CPSC 317 – Winter 1 2025 Introduction to Computer Networking

Application Layer Protocols

Module 3.1

Norm & Ibtissem

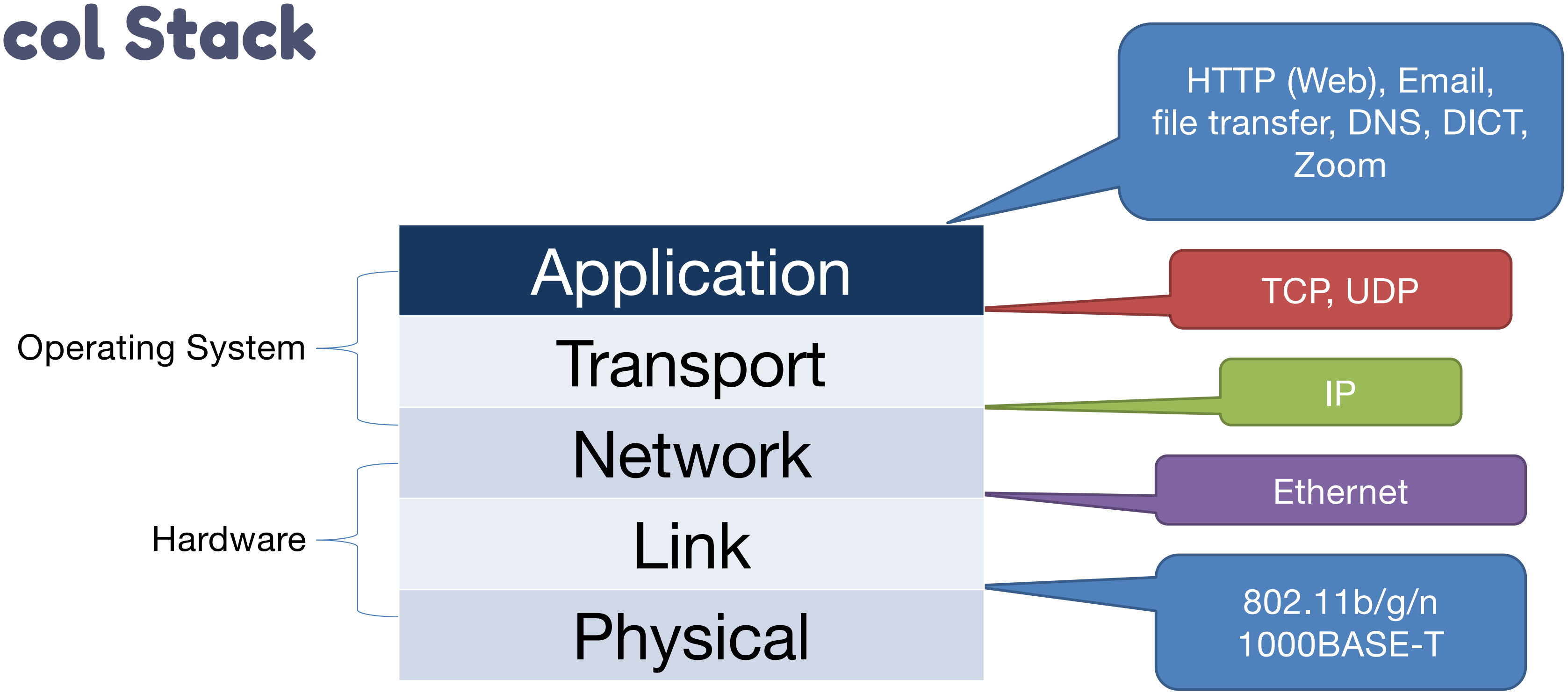
READING

- Reading: 2.1, 2.7

Administration

- Programming Assignment 1 is ongoing
- Quiz 1 starts September 22nd
- There is an iClicker question today!

Protocol Stack



Application

Transport

Network

Link

Physical

Operating System

Hardware

HTTP (Web), Email, file transfer, DNS, DICT, Zoom

TCP, UDP

IP

Ethernet

802.11b/g/n
1000BASE-T

Learning Goals - General

- Explain design considerations for application protocols
- Explain the advantages and disadvantages of open (defined by a standard) vs. closed (proprietary) protocols
- Explain the difference between a peer-to-peer and a client-server application protocol
- Explain the quality of service requirements for different applications
- Explain what a socket and a transport layer address is
- Effectively use Java APIs or C system calls to create/destroy sockets and send/receive data

Design Considerations For Application-layer Protocols

- Each application using the network will define its own protocol
- Open vs Proprietary
- Architecture: client-server, peer-to-peer (P2P)
  - Who is the client, who is the server
  - How does the client identify which server to contact
  - Rules for when client and/or server send/receive messages
- Choice of transport protocol
  - Desired quality of service
- Types and formats of messages (request, response, etc.)
  - Message syntax and semantics
  - Message encoding format (text, binary, etc.)

Open Vs Proprietary Protocols

- Open protocols: publicly known
    - Examples: DICT, HTTP, SMTP, SSH
    - Usually defined in RFC (Request for Comments) documents
    - Many different implementations

- Proprietary protocols
    - Examples: Skype, iCloud, Zoom
    - Only one implementation

Client-Server Architecture

- Well-defined roles for client and server
- Server is always on, with permanent address or host name
- Client establishes connection
- Connection is always between one client and one server (although the server will serve multiple clients at once)

Peer-to-Peer Architecture

- Connections typically between peers with the same hierarchical role
  - Some hierarchy may be used, but connection is not restricted to it
- Peers request service from other peers, provide service in return
- Self scalability: new peers bring new demand and new capacity
- Complex peer address management

What Quality Of Service Does An Application Need?

Data loss

- some apps can tolerate some loss (e.g., audio)
- other apps require 100% reliable data transfer (e.g., file transfer, web, email)

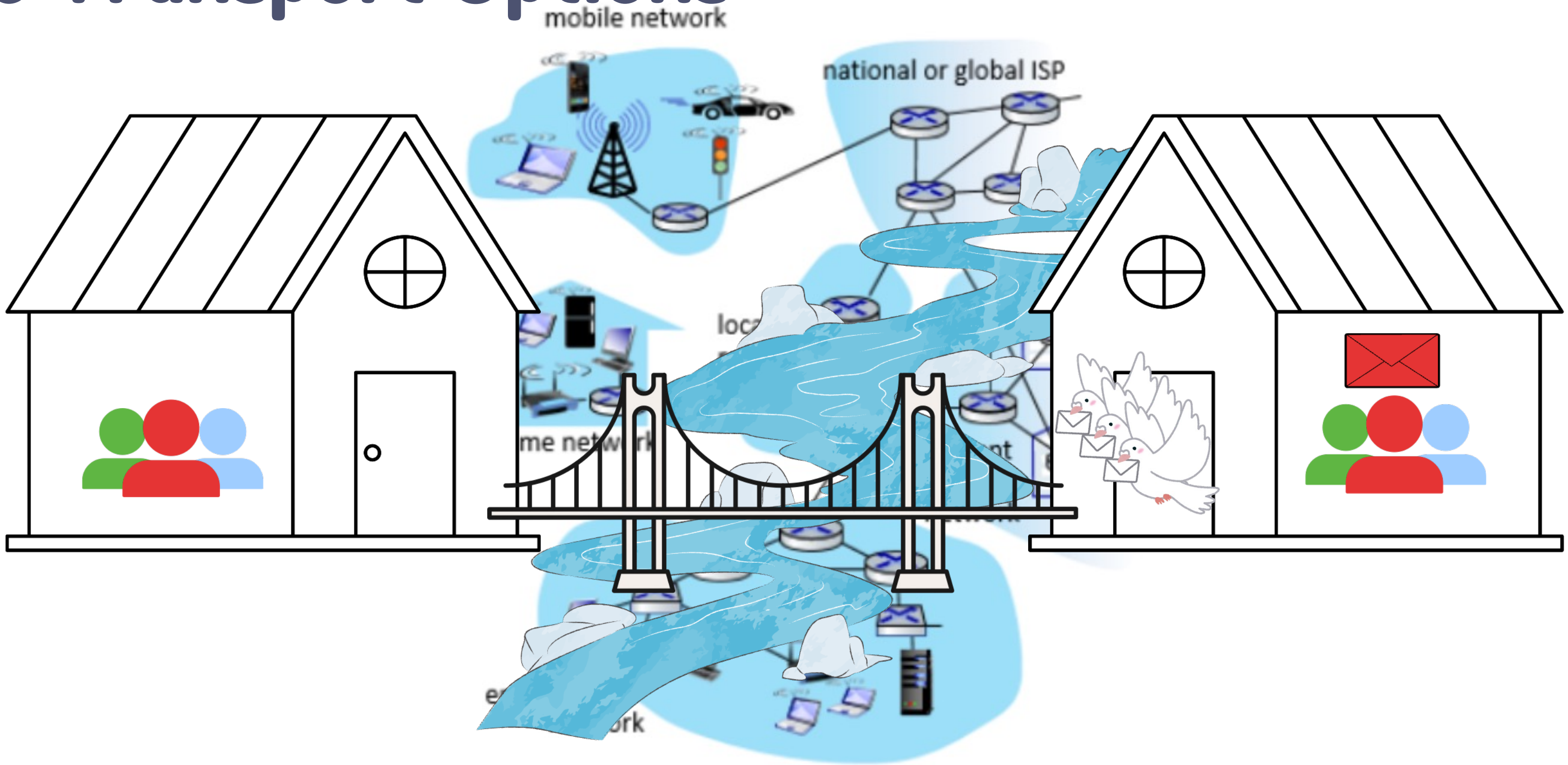
Time sensitivity

- some apps require low delay to be “effective” (e.g., interactive ones)

Bandwidth

- some apps (e.g., multimedia) require minimum amount of bandwidth to be “effective”
- other apps (“elastic apps”) make use of whatever bandwidth they get

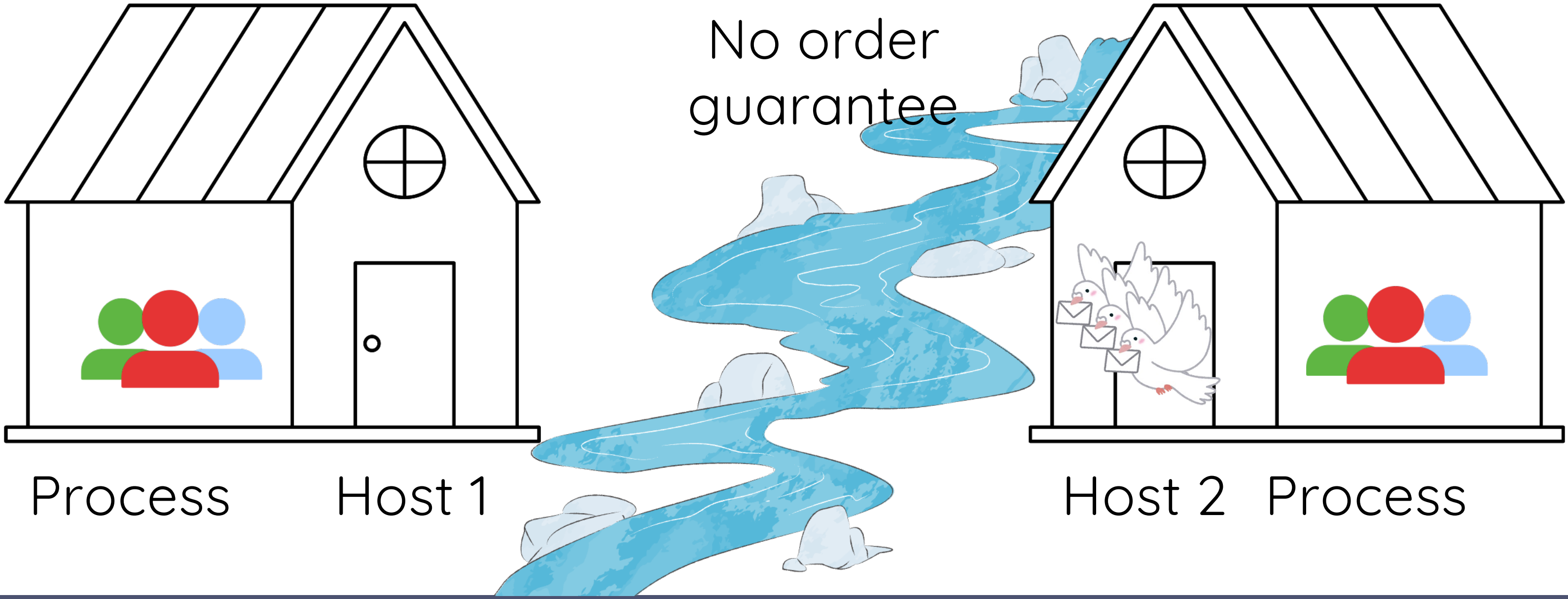
Two Transport Options



Two Transport Options

✓ Simple
Fast

✗ Unreliable
No order guarantee



Two Transport Options

✓ Reliable

In-order delivery

Congestion control

✗ Slower

Process
Host 1
Host 2
Process

Two Transport Options

Reliable stream	Unreliable packet
Connection	No connection
Reliable ordered delivery	Best effort
Flow/Congestion control	Nope
Possible delays	No (transport level) delay

Application Examples

- File transfer, web, email → TCP
  - Loss averse, not time sensitive, elastic bandwidth
- Text messaging → TCP or UDP
  - Loss averse, elastic bandwidth, somewhat time sensitive
- On demand multimedia streaming → UDP
  - Some loss tolerance, somewhat time sensitive
- Real time multimedia, VoIP, interactive games → UDP
  - Some loss tolerance, time sensitive, bandwidth requirements
- Domain Name Service → UDP
  - Loss tolerant, not time sensitive, elastic bandwidth

The DICT Protocol

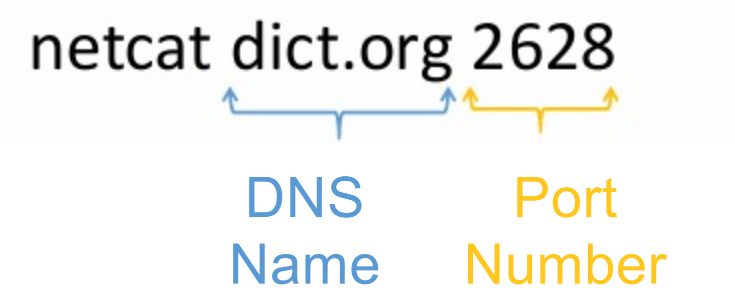
- Defined in RFC 2229
  - Google: dict protocol rfc
- Simple text-based, request-response protocol
- Commands: help, define, match, show db, show strat, quit
- Example: netcat dict.org 2628
  - Wait, what is this 2628??

A Digression On Network Transport

- We will talk about the transport layer protocols later (module 4), but …
- You need to know how applications see the layer below them in the network protocol stack
- Two things you need to be aware of:
  - A transport layer address – how network applications are identified
  - A socket – a network end point

Transport Layer Addresses

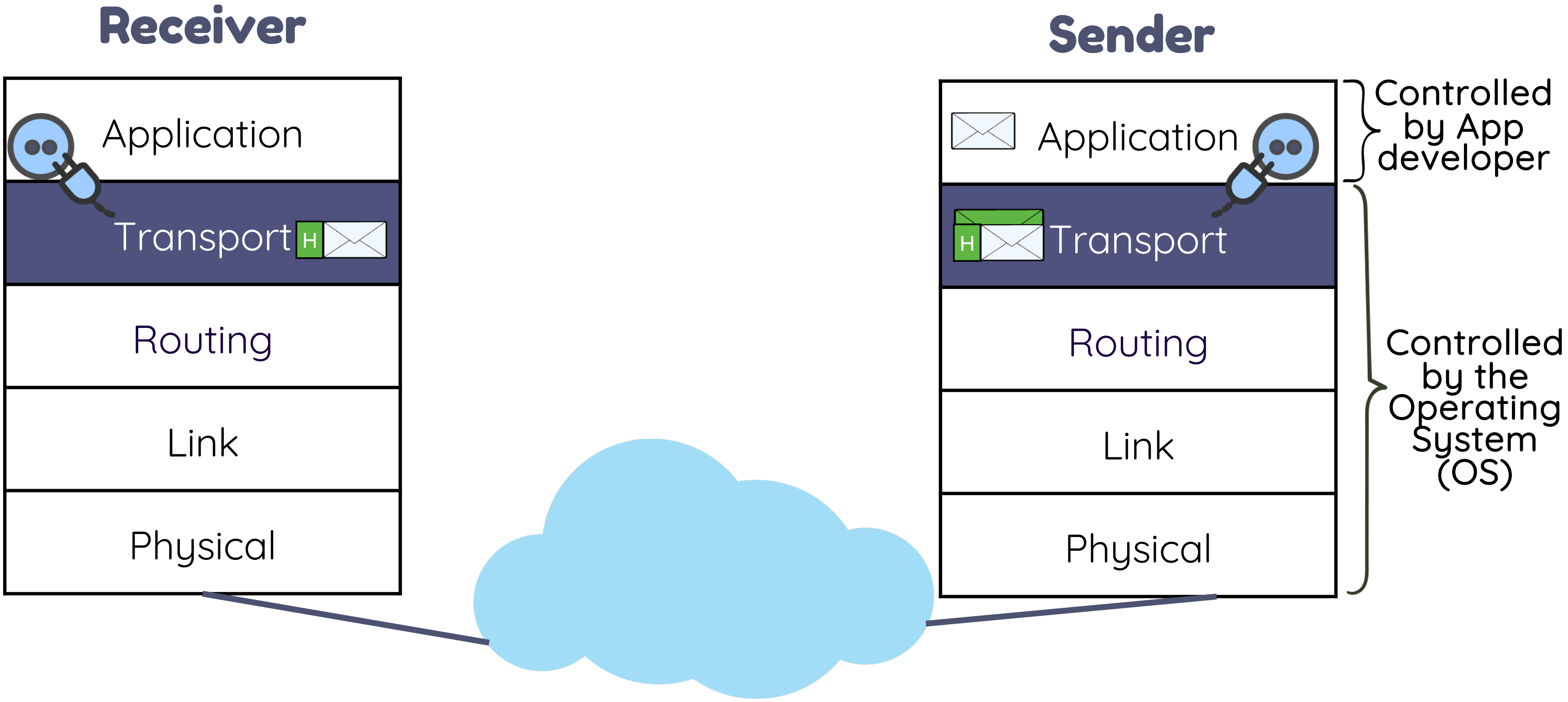
- A pair of a 32-bit IP host address and a 16-bit port number
  - or a 128-bit IPv6 host address and a 16-bit port number
- Usually the IP address is derived from a DNS name
  - www.cs.ubc.ca, google.ca, amazon.ca, …



```
netcat dict.org 2628
```

DNS Name

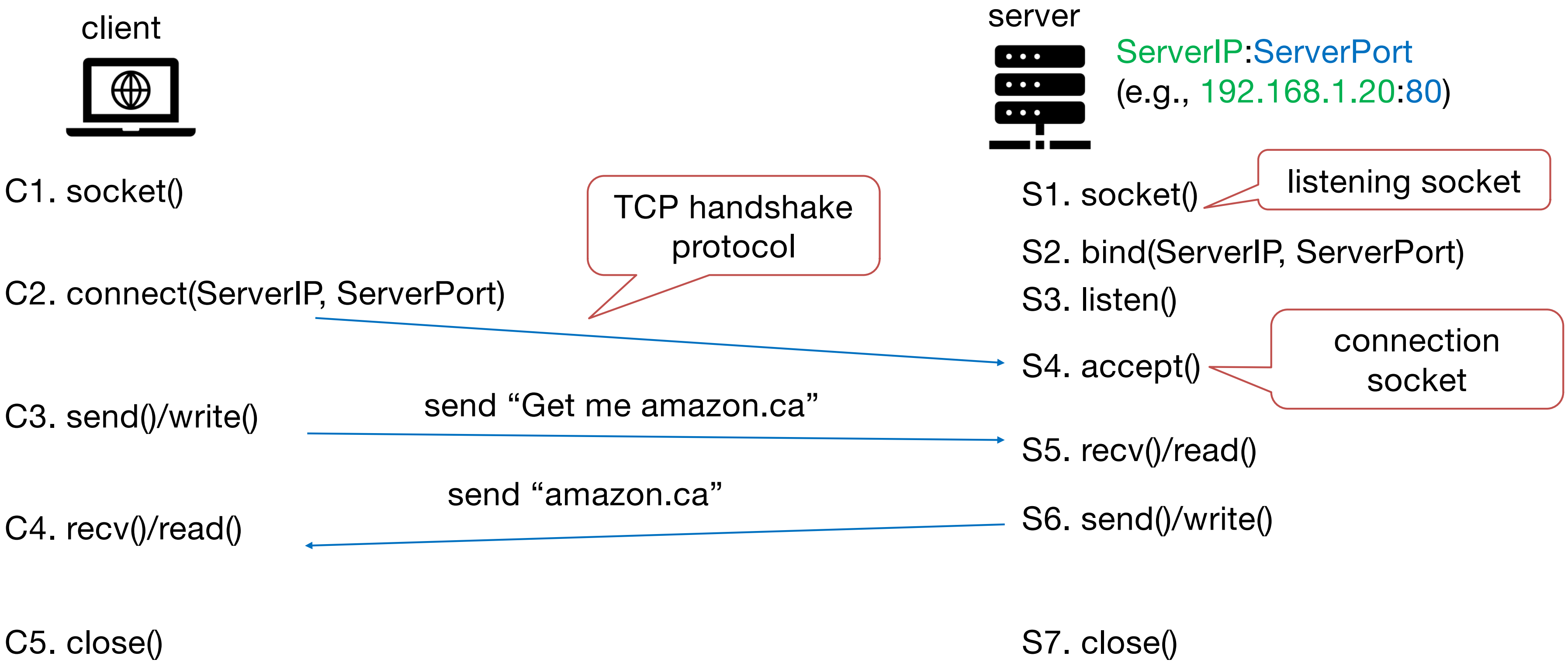
Port Number


Receiver
Application
Transport
H
Routing
Link
Physical
Sender
Application
H
Transport
Routing
Link
Physical
Controlled by App developer
Controlled by the Operating System (OS)

Sockets – Network Endpoints

- Created via socket() system call
  - Parameters to the socket() call identify the transport protocol and optionally the other participating process
- Destroyed via close()
  - like all other file descriptors
- Data sent and received using send() and recv()
  - or read() and write() if the socket is “connected”

Sockets In C (Tcp Client-Server)



client

server

ServerIP:ServerPort
(e.g., 192.168.1.20:80)

C1. socket()

C2. connect(ServerIP, ServerPort)

C3. send()/write()

C4. recv()/read()

C5. close()

S1. socket()

S2. bind(ServerIP, ServerPort)

S3. listen()

S4. accept()

S5. recv()/read()

S6. send()/write()

S7. close()

TCP handshake protocol

listening socket

connection socket

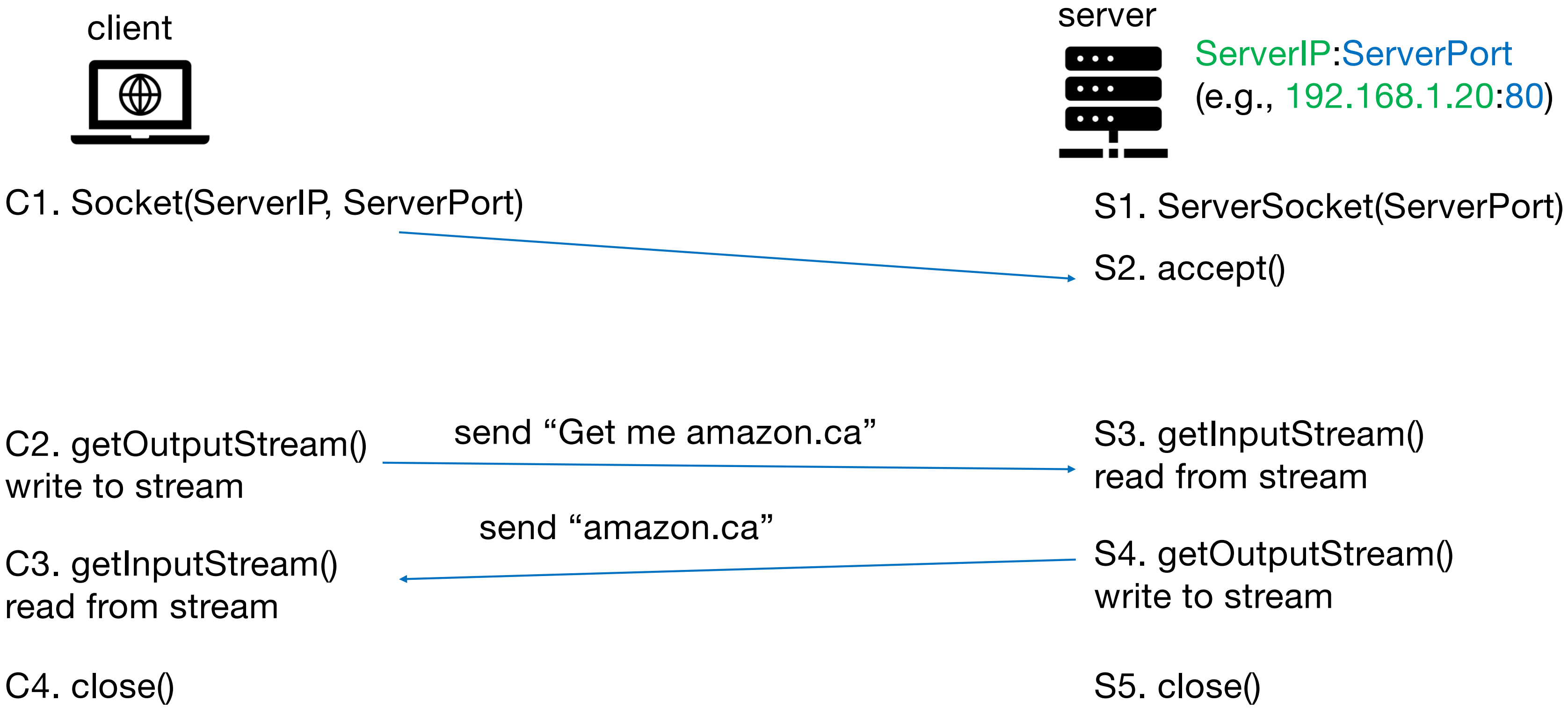
send “Get me amazon.ca”

send “amazon.ca”

Sockets in Java

- class Socket for the client side
- new Socket(host, port)
  - host is a String
  - port is an int
  - Creates a socket, connects it to the indicated server, and returns the socket
  - Throws various exceptions when things go wrong
- class ServerSocket() for the server side

Sockets In Java (Tcp Client-Server)



client

server

ServerIP:ServerPort
(e.g., 192.168.1.20:80)

C1. Socket(ServerIP, ServerPort)

S1. ServerSocket(ServerPort)

S2. accept()

C2. getOutputStream()
write to stream

send "Get me amazon.ca"

S3. getInputStream()
read from stream

send "amazon.ca"

C3. getInputStream()
read from stream

S4. getOutputStream()
write to stream

C4. close()

S5. close()

The Simplest Java Socket Client

```
private void body() {
    try (Socket s = new Socket(”dict.org”, 2628)) {
        BufferedReader input = new BufferedReader(new InputStreamReader(s.getInputStream()));
        String greeting = input.readLine();
        System.out.println(greeting);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

In-class Activity

- Form yourselves into groups (1 – 9 students per group)
- You should be able to chat conveniently in your group
- Go to PrairieLearn
- Click on Assessments
- Start the ICA31 assessment (Application Architecture and Transport Protocols)
- Talk in your group about the answers
  - Hearing other students ideas
  - Explaining your ideas to others

Next Topic:
Application Layer – The Web