CSC361

Computer Networking

Mantis Cheng

Dept of Computer Science

Unit 2

Models of TCP/IP

Important Concepts

- IP Service Model
- UDP/TCP Service Model
- Multiplexing & Demultiplexing
- Client/Server Architecture
- Sockets API
- Python Sockets

What We Learned So Far

- IP is an **unreliable datagram delivery** service.
- TCP provides a **reliable bidirectional byte stream** service over IP.
- Most Internet applications are built on top of UDP/TCP using Berkeley **socket** API.
- **New** Internet application **functionalities** should only be added at the endpoints.
- **Abstraction** and **encapsulation** are key to the success of the 4-layer model of TCP/IP.

The IP Service Model (17:46)

Summary

- The Network (or IP) layer is the **most important** layer in the Internet.
- Each IP packet consists a **header** and a **payload**.
- Each IP **packet** is transmitted over the Link layer as a payload in a **frame**.
- The header of each IP packet contains both **source** and **destination** IP addresses.
- The Network layer only uses **IP addresses** to route each packet.

Summary (continued)

- Each router only provides a Network layer service, but **not** any (Transport/Application) layer above it.
- IP is **connectionless**; each packet is routed **independently**.
- IP is **unreliable**; it delivers datagrams with **best-effort** but **no** guarantee.
- To prevent packet delivery looping, each packet carries a `Time-To-Live` counter.

Summary (continued)

- IP may **fragment** long data segments into multiple packets.
- IP **checksums** its header to prevent delivery to the wrong address.
- IPv4 uses **32-bit** addresses. (We won't cover IPv6 in this course.)
- IP provides multiple **protocol ID** for demultiplexing to upper layer services (e.g., TCP, UDP, etc.)

UDP Service Model (6:41)

Summary

- User Datagram Protocol (UDP) is an **unreliable** datagram service built on top of IP (Network) layer.
- UDP is **connectionless**.
- UDP uses port numbers to **multiplex/demultiplex** datagrams to different applications.
- The UDP provides a `source` port number for the sender and a `destination` port number for the receiver.

Summary (continued)

- An IP address plus port number pair is known as a **socket**, a communication endpoint.
- When an application sends a UDP datagram, the UDP layer adds a **port number** for this application, and IP layer adds its **local host IP address** as source.
- The application **must** specify the receiver's socket (its port number and IP address).
- The port numbers identify the applications at both ends.

TCP Service Model (16:27)

Summary

- Transmission Control Protocol (TCP) is a **reliable, bidirectional, byte-stream** service built on top of IP (Network) layer.
- TCP is **connection-based**.
- TCP uses a **3-way handshake** to establish a connection.
- Both ends **must** agree upon an **initial sequence number** for each direction when a connection is made.

Summary (continued)

- A connection is two **continuous** streams of bytes in both directions, indexed by **sequence numbers**.
- A connection is closed with a **4-way tear-down**.
- Each TCP segment is checksumed.
- TCP uses **Flow** control to prevent receiver **buffer overflow**.
- All bytes are transmitted in **order reliably**.
- TCP uses **Congestion** control to prevent **Internet collapse**.

Summary (continued)

- Some port numbers are **well-known** (e.g., `TCP/80` is `HTTP`, `UDP/53` is `DNS`).
- The first 1024 ports are reserved.
- The remaining (64K-1K) ports are typically assigned **dynamically**.
- Each TCP connection is identified by two pairs of sockets ((`source IP address`, `source port #`), (`dest IP address`, `dest port #`)).

Wireshark Demo

(UDP, TCP, DNS)

(curl info.cern.ch)

Client-Server Model (1:00:00)

(watch the first 35 minutes of this video at home)

Summary

- A server is a process **waiting** for client's requests.
- A client is another process, on the same or different machine, making requests to the server.
- A server may serve **multiple** clients at the same time (*concurrently*); or it may serve **one** client's request at a time (*iteratively*).
- Iterative servers are suitable for **short service time** applications.
- Concurrent servers are suitable for **indeterminate service time** applications.

Summary (continued)

- Client/Server may use UDP or TCP for network communication.
- Typical internet applications use Berkeley Socket API.
- A `socket` = `TCP/UDP` + `IP address` + `Port number`.
- Two sockets identify two communication endpoints.
- UDP is an **unreliable datagram** service; while TCP is a **reliable bidirection byte-stream** service.

Summary (continued)

- DNS translates `hostnames` to `IP addresses`.
- A **domain** defines a collection of machines that share some common characteristics.
- A **Top-Level-Domain** (TLD) defines a top-level partition of all domain/host names in the Internet.
- DNS is a distributed database maintaining the mapping of hostnames to IP addresses.
- Watch ICANN and the 7 Keys to the Internet (5:03) to appreciate the security of DNS.

Socket Programming (34:00)

(watch the first 16 minutes)

Summary

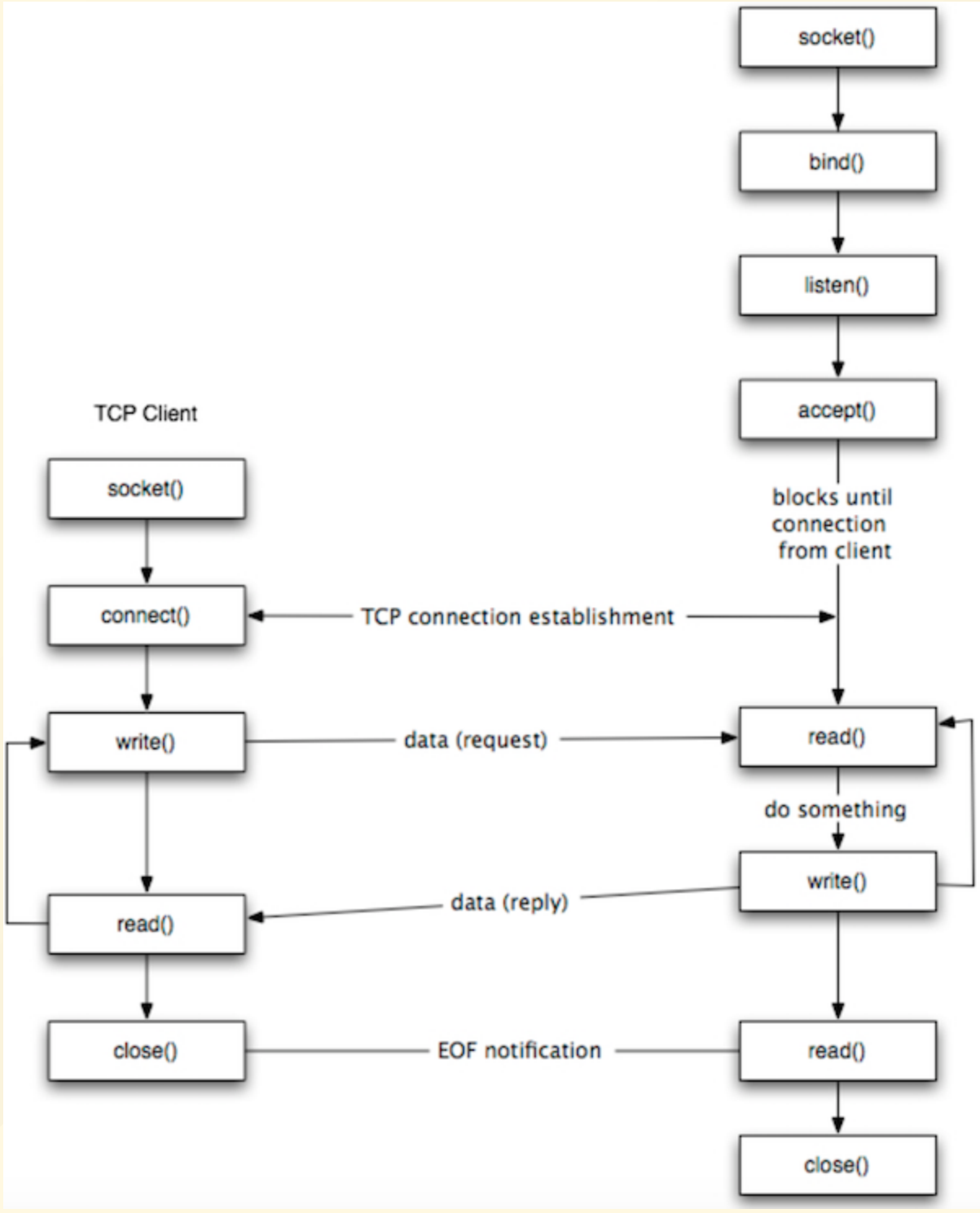
- A `socket` = `TCP/UDP` + `IP address` + `Port number`.
- A port number corresponds to a **unique application id**.
- There are many **well-known** port numbers. Each port number is **16-bit** (64K ports) number.
- The Berkeley socket API includes: `socket`, `bind`, `connect`, `listen`, `accept`, `read`, `write`, and `close`.

Summary (continued)

- **Server** side:
  1. create a `socket` for accepting requests
  2. `bind` the created socket to a port number
  3. `listen` for any requests coming into the socket
  4. `accept` one request from the socket
  5. `receive/read` and process the request
  6. `send/write` a response back to the client
  7. `close` connection

Summary (continued)

- **Client** side:
  1. create a `socket` for sending a request using UDP/TCP
  2. `connect` its socket to the server's socket
  3. `send/write` a request
  4. `receive/read` a response


socket()
bind()
listen()
accept()
TCP Client
socket()
blocks until
connection
from client
connect()
TCP connection establishment
write()
data (request)
read()
do something
write()
data (reply)
read()
close()
EOF notification
read()
close()

Summary (continued)

- `socket`: create a socket structure
- `connect`: establish a connection if needed
- `bind`: associate a socket to a port #
- `listen`: wait for requests
- `accept`: take one outstanding request
- `send/write`: transmit a request/response
- `receive/read`: receive a request/response
- `close`: close the connection if exist

Python Network Programming Summary (PDF)

(study this summary and follow the examples carefully)

Python Network Programming (21:38)

(watch this video at home)

Summary

- It is very easy to use Python to create network applications using Berkeley sockets.
- The Python socket API is essentially **identical** to the standard C version.
- Our programming labs are all written in Python.
- Please follow the above examples to learn how to use UDP or TCP to create network applications.
- You will then understand the difference between UDP and TCP better afterwards.

The End