Information Network

Lecture 9: TCP

Holger Thies

KYOTO UNIVERSITY

京都大学

Reminder

- Class grade: Homework and Short Test (40%), Final report (60%)
- In-class short test in lecture 14 (1/22).
- The test must be taken in class.
- If you can not attend the class (for a good reason) then you must contact the lecturer <u>before the test</u> to schedule a replacement test.
  - Email: thies.holger.5c@kyoto-u.ac.jp
- If you do not attend class 14 without priorly contacting the lecturer, the test will be counted as 0 points.
- Final report needs to be submitted until 1/24.

Final report

The final report of this lecture must be submitted on PandA until the deadline 1/24 **in word or pdf format.**

Your report should be in a single document and address **both** of the two topics below:

1. Choose either the transport layer or the network layer of the Internet Protocol Stack
   - Summarize the main purpose of the layer and explain what services it can provide.
   - Describe some of the main protocols that operate on this layer in detail
   - Explain how the layer interacts with and uses protocols and services of the upper and lower layers.
2. Describe a network application or protocol (on any layer) **that was not covered in class**.
   - Summarize the protocol's purpose, functionality, and its role within the Internet Protocol Stack.
   - Make some suggestions on improving the selected network protocol's efficiency, security, or scalability.

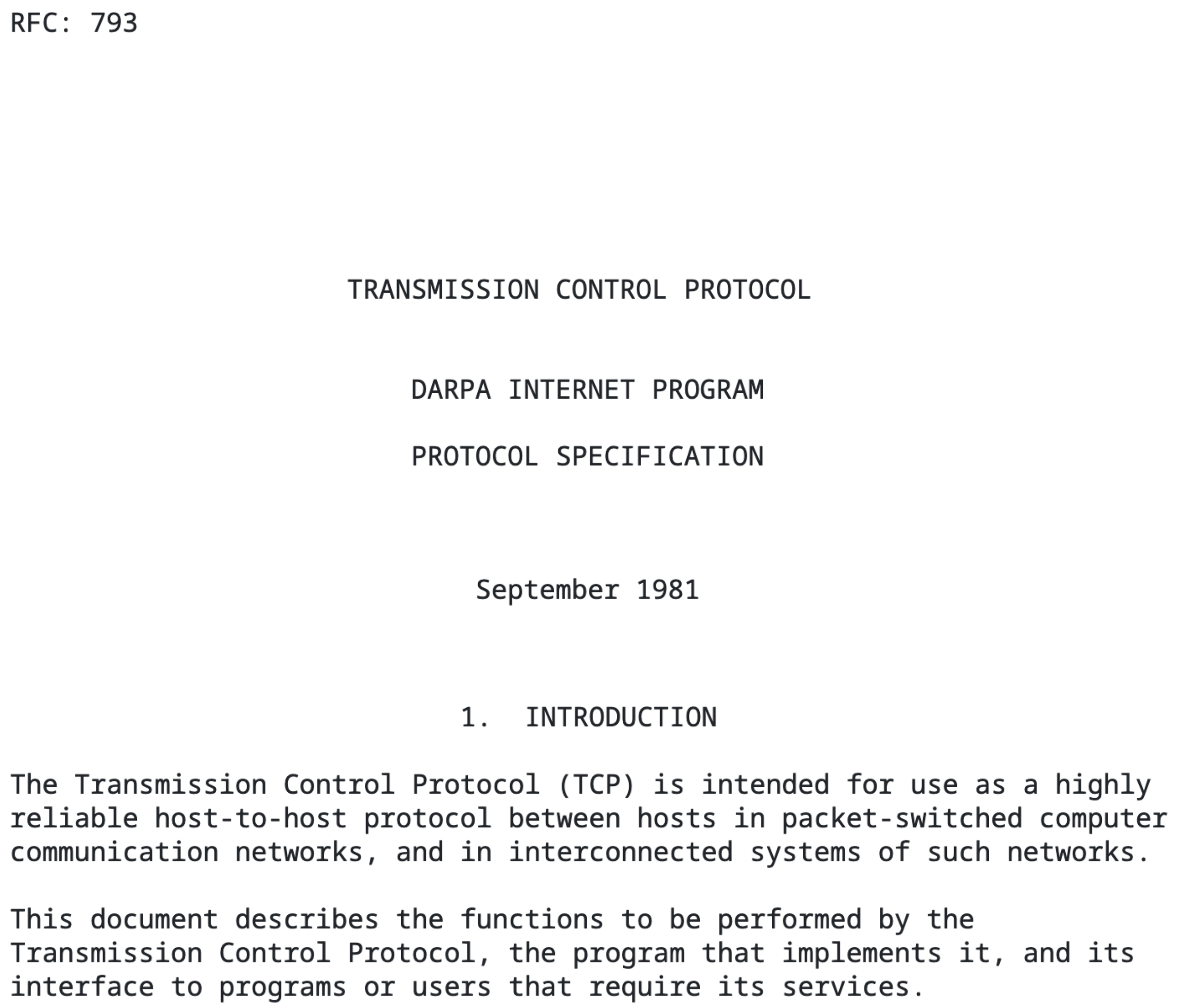
Final report

- Your report should be 5-7 pages long in total (including headings, references, figures, etc.) and has to be written in English.
- The font size of the main text should not be larger than 12pt.
- Otherwise, the format is free, as long as the report is written clearly and demonstrates understanding of the topic.
- Each student must write their own report.
- All external references must be cited properly.
- The deadline for submissions is January 24 (Friday).
- Late submissions are not allowed. Please submit your report early to avoid potential technical difficulties.
- **Any form of plagiarism, including the use of generative AI (ChatGPT etc.) is strictly prohibited.**
- **In case plagiarism is discovered there can be severe consequences such as automatically failing all courses in the academic year.**
    - See also Kyoto University Guideline for Liberal Arts Courses: https://www.z.k.kyoto-u.ac.jp/zenkyo/lesson

Chapter 3 outline

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.

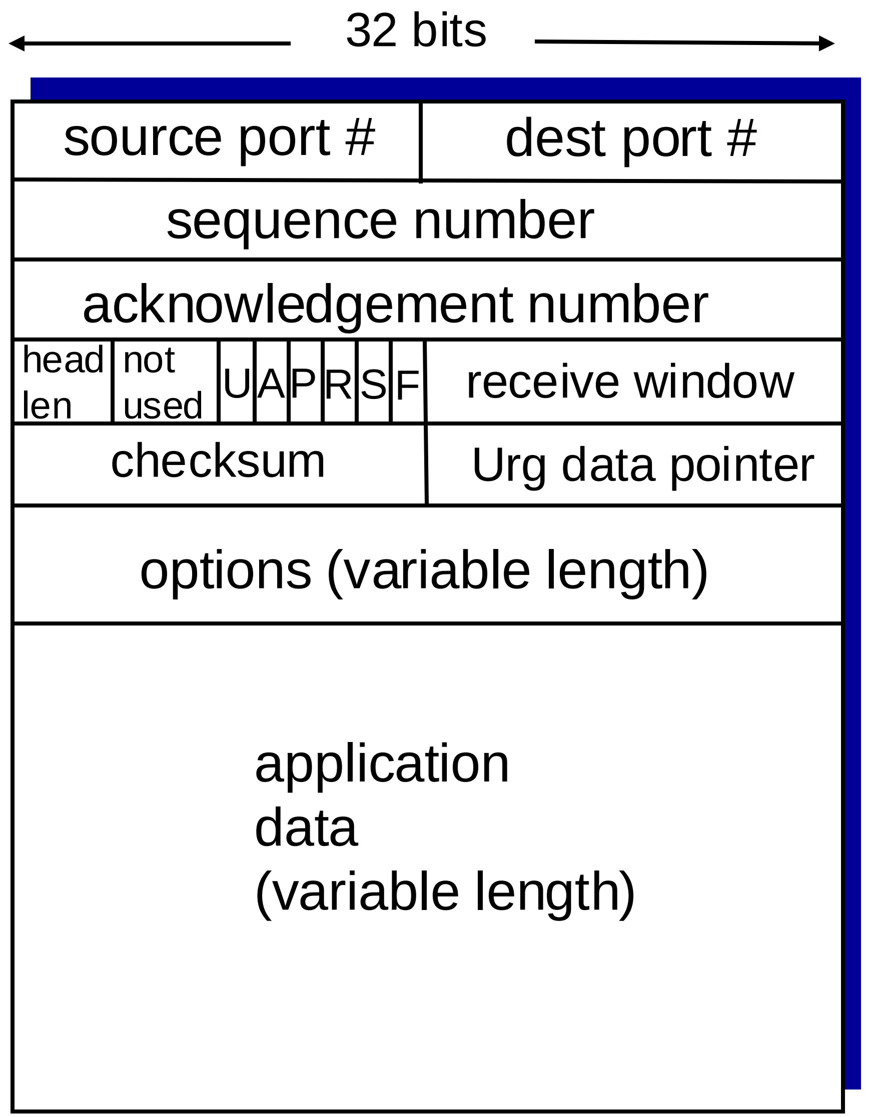
This document describes the functions to be performed by the Transmission Control Protocol, the program that implements it, and its interface to programs or users that require its services.

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 mesages) to initialize sender and receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

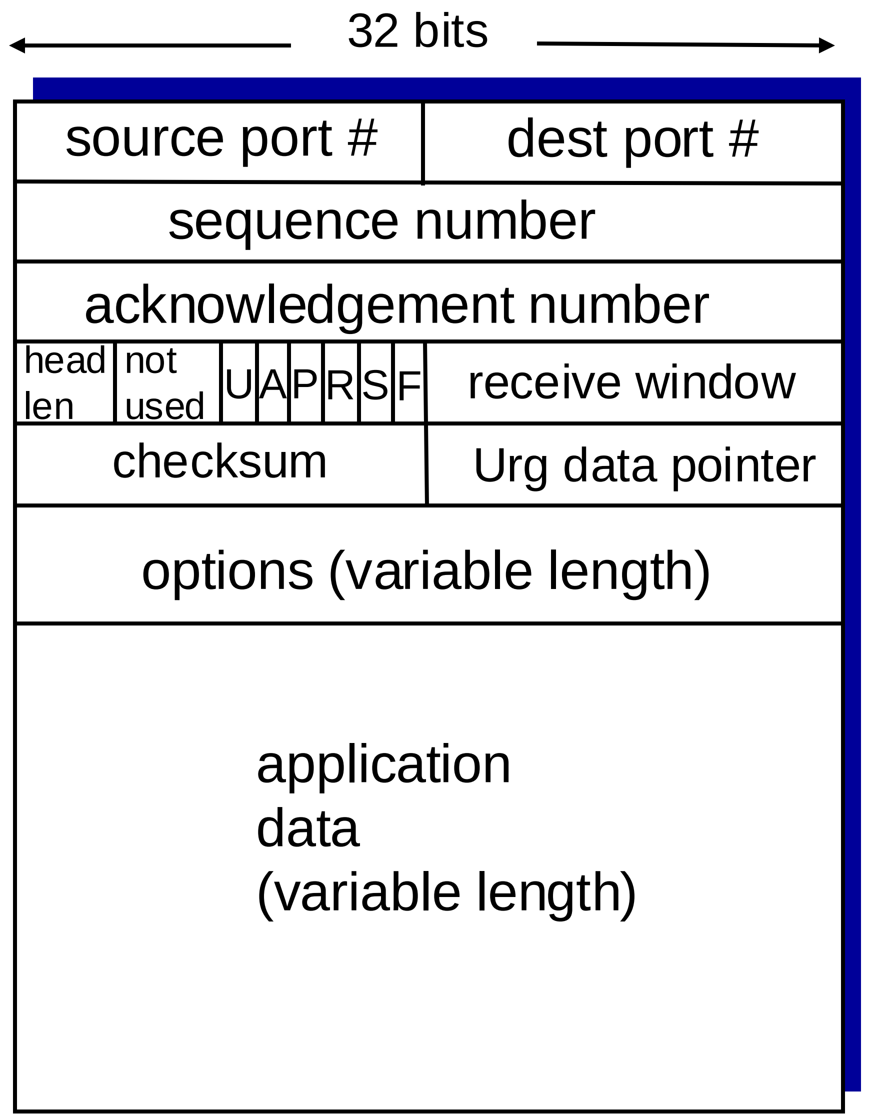
京都大学

TCP segment structure



TCP Segment Header

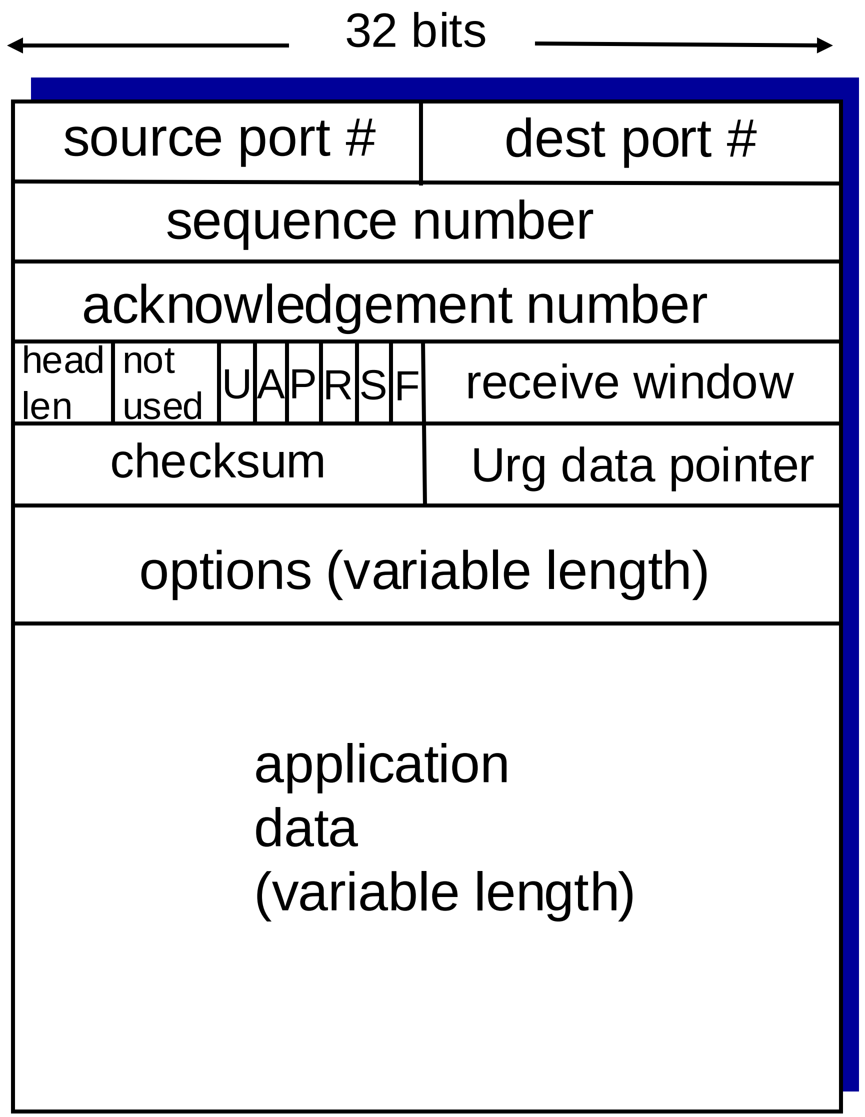
- Source Port, Destination Port
- Sequence Number
  - At the transport layer application data is split into several smaller segments.
  - Sequence Number is used to keep track of the position of the current segment in the sequence.
- Acknowledgment Number: Number of the next expected segment
  - Used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



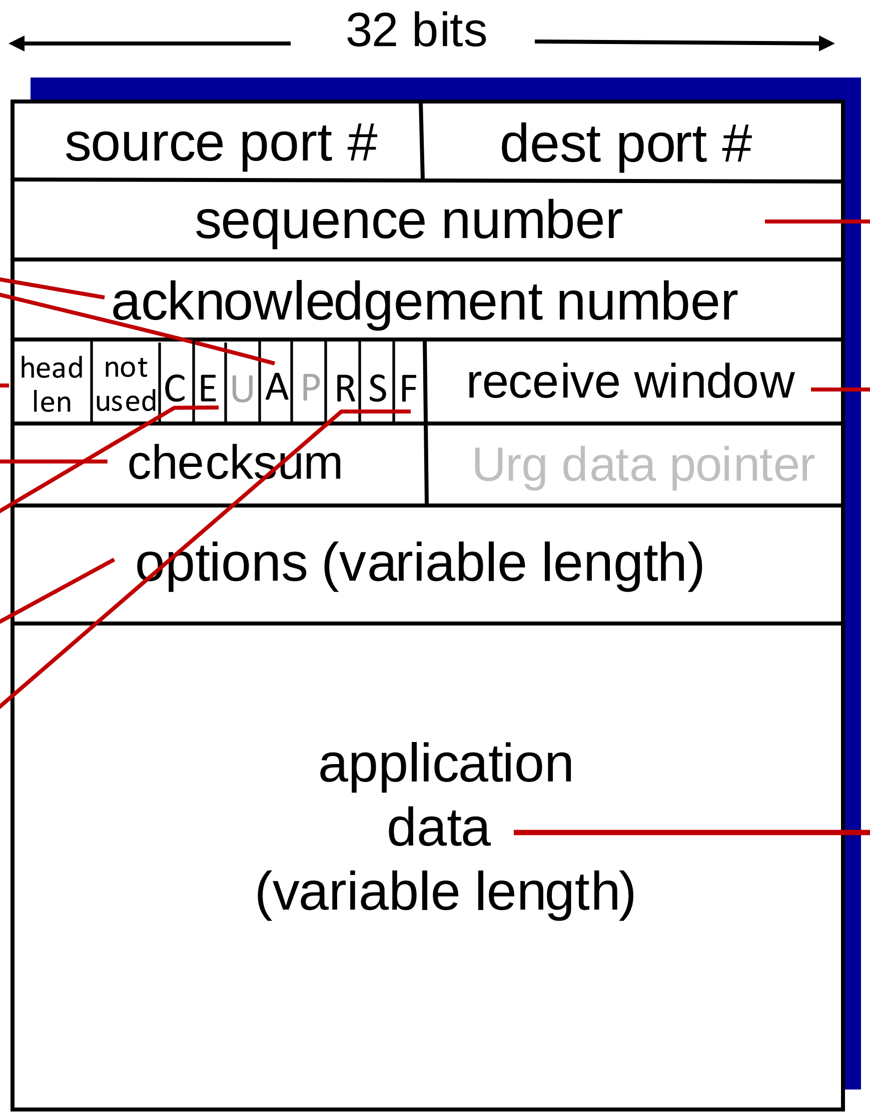
京都大学

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



TCP segment structure



ACK: seq # of next expected byte; A bit: this is an ACK

length (of TCP header)

Internet checksum

C, E: congestion notification

TCP options

RST, SYN, FIN: connection management

segment seq #: counting bytes of data into bytestream (not segments!)

flow control: # bytes receiver willing to accept

data sent by application into TCP socket

京都大学

TCP seq. numbers, ACKs

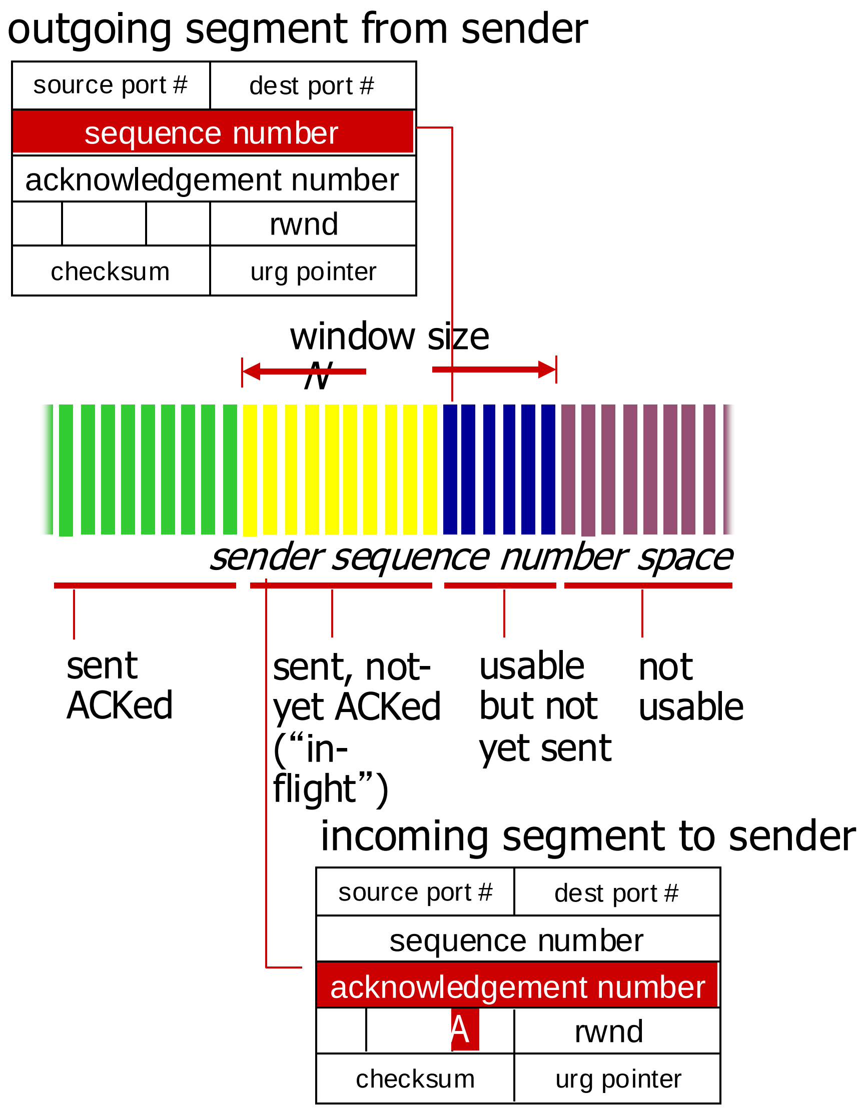
sequence numbers:

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

acknowledgements:

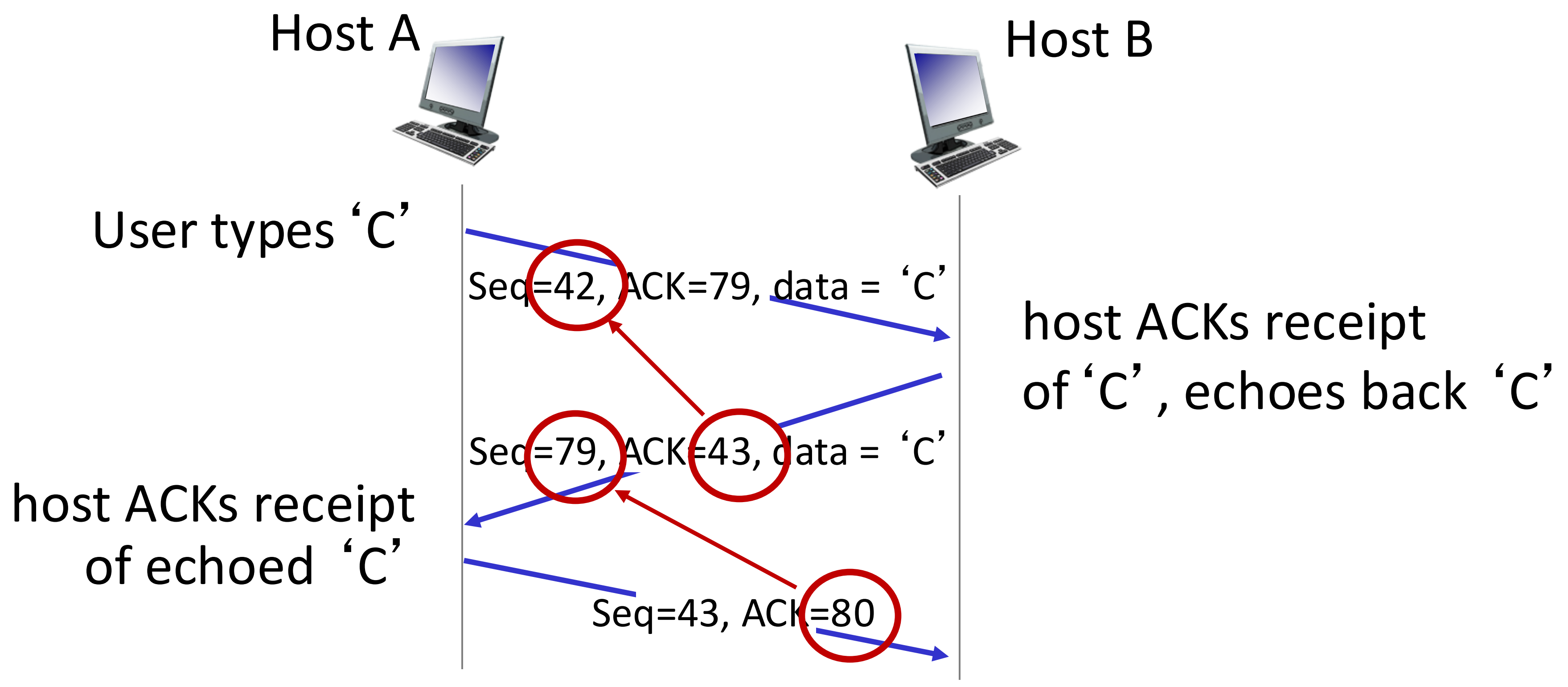
- Sequence number of next byte expected from other side
- cumulative acknowledgments

- 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


Host A
Host B
User types ‘C’
Seq=42, ACK=79, data = ‘C’
host ACKs receipt of ‘C’, echoes back ‘C’
Seq=79, ACK=43, data = ‘C’
host ACKs receipt of echoed ‘C’
Seq=43, ACK=80

simple scenario

Another example

京都大学

Chapter 3 outline

京都大学

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

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
  - Timer is for oldest unacknowledged segment

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

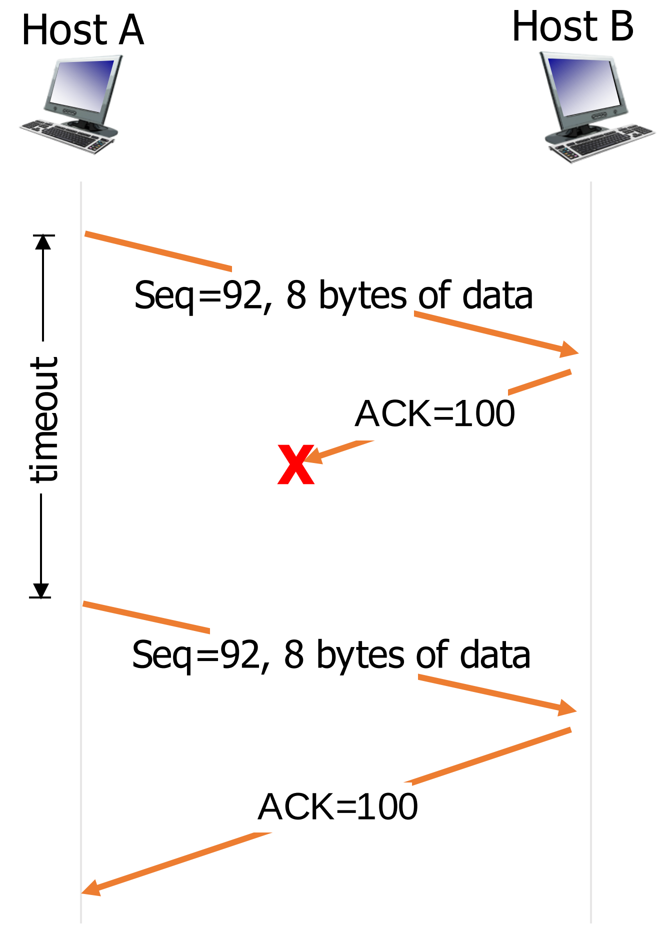
京都大学

TCP Receiver: ACK generation [RFC 5681]

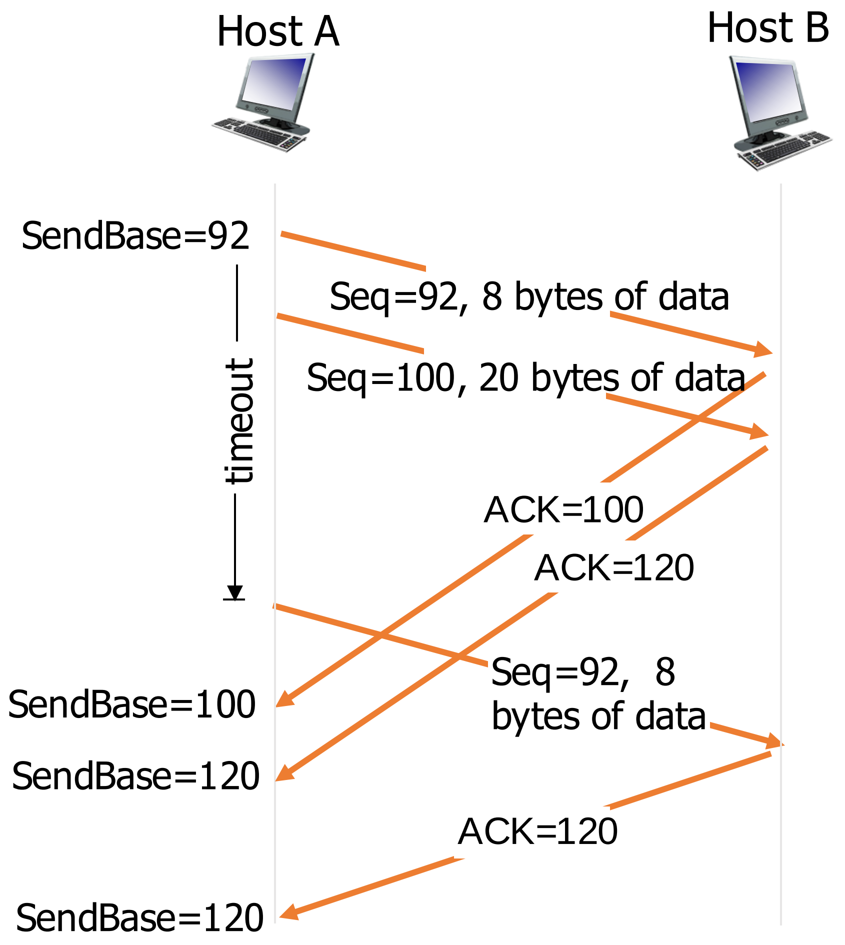
Event at receiver	*TCP receiver action*

京都大学

TCP: retransmission scenarios



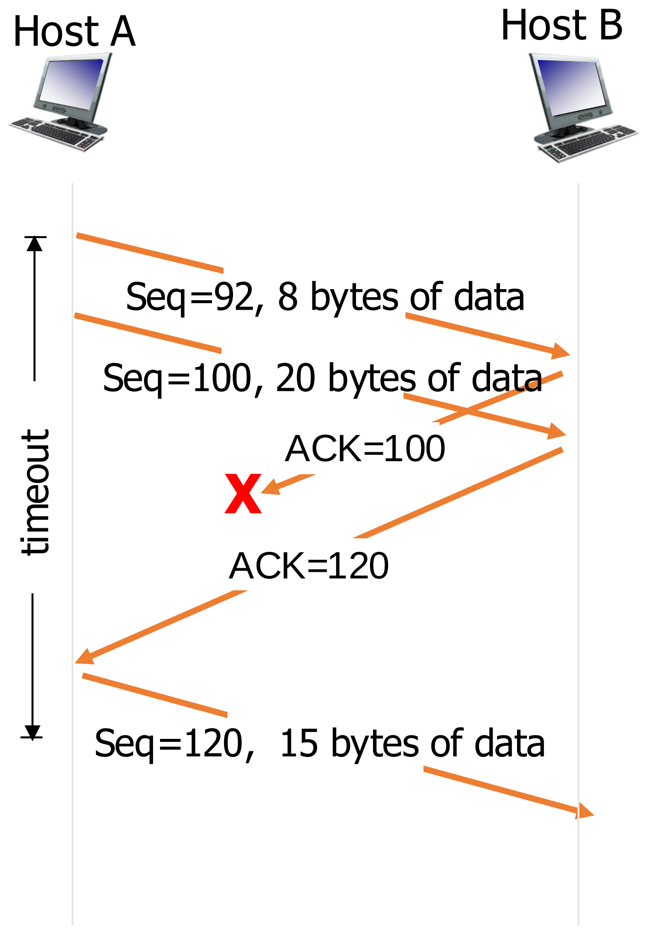
lost ACK scenario



premature timeout

京都大学

TCP: retransmission scenarios



cumulative ACK

TCP Example

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.

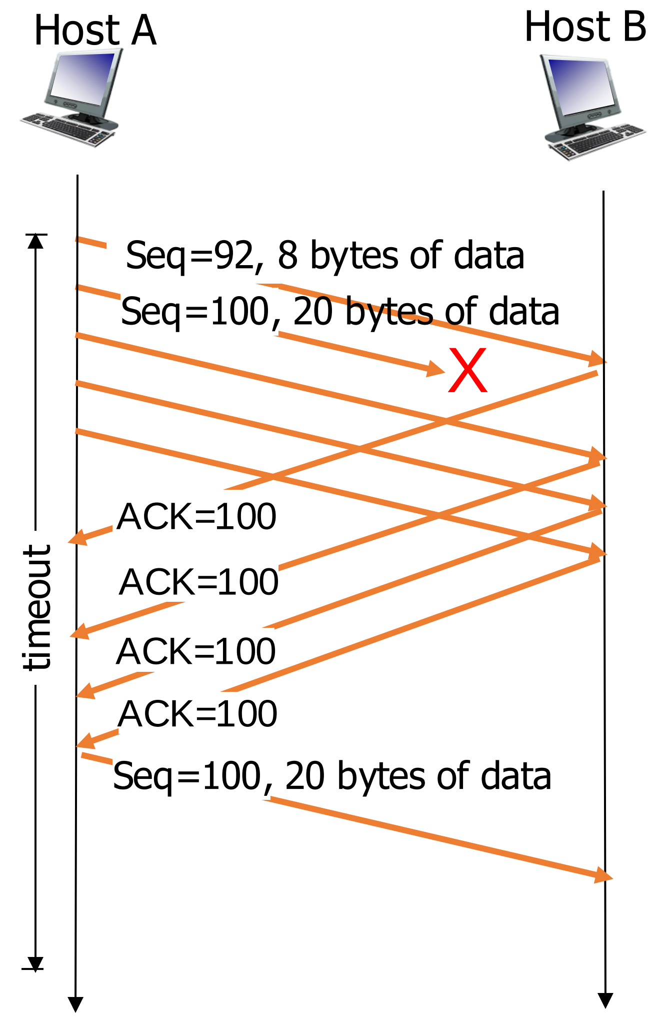
TCP fast retransmit

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

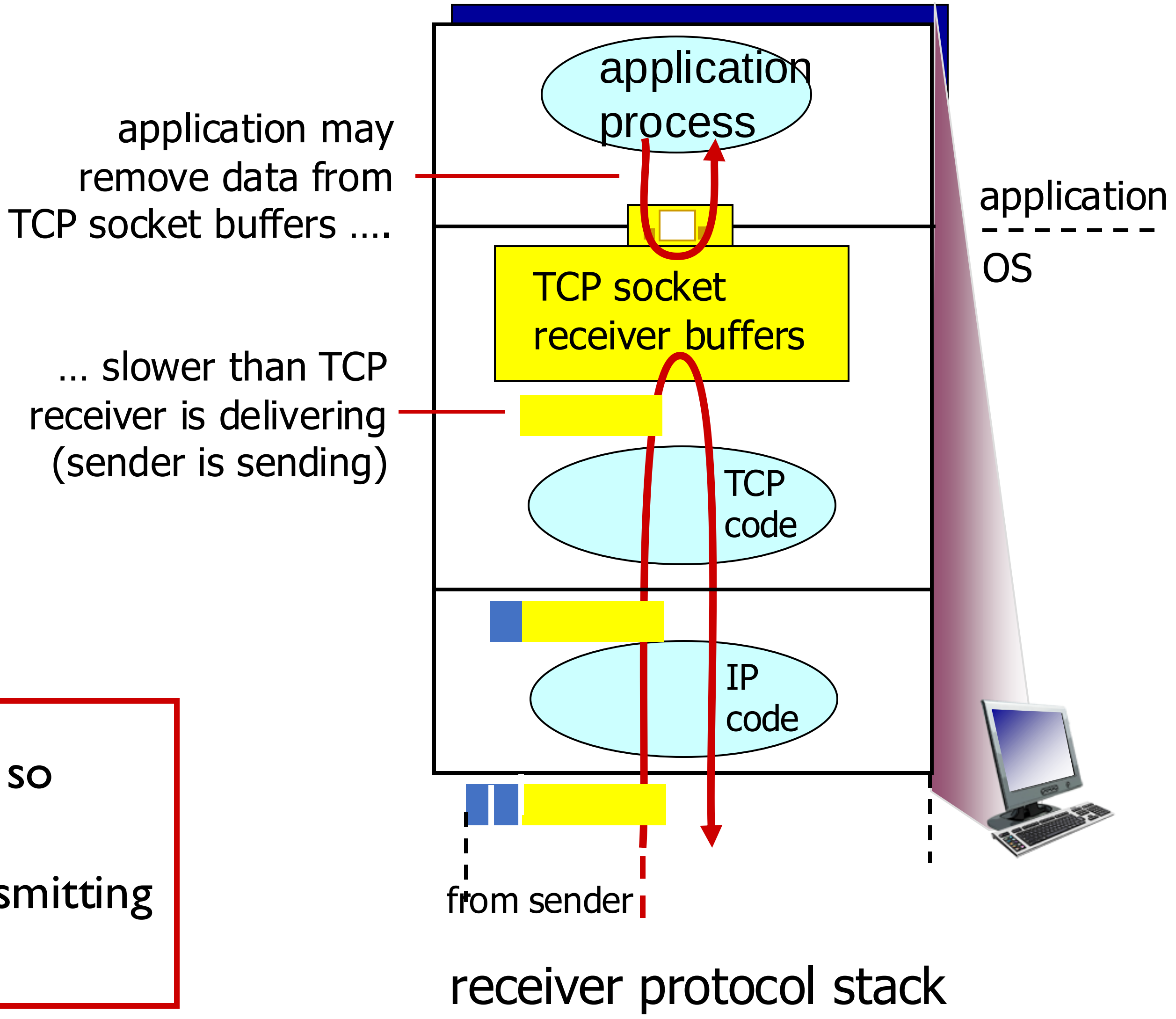
Chapter 3 outline

京都大学

Flow control and congestion control

- Both flow control and congestion control are about slowing down the sender, when more data is sent than can be handled.
- **Flow control** is about slowing down the sender when more data is sent than **the receiver** can handle.
- **Congestion control** is about slowing down the sender when more data is sent than **the network** can handle.
- In general, congestion control is more complex than flow control as it needs to operate across the entire network, involving multiple devices etc.

TCP flow control

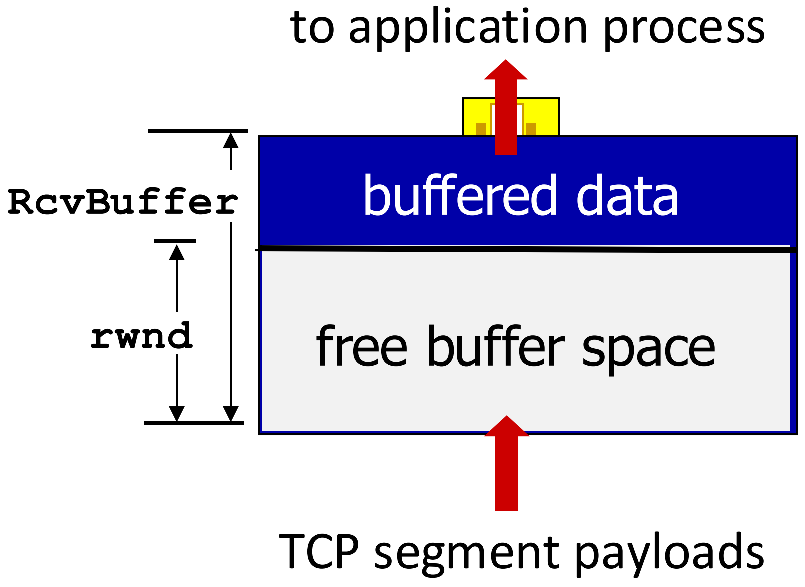

application process
application may remove data from TCP socket buffers ….
TCP socket receiver buffers
… slower than TCP receiver is delivering (sender is sending)
TCP code
IP code
application
OS
from sender
receiver protocol stack

flow control

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

TCP flow control

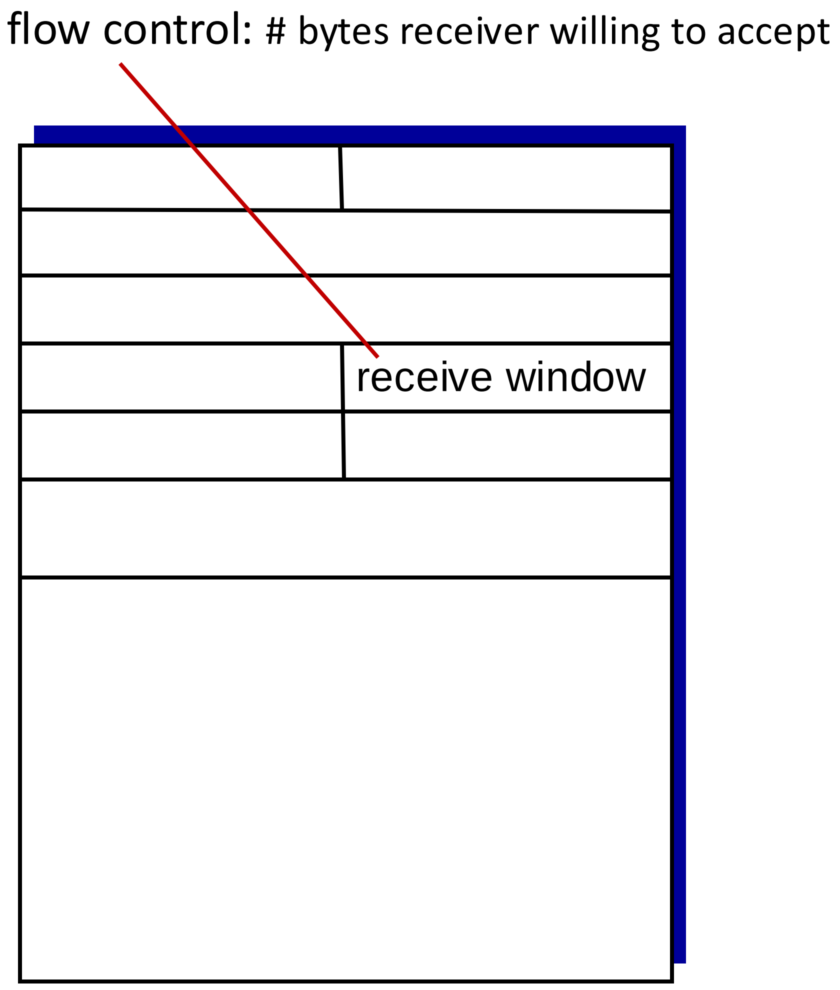
- TCP receiver "advertises" free buffer space in **`rwnd`** field in TCP header
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **`RcvBuffer`**
- sender limits amount of unacknowledged ("in-flight") data to received **`rwnd`**
- guarantees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



TCP receiver-side buffering

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 segment format

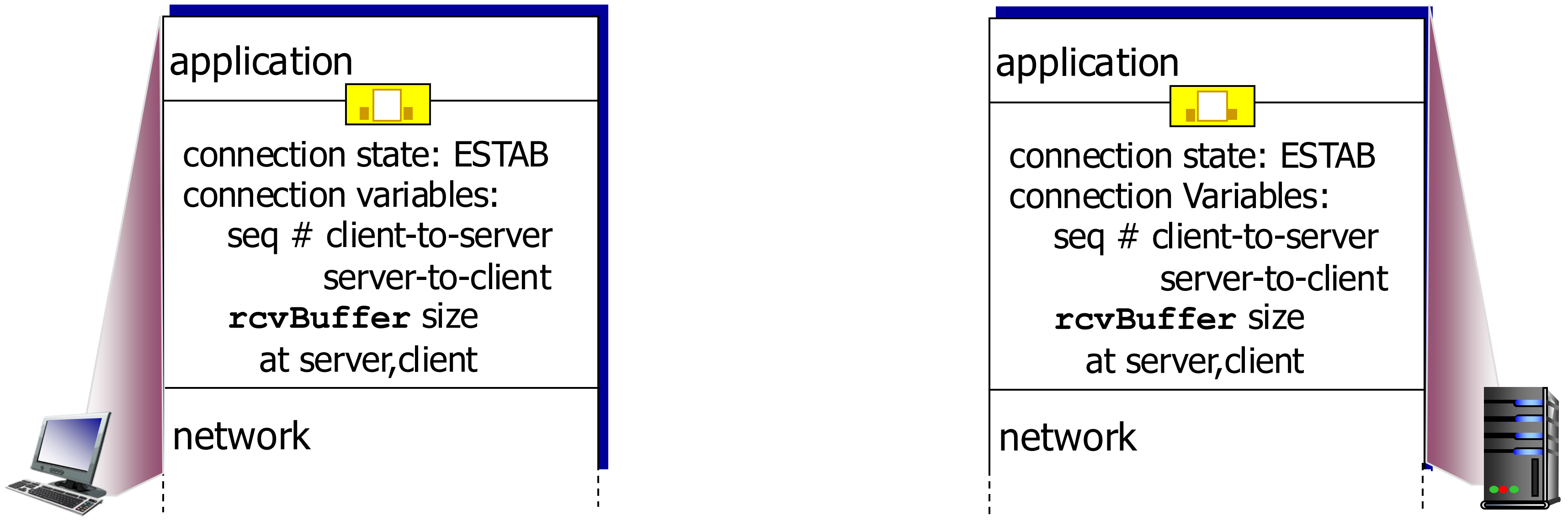
Chapter 3 outline

京都大学

Connection Management

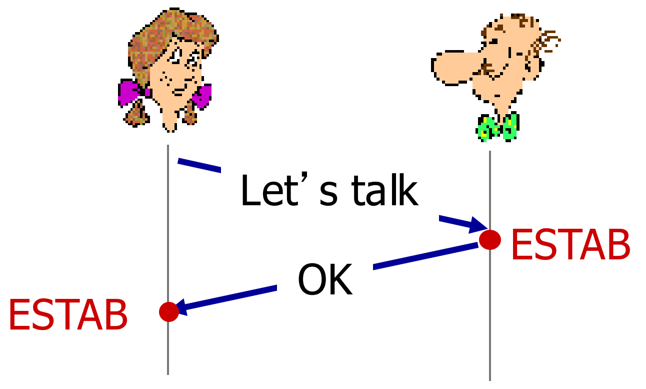
before exchanging data, sender/receiver "handshake":

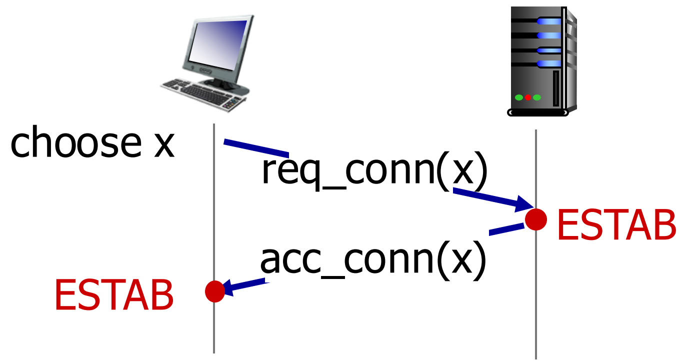
- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters



Agreeing to establish a connection

2-way handshake:



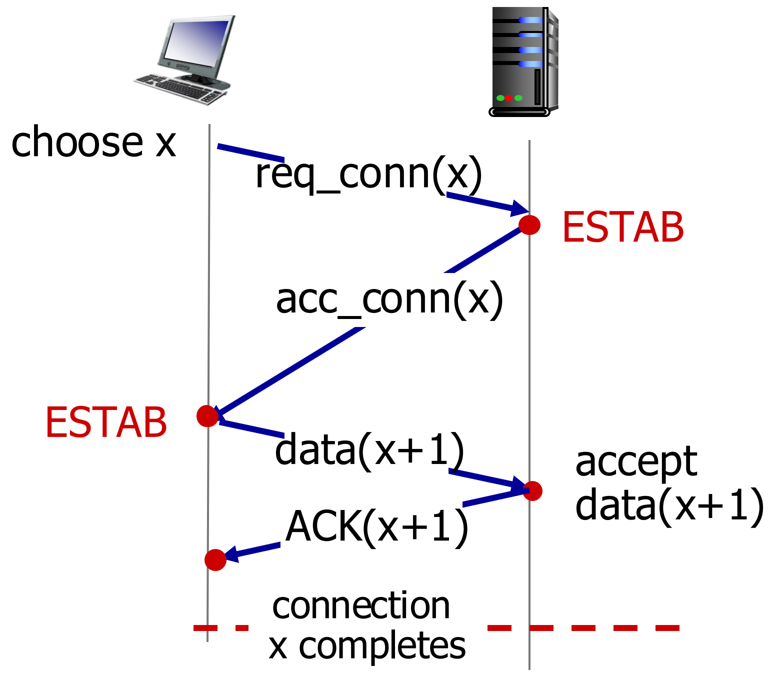


Q: will 2-way handshake always work in network?

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

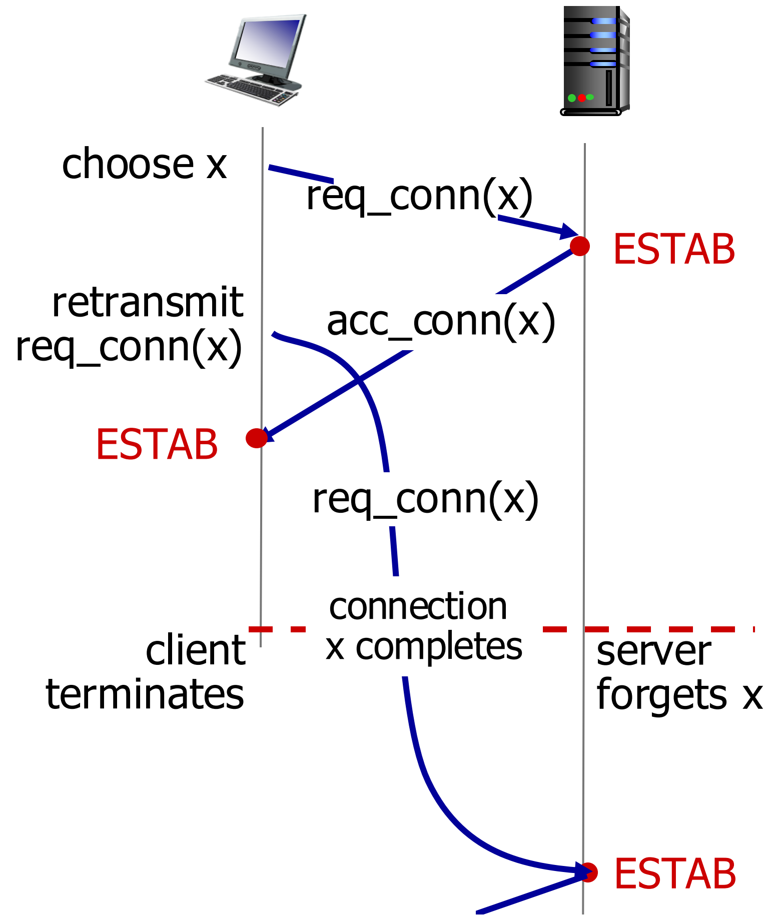
京都大学

2-way handshake scenarios



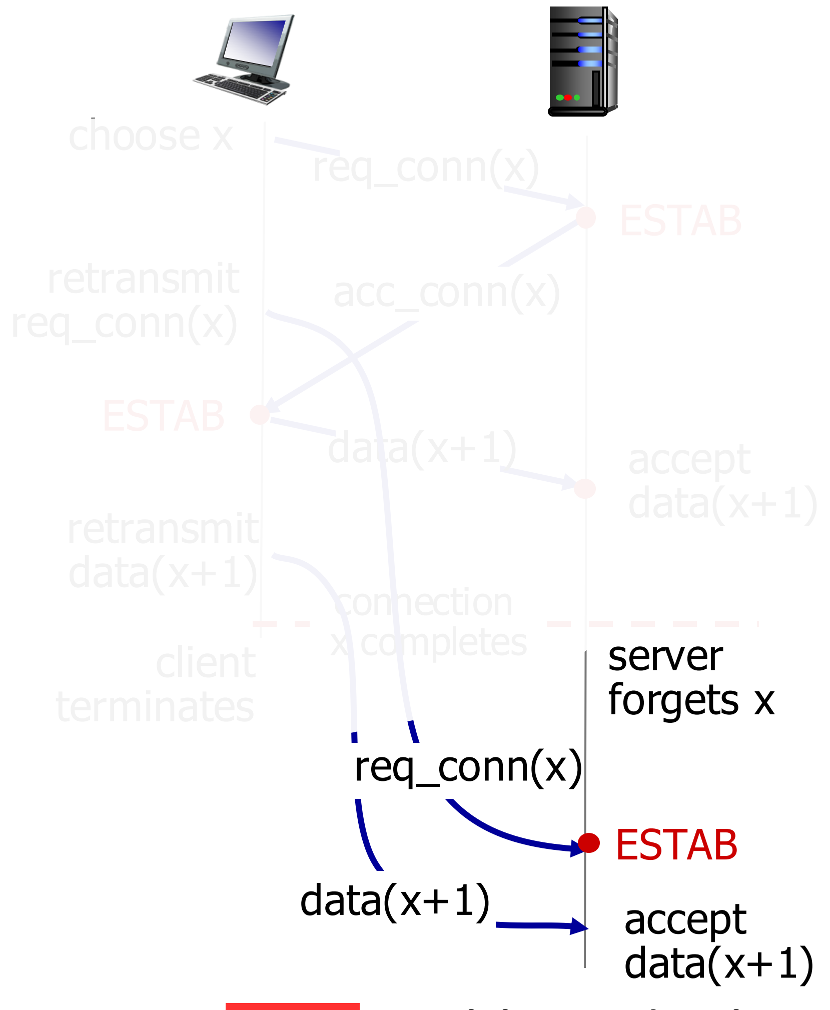
No problem!

2-way handshake scenarios



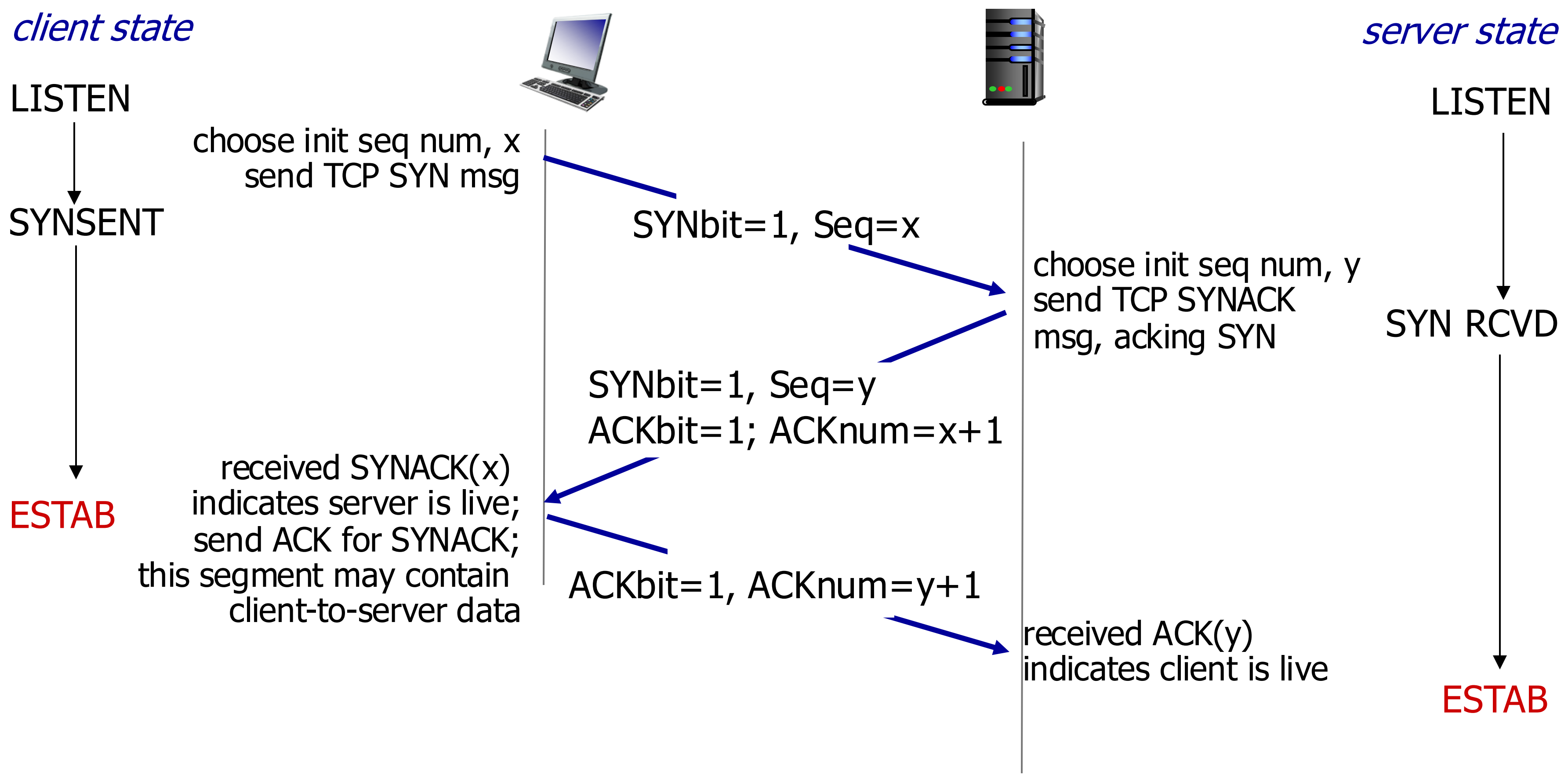
❌ Problem: half open connection! (no client)

2-way handshake scenarios



❌ Problem: duplicate data accepted!

TCP 3-way handshake

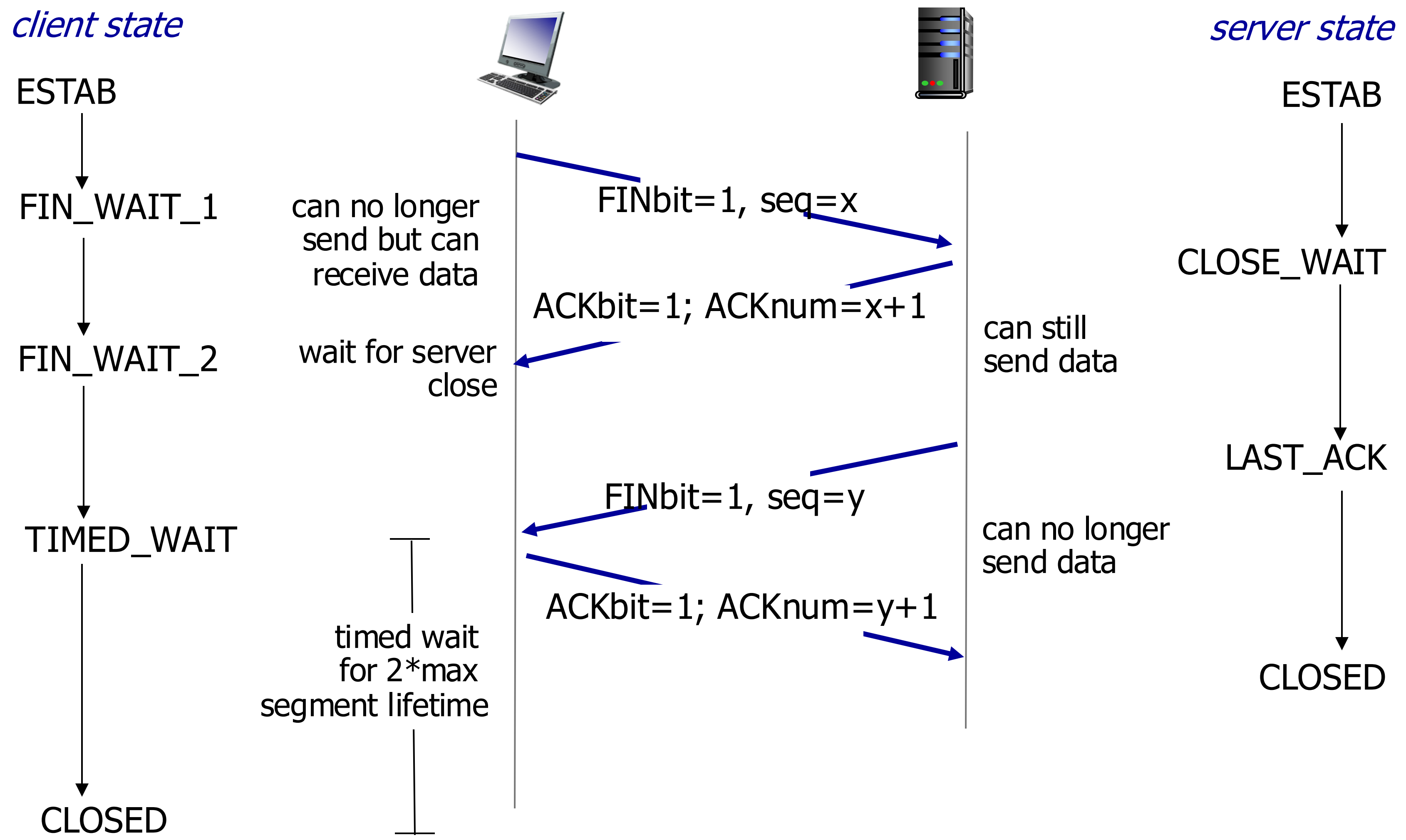


京都大学

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

TCP: closing a connection



京都大学

Chapter 3 outline

京都大学

Principles of congestion control

congestion:

- informally: “too many sources sending too much data too fast for *network* to handle”
- different from flow control!
- manifestations:
    - lost packets (buffer overflow at routers)
    - long delays (queueing in router buffers)

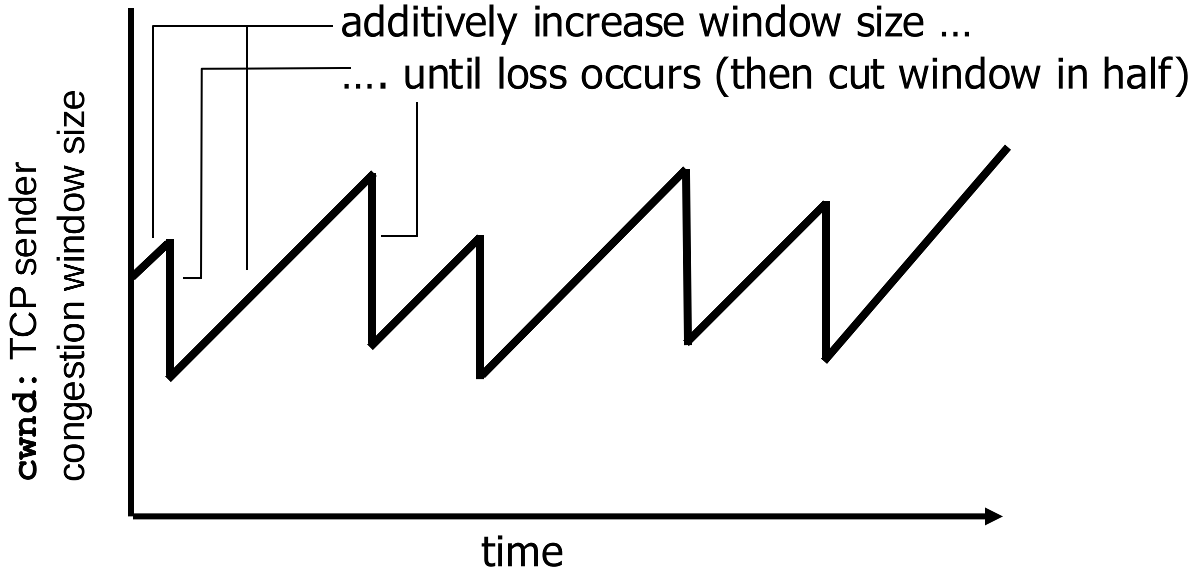
TCP congestion control

- TCP limits the send rate when the network is congested.
- How does TCP know when the network is congested?
  - If the network is congested, packets get lost.
  - Whenever TCP thinks that a packet got lost, it assumes that the network is congested.
- How does TCP slow down the send rate?
  - By limiting the size of the sender window.
- ACK received -> no congestion, increase window size
- ACK not received -> congestion, decrease window size

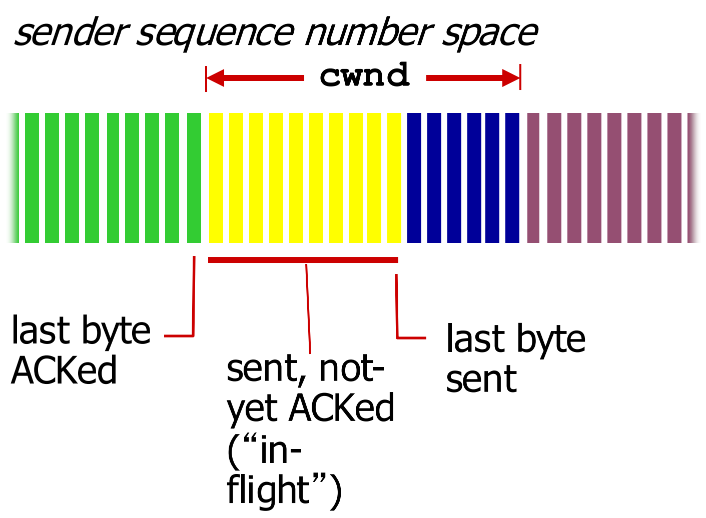
TCP congestion control: additive increase multiplicative decrease

- *approach:* sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **`cwnd`** by 1 MSS every RTT until loss detected
  - *multiplicative decrease*: cut **`cwnd`** in half after loss

AIMD saw tooth behavior: probing for bandwidth



TCP Congestion Control: details


sender sequence number space
cwnd
last byte ACKed
sent, not-yet ACKed ("in-flight")
last byte sent

- sender limits transmission:

$$\boxed{\texttt{LastByteSent-LastByteAcked} \leq \texttt{cwnd}}$$

- **`cwnd`** is dynamic, function of perceived network congestion
- The actual window size is limited by min(rwnd, cwnd)

TCP sending rate:

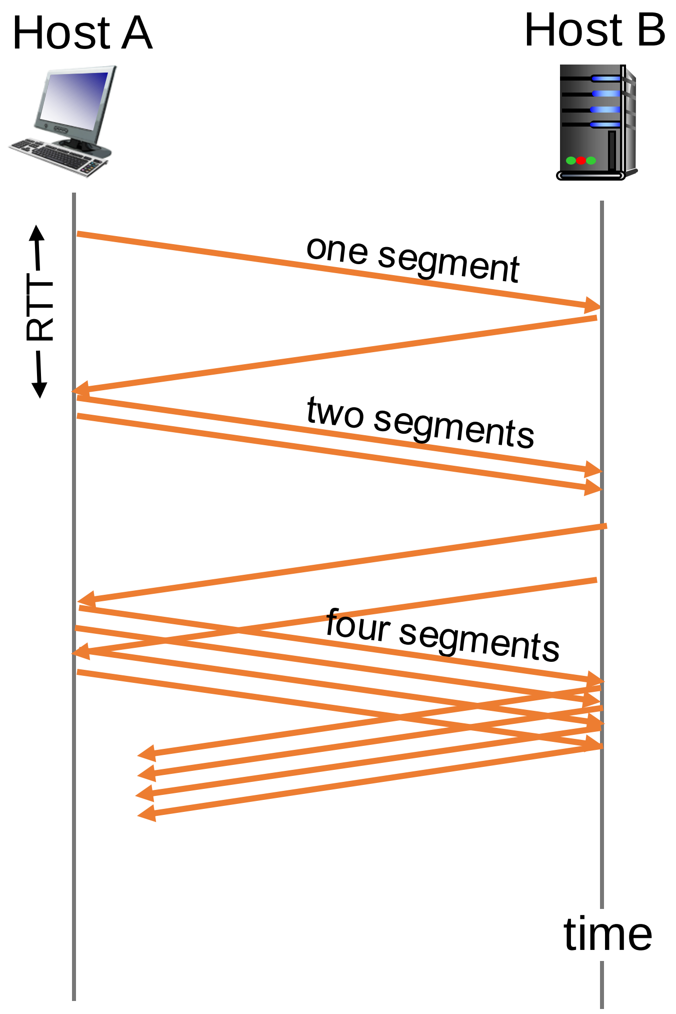
- *roughly:* send cwnd bytes, wait RTT for ACKS, then send more bytes

$$\boxed{\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}}$$

京都大学

TCP Slow Start

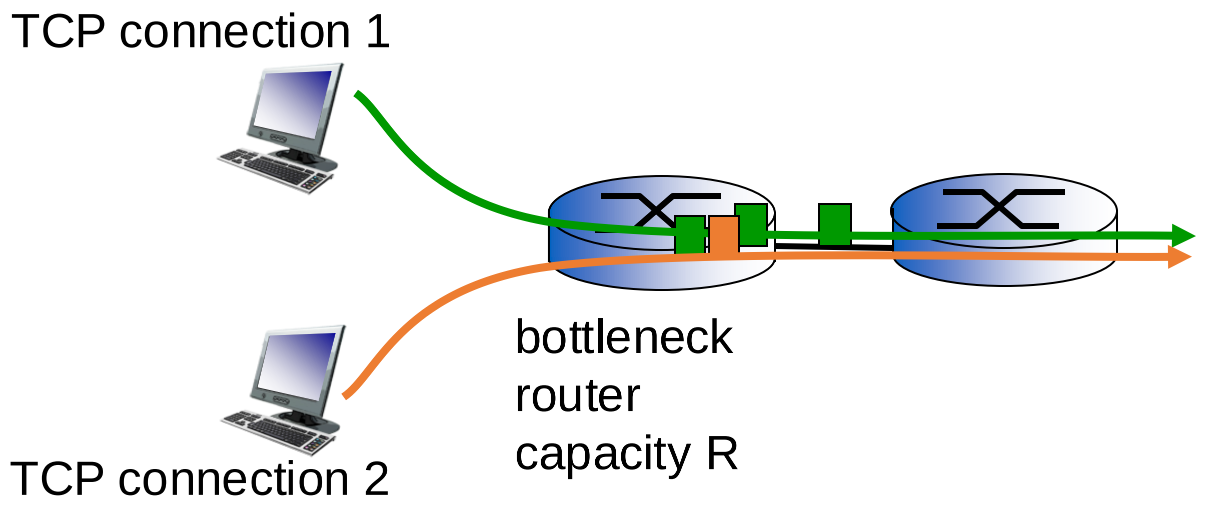
- when connection begins, increase rate exponentially until first loss event:
  - initially **`cwnd`** = 1 MSS
  - double **`cwnd`** every RTT
  - done by incrementing **`cwnd`** for every ACK received
- *summary:* initial rate is slow but ramps up exponentially fast



京都大学

TCP Fairness

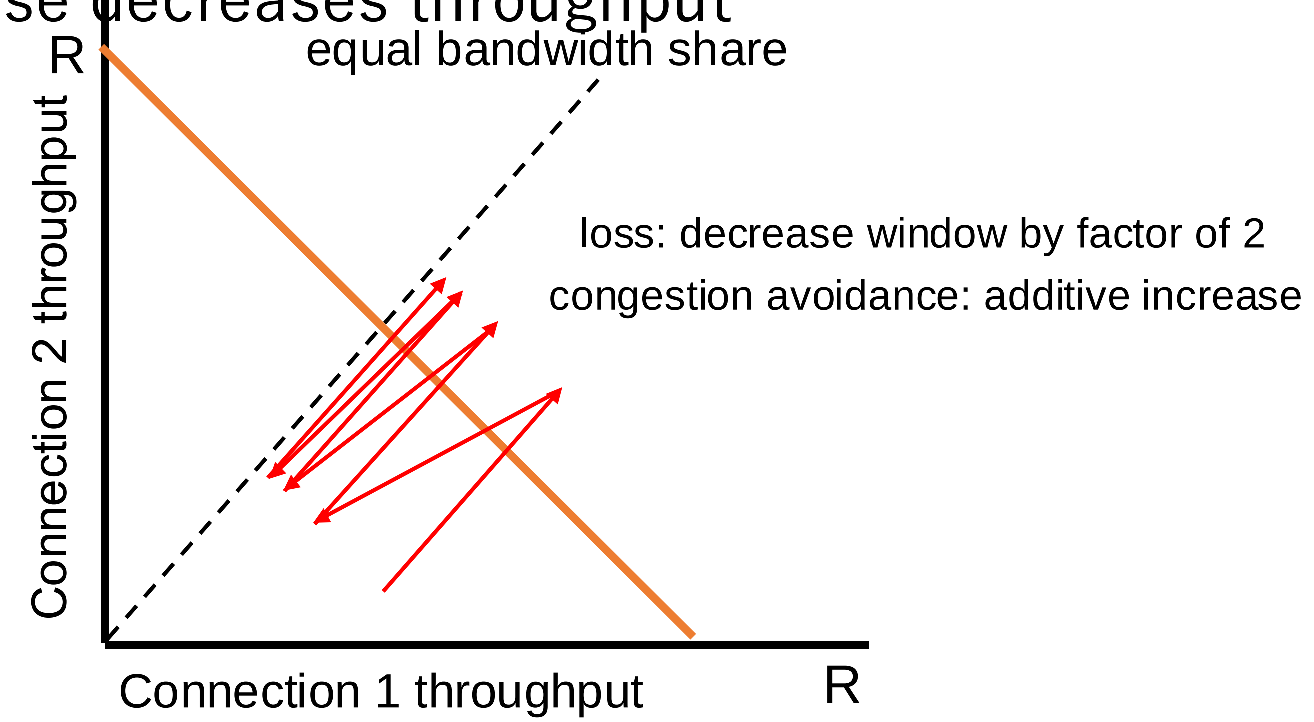
fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Why is TCP fair?

two competing sessions:

- additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally


R
equal bandwidth share
loss: decrease window by factor of 2
congestion avoidance: additive increase
Connection 2 throughput
Connection 1 throughput
R

京都大学

Fairness (more)

Fairness and UDP

- multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- instead use UDP:
  - send audio/video at constant rate, tolerate packet loss

Fairness, parallel TCP connections

- application can open multiple parallel connections between two hosts
- web browsers do this
- e.g., link of rate R with 9 existing connections:
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2

京都大学

Transport layer: summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation, implementation in the Internet
  - UDP
  - TCP

next:

- leaving the network "edge" (application, transport layers)
- into the network "core"

京都大学