Information Network

Lecture 8 : Reliable Data Transfer

Holger Thies

KYOTO UNIVERSITY

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

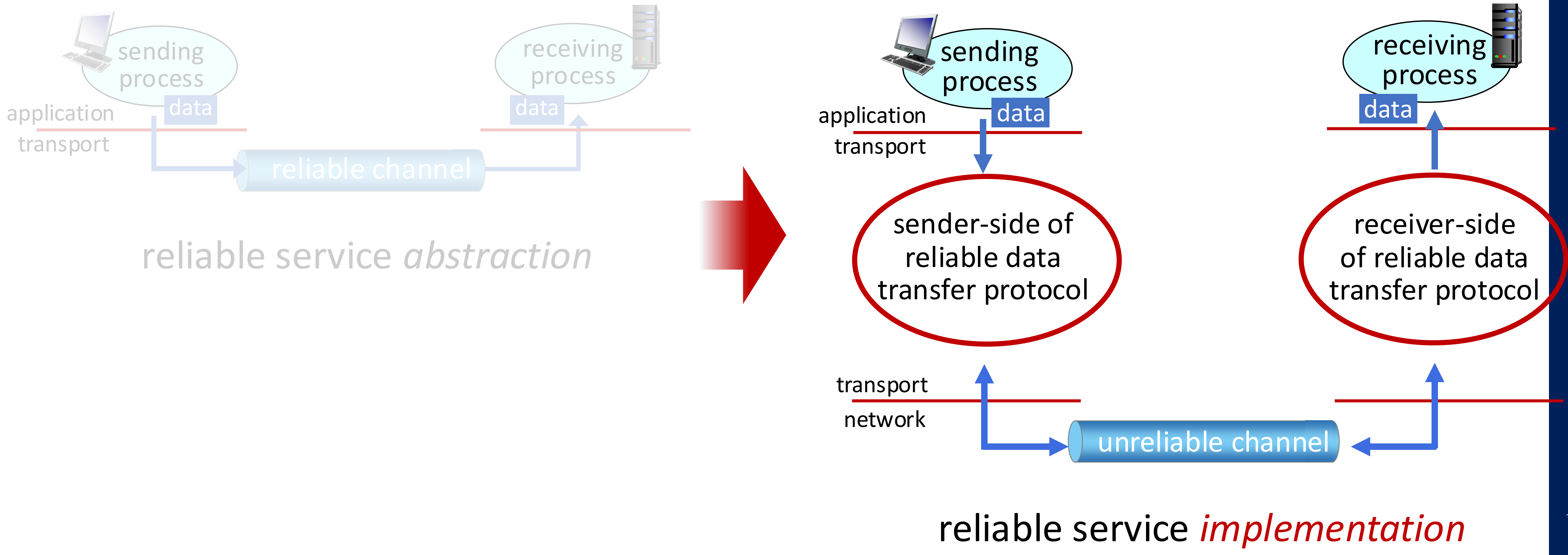
京都大学

Principles of reliable data transfer

reliable service *abstraction*

京都大学

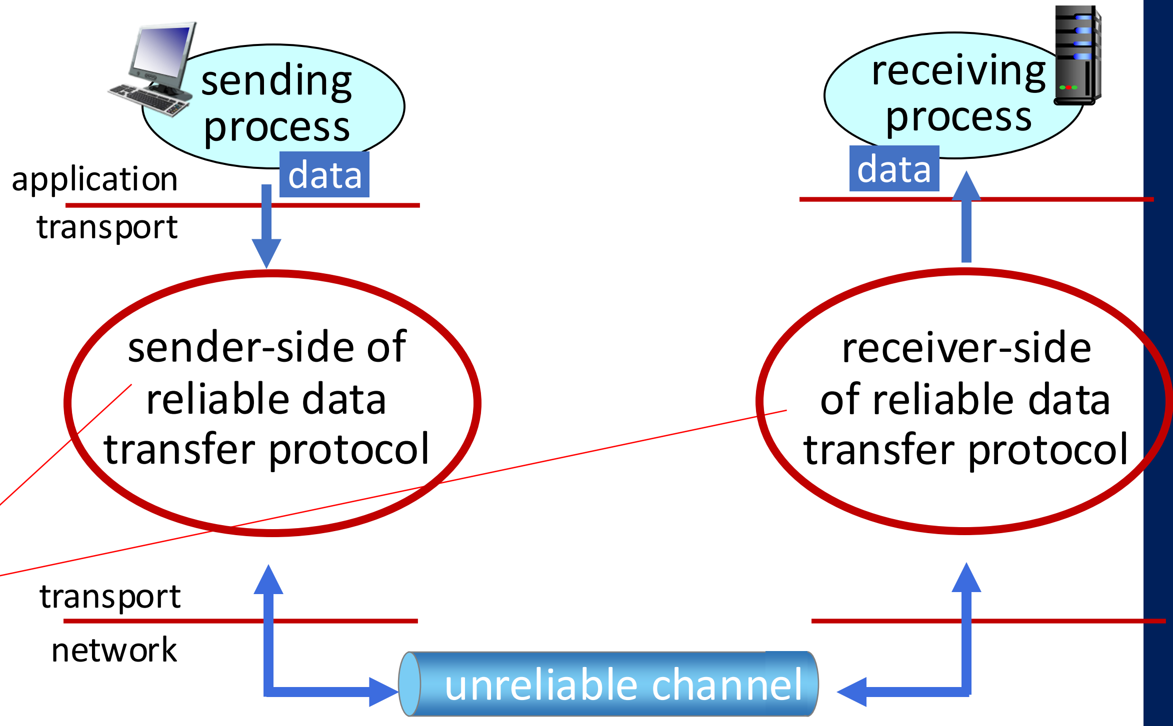
Principles of reliable data transfer



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?)

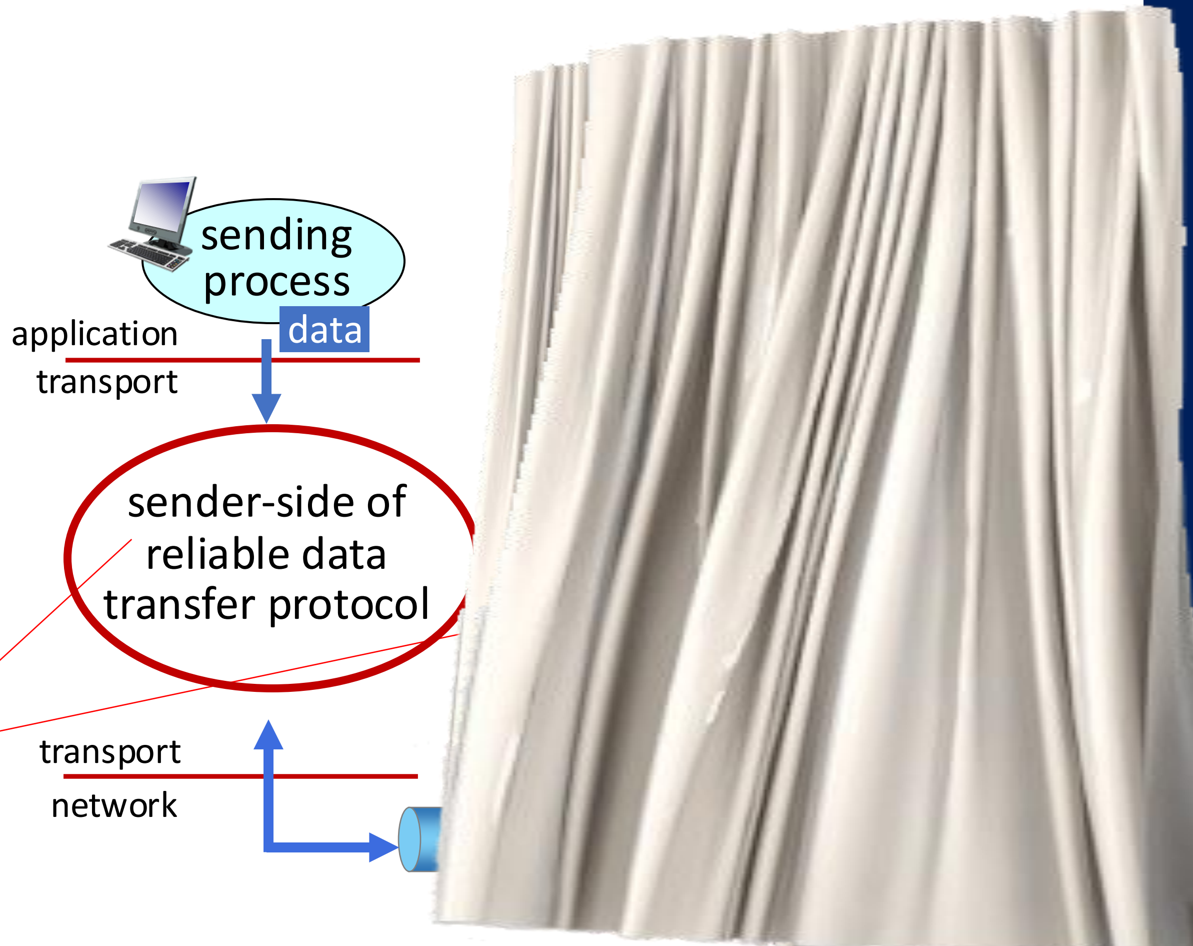


reliable service *implementation*

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

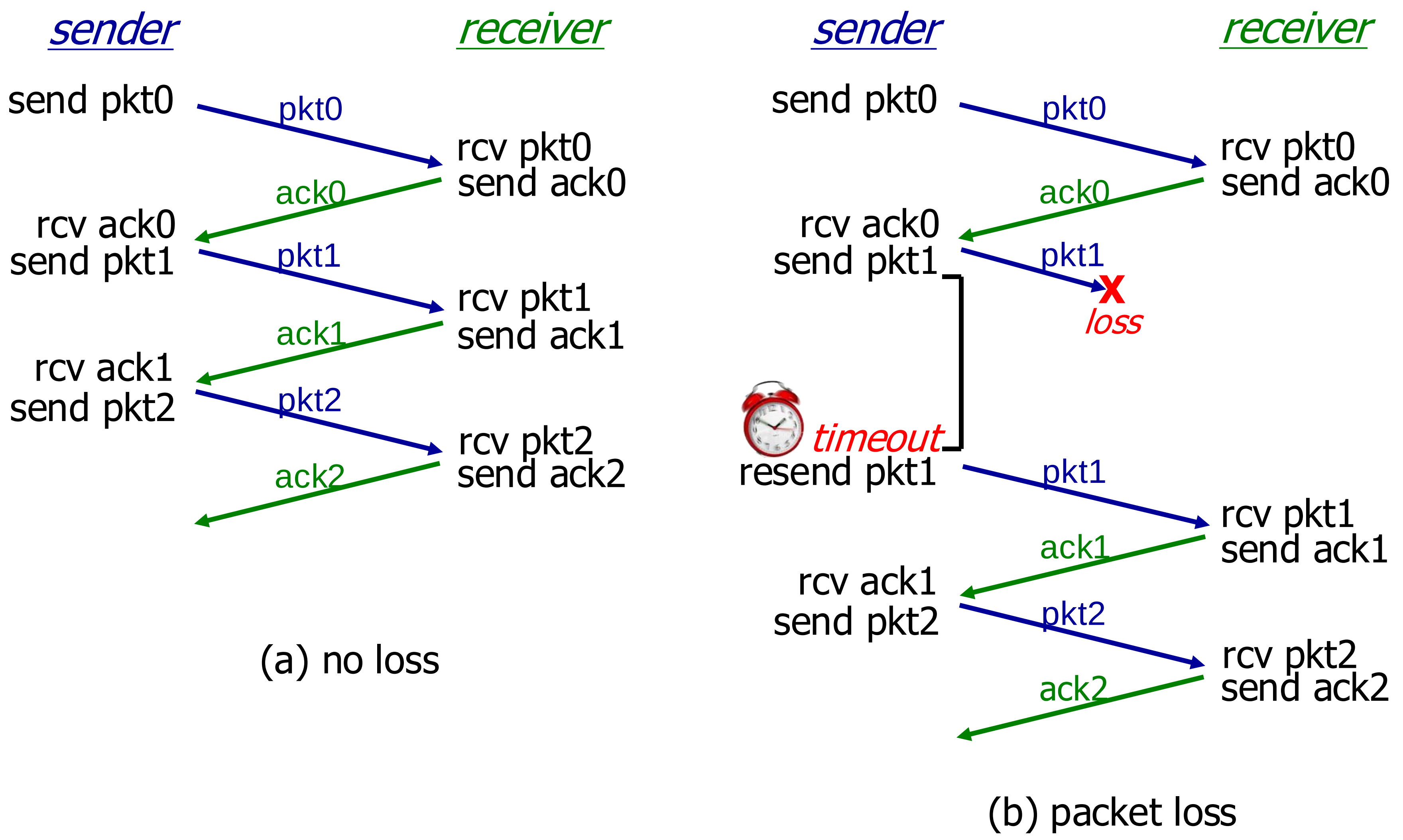


reliable service *implementation*

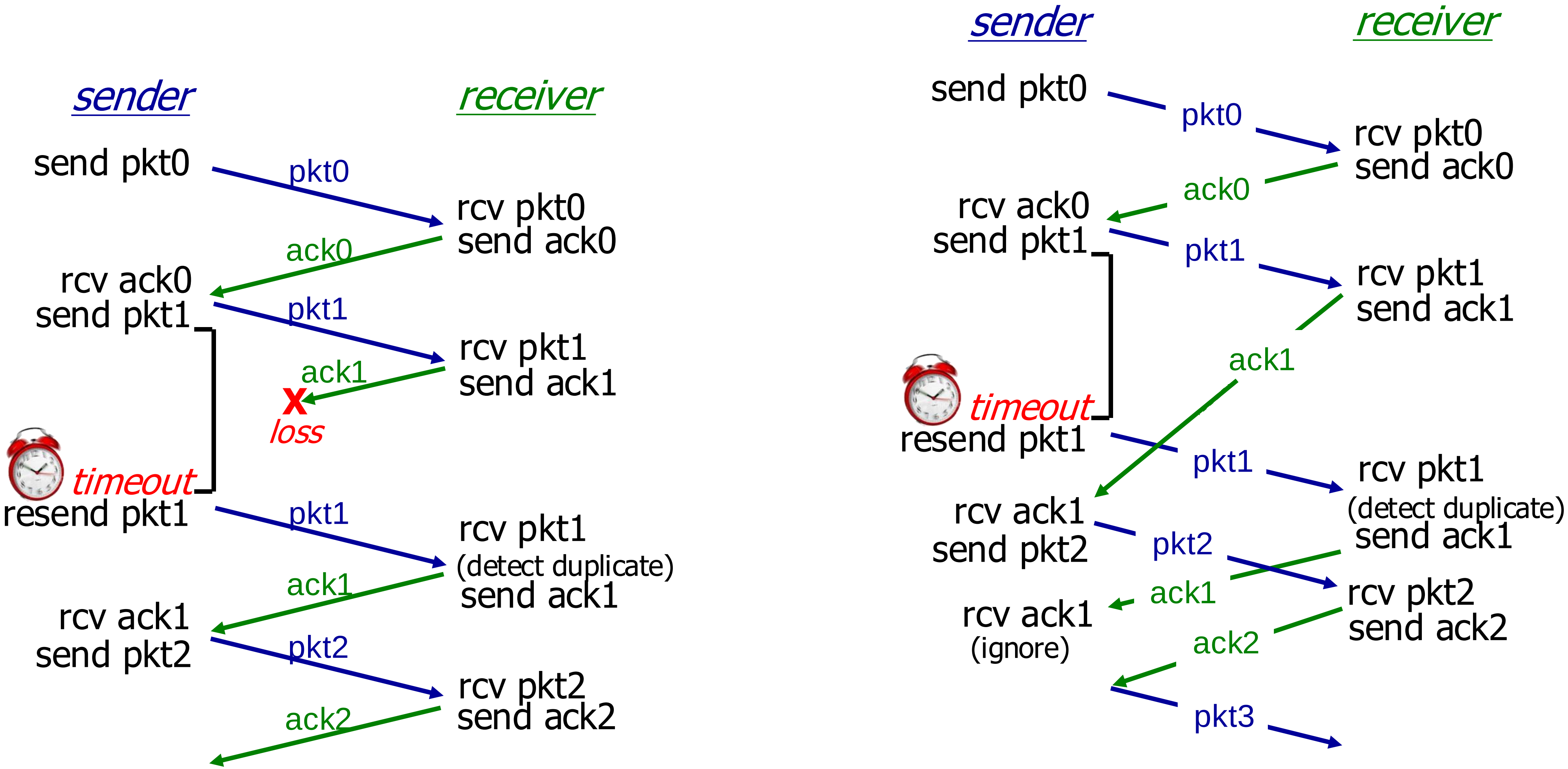
Sequence numbers

- 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



Reliable data transfer using ack messages



(c) ACK loss

(d) premature timeout/ delayed ACK

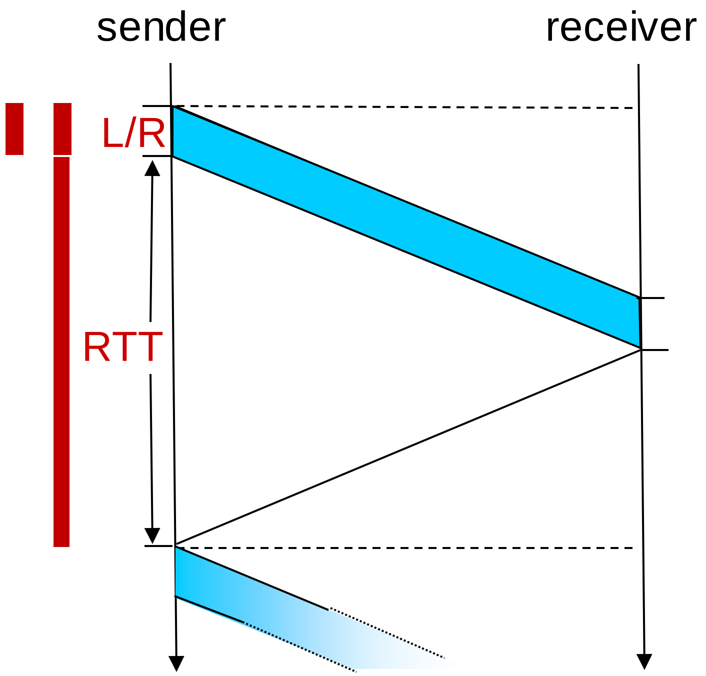
Note: 1 bit sequence number (0 or 1) suffices

Stop and wait protocol

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

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

$$= 0.027\%$$

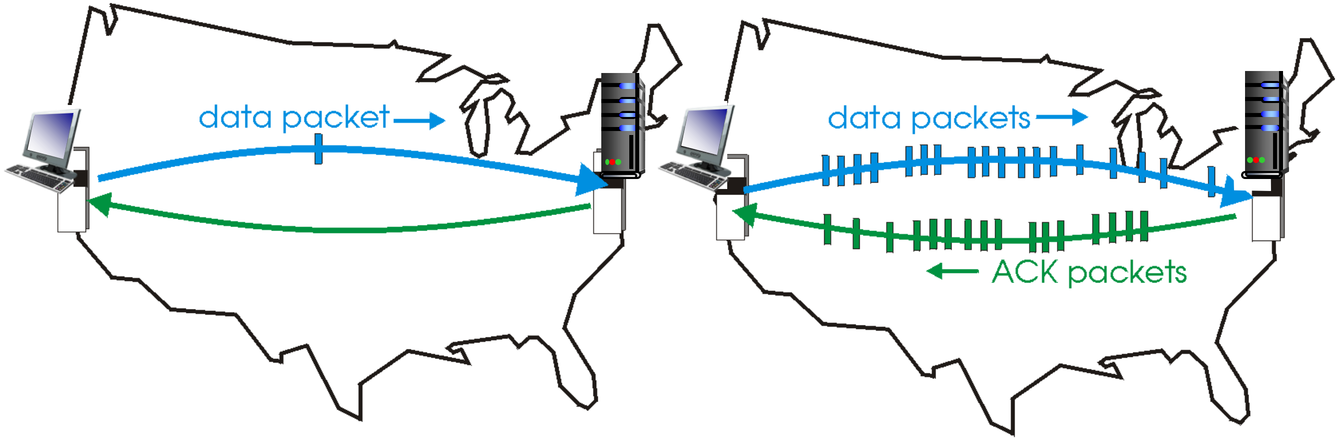


- D is the transmission delay (the amount of time required to push all the packet's bits into the wire)
- if RTT=30 msec, 1KB packet every 30 msec: 33kB/sec throughput over 1 Gbps link
- Protocol limits performance of underlying infrastructure (channel)

京都大学

Pipelined protocols

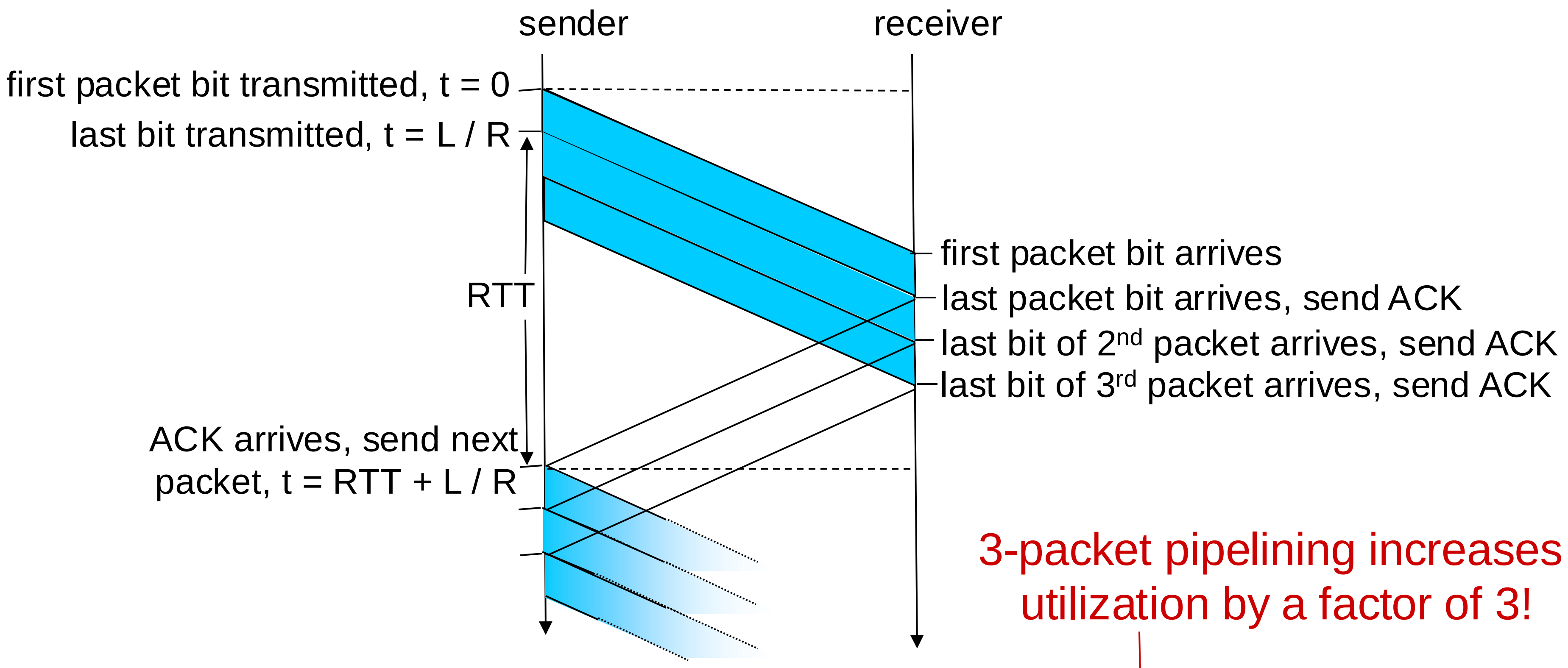
pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged packets



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

(b) a pipelined protocol in operation

Pipelining: increased utilization



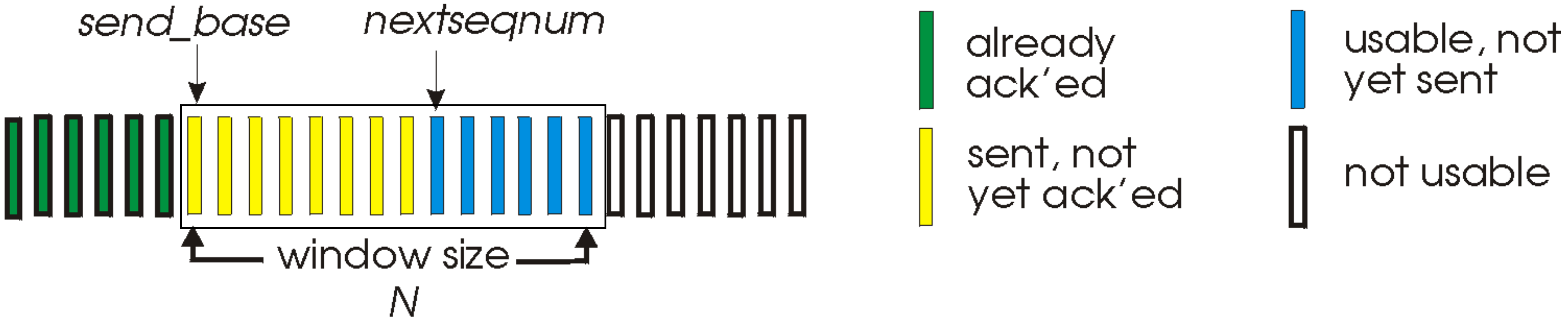
3-packet pipelining increases utilization by a factor of 3!

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

京都大学

Go-Back-N: sender

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



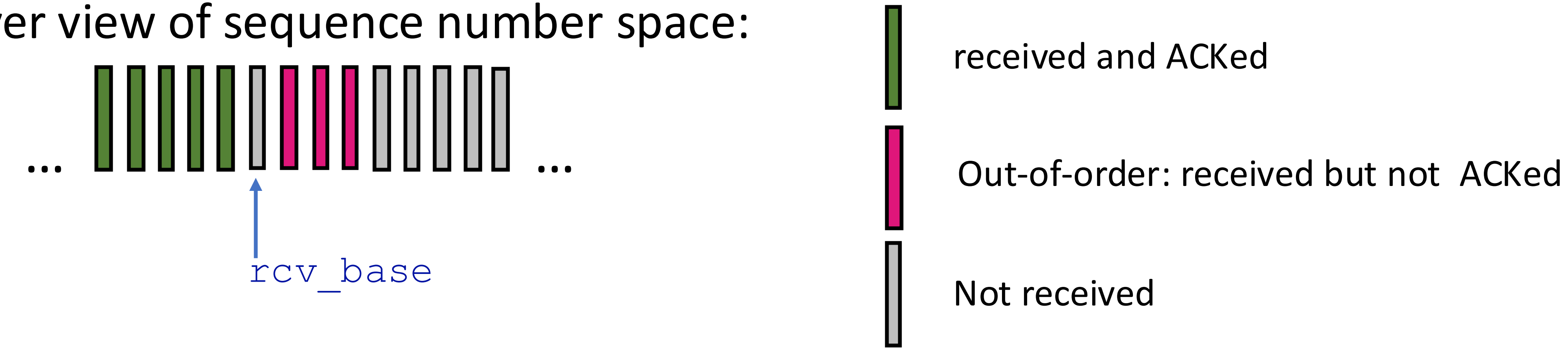
- ACK(n): ACKs all pkts up to, including sequence number n - *"cumulative ACK"*
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt n
- *timeout:* retransmit packet n and all higher sequence number packets in window

京都大学

Go-Back-N: receiver

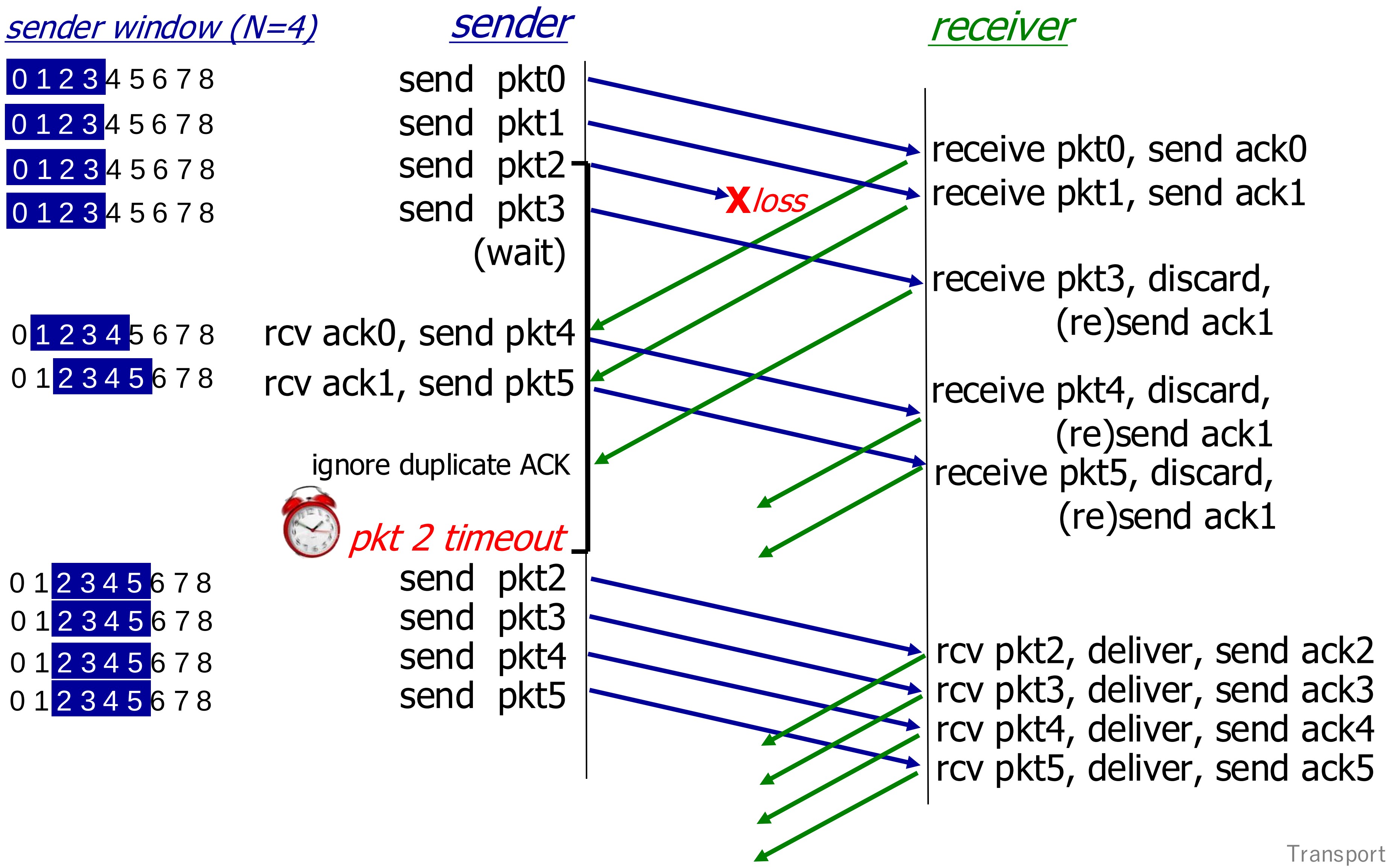
- ACK-only: always send ACK for correctly-received packet so far, with highest *in-order* seq number
  - may generate duplicate ACKs
  - need only remember `rcv_base`
- on receipt of out-of-order packet:
  - can discard (don't buffer) or buffer: an implementation decision
  - re-ACK packet with highest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

Receiver view of sequence number space:



京都大学

Go-Back-N in action



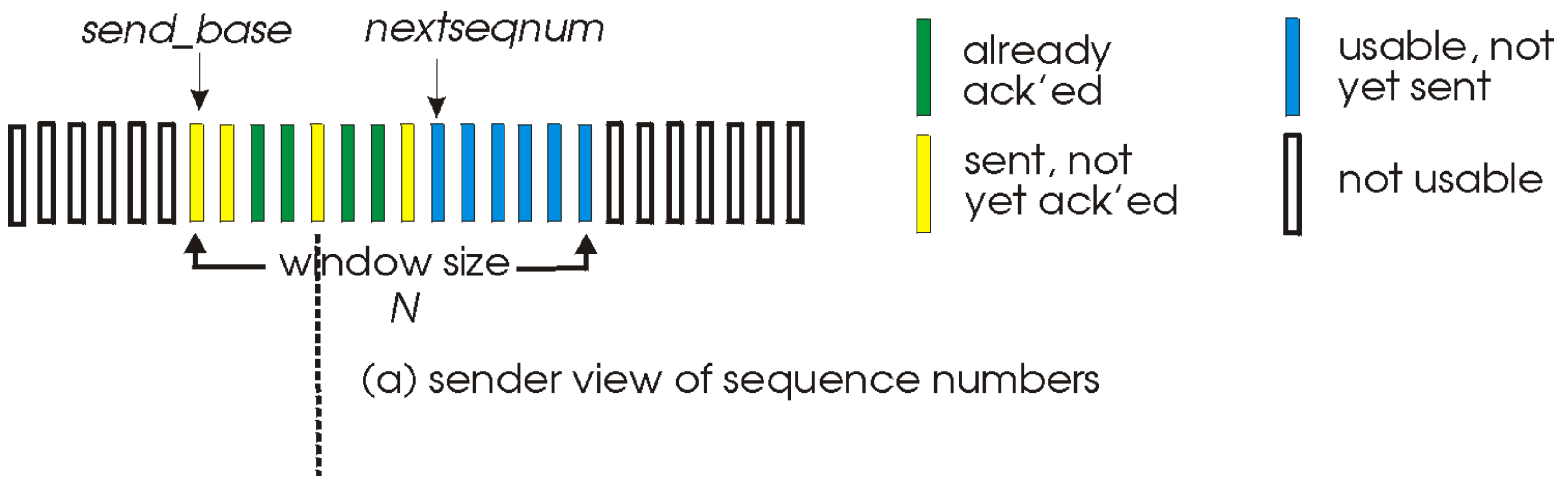
京都大学

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

Example

Host A sends a message to Host B consisting of 10 packets using a pipelined reliable data transfer protocol with the Go-Back-N strategy.

Assume the window size is 4 (that is, N=4) and that every 6th packet that A sends to B is lost (but no acknowledgment message from B to A is lost and there are no premature timeouts).

How many packets will A have to send in total to transmit the message to B?

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
- congestion control

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.

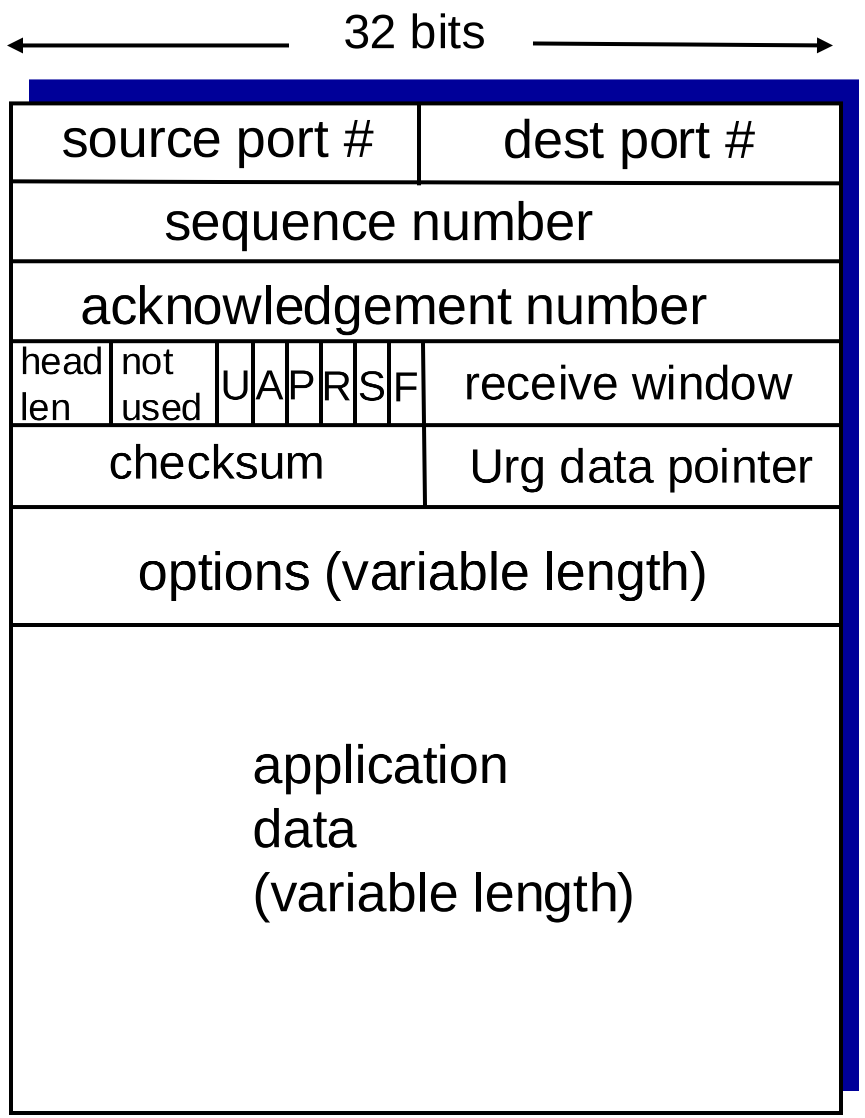
京都大学

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

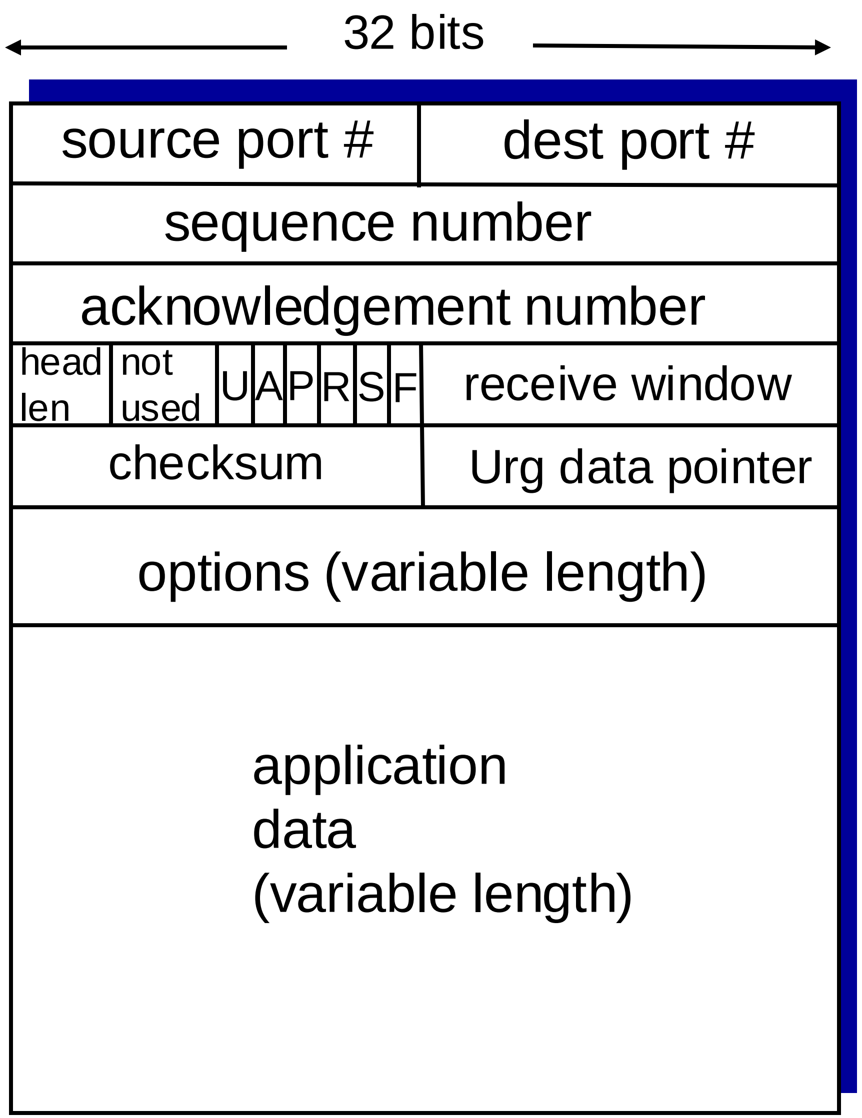
京都大学

TCP segment structure



TCP Segment Header

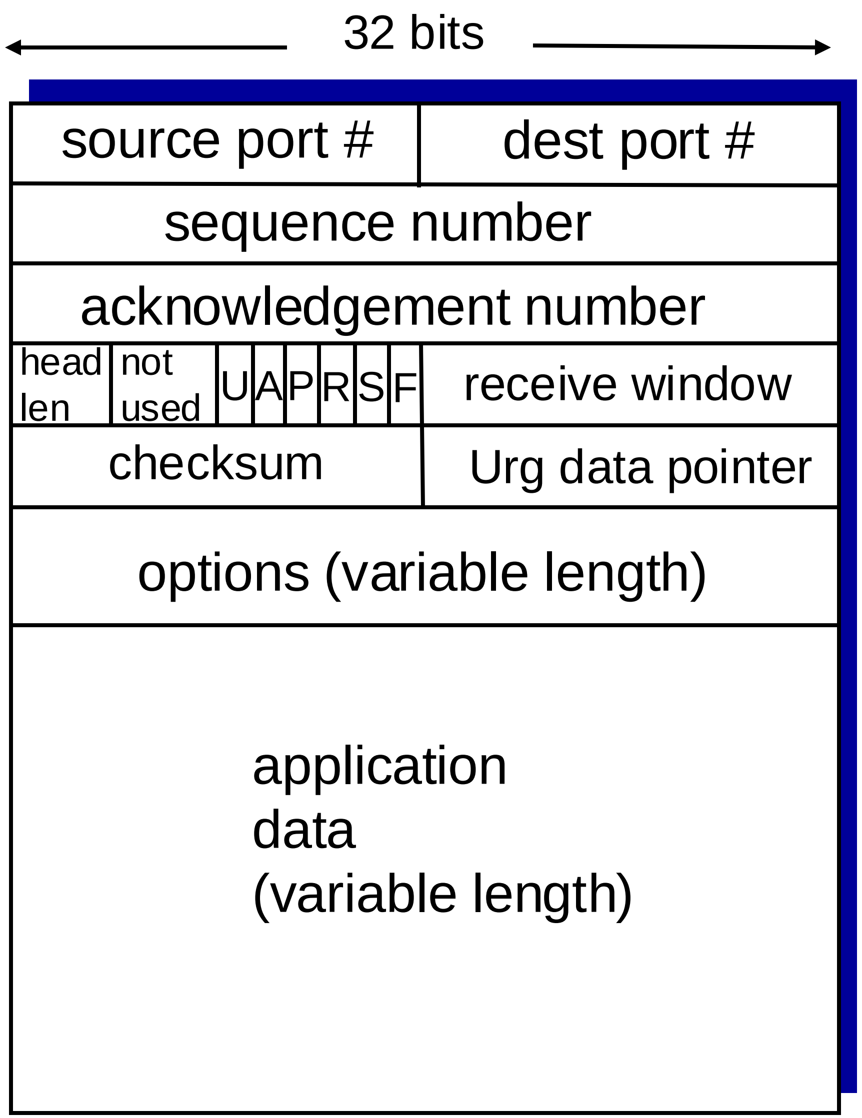
- Source Port, Destination Port
- Sequence Number
  - At the transport layer application data is split into several smaller segments.
  - Sequence Number is used to keep track of the position of the current segment in the sequence.
- Acknowledgment Number: Number of the next expected segment
  - Used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



京都大学

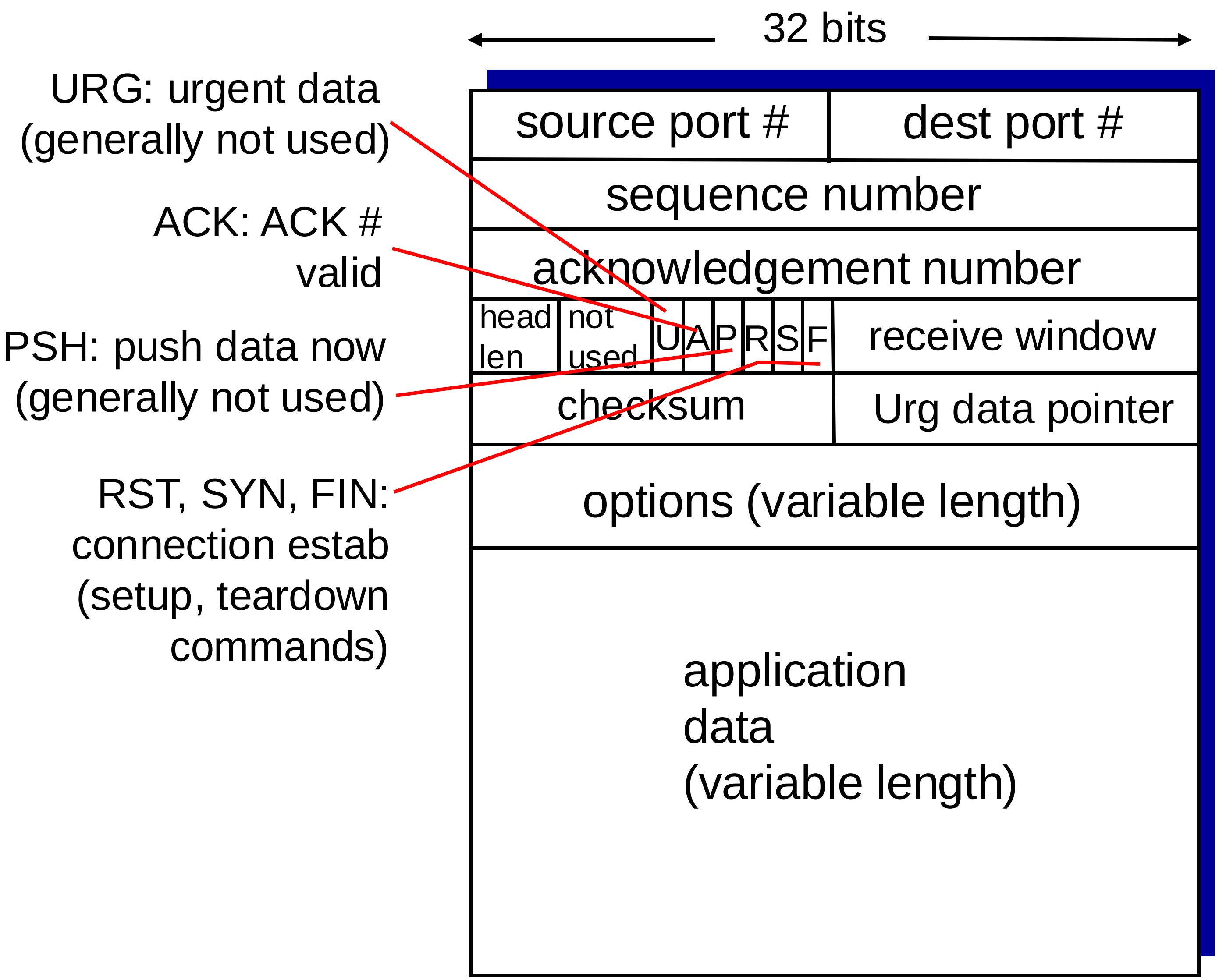
TCP Segment Header

- 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



京都大学

TCP control flags



TCP seq. numbers, ACKs

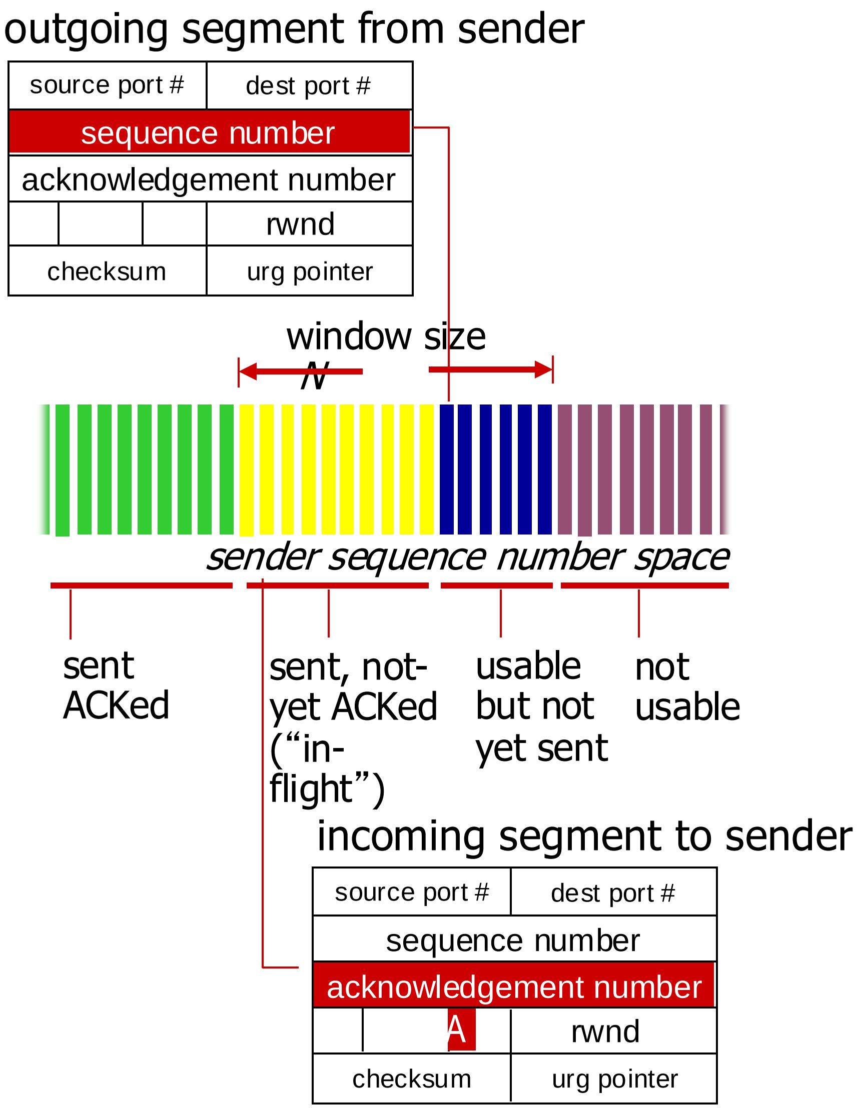
sequence numbers:

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

acknowledgements:

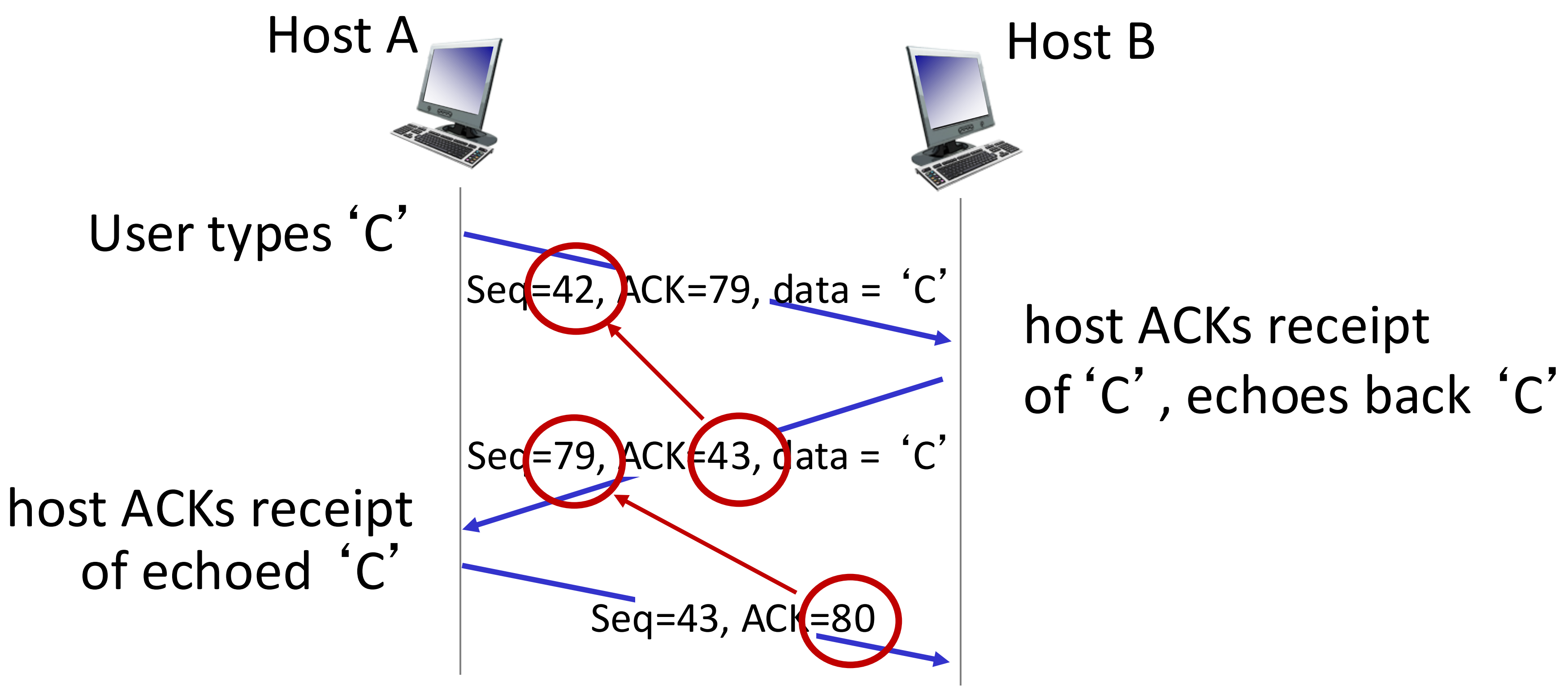
- Sequence number of next byte expected from other side
- cumulative acknowledgments

- 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



simple 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
- congestion control

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.

京都大学

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
  - 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

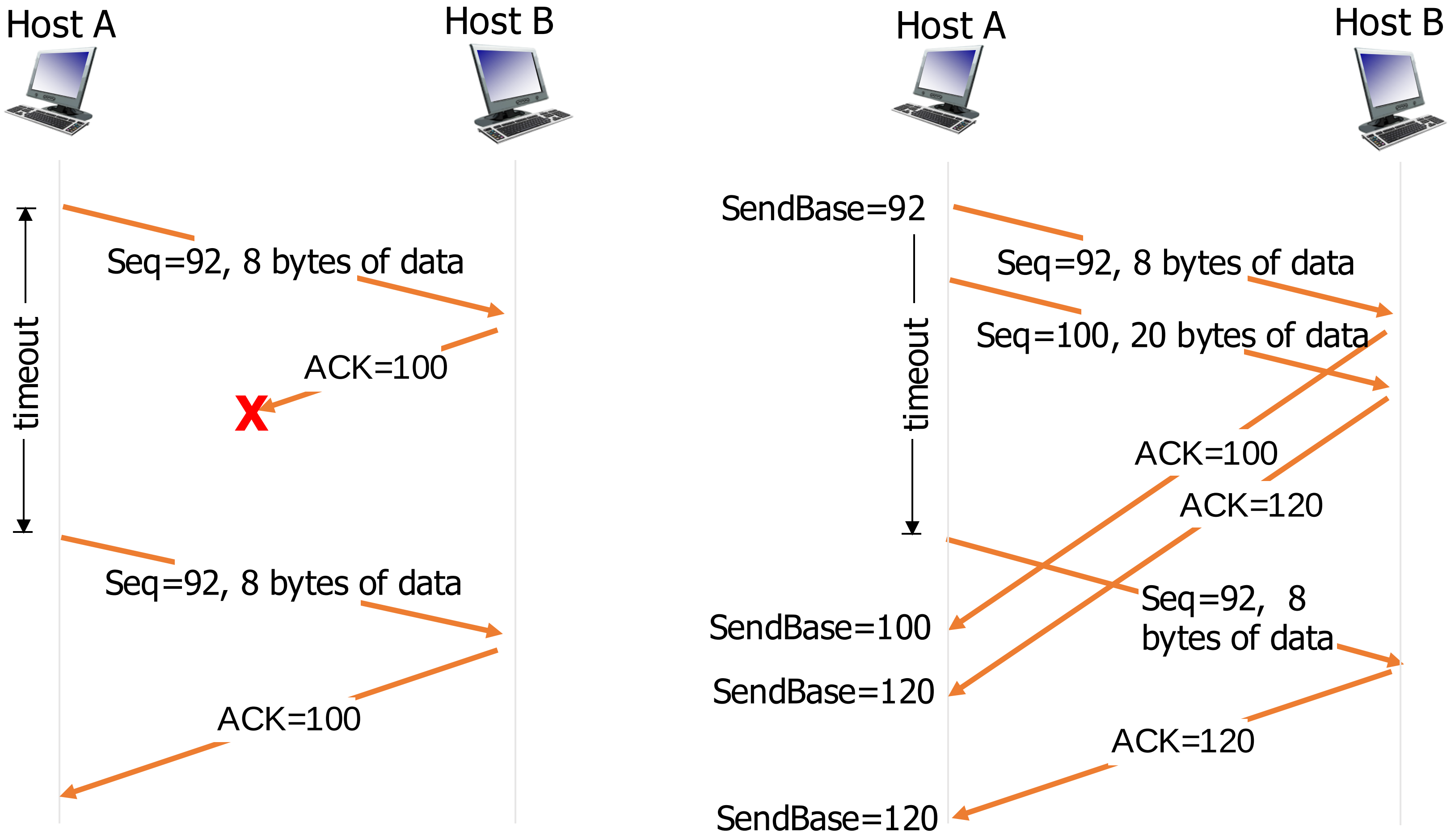
京都大学

TCP Receiver: ACK generation [RFC 5681]

Event at receiver	*TCP receiver action*

京都大学

TCP: retransmission scenarios

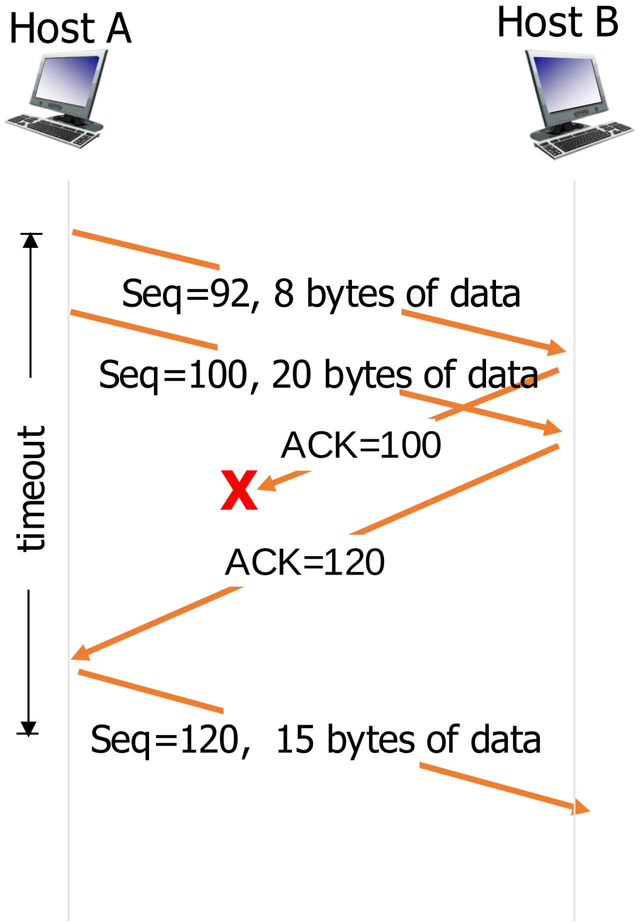


lost ACK scenario

premature timeout

京都大学

TCP: retransmission scenarios



cumulative ACK

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.

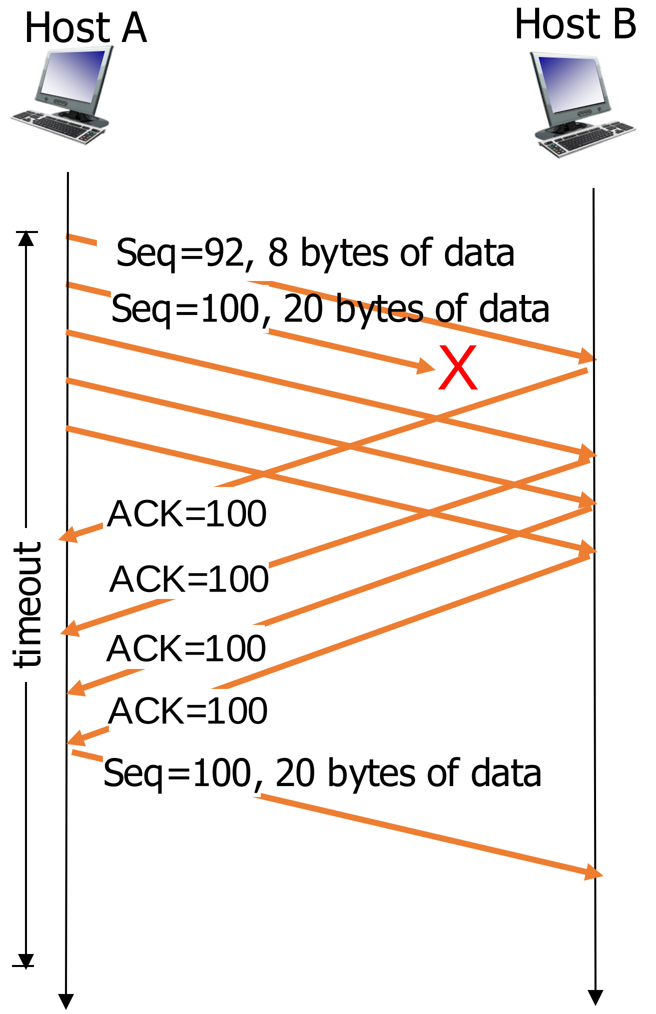
TCP fast retransmit

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

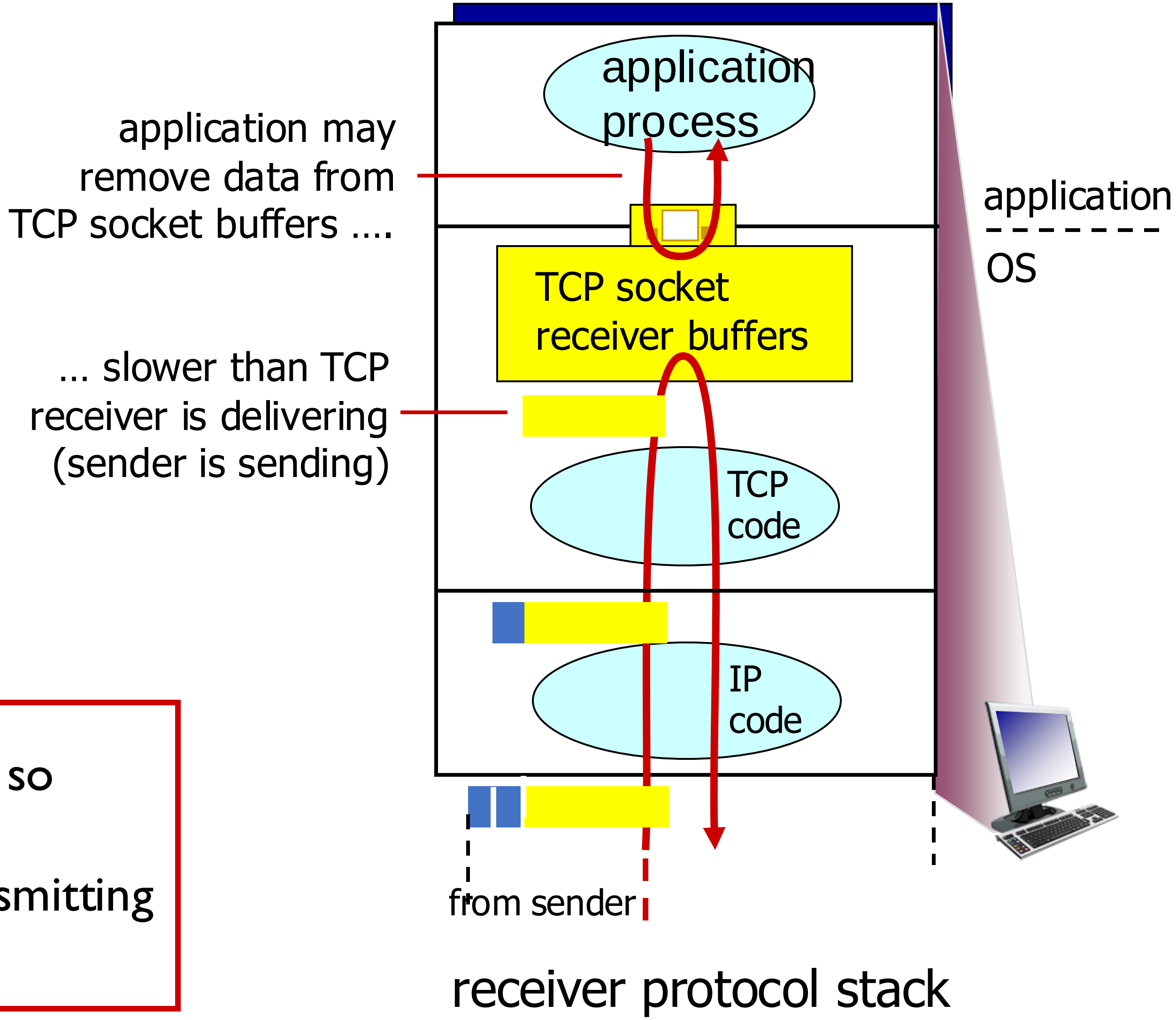
- segment structure
- reliable data transfer
- flow control
- connection management
- congestion control

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



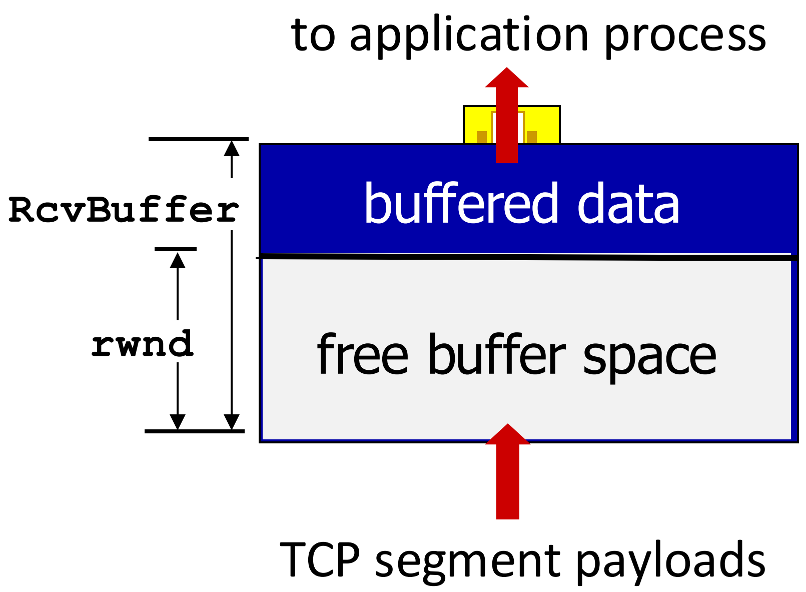
flow control

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 unacknowledged ("in-flight") data to received **`rwnd`**
- guarantees receive buffer will not overflow

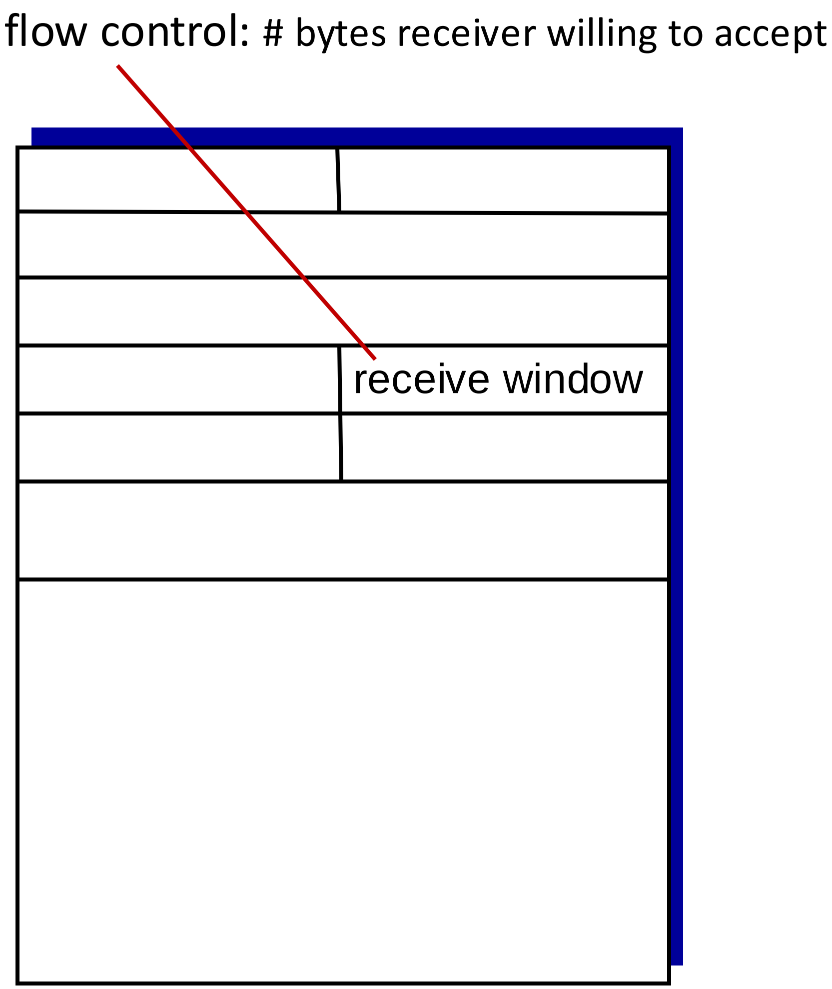


TCP receiver-side buffering

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



TCP segment format

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

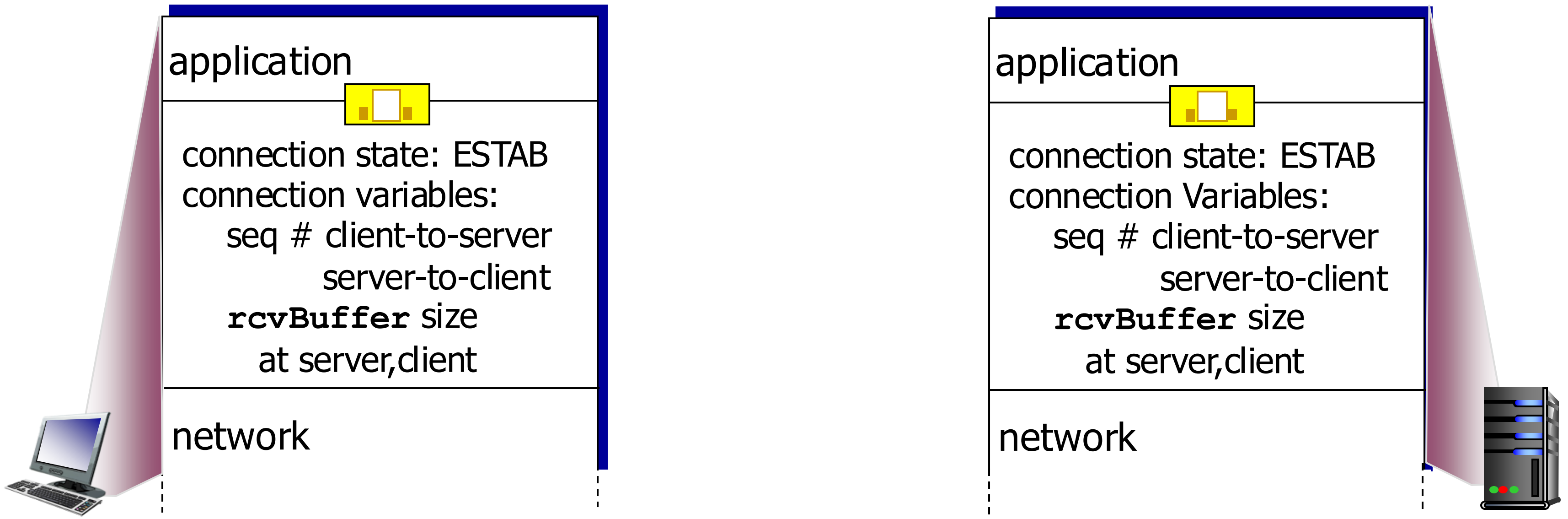
- segment structure
- reliable data transfer
- flow control
- connection management
- congestion control

京都大学

Connection Management

before exchanging data, sender/receiver "handshake":

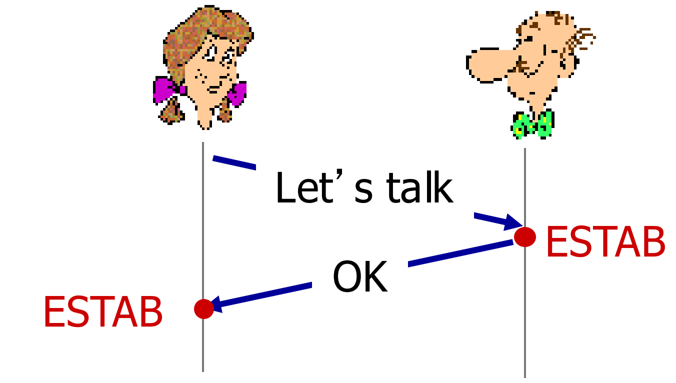
- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters

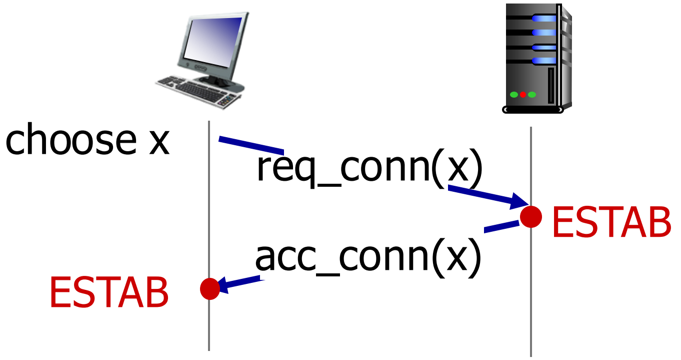


京都大学

Agreeing to establish a connection

2-way handshake:

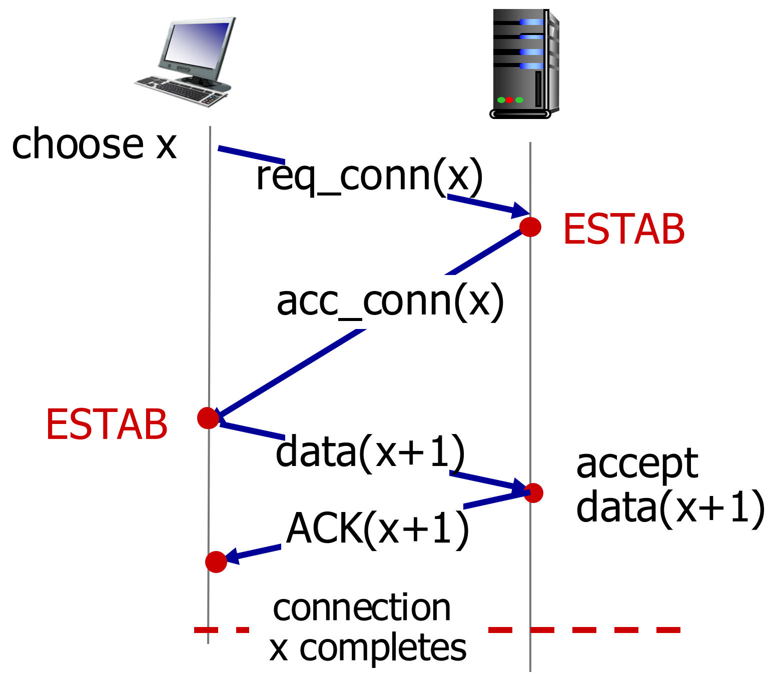




Q: will 2-way handshake always work in network?

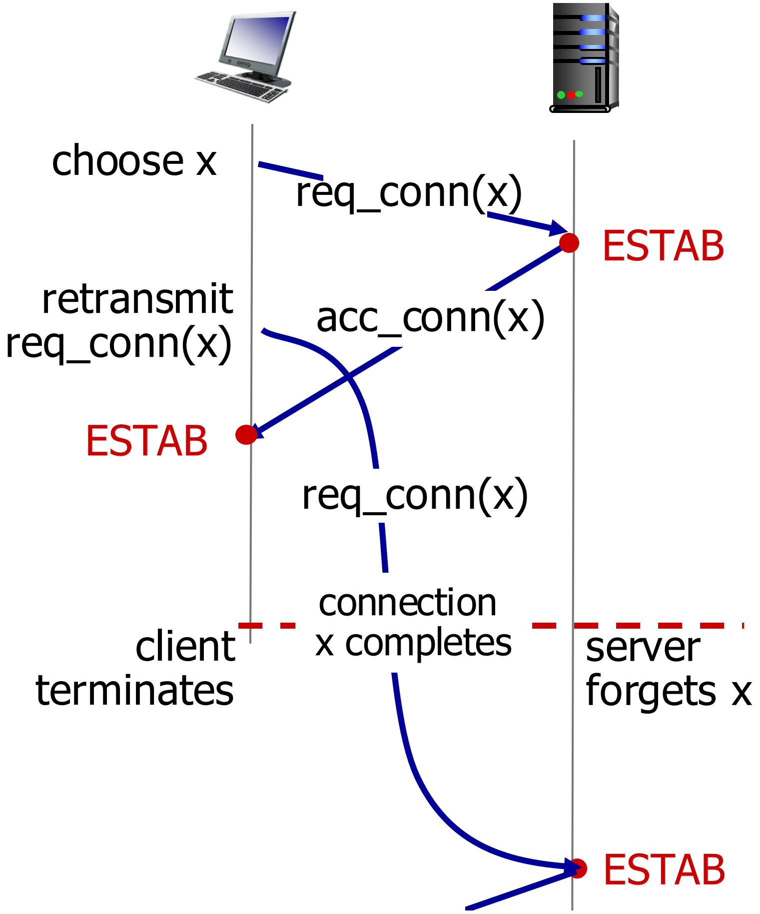
- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

2-way handshake scenarios



No problem!

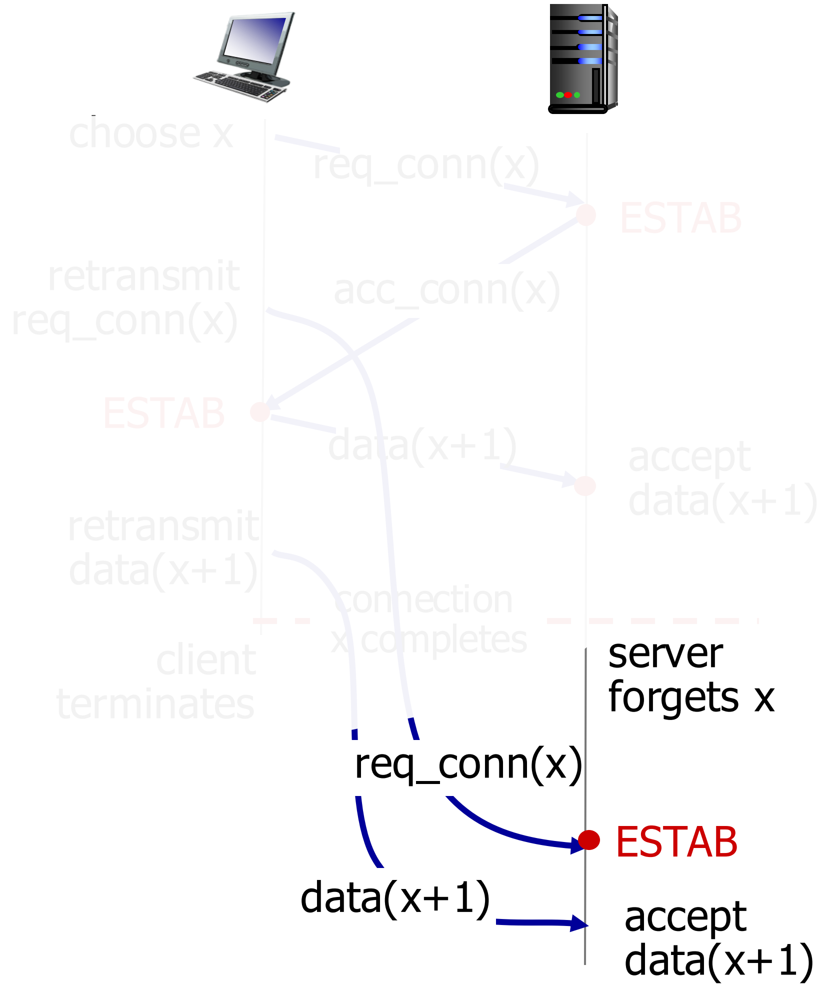
2-way handshake scenarios



Problem: half open connection! (no client)

京都大学

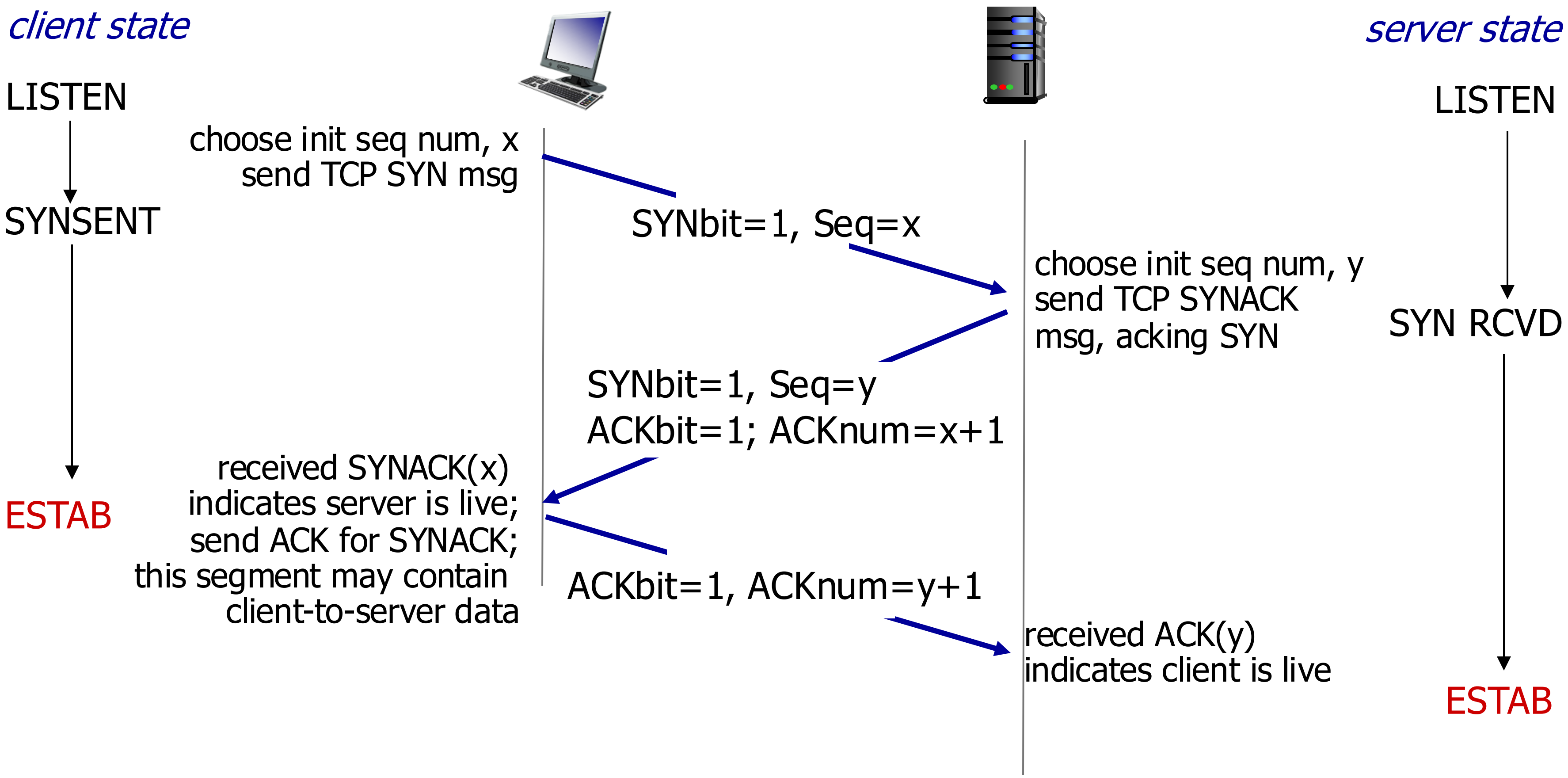
2-way handshake scenarios



Problem: duplicate data accepted!

京都大学

TCP 3-way handshake



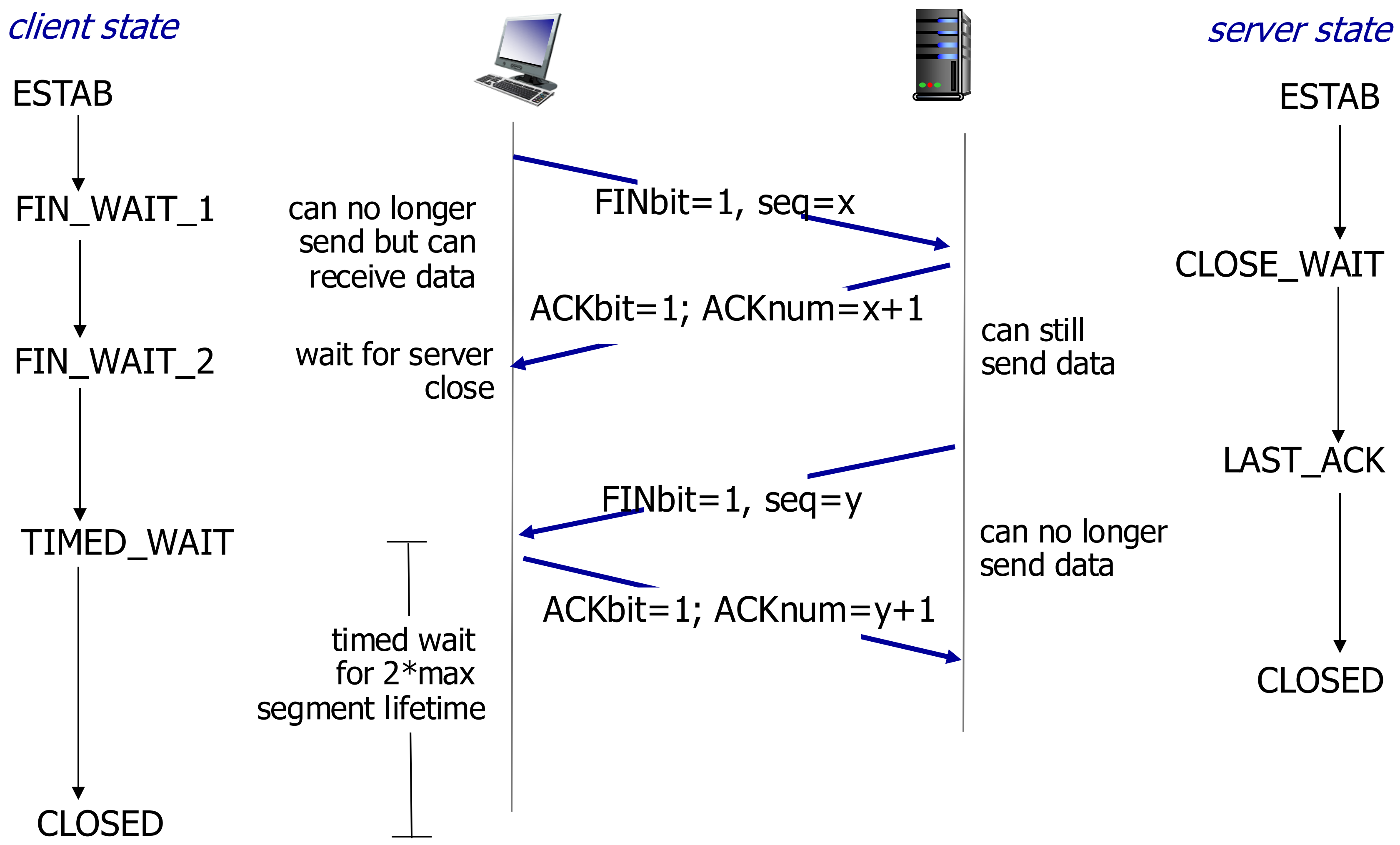
京都大学

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



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
- congestion control

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

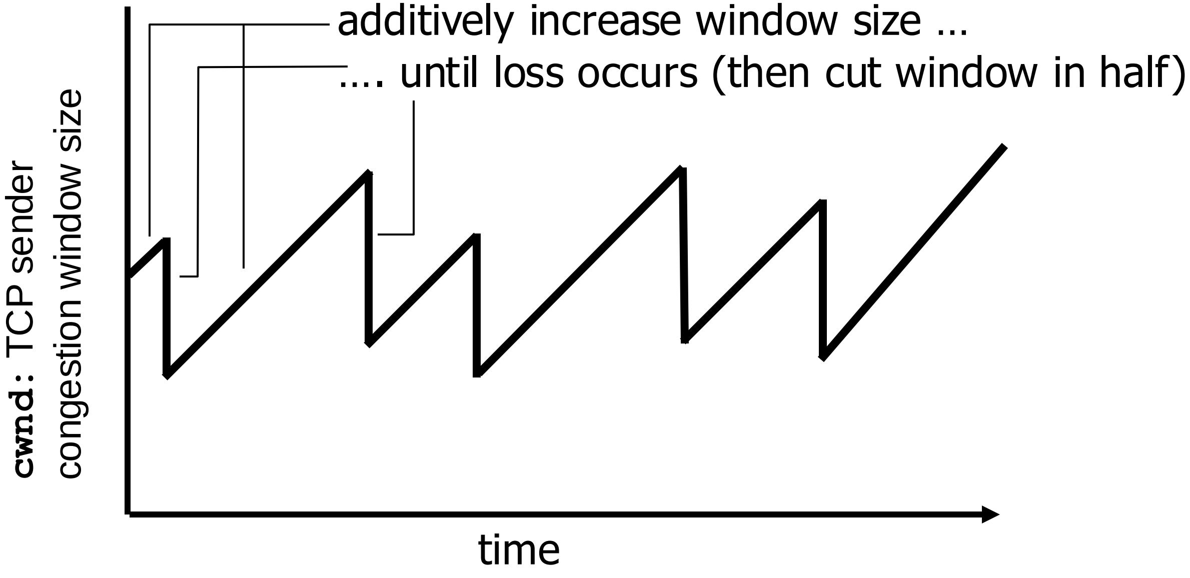
- TCP limits the send rate when the network is congested.
- How does TCP know when the network is congested?
  - If the network is congested, packets get lost.
  - Whenever TCP thinks that a packet got lost, it assumes that the network is congested.
- How does TCP slow down the send rate?
  - By limiting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

京都大学

TCP congestion control: additive increase multiplicative decrease

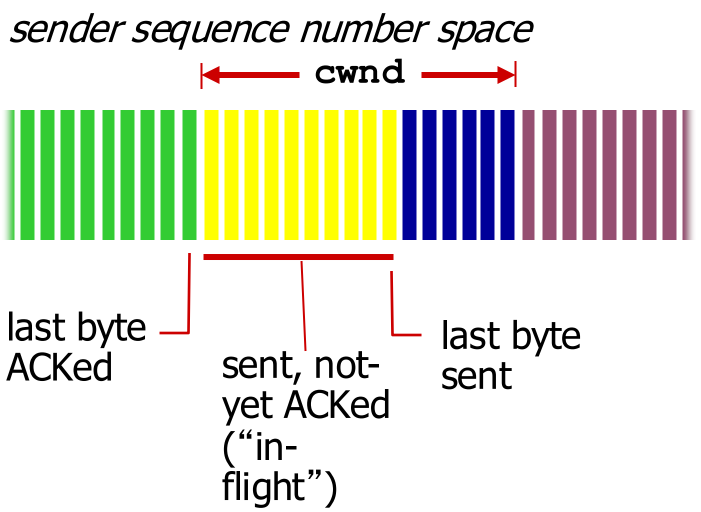
- *approach:* sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **cwnd** by 1 MSS every RTT until loss detected
  - *multiplicative decrease:* cut **cwnd** in half after loss

AIMD saw tooth behavior: probing for bandwidth



京都大学

TCP Congestion Control: details



- sender limits transmission:

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

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

TCP sending rate:

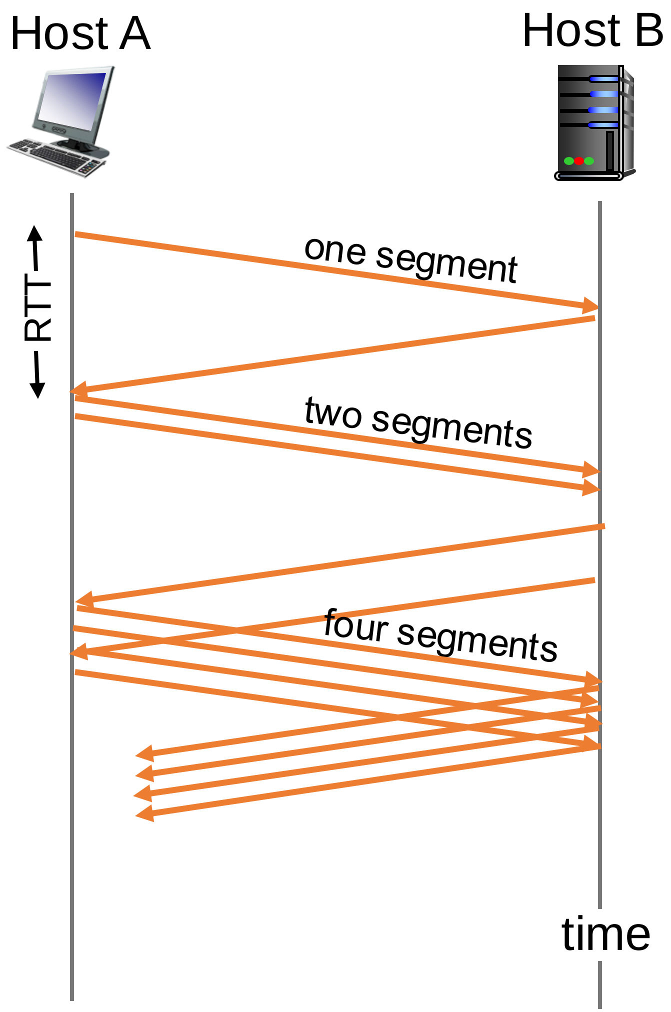
- *roughly:* send cwnd bytes, wait RTT for ACKS, then send more bytes

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

京都大学

TCP Slow Start

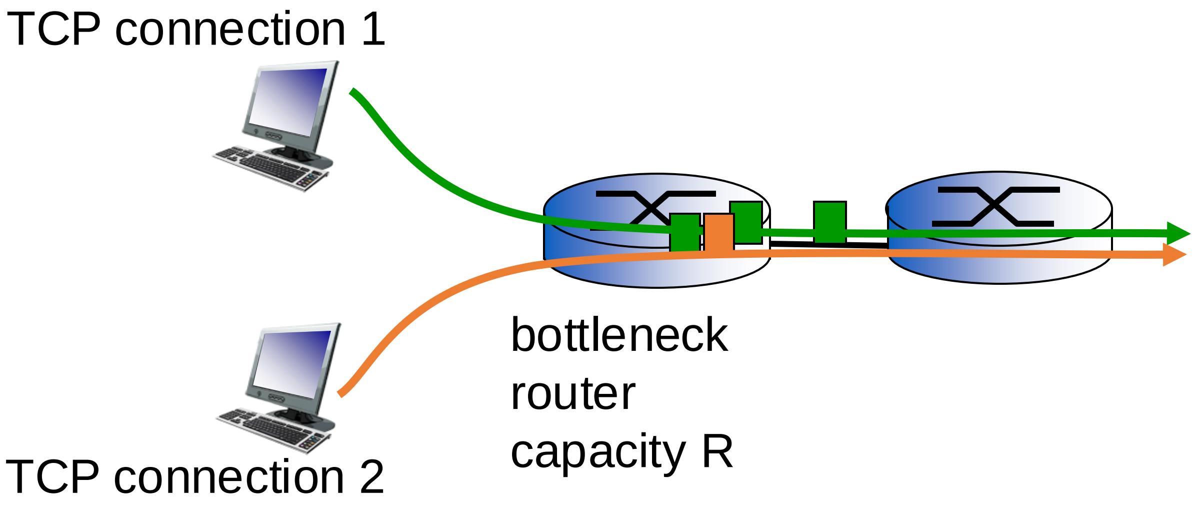
- when connection begins, increase rate exponentially until first loss event:
  - initially **`cwnd`** = 1 MSS
  - double **`cwnd`** every RTT
  - done by incrementing **`cwnd`** for every ACK received
- *summary:* initial rate is slow but ramps up exponentially fast



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

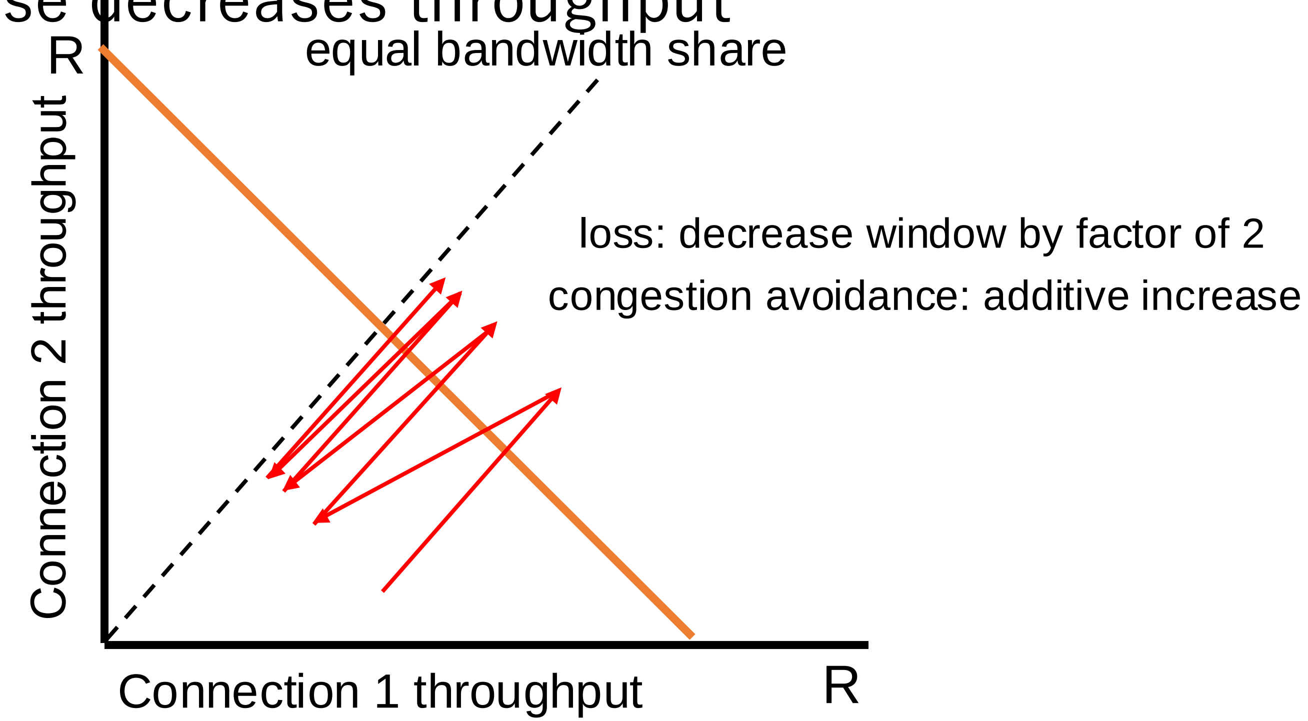


京都大学

Why is TCP fair?

two competing sessions:

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



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