Information Network

Lecture 7 : UDP & TCP

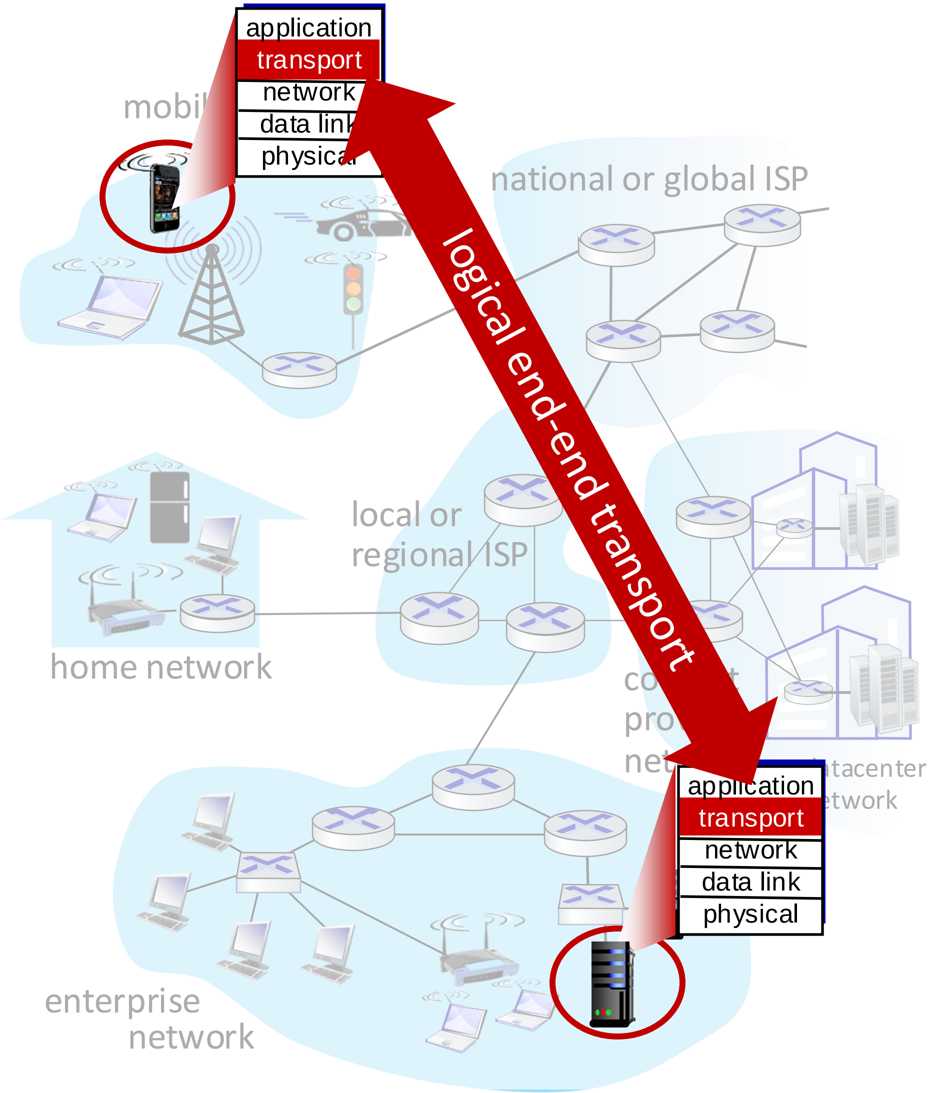
Holger Thies

KYOTO UNIVERSITY

京都大学

Two principal Internet transport protocols

- **TCP:** Transmission Control Protocol
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection setup
- **UDP:** User Datagram Protocol
  - unreliable, unordered delivery
  - extension of "best-effort" IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



京都大学

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

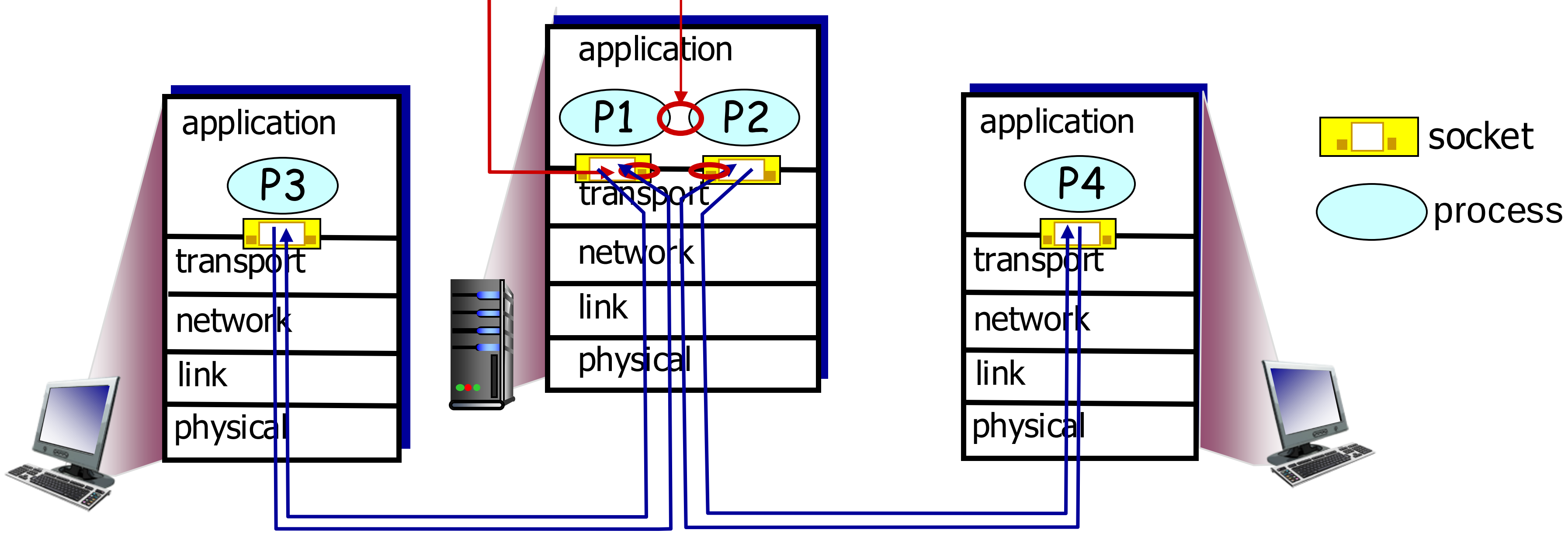
3.3 connectionless transport: UDP

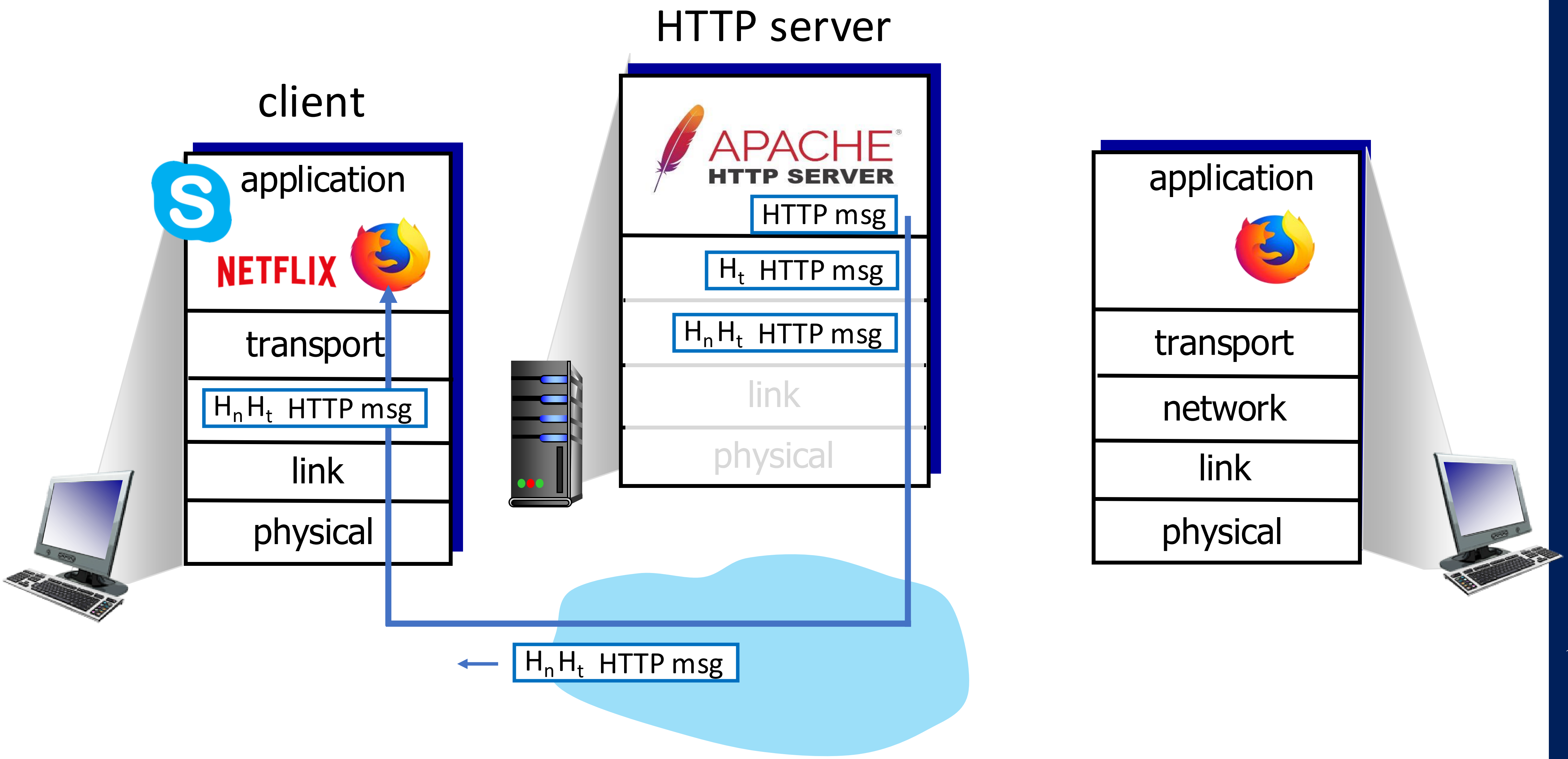
京都大学

Multiplexing/demultiplexing

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

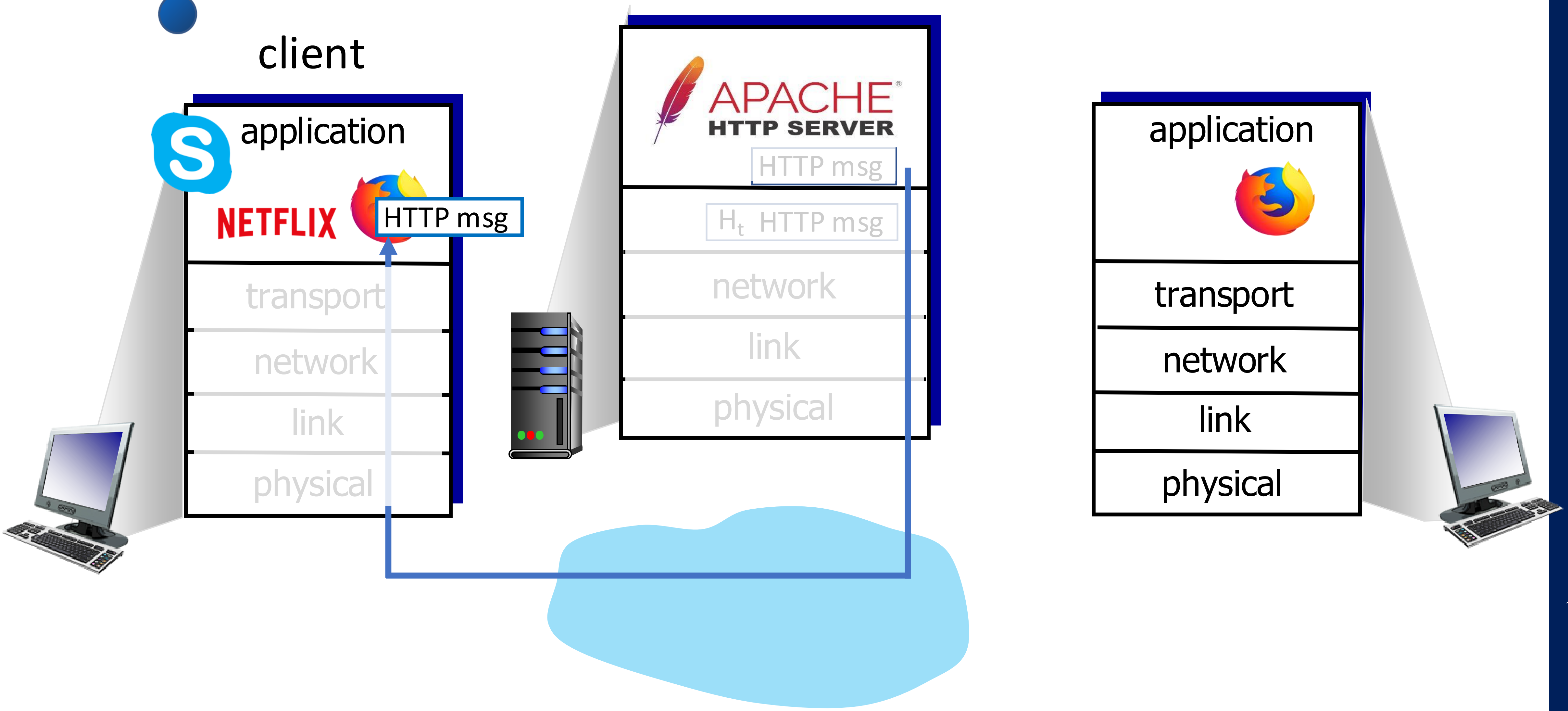
demultiplexing at receiver:
use header info to deliver received segments to correct socket

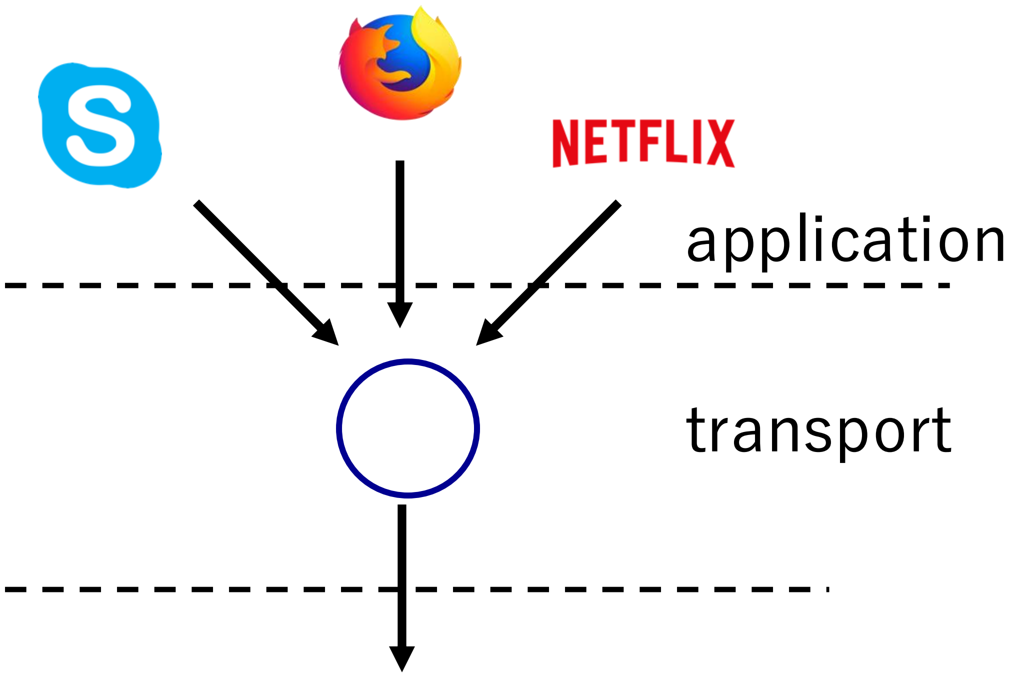



HTTP server
client
application
NETFLIX
transport
$H_n H_t$ HTTP msg
link
physical
APACHE
HTTP SERVER
HTTP msg
H_t HTTP msg
$H_n H_t$ HTTP msg
link
physical
$H_n H_t$ HTTP msg
application
transport
network
link
physical

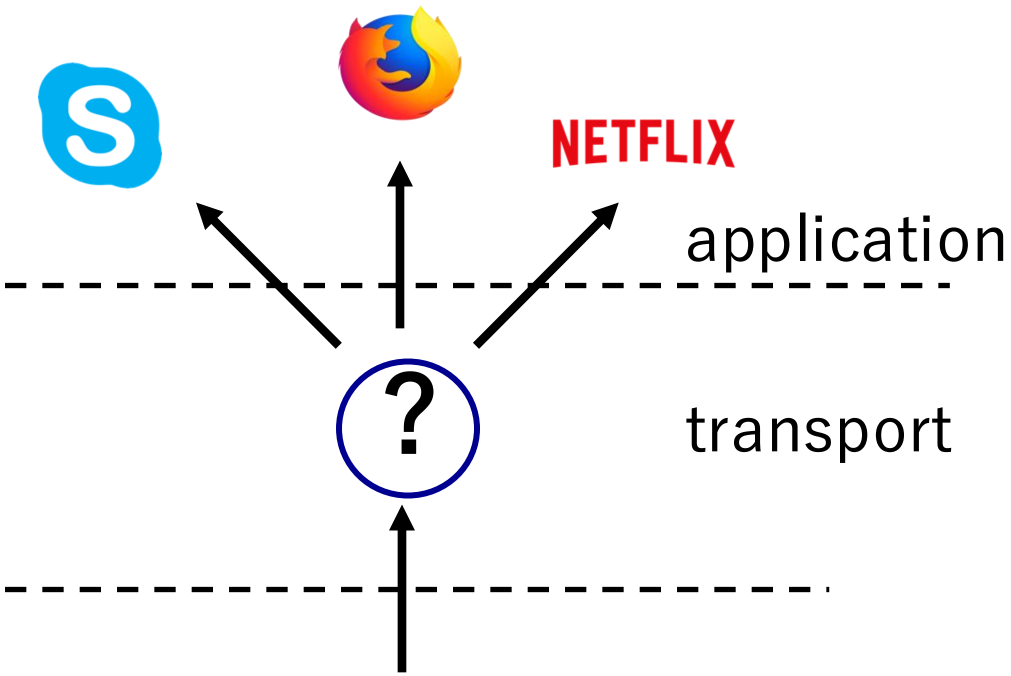
京都大学

Q: how did transport layer know to deliver message to Firefox browser process rather then Netflix process or Skype process?





multiplexing



de-multiplexing

Summary

- Multiplexing, demultiplexing: based on segment, datagram header field values
- **UDP:** demultiplexing using destination port number (only)
  - Connection-less multiplexing
- **TCP:** demultiplexing using 4-tuple: source and destination IP addresses, and port numbers
  - Connection-oriented multiplexing

京都大学

Today's lecture

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

京都大学

UDP: User Datagram Protocol [RFC 768]

INTERNET STANDARD

RFC 768 J. Postel
ISI
28 August 1980

User Datagram Protocol

Introduction

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

Format

```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|
|          data octets ...
+---------------- ...
```

UDP: User Datagram Protocol [RFC 768]

- Very simple 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

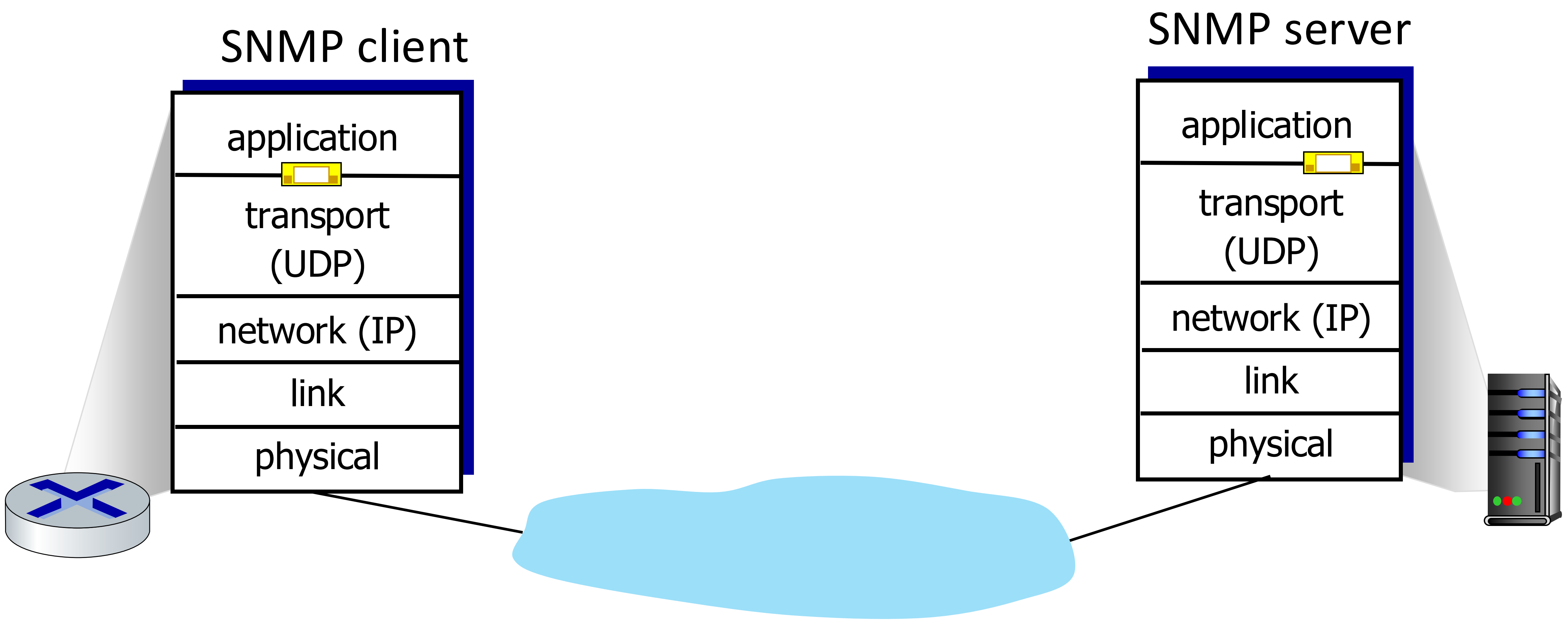
- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP (used to manage network printers etc.)
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

京都大学

Advantages of UDP over TCP

- Finer application-level control over what data is sent and when.
  - Packet will be immediately passed to network layer
  - Real-time applications often require a minimum sending rate.
  - TCP not well-suited for these applications.
- No connection establishment
  - No delay to establish connection.
- No connection state
  - A server can support many more active clients when the application runs over UDP rather than TCP.
- Small packet overhead
  - TCP segment has 20 bytes of header overhead, UDP only 8 bytes.

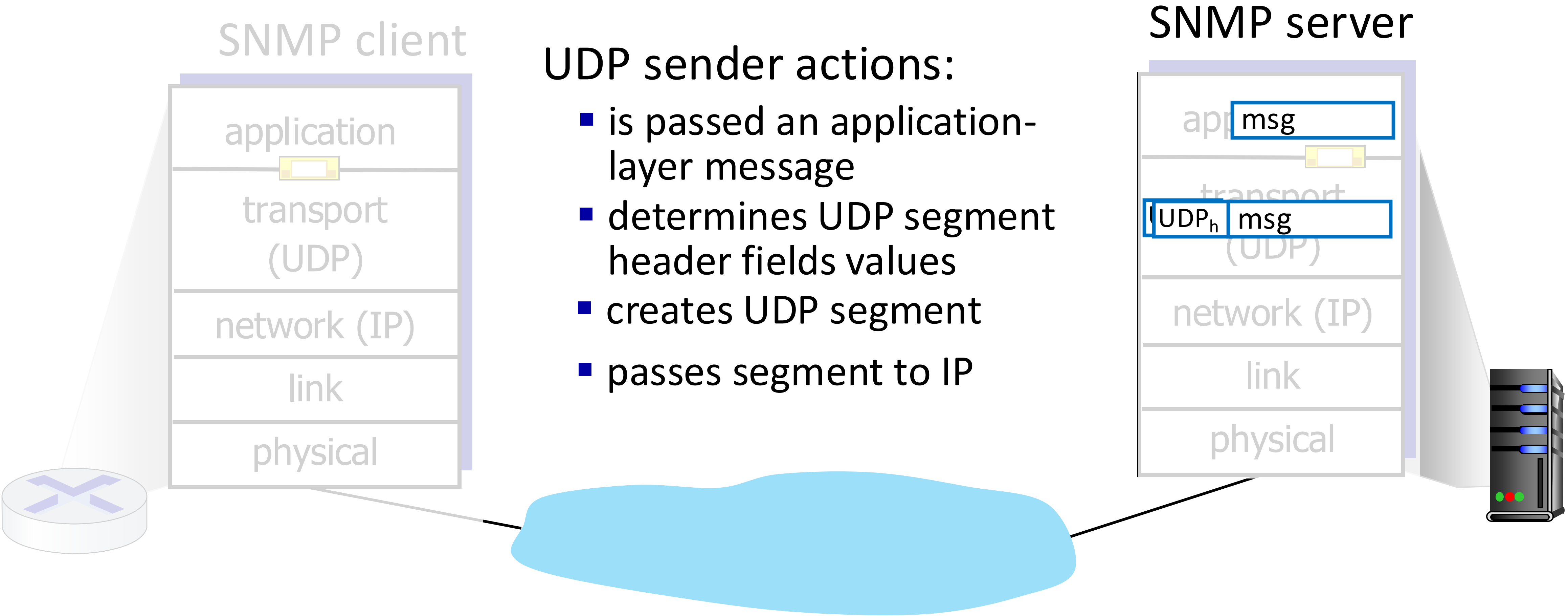
UDP: Transport Layer Actions



UDP: Transport Layer Actions

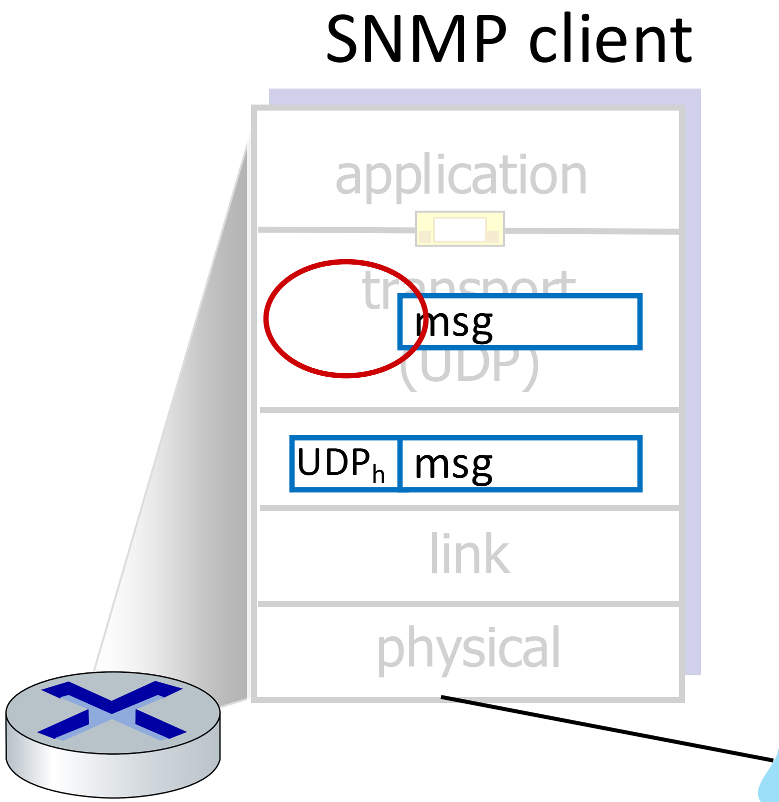
UDP sender actions:

- is passed an application-layer message
- determines UDP segment header fields values
- creates UDP segment
- passes segment to IP



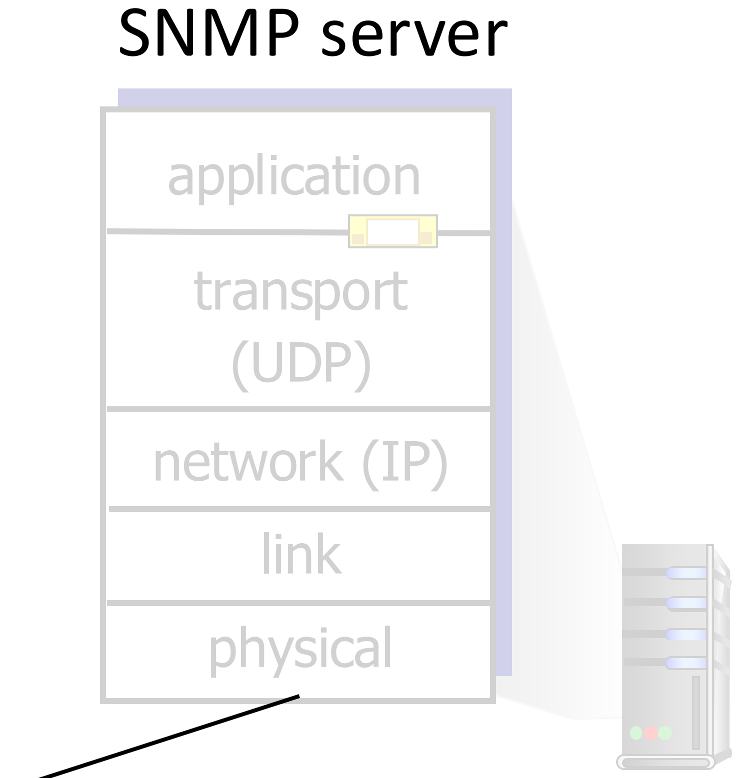
京都大学

UDP: Transport Layer Actions



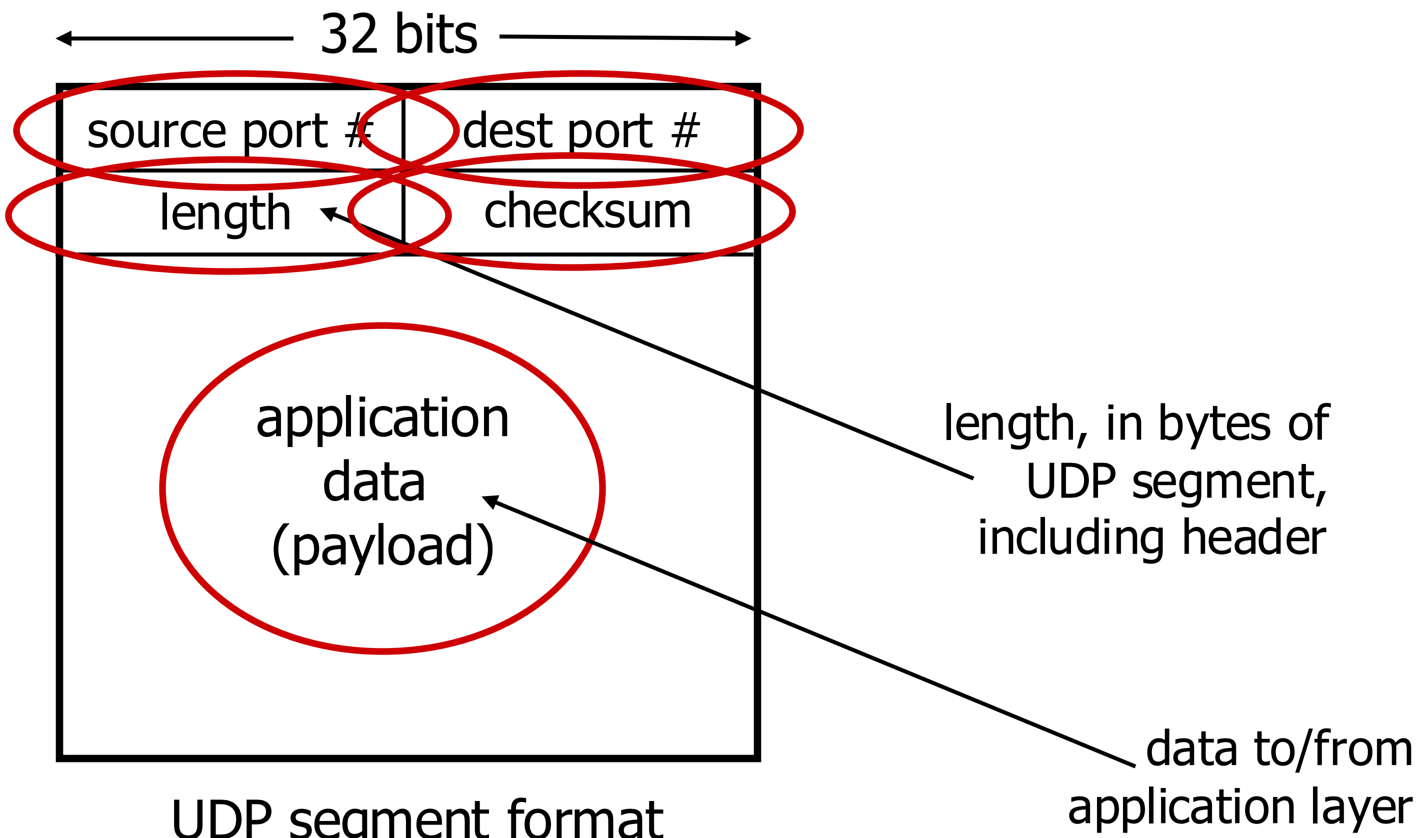
UDP receiver actions:

- receives segment from IP
- checks UDP checksum header value
- extracts application-layer message
- demultiplexes message up to application via socket



京都大学

UDP segment header



UDP segment format

京都大学

UDP checksum

Goal: detect errors (*i.e.*, flipped bits) in transmitted segment

	1st number	2nd number	sum
Transmitted:	5	6	11
Received:	4	6	11

$\underbrace{4 \quad 6}_{\text{receiver-computed checksum}} \neq \underbrace{11}_{\text{sender-computed checksum (as received)}}$

receiver-computed checksum ≠ sender-computed checksum (as received)

!

京都大学

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
- The UDP checksum is also called Internet checksum.

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?*

京都大学

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 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

京都大学

Internet checksum: weak protection!

example: add two 16-bit integers

		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0 1
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1 0
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	

Even though numbers have changed (bit flips), *no* change in checksum!

Internet Checksum Summary

- Used to detect errors (flipped bits) in received segment.
  - Receiver can decide what to do when an error is detected (discard segment, deliver despite error, give a warning, etc.)
- It is easy to compute and easy to verify.
- Not all errors can be detected.
  - Only one bit errors are guaranteed to be detected.
  - Easy to construct examples where 2-bit errors are not caught.
  - In typical data transmission scenarios, most (but not all!) errors can be detected.
  - Lower layers use more complicated checksums for better error detection.

Summary: UDP

- Simple protocol:
  - segments may be lost, delivered out of order
  - best effort service: “send and hope for the best”
- UDP has its plusses:
  - no setup/handshaking needed
  - faster
  - helps with reliability (checksum)
- build additional functionality on top of UDP in application layer (e.g., HTTP/3)

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

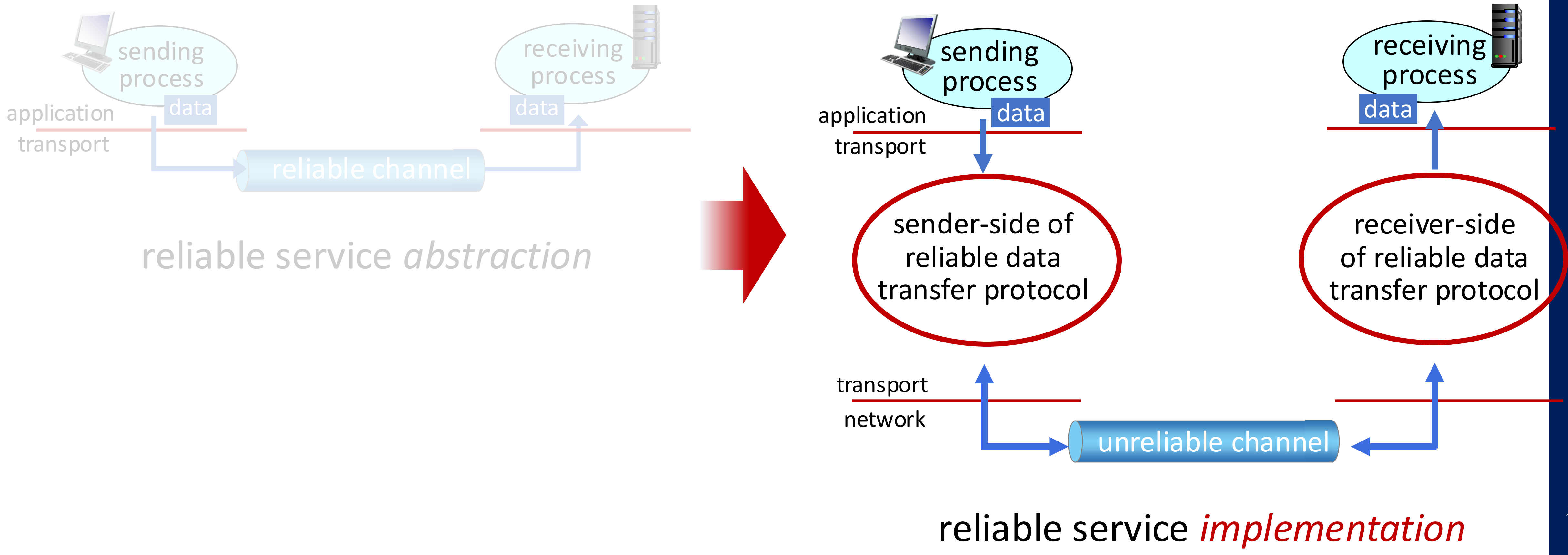
3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

京都大学

Principles of reliable data transfer

reliable service *abstraction*

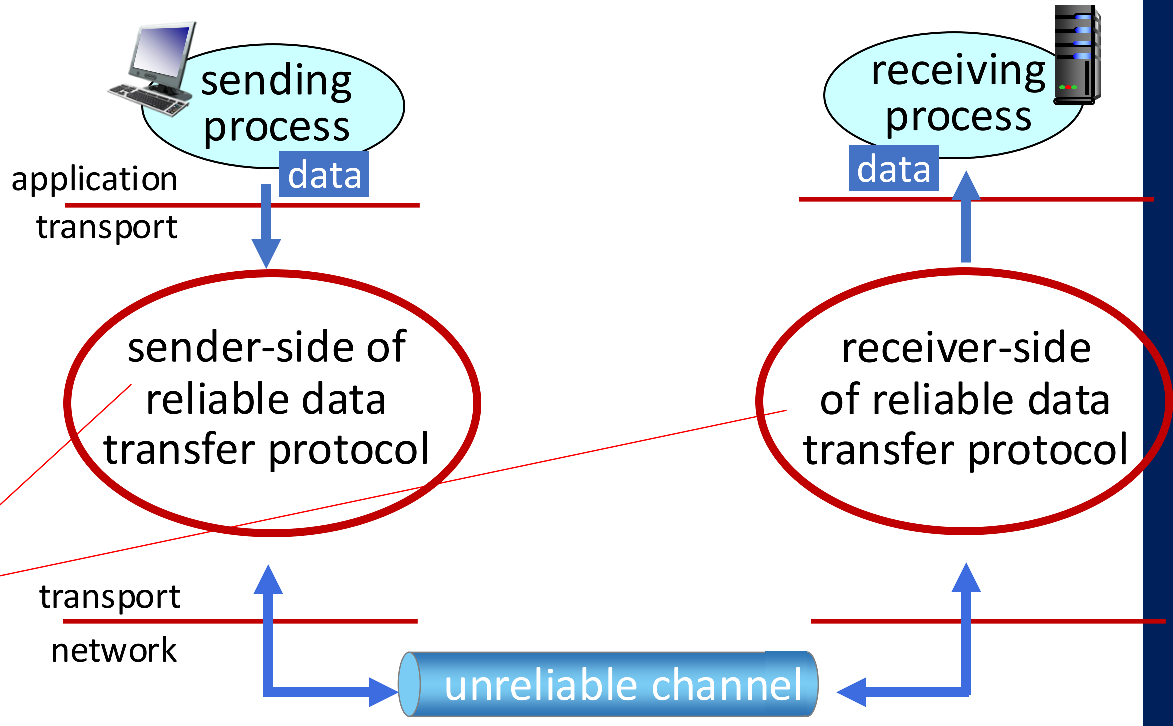
Principles of reliable data transfer



京都大学

Principles of reliable data transfer

Complexity of reliable data transfer protocol will depend (strongly) on characteristics of unreliable channel (lose, corrupt, reorder data?)



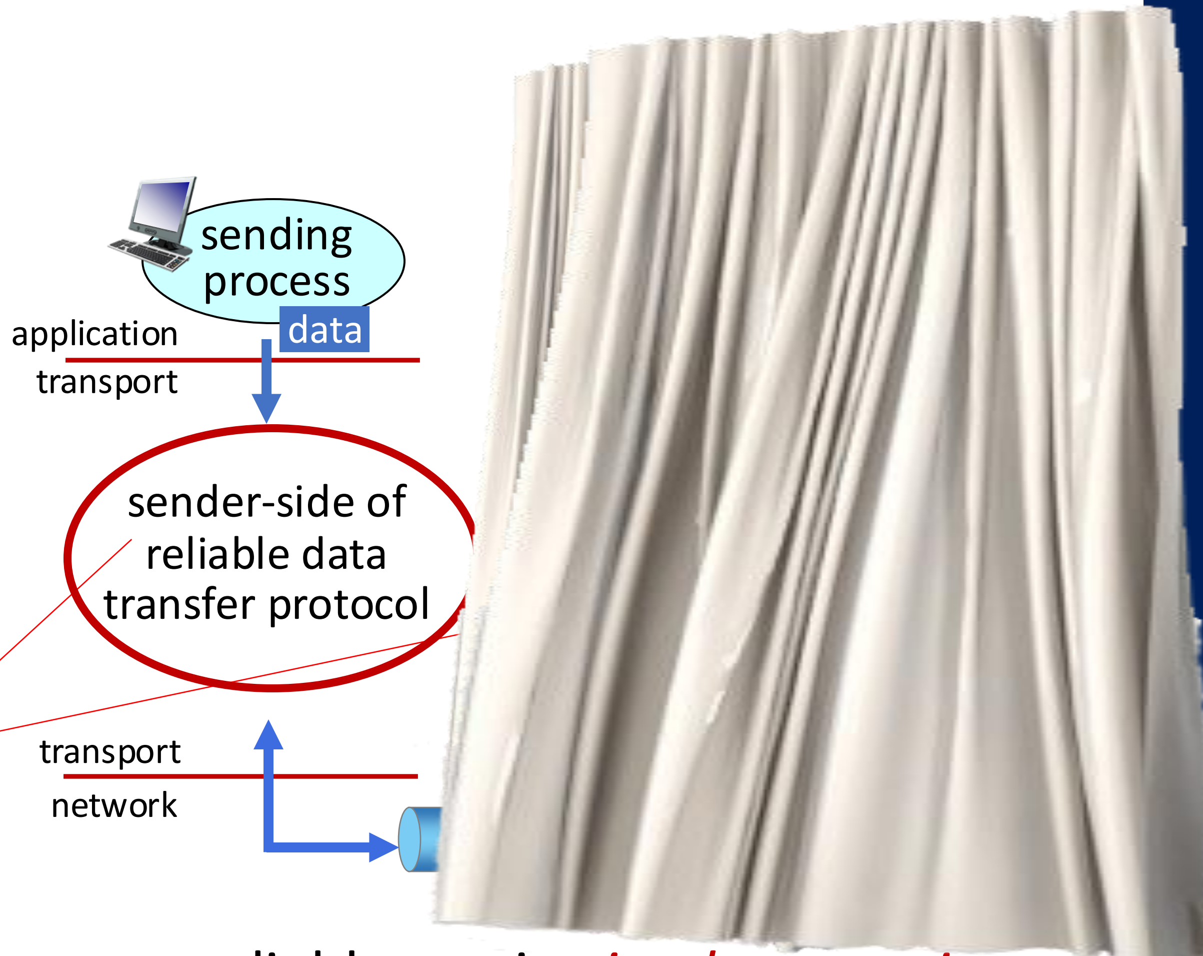
reliable service *implementation*

京都大学

Principles of reliable data transfer

Sender, receiver do *not* know the "state" of each other, e.g., was a message received?

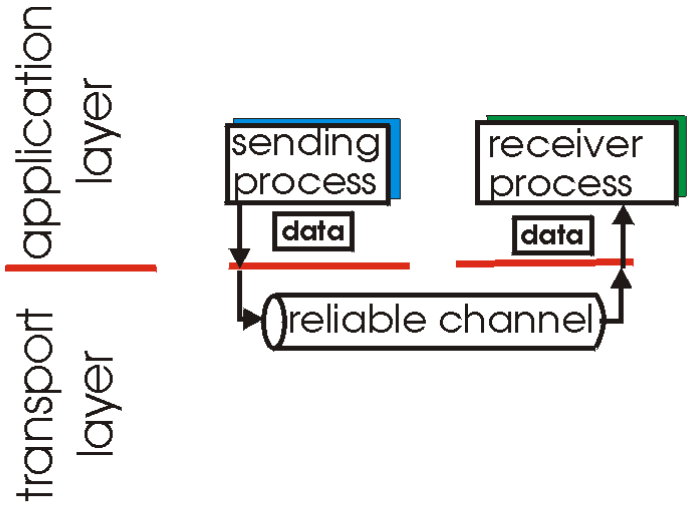
- unless communicated via a message



reliable service *implementation*

Principles of reliable data transfer

- important in application, transport, link layers

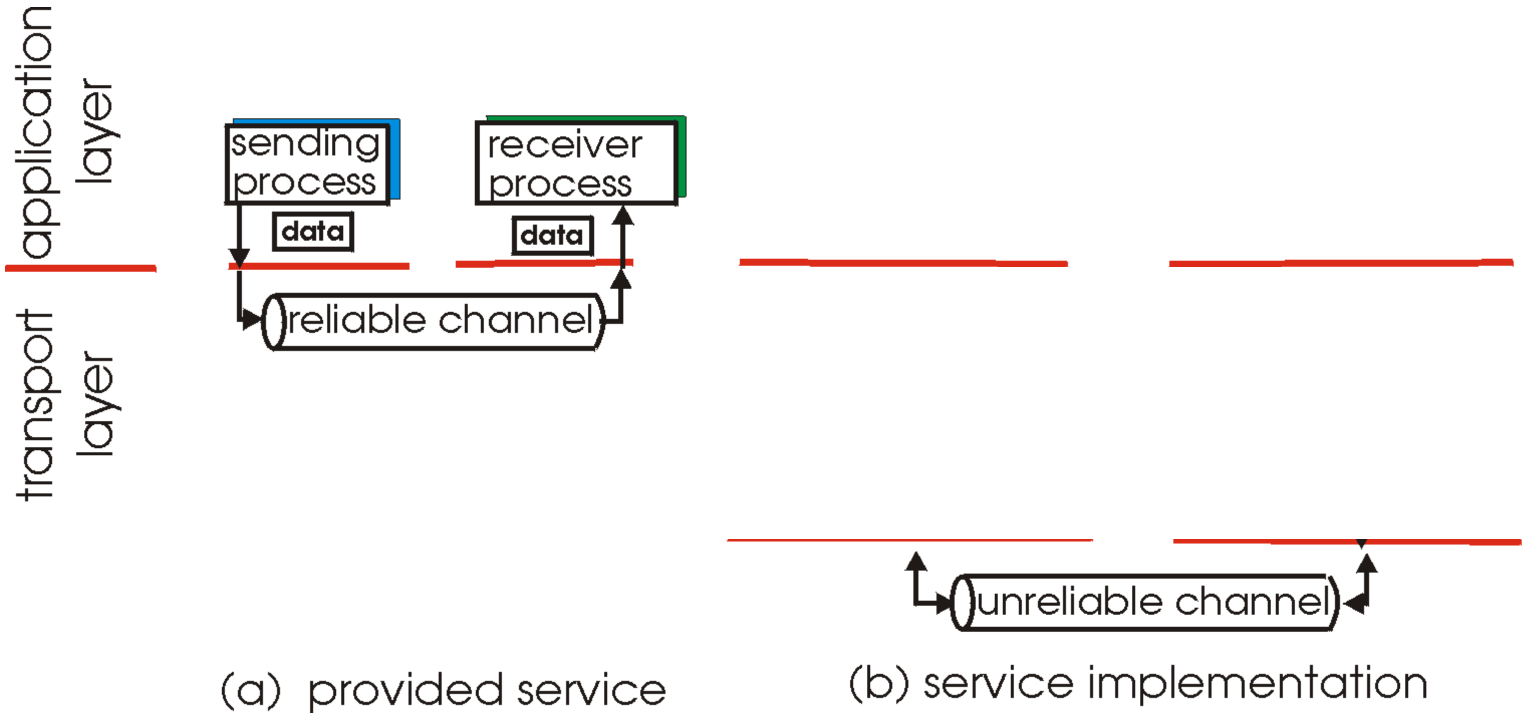


(a) provided service

京都大学

Principles of reliable data transfer

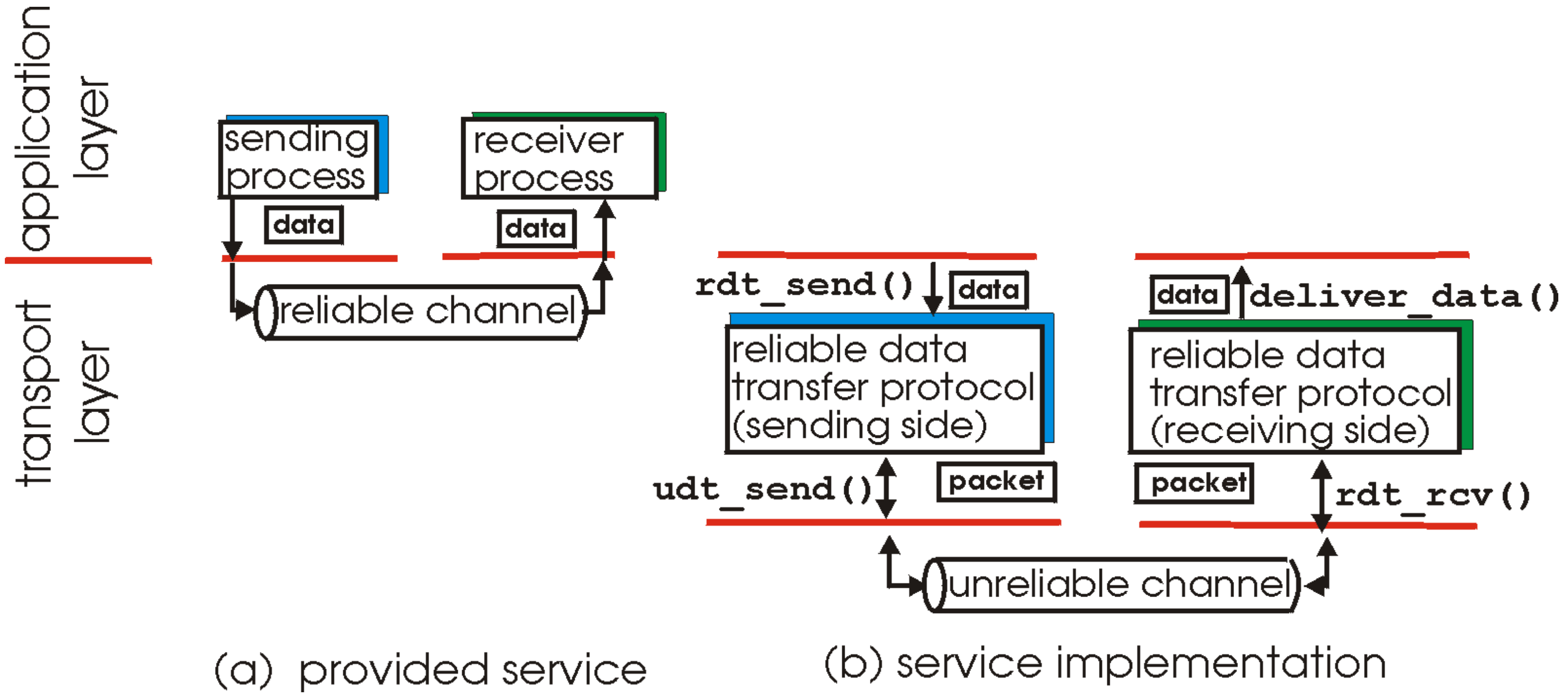
- important in application, transport, link layers
  - top-10 list of important networking topics!



京都大学

Principles of reliable data transfer

- important in application, transport, link layers
  - One of the most important networking topics!

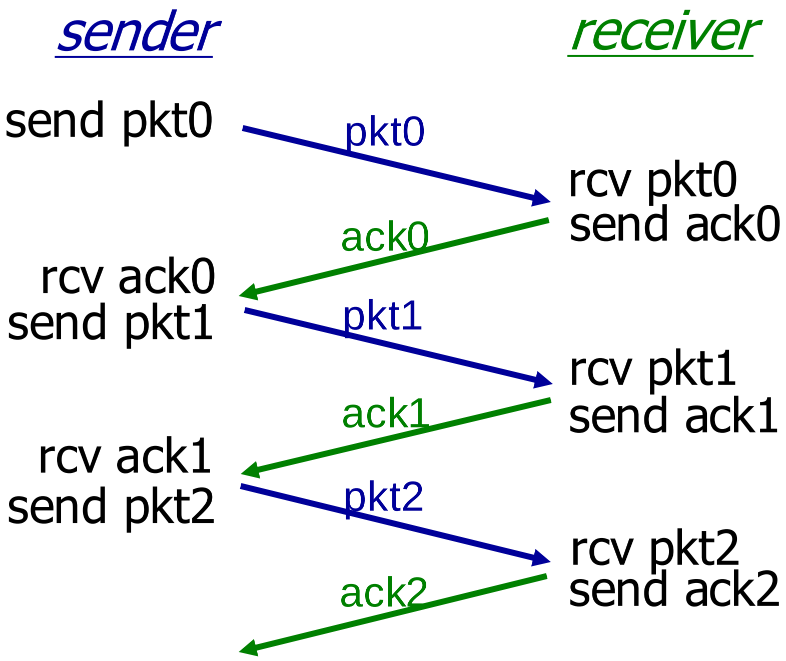


- characteristics of unreliable channel will determine complexity of reliable data transfer protocol
- We assume that unreliable channel can lose packets and that packets can have errors.

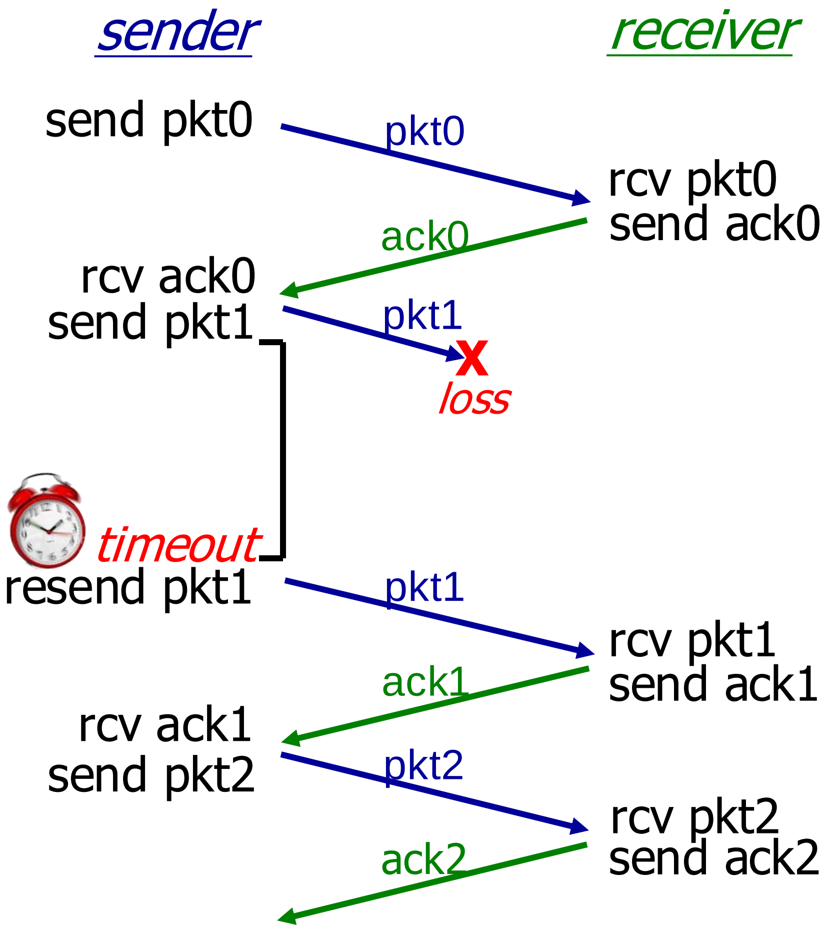
Sequence numbers

- Packets are sent one by one and are numbered (sequence numbers).
- When a packet is sent, its sequence number is contained in the header information.
- When the receiver receives a packet and it does not contain any errors an acknowledgment message (ACK) is sent for this packet and the receiver remembers the sequence number.
- The sender waits for the acknowledgment packet until it sends the next packet.
- If the sender does not receive an acknowledgment for some time, the same packet is sent again.
- If the receiver receives a packet with a sequence number other than expected or the packet has errors, the packet is discarded, and an acknowledgment message for the last valid packet is send.

Reliable data transfer using ack messages

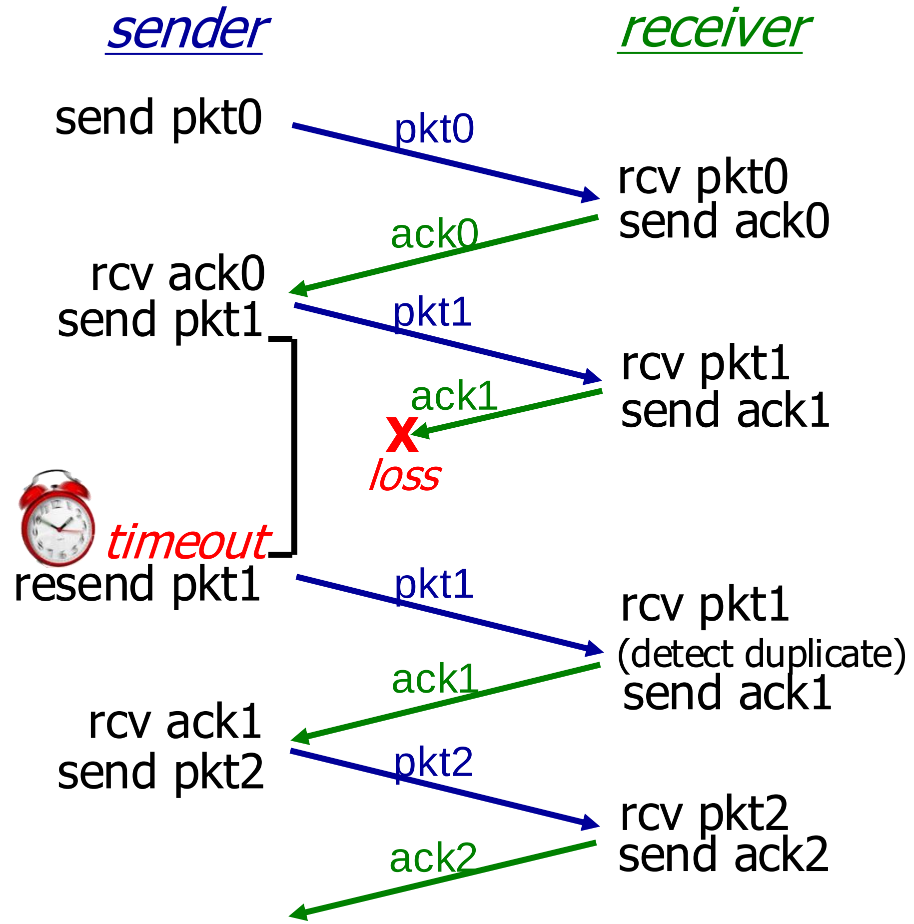


(a) no loss

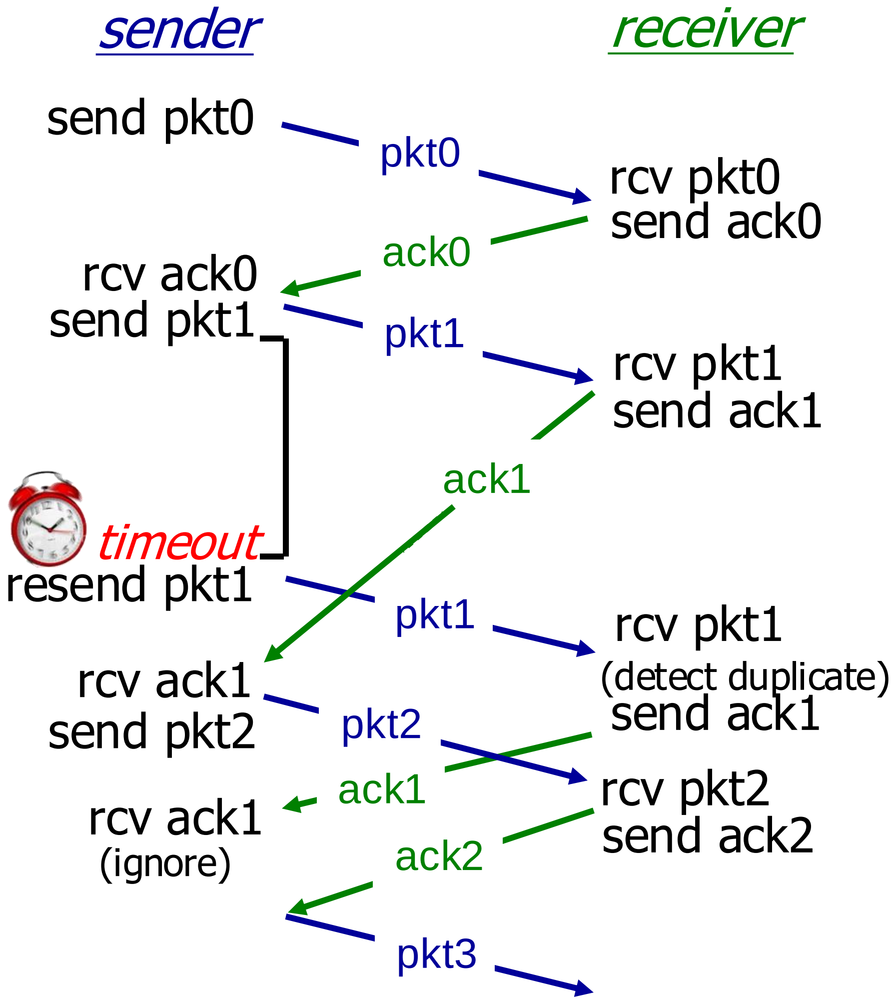


(b) packet loss

Reliable data transfer using ack messages



(c) ACK loss



(d) premature timeout/ delayed ACK

Note: 1 bit sequence number (0 or 1) suffices

京都大学

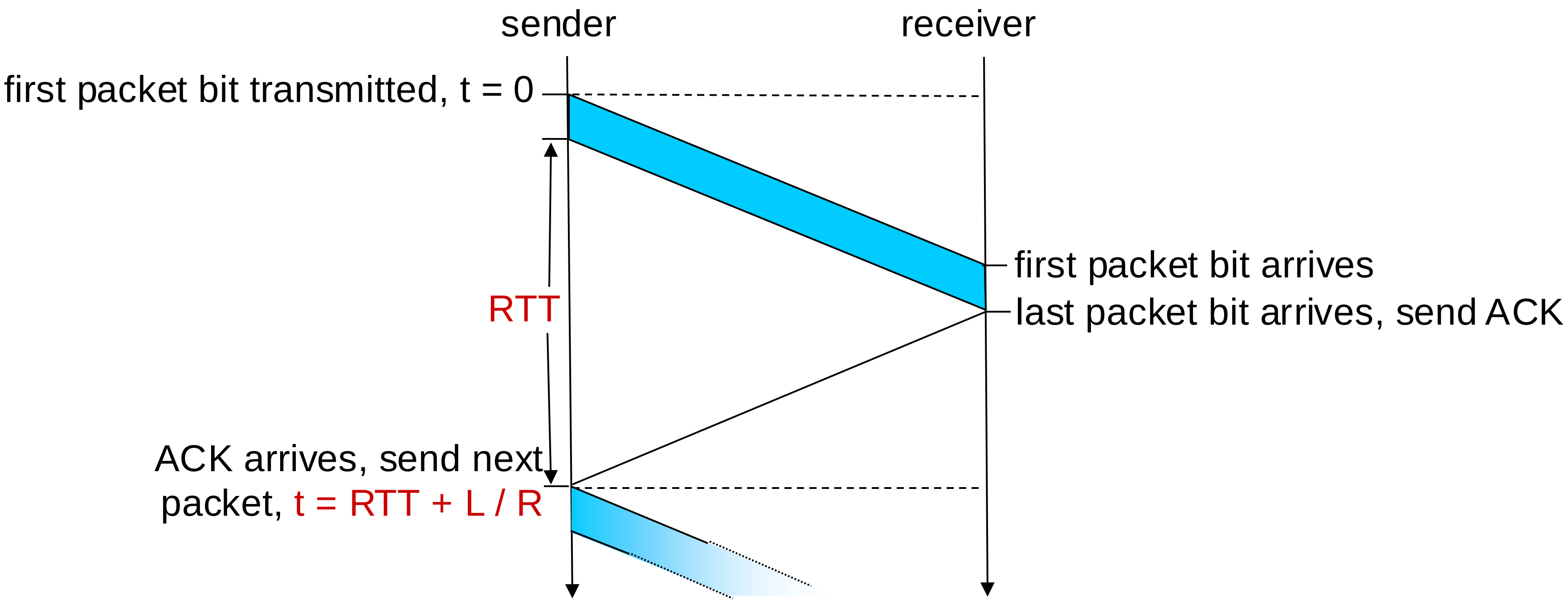
Performance

- Protocol is correct, but performance is not good
- We consider the utilization of the sender, i.e., the fraction of time the sender is busy sending data
- e.g.: 1 Gbps link, 15 ms propagation delay, 8000 bit packet
- Time to transmit data into channel:

$$D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$$

京都大学

Stop-and-wait operation

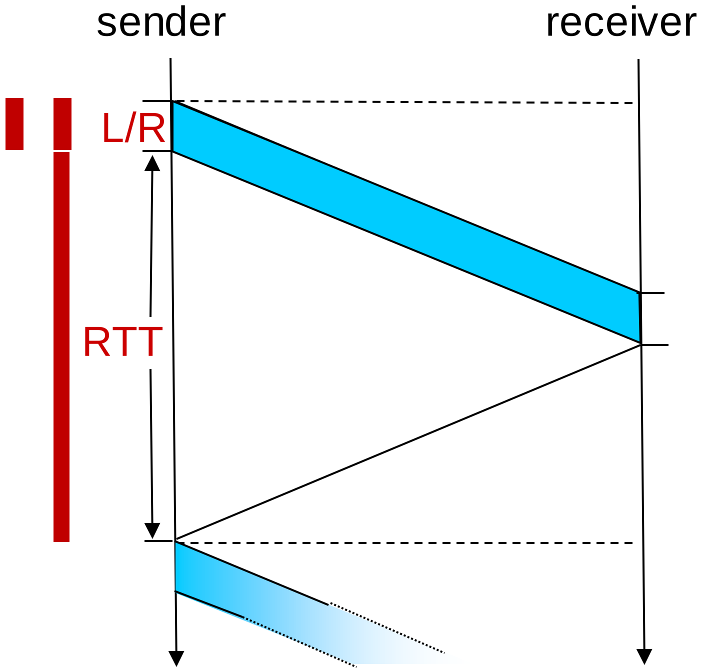


Stop-and-wait operation

$$U_{sender} = \frac{L / R}{RTT + L / R}$$

$$= \frac{.008}{30.008}$$

$$= 0.027\%$$

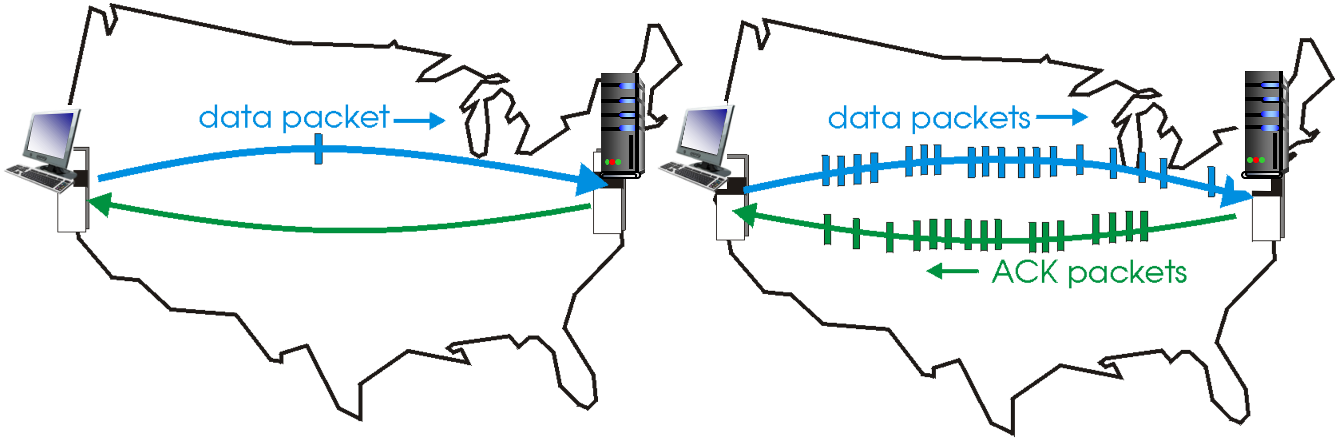


- if RTT=30 msec, 1KB packet every 30 msec: 33kB/sec throughput over 1 Gbps link
- Protocol limits performance of underlying infrastructure (channel)

京都大学

Pipelined protocols

pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged packets

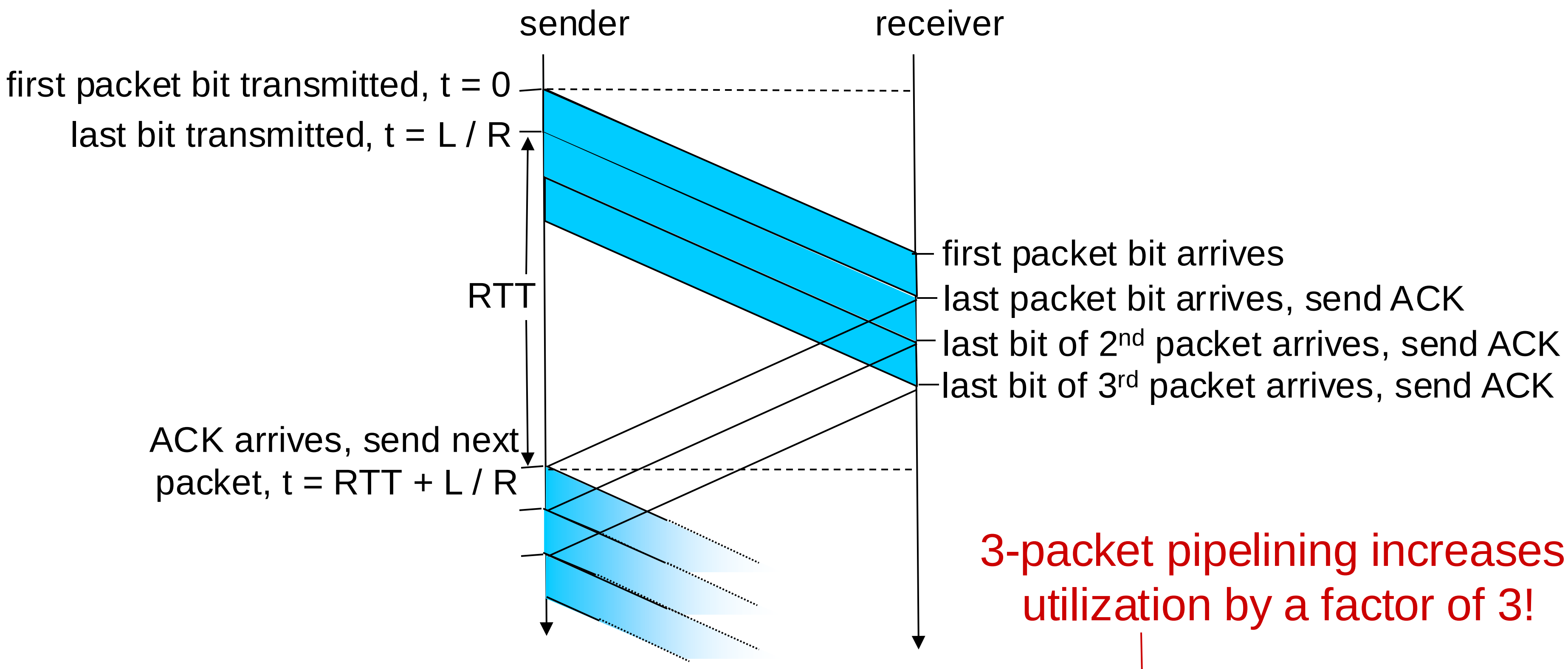


(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

京都大学

Pipelining: increased utilization



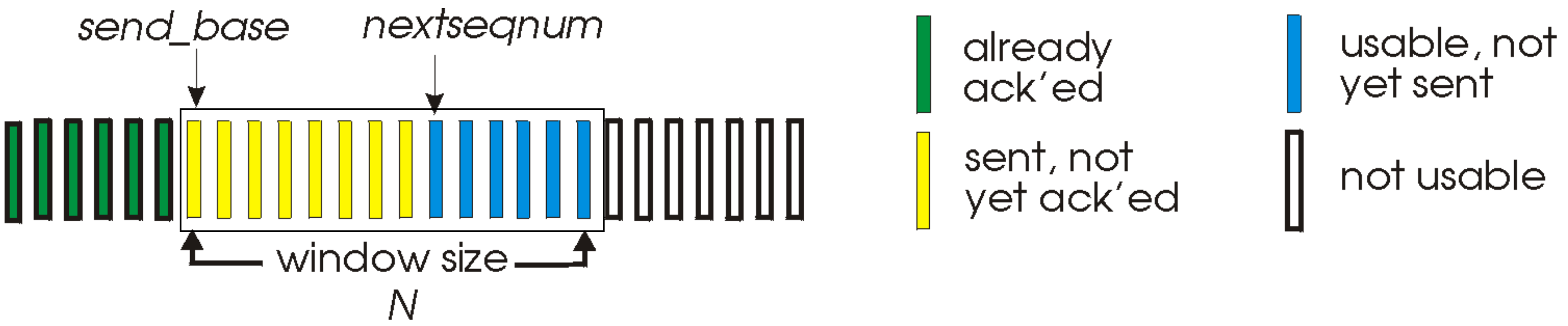
3-packet pipelining increases utilization by a factor of 3!

$$U_{sender} = \frac{3L / R}{RTT + L / R} = \frac{.0024}{30.008} = 0.00081$$

京都大学

Go-Back-N: sender

- k-bit sequence number in packet header
- "window" of up to N, consecutive unacknowledged packets allowed

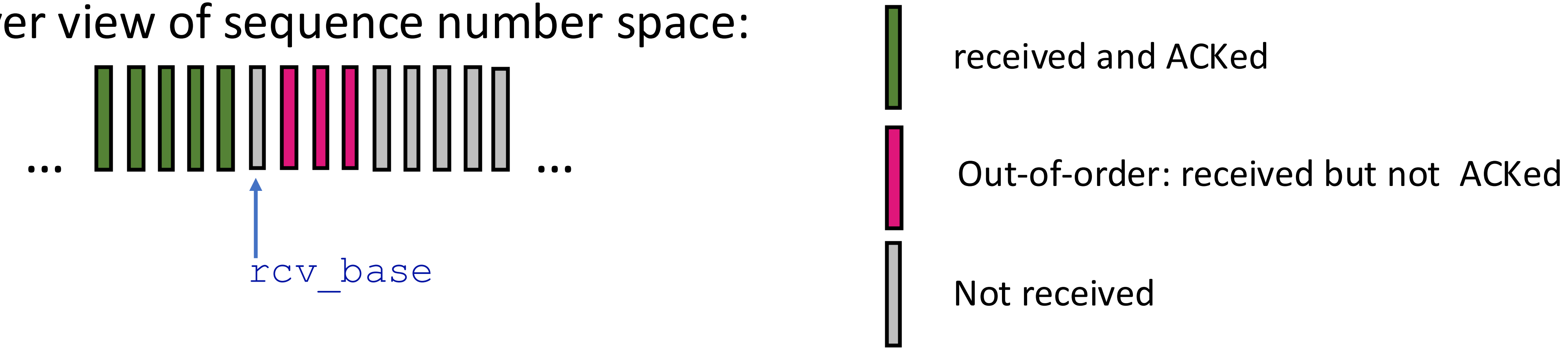


- ACK(n): ACKs all pkts up to, including sequence number n - *"cumulative ACK"*
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt n
- *timeout:* retransmit packet n and all higher sequence number packets in window

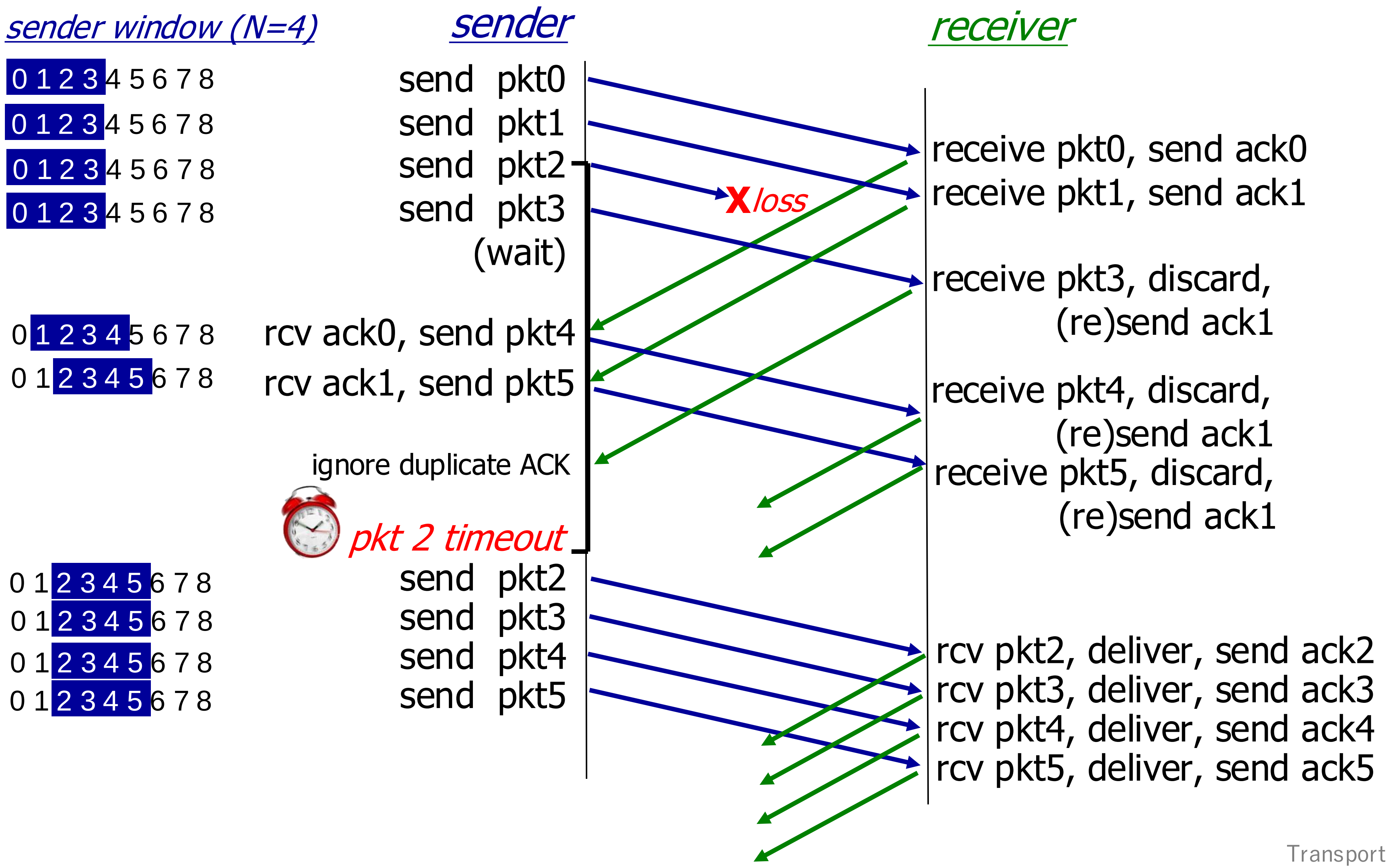
Go-Back-N: receiver

- ACK-only: always send ACK for correctly-received packet so far, with highest *in-order* seq number
  - may generate duplicate ACKs
  - need only remember `rcv_base`
- on receipt of out-of-order packet:
  - can discard (don't buffer) or buffer: an implementation decision
  - re-ACK packet with highest in-order sequence number

Receiver view of sequence number space:



Go-Back-N in action

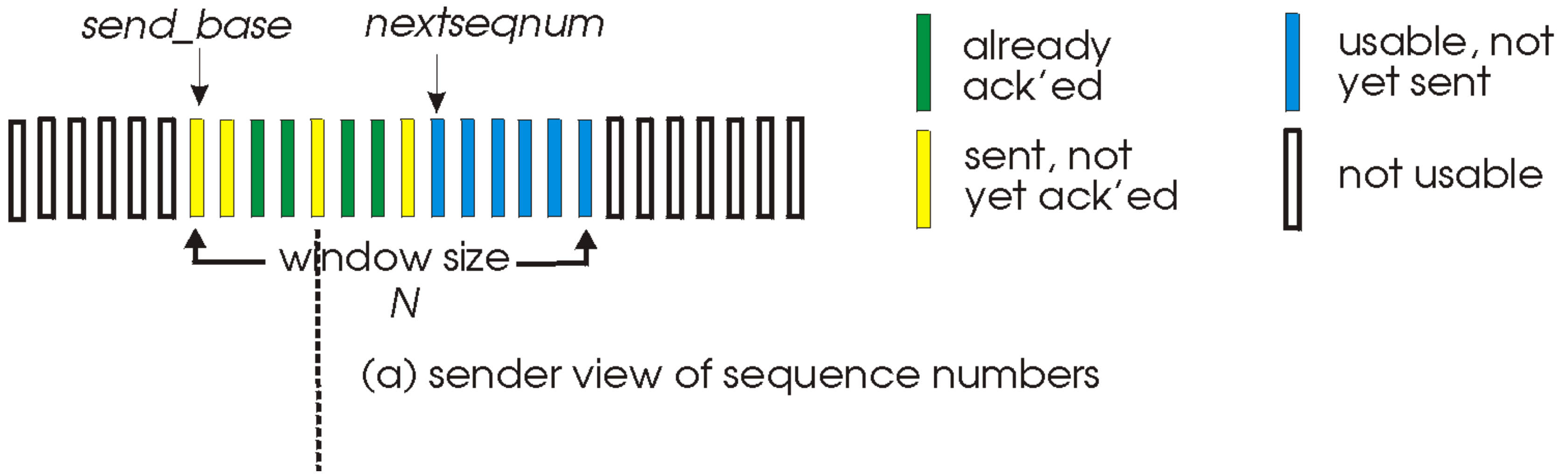


Selective repeat: the approach

- *pipelining*: *multiple* packets in flight
- *receiver individually ACKs* all correctly received packets
  - buffers packets, as needed, for in-order delivery to upper layer
- sender:
  - maintains (conceptually) a timer for each unACKed pkt
    - timeout: retransmits single unACKed packet associated with timeout
  - maintains (conceptually) "window" over *N* consecutive seq #s
    - limits pipelined, "in flight" packets to be within this window

京都大学

Selective repeat: sender, receiver windows



(a) sender view of sequence numbers

京都大学

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

京都大学

TCP: Transmission Control Protocol

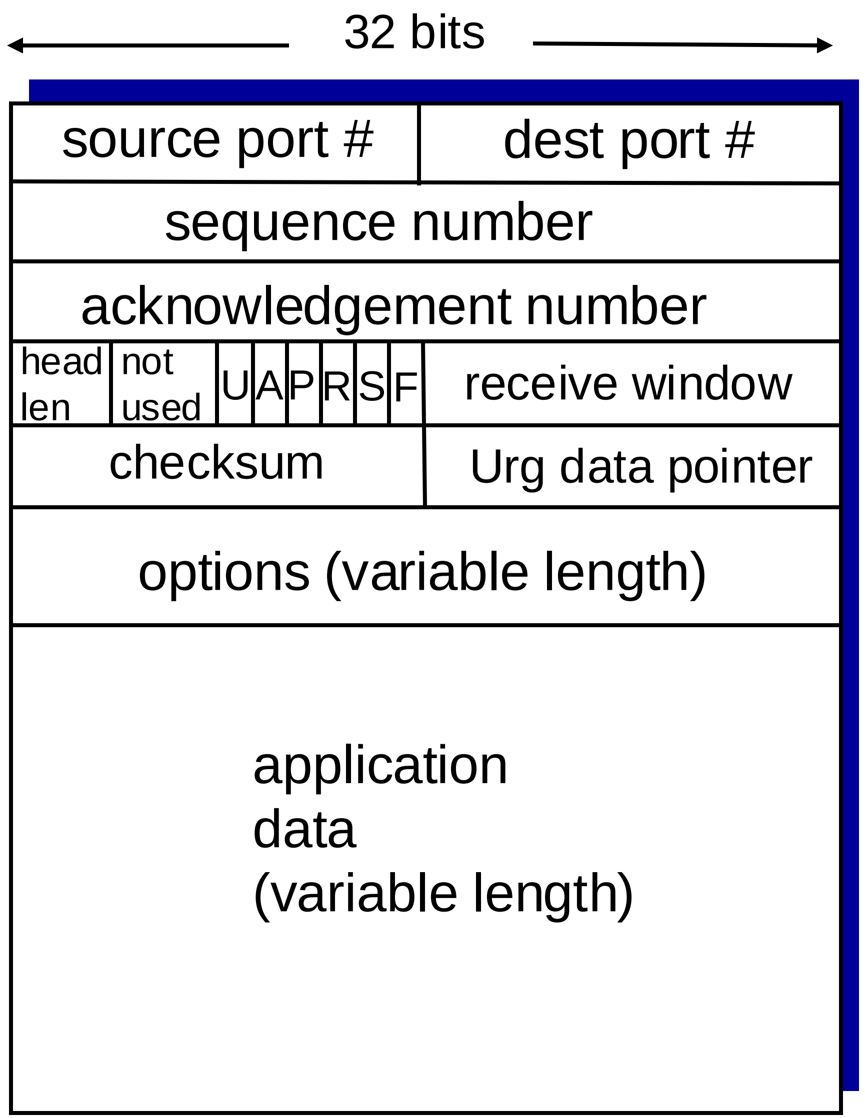
```
RFC: 793




                     TRANSMISSION CONTROL PROTOCOL


                        DARPA INTERNET PROGRAM

                        PROTOCOL SPECIFICATION



                            September 1981



                           1.  INTRODUCTION

The Transmission Control Protocol (TCP) is intended for use as a highly
reliable host-to-host protocol between hosts in packet-switched computer
communication networks, and in interconnected systems of such networks.

This document describes the functions to be performed by the
Transmission Control Protocol, the program that implements it, and its
interface to programs or users that require its services.
```

TCP: Overview

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte stream:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- full duplex data:
  - bi-directional data flow in same connection
- connection-oriented:
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

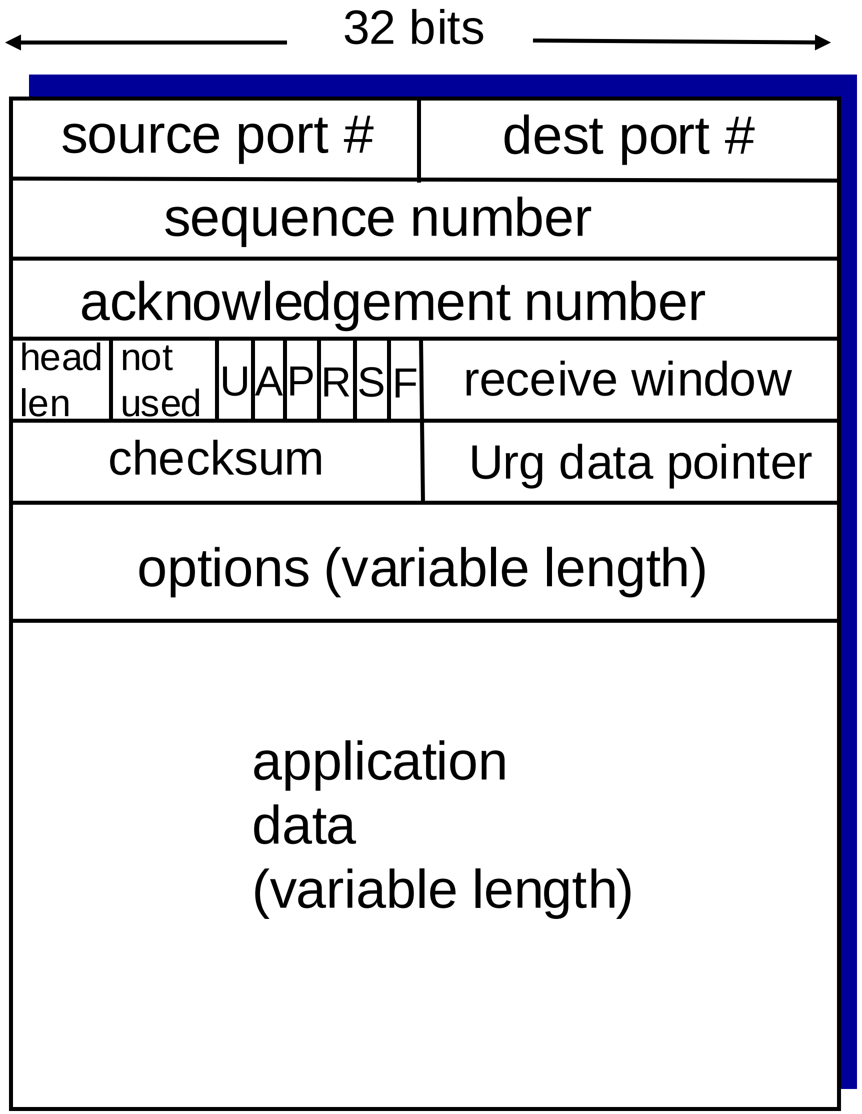
京都大学

TCP segment structure



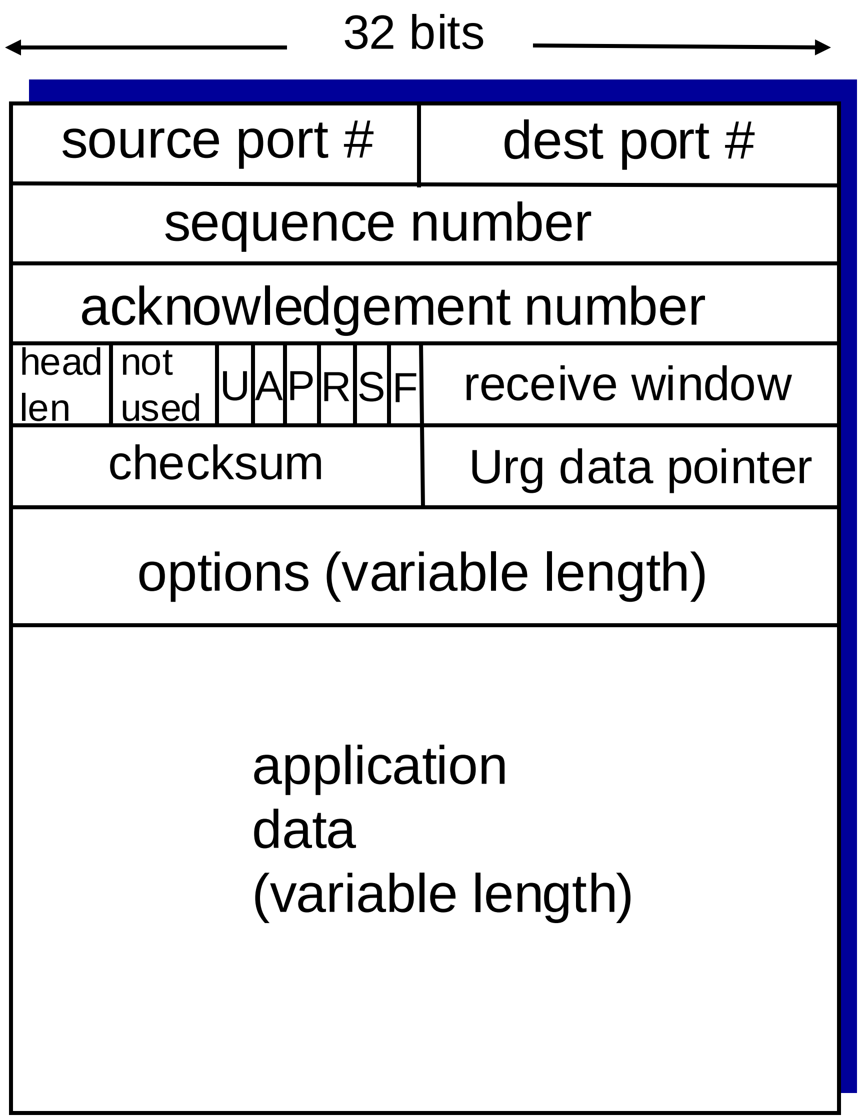
TCP Segment Header

- Source Port, Destination Port
- Sequence Number
  - At the transport layer application data is split into several smaller segments.
  - Sequence Number is used to keep track of the position of the current segment in the sequence.
- Acknowledgment Number: Number of the next expected segment
  - Used for reliable data transfer.



TCP Segment Header

- head len: Length of Header
  - For the receiver to know where the header ends and the application data begins
- 6 TCP control flags
- receive window
  - Number of bytes the receiver is willing to accept at a time
- Checksum
  - Same as in UDP
- Urg data pointer
  - Point out segments that are urgent
  - Rarely used
- Options
  - Additional options such as more complicated flow control
  - rarely used



京都大学

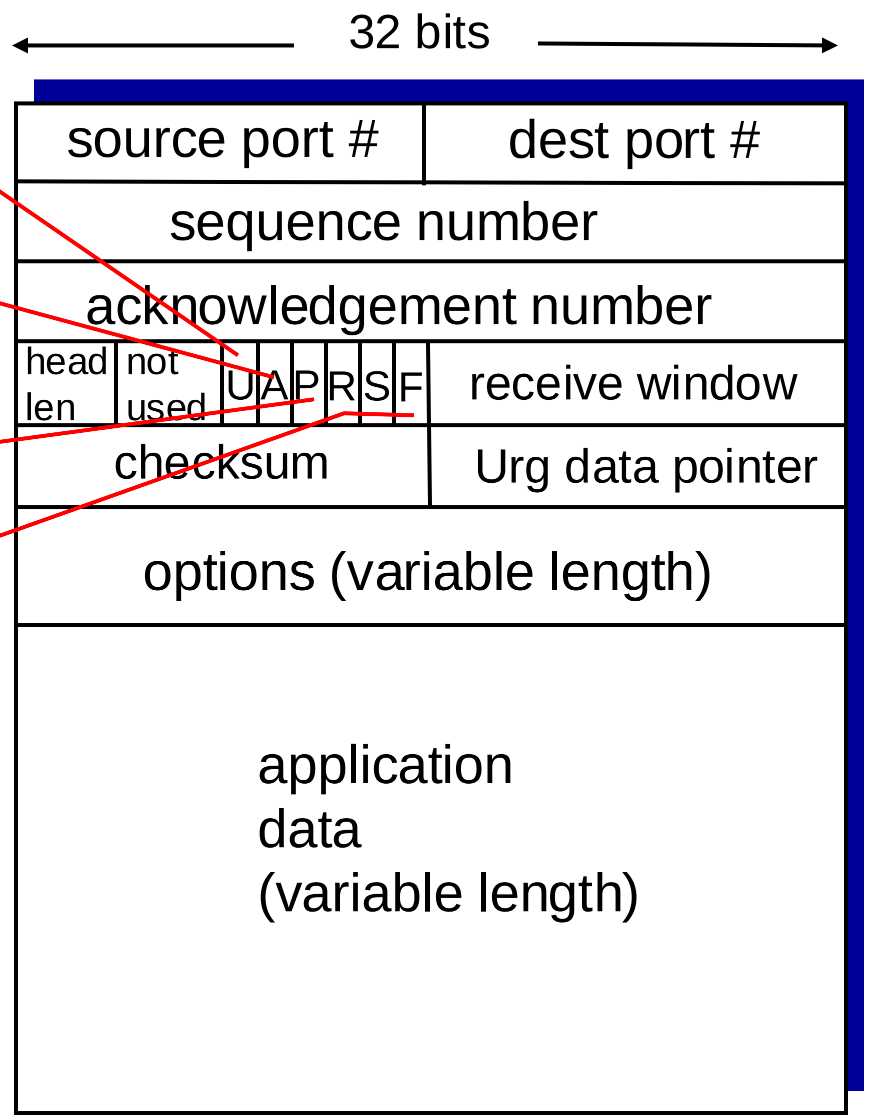
TCP control flags

URG: urgent data
(generally not used)

ACK: ACK #
valid

PSH: push data now
(generally not used)

RST, SYN, FIN:
connection estab
(setup, teardown
commands)



京都大学

TCP seq. numbers, ACKs

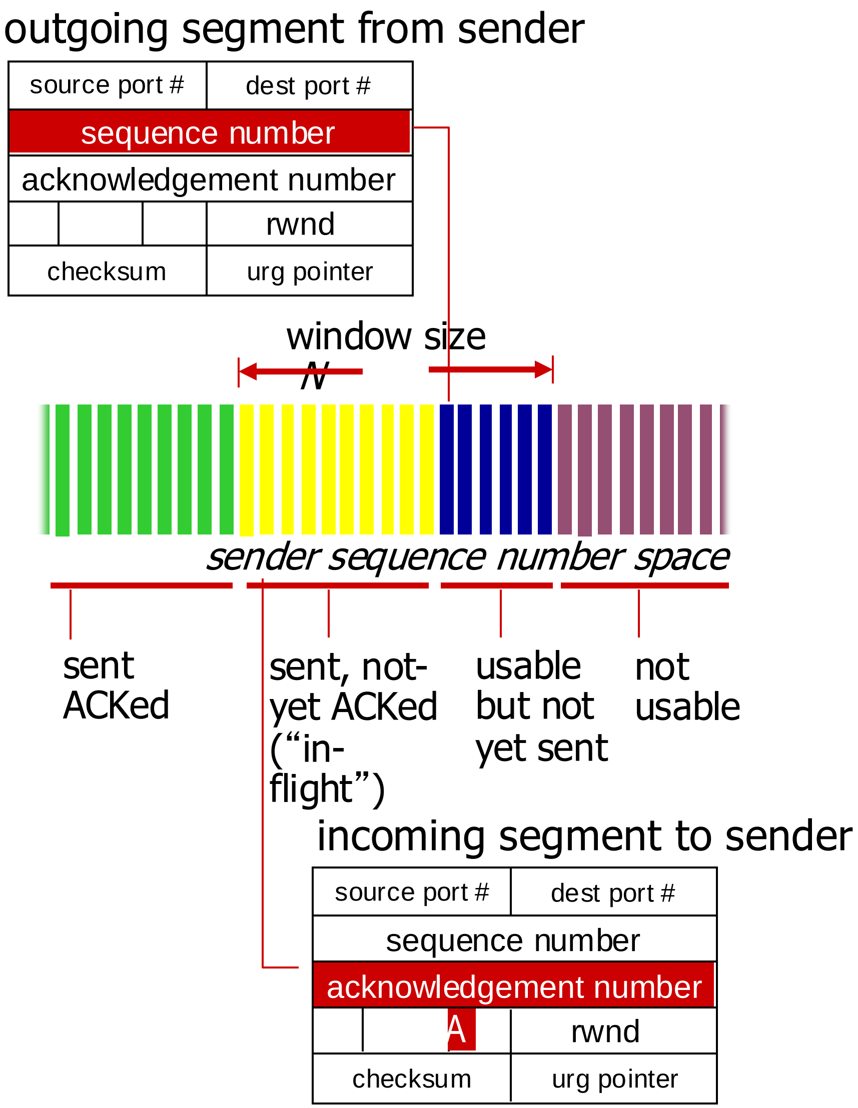
sequence numbers:

- byte stream "number" of first byte in segment's data

acknowledgements:

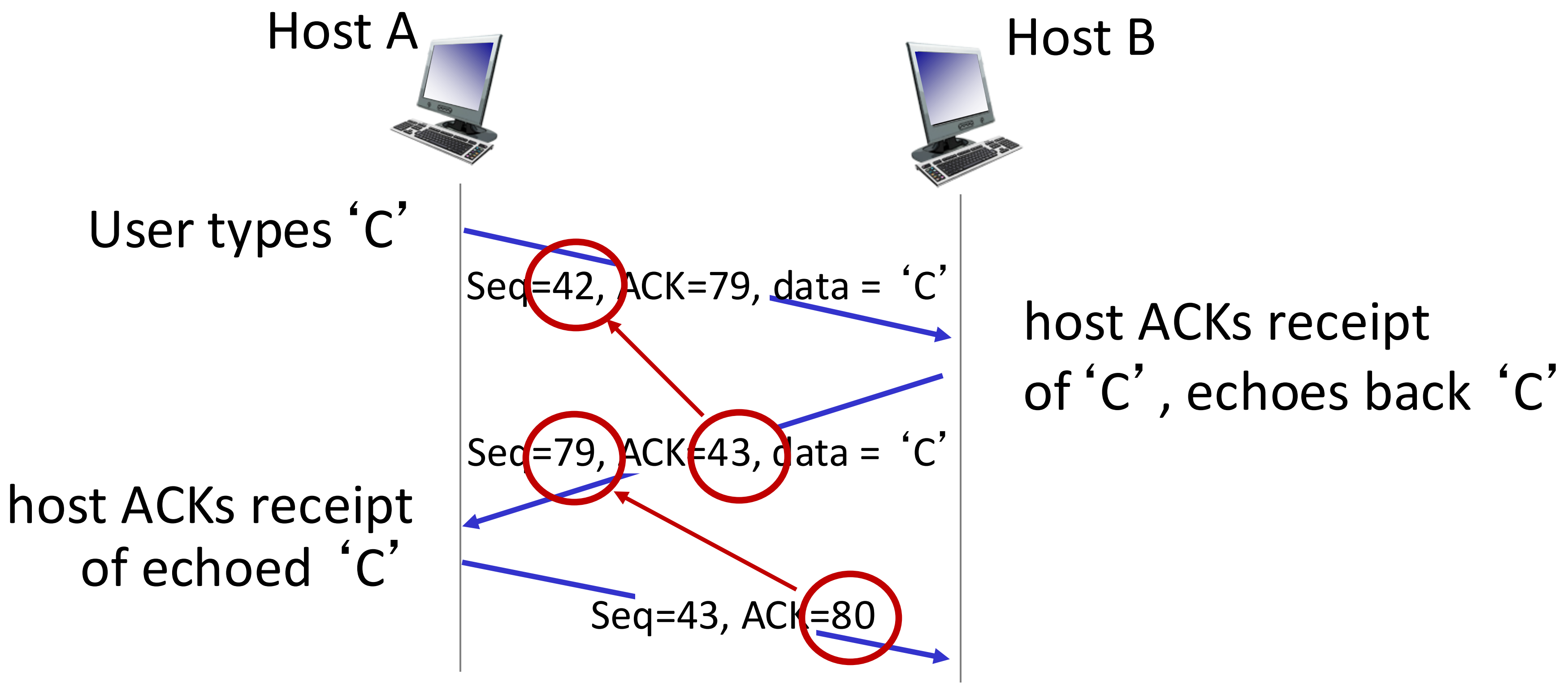
- Sequence number of next byte expected from other side
- cumulative ACK

- TCP does not specify how the receiver handles out-of-order segments, it is up to implementor



京都大学

TCP sequence numbers, ACKs



simple telnet scenario

京都大学

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

TCP reliable data transfer

```
Reliability:

  The TCP must recover from data that is damaged, lost, duplicated, or
  delivered out of order by the internet communication system.  This
  is achieved by assigning a sequence number to each octet
  transmitted, and requiring a positive acknowledgment (ACK) from the
  receiving TCP.  If the ACK is not received within a timeout
  interval, the data is retransmitted.  At the receiver, the sequence
  numbers are used to correctly order segments that may be received
  out of order and to eliminate duplicates.  Damage is handled by
  adding a checksum to each segment transmitted, checking it at the
  receiver, and discarding damaged segments.

  As long as the TCPs continue to function properly and the internet
  system does not become completely partitioned, no transmission
  errors will affect the users.  TCP recovers from internet
  communication system errors.
```

TCP reliable data transfer

- TCP creates reliable data transfer service on top of IP's unreliable service
  - pipelined segments
  - cumulative acks
  - single retransmission timer
- retransmissions triggered by:
  - timeout events
  - duplicate acks

京都大学

TCP Sender (simplified)

event: data received from application

- create segment with seq number
- seq number is byte-stream number of first data byte in segment
- start timer if not already running
  - think of timer as for oldest unACKed segment
  - expiration interval: **`TimeOutInterval`**

event: timeout

- retransmit segment that caused timeout
- restart timer

event: ACK received

- if ACK acknowledges previously unACKed segments
  - update what is known to be ACKed
  - start timer if there are still unACKed segments

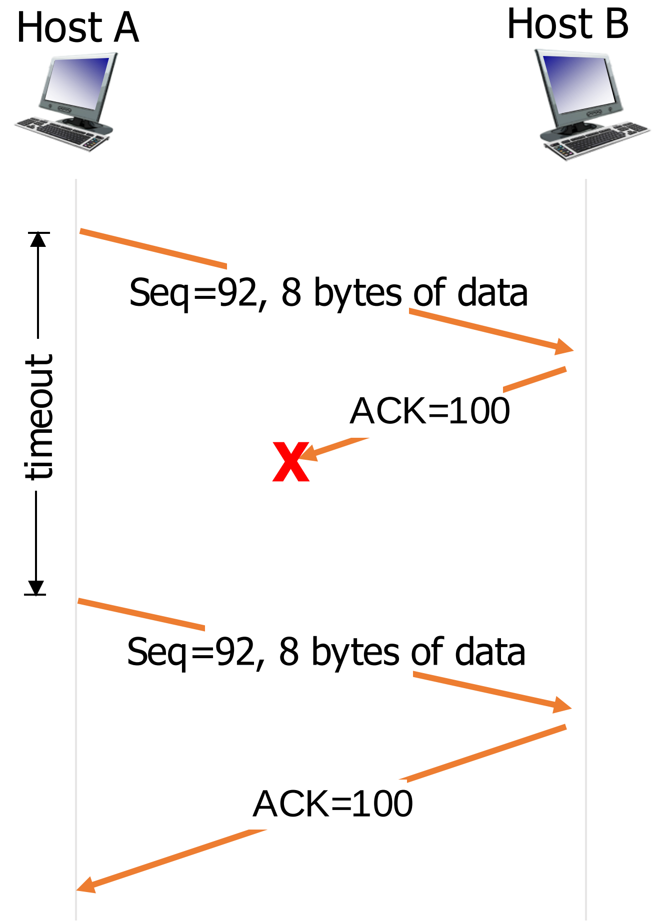
京都大学

TCP Receiver: ACK generation [RFC 5681]

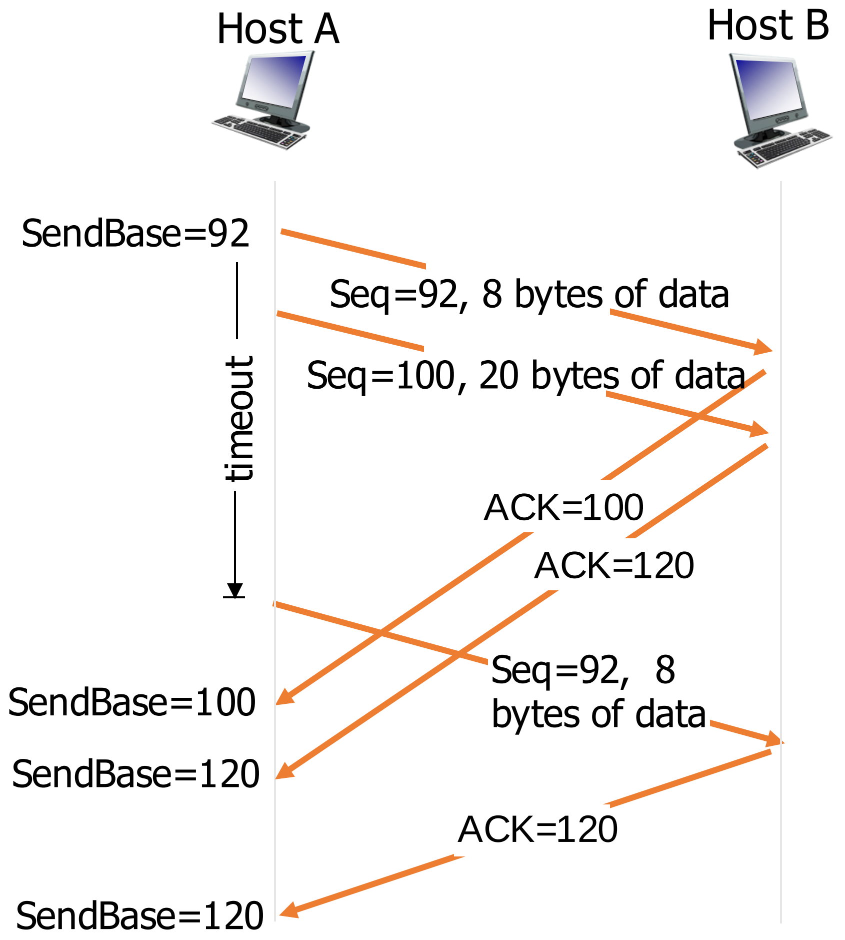
Event at receiver	*TCP receiver action*

京都大学

TCP: retransmission scenarios



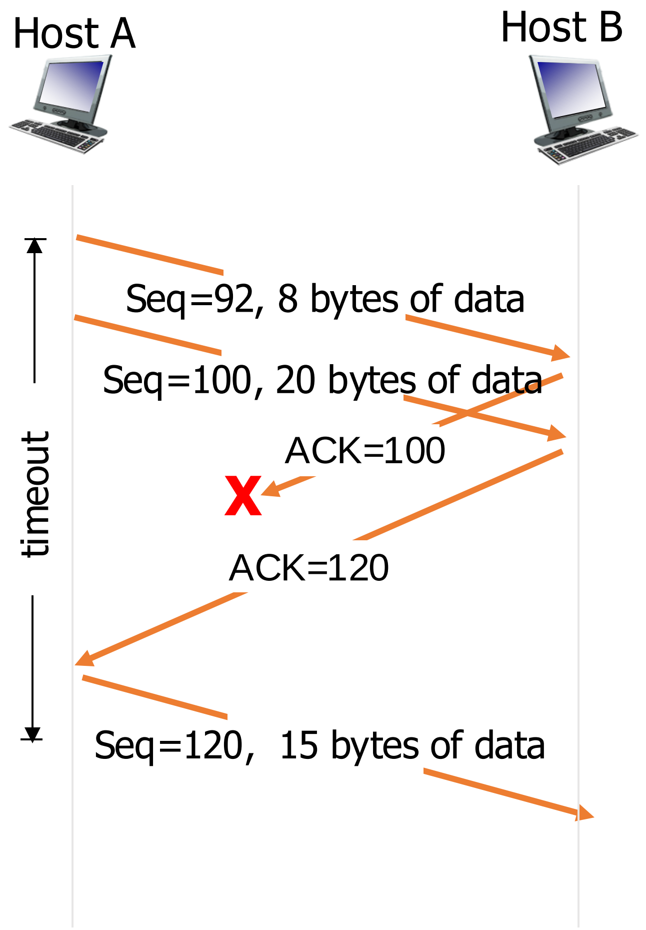
lost ACK scenario



premature timeout

京都大学

TCP: retransmission scenarios



cumulative ACK

京都大学

TCP fast retransmit

- time-out period often relatively long:
  - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs.

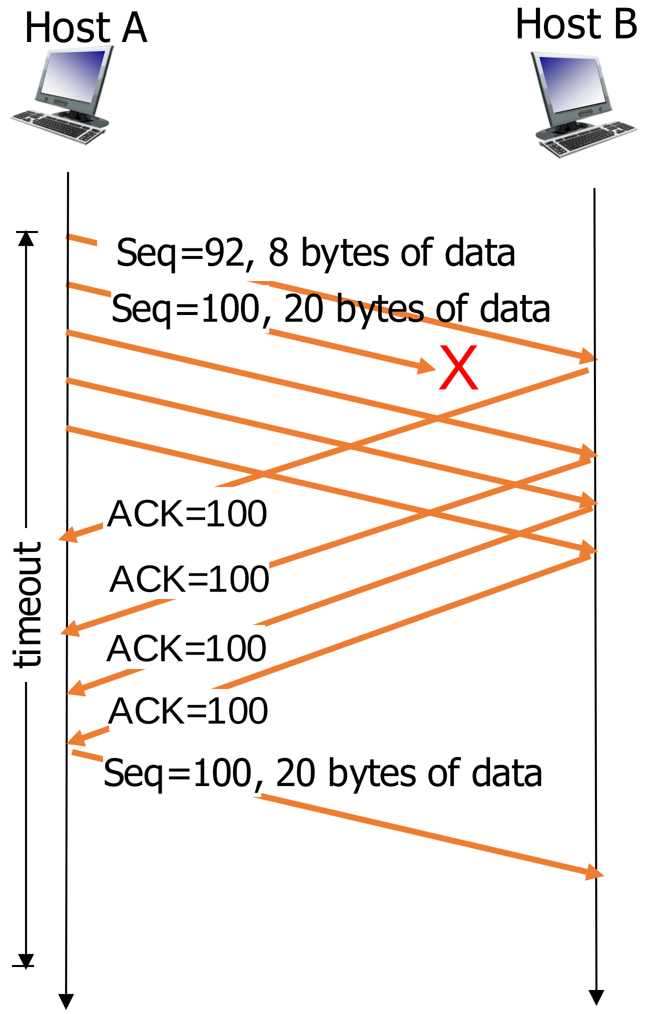
TCP fast retransmit

if sender receives 3 ACKs for same data ("triple duplicate ACKs"), resend unacked segment with smallest seq number

- likely that unacked segment lost, so don't wait for timeout

京都大学

TCP fast retransmit



fast retransmit after sender receipt of triple duplicate ACK

京都大学

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

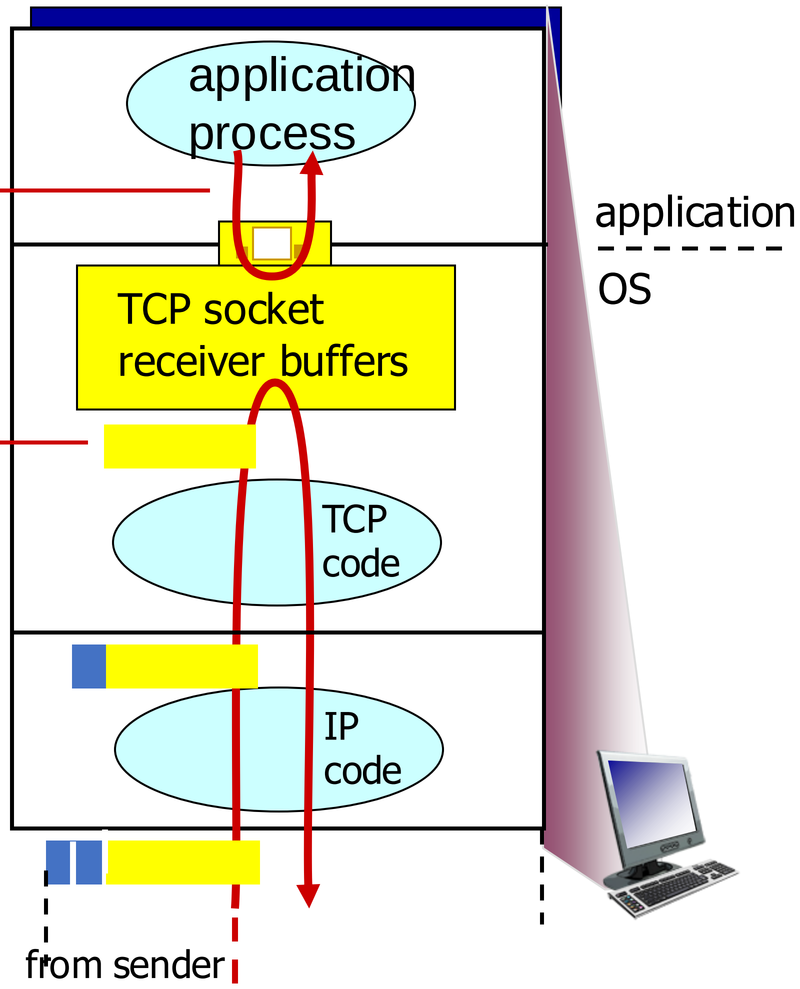
3.7 TCP congestion control

京都大学

TCP flow control

application may remove data from TCP socket buffers ….

… slower than TCP receiver is delivering (sender is sending)



receiver protocol stack

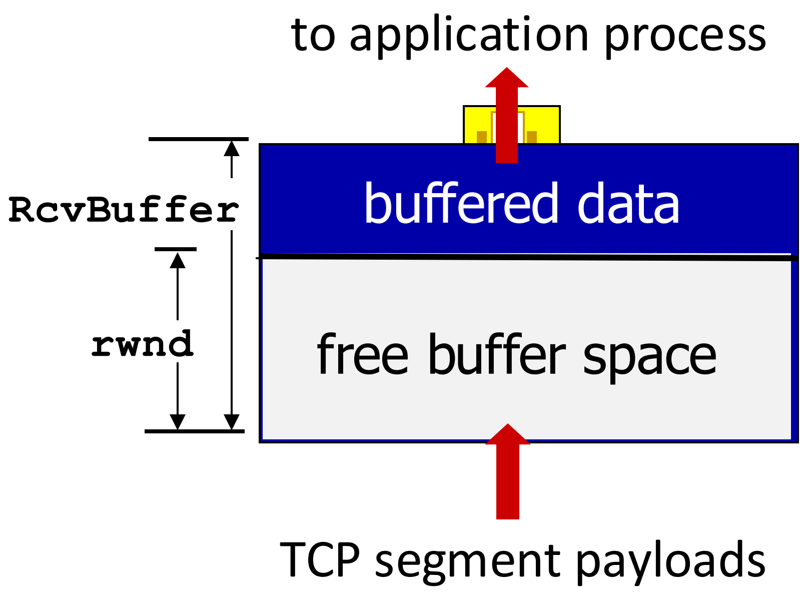
flow control

receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

京都大学

TCP flow control

- TCP receiver "advertises" free buffer space in **rwnd** field in TCP header
  - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **RcvBuffer**
- sender limits amount of unACKed ("in-flight") data to received **rwnd**
- guarantees receive buffer will not overflow



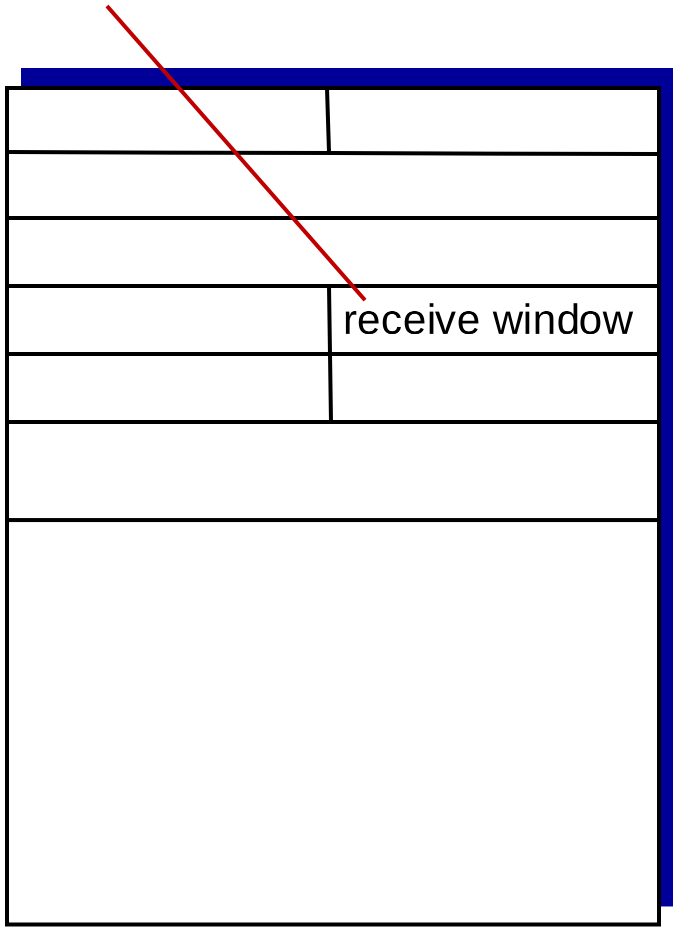
TCP receiver-side buffering

京都大学

TCP flow control

- TCP receiver "advertises" free buffer space in **`rwnd`** field in TCP header
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **`RcvBuffer`**
- sender limits amount of unACKed ("in-flight") data to received **`rwnd`**
- guarantees receive buffer will not overflow

flow control: # bytes receiver willing to accept



TCP segment format

京都大学

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

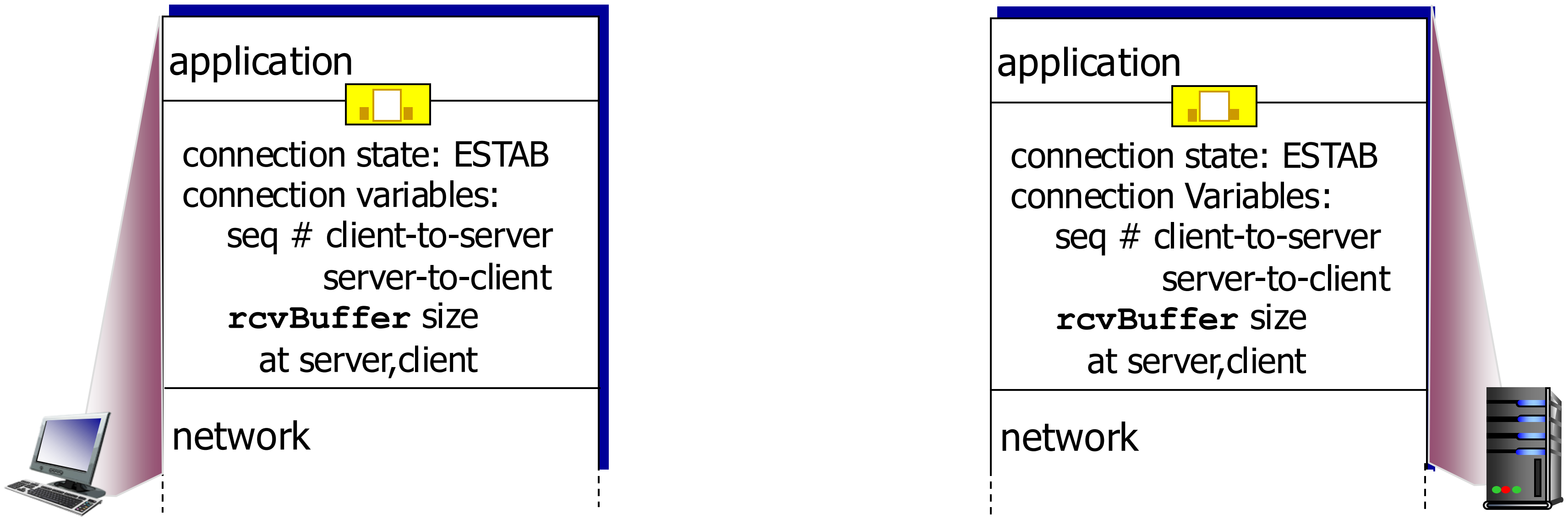
3.7 TCP congestion control

京都大学

Connection Management

before exchanging data, sender/receiver "handshake":

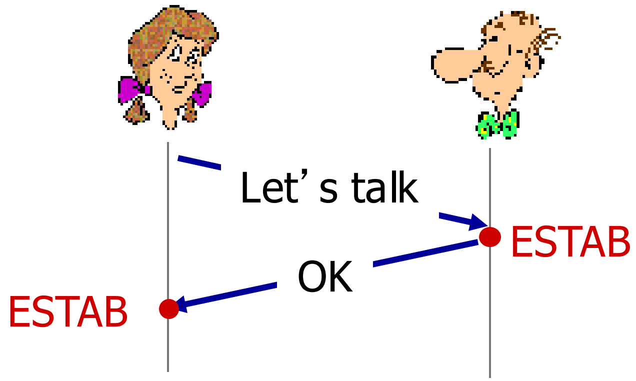
- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters

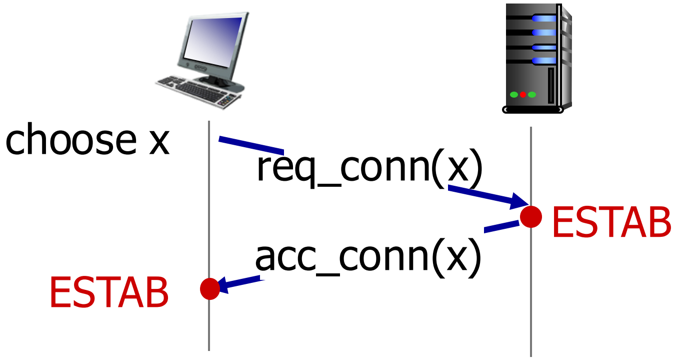


京都大学

Agreeing to establish a connection

2-way handshake:



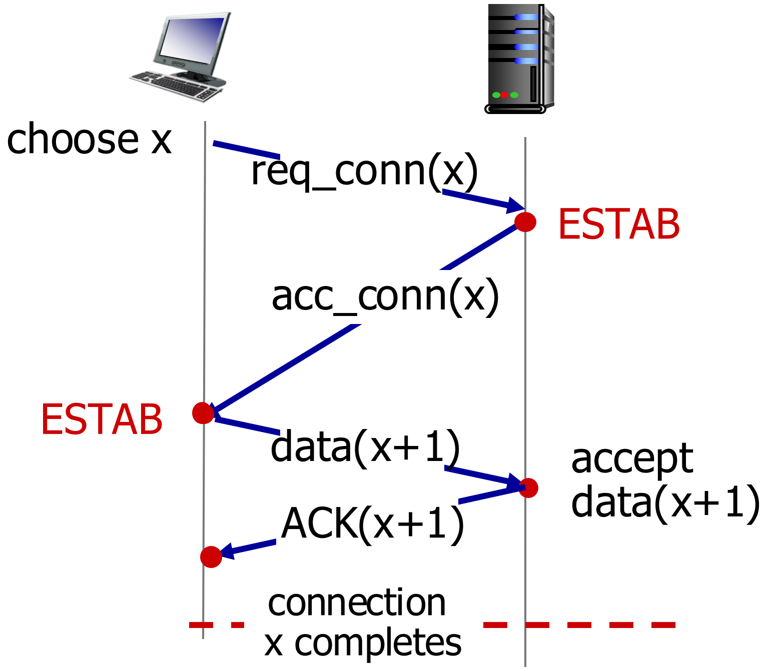


Q: will 2-way handshake always work in network?

- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

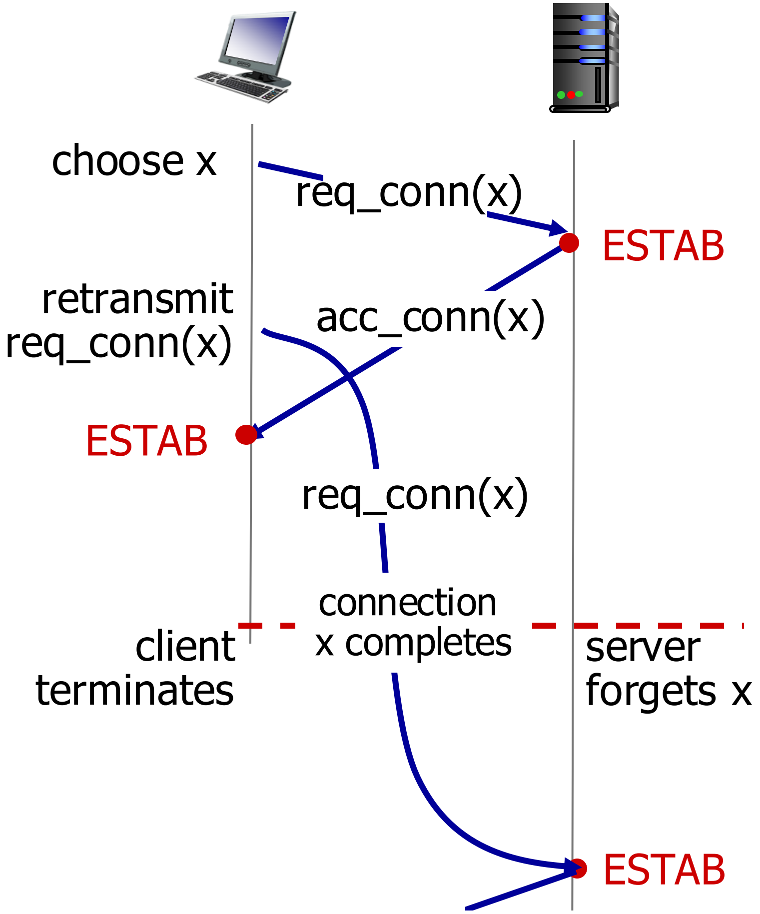
京都大学

2-way handshake scenarios



No problem!

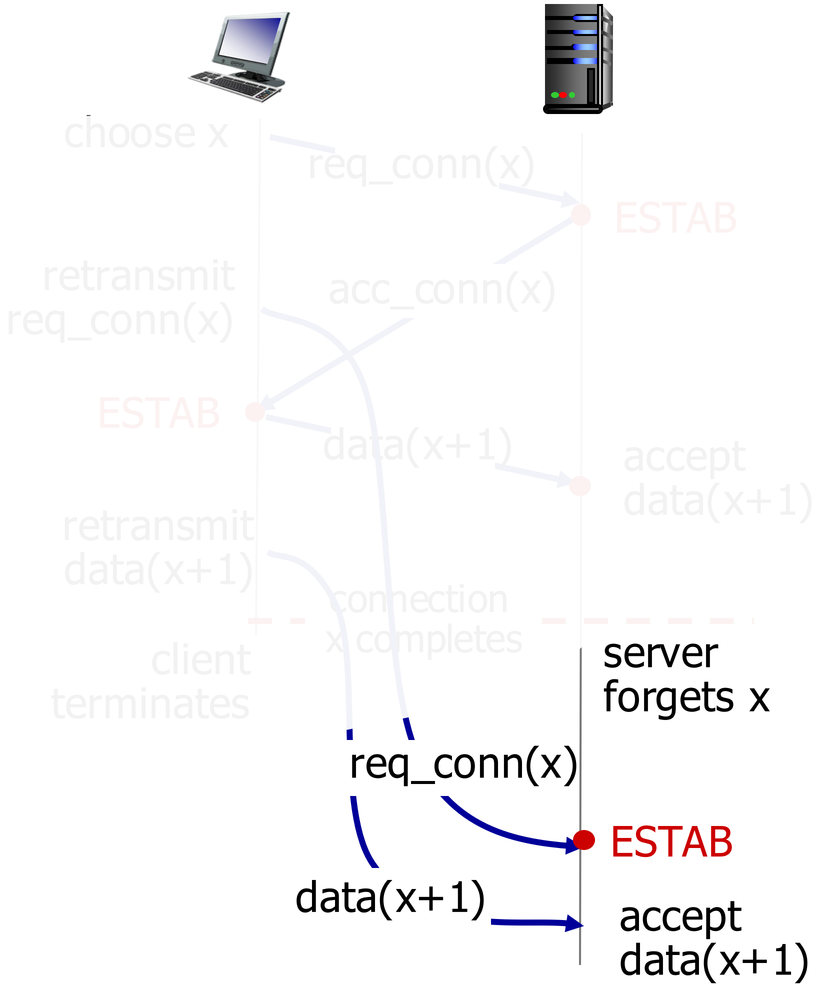
2-way handshake scenarios



Problem: half open connection! (no client)

京都大学

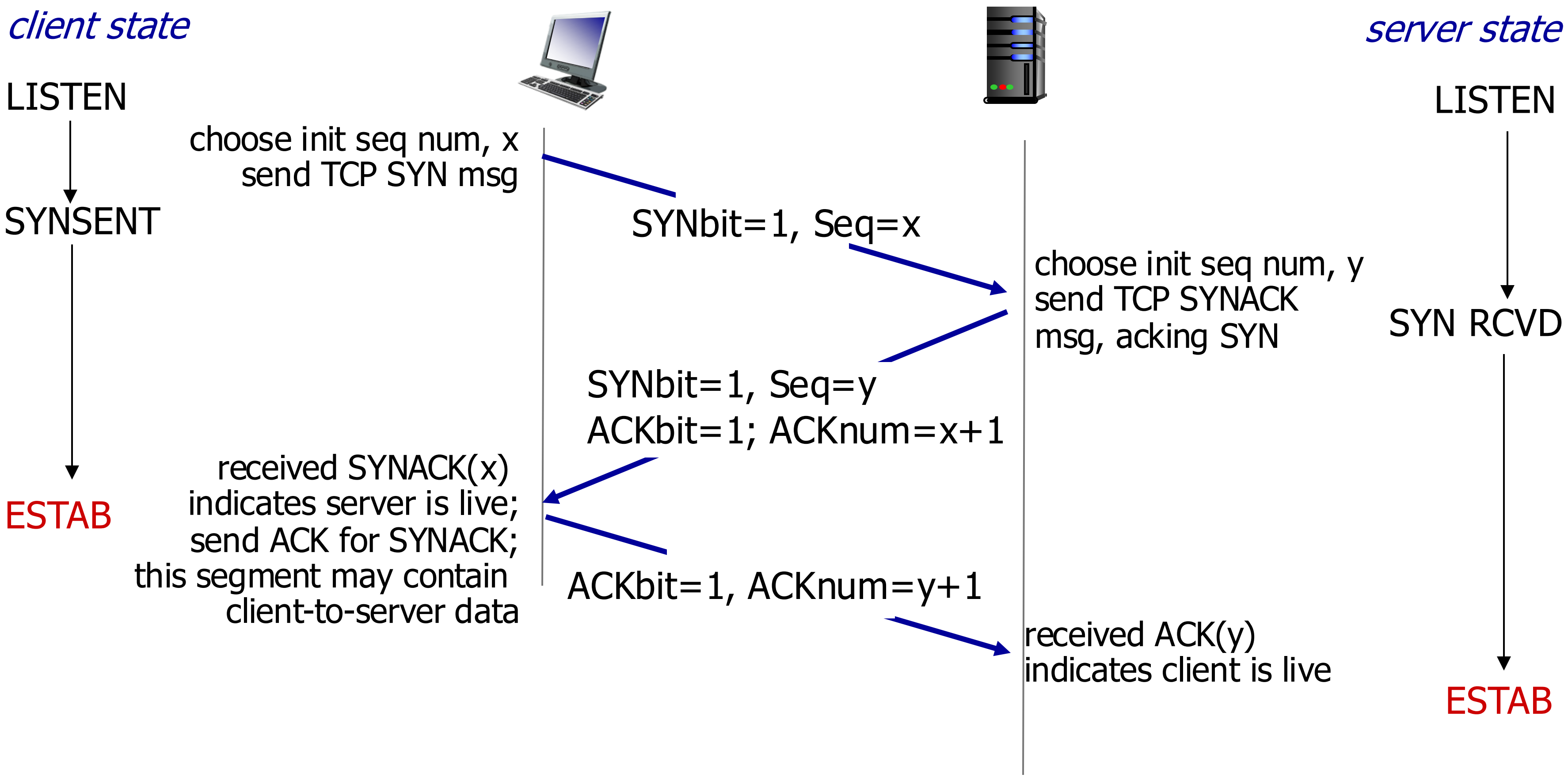
2-way handshake scenarios



Problem: duplicate data accepted!

京都大学

TCP 3-way handshake



京都大学

TCP: closing a connection

- client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN

京都大学

TCP: closing a connection