CSC4200/5200 – COMPUTER NETWORKING

Instructor: Susmit Shannigrahi

TRANSPORT LAYER PROTOCOLS

sshannigrahi@tntech.edu

GTA: dereddick42@students.tntech.edu

Tennessee TECH

Apps (HTTP)

Data

Apps (HTTP)

Transport (TCP/UDP)

Segments

Transport (TCP/UDP)

Network (IP)

Packets

Network (IP)

Link (Ethernet)

Frames

Link (Ethernet)

Ethernet Interface

Ethernet Interface

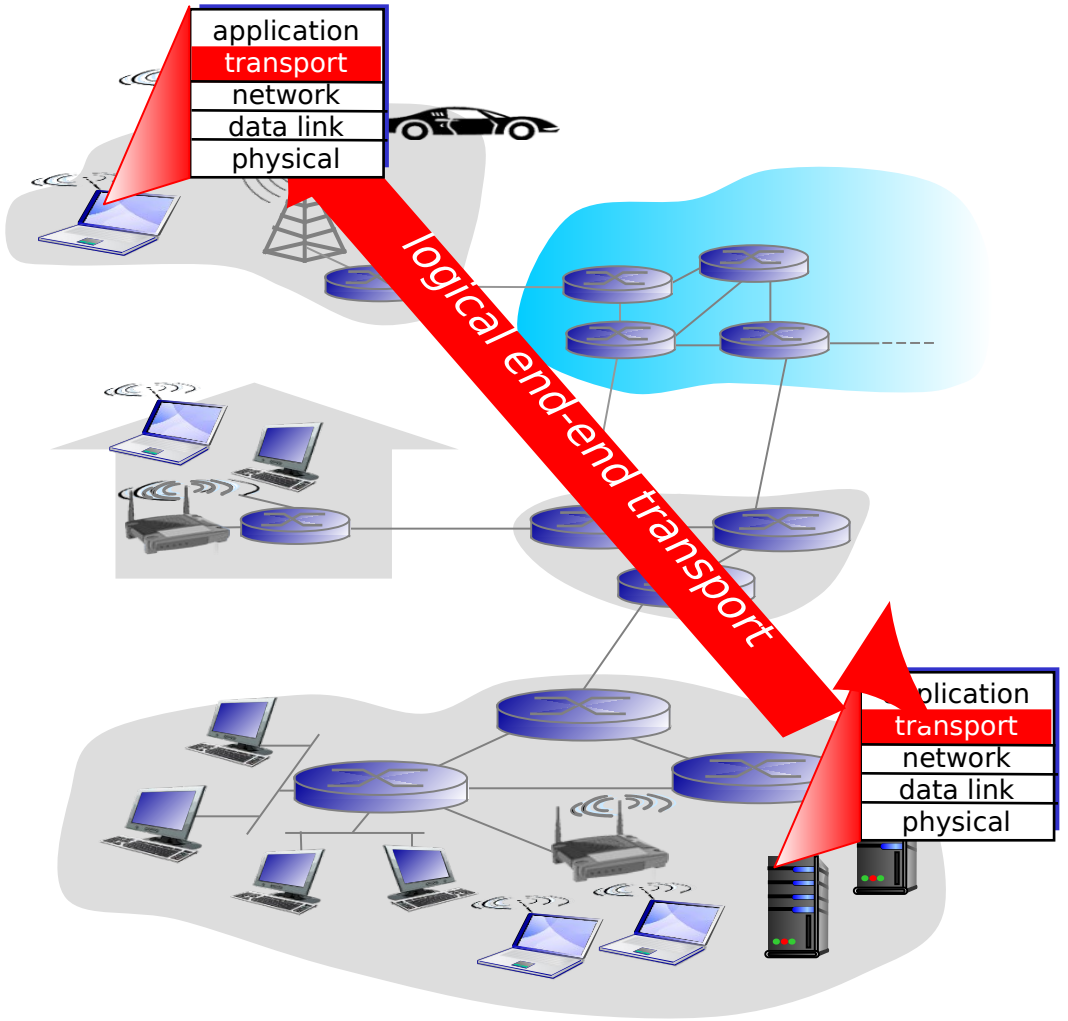
Bits (1010001)

What is transport layer?

- Problem: How to turn this host-to-host packet delivery service into a process-to-process communication channel?

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 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

Transport vs. network layer

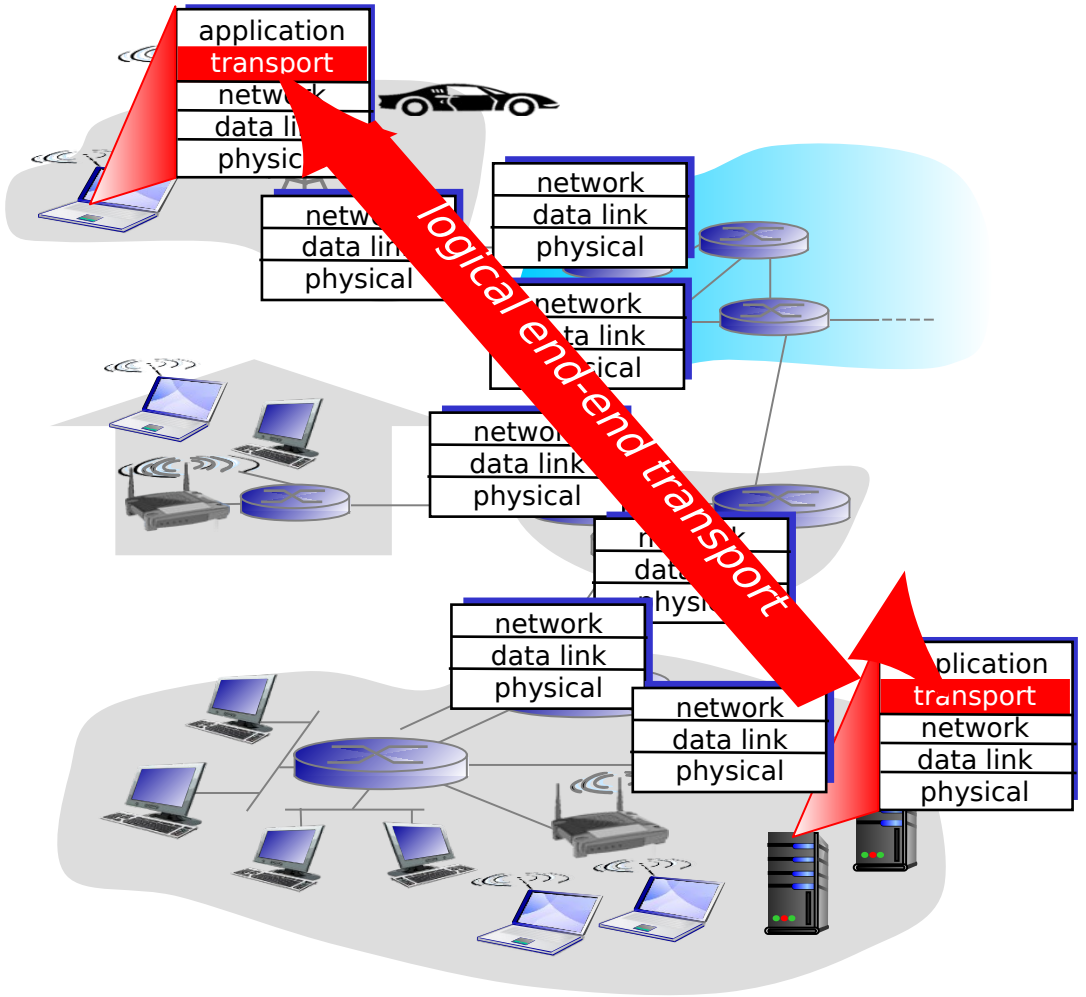
- network layer: logical communication between hosts
- transport layer: logical communication between processes
  - relies on, enhances, network layer services

Internet transport-layer protocols

Reliable, in-order delivery (TCP)

- congestion control
- flow control
- connection setup

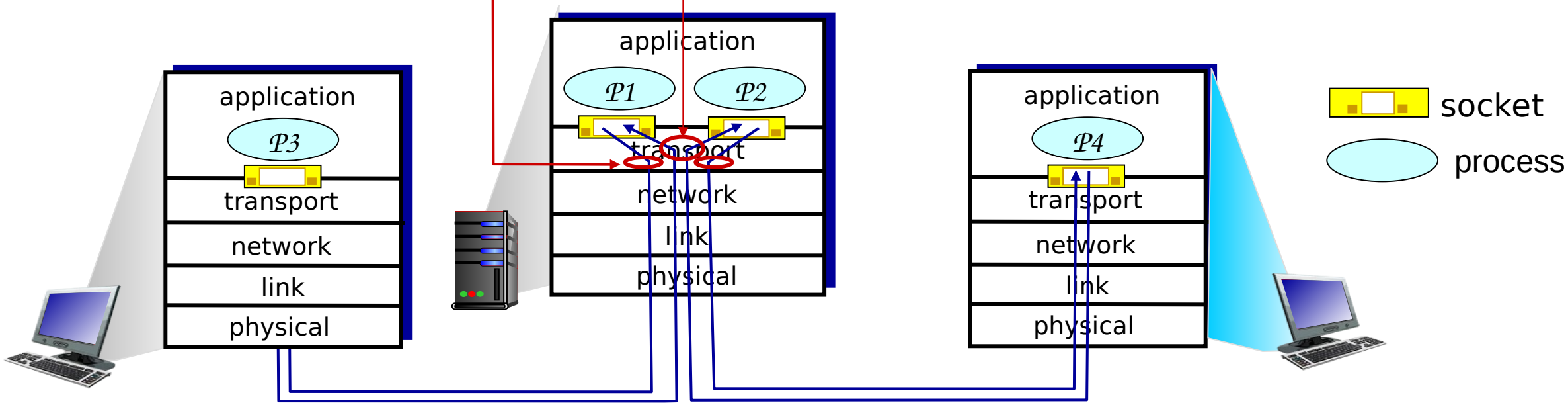
- unreliable, unordered delivery: UDP
  - no-frills extension of “best-effor t” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



Multiplexing/demultiplexing

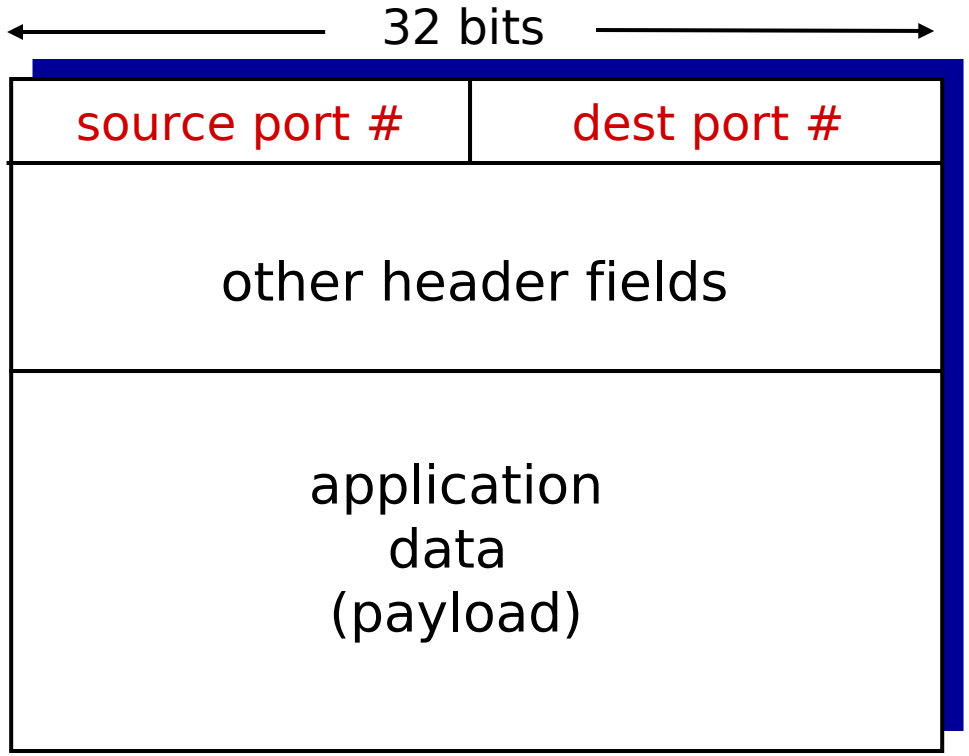
multiplexing at sender:
handle data from multiple sockets, add transport header (later used for demultiplexing)

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



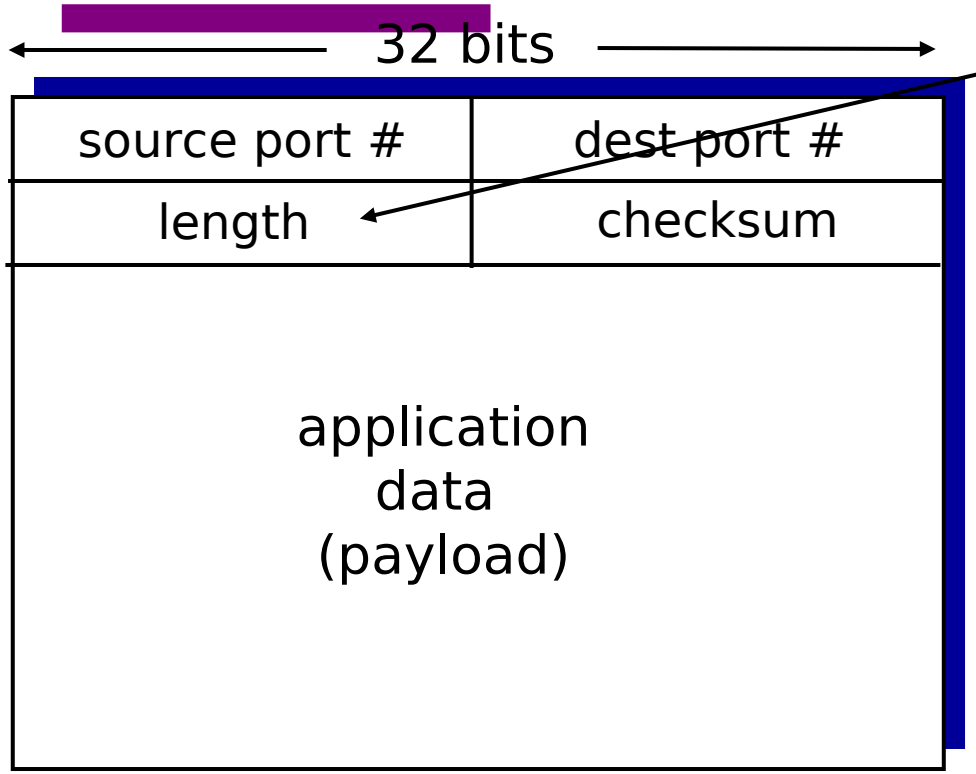
TCP/UDP segment format

UDP: User Datagram Protocol

- [redacted]
- Lightweight communication
  - Avoid overhead and delays of ordered delivery
  - Send messages to and receive them from a socket

- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

UDP: segment header



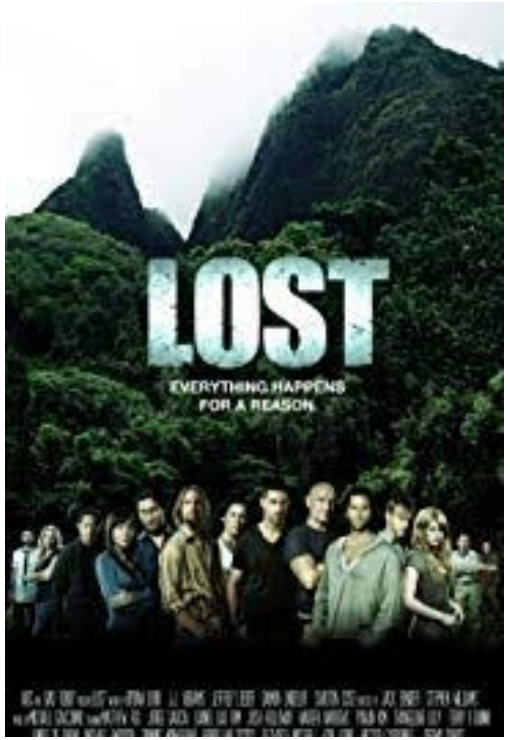
UDP segment format

length, in bytes of UDP segment, including header

Why would anyone use 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

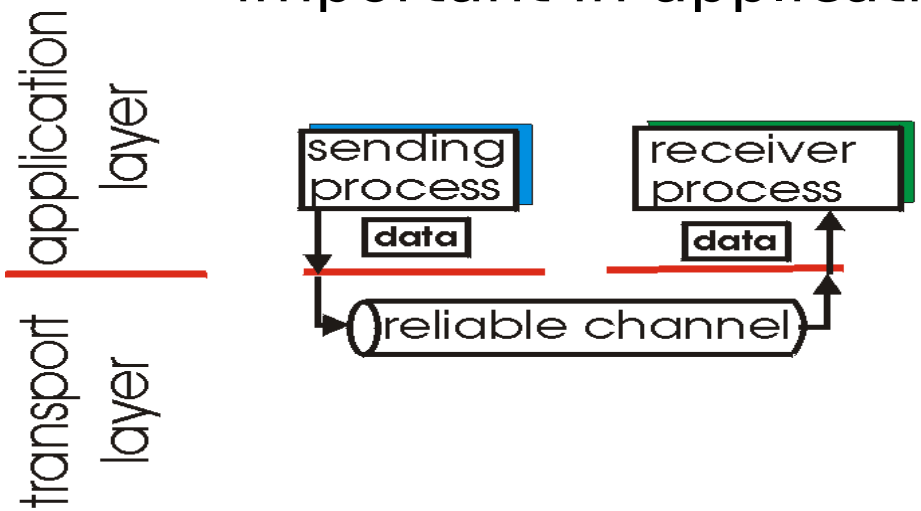
Who uses UDP?



- Multimedia applications
  - Sending a lost frame is not worth it
  - By the time the packet is retransmitted, it’s too late
- DNS
  - Small query
  - Connection establishment might be an overkill

Principles of reliable data transfer

- important in application, transport, link layers

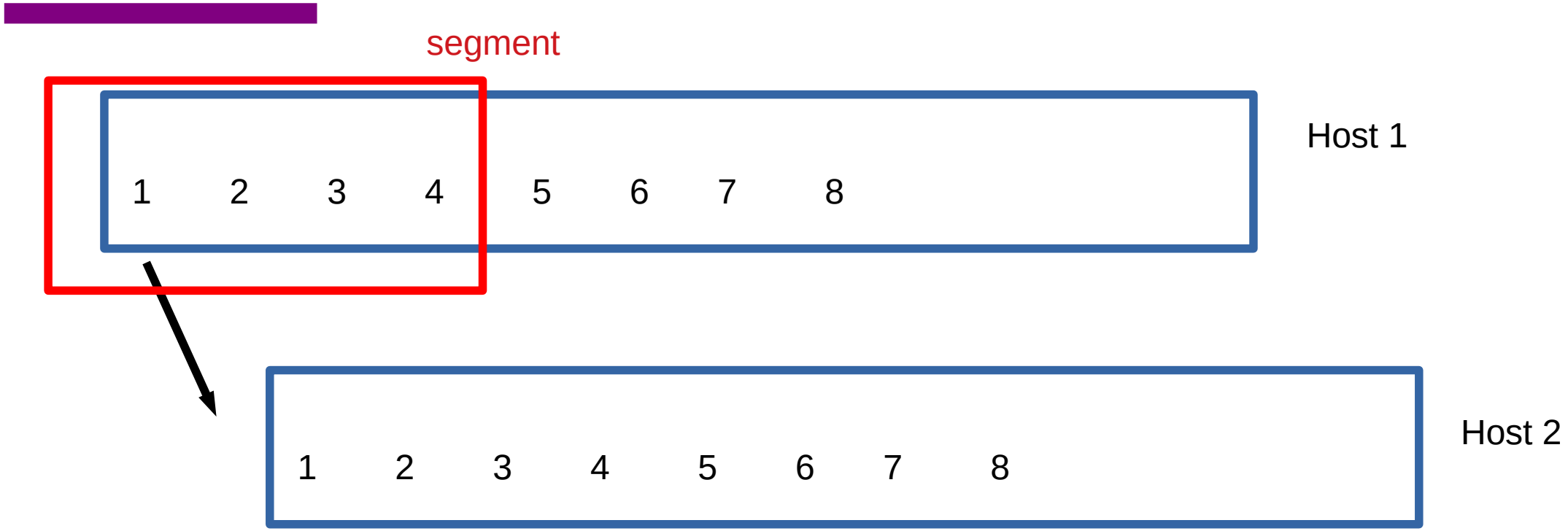


(a) provided service

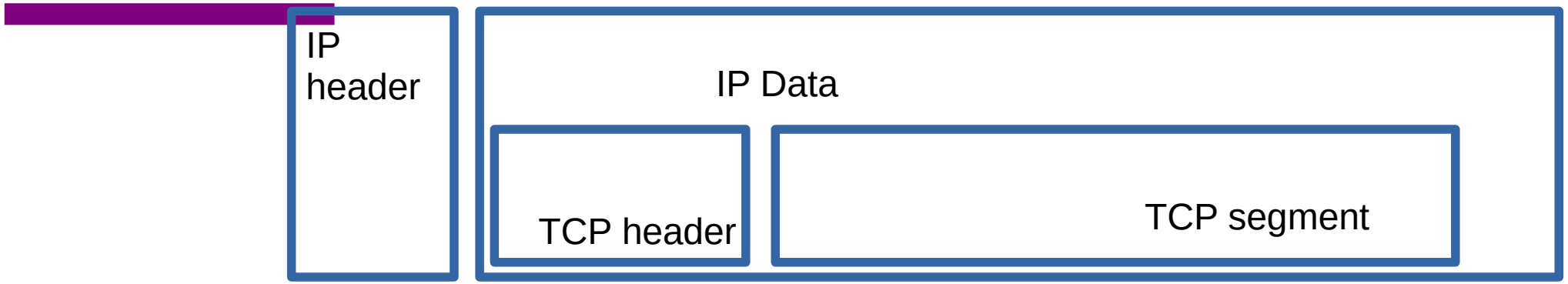
TCP – Transmission Control Protocol

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

TCP – Transmission Control Protocol



TCP Segment



IP → No more than MTU (1500 Bytes)

TCP header → 20 bytes

TCP segment → 1460 bytes

Why?

TCP Header

SYN

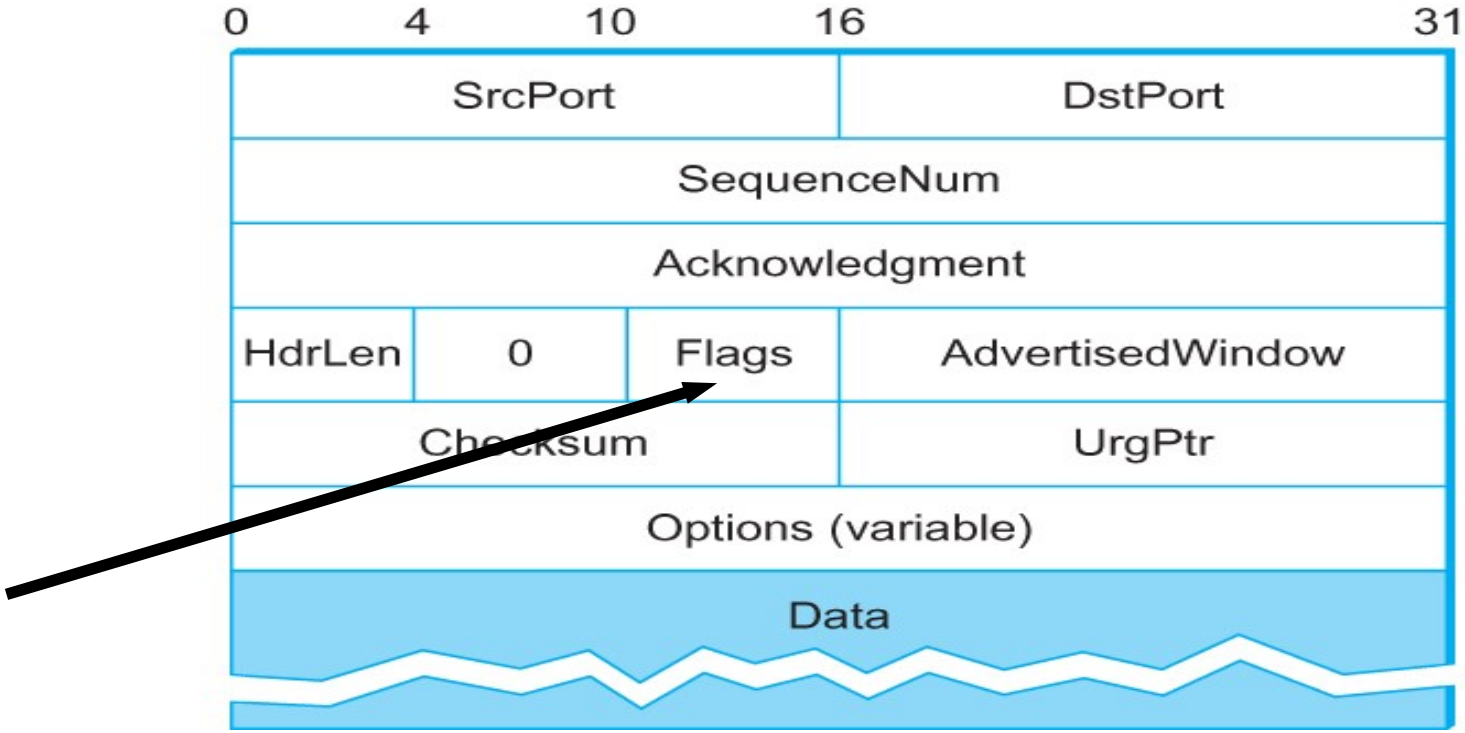
FIN

RST

PSH

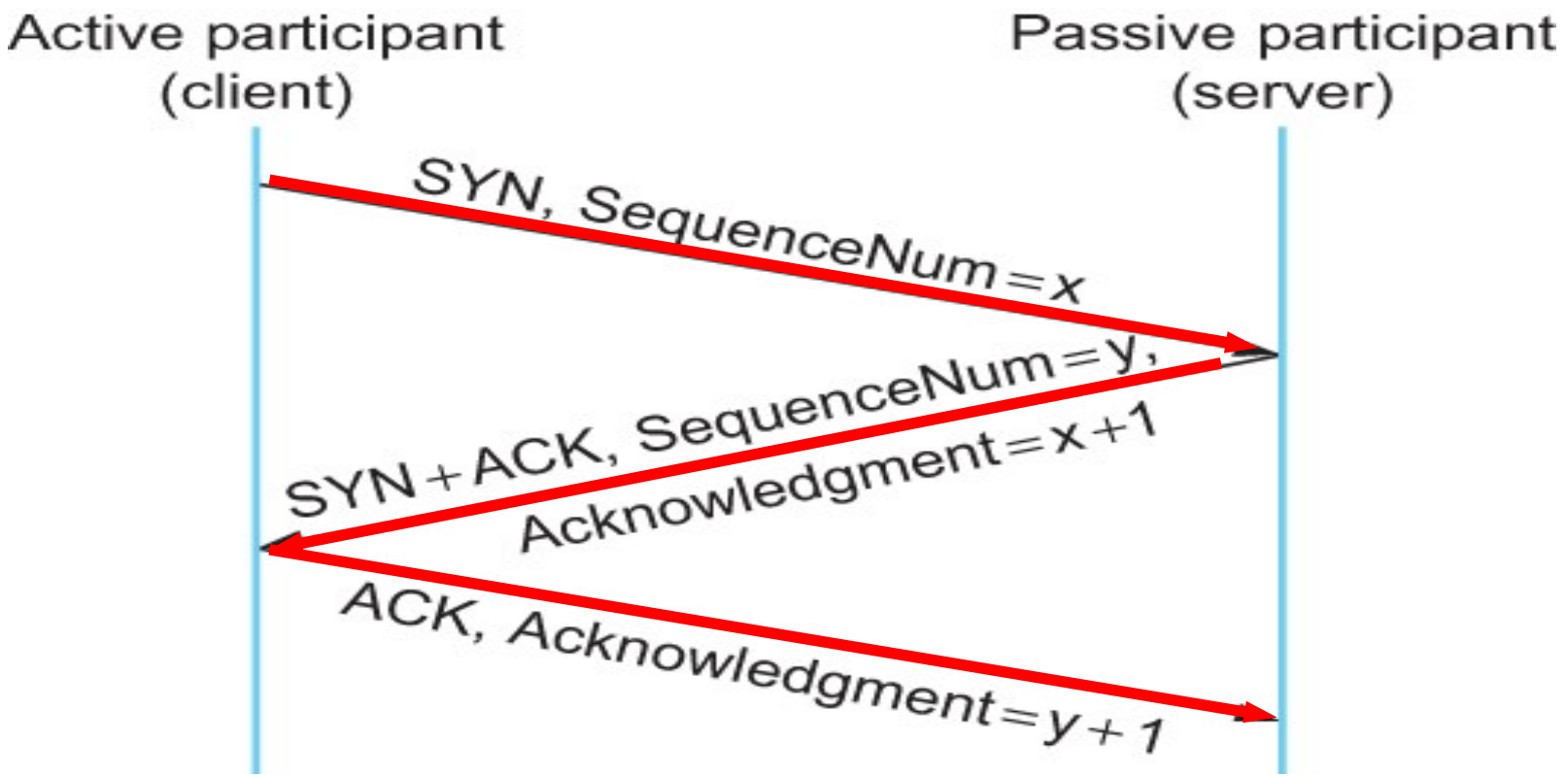
URG

ACK



TCP Header Format

TCP Three-way Handshake

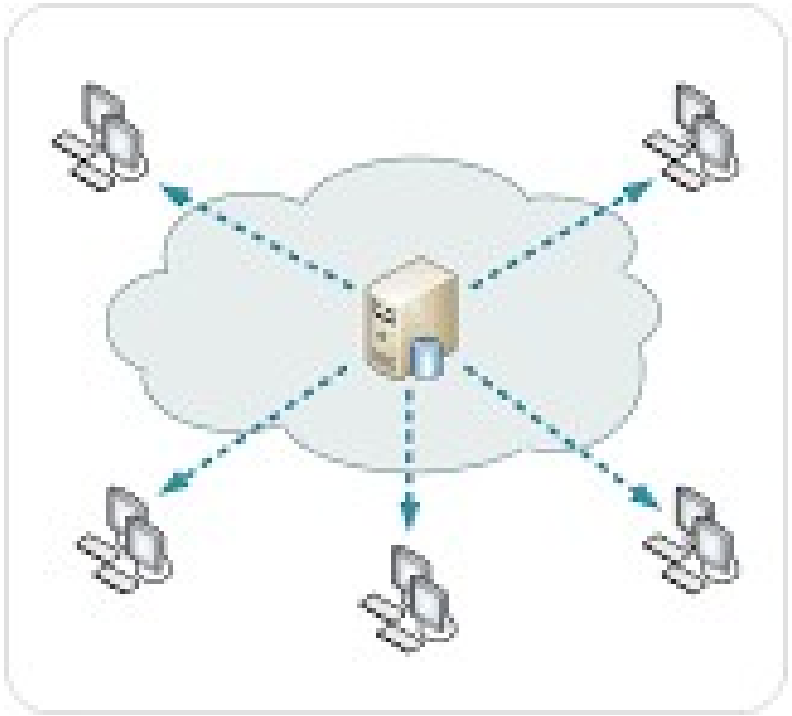


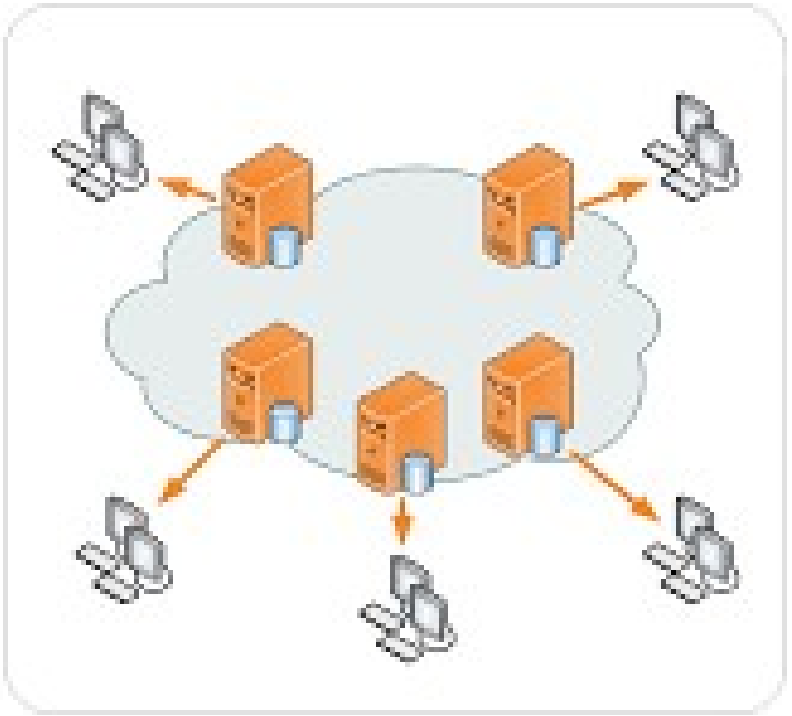
Timeline for three-way handshake algorithm

TCP – Transmission Control Protocol

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

CDNs





Next steps

Read through Chapter 5.2.3

More TCP

Programming assignment 3 – Read the article on content delivery network on WikiPedia