Networks

iGCSE Computer Science

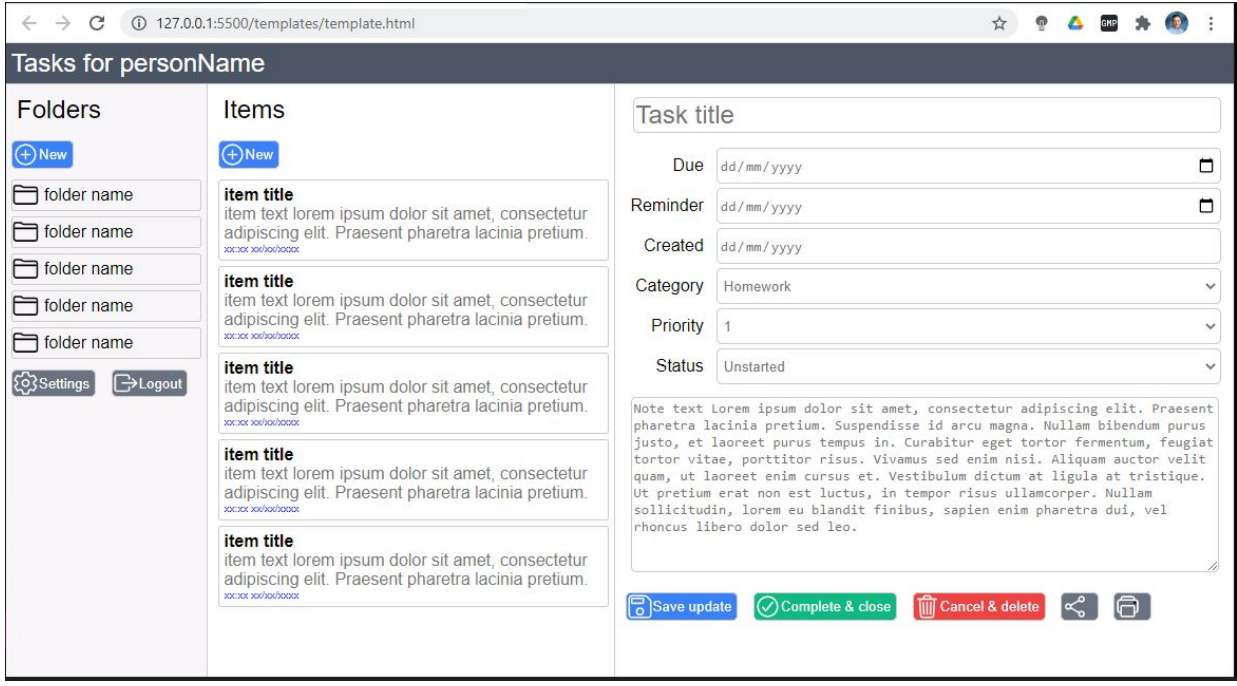
Unit overview

- Network types
- The Internet
  - MAC addresses, IP addresses
  - HTTP request life cycle - the URL, DNS, HTTP, HTML & CSS, cookies
- Network addresses
- Browsers and web traffic
- Serial & parallel communication
  - USB
- Error checking methods
  - Parity, check digits, check sums, ARQs

Project overview

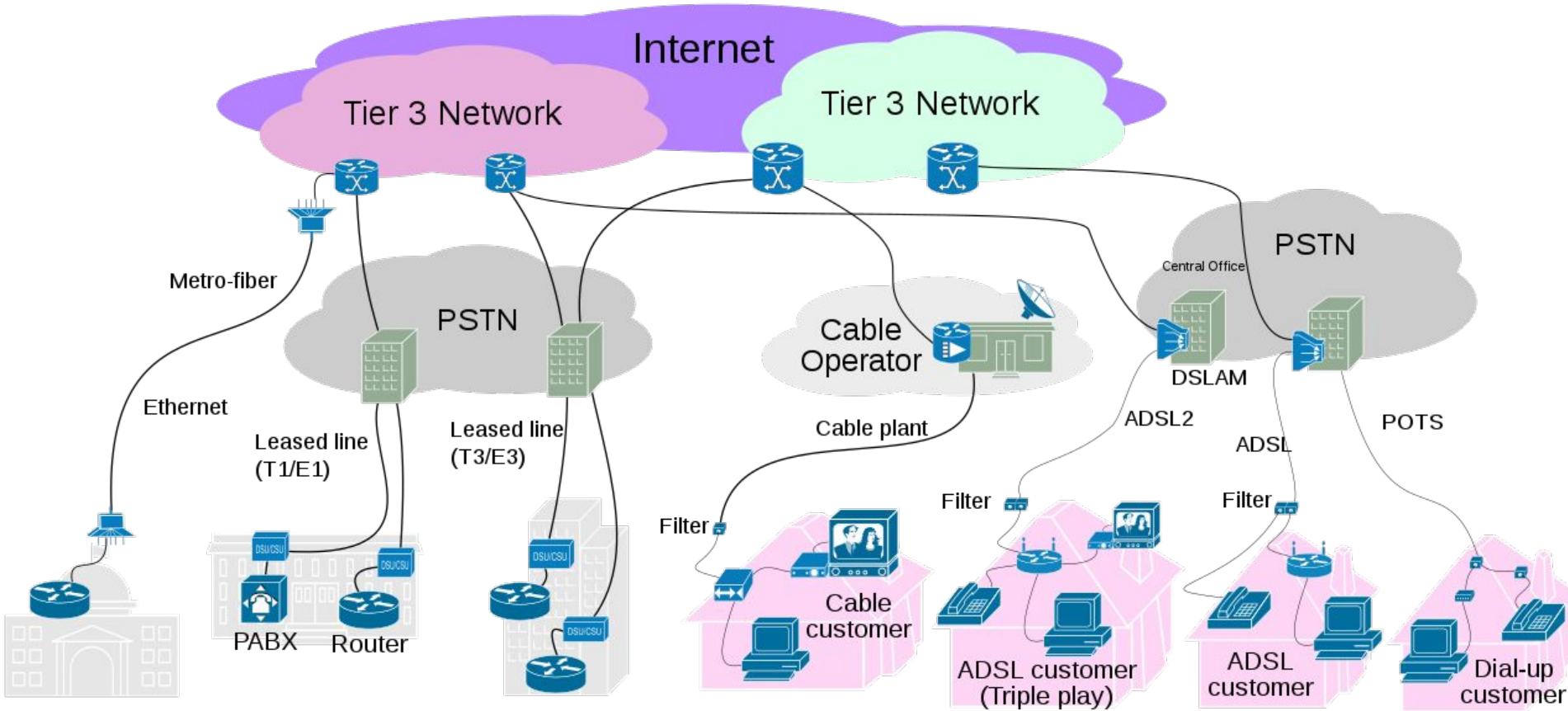
As a practical introduction to networks, we will build a web server and web application to run on it.

This exercise will continue into the next couple of units when we look at databases and security as well.



The internet

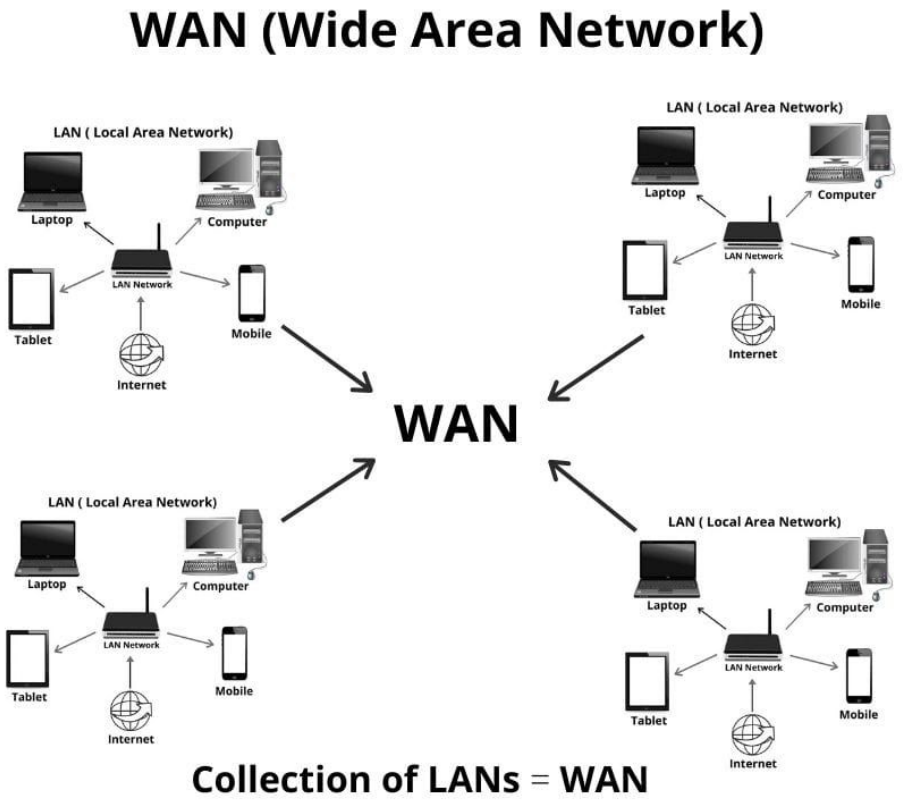
Internet: The big picture



Local & wide area networks

A LAN could be your home, or our school campus.

A WAN is the interconnection of several LANs across a geographically wide area.



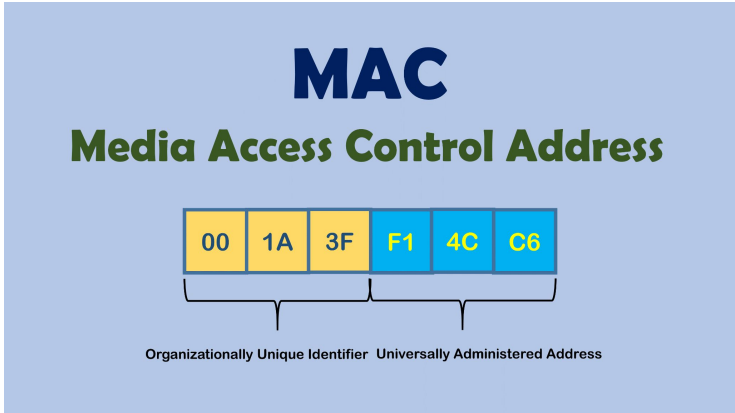
MAC address

A unique number assigned to every NIC (network interface card). Think of it as a serial number for your NIC. Used at by devices on the same hardware network as you to communicate.

To find the MAC addresses of your cards:

- Windows 10 command prompt `ipconfig /all`
- Mac OSX terminal: `ifconfig -a`
- Python:

```
import uuid
mac = uuid.getnode()
print("Mac address: ", hex(mac))
```



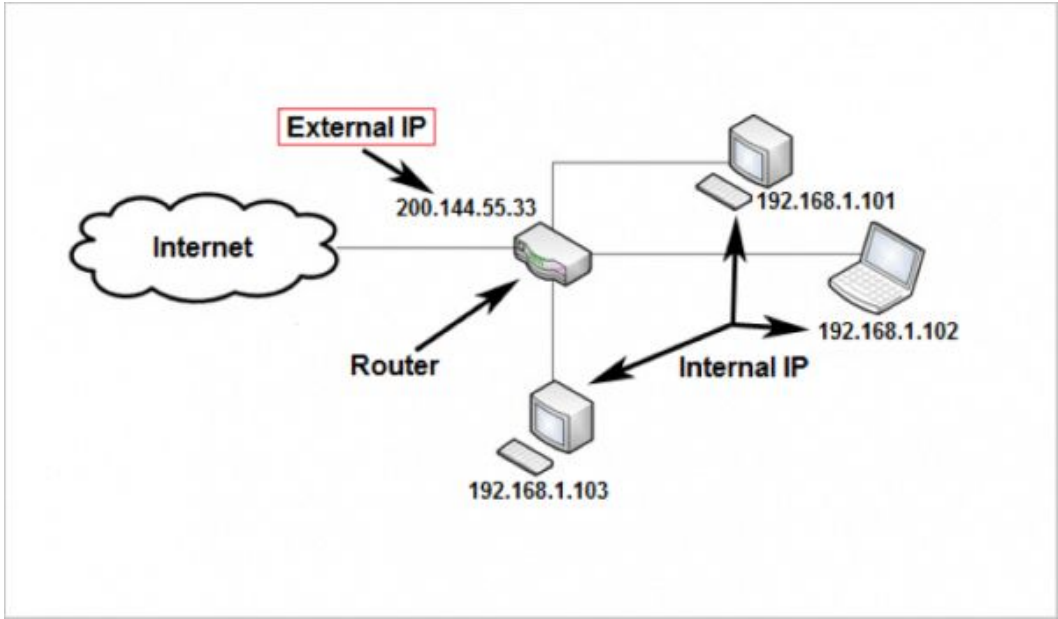
IP address

Before you can communicate on the internet, you require an IP address.

It must be unique in order to communicate properly. Assigned by your ISP to avoid collisions.

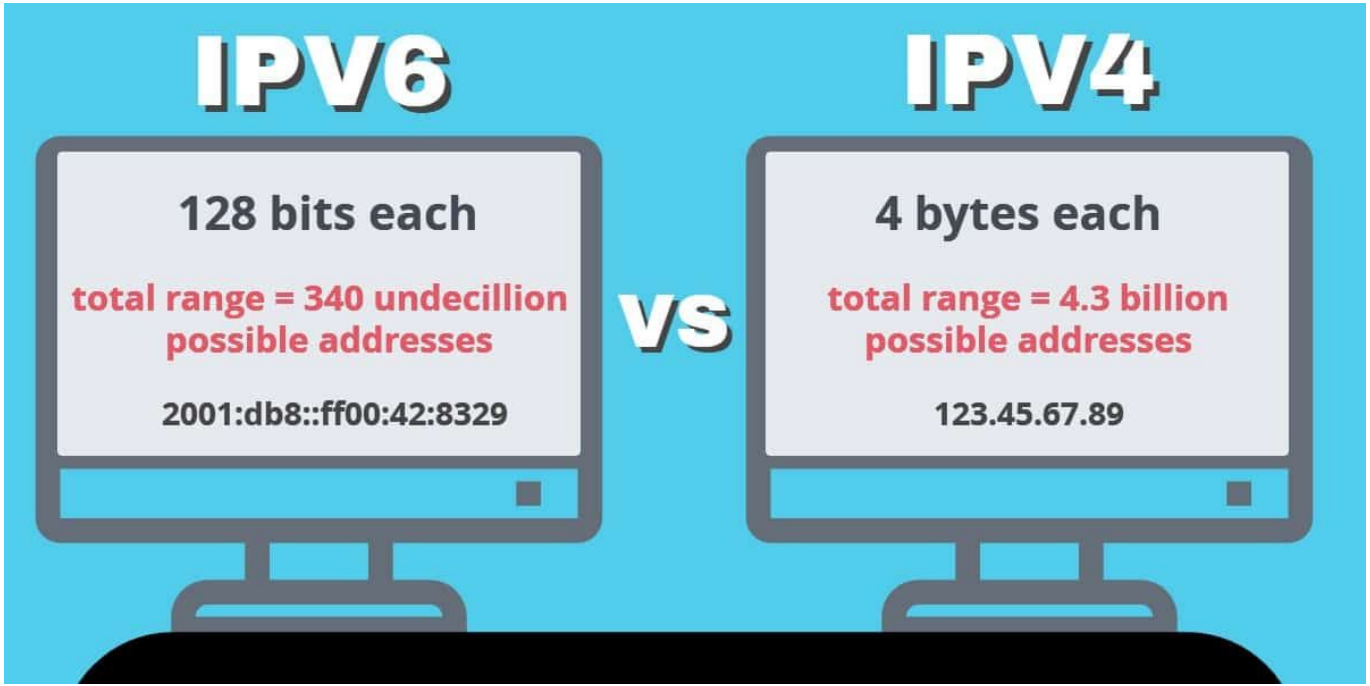
Reality is IP4 addresses are exhausted so tricks such as Network Address Translation are used to mitigate.

Your router hosts a private network and translates your requests to a shared external IP address. It processes replies to determine which internal computer to send forward the response to.



IP address

Long term solution is the rollout of IP6.



IP address

To find the IP addresses of your cards:

- Windows 10 command prompt `ipconfig /all`
- Mac OSX terminal: `ifconfig -a`
- Python:

```
import socket
ip = socket.gethostbyname(socket.gethostname())
print("IP address : ", ip)
```

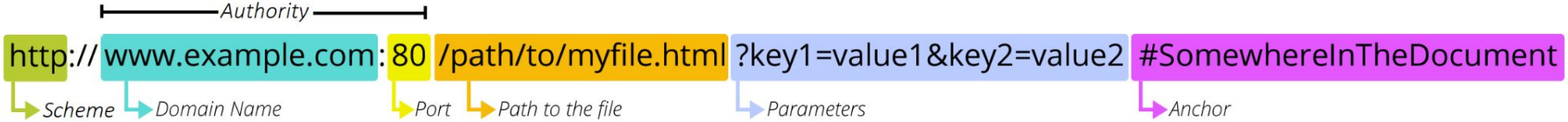
Web browser

Key steps in the web browser process

- 1. You enter a requested address, known as a URL.
- 2. Resolve the domain name to an IP address
- 3. Send HTTP request for the document to the server at that address
- 4. Receive HTML or other response
- 5. Parse it and render to your screen

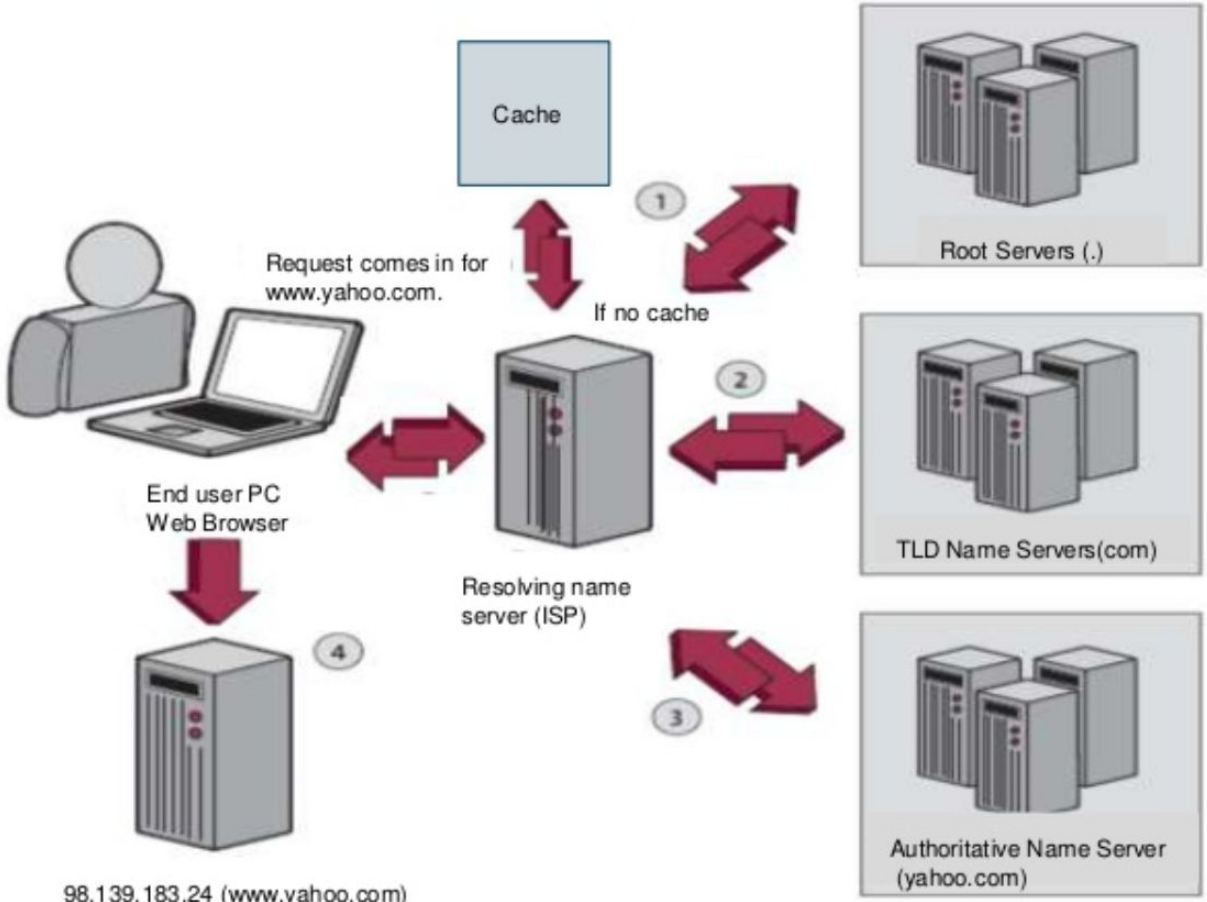
The next few slides will elaborate on these steps.

URL



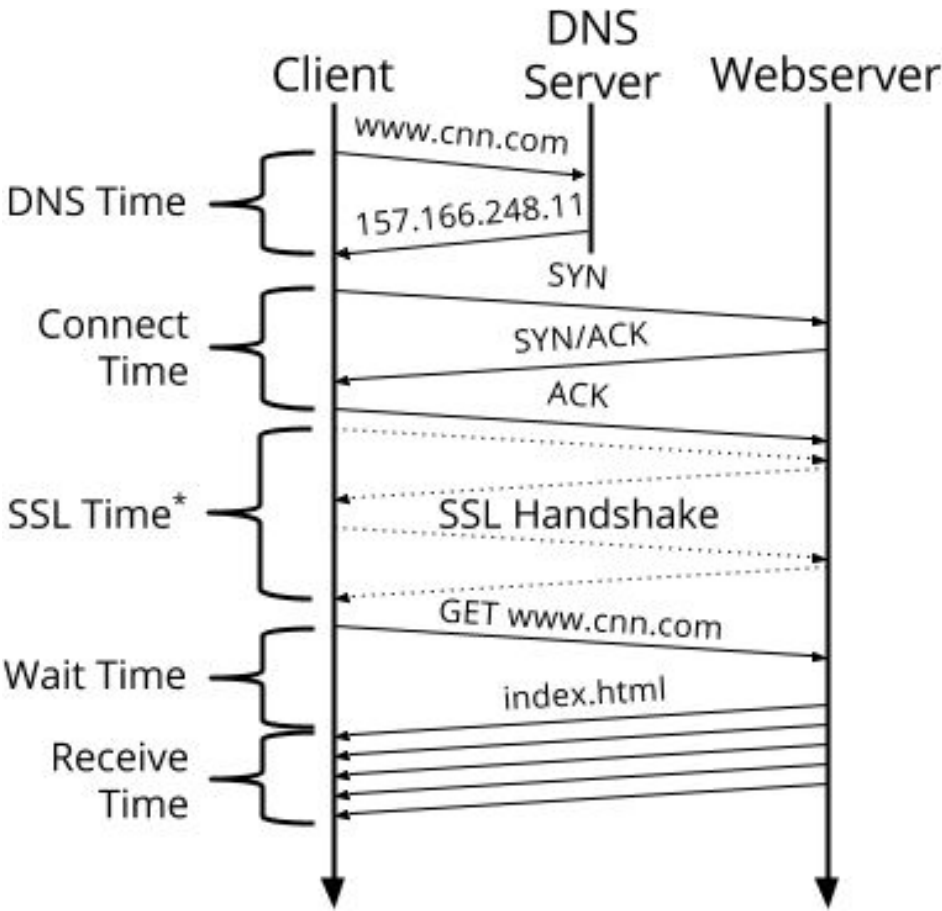
- Scheme, also known as the protocol. Generally **http** or **https**
- Domain. Uses DNS to convert to an IP address (network address of an individual computer)
- Folder & file requested. If not provided then the server running on the target computer will supply a default page.

DNS


Cache
1
Request comes in for
www.yahoo.com.
If no cache
Root Servers (.)
2
End user PC
Web Browser
Resolving name
server (ISP)
TLD Name Servers(com)
4
3
Authoritative Name Server
(yahoo.com)
98.139.183.24 (www.yahoo.com)

HTTP and HTML

Lifecycle of a request



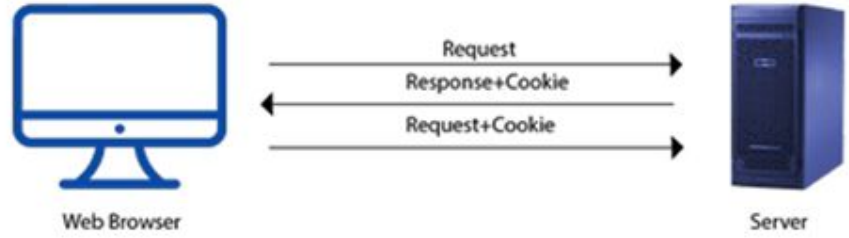
HTML & web content

HTML

HTML documents are generally split into many parts but can be thought of in 3 layers:

- **Structure**: The HTML file itself. It contains the content and structure of the document such as headings, paragraphs, tables.
- **Presentation**: Stylistic detail such as font, colour. These days this is 99% done through CSS.
- **Behaviour**: Javascript programming to improve the user interactive experience

Cookies



It's all too real: https://twitter.com/5tevieM/status/1375116382770171906

A small file placed on the browsing computer, by the web server.

Automatically attached back to any future request made to that server. It facilitates things like “logging on” to a website without having to enter your password every page you visit.

Rather than transmitting your password, the server typically creates a session number unique to your visit, so future visits you send that number and it knows it is you.

It can also be used to track your browsing habits, curate advertising etc.

Cookies are not programs. They cannot perform any operations, they are not viruses or malware. Cookies can be disabled in your browser settings, however this could make some websites unusable (anything that requires logins or to remember your settings).

Other networking

Simplex, Duplex communications

Simplex data transmission - Communication between devices is in one direction.

- Garage door opener
- Wireless microphone
- TV, radio signals from broadcaster to your receiver
- TV remote control

Half-duplex data transmission - Signals travel in both directions but only one transmits at a time.

- Two way radios “walkie-talkie”

Duplex (or Full Duplex) data transmission - Signal travels in both directions simultaneously.

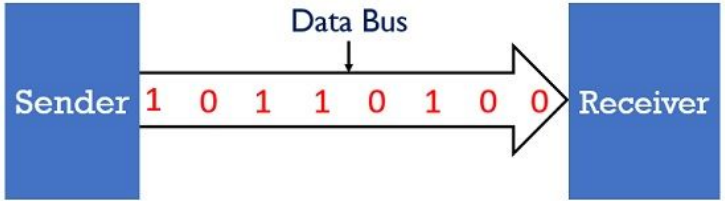
- A phone call

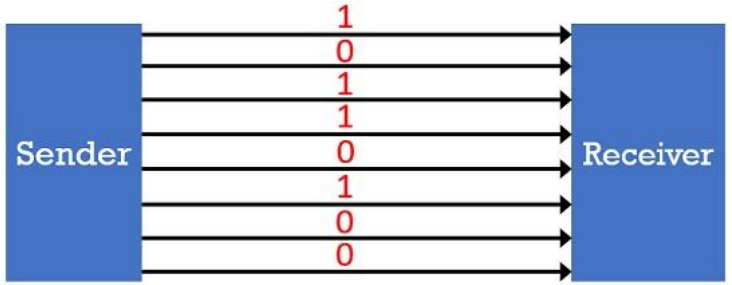
Serial & parallel communication

Serial: 1 bit at a time in a continuous flow.

Parallel: Sending multiple bits per clock pulse. Eight wires would allow 8 bits (1 byte) per signal pulse.

Serial interfaces are simpler, therefore cheaper to setup



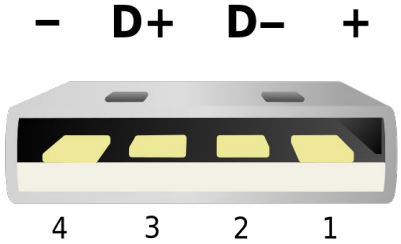


USB

USB is a modern serial interface used for connecting peripheral devices.

Data is transmitted sequentially, in a half-duplex asynchronous manner.

One of the improvements of USB 3+ is it is actually capable of full-duplex.



Face off: serial v parallel

Why is it that serial based communications (such as USB) are so dominant at the moment? Wouldn't parallel make better sense to be faster? (send more information per pulse)

Overhead: Since bits are transferred simultaneously, we have line up them in an order at the receiver, so parallel data gets converted to serial form anyway.

Synchronisation: While all bits leave at same time, they may not be received together. Slight differences in path length and electrical properties can cause the bits to no longer aline with each other (fractions of a millimeter make a difference when sending megabits per second). This synchronization requires the receiver to wait until all bits received. Parallel must therefore run slower than equivalent serial.

Crosstalk: When wires lie parallel to each, the signal in wires may get attenuated or disturbed due to electrical induction (electromagnetic interference) reducing accuracy of transmission at distance and speed.

Error checking

Why check for errors?

Network based communications are very error prone. It is amazing we have developed a world where high bandwidth data networks are so ubiquitous given all the environmental and technical challenges involved in getting it to work so seamingly seamlessly!

Electrical interference, damaged cables, weather, radio signals - can all interfere in data transmission over wire based or wireless based methods. The following is an overview of some common error detecting and correcting methods used.

Parity bits

Parity bits are used as the simplest form of error detecting code. Parity bits are applied to the smallest units of a communication protocol, typically bytes, although they can also be applied separately to an entire message string of bits.

7 bits of each byte are used for data, 1 bit is used for a parity check. The parity bit is usually the most significant bit (the left most of the 8).

An agreed protocol is established between the sender and receiver as to odd or even parity.

- Even parity - set the parity bit will be set to ensure an even number of 1 bits.
- Odd parity - set the parity bit will be set to ensure an odd number of 1 bits.

The receiver will check the byte received to ensure parity matches. If not, there was an error in transmission.

Check digits

We've looked at check digits already.

- Examples: credit card numbers, HKID numbers and ISBN numbers.

An algorithm is used to generate a digit that is added to the data to verify it has been correctly received.

Most common check digit algorithms are somewhat based on the idea of using a rule similar to the following though you will remember credit cards and HKID used a slightly more complicated algorithm.

```
Check digit = (the sum of the digits) % 10
```

Check sums

Note: The “algorithm” given in *example 3, figure 2.15, page 21* of the book is dumb.

Adding extra data that can be used to verify the successful transmission of the message.
Typically one byte, but it could just as easily be 16 bit, 32 bit or 64 bit (protocol dependant).

Calculating the checksum for data is straightforward. Assuming we are using bytes:

```
Checksum = (Sum the values of all bytes being transmitted) % 256
```

Text	`To infinity and beyond.`
Hex	`54 6F 20 69 6E 66 69 6E 69 74 79 20 61 6E 64 20 62 65 79 6F 6E 64 2E`
Bytes	`01010100 01101111 00100000 01101001 01101110 01100110 01101001 01101110 01101001 01110100 01111001 00100000 01100001 01101110 01100100 00100000 01100010 01100101 01111001 01101111 01101110 01100100 00101110`
Decimal	`84 111 32 105 110 102 105 110 105 116 121 32 97 110 100 32 98 101 121 111 110 100 46`

Summing the values together gives 2159.

2159 % 256 = 111

Therefore the checksum will be 6F hex or 111 decimal.

Check sums

Example

Text	`To infinity and beyond.`
Hex	`54 6F 20 69 6E 66 69 6E 69 74 79 20 61 6E 64 20 62 65 79 6F 6E 64 2E`
Bytes	`01010100 01101111 00100000 01101001 01101110 01100110 01101001 01101110 01101001 01110100 01111001 00100000 01100001 01101110 01100100 00100000 01100010 01100101 01111001 01101111 01101110 01100100 00101110`
Decimal	`84 111 32 105 110 102 105 110 105 116 121 32 97 110 100 32 98 101 121 111 110 100 46`

Summing the values together gives 2159.

2159 % 256 = 111

Therefore the checksum will be 6F (hex) or 111 (decimal).

Check sums

2s complement checksum

This is the "word size" minus the checksum.

In this case, that would be 256 - 111 = 145 decimal or 91 hex.

("word size" in this sense being the length of each unit of data, which is 8 bits in this example and so $2^8 = 256$).

What would it take to create a checksum generator in Python? Not much!

```
~~~ Checksum calculator ~~~
Enter text:The sky calls to us - Carl Sagan
The checksum is 203 or 0xcb
>
```

ARQ (Automatic Repeat reQuest)

Sender receives an acknowledgment from the receiver implying that the frame or packet is received correctly before a timeout occurs.

Timeout is a specific time period within which the acknowledgment has to be sent by the receiver to the sender.

If the sender does not receive the acknowledgment before the specified time, it is implied that the frame or packet has been corrupt or lost during the transmission.

The sender retransmits the packet and these protocols ensure that this process is repeated until the correct packet is transmitted.

Review