Socket programming with Python

Agenda

- Understanding sockets
  - Types of sockets
  - Local and remote addresses
  - Ephemeral ports
- Creating an echo client-server
  - Single connection
  - Multi connection
  - Multi-threaded

Key terms

Socket

> A network socket is a software structure within a network node of a computer network that serves as an endpoint for sending and receiving data across the network

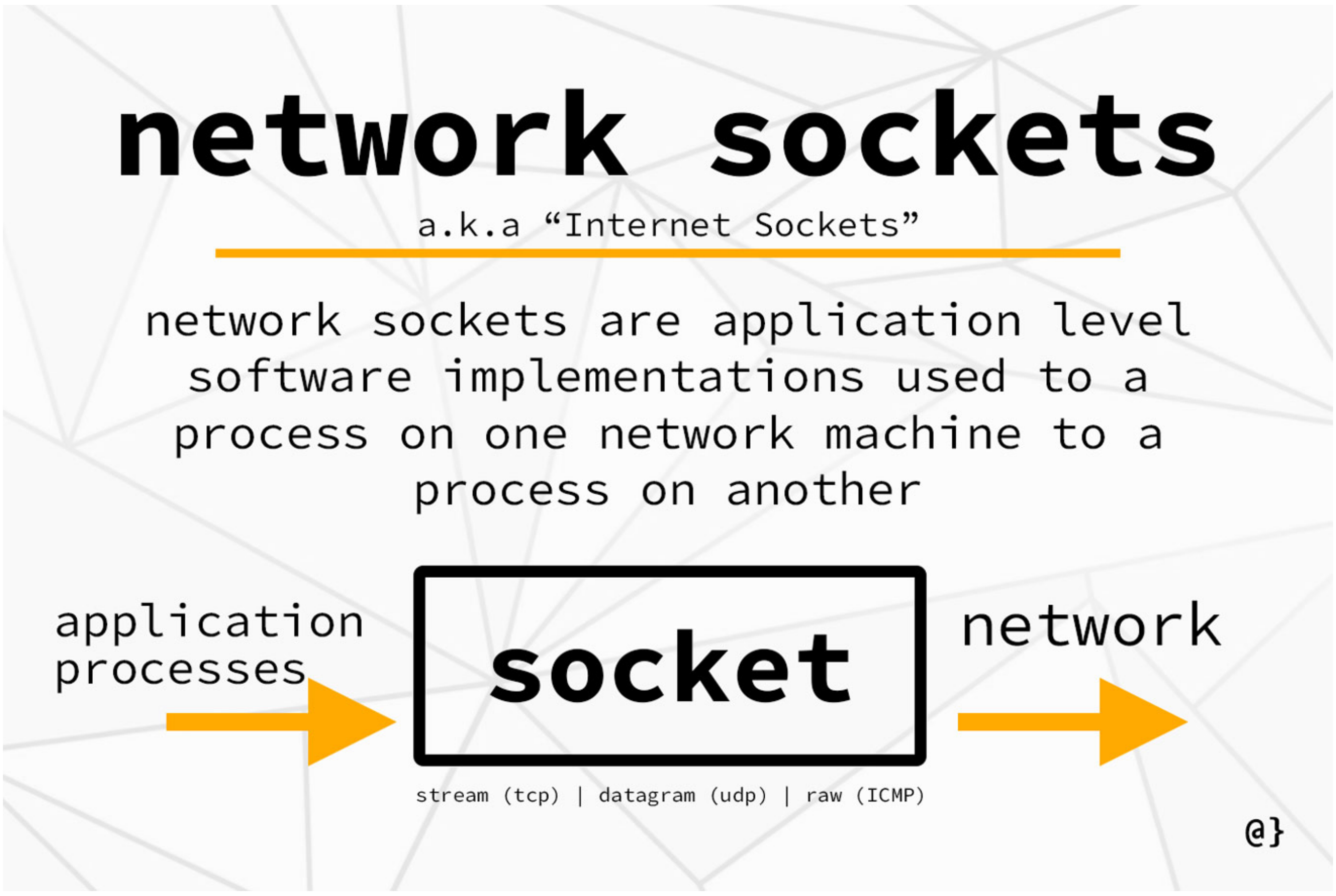
Ephemeral port

> A port that is dynamically allocated by the system and is used for a short period of time

What are sockets?

> Sockets and the socket API are used to send messages across a network. They provide a form of inter-process communication (IPC).

Sockets are nothing but interfaces provided to developers transfer data over the network. Any message sent from one process to another, must go through the underlying layers. The software interface between these layers is called a **socket**. Socket programming is a way of connecting two nodes on a network to communicate with each other.


network sockets
a.k.a "Internet Sockets"
network sockets are application level software implementations used to a process on one network machine to a process on another
application processes
socket
network
stream (tcp) | datagram (udp) | raw (ICMP)
@}

One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.

> As developers, we have control over everything that happens before the message is sent through the socket, and everything that happens after the message is received from a socket. The only control we have over the transport layer is maybe choosing the protocol (TCP, UDP...) or some parameters.

> A socket is an endpoint instance defined by an IP address and a port in the context of either a particular connection or the listening state

A socket comprises 5 things:

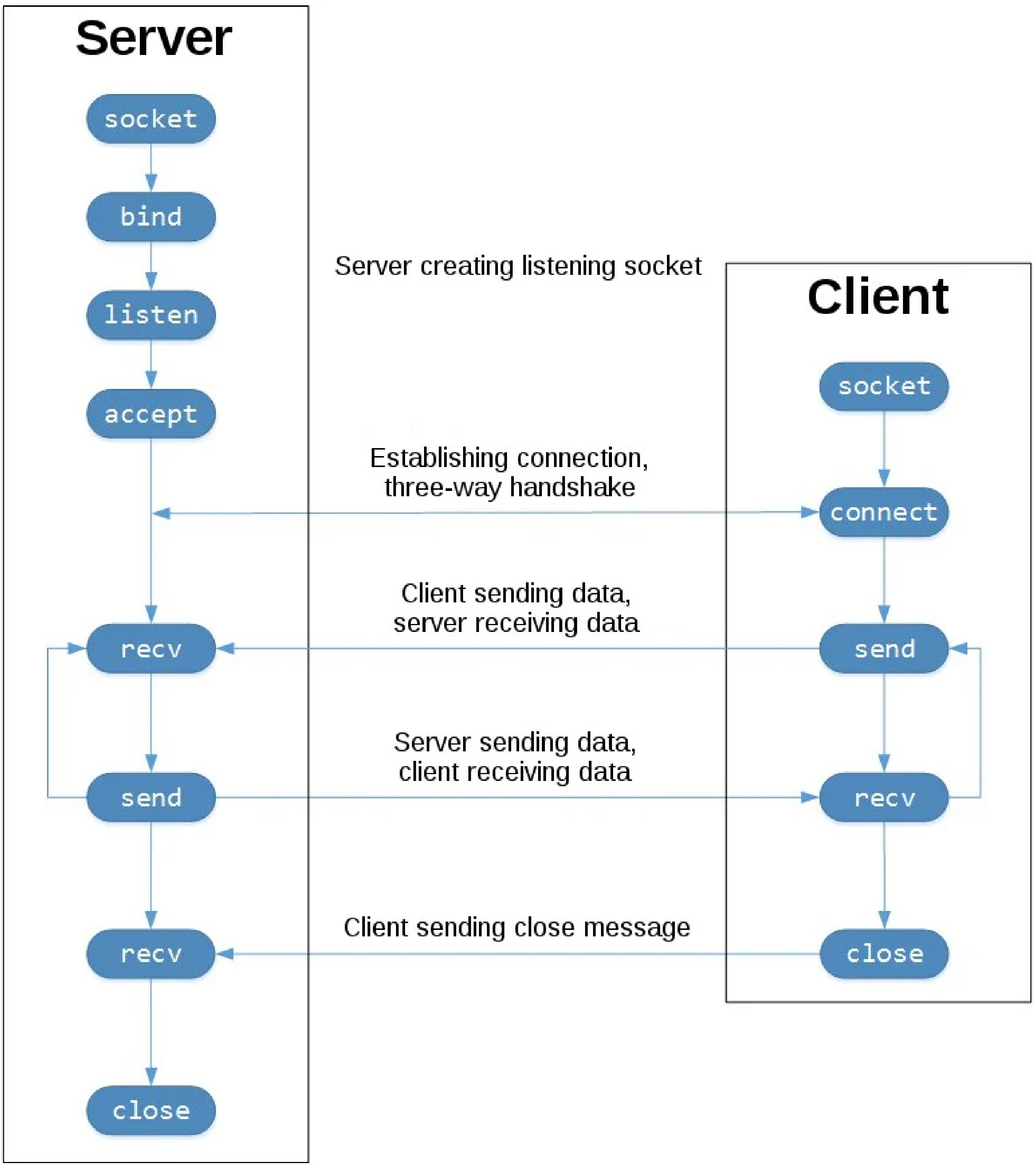
1. Local address
2. Local port
3. Remote address
4. Remote port
5. Protocol

Types of sockets

1. `Stream socket`: a socket that is used to send and receive data using TCP in a connection-oriented manner.
2. `Datagram socket`: Datagram sockets are used to support User Datagram Protocol (UDP) applications that rely on connectionless data transfer. Each packet sent via datagram sockets is individually addressed and routed but takes no measure to ensure order or arrival. Datagram sockets are considered to be "unreliable" transport services.

3. `Raw socket`: Raw sockets allow the send/receive of Internet Protocol packets from the network layer without any specific constraint on protocol (TCP, UDP, etc.). As such, header specifications are made at the application layer when sending and much of the encapsulation is left up to application developers.

Life cycle of a socket


Server
socket
bind
listen
accept
recv
send
recv
close
Server creating listening socket
Establishing connection, three-way handshake
Client sending data, server receiving data
Server sending data, client receiving data
Client sending close message
Client
socket
connect
send
recv
close

Server

1. Create a socket
2. Bind the socket to a local address and port

3. Start listening for incoming connections
4. Accept incoming connections
5. Receive data
6. Send data
7. Close the socket when done

Client

1. Create a socket
2. Connect to a remote address and port
3. Send data
4. Receive data
5. Close the socket when done

Ephemeral ports

The server port is one that cannot be instantly changed since it is being used by multiple clients. However, the client port is one whose value is not of importance. As long as the client port is present, the connection and data transmission will be successful. So to avoid the overhead of binding a socket to the port, the system will automatically allocate a port for the client. This is a short-lived port and is hence known as an ephemeral port.

Socket programming with Python

Creating a socket

```
import socket

socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Here, we create a socket object with the address family `AF_INET` and the socket type `SOCK_STREAM`. This means that the socket will use the IPv4 protocol and will be a TCP socket.

If we want to use the UDP protocol, we would use the socket type `SOCK_DGRAM`.

Binding a socket to a local address and port

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('127.0.0.1', 5000))
```

Listening for incoming connections

```
sock.listen(1)
```

The parameter `1` specifies the maximum number of queued connections. It specifies the number of unaccepted connections that the system will allow before refusing new connections. Starting in Python 3.5, it's optional. If not specified, a default backlog value is chosen

Find out more about the parameter here.

Accepting incoming connections

```
conn, addr = sock.accept()
```

The `accept` method blocks execution and waits for an incoming connection. When a client connects, it returns a new socket object representing the connection and a tuple holding the address of the client

The `conn` variable is the new connection (socket) object and the `addr` variable is the address of the client. We can use the `conn` variable to send and receive data. The difference between the `conn` and `sock` variables is that `sock` is the listening socket and `conn` is the new connection that has both local and remote addresses.

Connecting to a remote address and port

```
sock.connect(('127.0.0.1', 5000))
```

The connect method is used by the client to connect to the server.

Receiving and sending data

```
data = conn.recv(1024)
```

The `recv` method is used to receive data from the socket. The parameter specifies the maximum number of bytes to be received.

```
conn.sendall(b'Hello World')
```

Unlike `send`, this method continues to send data from bytes until either all data has been sent or an error occurs. None is returned on success.

Closing the socket

```
sock.close()
```

Further reading

- Detailed Socket programming with Python
- Creating a port scanner with Python
- Non-blocking sockets