Introduction to Computer Networks

CS640

Transport Introduction

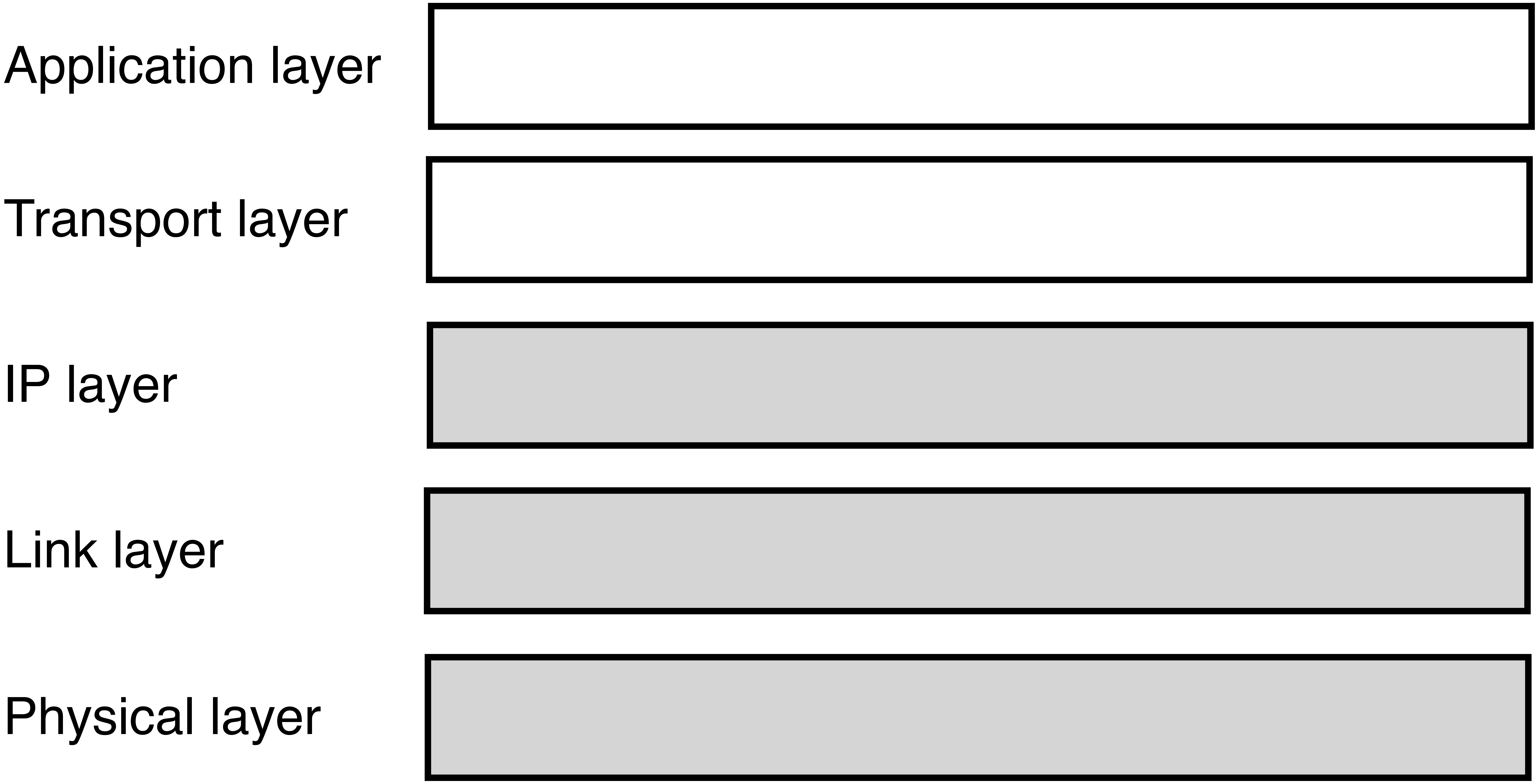
https://pages.cs.wisc.edu/~mgliu/CS640/S25/index.html

Ming Liu
mgliu@cs.wisc.edu

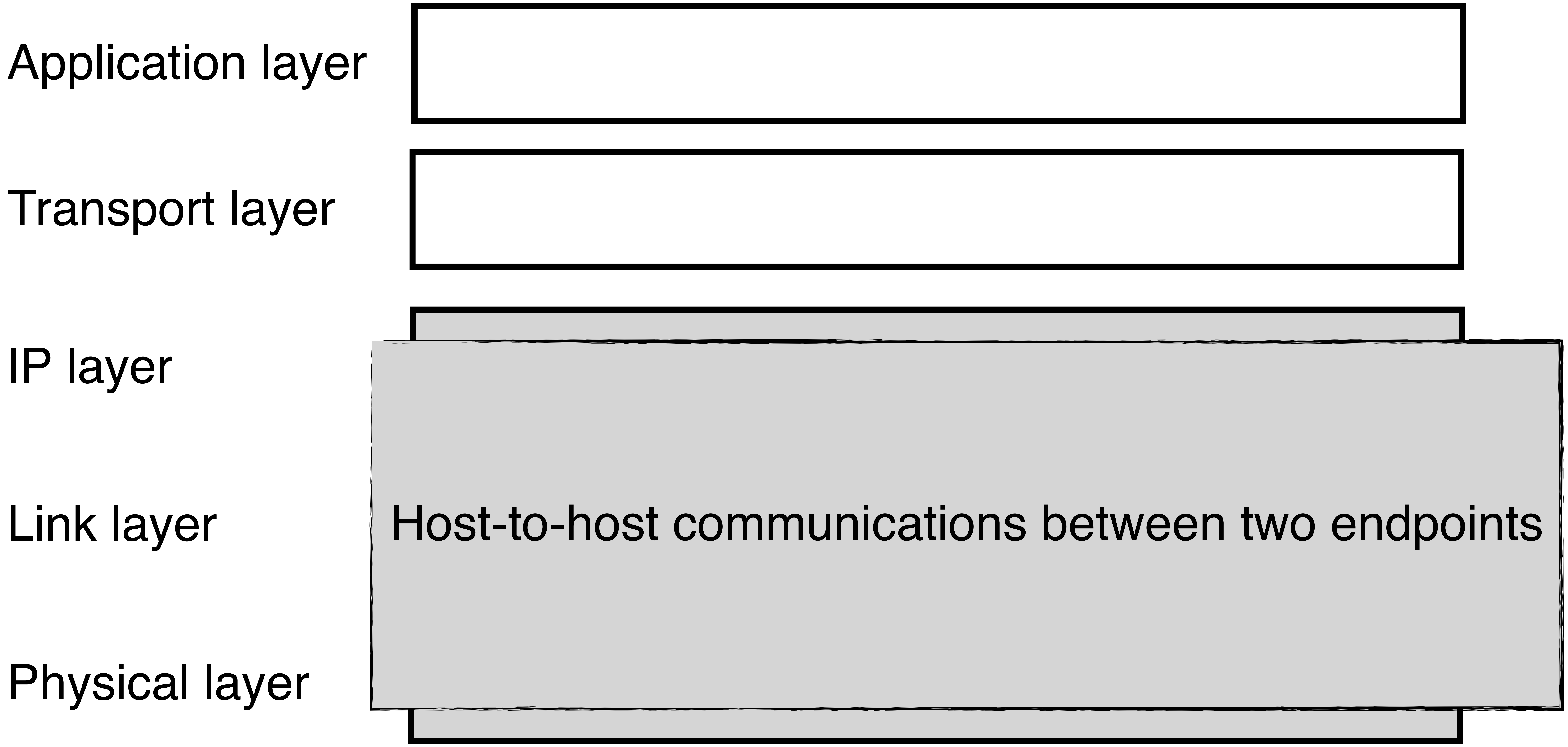
Outline

- Last
  - NAT, IPv6, and Multicast
- Today
  - Transport Introduction
- Announcements
  - Lab3 due on 04/01/2025 12:01PM

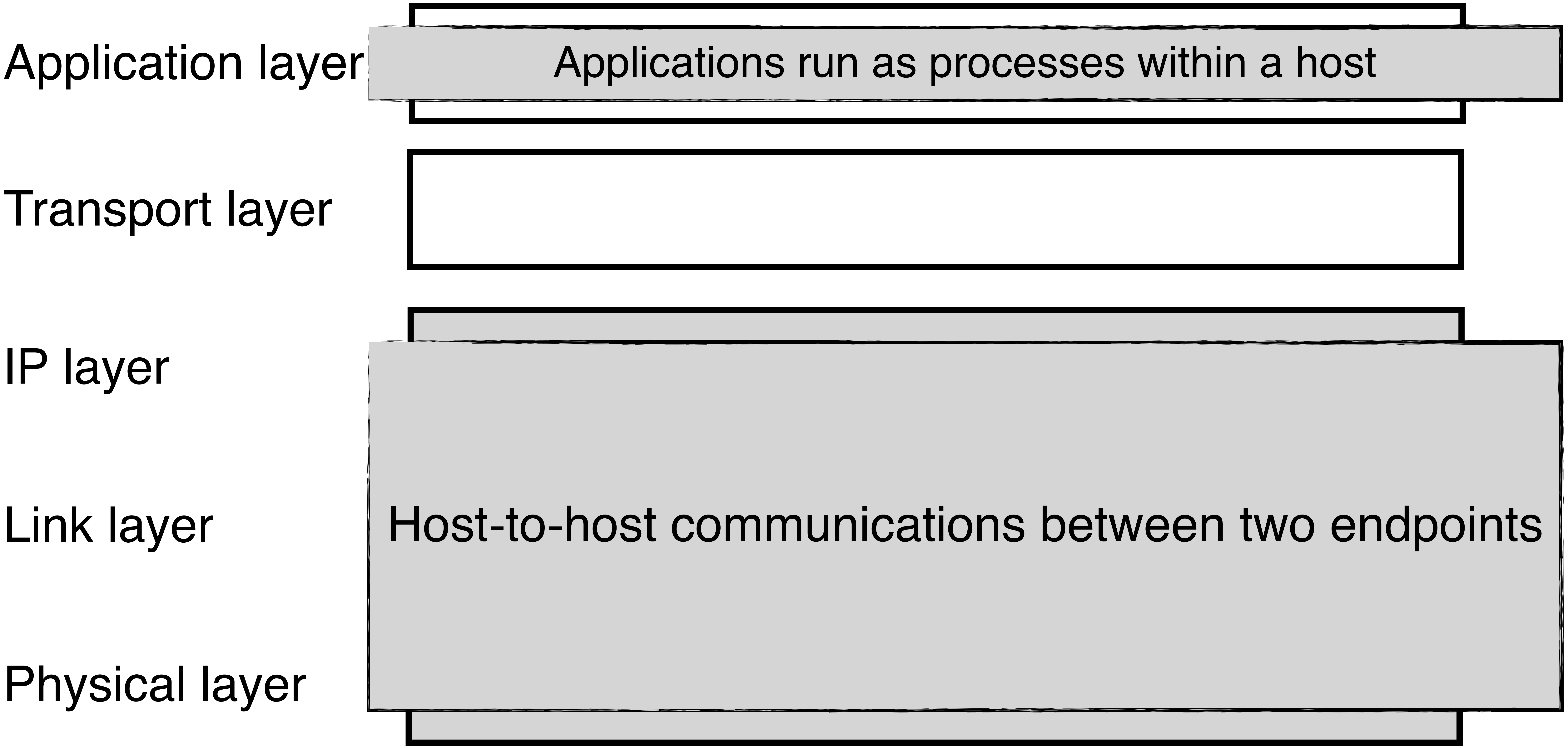
Transport Layer in the TCP/IP Model



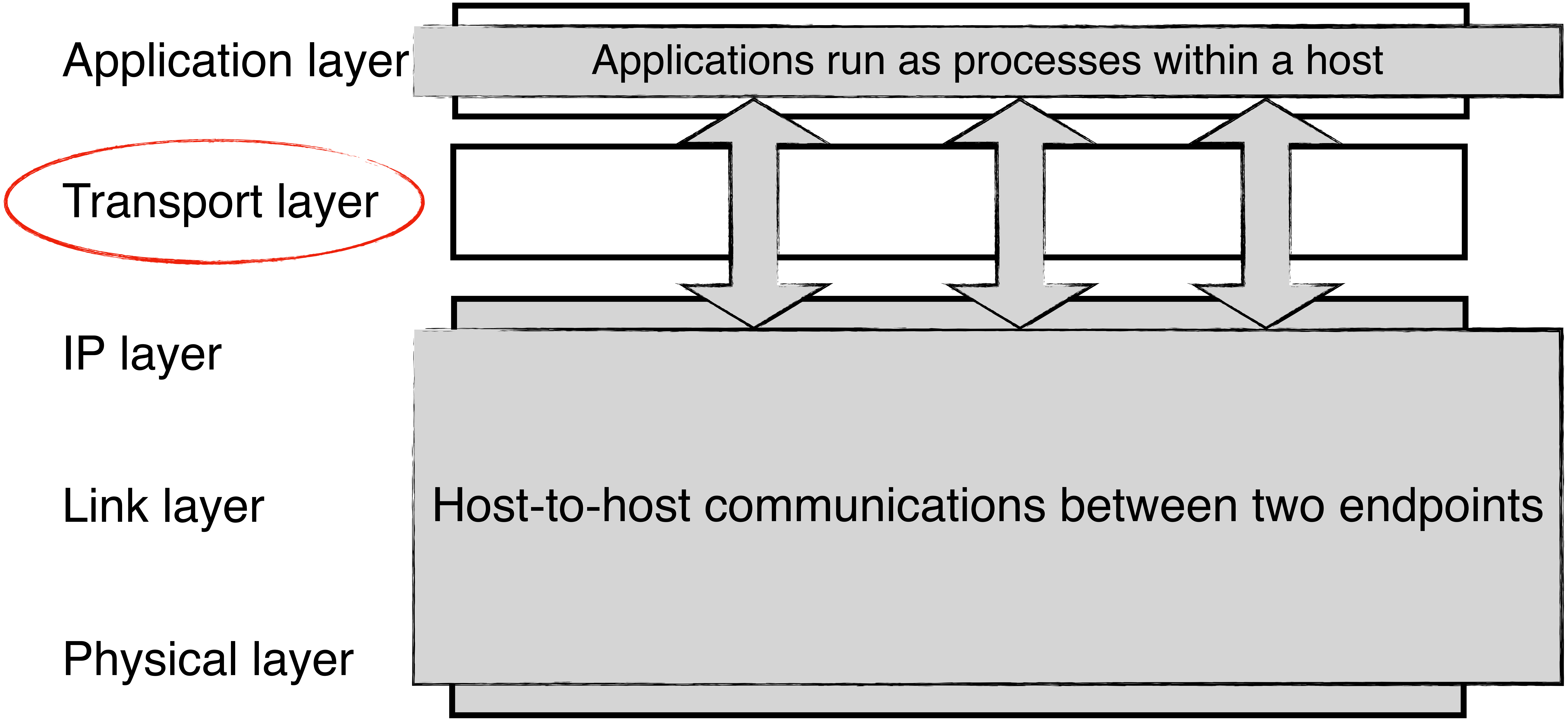
Transport Layer in the TCP/IP Model



Transport Layer in the TCP/IP Model



Transport Layer in the TCP/IP Model



What functionalities does the transport layer provide?

What functionalities does the transport layer provide?

Process-to-process communication channels

System Model

- Design requirements
  - Support arbitrary large messages
  - Support multiple application processes on a host (multiplexing)
  - Support message delivery with certain guarantees
    - Packet order
    - Exact one copy
    - ……

System Model

- Design requirements
  - Support arbitrary large messages
  - Support multiple application processes on a host (multiplexing)
  - Support message delivery with certain guarantees
    - Packet order
    - Exact one copy
    - ……

- Limitations
  - Fixed-sized socket buffer in the OS
  - Fixed-sized data transmission unit in the network
  - Computing and communication entities run at different speeds

System Model

- Design requirements
  - Support arbitrary large messages
  - Support multiple application processes on a host (multiplexing)
  - Support message delivery with certain guarantees
    - Packet order
    - Exact one copy

Challenge: The underlying network (IP) layer is best-effort.

  - Fixed-sized socket buffer in the OS
  - Fixed-sized data transmission unit in the network
  - Computing and communication entities run at different speeds

What functionalities does the transport layer provide?

Process-to-process communication channels

Q1: How can we set up the process-to-process channel?
Q2: How can we multiplex concurrent channels over the physical link?
Q3: How can we control the transmission rate?
Q4: How can we achieve reliable delivery?
Q5: How can we share the in-network bandwidth resources?

User Datagram Protocol (UDP)

- Extend the IP service model to the process-to-process channel
  - Best-effort
  - Unreliable and unordered datagram service

User Datagram Protocol (UDP)

- Extend the IP service model to the process-to-process channel
  - Best-effort
  - Unreliable and unordered datagram service

- UDP is a simple message-oriented transport protocol (RFC 768)
  - #1: Add multiplexing/demultiplexing
  - #2: Add reliability through optional checksum

Demultiplexing Key: Port

- Ports are numeric locators
  - Enable messages to be multiplexed to proper messages
  - Ports are addresses on individual hosts, not across the Internet

Demultiplexing Key: Port

- Ports are numeric locators
  - Enable messages to be multiplexed to proper messages
  - Ports are addresses on individual hosts, not across the Internet

How do we learn the port?

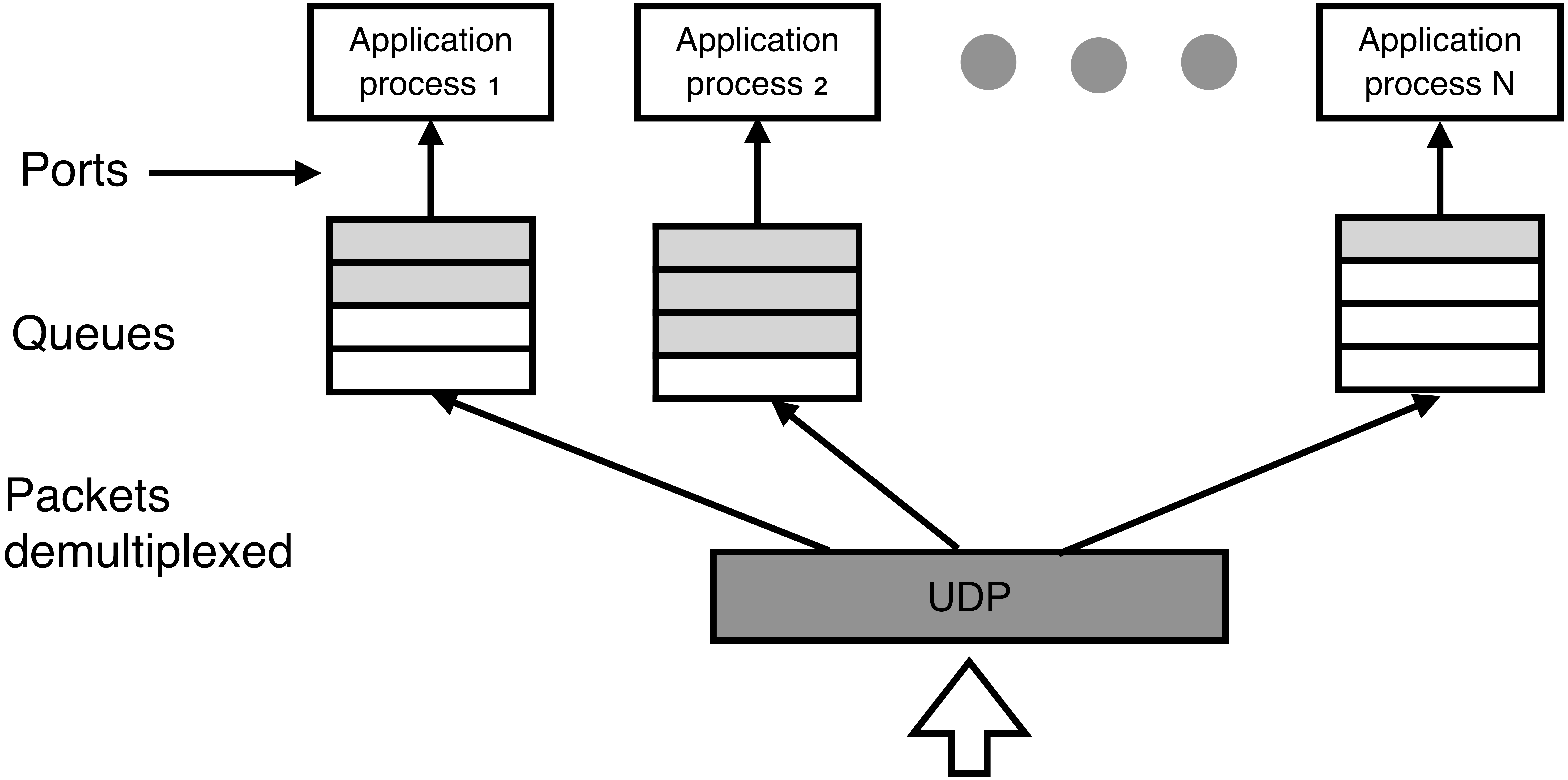
Demultiplexing Key: Port

- Ports are numeric locators
  - Enable messages to be multiplexed to proper messages
  - Ports are addresses on individual hosts, not across the Internet

- Port learning approaches:
  - #1: Servers have well-known ports
    - DNS port # = 53
    - Check /etc/services
  - #2: Port mapped service
    - Dynamically allocated

Port: A System Perspective

- Ports are implemented as message queues



UDP Header Format

0 16 31

SrcPort	DstPort
Checksum	Length

- Port: 65536 possible ports
- Length: 65535 bytes (8 bytes header + 65527 bytes data)

UDP Checksum

- Optional in the current Internet

- UDP uses the same checksum algorithm as IP
  - Internet checksum

UDP Checksum

- Optional in the current Internet

- UDP uses the same checksum algorithm as IP
  - Internet checksum

UDP checksum is computed over **pseudo header** + UDP header + data

UDP Checksum

- Optional in the current Internet

The psuedo header consists 3 fields from the IP header: protocol number (TCP or UDP), IP src, IP dst, and UDP length field

- The pseudo header enables verification that message was delivered between the correct source and destination
- IP dest address was changed during delivery, checksum would reflect this

UDP checksum is computed over **pseudo header** + UDP header + data

UDP Checksum Example

	Decimal	Binary	Hex
Source IP	192.168.0.31	1100 0000 1010 1000 0000 0000 0001 1111	C0 A8 00 1F
Destination IP	192.168.0.30	1100 0000 1010 1000 0000 0000 0001 1110	C0 A8 00 1E
UDP protocol	17	0000 0000 0001 0001	00 11
Length	10 = 8 + 2	0000 0000 0000 1010	00 0A
UDP Source Port	20	0000 0000 0001 0100	00 14
UDP Destination Port	10	0000 0000 0000 1010	00 0A
UDP Length	10	0000 0000 0000 1010	00 0A
UDP Data	“Hi”	0100 1000 0110 1001	48 69
Add			
Add carry bit			
One’s complement			

UDP Checksum Example

	Decimal	Binary	Hex
Source IP	192.168.0.31	1100 0000 1010 1000 0000 0000 0001 1111	C0 A8 00 1F
Destination IP	192.168.0.30	1100 0000 1010 1000 0000 0000 0001 1110	C0 A8 00 1E
UDP protocol	17	0000 0000 0001 0001	00 11
Length	10 = 8 + 2	0000 0000 0000 1010	00 0A
UDP Source Port	20	0000 0000 0001 0100	00 14
UDP Destination Port	10	0000 0000 0000 1010	00 0A
UDP Length	10	0000 0000 0000 1010	00 0A
UDP Data	"Hi"	0100 1000 0110 1001	48 69
Add		1 1100 1010 0011 1001	1 CA 39
Add carry bit		1100 1010 0011 1001 + 1	CA39 + 0001 = CA3A
One's complement		0011 0101 1100 0101	35C5

UDP in Linux

```
UDP(7)                    Linux Programmer's Manual                    UDP(7)

NAME       top

       udp - User Datagram Protocol for IPv4

SYNOPSIS       top

       #include <sys/socket.h>
       #include <netinet/in.h>
       #include <netinet/udp.h>

       udp_socket = socket(AF_INET, SOCK_DGRAM, 0);

DESCRIPTION       top

       This is an implementation of the User Datagram Protocol described
       in RFC 768.  It implements a connectionless, unreliable datagram
       packet service.  Packets may be reordered or duplicated before
       they arrive.  UDP generates and checks checksums to catch
       transmission errors.

       When a UDP socket is created, its local and remote addresses are
       unspecified.  Datagrams can be sent immediately using sendto(2)
       or sendmsg(2) with a valid destination address as an argument.
       When connect(2) is called on the socket, the default destination
       address is set and datagrams can now be sent using send(2) or
       write(2) without specifying a destination address.  It is still
       possible to send to other destinations by passing an address to
       sendto(2) or sendmsg(2).  In order to receive packets, the socket
       can be bound to a local address first by using bind(2).
       Otherwise, the socket layer will automatically assign a free
       local port out of the range defined by
       /proc/sys/net/ipv4/ip_local_port_range and bind the socket to
       INADDR_ANY.
```

```
SEND(2)                   Linux Programmer's Manual                   SEND(2)

NAME       top

       send, sendto, sendmsg - send a message on a socket

SYNOPSIS       top

       #include <sys/socket.h>

       ssize_t send(int sockfd, const void *buf, size_t len, int flags);
       ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
                      const struct sockaddr *dest_addr, socklen_t addrlen);
       ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags);
```

```
RECV(2)                   Linux Programmer's Manual                   RECV(2)

NAME       top

       recv, recvfrom, recvmsg - receive a message from a socket

SYNOPSIS       top

       #include <sys/socket.h>

       ssize_t recv(int sockfd, void *buf, size_t len, int flags);
       ssize_t recvfrom(int sockfd, void *restrict buf, size_t len, int flags,
                        struct sockaddr *restrict src_addr,
                        socklen_t *restrict addrlen);
       ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
```

UDP in Practice

- Minimal specifications make UDP very flexible
  - An end-to-end protocol built atop the UDP

- Examples:
  - Most commonly used in multimedia applications
  - RPCs
  - Many others

UDP in Practice

- Minimal specifications make UDP very flexible
  - An end
- Example
  - Most co
  - RPCs
  - Many o

The QUIC Transport Protocol: Design and Internet-Scale Deployment

Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, Zhongyi Shi *
Google
quic-sigcomm@google.com

ABSTRACT

We present our experience with QUIC, an encrypted, multiplexed, and low-latency transport protocol designed from the ground up to improve transport performance for HTTPS traffic and to enable rapid deployment and continued evolution of transport mechanisms. QUIC has been globally deployed at Google on thousands of servers and is used to serve traffic to a range of clients including a widely-used web browser (Chrome) and a popular mobile video streaming app (YouTube). We estimate that 7% of Internet traffic is now QUIC. We describe our motivations for developing a new transport, the principles that guided our design, the Internet-scale process that we used to perform iterative experiments on QUIC, performance improvements seen by our various services, and our experience deploying QUIC globally. We also share lessons about transport design and the Internet ecosystem that we learned from our deployment.

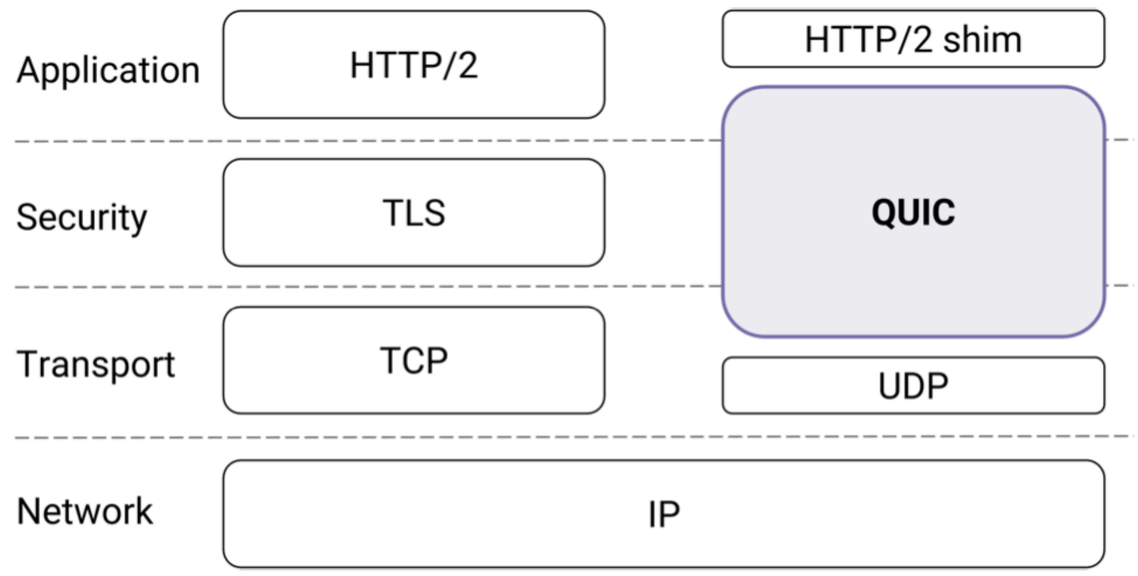


Figure 1: QUIC in the traditional HTTPS stack.

TCP (Figure 1). We developed QUIC as a user-space transport with UDP as a substrate. Building QUIC in user-space facilitated its deployment as part of various applications and enabled iterative

[1] The QUIC Transport Protocol: Design and Internet-Scale Deployment, Sigcomm'17

How does UDP realize these functionalities?

Q1: How can we set up the process-to-process channel?
Q2: How can we multiplex concurrent channels over the physical link?
Q3: How can we control the transmission rate?
Q4: How can we achieve reliable delivery?
Q5: How can we share the in-network bandwidth resources?

UDP Issues

- #1: Arbitrary communication
  - Senders and receivers can talk to each other in any ways

UDP Issues

- #1: Arbitrary communication
  - Senders and receivers can talk to each other in any ways

- #2: No reliability guarantee
  - Packets can be lost/duplicated/reordered during transmission
  - A checksum is not enough

UDP Issues

- #1: Arbitrary communication
  - Senders and receivers can talk to each other in any ways
- #2: No reliability guarantee
  - Packets can be lost/duplicated/reordered during transmission
  - A checksum is not enough
- #3: No resource management
  - Each channel works as an exclusive network resource owner
  - No adaptive support for the physical networks and applications

Transmission Control Protocol (TCP) — RFC793

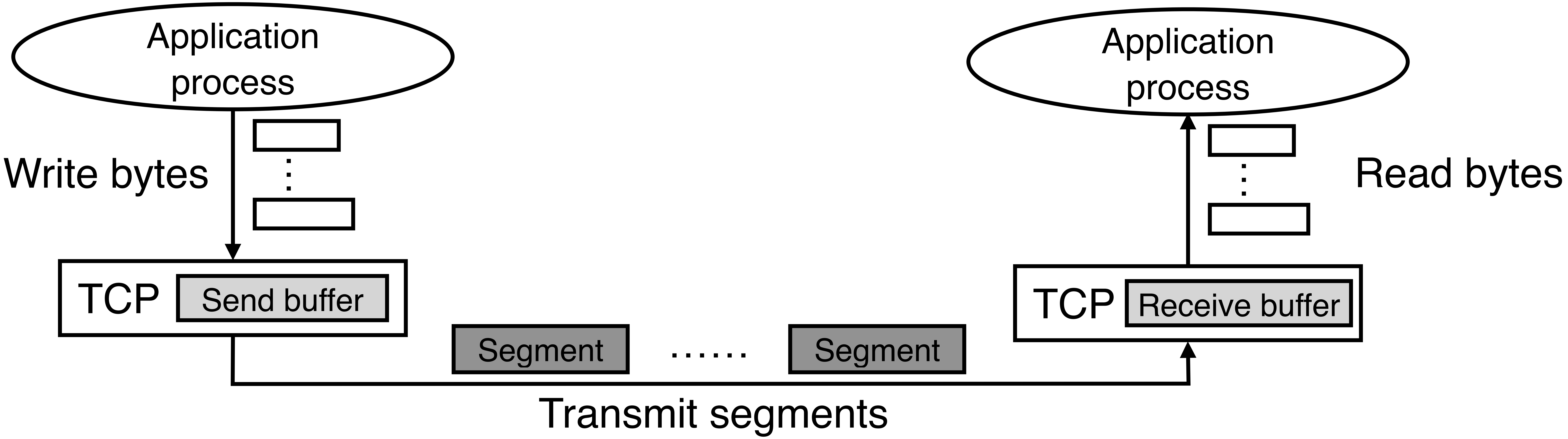
- TCP is the most widely used Internet protocol

- TCP is a two-way, reliable, byte stream oriented protocol

- TCP is closely tied to the Internet Protocol (IP)

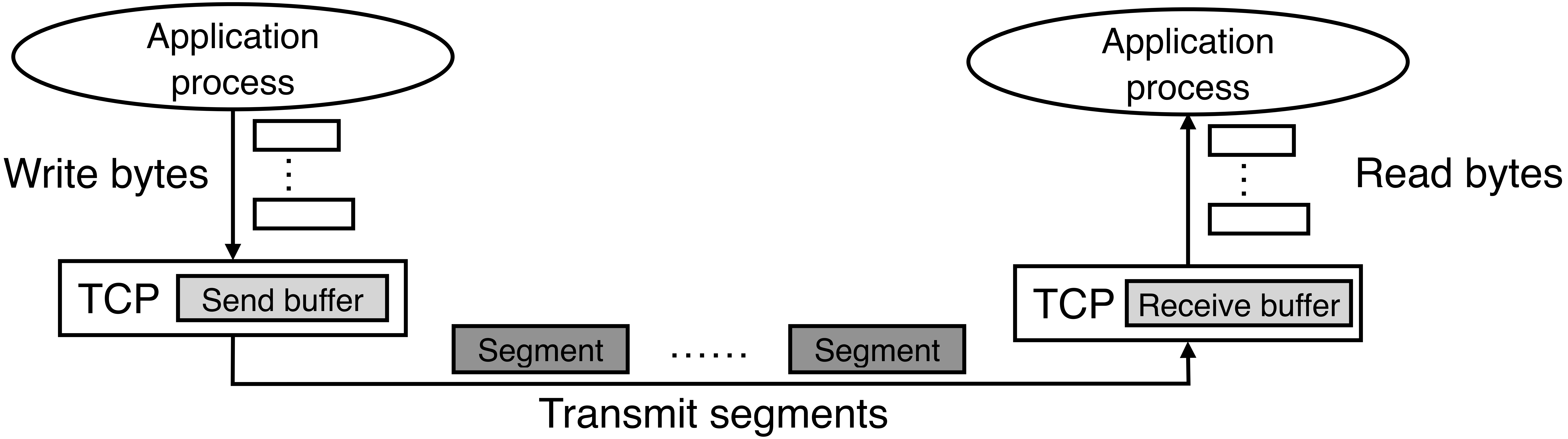
TCP Features

- #1: Connection-oriented
  - Communication happens after the connection is established



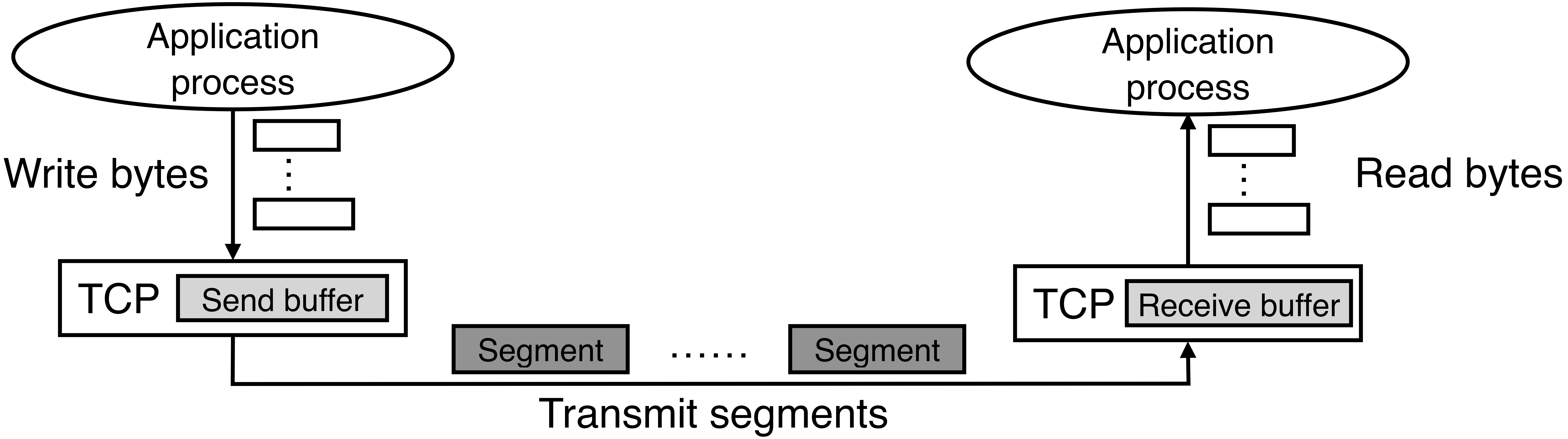
TCP Features (cont'd)

- #2: Byte-stream
  - Applications write/read bytes
  - TCP sends segments



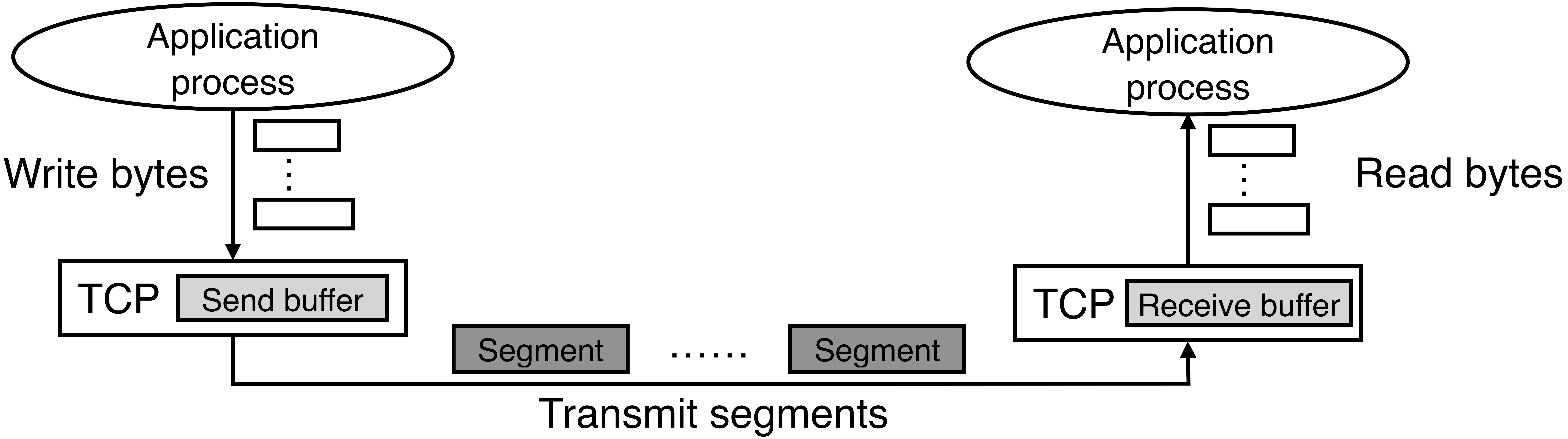
TCP Features (cont'd)

- #3: Two-way communication



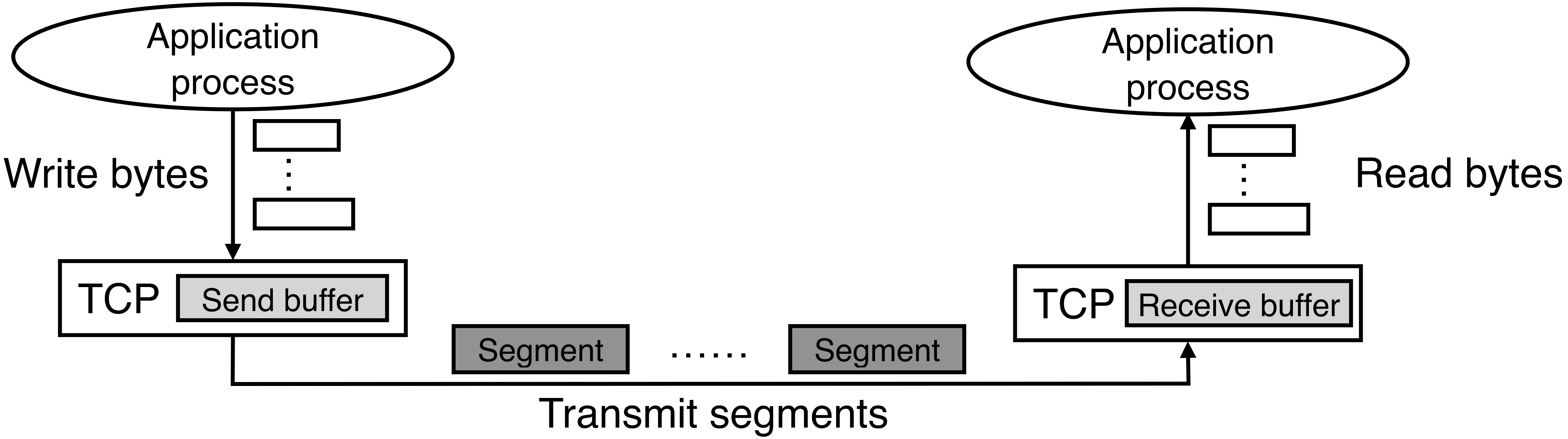
TCP Features (cont'd)

- #4: Keep senders from over-running the receiver
  - Flow control

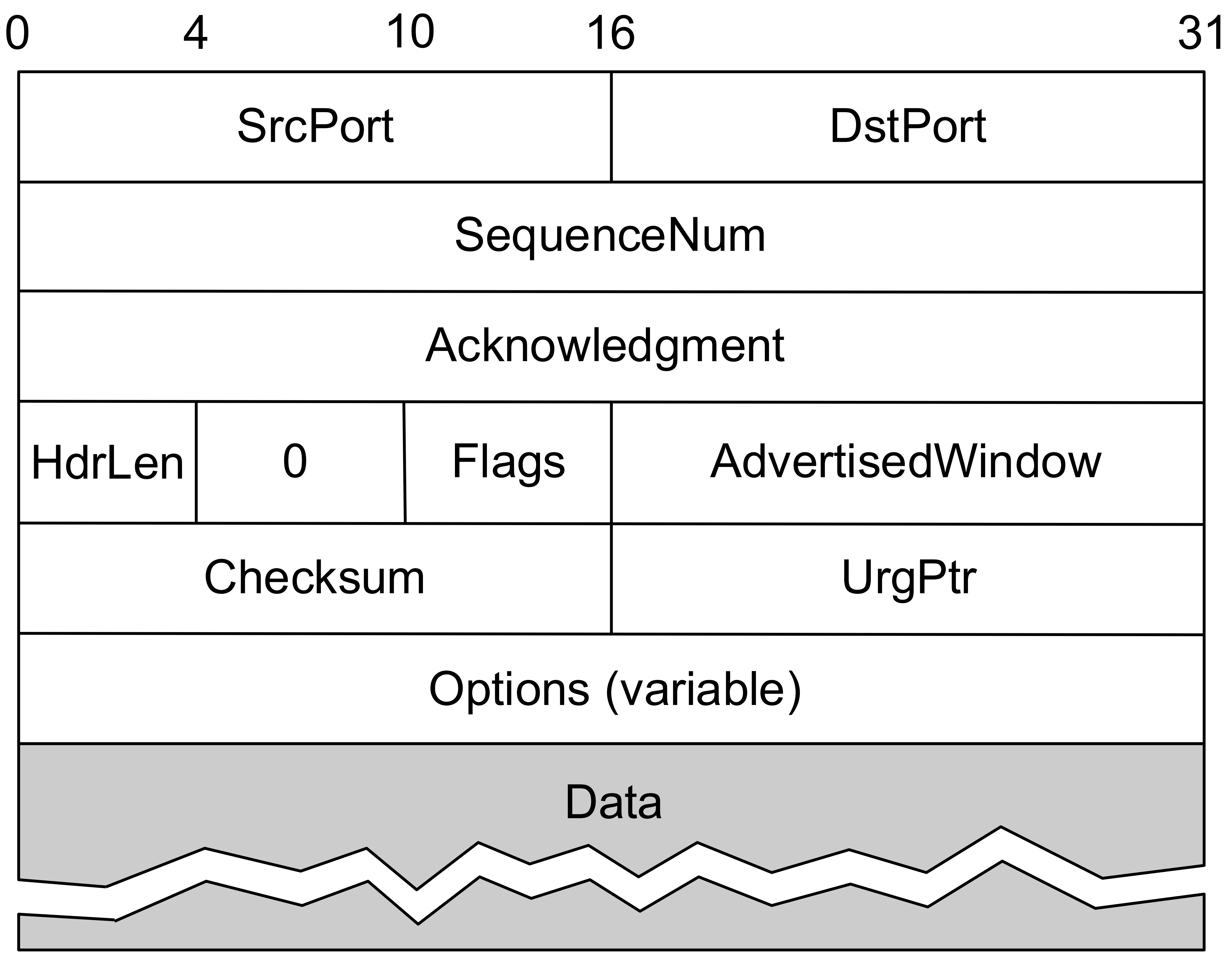


TCP Features (cont'd)

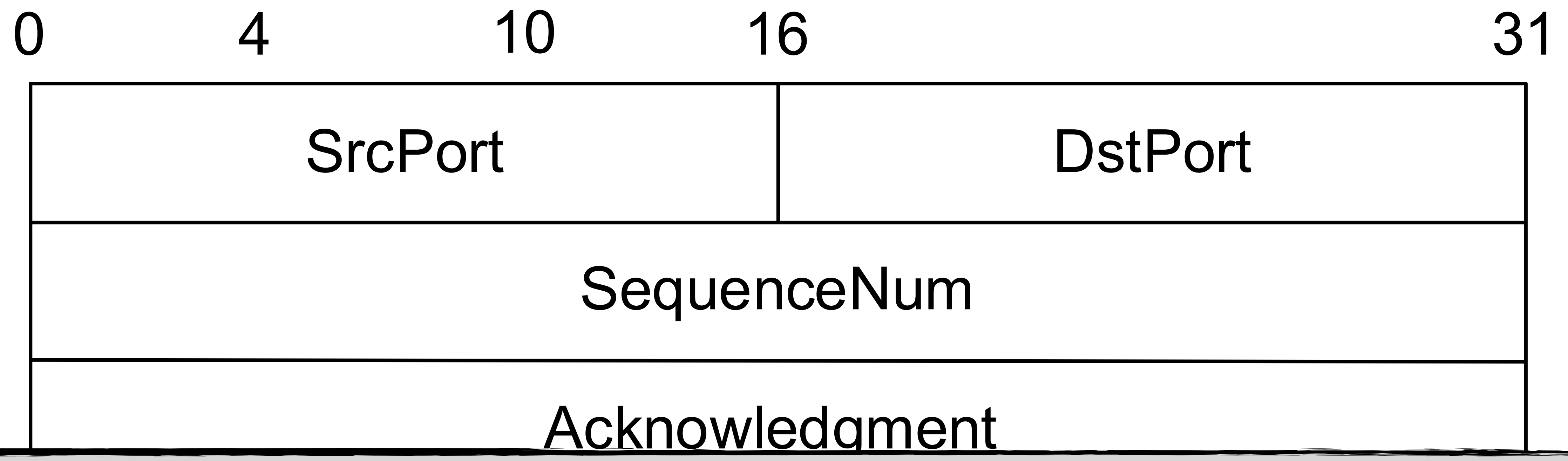
- #5: Keep senders from over-running the network
  - Congestion control



TCP Header Format



TCP Header Format



Our roadmap

- L17/L18: TCP connection management
- L19/L20: TCP reliability support
- L21/L22: TCP congestion control
- L23: TCP in-network support
- L24: Linux Networking Stack

Summary

- Today
  - Transport Introduction

- Next lecture
  - TCP connection management (I)