

CMSC 132: Object-Oriented Programming II



Networking

Department of Computer Science
University of Maryland, College Park

Networking

■ Internet

- Designed with multiple layers of abstraction
- Underlying medium is unreliable, packet oriented
- Packet-Switching

- Animation:

http://www.pbs.org/opb/nerds2.0.1/geek_glossary/packet_switching_flash.html

- Provides two views
 - Reliable, connection oriented (TCP)
 - Unreliable, packet oriented (UDP)

■ Java

- Object-oriented classes & API
 - Sockets, URLs
 - Extensive networking support

Internet (IP) Address

- **Unique address for machine on internet**
 - Get from ISP when connecting to internet
 - Allows network to find your machine
- **Format**
 - 32-bit unsigned integer $\Rightarrow$ 128.8.128.8
 - Domain name $\Rightarrow$ cs.umd.edu
- **Name and address for local machine**
 - localhost
 - 127.0.0.1

Internet (IP) Address

■ Problem

- Running out of 32-bit IP addresses
- Caused by initial address allocation
 - Stanford & MIT initially given more IP addresses than China
 - fixed in 2000
 - Univ. of Maryland is currently assigned 131,072 IP addresses
- Switching to 128-bit IP addresses in IPv6
 - 1+ million addresses per square meter on Earth

IP Address – DNS

■ Domain Name System (DNS)

- Protocol for translating domain names to IP addresses

 - Example: `cs.umd.edu` → `128.8.128.44`

- Multiple DNS servers on internet

- DNS server may need to query other DNS servers

 - `edu` DNS server queries `umd.edu` server to find `cs.umd.edu`

Ports

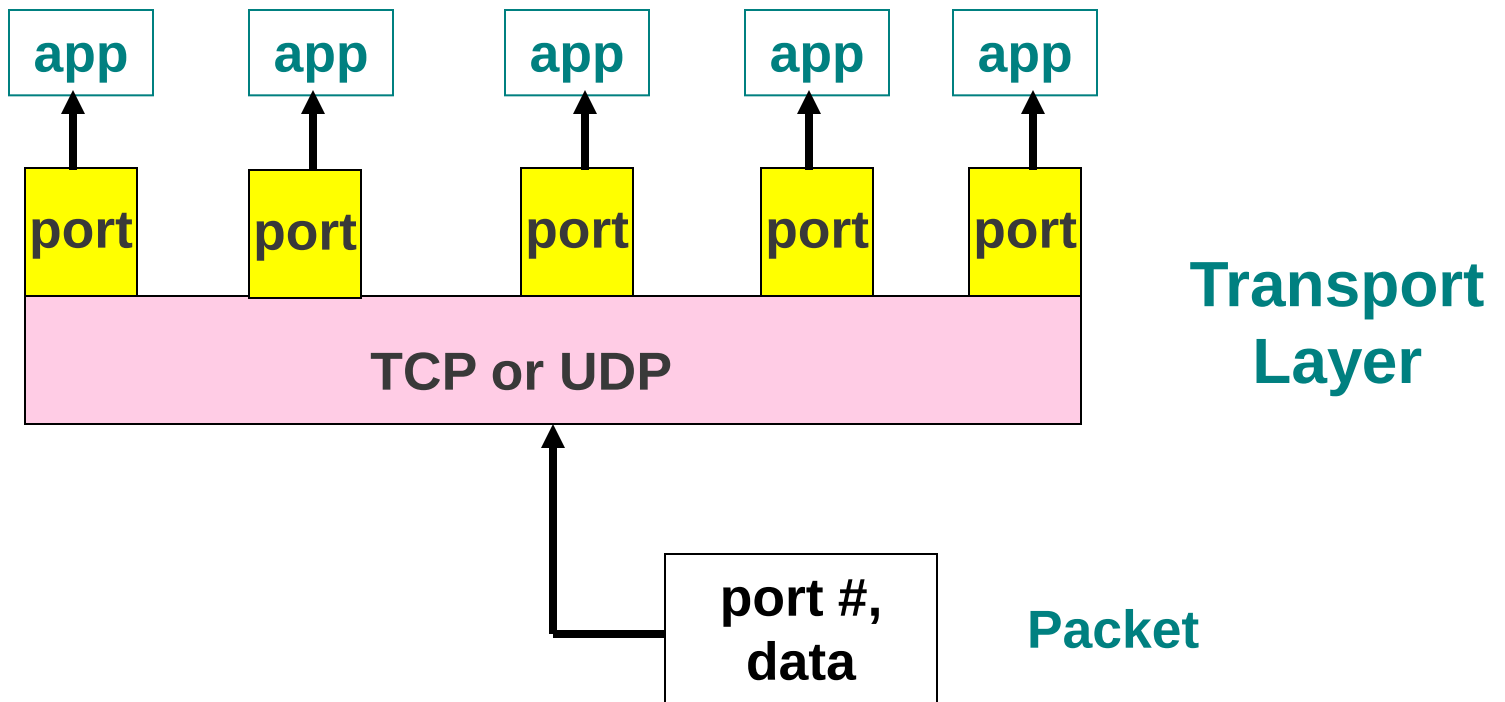
- **Abstraction to identify (refine) destination**
 - Provide multiple destinations at single IP address
- **Format**
 - Unsigned 16-bit integer (0 to 65,535)
 - Ports 0 to 4096 often reserved & restricted
- **Many ports pre-assigned to important services**
 - 21 ftp (file transfer)
 - 23 telnet (remote terminal)
 - 25 SMTP (email)
 - 80 http (web)
 - ...

Sockets

- **Application-level abstraction**
 - Represents network connection
 - Implemented in software
 - Supports both UDP and TCP protocols
- **History**
 - Introduced in Berkley UNIX in 1980s
 - Networking API

Sockets

- Socket is bound to port number
 - Receives data packet
 - Relays to specific port



Uniform Resource Locators (URLs)

■ Represent web resources

- Web pages
- Arbitrary files
- ...

■ Examples

- <http://www.cs.umd.edu/index.html>
- ftp://www.cs.umd.edu/pub/doc/csd_policies.pdf
- <https://login.yahoo.com/>
- <file:///dir/my.txt>

Uniform Resource Locators (URLs)

■ Consists of

■ Protocol

- http:

- https: (secure http)

- file:

- ...

■ IP address (or domain name)

■ Port (optional, 80 if not specified)

- <http://www.cs.umd.edu:80/>

■ Reference to anchor (optional)

■ Query terms

Internet Connections

■ Two types of connections

1. Connection-oriented (TCP)
2. Packet-oriented (UDP)

Transmission Control Protocol (TCP)

- Connection oriented
- Message split into datagrams
- Send datagrams as packets on network layer
- Provides illusion of reliable connection
 - Extra messages between sender / recipient
 - Resend packets if necessary
 - Ensure all packets eventually arrive
 - Store packets and process in order
 - Provides warning if packets are lost

Transmission Control Protocol (TCP)

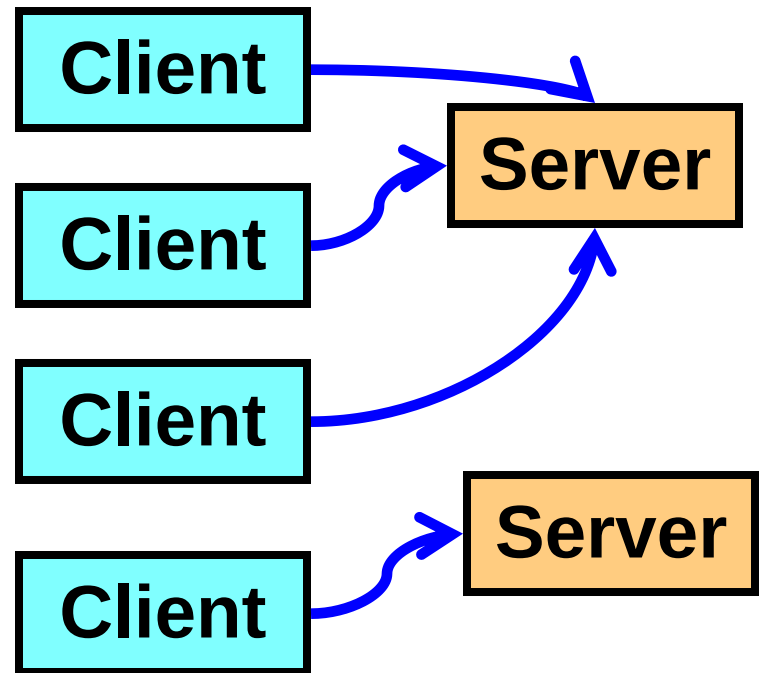
- **Reliable but more overhead for small messages**
- **Application can treat as reliable connection**
 - **Despite unreliability of underlying IP (network)**
- **Examples**
 - **ftp (file transfer)**
 - **ssh (remote secure shell)**
 - **http (web)**
- **Vast majority of internet traffic is TCP**

User Datagram Protocol (UDP)

- More like sending a postcard
- Limited size message
- Might get lost with no notification
- Useful in some specialized cases
 - messages are small
 - if a packet is lost, would rather just lose it than delay receipt of next packet

Client / Server Model

- Relationship between two computer programs
- Client
 - Initiates communication
 - Requests services
- Server
 - Receives communication
 - Provides services
- Other models
 - Master / worker
 - Peer-to-peer (P2P)



Client Programming

■ Basic steps

1. Determine server location – IP address & port
2. Open network connection to server
3. Write data to server (request)
4. Read data from server (response)
5. Close network connection
6. Stop client

Simple Server Programming

■ Basic steps

1. Determine server location - port (& IP address)
2. Create ServerSocket to listen for connections
3. Loop

```
while (true) {
```

```
    Accept network connection from client
```

```
    Read data from client (request)
```

```
    Write data to client (response)
```

```
    Close network connection to client
```

```
}
```

Java Networking Classes

- IP addresses
 - InetAddress
- Packets
 - DatagramPacket
- Sockets
 - Socket - TCP client sockets
 - ServerSocket - TCP server sockets
 - DatagramSocket - UDP sockets (server or client)
 - Sockets transfer data via Java I/O **streams**
- URL Connection Classes
 - High-level description of network service
 - Access resource named by URL
 - Examples
 - URLConnection ⇒ Reads resource
 - HttpURLConnection ⇒ Handles web page
 - JarURLConnection ⇒ Manipulates Java Archive

Java Networking Examples

- **TCP Client/Server: See tcpServerClient package**
- **UDP Client/Server: See udpServerClient package**
- **URL Reader: See urlReader package**
- **Toy Web Server: See toyWebServer package**

Advanced Server Programming

- **Server supports multiple connections / clients**
- **Two approaches**
 1. **Loop**
 - **Handles multiple connections in order**
 - **Limits on amount of network traffic**
 - **Not resilient in face of slow / stopped clients**
 2. **Multithreading**
 - **Allows multiple simultaneous connections**