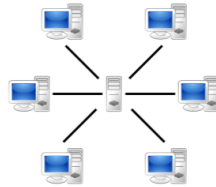


Application Layer (I)

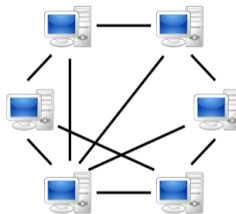
Two types of architecture

- Client and Server



Img ref: <https://en.wikipedia.org/wiki/Peer-to-peer#/media/File:Server-based-network.svg>

- **Client** (caller)
 - actively opens connection to the server
 - resource requesters
 - must know server's IP address and port number
- **Server** (callee)
 - waits for clients to connect
 - resource providers
 - may handle multiple simultaneous client connections
- P2P (peer-to-peer)



Img ref: <https://en.wikipedia.org/wiki/Peer-to-peer#/media/File:P2P-network.svg>

- Peer
 - **not owned by the service provider**, but are instead desktops and laptops controlled by users
 - **equal privileged**
 - both client and server, client when requesting resources from other peers, server when providing resource to other peers
- **Minimal (or no) reliance on dedicated servers** in data centers
- Application exploits **direct communication between pairs** of intermittently connected hosts/peers
 - Applications

- file sharing (BitTorrent)
- peer-assisted download acceleration (Xunlei)
- Blockchain
- Some applications have **hybrid architectures**
 - **Servers** are used to **track the IP addresses of users**, but user-to-user messages are sent directly between user hosts.
- **Self-scalability**
 - E.g., in a P2P file-sharing application, although each **peer generates workload** by requesting files, each peer **also adds service capacity** to the system by distributing files to other peers.
- **Cost effective**
 - Normally **don't require significant server infrastructure** and server **bandwidth** (in contrast with clients-server designs with datacenter)
- Challenges:
 - **ISP Friendly**
 - Most residential ISPs (including DSL and cable ISPs) have been dimensioned for “**asymmetrical**” bandwidth usage, that is, for **much more downstream than upstream traffic**. But P2P video streaming and file distribution applications **shift upstream traffic from servers to residential ISPs**, thereby putting significant stress on the ISPs. Future P2P applications need to be designed so that they are friendly to ISPs.
 - **Security**
 - Due to the highly distributed and open nature
 - **Incentives**

Underlying Point-to-point Communication

- Underlying network provides **reliable connection-oriented** or **unreliable connection-less** service (TCP or UDP)
- A quadruple to uniquely identify an application communication between a client and a server
 - **source/dest IPs, source/dest port numbers** (port #, id that is unique within the host)
 - Client typically uses *ephemeral* source (local) port number
 - Server connects to local port from typically a *well-known* one
 - Why does it need a quadruple to uniquely identify a communication?
 - Both client and sever can have **multiple processes running simultaneously**. A client can send multiple web requests to different websites, and a server can receive multiple requests from different clients.
- E.g., open a web browser on your laptop and then try to connect to CS331 and CS333 course websites. Both course websites are located on the Colby CS server. Destination port 80 can let server know that the two requests are web requests, the source destination port can let the server tell which response is to which request.

Command Line Utility: nc

- Netcat (nc), an utility for reading from and writing to network connections using TCP or UDP.
- It can be driven by other programs or scripts.
- Good tool for network debugging and investigation.
 - E.g., Scan all ports up to 100

```
gloin:~> nc -z -v dori.cs.colby.edu 1-100
```

- REF
 - <https://en.wikipedia.org/wiki/Netcat>
 - <https://www.digitalocean.com/community/tutorials/how-to-use-netcat-to-establish-and-test-tcp-and-udp-connections-on-a-vps>
- Another application of using nc is to transfer files.

Client

```
mbp-190250:~ yingli$ echo "Hello gloin" > test.txt
mbp-190250:~ yingli$ cat test.txt
Hello gloin
mbp-190250:~ yingli$ nc gloin.cs.colby.edu 12345 < test.txt
mbp-190250:~ yingli$
```

Server

```
gloin:~> nc -l 12345 > output.txt
gloin:~> cat output.txt
Hello gloin
gloin:~>
```

- In the above example, I was trying to send a file, test.txt, from my laptop to gloin, and saved the file into output.txt.
- The gloin is the server in this case, waiting for the file from client, my laptop, listening on port 12345.
- My laptop is the client in this case, which connects to the server on port 12345 and sends files to it.

- E.g., SSH to dori and gloin for the example.

```
dori:~> nc gloin.cs.colby.edu 12345
Hi gloin, this is dori
Hi dori, I'm gloin, nice to meet you
```

Client

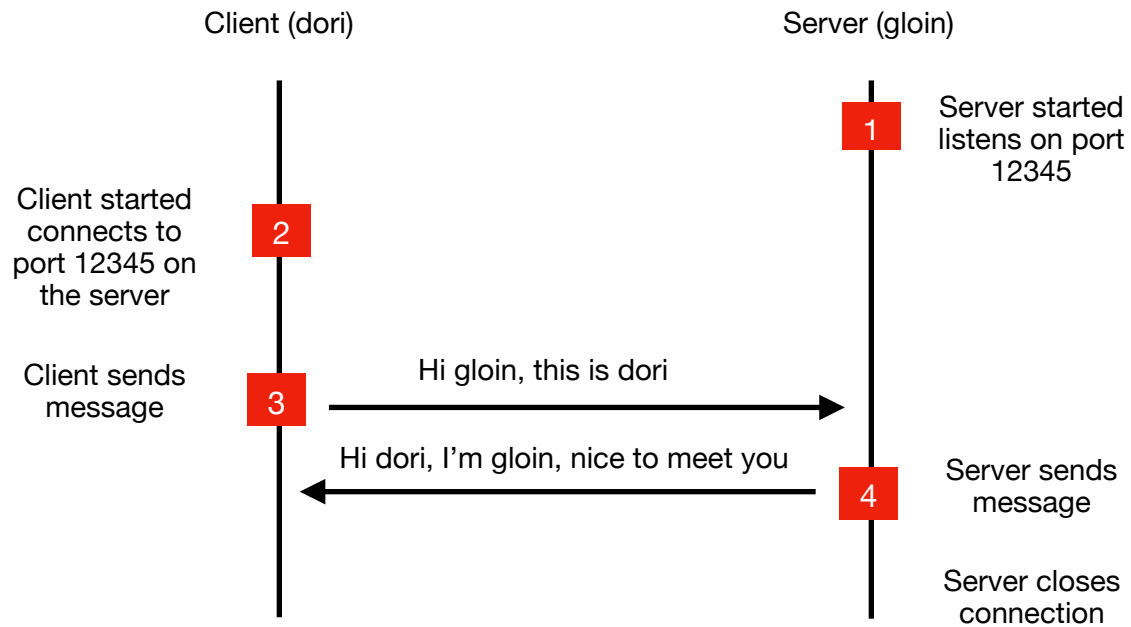
2. connecting the client to a server on gloin on port 12345
3. typing a message to the server "Hi gloin, this is dori"

```
gloin:~> nc -l 12345
Hi gloin, this is dori
Hi dori, I'm gloin, nice to meet you
```

Server

1. starting server on port 12345
4. typing response to the client and breaking the connection by type CTRL + C

- Sequence diagram for the above example



Disclaimer: Notes adapted from the textbook and online resources.