Programming and Communications III: Sockets

Jordi Ricard Onrubia Palacios

Departament d'Informàtica i Enginyeria Industrial Universitat de Lleida

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

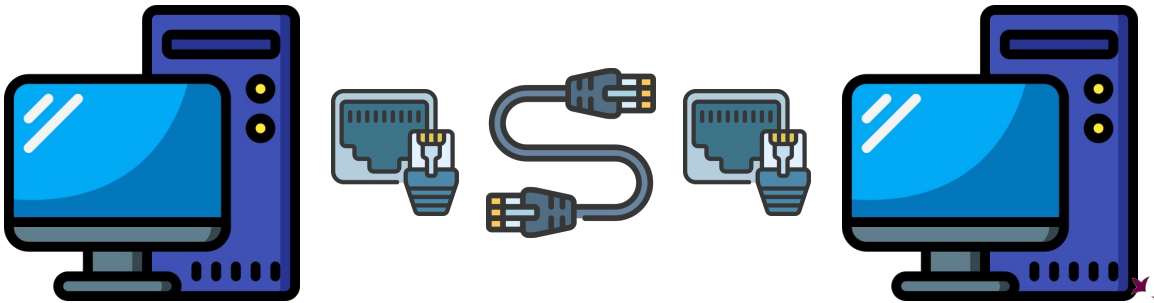
Programming and Communications III: Sockets

- Overview
  - Types
  - Architectures
- Python Libraries
  - Standard Library
  - ZeroMQ

Overview

Sockets are communication mechanisms, making possible transfer information between different processes even in different machines.

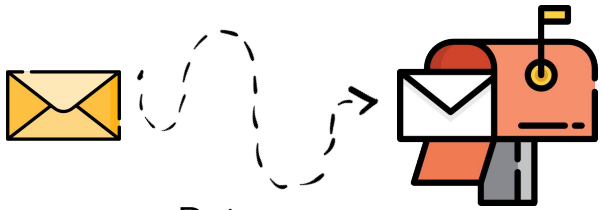
Sockets are assigned to a port in a machine, so they can be identified and accessed by the TCP layer. (A port is an interface from where data can be sent and received)



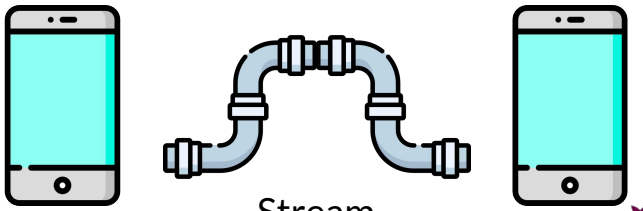
Types of Sockets

Datagram (UDP): No-connection oriented socket, your data is sent to another socket, like sending letters by mail.

Stream (TCP): Connection oriented socket, sequenced and a unique flow of data with mechanisms for creating and destroying connections and detecting errors. A connection is established between two sockets and data is transferred between them. Like calling by phone, a connection is established and a conversation is done until finished.



Datagram



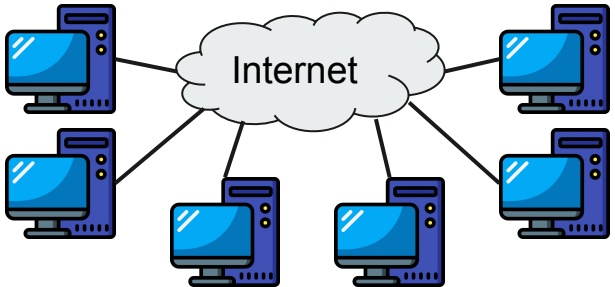
Stream

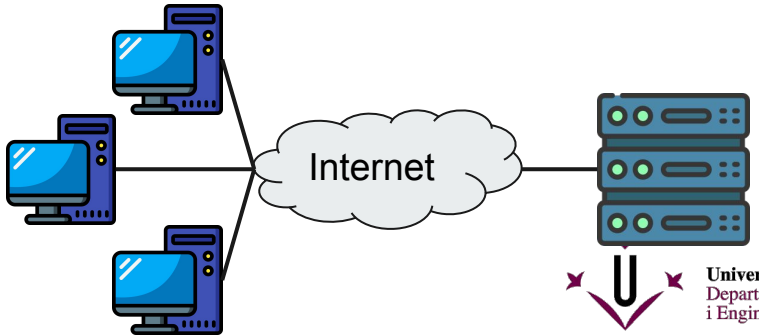
Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Common Architectures

Peer to peer (p2p): A peer-to-peer (P2P) network is created when two or more PCs are connected and share resources without going through a separate server computer. The connection can be an ad hoc connection, a number of computers connected on the same bus, such an office or a network build with special protocols.

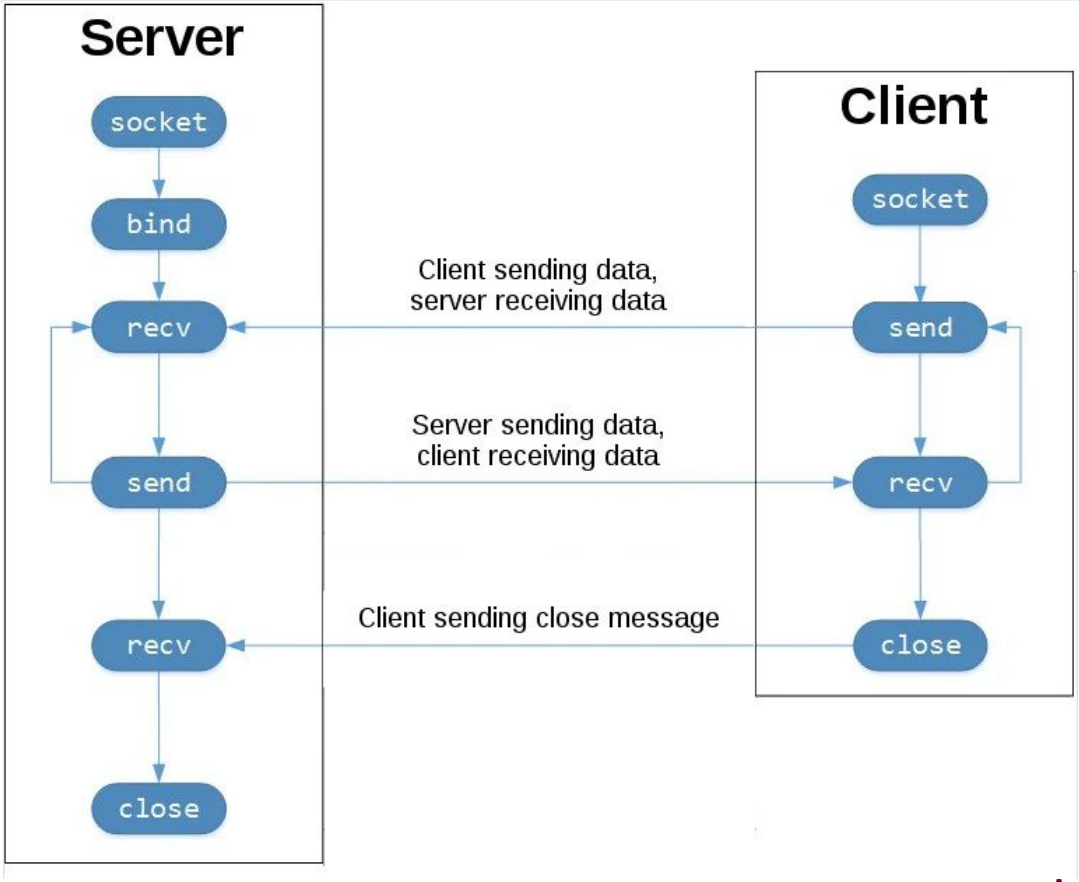
Client-Server: A centralized structure composed by a provider, the server, and many requesters, the clients. The clients will send request to the server without interacting with the other clients and the server will provide response to the request



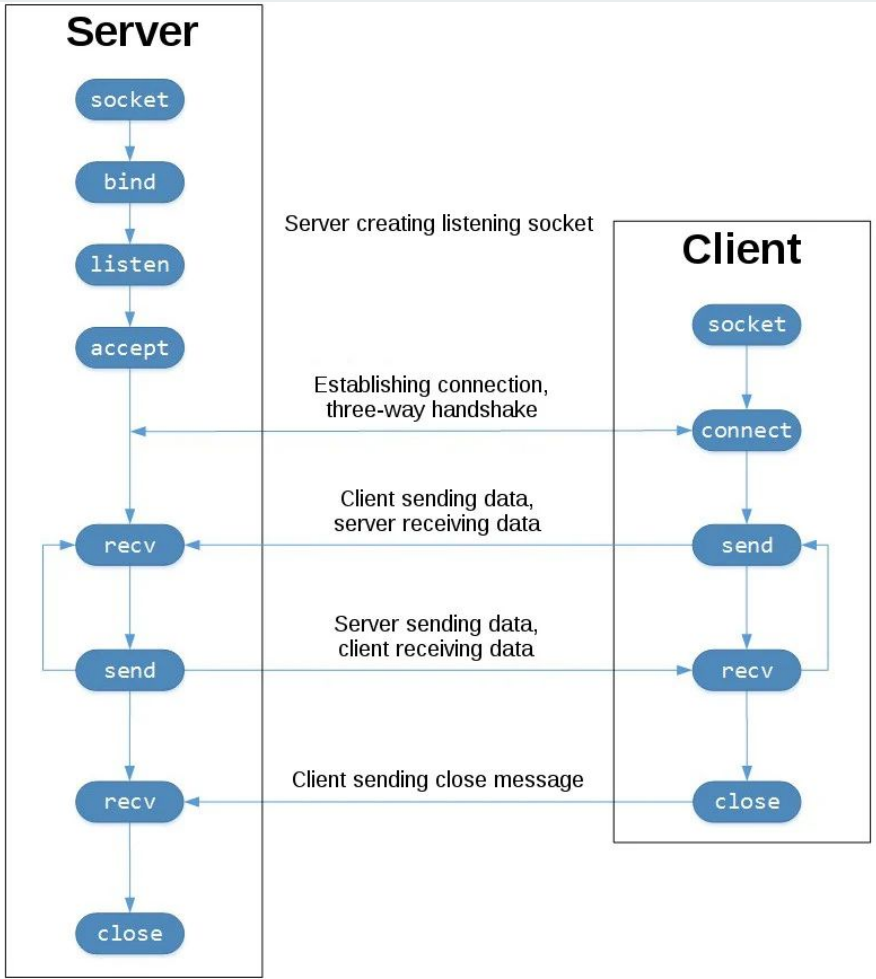


Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Common Architectures: Client-Server UDP



Common Architectures: Client-Server TCP



Universitat de Lleida
Departament d'Informàtica i Enginyeria Industrial

Programming and Communications III: Sockets

- Overview
  - Types
  - Architectures
- Python Libraries
  - Standard Library
  - ZeroMQ

Python Libraries: Standard Library

Socket Creation:

To create a socket we need to sent to the constructor the family type of the socket, in our case we will use always the socket.AF_INET, and the second parameter the type of the socket, as we have seen before we have the datagram socket, referred in python as socket.SOCK_DGRAM and the streaming socket, referred as socket.SOCK_STREAM.

Socket Creation

```
# Datagram socket UDP Socket
socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Streaming socket TCP Socket
socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Python Libraries: Standard Library

Socket Server Preparation:

Binding:

If we are using UDP sockets we will need to bind one socket to a port, this is done by the server part. After this step, the UDP socket is ready to receive messages. For TCP sockets we must add one step to be ready to communicate.

Listening:

TCP sockets require to listen for requests to establish a conversation. Once the TCP server hears a connection request, it will be in charge of accepting the request to begin to communicate.

Once accepted, we will receive the connection, that will be used for further communications and the address of the client socket.

Python Libraries: Standard Library

Socket Server Preparation

```
# Datagram socket UDP Socket, binding step
s.bind((HOST, PORT))

# Streaming socket TCP Socket, binding, listening #
and accept
s.bind((HOST, PORT))
s.listen()
conn, addr = s.accept()
```

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Python Libraries: Standard Library

Socket Client Preparation:

Connecting:

For UDP clients we do not need to do anything else, once the server is available to receive messages, the client only needs to send messages.

For TCP clients we need to send to the TCP server a connection request (we have seen previously that the TCP server needs to accept the client request to establish a connection). Once the connection request is accepted, the client will be able to send messages.

Socket TCP Client Preparation

```
# Streaming socket TCP Socket connection
s.connect((HOST, PORT))
```

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Python Libraries: Standard Library

Socket Data Sending:

We have two different ways of sending data, if we are using a UDP socket, we are sending a message to an specific location, we do not have any connection established, therefore, we need to specify the receiver. When we are working with TCP sockets, we are establishing a connection, this connection is returned at the time of accepting the request from the client. Therefore we can use that connection to return a message. Otherwise, we can use the address also returned by the accept to send a message.

Socket Message Sending

```
# Sending a message with the socket
s.sendto(data, (HOST, PORT))
# Sending a message with the connection returned # by
the accept
conn.sendall(data)
```

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Python Libraries: Standard Library

Socket Data Receiving:

The same as before, we have different ways of receiving data, if we do not have a connection as in the UDP we can use the socket.recvfrom, this will return the data and the address from where this data is coming from, very useful for when we do not have a connection oriented. When we have a connection between the client and the server, we can use the socket.recv, this will only return the data. Note that both methods expect the number of bytes to be retrieved from the message. If we pass less bytes than the size of the data received, this data will be cut.

Socket Message Receiving

```
# Recvfrom returns a touple (data, address)
data = s.recvfrom(1024)
# Recv returns only the data
data = s.recv(1024)
```

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Python Libraries: Standard Library

Attention:

Check that every time we are communicating, we are doing it directly from the client to the client, therefore, if the client or the server losses the communication, the process of sending the message will throw an error. Therefore, the python standard socket library requires a persistent connection.

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

Python Libraries: ZeroMQ

ZeroMQ is a library used to implement messaging and communication systems between applications and processes, supports certain network communication patterns using sockets. This library has a different way of working thanks to the asynchronous advantages given by the message queues.

Furthermore, ZeroMQ supports common messaging patterns (pub/sub, request/reply, client/server and others) over a variety of transports (TCP, in-process, inter-process, multicast, WebSocket and more).

All of this in short and clean pieces of code.

Python Libraries: ZeroMQ

Messaging Patterns: Request - Reply

Connects a set of clients to a set of services. Is intended for service-oriented architectures of various kinds. It comes in two basic flavors: synchronous (REQ and REP socket types), and asynchronous socket types (DEALER and ROUTER socket types)

- A REQ socket is used by a client to send requests to and receive replies from a service.
- A REP socket is used by a service to receive requests from and send replies to a client.
- DEALER and ROUTER work as an asynchronous replacement for REQ and REP respectively.

Python Libraries: ZeroMQ

Repliyer

```
import zmq

context = zmq.Context()

sock = context.socket(zmq.REP)
sock.bind("tcp://127.0.0.1:5678")

while True:
    message = sock.recv()
    sock.send_string("Echo: General Kenobi")
    print("Echo: {}".format(message))
    if message:
        break
```

Requester

```
import zmq

context = zmq.Context()

sock = context.socket(zmq.REQ)
sock.connect("tcp://127.0.0.1:5678")

sock.send_string("Hello There")
print(sock.recv())
```

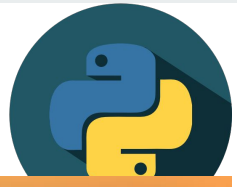
Python Libraries: ZeroMQ

Messaging Patterns: Publish - Subscribe

Connects a set of publishers to a set of subscribers. Used for one-to-many distribution of data from a single publisher to multiple subscribers in a fan out fashion.

- Topics: ZeroMQ use topics to categorize the messages. This way, subscribers can subscribe for specific categories in which they are interested in.
- A PUB socket is used by a publisher to distribute data to an specific topic.
- A SUB socket is used by a subscriber to subscribe to data distributed by a publisher.

Python Libraries: ZeroMQ



Subscriber

```python
import zmq

context = zmq.Context()

sock = context.socket(zmq.SUB)

sock.setsockopt_string(zmq.SUBSCRIBE, "3")
sock.connect("tcp://127.0.0.1:5680")

while True:
    message = sock.recv()
    print(message)
```

Publisher

```python
import zmq
import time

context = zmq.Context()

sock = context.socket(zmq.PUB)
sock.bind("tcp://127.0.0.1:5680")

id = 0

while True:
    time.sleep(1)
    id, now = id+1, time.ctime()

    message = "1-Update! >> #{id} >> {time}".format(id=id, time=now)
    sock.send_string(message)

    message = "2-Update! >> #{id} >> {time}".format(id=id, time=now)
    sock.send_string(message)

    id +=1
```

Python Libraries: ZeroMQ

Messaging Patterns: Pipeline

Intended for task distribution. Is intended for task distribution, typically in a multi-stage pipeline where one or a few nodes push work to many workers, and they in turn push results to one or a few collectors.

- The PUSH socket type talks to a set of anonymous PULL peers, sending messages using a round-robin algorithm.
- The PULL socket type talks to a set of anonymous PUSH peers, receiving messages using a fair-queuing algorithm.

Python Libraries: ZeroMQ

Worker

```
import zmq

context = zmq.Context()

sock = context.socket(zmq.PULL)
sock.connect("tcp://127.0.0.1:5690")

while True:
    message = sock.recv()
    print("Received: {}".format(message))
```

Lazy Worker

```
def get_work_time():
    mu = 2
    sigma = 1
    return math.fabs(np.random.normal(mu, sigma, 1))

context = zmq.Context()

sock = context.socket(zmq.PULL)
sock.connect("tcp://127.0.0.1:5690")

while True:
    work_time = get_work_time()
    print("Working during: {}".format(work_time))
    time.sleep(work_time)
    message = sock.recv()
    print("Received: {}".format(message))
```

Manager

```
context = zmq.Context()

sock = context.socket(zmq.PUSH)
sock.bind("tcp://127.0.0.1:5690")

id = 0

while True:
    time.sleep(1)
    id, now = id+1, time.ctime()

    message = "{id} - {time}".format(id=id, time=now)

    sock.send_string(message)

    print("Sent: {msg}".format(msg=message))
```

Python Libraries: ZeroMQ

As seen in the examples it always follows a pattern.

1. We initialize a context.
2. Define the type of the socket we want to initialize.
3. Operate with the proper methods.

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial

More info at:

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

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

https://zeromq.org/languages/python/

https://zeromq.org/socket-api/#request-reply-pattern

https://zeromq.org/socket-api/#publish-subscribe-pattern

https://zeromq.org/socket-api/#pipeline-pattern

Universitat de Lleida
Departament d'Informàtica
i Enginyeria Industrial