Chapter 2

- 🔴 Network architecture
  - Client-server architecture
  - Peer-to-peer networks
  - N-tier network architectures
- 🔴 Application layer protocols
  - Email
    - X.400
    - SMTP - Simple Mail Transfer Protocol
    - POP3 - Post Office Protocol version 3
    - IMAP4 - Internet Message Access Protocol version 4
  - Virtual terminal protocols
    - Telnet
    - SSH - Secure Shell
    - FTP - File Transfer Protocol
  - The web
    - HTTP - Hypertext Transfer Protocol
    - HTTPS - Hypertext Transfer Protocol Secure
  - Other application layer protocols
    - NTP - Network Time Protocol
    - VoIP - Voice over Internet Protocol
    - SMB - Server Message Block
    - NFS - Network File System
    - RTSP - Real Time Streaming Protocol
    - X Window protocols

🔴 Network architecture

The manner in which ‘services’ are arranged and are accessed

Client-server architecture

- Servers provide services
- Clients use software that use services offered by servers
- Clients send *requests* and await *responses*
- Servers send *responses* and await *requests*
- **Example**: *Web page* and *web server*

> 💡 Waiter⏱️ and server🧆

- Several requests may be sent
- Server software often referred to as *daemons*
  - *Daemon* listens to port, accepts incoming requests and handles them while the computer is on
- FTP executable program:
  - `ftp`
  - `ftpd` (d - daemon)
- Web client requests files from web server
- **Major drawback** of client-server architecture: **THE CLIENT ALWAYS INITIATES COMMUNICATION**
- *Peers* are required in such systems
  - *Peers* are equals and can initiate communication to one another whenever required

Peer-to-peer networks

A network architecture that enables different participants to work as ‘equals’ or ‘peers’ in a network

---

- Each peer carries some of the workload
- *Napster* referred to as first popular application that used P2P
  - Much harder for copyright holders to break a sharing system if there were no central server
  - Massive number of files stored across multiple users, with each user storing a small number of files
- **Mobile ad-hoc sensor networks (MANETS)**
  - MANET node - tiny computer that runs from small batteries
  - Nodes communicate with each other wirelessly
  - Nodes equipped with sensors to detect movement, temperature, dampness or whatever else required
  - Impossible to predict which nodes will be able to communicate with which nodes
  - Nodes could go to sleep

N-tier network architectures

Client-server : 2-tiered

P2P : 1 tiered

- Develop systems in ‘chunks’ to separate logics from each other, resulting in N-tier networks
- **Important 3-tier architecture:**
  1. Client
  2. Server
  3. Business data (back-end)

🔴 Application layer protocols

Email

- X.400
- SMTP (Simple Mail Transfer Protocol) - predominant email protocol

Important port numbers:

SMTP : 25

POP3 : 110

IMAP4 : 143

TELNET : 23

SSH : 22

FTP : 21

HTTP : 80

HTTPS : 443

X.400

- Deployed around 1995
- Series of ITU-T standards
- ISO also published as standards
- Was designed for ISO/OSI model and later adapted for TCP/IP networks
- Snail mail addresses
- These email addresses include the following:
  - C (Country) field - specifies country
  - A (Administration Management Domain) field - points to post office or telecommunications office
  - P (Private Management Domain) field - private service provider
  - O (Organisation Name) field - identifies organisation
  - OU (Organisational Unit) field - indicate department within organisation

- G (Given name) field - name of user
- I (Initials) field - initials of user
- S (Surname) field - surname of user

```
//J Doe working in the sales department of Acme
I=J;S=Doe;O=acme;OU=sales;A=sapo;C=za
```

> 💡 **EDI - Electronic Data Interchange**
> The interchange of official business documents, such as invoices, requests for quotations and customs documents

SMTP - Simple Mail Transfer Protocol

- Most widely used protocol to transfer emails
- 1980
- Port 25
- Refined by RFC 5321
- Was not developed with security in mind
- Two agents
  - *MUA* (Mail User Agent) - user interacts with email system
    - Piece of code that accepts a message, address of sender and some other relevant inputs. It then forwards it to the MTA
    - Recipient UA accepts incoming email and typically stores it in a specific directory on a (shared) computer system where the user receives emails.
  - *MTA* (Mail Transfer Agent) - transfer of emails
    - Need to find a route from the sender to the UA of recipient and forward the email along the route
    - For security, configured to only accept emails from users who are connected to a specific network
    - `Received` header line to emails they processed

```
Received: from EUR05-DB8-obe.outbound.protection.outl
ook.com
by mx.google.com
```

- **Interaction between client and SMTP - conversation**

1. Client connects to server and server responds with a line of text which often contains information about the server, but a condition code could be returned.
2. Client greets the server with `HELO (normal helo)` or `EHLO (extended helo)`.

```
ehlo = "EHLO" SP ( Domain / address-literal ) CRLF
helo = "HELO" SP Domain CRLF

ehlo-ok-rsp = ( "250" SP Domain [ SP ehlo-greet ] CRLF )
              / ( "250-" Domain [ SP ehlo-greet ] CRLF
                 *( "250-" ehlo-line CRLF )
                 "250" SP ehlo-line CRLF )
```

> 💡 Response codes:
>
> **200 - 299** : Success codes
>
> **300 - 599** : Error codes

3. Client indicates who the sender is.

```
mail = "MAIL FROM:" Reverse-path
       [SP Mail-parameters] CRLF
```

   If server responds with `MAIL FROM:` message with a positive response code, the client can proceed.

4. Client needs to identify intended recipients with `RCPT TO:`

```
rcpt = "RCPT TO:"
          ( "<Postmaster@" Domain ">" /
            "<Postmaster>" /
            Forward-path )
                  [SP Rcpt-parameters] CRLF
```

`Forward-path` refers to email address of a recipient.

5. Client sends email contents with `DATA` message and send the email message one line at a time without waiting for a response.
6. Client ends message with a full-stop, carriage return and linefeed.
7. Client sends `QUIT` message to server to close the connection.

POP3 - Post Office Protocol version 3

- Stores emails locally on user's computer
- Local mailbox on user's computer
- Emails will be download and stored locally as long as email program is open
- Uses SMTP to send emails in outbox
- Port 110
- Protocol function in 2 stages: Authorization and Transaction
  - Authorization
    - User supplies username and password (cleartext)
    - Use `APOP` to avoid transmission of sensitive information as cleartext
  - Transaction
    - Commands to be used:
      - `RETR 1` : used to retrieve first message in the queue on the server
      - `DELE 1` : delete first message in queue
      - `STAT` : number of messages waiting on server
      - `RETR 1, X` : retrieves X number of lines of message 1

IMAP4 - Internet Message Access Protocol version 4

- Often seen as alternative to POP3
- Users manage mailboxes on email server, rather than locally
- Users may create folders or directories for different email categories
- Used to manage structured set of folders on email system
- Inbox is one of the folders, where incoming email is stored automatically
- User downloads email to access it, but email is still stored on email server, not locally
- Port 143
- **Authorisation process:**
  1. User sends `LOGIN` message
  2. User selects folder to use with `SELECT`
  3. Email is copied to client via `FETCH`
  4. Terminate session with `LOGOUT`

  Example of `FETCH` message syntax:

```
fetch = "FETCH" SP sequence-set SP (
        "ALL" / "FULL" / "FAST" /

fetch-att / "(" fetch-att *(SP fetch-att) ")")
```

  - `sequence-set` : list of emails
  - `ALL` `FULL` `FAST` : macros to determine what needs to be fetched
    - `ALL` : indicate flags, date, size and header of email should be downloaded
    - `FULL` : indicate flags, date, size, header and body of email should be downloaded
    - `FAST` : indicate flags, date and size of email should be downloaded
  - Can also specify individual attributes:

```
fetch-att = "ENVELOPE" / "FLAGS" / "INTERNALDATE" /
            "RFC822.SIZE" /
            "BODY" ["STRUCTURE"] / "UID" /
            "BODY" section [partial] /
            "BODY.PEEK" section [partial] /
            "BINARY" [".PEEK"] section-binary [partial] /
            "BINARY.SIZE" section-binary
```

`FLAGS`:

```
flag = "\Answered" / "\Flagged" / "\Deleted" /
       "\Seen" / "\Draft" / flag-keyword / flag-extension
```

Virtual terminal protocols

- Terminals used to send requests to computer and responses were displayed on terminal screen
- Character orientated
- Lack of processing - *dumb* terminals
- Downside - Terminals were connected to local computers
- Virtual terminal protocols were developed to alleviate this problem
- One user would type in their terminal, but it would display in another user's terminal via the network

Telnet

- Best-known VT protocol
- Telnet server provides information about computer on which it is running and displays the following

```
Login: _

//Characters are sent to the server and echoed back to t
he user
```

- Once the user hits enter, the following is displayed:

```
Password: _

//Characters are not echoed back to the user, so the use
r cannot see any characters entered
```

- Telnet is deemed **unsafe** as it uses **cleartext** for the username and password
- Telnet servers were commonly employed on Unix
- Port 23
- Even though these terminals were referred to as *dumb* terminals, they had some processing power like:
  - Clearing the screen
  - Positioning cursor at a specific spot
  - Changing text colour
  - ANSI escape sequences
    - `ESC [2J` - clear the screen
    - `ESC [y;xH` - move cursor to position `(x,y)` on the screen
- Available on most computers
  - `telnet dest-host` where `dest-host` is the host running a Telnet server
  - `help` to display available commands
  - **useful command**: `set localecho` - characters will be displayed immediately when they are typed (password will also be displayed when typing)
- **Some important commands**:

```
Microsoft Telnet> help

Commands may be abbreviated. Supported commands are:

c    - close                 close current connection
d    - display               display operating parameters
o    - open hostname [port]  connect to hostname (default port 23).
q    - quit                  exit telnet
set  - set                   set options (type 'set ?' for a list)
sen  - send                  send strings to server
st   - status                print status information
u    - unset                 unset options (type 'unset ?' for a list)
?/h  - help                  print help information
Microsoft Telnet>
```

🔺 **Telnet can be configured to run on any port**

Using Telnet as a client or server development tool

- Non-standard use of Telnet client
- Very useful to develop clients or servers and learning about networks

> 💡 **Telnet client** - a program that accepts characters that are typed as input and sends them to some port at some destination address

- Can use Telnet client to interact with many servers to 'see' communication under the hood
- **Example**:

`telnet localhost 25`

```
1 Trying 127.0.0.1...
2 Connected to localhost.
3 Escape character is '^]'.
4 220 server.example ESMTP Exim 4.93 Ubuntu
```

```
 5 helo localhost
 6 250 styx.mso Hello localhost [127.0.0.1]
 7 mail from: sam@example.com
 8 250 OK
 9 rcpt to: sam@example.net
10 250 Accepted
11 data
12 354 Enter message, ending with "." on a line by itself
13 I just mailed to say I love you!
14 .
15 250 OK id=1pVDaM-000CU6-I2
16 quit
17 221 server closing connection
18 Connection closed by foreign host.
```

- Server takes care of line editing and the user cannot make mistakes
- Server will see the line as erroneous and user will have to start over

SSH - Secure Shell

- More secure
- Encrypts all traffic
- Server configured to only accept certain authentication methods
- Shipped with most operating systems
- Popular *tunnelling* protocol
- File transfer capabilities with `scp` (secure copy) command
- Port 22

FTP - File Transfer Protocol

- Used for transferring files across network
- Shipped with most operating systems with `ftp` program in cmd
- Use `get` and `put` commands to upload and download files (`mget` and `mput` for multiple files)
- Can specify patterns of file names (`mput *abc*`)
- Use `lcd` to change **local** directory and `cd` to change **server** directory
- FTP client commands differ from FTP protocol commands

    - `get` - `RETR`
    - `put` - `STOR`
- User must be authorised to upload or download files
- Username `anonymous` used to download and upload **public** files
    - Can only use of server is configured for anonymous use
    - Server will ask user for email address for record keeping purposes
- Some servers check the email address, some don't
- **Example of anonymous FTP**:

```
> ftp redacted.ac.za
Connected to redacted.ac.za.
220---------- Welcome to Pure-FTPd [privsep] [TLS] ----------
220-You are user number 7 of 5000 allowed.
220-Local time is now 11:22. Server port: 21.
220-Only anonymous FTP is allowed here
220-IPv6 connections are also welcome on this server.
220 You will be disconnected after 2 minutes of inactivity.
User (redacted.ac.za:(none)): anonymous
230 Anonymous user logged in
ftp>
```

- Pure client-server connection is problematic
    - If a user requests a large file, the user will have to wait for server to respond as the server cannot be interrupted
    - Solved by using the following:
        - Control connection
            - Authentication and sending commands in general
        - Data connection
            - Transferring of files
    - While file is transferring, a new command can be used to abort the transfer of files with data connection
- When FTP connection is established:
    - Control connection opens on port 21
    - As soon as file is to be transferred, client listens on port 20 for a data connection

    - Client begins to act as server and vice versa
    - `RETR` - file sent from ‘server’ program to ‘client’ program
    - `STOR` - file sent from ‘client’ program to ‘server’ program
    - `ls` command lets server compile a file with the names of all files in the current directory on the server
        - Example:

```
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
README.html
mirrors
pub
robots.txt
226 Directory send OK.
ftp: 42 bytes received in 0.00Seconds 42.00Kbytes/sec.
ftp>
```

- Firewalls were eventually installed to prevent incoming connection from ‘servers’ inside companies
    - This broke FTP
- FTP *passive mode*
    - FTP server on outside requests new port and starts listening on that port
    - Client on inside is notified of the port number via control connection
    - Client opens data connection to the outside server on that port, which allows the client to open both control and data connections as client
    - `PASV` command used to enter *passive mode*
    - Most modern FTP clients automatically issue `PASV` command and operate in *passive mode* by default

The web

- Tim Berners-Lee created a system with hyperlinks
- *Web* is an application (using specific protocols) that use the *Internet* (network consisting of network layers)

HTTP - Hypertext Transfer Protocol

- Primary protocol used to build the web
- Port 80
- HTTP messages consist of headers and a payload

```
1 GET / HTTP/1.1\r\n
2 Host: example.com\r\n
3 Connection: keep-alive\r\n
4 User-Agent: Mozilla/5.0  ... AppleWebKit ...\r\n
5 Accept: text/html,...\r\n
6 Accept-Encoding: gzip, deflate\r\n
7 Accept-Language: en-ZA,en;q=0.9...\r\n
8 \r\n
```

  - 1: Indicates default file should be fetched in the root directory (/) with HTTP version 1.1. The line is terminated with (\r\n)
  - 2: Tells HTTP server which site should be served
  - 3: Requests server to keep connection open for subsequent requests
  - 4: Identifies the browser or web engine that is used
  - 5: Describes the type of content browser is looking for
  - 6: Preference for encoding
  - 7: Preference for language
  - 8: Blank line to terminate request
- **Response:**

```
 1 HTTP/1.1 200 OK\r\n
 2 Content-Encoding: gzip\r\n
 3 Age: 429553\r\n
 4 Content-Type: text/html; charset=UTF-8\r\n
 5 Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT\r\n
 6 Server: ECS (bsa/EB16)\r\n
 7 Vary: Accept-Encoding\r\n
 8 Content-Length: 648\r\n
 9 \r\n
10 <!doctype html>\n
11 <html>\n
12 <head>\n
13     <title>Example Domain</title>\n
14 ...
15 </html>\n
```

HTTPS - Hypertext Transfer Protocol Secure

- Identical to HTTP, but uses encryption
- Port 443
- Uses encryption on the transport layer
- SSL (Secure Sockets Layer) and TLS (Transport Layer Security)
    - Use layer 4 to provide encrypted endpoint-to-endpoint connection between processes as encrypted transport service

Other application layer protocols

NTP - Network Time Protocol

- Used to obtain current time from some servers
- Does not cater for ETA so response time is always ‘old’. Protocol uses mechanisms to estimate correct time.

VoIP - Voice over Internet Protocol

- Intended for transmitting telephone calls over the network
- SIP (Session Initiation Protocol) used to establish a telephone connection

SMB - Server Message Block

NFS - Network File System

- Mount remote file systems on a local host
- Popular in Microsoft environments

- Mount remote file systems on a local host
- Popular in Unix contexts

RTSP - Real Time Streaming Protocol

- Stream multimedia entertainment to users
- Often used for internet radio

X Window protocols

- Enables system to display graphical content on remote devices
- System instructs server to draw figures or display text at specific coordinates of the display
- Tunnelled via SSH

🔴 Sniffing traffic

- *WireShark* - tool for sniffing
- Most LANs will use Ethernet switches rather than a hub
- Switch normally only sends traffic to one’s own computer
- Sniffing is valuable when developing software using network protocols