The programming interface for Ethernet on R-Pi platforms. (All other platforms are somewhat similar.)

`mac, ip, dns, gateway, subnet`

Here is one Linux command to get network info. (or `ifconfig wlan0` for a Wi-Fi connection)
(or `ifconfig –a` to discover any connections)

```
ifconfig eth0

eth0 Link encap:Ethernet  HWaddr c7:35:ce:fd:8e:a1
     inet addr:192.168.0.16  Bcast:192.168.0.255  Mask:255.255.255.0
     inet6 addr: fe80::ba27:ebff:fefc:9fd2/64 Scope:Link
     UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
     RX packets:336 errors:0 dropped:0 overruns:0 frame:0
     TX packets:304 errors:0 dropped:0 overruns:0 carrier:0
     collisions:0 txqueuelen:1000
     RX bytes:27045 (26.4 KiB)
     TX bytes:43758 (42.7 KiB)
```

The `gateway` and `dns` are configured automatically when your R-pi signs onto your ISP's service.
These are stored in various configuration files.

"`ifconfig`" was demonstrated on an R-pi

1

Raspberry Pi's Ethernet Connection

Python provides two levels of access to network services.
At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers.

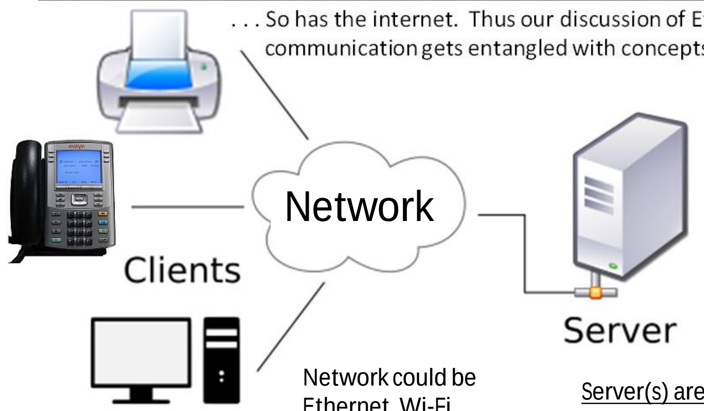
Python also has libraries that provide higher-level access to specific application-level network protocols, such as FTP, HTTP, and so on.

https://www.tutorialspoint.com/python/python_networking.htm

2

Ethernet has been designed around a *client-server model* of communications

. . . So has the internet. Thus our discussion of Ethernet as an example of synchronous communication gets entangled with concepts that relate to the Internet also.



This type of illustration is typical, but how does one picture a program?

Actually, one computer can run both client and server programs at the same time and/or several of each at the same time. It is important to see clients and servers as *programs*, not literal machines or hardware.

Clients are programs. (not literal hardware!)

Examples of clients:
Web browser, Opera
e-mail client, Outlook
VOIP phone, CsipSimple
Networked printer

Network could be Ethernet, Wi-Fi, Sonet, ATM, etc. These could be interconnected to the Internet or could be stand-alone.

Server(s) are programs. (not literal hardware!)

Examples of servers:
Web server, Apache
VIOP (phone) server, Asterisk
Print server, IPP print spooler.

Clients and servers connect to each other via sockets
A socket address is an IP address combined with a port address

https://en.wikipedia.org/wiki/File:1140E.jpg
https://commons.wikimedia.org/wiki/File:Client-server-model.svg

3

So you want to do a Raspberry Pi project that uses the Ethernet (& possibly the Internet). . .

Your project will be a server. . .
if it is a resource for other programs to use

Web cam

Temperature and humidity sensor

Private file server and file backup storage

Media server

. . . or your project will be a client.
if it is drawing on external resources

Weather station display—from accessible data

Media player

NIST time display

Or it could be both.

A remote-controlled (client) robot with camera, temperature sensors (severs) on board.

4

How a server program works

1.) When the OS starts it sets up the IP stack. This is an object that can be called to mange the network traffic on this computer.
Typically the IP stack process is interrupt driven and important parts of it are always in memory for fast responses.

2.) When a server program (say Apache) starts, it needs to establish connections to the internet. These are called "sockets."
`serverSoc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)` # a generic socket is put in memory.
AF_INET says this will be an IPv4 socket. SOCK.STREAM says that this socket will use transmission control protocol (TCP).
So far, the socket is only in memory, only visible to the server program.

3.) The server program needs to make the IP stack bind the new socket to a port visible on the network (give the socket an address).
`serverSoc.bind((socket.gethostname(), 80)` # Socket connects to host, typically a router, port 80.
A port number is to an IP address as an apartment number is to a street address.
Port numbers direct traffic to the correct application program within the computer.
Traffic sent to this computer's IP address and further addressed to port 80 will land in this socket in this program (Apache).

4.) Make the socket listen to the Internet
`serverSoc.listen(5)`
The "5" requests the socket to buffer up to five connection requests (requests for client sockets) before claiming to be busy.
The OS will now create a software interrupt when traffic lands in this server socket.
The ISR of the OS will put arriving traffic in `serverSoc`
If nothing more is done, the buffer of 5 connection requests will soon fill and further traffic will be rejected (error sent back).

5.) Typically the server program will loop endlessly. It will use interrupt driven I/O or polling to deal with requests.
A server socket is used as a sort of telephone operator. It listens for client connection requests and processes them.

```
while True:
        (clientsocket, address) = serversocket.accept() # accept req. for client soc
        ct = client_thread(clientsocket) # create a thread to make a client socket
        ct.run()# run the thread
```

. . .continues on the next slide

5

How a server program works

6.) All the real work of the server program is done through the client sockets. Each client socket represents a connection to a program running (typically) on another machine somewhere else in the world. The client program and the server program are in a sense melded together through the client socket(s). Typically a server program has only one server socket and many client sockets. Or, if the server supports multiple communication standards, then it creates one server socket for each communication standard and puts each server socket on a separate port address. Client sockets usually get port numbers greater than 49151.

7.) There are several protocols that the server program should observe to promote efficient computing. For example if a server program is going to shut down it should first send appropriate messages to all client sockets to let them know what is about to happen and respond appropriately.

8.) The server program also must be set up to deal sensibly with client sockets that get filled with garbage or become unresponsive. IP protocol is generally very forgiving (to accommodate large network delays) and will typically not take action in the time frame desired. The person who writes the server program will probably want to deal directly with sockets that are connected to recalcitrant or malicious clients.

The client-server roles were discussed in the context of web browsing.
Protocols such as `http` and languages such as `html` were related to the client-server roles

https://docs.python.org/3/howto/sockets.html

6

How a client program works

1.) When the OS starts it sets up the IP stack. This is an object that can be called to mange the network traffic on this computer.
Typically the IP stack process is interrupt driven and important parts of it are always in memory for fast responses.

2.) When a client program (say Chrome Browser) starts, it needs to establish a connection to the server program.
The connection is called a "client socket."
`with socket.socket(socket.AF_INET, socket.SOCK_STREAM)as clientSoc` # a generic socket is made
AF_INET says this will be an IPv4 socket. SOCK.STREAM says that this socket will use transmission control protocol (TCP).
So far, the socket is only in memory visible to the server program.

3.) The client program needs to make the IP stack bind the new socket to a port (give the socket an address) and connect to the server.
`clientSoc.connect('64.68.90.4', 80)` # Socket connects to host via router(s), maybe this is google.com, port 80.
Traffic sent out this port will land in the port-matching socket in the server program (Apache).

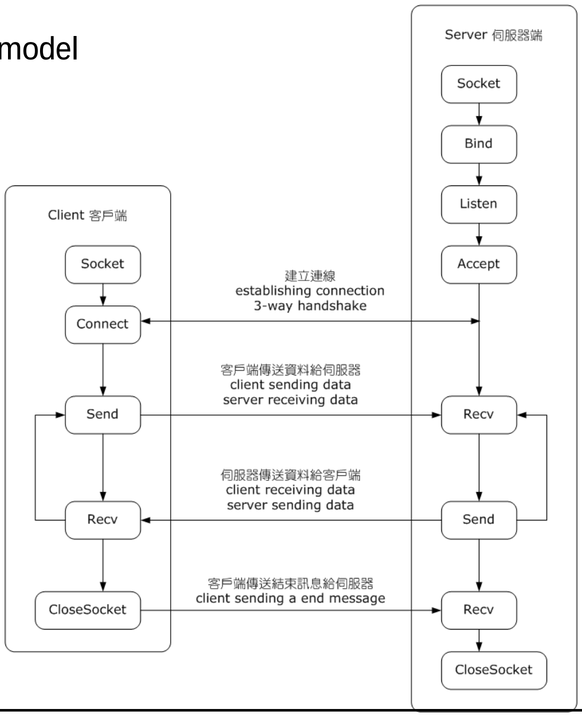
4.) If the server responds, the returned message will change the port address of `clientSoc` on the client computer to match that of the `clientSoc` on the server. This new port number will typically be something greater than 49151.
Port addresses between 0 and 49151 (inclusive) are standardized. Each represents a protocol. (Address 80 means HTTP)
Every new client port address that is granted will use the same protocol as the connection request used. (in this example, HTTP)
https://en.wikipedia.org/wiki/Port_(computer_networking)#Common_port_numbers
Now the server program and the client program are melded together via the symmetric `clientSoc` in each program.

5.) Yes, it gets more complicated. Lots more complicated. Take a network programming course!

https://realpython.com/python-sockets/

7

TCP Flow diagram for the client-server model



END OF ETHERNET DISCUSSION

https://commons.wikimedia.org/wiki/File:InternetSocketBasicDiagram_zhtw.png

8