IOWA

CS3640

Transport Layer (1): Multiplexing and UDP

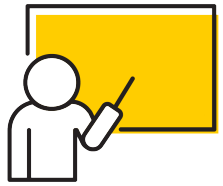
Prof. Supreeth Shastri
Computer Science
The University of Iowa

Midterm Format and Question

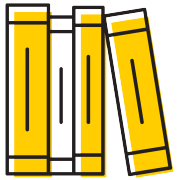
Category	Example questions and topics	Weight
Networking **Principles**	*End-to-end argument; Routing vs. forwarding; Protocol layering*	25%
Networking **Protocols**	*TCP vs. UDP; HTTP headers and extensions; Designing CDNs*	25%
Networking **Problems**	*Understanding delays; Mitigating congestion; Security challenges*	25%
Network **Programs**	*Explain how traceroute works; Socket programming; Video Streaming*	25%

*There will be an optional **bonus question** carrying 10% extra points (expect it to be challenging)*

Preparations and logistics



*Revisit the **lectures and slides**:*
https://shastri.info/teaching/cs3640



*Read the **textbook**:*
Kurose-Ross chapters 1-3

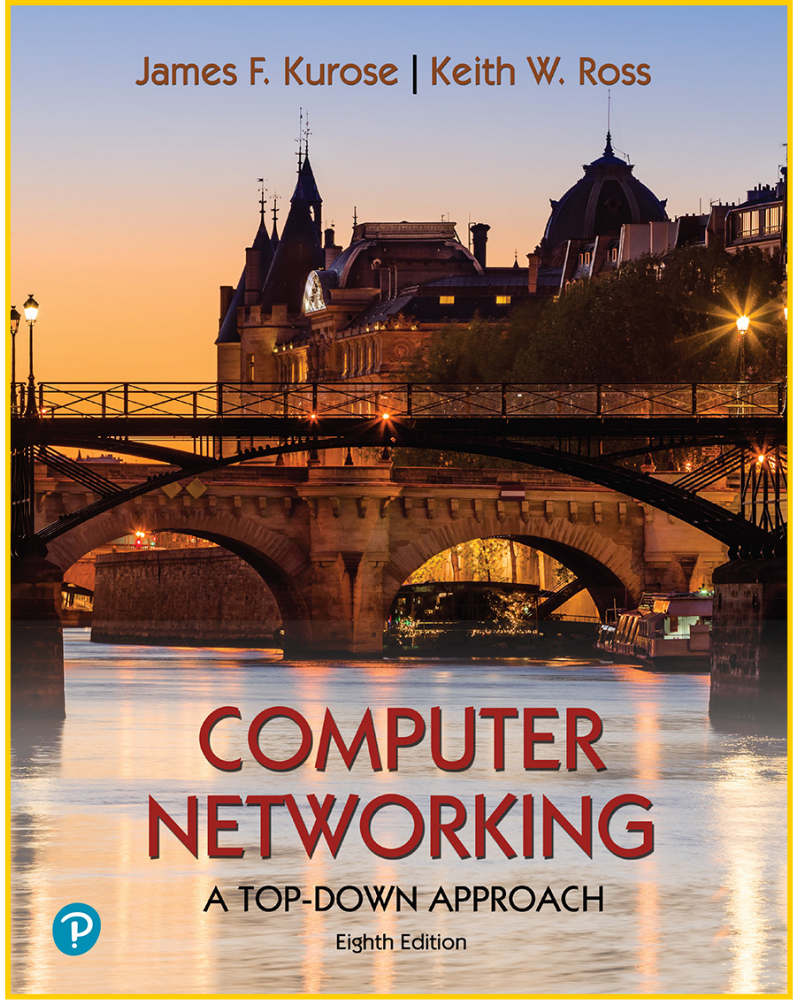
*Midterm **schedule**:*
*3/9 Thursday at 6:30PM in **100 PH***

It is a 1-hour pen-and-paper exam (closed book, closed notes, closed electronics)

Lecture goals

introduction to key transport layer services, and an overview of UDP

- *Transport-layer services*
- *Multiplex and demultiplex*
- *UDP*



Chapter 3.1 - 3.3

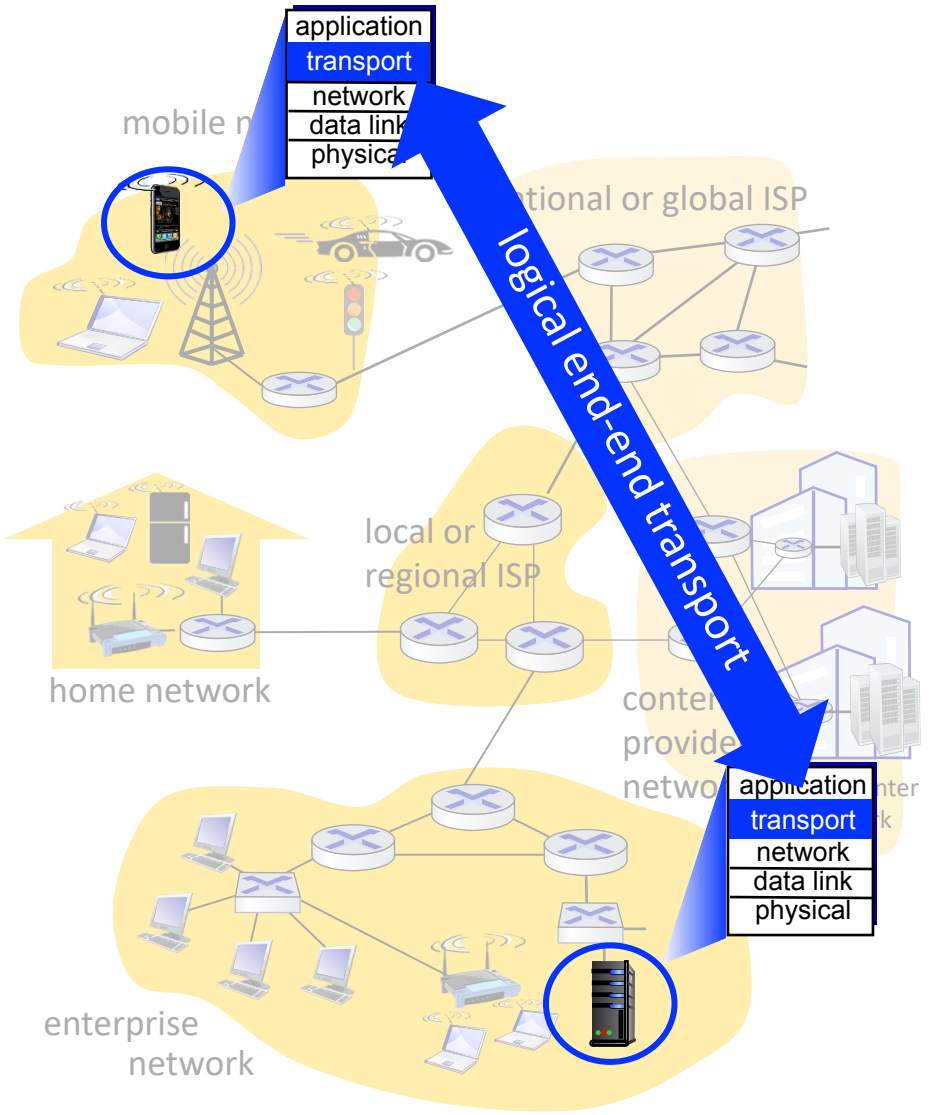
IOWA

Transport-layer services

Key goal: to provide logical communication between application processes running on different hosts

Only end-systems need transport layer

- sender: breaks application messages into segments, and passes to the network layer
- receiver: reassembles received segments into messages, passes to the application layer



Transport vs. network layer services

Network layer: logical communication b/w **network hosts**

Transport layer: logical communication b/w **processes**

a household analogy

Kids in Alice's house exchanging letters with kids in Bob's house

- houses == hosts
- kids == processes
- letters in envelopes == application messages
- postal service == network layer
- Alice and Bob == transport layer (mux-demux)

Transport-layer services of the Internet

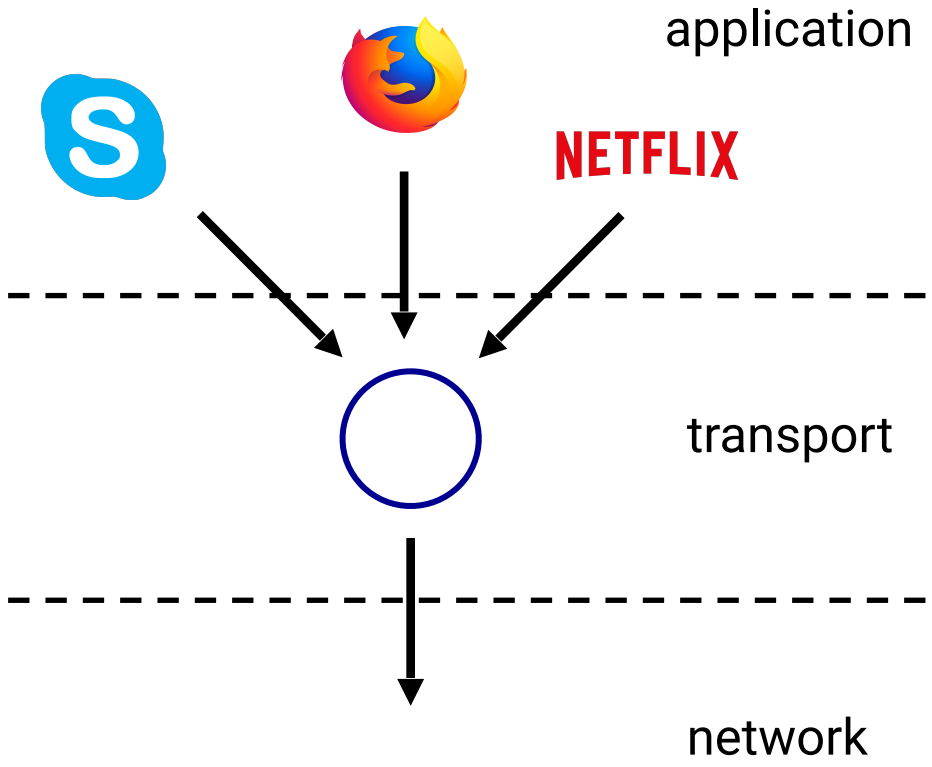
	TCP	UDP
multiplexing and demultiplexing	✔	✔
reliable data transfer	✔	🚫
flow control	✔	🚫
congestion control	✔	🚫
timing and delay guarantees	🚫	🚫
throughput guarantees	🚫	🚫

Q: why this particular choice of services by each protocol?

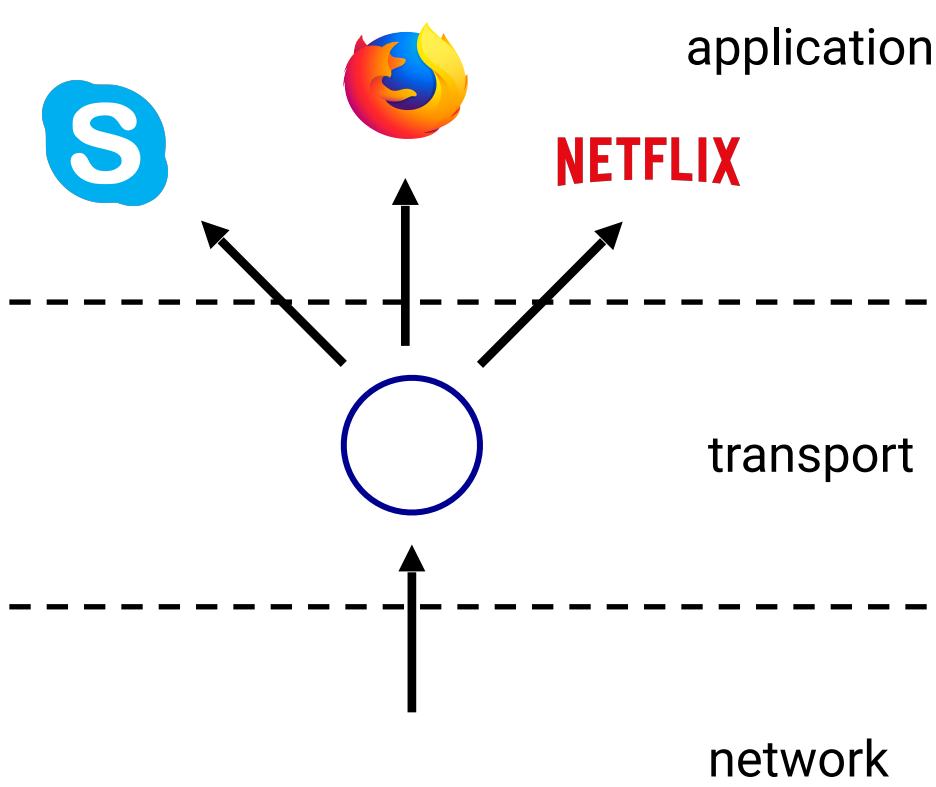
Q: are these even possible on the Internet?

Multiplexing / **Demultiplexing**

multiplexing



demultiplexing

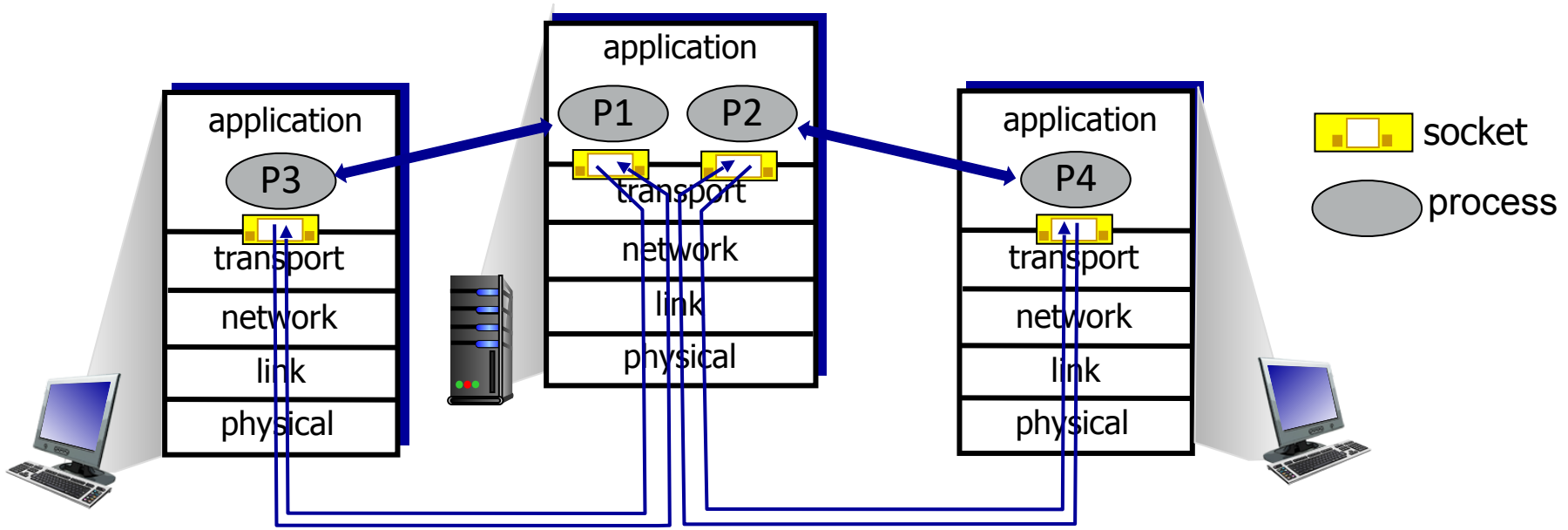


multiplexing at sender

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



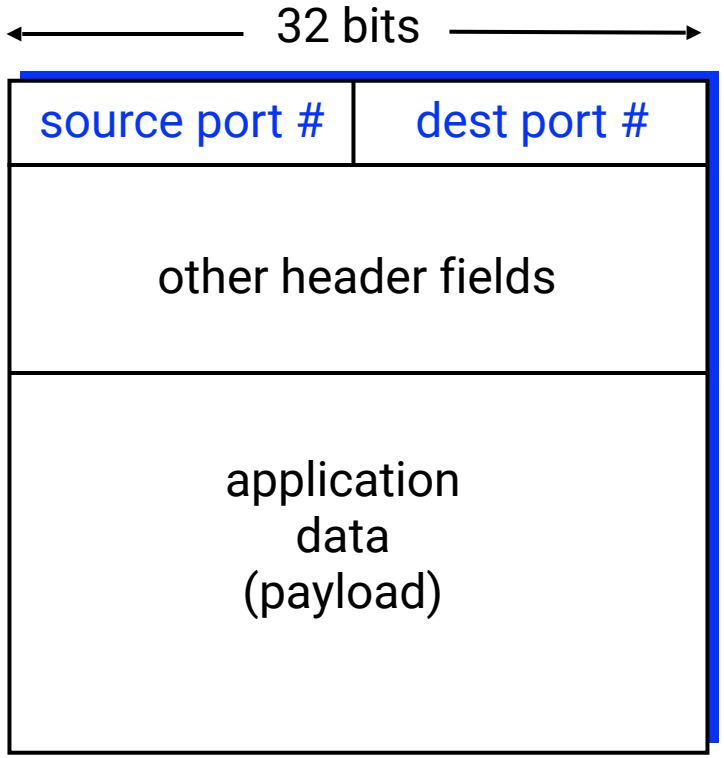
How demultiplexing works

1. host receives IP datagrams

- each datagram has *source IP address*, *destination IP address*, and carries one transport-layer segment
- the transport segment contains a *source port number* and *destination port number*

2. host directs the TCP/UDP segment to the appropriate socket

- based on the combination of IP addresses and port numbers



TCP/UDP segment format

Connectionless demultiplexing

1. **when creating UDP socket, the application must specify the local port number**
   - DatagramSocket mySocket = new DatagramSocket(12534);

2. **when sending a segment into UDP socket, the host must specify**
   - destination IP address, destination port number

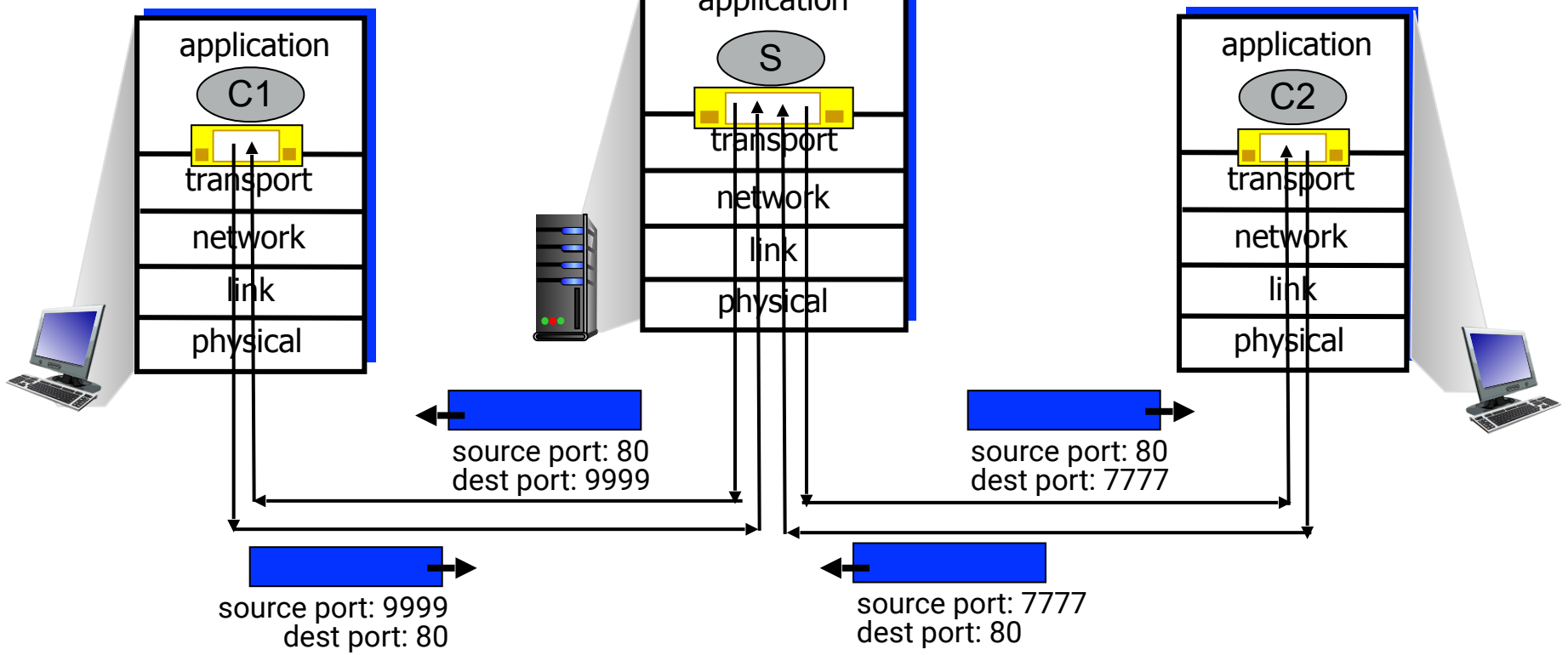
3. **when a host receives a UDP segment, it will**
   - check destination port number, and direct it to the socket bound to that port
   - *demux is not dependent on either source IP address or source port number*

Connectionless demultiplexing

DatagramSocket C1
= new DatagramSocket (9999);

DatagramSocket S
= new DatagramSocket (80);

DatagramSocket C2
= new DatagramSocket (7777);



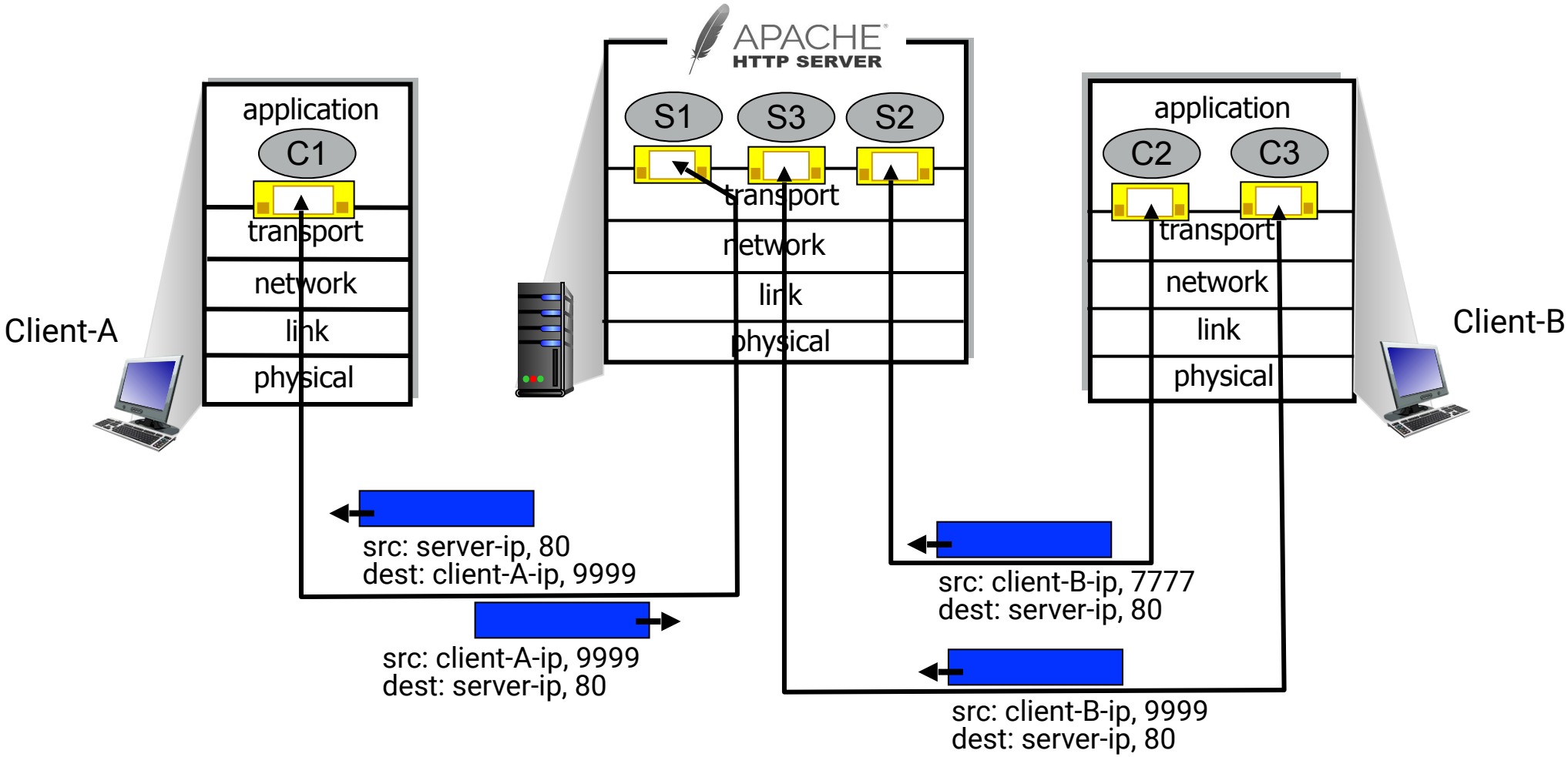
Connection-oriented demultiplexing

1. **connection-oriented sockets (TCP) are identified by the 4-tuple**
   - src IP address, src port number, dest IP address, dest port number
   - this is done via connect() and accept() socket calls

2. **when host receives a TCP segment, it will**
   - use all four values to direct the segment to the appropriate socket

3. **server may support multiple simultaneous TCP connections, where**
   - each socket is uniquely identified by its own four tuple
   - each socket is associated with a different client application

Connection-oriented demultiplexing



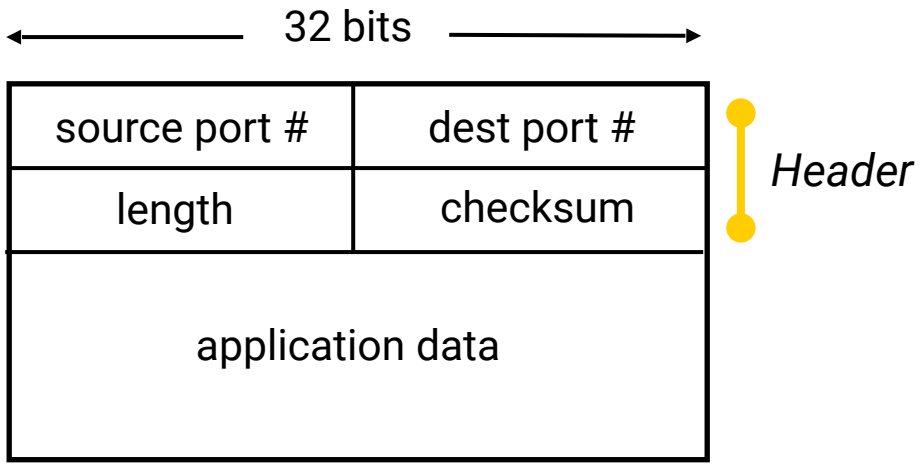
Three segments, all destined to <server-IP, 80> are demultiplexed to ***different*** sockets at the server

UDP

User Datagram Protocol

- a bare-bones datagram transport over IP
- defined in an RFC that is ~500 words
- service offered: mux-demux, error detection

UDP segment format



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].

Why UDP even exists?

connection-less	reduces latency and RTT delays
stateless client and server	simpler implementation
no reliability or in-order delivery	less overhead (due to small header size)
no flow control	can send data as fast as desired
no congestion control	can function in the face of congestion

some applications **do not need** these services (*e.g., video streaming needs a simple low latency transport*)

some applications **cannot incur** the associated **overhead** (*e.g., SNMP needs to operate even during network congestions*)

some applications can **efficiently implement** missing services by themselves (e.g., HTTP/3 shifted to UDP after absorbing many of the TCP features)

UDP checksum

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

sender

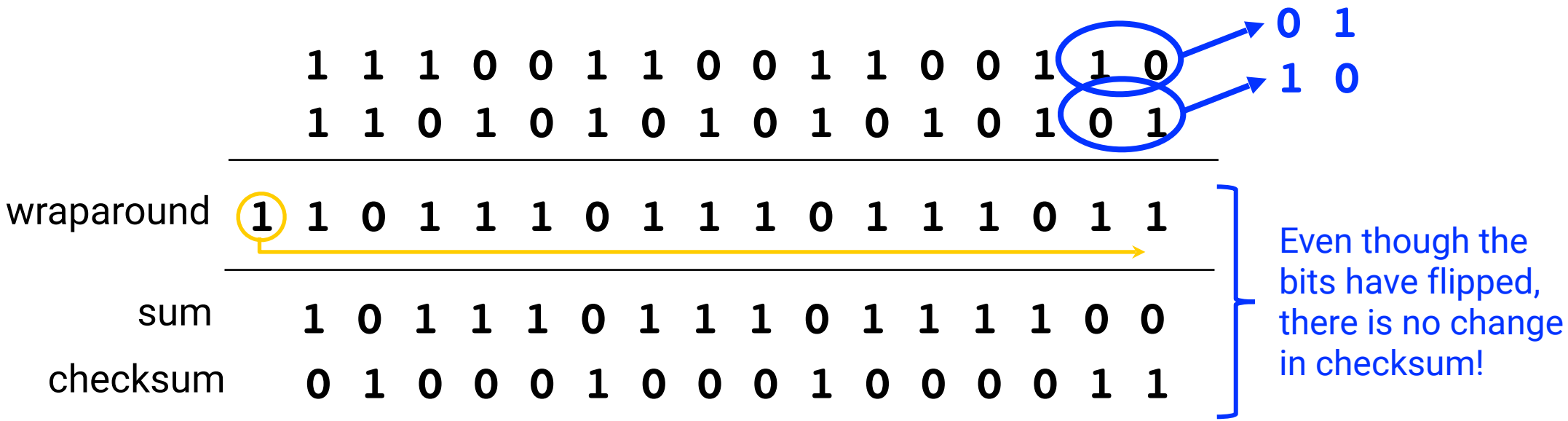
- treat contents of UDP segment as a sequence of 16-bit words
- add all the words (wrapping around any overflows) to produce a 16-bit result
- compute the checksum: by taking one's complement of the result
- put this value into UDP checksum field

receiver

- add all the 16-bit words including the checksum
- if no errors are introduced, then the sum will come out to be 1111 1111 1111 1111
- any zero in the sum indicates bit error(s)

UDP checksum shortcomings

example: add two 16-bit integers

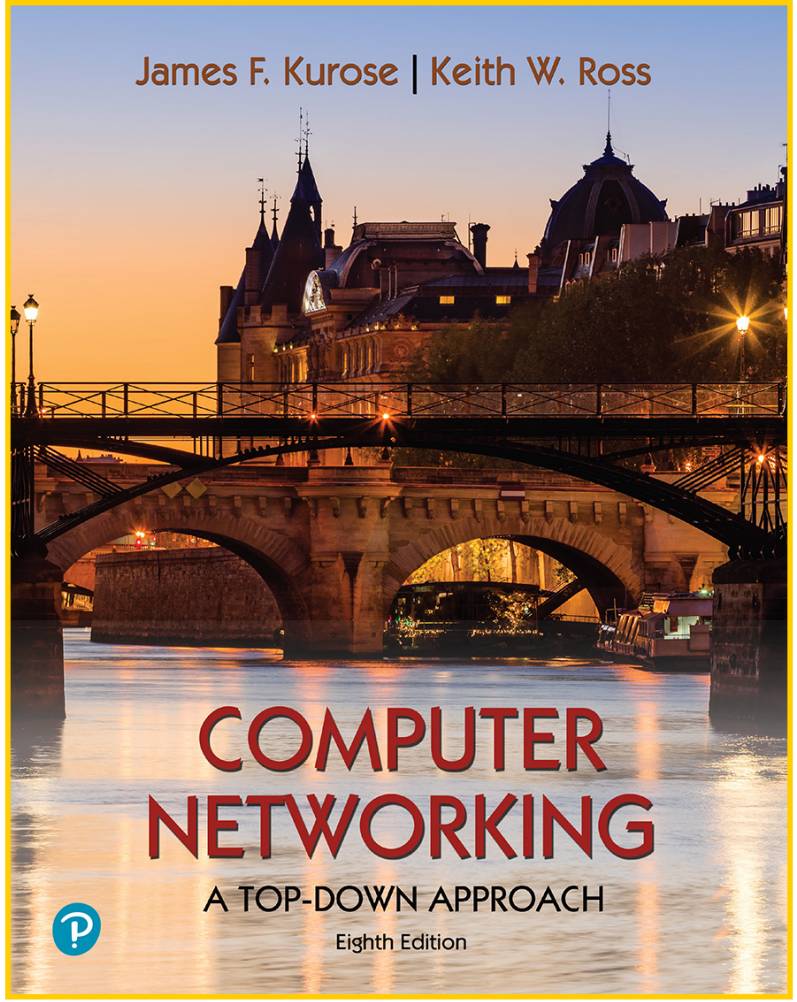


- This is a weak protection i.e., it can only detect single-bit errors
- Even when detected, UDP has no ability to correct the error
- UDP checksum is optional (if unused, set all the checksum bits to zero)

Next lecture

how to achieve reliable data transport over an unreliable network such as the Internet?

- *Challenges and techniques*
- *Go-Back-N*
- *Selective Repeat*



Chapter 3.4

Spot Quiz (ICON)