Conceitos e Tecnologias para Dispositivos Conectados (C115)

Prof. Samuel Baraldi Mafra

Inatel
Instituto Nacional de Telecomunicações

Objetivo geral

O objetivo desse curso é apresentar conceitos e tecnologias relacionadas a "**softwarização**" de redes de comunicações e ao advento da Internet das Coisas.

Programa

- Sockets, Programação para Redes e APIs (Application Programming Interfaces);
- Web services, Hypervisores, Virtualização, Máquinas Virtuais;
- Virtualização de roteadores, comutadores, rede e suas funções (NFV);
- Conceitos e Tecnologias de Redes Definidas por Software (SDN);
- Conceitos e Tecnologias para Internet das Coisas (IoT);
- Estudo de caso.

Bibliografia

- William Stallings, Foundations of modern networking : SDN, NFV, QoE, IoT, and cloud, Pearson Education, ISBN 978-0-13-417539-3.
- Stevan Júnior, Sérgio Luiz, Internet das Coisas - IoT : fundamentos e aplicações em Arduino e NodeMCU ; São Paulo, SP : Editora Érica, 2018. , ISBN-978-85-365-2607-2
- Olivier Hersent, The internet of things : key applications and protocols, 344 p. : il. , ISBN: 978-1-11999-4350
- Mieso Denko, "Mobile Opportunistic Networks: Architectures, protocols and applications", Auerbach Publications, 2010.

Bibliografia

- TANENBAUM, Andrew S.; SOUZA, Vandenberg D. de. Redes de computadores. Revisão de Edgard Jamhour. 4.ed. Rio de Janeiro: Campus, 2003. 945 p., il. ISBN 85-352-1185-3.
- KUROSE, James F.; ROSS, Keith W. Redes de computadores e a internet: uma abordagem top-down. Tradução de Arlete Simille Marques; Revisão de Wagner Luiz Zucchi. 3. ed. São Paulo: Pearson Addison-Wesley, 2005. 634 p., il. ISBN da 3ª ed.: 85-88639-18-1.
- YUCE, Mehmet R.; KHAN, Jamil Y. (co-autor). Internet of things (IoT): systems and applications. Massachusetts: Editora Jenny Stanford, 2019. 349 p., il. ISBN 978-981-4800-29-7.
- WAHER, Peter. Learning internet of things: explore and learn about internet of things with the help of engaging and enlightening tutorials designed for raspberry Pi. Estados Unidos da América, EUA: Packt Publishing, 2015. 221 p., il. ISBN 978-1-78355-353-2
- SOLAIMAN, Basel (co-autor). Information fusion and analytics for Big Data and IoT. Boston, MA: Editora Artech House, 2016. 252 p., il. ISBN 978-1-63081-087-0

Avaliação

- A avaliação será feita através de exercícios, um estudo de caso e um projeto no final da disciplina.
- Poderão ser feitos em dupla ou individual;
- NP1=0.3***exerc1**+0.7***ec**
  NP2=0.3***exerc2**+0.7***projeto**
  onde,
  exerc1=Exercícios de aulas até entrega do estudo de caso.
  exerc2=Exercícios de aulas após o estudo de caso até entrega do projeto.
  ec=estudo de caso Mininet.
  projeto=Projeto de solução IoT.

Em caso de não entrega dos exercícios no prazo estipulado, o aluno poderá entregar o exercício no início da próxima aula da disciplina, entretanto os exercícios valerão no máximo 50% dos pontos da atividade.

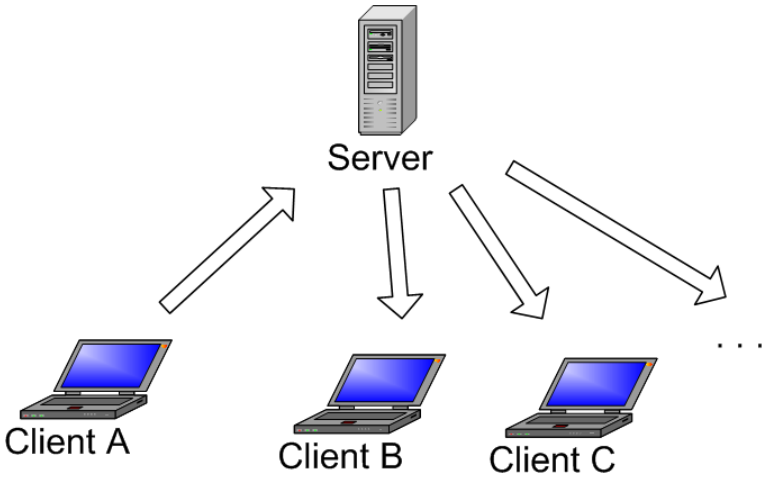
1 Apresentação da disciplina e introdução aos conceitos de sockets.
2 Exercícios envolvendo Sockets UDP e TCP
3 Web services, Hypervisores
4 Virtualização, Máquinas Virtuais
5 Introdução a virtualização de funções de rede (NFV)
6 Redes definidas por software: Introdução, definições e requisitos
7 Redes definidas por software: abstrações
8 Openflow: Introdução,tabela de fluxos,wildcrads, exemplos de regras.
9 Instalação do software Mininet, principais comandos e configurações de rede.
10 Análise de parâmetros de desempenho no software Mininet
11 Análise de regras no software Mininet
12 Construção de topologias customizadas no software Mininet
13 Introdução as redes de internet das coisas (IoT)
14 Arquitetura de redes IoT
15 Tecnologias para redes IoT: CoAP
16 Tecnologias para redes IoT: MQTT
17 Tecnologias para redes IoT: 802.15.4 e Zigbee
18 Tecnologias para redes IoT: 6LowPAN
19 Apresentações de projetos
20 Apresentações de projetos

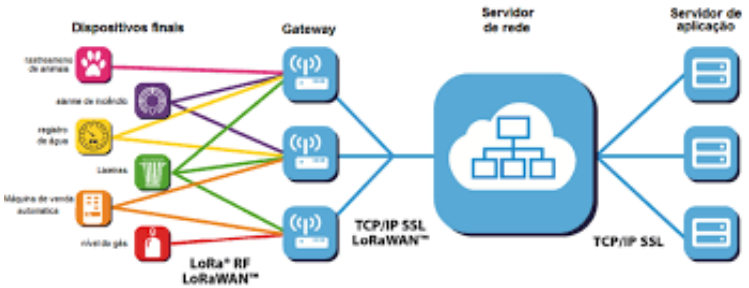
As aulas vão ocorrer nas terças feiras das 13:30 -15:10, a cada duas semanas.

Horário de atendimento:

Terça-Feira 17:30 - 19:30

Comunicação entre aplicações através de uma rede.





Revisão: Camada de transporte

- A camada de transporte fornece comunicação lógica entre processos de aplicações, sendo implementados nos sistemas finais.
- São estabelecidas conexões lógicas, fim-a-fim, entre cada par de aplicações que se comunicam.

Serviços providos pela camada de transporte

- A função básica da camada de transporte é promover uma transferência de dados fim-a-fim confiável (TCP), eficiente e econômica entre aplicações origem e destino, independente das camadas abaixo.
- São oferecidos dois tipos básicos de serviços:
  - O serviço orientado à conexão e confiável: protocolo TPC.
  - O serviço não orientado à conexão e não confiável: protocolo UDP.

UDP X TCP

UDP

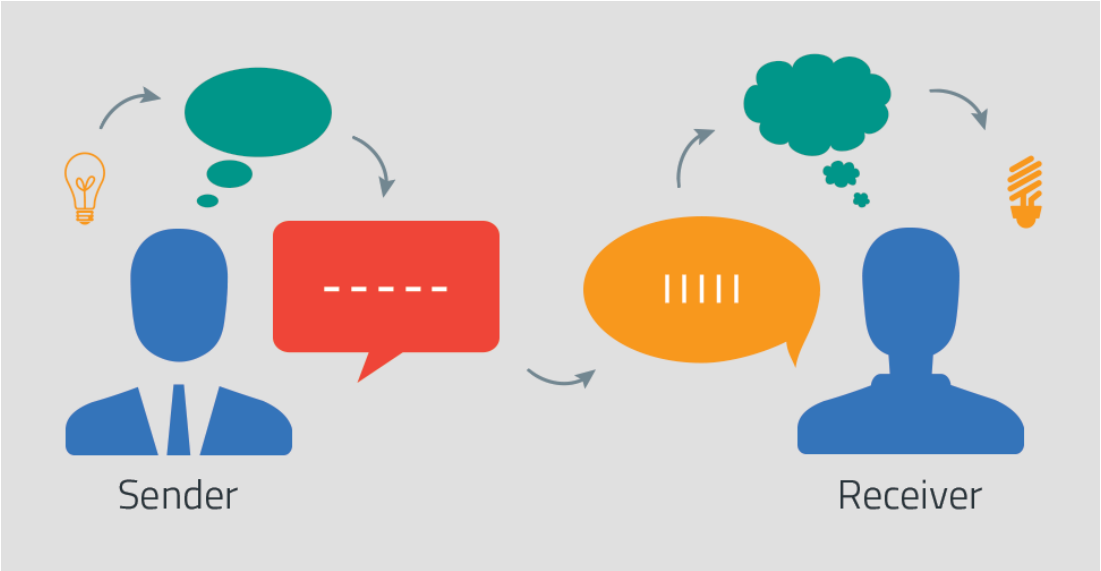
- Datagrama;
- Não necessita de conexão;
- Não possui controle de erro;
- Mais simples e rápido.

TCP

- Stream;
- Orientado a conexão;
- Controle de erro;
- Garante entrega.
- Entrega ordenada.

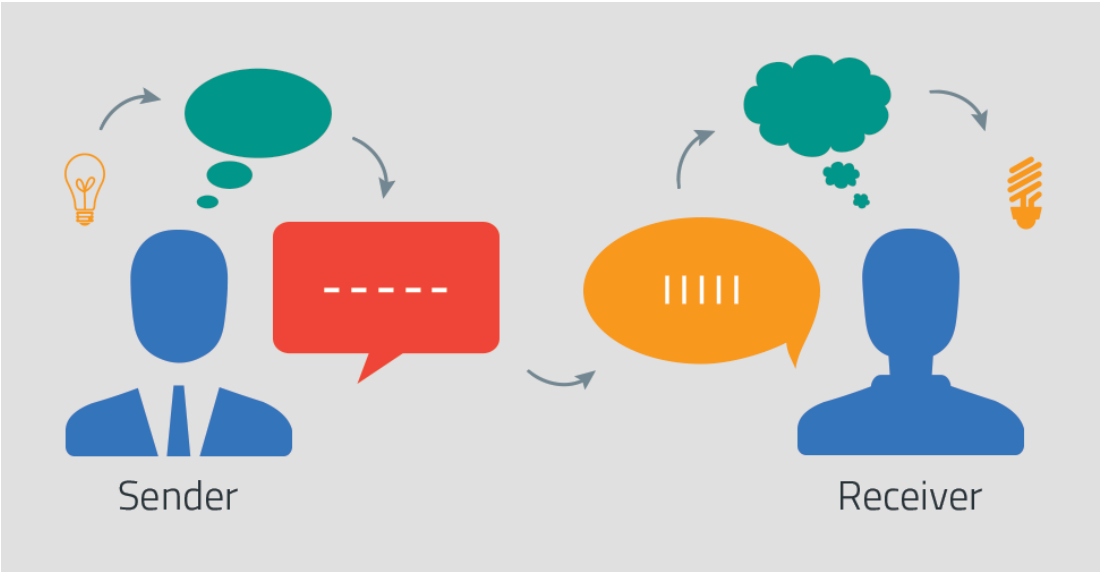
Sockets

- Como um programa se comunica com outro programa em uma rede?



Sockets

- Como um programa se comunica com outro programa em uma rede?



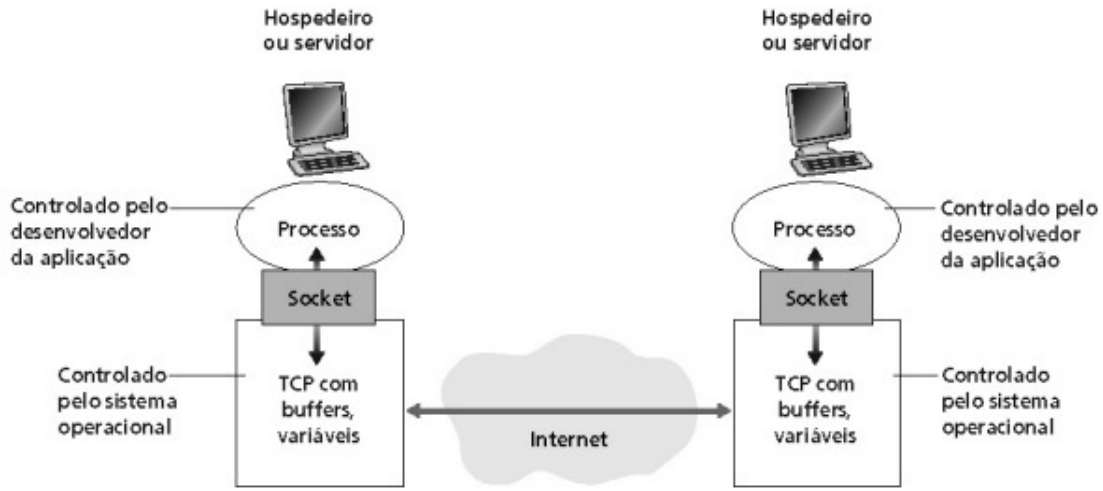
- Interface de programação de aplicativo (API) conhecida como sockets.

Interface De Programação De Aplicações

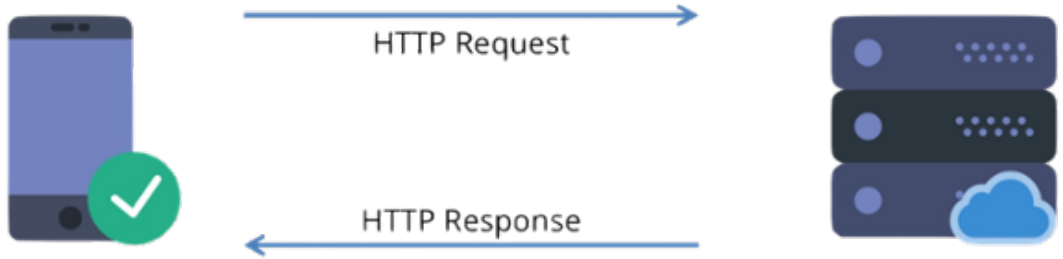
- API é um conjunto de definições e protocolos usados no desenvolvimento e na integração de softwares de aplicações.
- Uma API permite que uma solução se comunique com outros produtos e serviços sem precisar saber como eles foram implementados.
- Os APIs de sockets abstraem a camada de rede para que uma aplicação possa se comunicar com outra sem ter que se preocupar com detalhes da pilha TCP/IP que gere a rede abaixo dessa aplicação.

O que é um socket?

- Socket é um ponto final de um fluxo de comunicação entre processos através de uma rede de computadores.
- Socket é a interface entre a camada de aplicação e a de transporte dentro de uma máquina.
- Socket provê a comunicação entre dois processos que estejam na mesma máquina (Unix Socket) ou na rede (TCP/IP Sockets).



- Cliente - Servidor
- No padrão Request-Response há dois atores: Cliente e Servidor. Nesse protocolo o servidor fica o tempo todo ouvindo requisições que podem chegar dos seus cliente, porém ele estabelece uma conexão apenas temporária com o servidor, a medida que for realizando as requisições e obtendo as respostas, após isso a conexão é quebrada e o cliente não será notificado de nenhuma modificação.



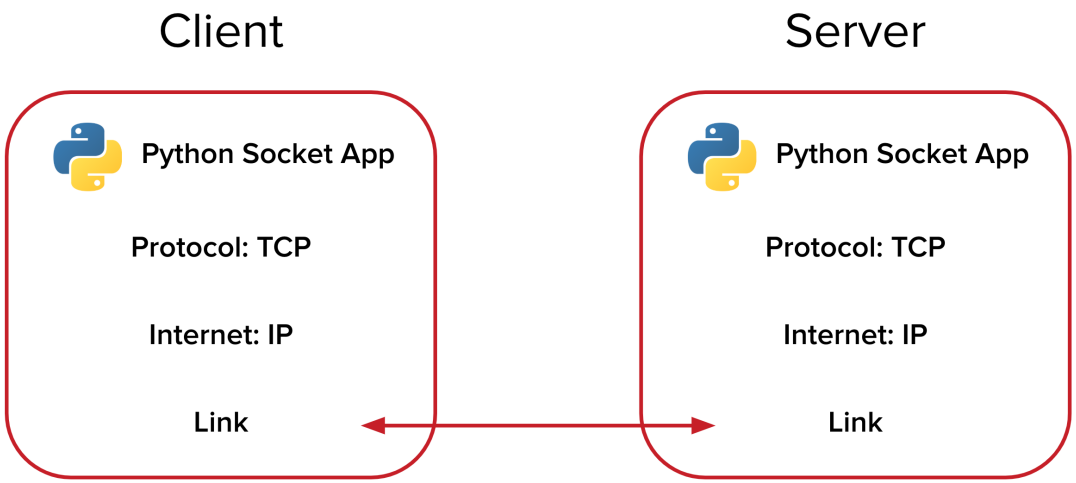
Sockets

- Um processo envia/recebe mensagens para/de seu socket;
- O socket é análogo a uma porta/ porteiro de um prédio:
  - O processo de envio empurra a mensagem para fora da porta;
  - O processo de envio confia na infra-estrutura de transporte no outro lado da porta que leva a mensagem para o socket no processo de recepção.
  - Um processo é identificado pelo endereço IP e porta.
  - Os sockets para Unix e internet são baseados na API Berkeley sockets. No windows há o Winsock.

Portas

- O uso de portas permite que computadores / dispositivos executem vários serviços / aplicativos.
- Analogia apartamentos
  - Todos os apartamentos compartilham o mesmo endereço.
  - No entanto, cada apartamento também possui um número de apartamento que corresponde ao número da porta.
- Portas até 1024 são reservadas.
- Exemplos de portas:
  - Porta 80- Servidor HTTP;
  - Porta 21- Servidor FTP;
  - Portas 25,587- Servidor Simple Mail Transfer Protocol (SMTP);
  - Porta 995- Porta SSL / TLS (POP3),
  - Porta 993 - Porta SSL / TLS (IMAP).

Programação de sockets em Python:



Linguagens disponíveis

- Java;
- C;
- Python.

Comandos

- Biblioteca Socket;
- socket(familia, tipo); Cria um socket de determinada família e tipo.
  Onde a familia é dada por:
  - AF_INET: IPV4;
  - AF_INET6: IPv6 e IPv4;
  - AF_UNIX, AF_LOCAL;
  - AF_BLUETOOTH.

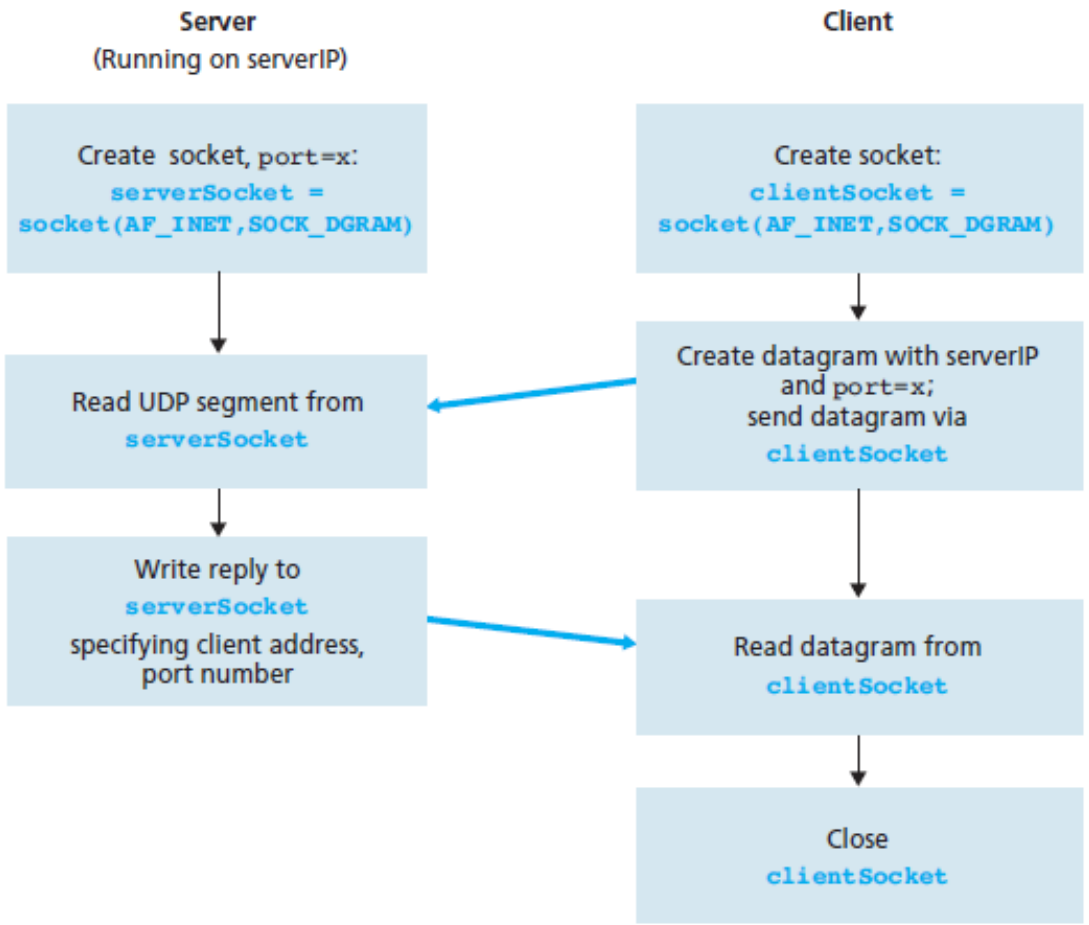
  Já o tipo:
  - SOCK_STREAM (TCP);
  - SOCK_DGRAM (UDP);

- bind((host,porta)): Associa o servidor a determinado endereço e porta.
- listen(): Este método configura e inicia o ouvinte TCP. Estabelece o numero de clientes na fila para se conectar ao servidor;
- accept(): Aceitar a conexão;

Comandos

- connect((host,porta)): Estabelece uma conexão com o servidor e inicie o handshake de três vias.
- connect_ex()
- send(): Enviar uma mensagem TCP;
- recv(): Receber uma mensagem TCP;
- sendto(): Enviar uma mensagem UDP;
- recvfrom(): Receber uma mensagem UDP;
- close(): Finalizar a conexão.

Socket UDP


Server
(Running on serverIP)
Client
Create socket, port=x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)
Create socket:
clientSocket =
socket(AF_INET,SOCK_DGRAM)
Create datagram with serverIP
and port=x;
send datagram via
clientSocket
Read UDP segment from
serverSocket
Write reply to
serverSocket
specifying client address,
port number
Read datagram from
clientSocket
Close
clientSocket

UDP-server

Programa servidor que recebe uma frase de tamanho 1024 bytes e coloca em letras maiúsculas.

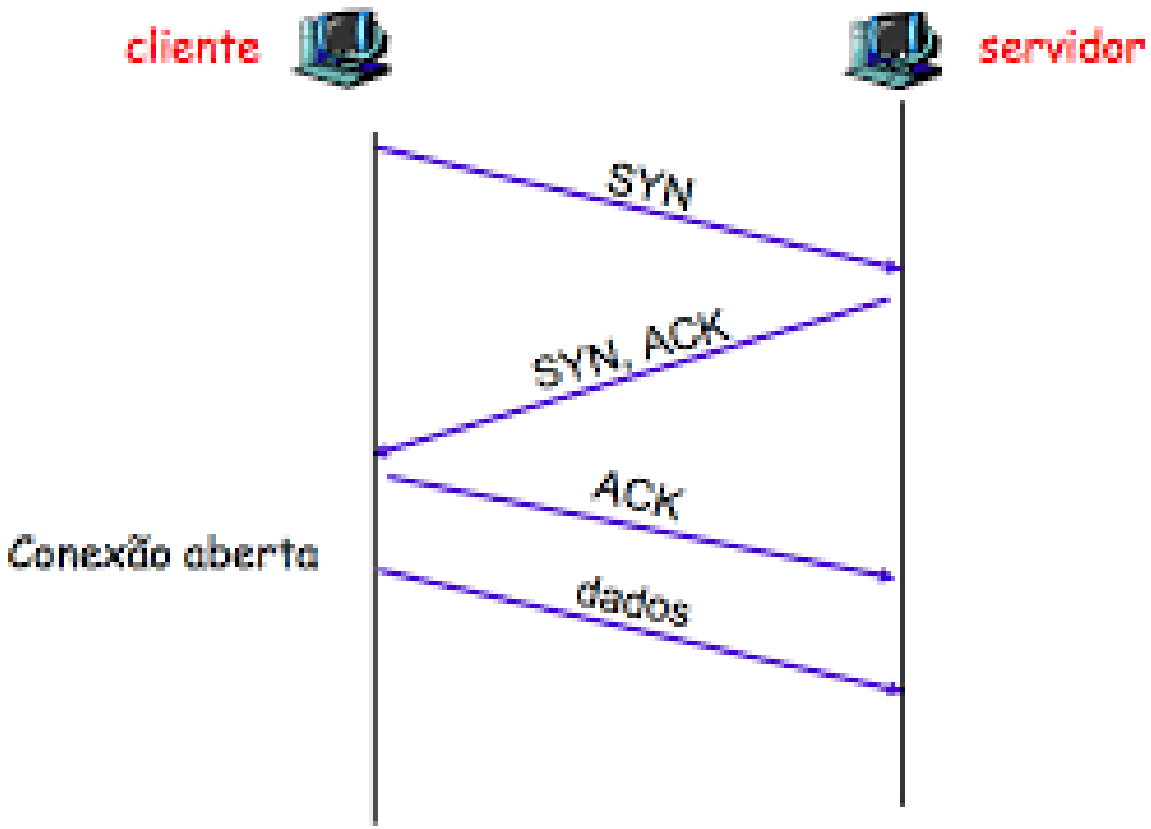
```
1  from socket import *
2  #Porta do servidor
3  PortaServidor= 12000
4
5  SocketServidor = socket(AF_INET,SOCK_DGRAM)
6  SocketServidor.bind(('',PortaServidor))
7  SocketServidor.settimeout(1000)
8
9  print("Server Ready")
10 while 1:
11
12   Palavra,EnderecoCliente = SocketServidor.recvfrom
       (1024)
13   PalavraModificada =Palavra.decode('utf-8').upper()
14   print("Enviando para o cliente: ",PalavraModificada)
15   SocketServidor.sendto(PalavraModificada.encode('utf
       -8'),EnderecoCliente)
```

UDP-client

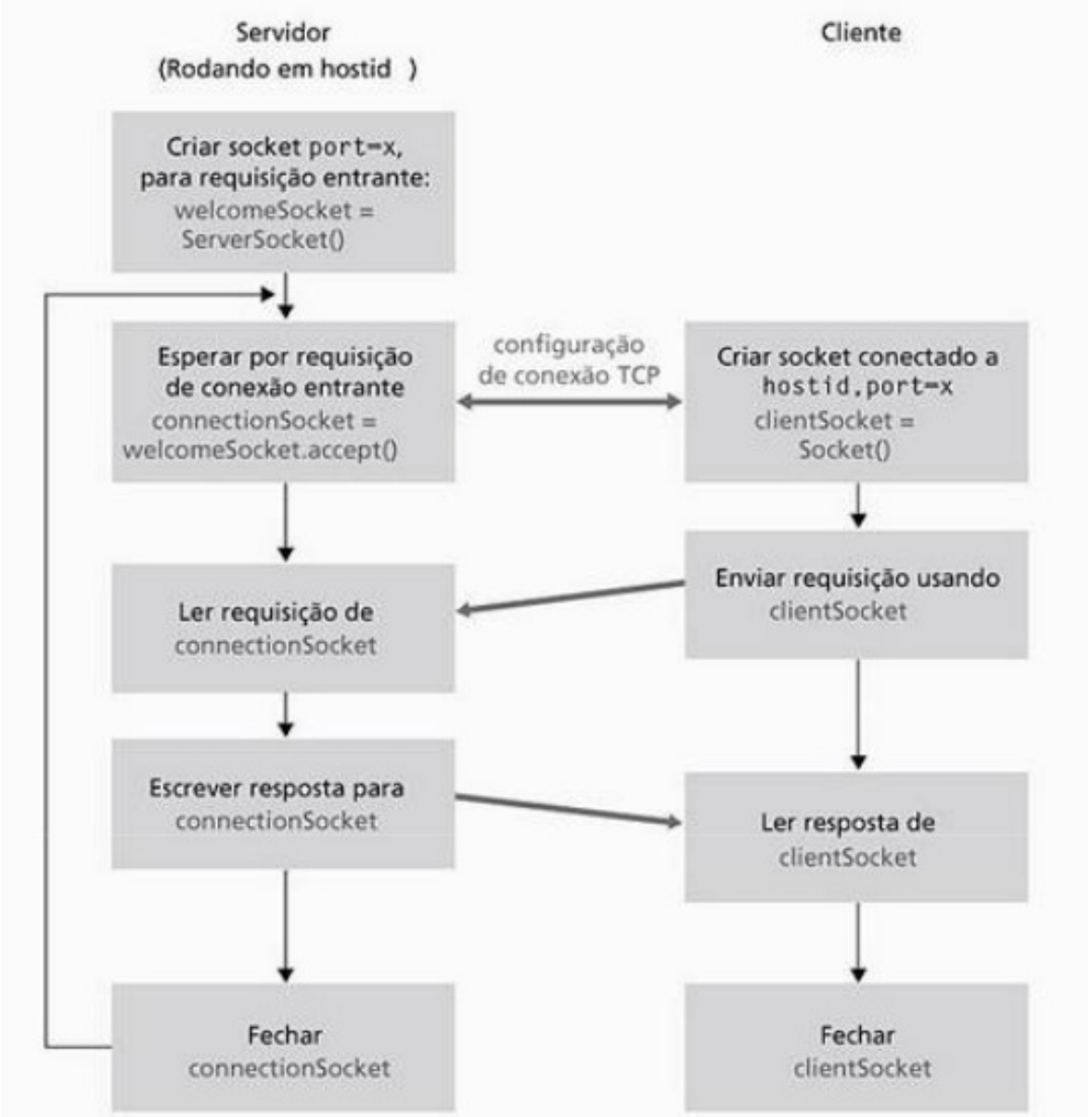
Programa cliente que envia uma frase de tamanho 1024 bytes.

```
1  from socket import *
2  #Server Addess.
3  servername = 'localhost'
4  serverPort = 12000
5  #Create an INET , STREAM socket (UDP).
6  clientSocket = socket(AF_INET,SOCK_DGRAM)
7  clientSocket.connect((servername,serverPort))
8  clientSocket.settimeout(10)
9  sentence = input('Input text: ')
10 #sendto server adress + server port.
11 clientSocket.sendto(sentence.encode('utf-8'),(
       servername,serverPort))
12 modifiedSentence= clientSocket.recv(1024)
13 print("From server: ",modifiedSentence.decode('utf-8')
       )
14 clientSocket.close()
```

Three way handshake



Socket TCP


Servidor
(Rodando em hostid)
Cliente
Criar socket port=x,
para requisição entrante:
welcomeSocket =
ServerSocket()
Esperar por requisição
de conexão entrante
connectionSocket =
welcomeSocket.accept()
configuração
de conexão TCP
Criar socket conectado a
hostid,port=x
clientSocket =
Socket()
Enviar requisição usando
clientSocket
Ler requisição de
connectionSocket
Escrever resposta para
connectionSocket
Ler resposta de
clientSocket
Fechar
connectionSocket
Fechar
clientSocket

TCP-server

Programa servidor que recebe uma frase de tamanho 1024 bytes e coloca em letras maiúsculas.

```
1  from socket import *
2
3  serverPort = 3000
4  serverSocket = socket(AF_INET, SOCK_STREAM)
5  #atribui a porta ao socket criado
6  serverSocket.bind(('', serverPort))
7  #aceita conexoes
8  #com no maximo um cliente na fila
9  serverSocket.listen(1)
10 print('The server is ready to receive')
11 while True:
12   connectionSocket, addr = serverSocket.accept()
13     #recebe a mensagem do cliente em bytes
14   mensagem = connectionSocket.recv(1024)
15   #envio tbm deve ser em bytes
16   mensagem = mensagem.upper()
17   connectionSocket.send(mensagem)
18   connectionSocket.close()
```

TCP-client

Programa cliente que envia uma frase de tamanho 1024 bytes.

```
1  from socket import *
2  serverName = 'localhost'
3  mensagem = "conceitos e tecnologias para dispositivos
       conectados"
4  serverPort = 3000
5  clientSocket = socket(AF_INET, SOCK_STREAM)
6  clientSocket.connect((serverName, serverPort))
7  #a mensagem deve estar em bytes antes de ser enviada
       ao buffer de transmissao
8  clientSocket.send(mensagem.encode())
9  #recebe a resposta do servidor
10 msg = clientSocket.recv(1024).decode()
11 #devemos converter a mensagem de volta para string
       antes de imprimi-la
12 print('Resposta:' , msg)
13 #fecha a conexao
14 clientSocket.close()
```

Citação do dia

- Servidor envia citações quando um cliente faz uma conexão.

```
1  #!/usr/bin/env python
2
3  import socket
4
5  s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
6
7  message = b''
8  addr = ("djxmmx.net", 17)
9
10 s.sendto(message, addr)
11
12 data, address = s.recvfrom(1024)
13 print(data.decode())
```

arquivo: texto.py

Obter IP de um site

- Função gethostname retorna o host
- Função gethostbyname retorna o endereço IPV4 local.

```
1 import socket
2 Meu_host_name = socket.gethostname()
3 print("Nome do host local e {}".format(Meu_host_name))
4 Meu_IP = socket.gethostbyname(Meu_host_name)
5 print(" Endereco IP do local host e {}".format(Meu_IP)
    )
```

arquivo: gethost.py

Obter IP de um site

- Função gethostbyname retorna o endereço IPV4 do site.

```
1 #!/usr/bin/env python
2 import socket
3 ip = socket.gethostbyname('uol.com.br')
4 print(ip)
```

arquivo: getipsite.py

Não funciona para todos os sites devido a redirecionamentos.

Obter IP de um site

- Função getaddrinfo lista os endereços IP e números de porta para host hostname e serviço servname.

```
1 import socket
2 from pprint import pprint
3
4
5 addrinfo = socket.getaddrinfo('uol.com.br', 'www')
6
7 print(addrinfo)
```

arquivo: site.py

UDP-server

```
1  #!usr/bin/python
2  import socket
3  sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
4  udp_host = socket.gethostname()
5  udp_port = 12345
6  sock.bind((udp_host,udp_port))
7  while True:
8      print("Esperando pelo cliente...")
9      data,addr = sock.recvfrom(1024)
10     Data=data.decode()
11     print("Mensagem recebida:",Data," from",addr)
```

arquivo: udpserver2.py

UDP-client

```
1 import socket
2 sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
3 udp_host = socket.gethostname()
4 udp_port = 12345
5 msg = "Bom dia!"
6 print("UDP host:", udp_host)
7 print("UDP Porta:", udp_port)
8 sock.sendto(msg.encode(),(udp_host,udp_port))
```

arquivo: udpclient2.py

Transferir arquivos pelo computador

```
1  import socket
2  print("Clinte")
3  HOST='localhost'
4  PORT=57000
5  s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
6  print("conectando com servidor...")
7  s.connect((HOST,PORT))
8  print("abrindo arquivo...")
9  arq=open('oi.txt','rb')
10 print("enviado  arquivo")
11 for i in arq:
12     #print i
13     s.send(i)
14 print("saindo...")
15 arq.close()
16 s.close()
```

arquivo: *clientsock.py*

Transferir arquivos pelo computador arquivo: *servsock.py*

```
1  import socket
2  print("Servidor")
3  HOST = "localhost"
4  PORT = 57000
5  s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6  print("Escutando a porta...")
7  s.bind((HOST,PORT))
8  s.listen(1)
9  print("Aceitando a conexao...")
10 conn,addr= s.accept()
11 print("recebendo o arquivo...")
12 arq = open('oi.txt','wb')
13 while 1:
14
15     dados=conn.recv(1024)
16     if not dados:
17         break
18     arq.write(dados)
19 print("saindo...")
20 arq.close()
21 conn.close()
```

UDP ping

```
1  # -*- coding: utf-8 -*-
2
3  import random
4  from socket import *
5
6  serverSocket = socket(AF_INET, SOCK_DGRAM)
7  serverSocket.bind(('', 12345))
8
9  while True:
10
11     rand = random.randint(0, 10)
12
13     message, address = serverSocket.recvfrom(1024)
14
15
16     if rand < 4:# taxa de perdas de 40 porcento
17         continue
18
19     serverSocket.sendto(message, address)
```

arquivo: UDPPingerServer.py

UDP ping: arquivo: UDPPingerClient.py

```
1  import time
2  from socket import *
3  clientSocket = socket(AF_INET, SOCK_DGRAM)
4  clientSocket.settimeout(1)
5  remoteAddr = (gethostname(), 12345)
6  for i in range(10):
7      sendTime = time.time()
8      message = 'PING ' + str(i + 1) + " " + str(time.
       strftime("%H:%M:%S"))
9      clientSocket.sendto(message.encode(), remoteAddr)
10     try:
11         data, server = clientSocket.recvfrom(1024)
12         Data=data.decode()
13         recdTime = time.time()
14         rtt = recdTime - sendTime
15         print("Message Received", Data)
16         print("Round Trip Time", rtt)
17         print()
18     except timeout:
19         print('REQUEST TIMED OUT')
20 clientSocket.close()
```

Trabalho 1

- 1) Qual a importância dos sockets para a comunicação em redes de computadores?
- 2) O que acontece se iniciar um cliente antes do servidor em um socket TCP? O que acontece se iniciar um cliente antes do servidor em um socket UDP?

3) Crie um programa servidor que envia uma questão de múltipla escolha para um cliente após este se conectar, o cliente deve responder a questão e o servidor retornar a resposta:
Exemplo de questão
Qual é a capital da Itália?

- a) Roma;
- b) Paris;
- c) Lisboa;
- d) Londres.

O servidor deve retornar uma mensagem dizendo "Você acertou!"ou "Você errou, a resposta correta é Roma (letra a)".
Fazer variações da questão exemplo.

Trabalho 1

Criar um arquivo compactado com o código comentado e um documento word com os enunciados, respostas e prints dos testes.
Data da entrega: 22/02/2022 pelo teams.