Lecture 3 (Application Layer)

When creating a network app it is enough to write programs that runs on the end systems and communicates over the network. The layered architecture helps in avoiding writing code to interact with *network-core devices* (such as switches and routers).

1 Principles of Network Applications

1.1 Network Paradigms

1.1.1 Client-Server Paradigm

1.1.1.1 Server

1. Always on host
2. Permanent IP address
3. Usually present in data centres

1.1.1.2 Client

1. Intermittently connected
2. May have dynamic IP addresses
3. Do not communicate directly with each other

1.1.2 Peer-to-Peer Paradigm

1. No always-on server
2. Direct communication
3. Peers request service from and provide service to other peers on the network (self-scalability)
4. Intermittent connection

1.2 Process Communication

This is of two types:

1. Inter-process communication (handled by the OS)
2. Inter-host communication (via networking)
    i. Client process: initiates communication

ii. Server process: waits to be contacted

Note: Applications with P2P architecture have both client and server processes.

1.3 Socket

1. Process sends/receives messages to/from its socket
2. It is analogous to a door:
    - Sending process *shoves* message out of the door
    - Relies on transport infrastructure on the other side of door to deliver message to socket at receiving process
    - Two sockets involved - sending and receiving sockets

1.4 Addressing Processes

1. Each host device has a unique 32-bit IP address, that acts as an *identifier*
2. TO address the issue of multiple processes running on the same host, the concept of **port number** is used

1.5 Information in the Header(?)

1. Type of message exchanged: request/response
2. Message syntax: what fields present in the message
3. Message semantics: what do the fields mean
4. Rules: when and how processes send and respond to messages

Various open-protocols defining the syntax for the above information are available:

- HTTP
- SMTP
- FTP

1.6 Requirements of Network Applications

1. Data integrity: some require lossless transmission, some can tolerate losses
2. Delay: some require low delay to be effective
3. Throughput: some apps need a minimum throughput
4. Security

1.7 Internet Transport Protocols

1.7.1 Transmission Control Protocol (TCP)

1. Reliable transport between sending and receiving processes
2. Flow control 9to not overwhelm the receiver
3. Congestion control

4. Connection-oriented: setup needed before communication
5. No timining, throughput, security guarantee

1.7.2 User Datagram Protocol (UDP)

1. Unreliable data transfer
2. Does not provide any other feature provided by TCP

2 HTTP

1. Follows a **client-server model**
2. Uses **TCP protocol**:
    i. Client initiates TCP request to server at port 80
    ii. Server accepts TCP connection
    iii. HTTP messages exchanged
    iv. TCP connection closed
3. It is **stateless**: no information about the client is stored at the server
4. This has two types:
    i. **Non-Persistent HTTP** - atmost one object is sent for each connection
    ii. **Persistent HTTP (HTTP 1.1)** - multiple objects can be sent over a single TCP connection
5. Two types of HTTP messages:
    i. **Request**
    ii. **Response**

2.1 HTTP Request

```
<method> <URL> <version> \r\n
<header field name> <value> \r\n
.
.
.
<header field name> <value> \r\n
\r\n
<body>
```

There are two types of methods:

1. **POST method** - data is sent in the body of the request
2. **GET method** - data is sent in the URL field
   (There is a concept of **conditional GET** in which the response is sent only if content is modified since the requested date)
3. **HEAD method** - only request for headers (as response to a *GET method*)
4. **PUT method** - replaces file at URL with body of *POST method*

2.2 HTTP Response

This contains the response code of the request.

2.3 Storing Information about Client

This is done using **cookies**. This is made up of four components:

1. Header line of HTTP response
2. Header line of next HTTP request
3. Cookie is stored on client and is managed by the browser
4. Corresponding information also stored in server backend

2.4 Web Cache (Proxy Servers)

1. Browser is configured to point to a cache
2. All HTTP requests are sent to the cache:
   i. *if* request is found, cache returns object to client
   ii. *else* request is sent to origin server and response cached + returned to client