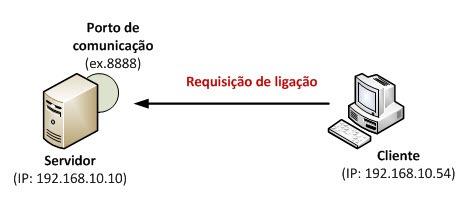
**O que é um Socket?** É um mecanismo de comunicação, normalmente utilizado em conjunto com o protocolo TCP/IP (Protocolo de Controle de transmissão) e IP (Protocolo de internet)

Amplamente utilizado em arquiteturas de comunicação, **cliente / servidor.**

Fonte da imagem: <https://pplware.sapo.pt/tutoriais/networking/redes-sabe-o-que-sao-sockets-de-comunicacao-parte-i/>



Basicamente para funcionar a comunicação entre cliente / servidor, precisamos do endereço IP de uma das máquinas para criarmos uma instância do socket (SERVIDOR), desse modo estamos criando uma comunicação unidirecional, ou seja, só podemos enxergar a nossas mensagens, esta é um abordagem obsoleta nos dias atuais, a arquitetura utilizada em nosso projeto é baseada nessa abordagem, descrita na imagem porém o servidor pode receber múltiplas instancias (conexões) e funciona em um modo de “espelhamento”, ou seja ele recebe os dados e replica para todas as instancias(clientes) conectados, como no diagrama abaixo:

Uma imagem contendo equipamentos eletrônicos

Descrição gerada automaticamente

**Código utilizado no Servidor:**

from socket import \*

import threading

HOST = ''

PORT = 5000

s = socket(AF\_INET, SOCK\_STREAM)

s.bind((HOST, PORT))

s.listen(10)

listaClient = set()

def envia\_cliente(client\_conn):

    while True:

        data = client\_conn.recv(1024)

        if(not data):

            print(client\_conn.getpeername(), 'saiu')

            return

        print(data)

        for c in listaClient:

            c.send(data)

print("AGUARDANDO CONEXÃO")

while True:

    conn, addr = s.accept()

    print('Cliente Conectado', addr)

    listaClient.add(conn)

    threading.Thread(target=envia\_cliente, args=(conn,)).start()

while True:

    data = conn.recv(1024)

    print("Recebido -  ", repr(data))

    conn.sendall(data)

conn.close()

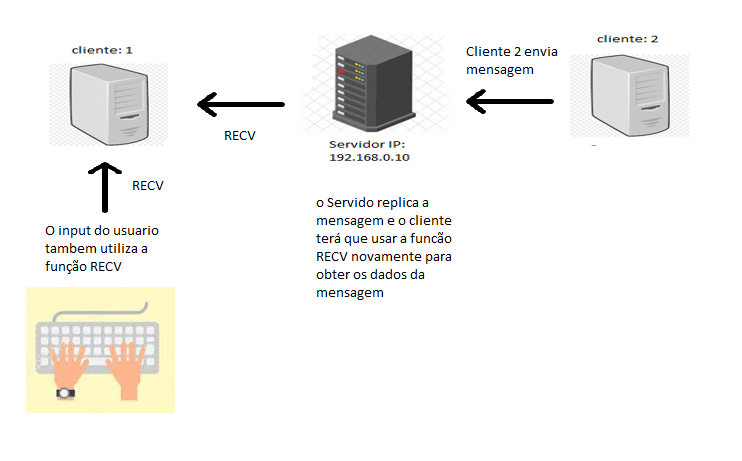
input()

O código acima é responsável por criar um servidor, essa é a parte mais importante de toda a comunicação, porque o servidor é responsável por transferir os dados para todos os sockets conectados(clientes), isso só é possível porque temos rotinas que ficam rodando em background (threads), as threads ficam rodando a todo momento devido a estar dentro um **while true**, ou seja, ficará em um loop infinito chamando a função envia\_cliente a todo tempo de execução, que fica esperando para enviar os dados para todos os sockets conectados(clientes), essa função funciona como um gatilho no momento em que chega os dados, logo é disparado a função envia\_cliente.

A arquitetura utilizada na parte do servidor, é amplamente utilizada, pois funciona em modo assíncrono, o modo assíncrono basicamente é um funcionamento independente de eventos, sendo assim as funções RECV e SEND do python não bloqueiam a aplicação, esperando retorno, portando essa arquitetura nos permite conectar até 10 Sockets(clientes) e conversarem entre si, assim como em grupo de mensagem do whatsapp, telegrama etc..

**Socket (Cliente)**

Essa foi a parte mais complexa da comunicação, pois o cliente precisa utilizar duas vezes a função RECV do python, isso é um grande problema, enquanto a função RECV não recebe dados ela continua executando bloqueando todo o fluxo do programa. segue abaixo o diagrama abaixo



Para conseguirmos realizar a comunicação entre clientes, tivemos que usar uma arquitetura semelhante à utilizada no servidor, tivemos que separar em duas funções o envio e o recebimento de dados, e especificamente a parte de recebimento de dados que vem do servidor, tivemos que separar em uma Thread, sendo assim a todo momento ficamos consultando se há mensagem no socket, todo esse processo funciona em background (2º plano), sendo assim quando o usuário tentar enviar a mensagem a função RECV estará escutando normalmente.

**Código utilizado no Socket do lado do cliente:**

def chatConnect(self):

       PORT = 5000

        ADDR = (self.IPusuario.get(), PORT)

       client\_socket = socket.socket(AF\_INET, SOCK\_STREAM)

        client\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

        client\_socket.connect(ADDR)

        def recebe\_mensagem():

            while True:

                try:

                    msgCrypt = client\_socket.recv(2048).decode("utf8")                           wordsArray = msgCrypt.split(',')

                    final = ""

                    for n in wordsArray:

                        l = int(n)

                        l = l - 20

                        l = chr(l)

                        final = final + l

                    listaMensagem.insert(END, final)

                except OSError:

                    break

        def enviar\_mensagem(event=None):

            msg = inputMensagem.get()

            inputMensagem.delete(0, END)

            v = []

            for letra in msg:

                l = ord(letra) + 20

                v.append(l)

            msgCrypt = ','.join(map(str, v))

            client\_socket.send(bytes(msgCrypt, "utf8"))

        janela = tk.Tk()

        janela.title("Comunicação Criptografada")

        container\_mensagem = Frame(janela)

        listaMensagem = Listbox(container\_mensagem, height=15, width=50)

        listaMensagem.pack(side=LEFT, fill=BOTH)

        listaMensagem.pack()

        container\_mensagem.pack()

        inputMensagem = Entry(janela)

        inputMensagem.bind("<Return>", enviar\_mensagem)

        inputMensagem.pack()

        btnEnviar = Button(janela, text="Enviar", command=enviar\_mensagem)

        btnEnviar.pack()

        recebe\_mensagemThread = Thread(target=recebe\_mensagem)

        recebe\_mensagemThread.start()