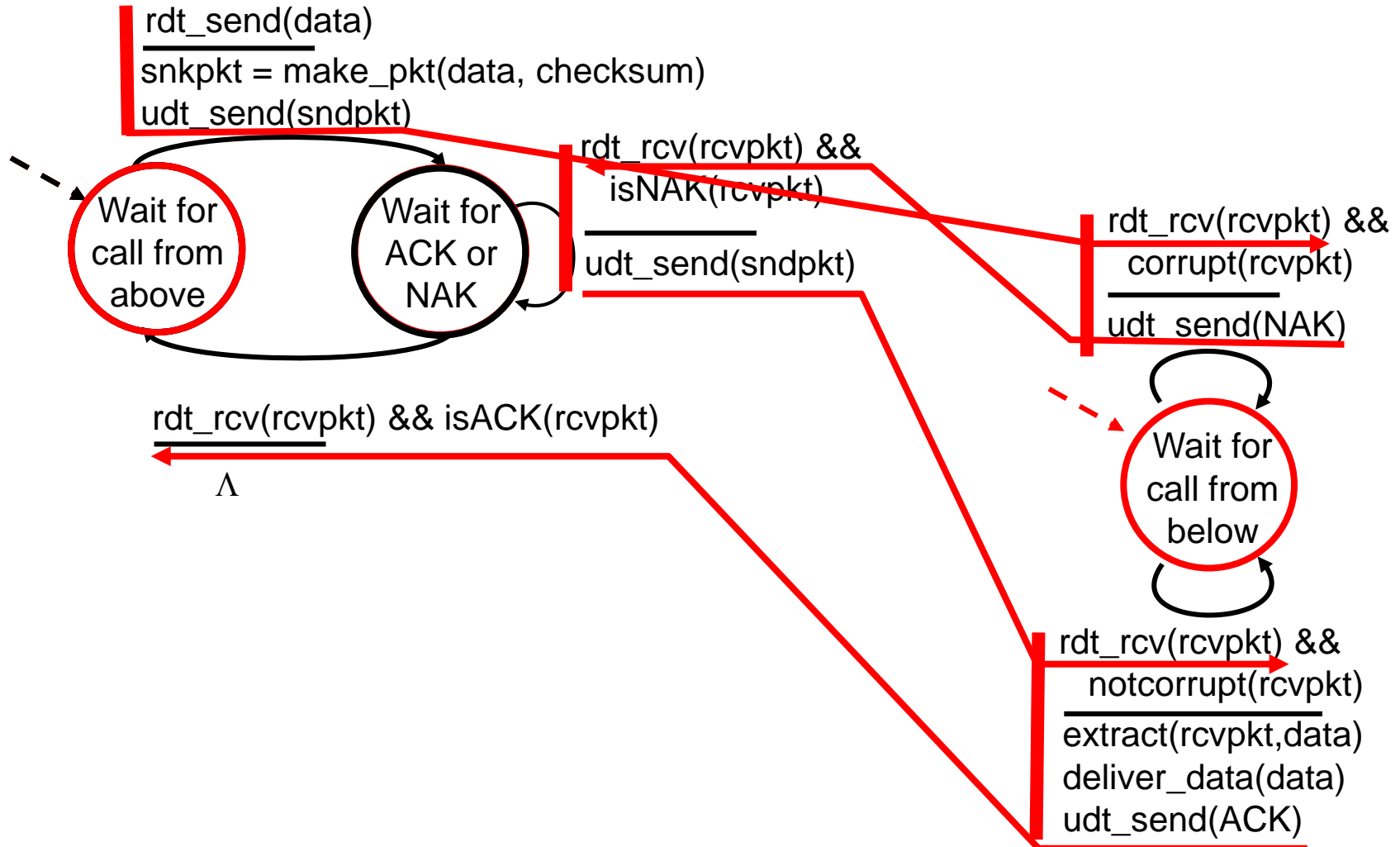
rdt\_rcv(rcvpkt) && notcorrupt(rcvpkt)  
extract(rcvpkt,data)  
deliver\_data(data)  
udt\_send(ACK)



# rdt2.0: operation with no errors



# rdt2.0: error scenario



# 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

# Next class

- ❖ Please read Chapter 3.3-3.3 of your textbook  
**BEFORE** Class