

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Sistemas Distribuídos *- Sockets -*

Arquitetura TCP/IP

Modelo OSI

Aplicação

Apresentação

Sessão

Transporte

Rede

Enlace

Física

Arquitetura TCP / IP

Aplicação

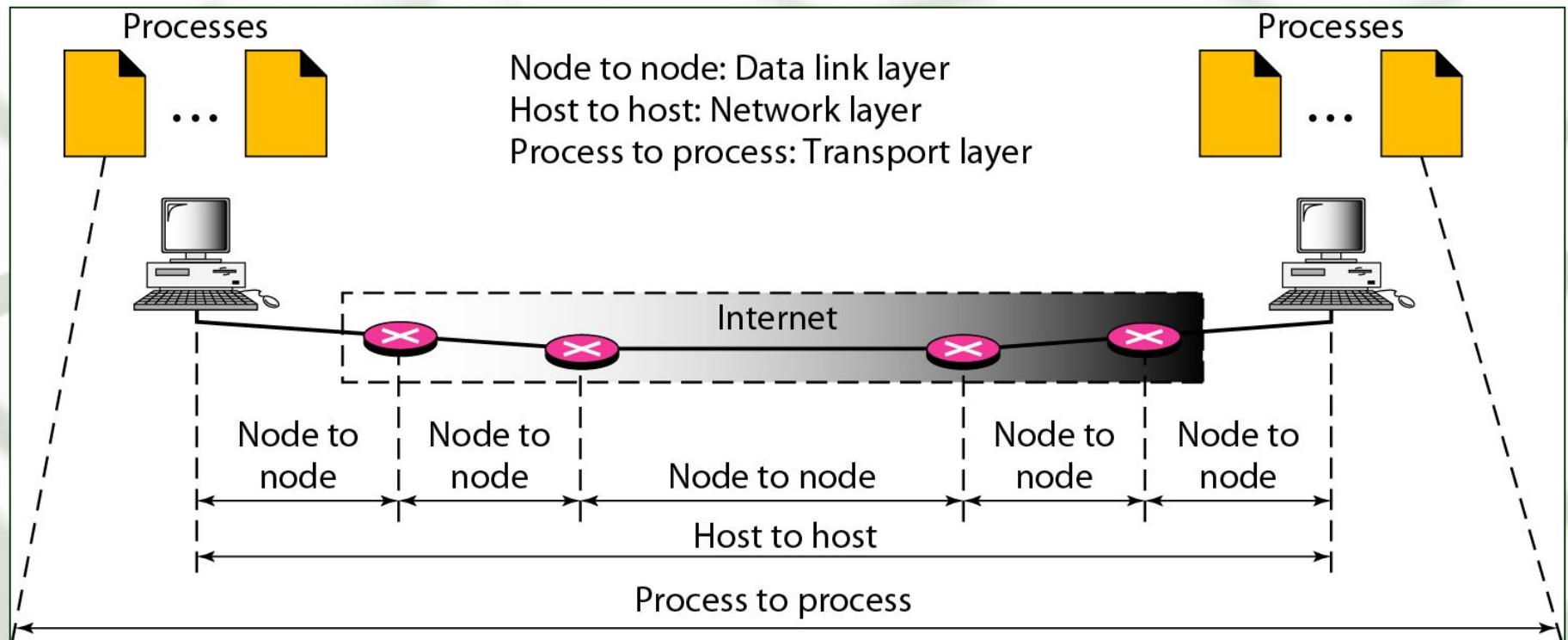
Transporte

Internet / Inter-Redes

Host/Rede
Interface de Rede

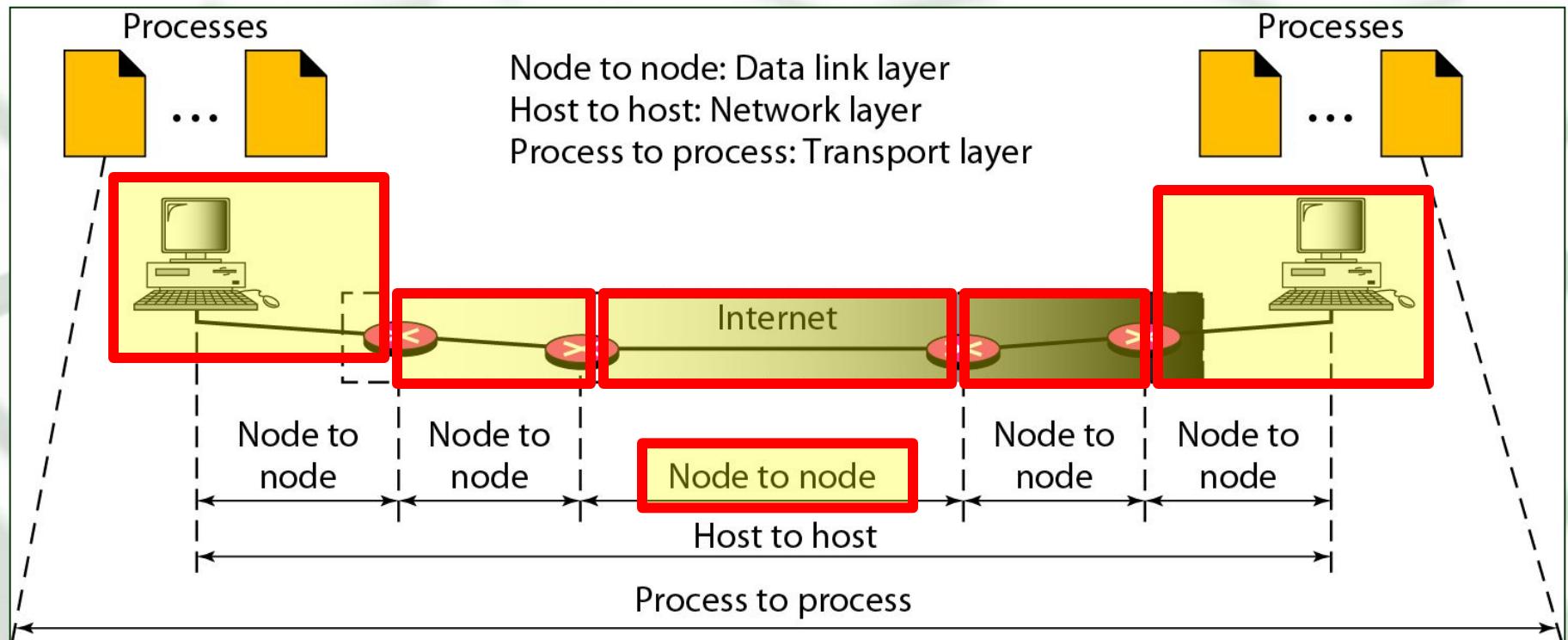
Arquitetura TCP/IP

■ Modelo de Comunicação em Camadas.



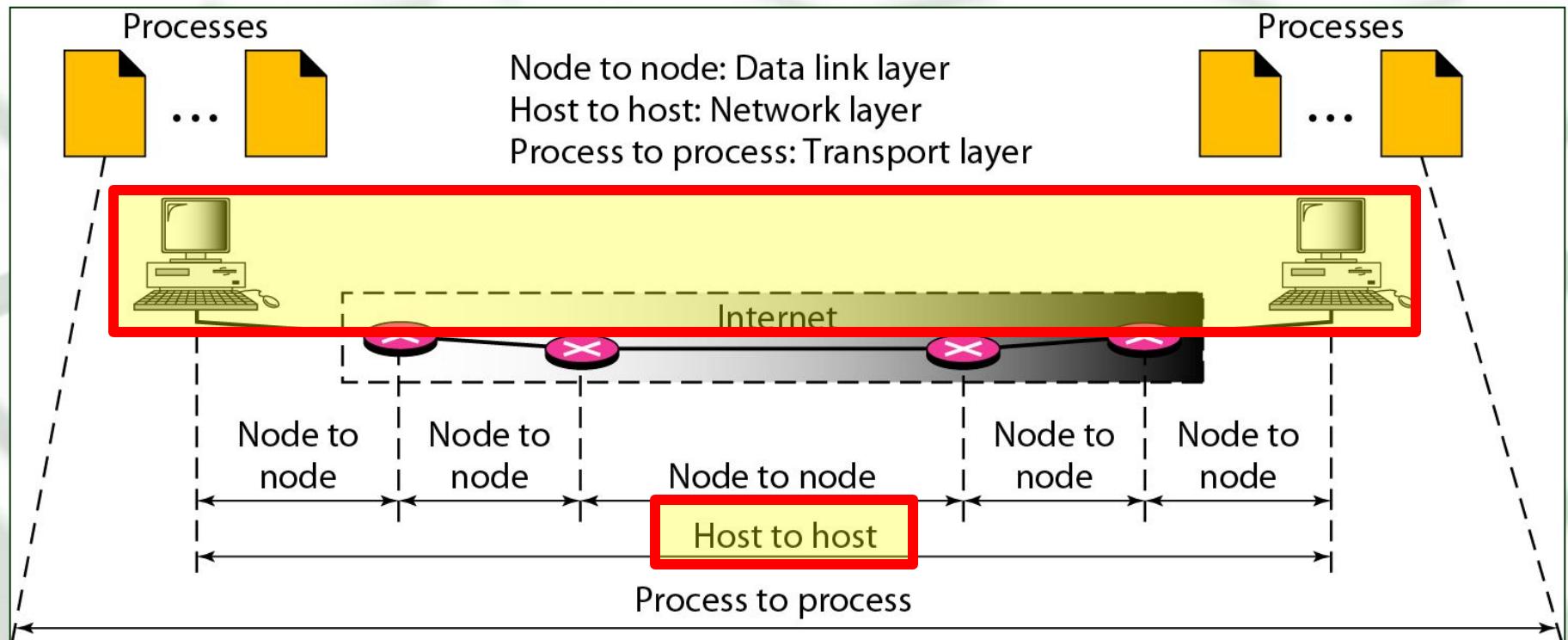
Arquitetura TCP/IP

■ Comunicação *node-to-node* => Enlace.



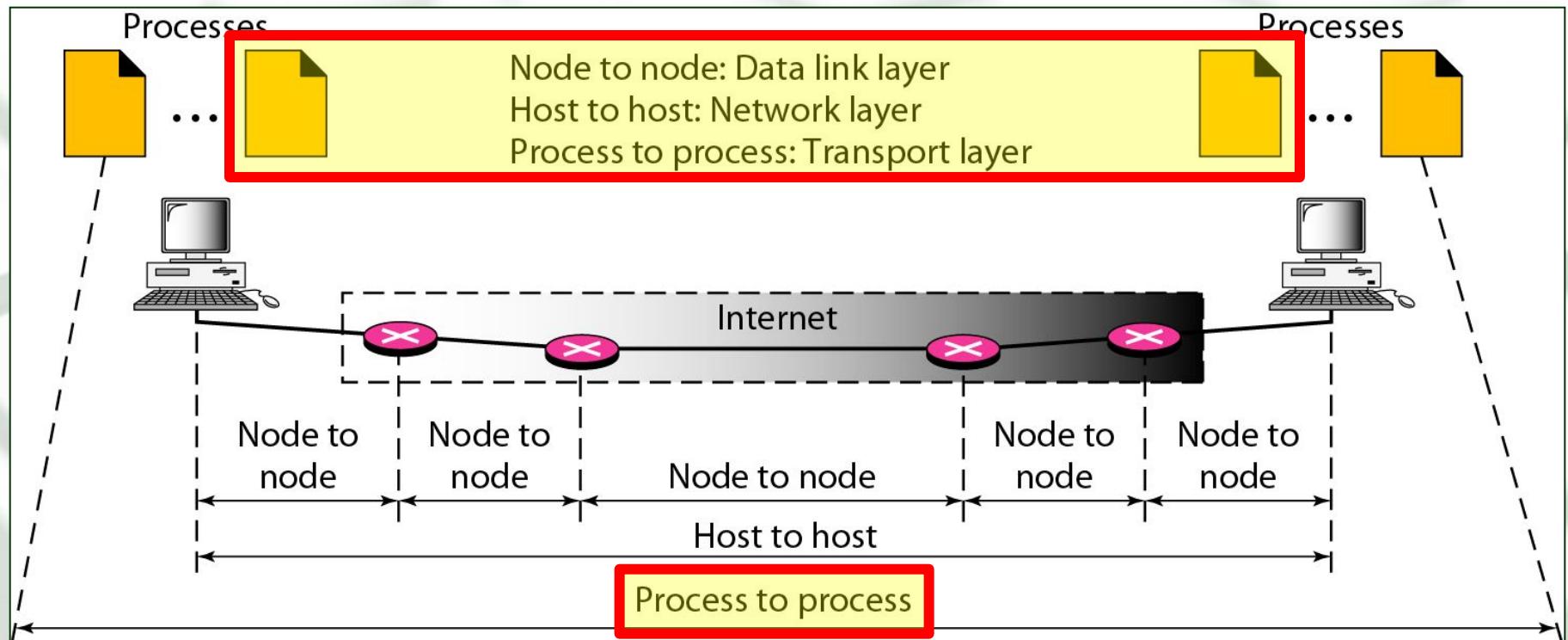
Arquitetura TCP/IP

■ Comunicação *host-to-host* => Rede.



Arquitetura TCP/IP

■ Comunicação *process-to-process* => Transporte.



Arquitetura TCP/IP

Modelo OSI

Aplicação

Apresentação

Sessão

Transporte

Rede

...

...

Física

Arquitetura TCP / IP

Aplicação

Transporte

-Redes

e

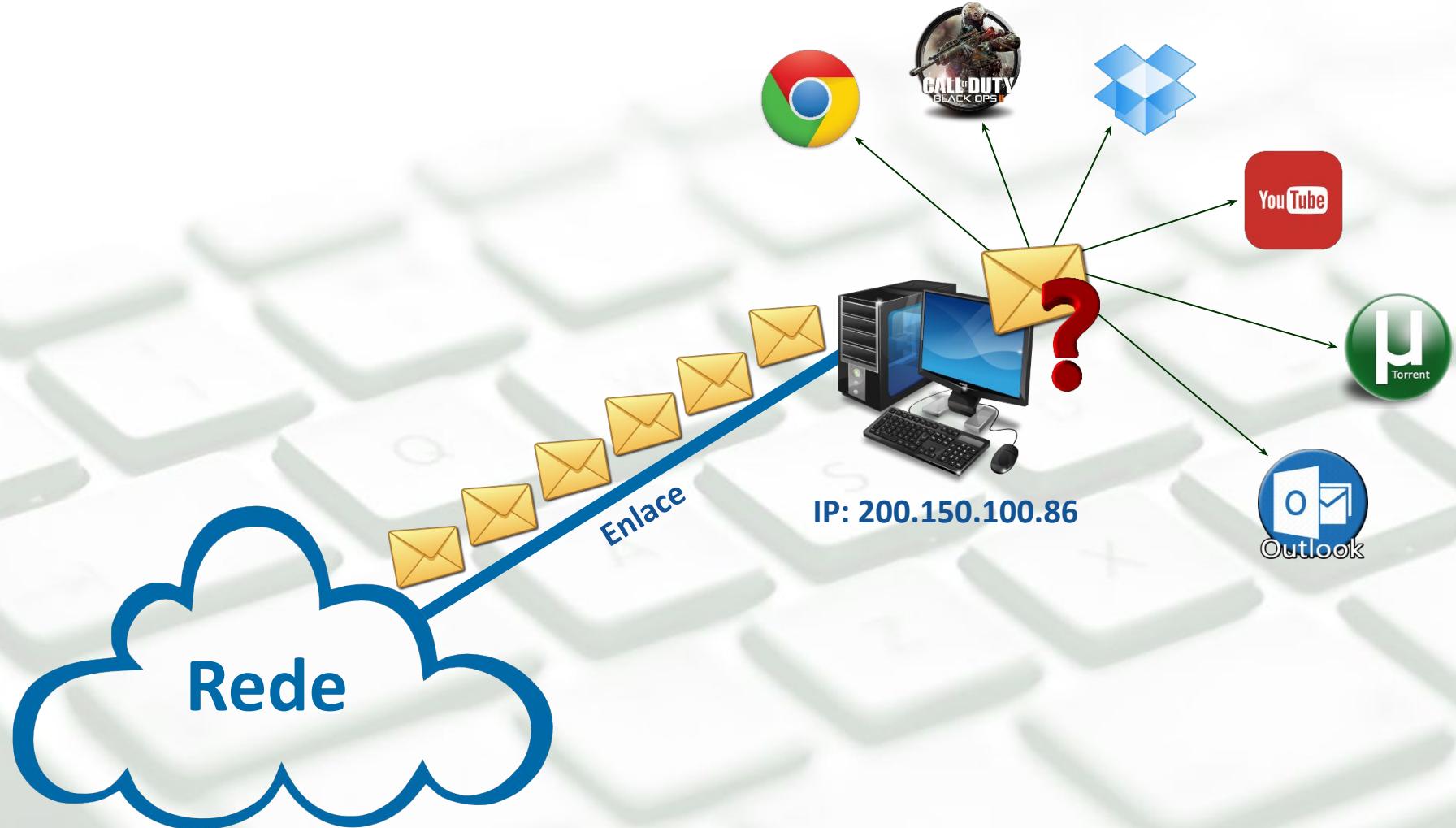
Rede

A Camada de Transporte é responsável por transportar os dados entre as aplicações distribuídas.

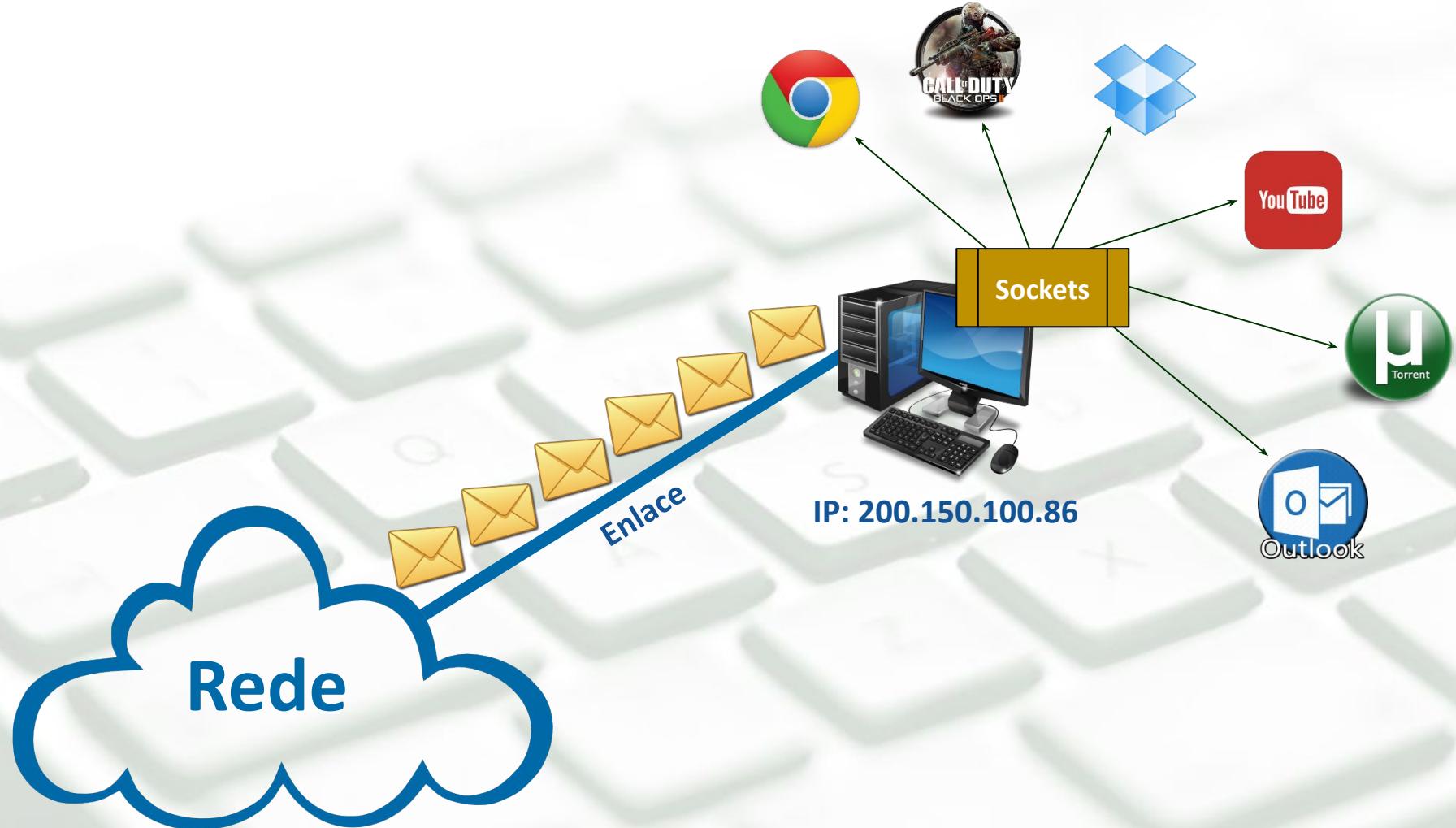
Camada de Transporte

- O sentido real de uma rede de computadores é fazer com que **aplicações (processos) troquem informações entre si => IPC (*Inter-Process Communication*)**.
- Cada *host* porém, pode estar executando, paralelamente e concorrentemente, inúmeros processos que consomem a mesma rede de comunicação.
- A **Camada de Transporte** é a responsável pelo gerenciamento da comunicação **inter-processos** em execução nos *endpoints*.
- *Vamos entender melhor como isso funciona...*

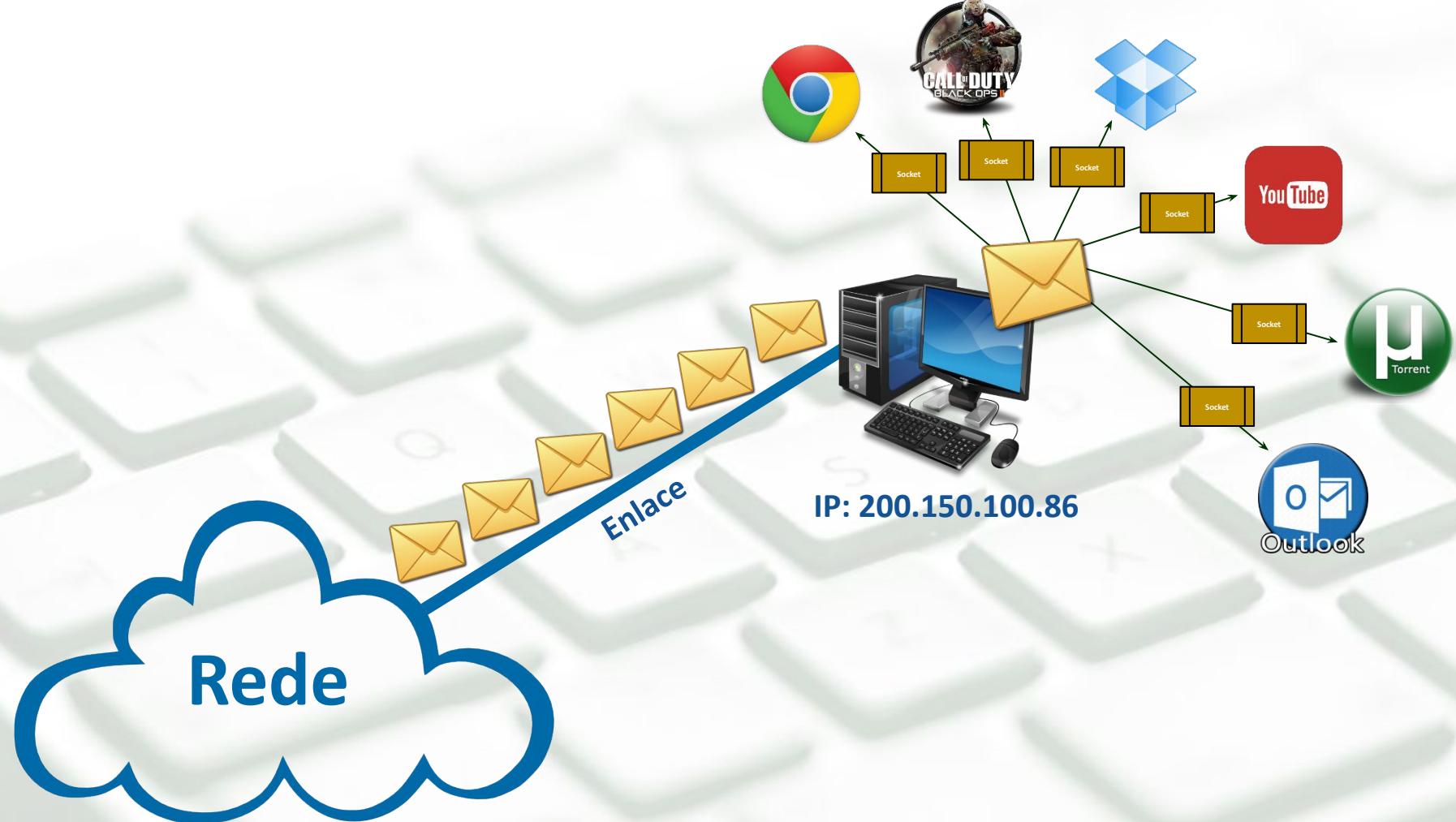
Arquitetura TCP/IP



Arquitetura TCP/IP



Arquitetura TCP/IP



Sockets

BSD Sockets

A API PRIMORDIAL

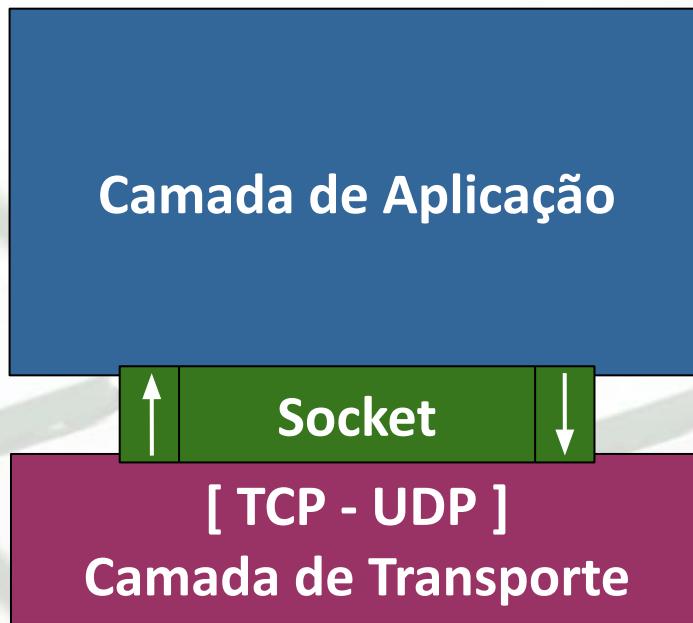


Sockets

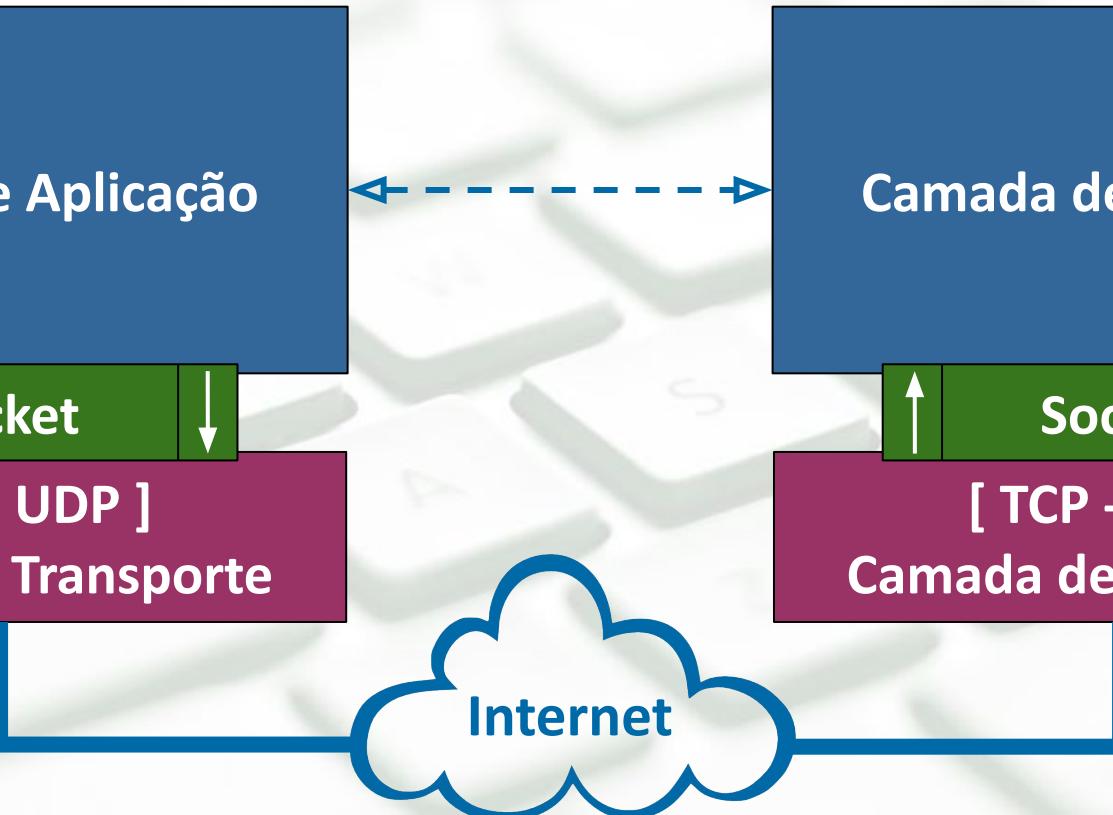
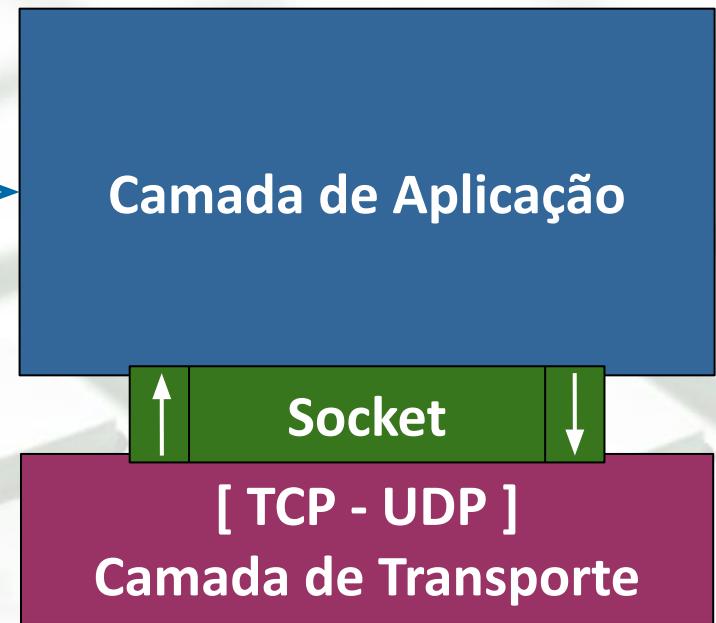
- **Socket (soquete)** é uma **API (Application Programming Interface)** que abstrai (*simplifica*) o uso da camada de rede para as aplicações (IPC).
- A API **Socket** permite que uma aplicação se comunique através da rede **sem que o desenvolvedor se preocupe com os detalhes técnicos de baixo nível da implementação da pilha TCP/IP.**

Sockets

Cliente

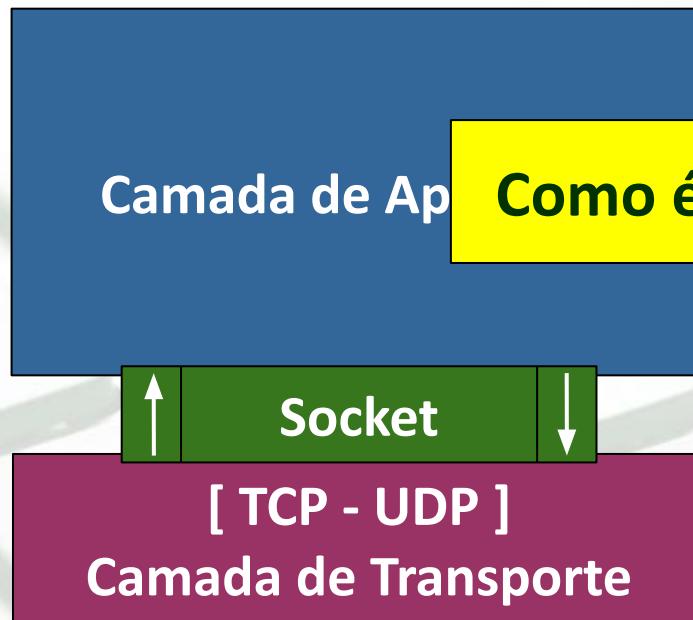


Servidor

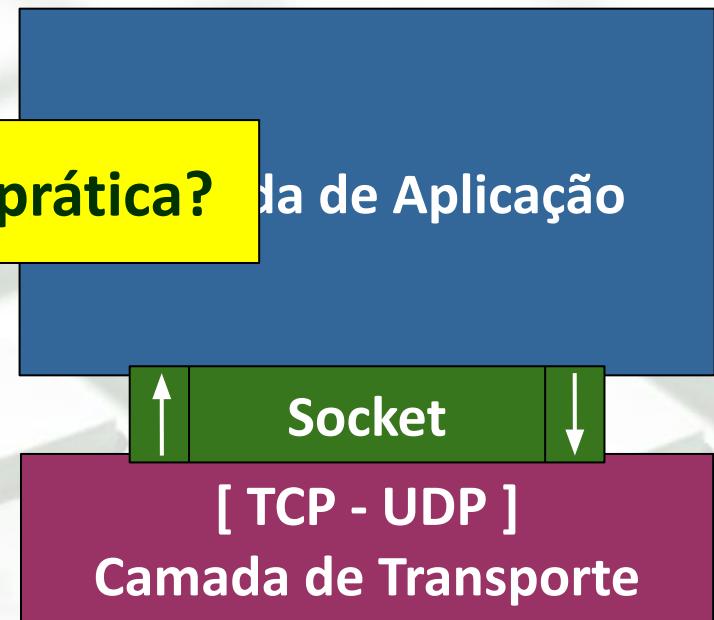


Sockets

Cliente



Servidor



Como é um Socket na prática?

Sockets

- A representação de um socket se resume à combinação de duas informações:
 - Endereço IP da interface do *endpoint*;
 - Porta de comunicação, TCP ou UDP;

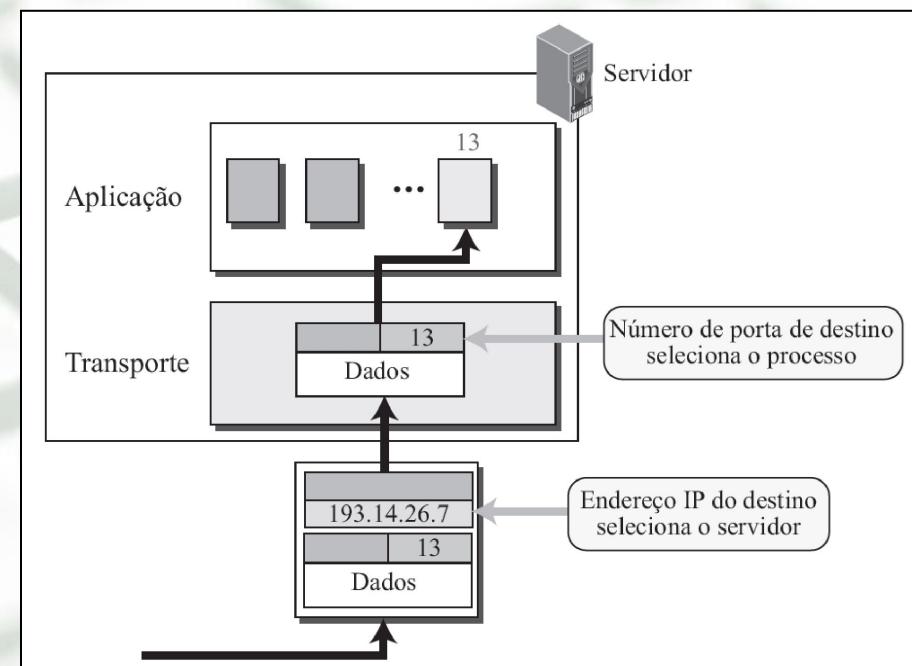
ENDEREÇO_IP : PORTA_DE_COMUNICAÇÃO



Uma porta mapeia um determinado processo que está utilizando a rede de comunicação naquele endpoint.

Sockets

- Perceba que para todo nível de comunicação, um esquema de endereçamento é adotado...
- Em nível de Enlace...
 - MAC Address
- Em nível de Rede...
 - IP Address
- Em nível de Transporte...
 - Port Number



Sockets

■ Retorno do comando: **netstat -tunp**

t	tcp
u	udp
n	valores numéricos
p	processos
l	estado listen: servidor
a	todas, como cliente e servidor

```
Arquivo Editar Ver Pesquisar Terminal Ajuda
adriano@adriano-All-Series:~$ netstat -tunp
(Nem todos os processos puderam ser identificados, informações sobre processos
de outrem não serão mostrados, você deve ser root para vê-los todos.)
Conexões Internet Ativas (sem os servidores)
Proto Recv-Q Send-Q Endereço Local          Endereço Remoto      Estado      PID/Program name
tcp      0      0 10.0.0.109:34212           162.125.19.131:443  ESTABELECIDA 4990/dropbox
tcp      32     0 10.0.0.109:53798           192.184.81.204:443  ESPERANDO_FECHAR 7001/chrome --type=
tcp      0      0 10.0.0.109:44916           157.240.226.17:443  ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:44884           157.240.226.17:443  ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:40936           142.251.128.37:443  ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:46584           31.13.74.52:443    ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:52556           172.67.68.82:443    ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:44840           157.240.226.17:443  ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:37594           172.217.192.188:5228 ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:52206           192.16.58.8:80       ESTABELECIDA 7001/chrome --type=
tcp      32     0 10.0.0.109:53806           192.184.81.204:443  ESPERANDO_FECHAR 7001/chrome --type=
tcp      32     0 10.0.0.109:53796           192.184.81.204:443  ESPERANDO_FECHAR 7001/chrome --type=
tcp      0      0 10.0.0.109:40108           31.13.74.18:443    ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:35610           151.101.178.133:443 ESTABELECIDA 7001/chrome --type=
tcp      32     0 10.0.0.109:53800           192.184.81.204:443  ESPERANDO_FECHAR 7001/chrome --type=
tcp      32     0 10.0.0.109:53804           192.184.81.204:443  ESPERANDO_FECHAR 7001/chrome --type=
tcp      32     0 10.0.0.109:53802           192.184.81.204:443  ESPERANDO_FECHAR 7001/chrome --type=
tcp      0      0 10.0.0.109:40100           31.13.74.18:443    ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:55378           162.125.19.9:443    ESTABELECIDA 4990/dropbox
tcp      0      0 10.0.0.109:47274           142.250.218.14:443  ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:60564           142.251.129.202:443 ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:53029           216.58.222.14:443  ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:33630           142.251.132.227:443 ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:41914           64.233.186.189:443  ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:37960           142.251.129.202:443 ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:46359           142.250.219.3:443   ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:38226           172.217.30.164:443  ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:46614           142.251.129.202:443 ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:59018           216.239.32.116:443  ESTABELECIDA 7001/chrome --type=
adriano@adriano-All-Series:~$
```

Sockets

■ Retorno do comando: **netstat -tunp**

- t tcp**
- u udp**
- n valores numéricos**
- p processos**
- l estado listen: servidor**
- a todas, como cliente e servidor**

```

Arquivo Editar Ver Pesquisar Terminal Ajuda
adriano@adriano-All-Series:~$ netstat -tunp
(Nem todos os processos puderam ser identificados, informações sobre processos
de outrem não serão mostrados, você deve ser root para vê-los todos.)
Conexões Internet Ativas (sem os servidores)
Proto Recv-Q Send-Q Endereço Local          Endereço Remoto      Estado      PID/Program name
tcp      0      0 10.0.0.109:34212           162.125.19.131:443  ESTABELECIDA 4990/dropbox
tcp      0      0 10.0.0.109:46584           31.13.74.52:443    ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:52556           172.67.68.82:443   ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:44840           157.240.226.17:443 ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:37594           172.217.192.188:5228 ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:52206           192.16.58.8:80      ESTABELECIDA 7001/chrome --type=
tcp      32     0 10.0.0.109:53806           192.184.81.204:443 ESPERANDO_FECHAR 7001/chrome --type=
tcp      32     0 10.0.0.109:53796           192.184.81.204:443 ESPERANDO_FECHAR 7001/chrome --type=
tcp      0      0 10.0.0.109:40108           31.13.74.18:443    ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:35610           151.101.178.133:443 ESTABELECIDA 7001/chrome --type=
tcp      32     0 10.0.0.109:53800           192.184.81.204:443 ESPERANDO_FECHAR 7001/chrome --type=
tcp      32     0 10.0.0.109:53804           192.184.81.204:443 ESPERANDO_FECHAR 7001/chrome --type=
tcp      32     0 10.0.0.109:53802           192.184.81.204:443 ESPERANDO_FECHAR 7001/chrome --type=
tcp      0      0 10.0.0.109:40100           31.13.74.18:443    ESTABELECIDA 7001/chrome --type=
tcp      0      0 10.0.0.109:55378           162.125.19.9:443   ESTABELECIDA 4990/dropbox
udp      0      0 10.0.0.109:47274           142.250.218.14:443 ESTABELECIDA 7001/chrome --type=
udp      0      0 10.0.0.109:60564           142.251.129.202:443 ESTABELECIDA 7001/chrome --type=
udp      0      0 10.0.0.109:53029           216.58.222.14:443  ESTABELECIDA 7001/chrome --type=
udp      0      0 10.0.0.109:33630           142.251.132.227:443 ESTABELECIDA 7001/chrome --type=
udp      0      0 10.0.0.109:41914           64.233.186.189:443 ESTABELECIDA 7001/chrome --type=
udp      0      0 10.0.0.109:37960           142.251.129.202:443 ESTABELECIDA 7001/chrome --type=
udp      0      0 10.0.0.109:46359           142.250.219.3:443  ESTABELECIDA 7001/chrome --type=
udp      0      0 10.0.0.109:38226           172.217.30.164:443 ESTABELECIDA 7001/chrome --type=
udp      0      0 10.0.0.109:46614           142.251.129.202:443 ESTABELECIDA 7001/chrome --type=
udp      0      0 10.0.0.109:59018           216.239.32.116:443 ESTABELECIDA 7001/chrome --type=
```

Um par de sockets define uma conexão

```

adriano@adriano-All-Series:~$ 
adriano@adriano-All-Series:~$ 
```

Portas de Comunicação

- Na arquitetura TCP/IP, uma porta de comunicação é um número inteiro de 16 bits.
 - Valores entre 0 <=> 65.535

Header Protocolo TCP:

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	3			
Source Port															Destination Port																				
Sequence Number																																			
Acknowledgment Number																																			
Data Offset	Reserved	U R	A C	P S	R S	S Y	F I	Window																											
Checksum															Urgent Pointer																				
Options															Padding																				
data																																			

Portas de Comunicação

- Na arquitetura TCP/IP, uma porta de comunicação é um número inteiro de 16 bits.
 - Valores entre 0 <=> 65.535

Nome	Faixa	Descrição
Portas Conhecidas	0 – 1023	Atribuídas e controladas pela IANA.
Portas Registradas	1024 – 49151	Registro junto à IANA. <u>Link para visualização</u>
Portas Dinâmicas	49152 – 65535	Portas para uso geral.

Portas de Comunicação

- Na arquitetura TCP/IP uma porta é um número inteiro entre 0 & 65535
 - Valores entre 0 <=> 65535

Nome	Faixa
Portas Conhecidas	0 – 1023
Portas Registradas	1024 – 49151
Portas Dinâmicas	49152 – 65535

Algumas Portas Conhecidas e Registradas

20 & 21: FTP
22: Secure Shell (SSH)
23: Telnet
25: SMTP
53: Domain Name System (DNS)
67 & 68: DCHP
80: HTTP Padrão
110: Post Office Protocol (POP3)
143: IMAP
161: SNMP
443: HTTP Secure (HTTPS)
3306: MySQL
5004: Real Time Protocol (RTP)
5938: TeamViewer

Modelo Cliente x Servidor



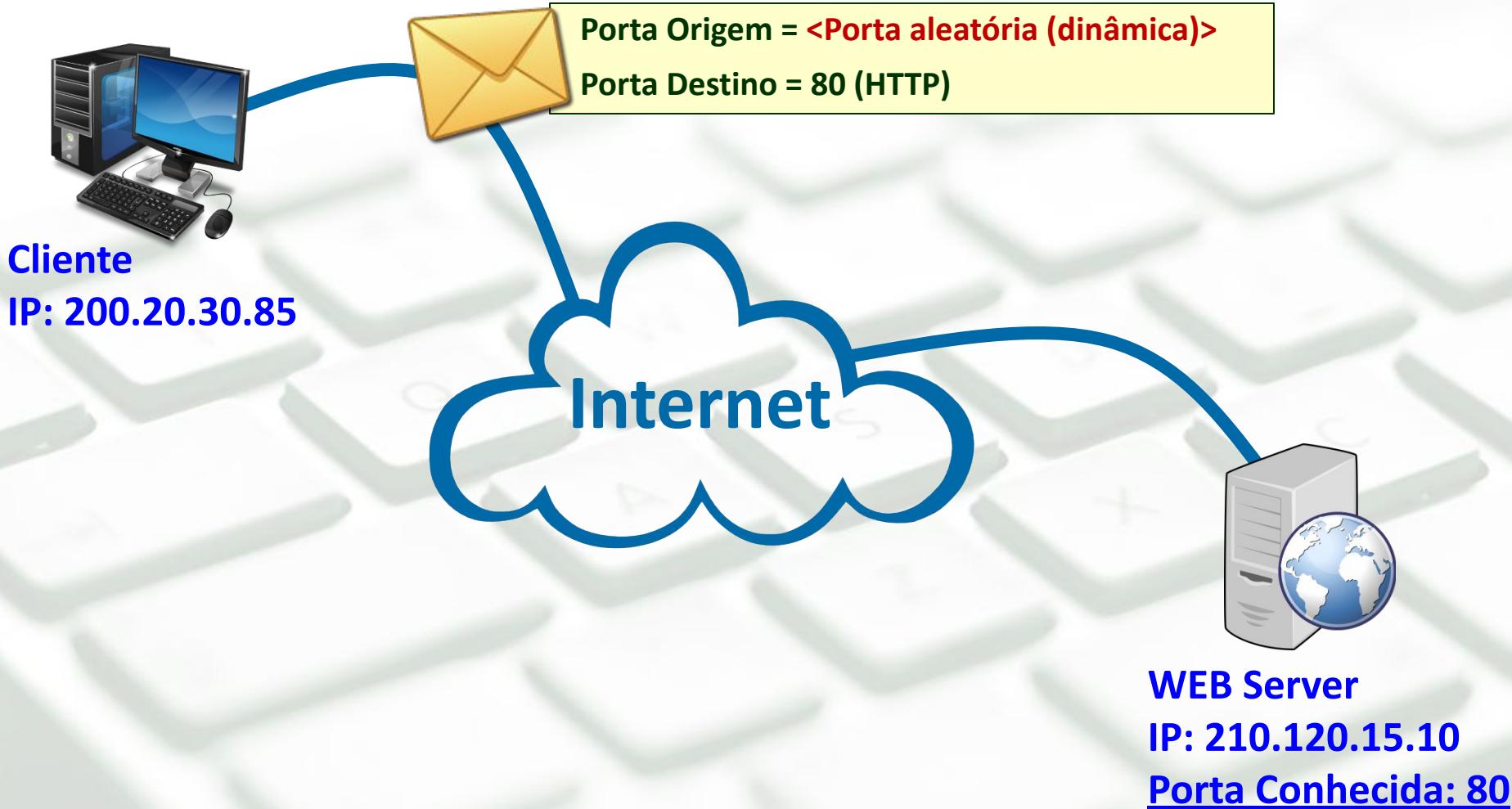
Cliente
IP: 200.20.30.85



WEB Server
IP: 210.120.15.10
Porta Conhecida: 80

O IP e porta (socket) do processo servidor deve ser conhecido.

Modelo Cliente x Servidor



Modelo Cliente x Servidor

Socket Cliente

Google Chrome
200.20.30.85:58569



Cliente
IP: 200.20.30.85

Socket Origem = 200.20.30.85:58569
Socket Destino = 210.120.15.10:80



Socket Servidor

Apache Server
210.120.15.10:80



WEB Server
IP: 210.120.15.10
Porta Conhecida: 80

A combinação **IP:PORTA** identifica de forma exclusiva um processo em execução na Internet.

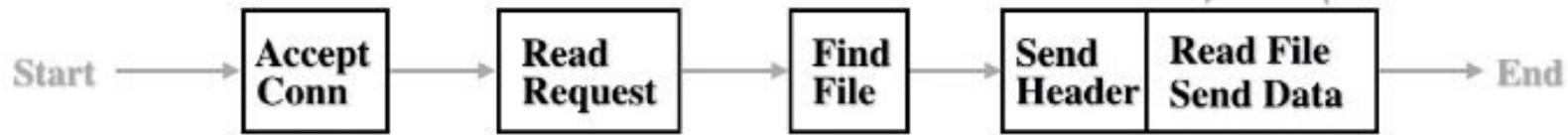
Implementação do Servidor

■ Servidor Single-Client

- Socket único para tratar requisições de clientes.
- I/O Bloqueante.
- Novo cliente só pode ser atendido após o anterior ter finalizado seu atendimento.

Exemplo de Funcionamento de um Servidor WEB

Fluxo de processamento:



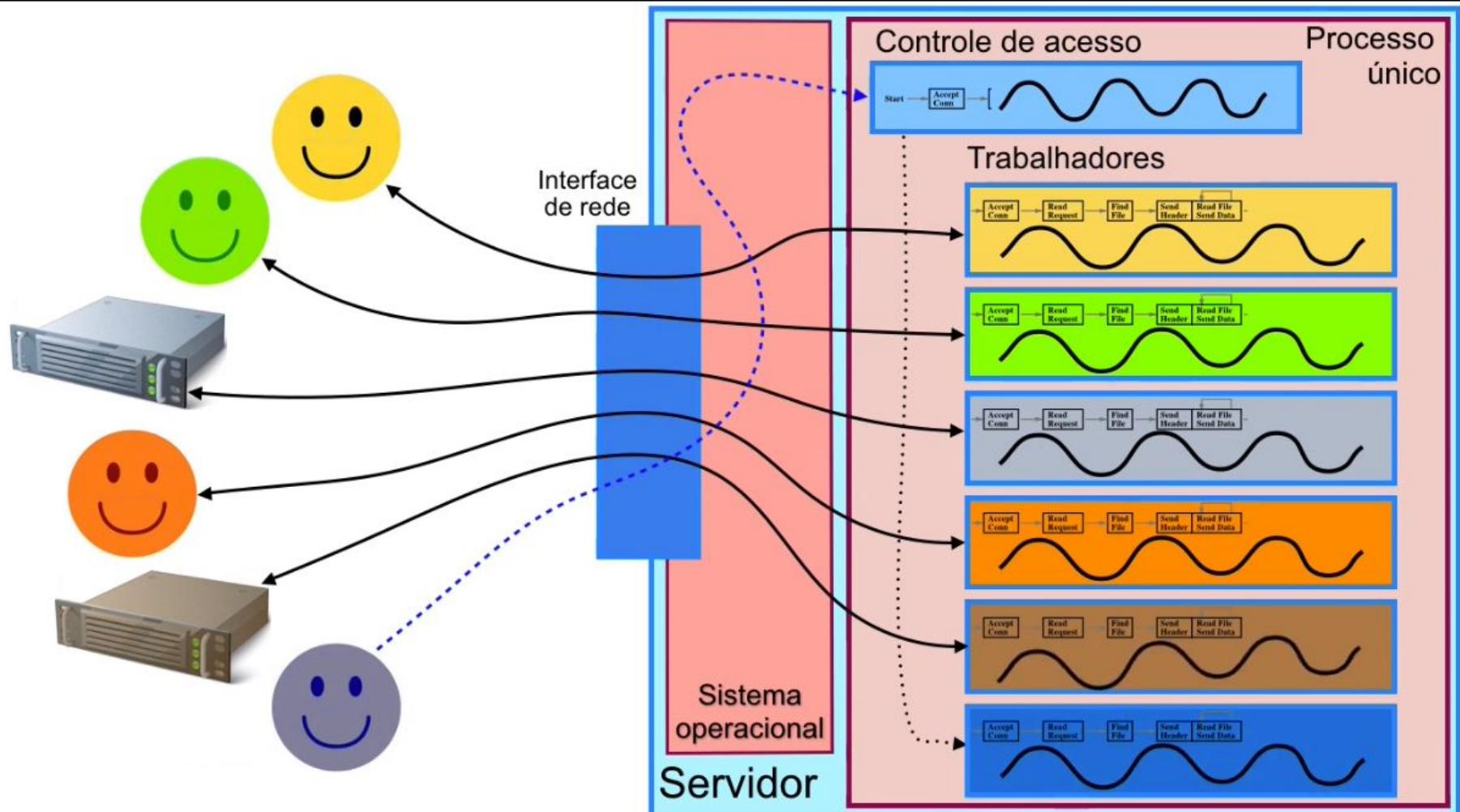
Implementação do Servidor

■ Servidor Multi-Client

- Workers criados sob-demanda, dedicados para atendimento de cada novo cliente conectado.

- A distribuição das tarefas de atendimento (requisições) pode ser baseado em múltiplos processos, *threads* ou corrotinas.

Servidor Multithread



Berkeley Sockets

- Em termos de desenvolvimento, a API Socket criada pela ***Universidade de Berkeley*** (1983) tornou-se padrão para a grande maioria das implementações em diversas linguagens de programação, inclusive Python.
- A API Berkeley consiste em um conjunto definido de funções primitivas para gerenciamento dos *sockets*, que veremos em detalhes a seguir...

Berkeley Sockets

Python » English ▾ 3.13.2 ▾ 3.13.2 Documentation » The Python Standard Library » Networking and Interprocess Communication » `socket` — Low-level networking interface

Table of Contents

- `socket` — Low-level networking interface
 - Socket families
 - Module contents
 - Exceptions
 - Constants
 - Functions
 - Creating sockets
 - Other functions
 - Socket Objects
 - Notes on socket timeouts
 - Timeouts and the `connect` method
 - Timeouts and the `accept` method
 - Example

Previous topic

Developing with `asyncio`

Next topic

`ssl` — TLS/SSL wrapper for socket objects

This Page

[Report a Bug](#)
[Show Source](#)

socket — Low-level networking interface

Source code: [Lib/socket.py](#)

This module provides access to the BSD socket interface. It is available on all modern Unix systems, Windows, MacOS, and probably additional platforms.

Note: Some behavior may be platform dependent, since calls are made to the operating system socket APIs.

Availability: not WASI.

This module does not work or is not available on WebAssembly. See [WebAssembly platforms](#) for more information.

The Python interface is a straightforward transliteration of the Unix system call and library interface for sockets to Python's object-oriented style: the `socket()` function returns a *socket object* whose methods implement the various socket system calls. Parameter types are somewhat higher-level than in the C interface: as with `read()` and `write()` operations on Python files, buffer allocation on receive operations is automatic, and buffer length is implicit on send operations.

«

Berkeley Sockets

■ Principais Primitivas

Socket() # Cria a estrutura de dados para comunicação

#Métodos Server

.bind() # Associa endereço e porta local à um Socket
.listen() # Coloca-se à disposição para aceitar conexões
.accept() # Bloqueia processo até obter conexão

#Métodos Client

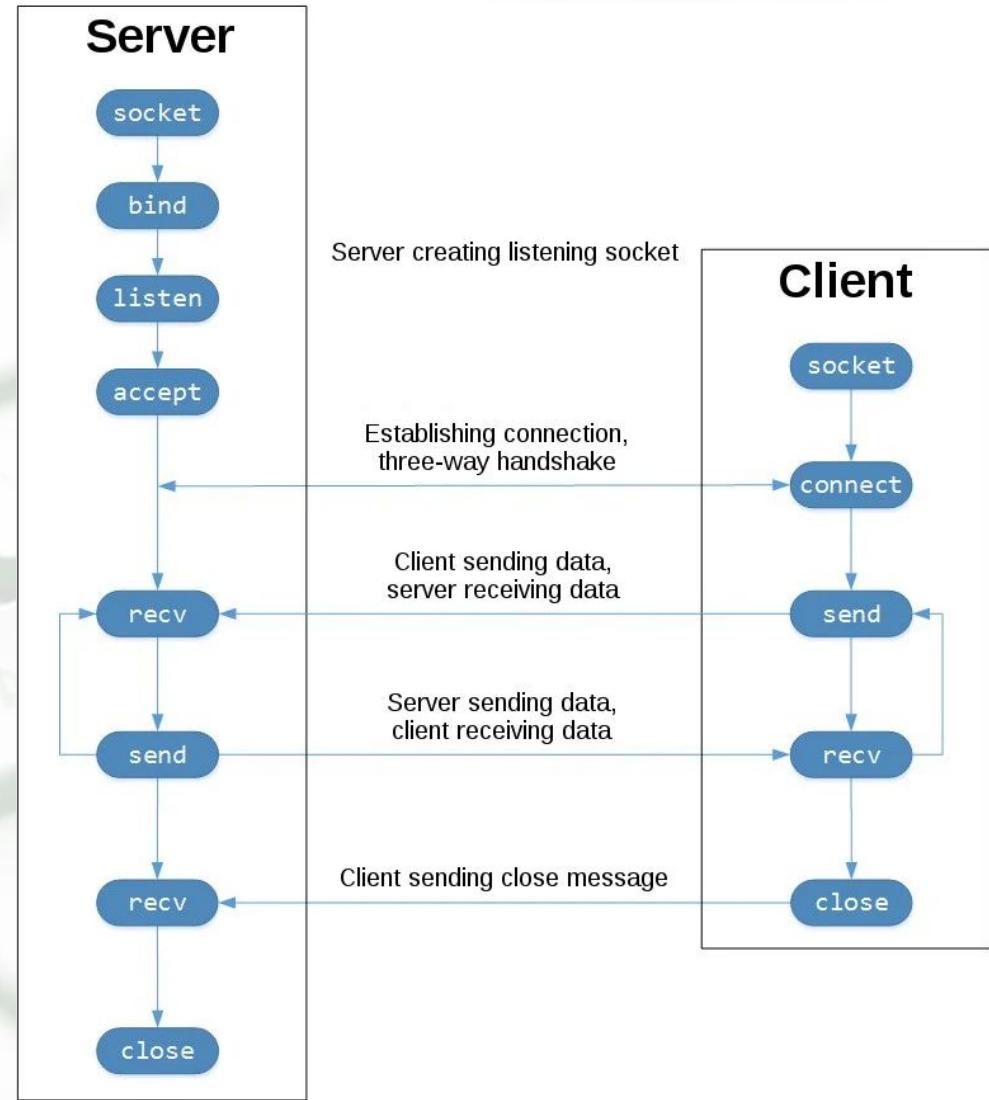
.connect() # Tenta estabelecer conexão remota com Servidor

#Métodos Client-Server

.send() # Envia dados pela conexão
.recv() # Recebe dados pela conexão
.close() # Finaliza conexão

Fluxo Sockets

■ Fluxo de Primitivas ■ Padrão TCP

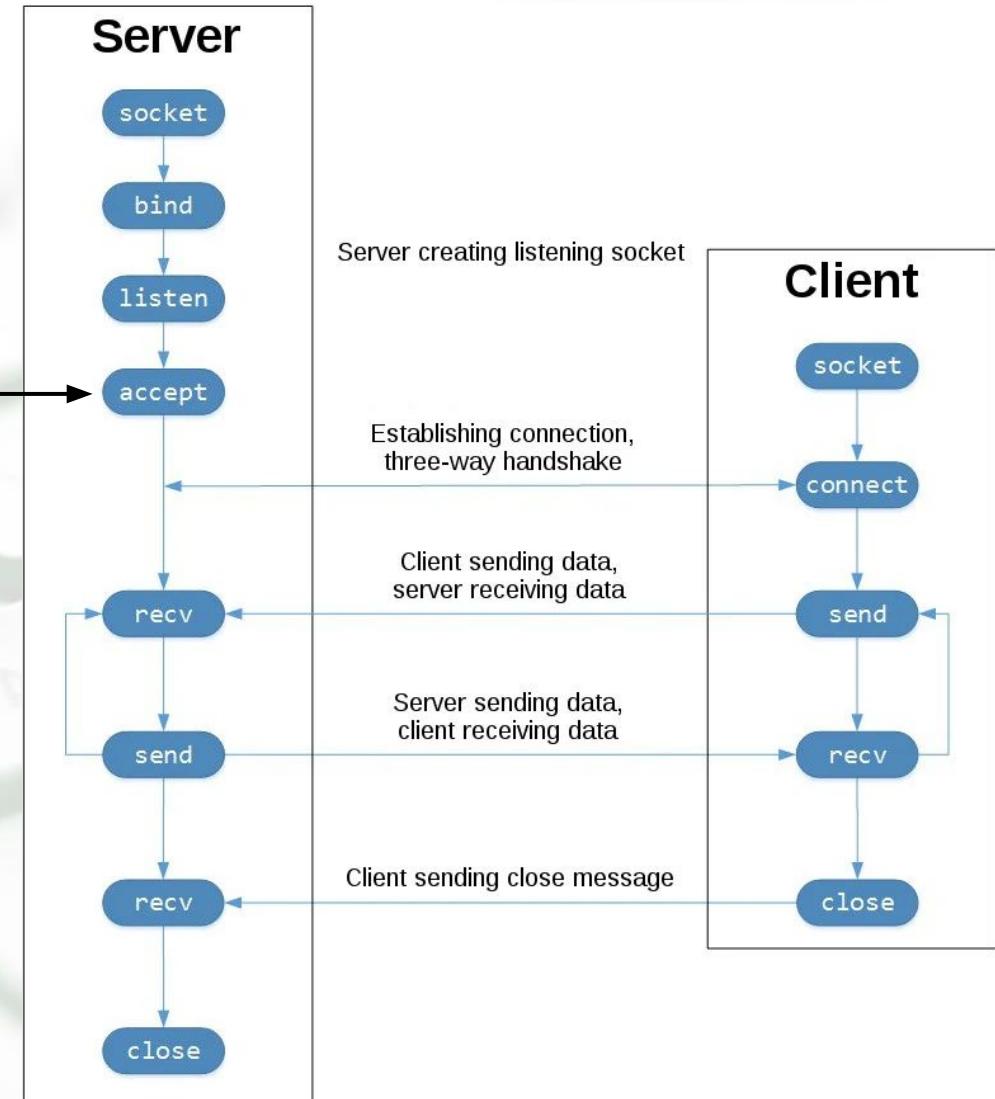


Fluxo Sockets

■ Fluxo de Primitivas

- Padrão TCP

O retorno do método **accept** é
um novo objeto **Socket()**
contendo as informações do
cliente (host, port)

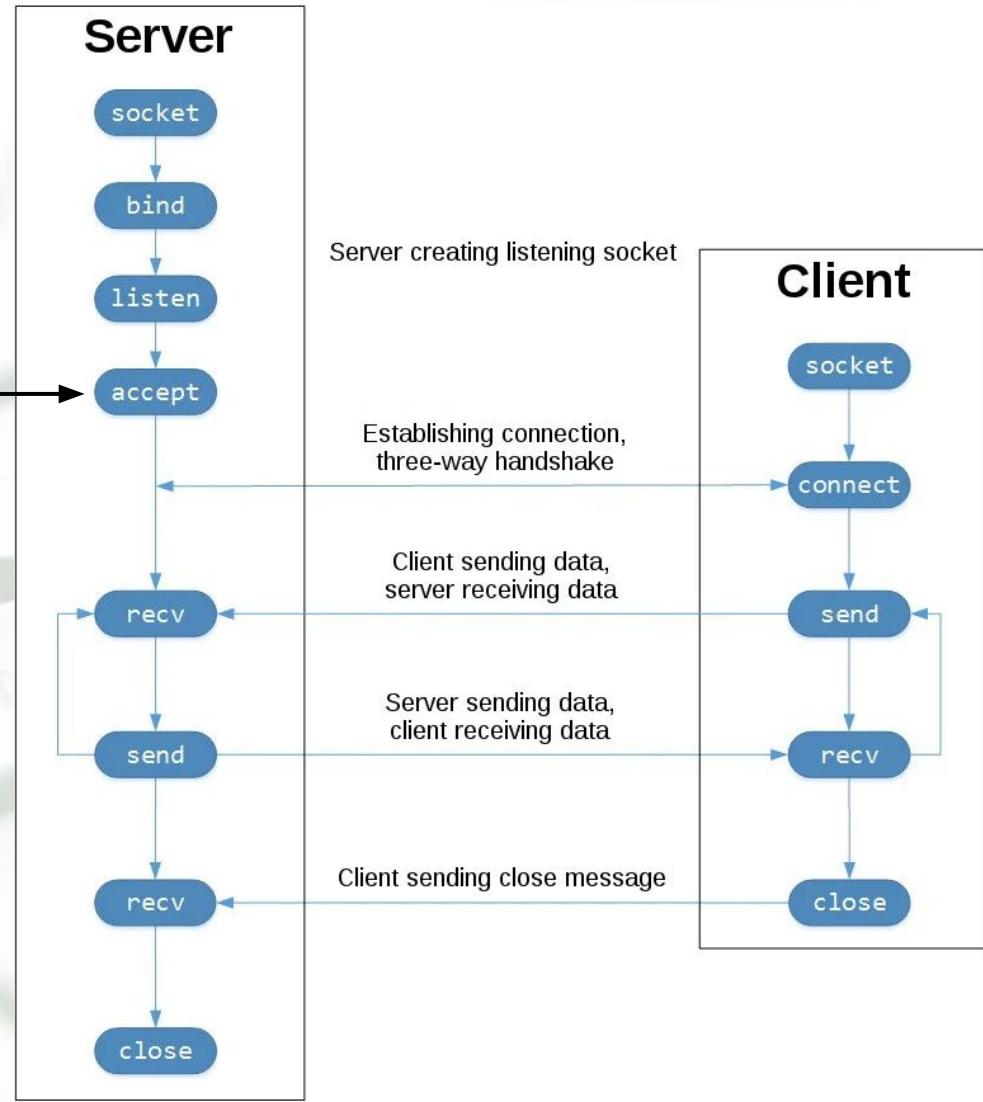


Fluxo Sockets

- Fluxo de Primitivas
 - Padrão TCP

O retorno do método **accept** é
um novo objeto **Socket()**
contendo as informações do
cliente (host, port)

Isso é muito útil pois permite a criação de um *socket* apenas para “aceitar conexões”, enquanto outros sockets (oriundos deste) podem tratar cada cliente exclusivamente.



Servidor Básico Python

```
import socket

HOST = '127.0.0.1'    # Interface padrão localhost / loopback
PORT = 65432           # Porta de escuta (não registrada)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
print(f'Servidor em Execução em {s.getsockname()}' )

while True:
    conn, addr = s.accept()
    print(f'Conectado por {addr}')
    data = conn.recv(1024).decode()
    print(f'Recebido: {data}')
    conn.send(str('Hello Client').encode())
    conn.close()
```

Servidor Básico Python

```
import socket

HOST = '127.0.0.1' # Interface padrão localhost / loopback
PORT = 65432        # Porta de escuta (não registrada)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
print(f'Servidor em Execução')

while True:
    conn, addr = s.accept()
    print(f'Conectado por {addr}')
    data = conn.recv(1024).decode()
    print(f'Recebido: {data}')
    conn.send(str('Hello Client').encode())
    conn.close()
```

AF_INET == IPv4
AF_INET6 == IPv6
SOCK_STREAM == TCP
SOCK_DGRAM == UDP

Servidor Básico Python

```
import socket

HOST = '127.0.0.1'    # Interface padrão localhost / loopback
PORT = 65432           # Porta de escuta (não registrada)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))

s.listen()  # Aguarda conexão de cliente (bloqueante)
print(f'Servidor em Execução em {s.getsockname()}')

while True:
    conn, addr = s.accept()
    print(f'Conectado por {addr}')
    data = conn.recv(1024).decode()
    print(f'Recebido: {data}')
    conn.send(str('Hello Client').encode())
    conn.close()
```

Servidor Básico Python

```
import socket

HOST = '127.0.0.1'      # Interface padrão localhost / loopback
PORT = 65432             # Porta de escuta (não registrada)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
print(f'Servidor em Execução em {s.getsockname()}')

while True:
    conn, addr = s.accept() ←
    print(f'Conectado por {addr}') ←
    data = conn.recv(1024).decode()
    print(f'Recebido: {data}')
    conn.send(str('Hello Client').encode())
    conn.close()
```

conn == Socket para atender cliente
addr == Informações da conexão

Servidor Básico Python

```
import socket

HOST = '127.0.0.1'    # Interface padrão localhost / loopback
PORT = 65432           # Porta de escuta (não registrada)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
print(f'Servidor em Execução em {s.getsockname()}')

while True:
    conn, addr = s.accept()
    print(f'Conectado por {addr}')
    data = conn.recv(1024).decode() ← Mensagem recebida pelo
                                    socket criado anteriormente
    print(f'Recebido: {data}')
    conn.send(str('Hello Client').encode())
    conn.close()
```

Mensagem recebida pelo
socket criado anteriormente

Servidor Básico Python

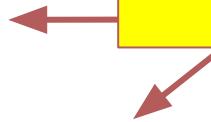
```
import socket

HOST = '127.0.0.1' # Interface padrão localhost / loopback
PORT = 65432        # Porta de escuta (não registrada)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
print(f'Servidor em Execução em {s.getsockname()}')

while True:
    conn, addr = s.accept()
    print(f'Conectado por {addr}')
    data = conn.recv(1024).decode()
    print(f'Recebido: {data}')
    conn.send(str('Hello Client').encode())
    conn.close()
```

Dados enviados e recebidos devem ser codificados (serialização)



Servidor Básico Python

```
import socket

HOST = '127.0.0.1'    # Interface padrão localhost / loopback
PORT = 65432           # Porta de escuta (não registrada)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
print(f'Servidor em Execução em {s.getsockname()}')

while True:
    conn, addr = s.accept()
    print(f'Conectado por {addr}')
    data = conn.recv(1024).decode()
    print(f'Recebido: {data}')
    conn.send(str('Hello Client').encode())
    conn.close() # Fechamento do Socket
```

Cliente Básico Python

```
import socket

# Dados (alvo) para conexão com Servidor
SERVER = ("127.0.0.1", 65432)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    s.connect(SERVER)
    msg = "Hello Server"
    s.send(msg.encode())
    resp = s.recv(1024).decode()
    print(f'Recebido: {resp}')
except Exception as e:
    print(e)
```

Laboratório #07-01

- Lembra-se do **Netcat** (nc)?
 - Utilizado na disciplina de *Administração de Redes*?
- Implemente de uma aplicação similar ao Netcat.

Requisitos

- *Várias trocas de mensagens numa mesma sessão.*
- *Comportamento síncrono (pergunta-resposta).*
- *Aplicação server hospedada na nuvem AWS*
- *Implemente esquema de DDNS na Instância AWS para oferecer **Transparência de Acesso** ao cliente.*
- *Qualquer lado envia ‘exit’ para terminar a sessão.*

Laboratório #07-01

- Ainda no **Lab #07-01**, teste e analise...
- O que acontece com o cliente quando há uma desconexão forçada do server?
Como seria possível contornar esse problema?
- É possível enviar mais de uma informação (p. exemplo: data/hora, autor e mensagem) em um mesmo pacote de dados (único método `send`)? Como?

Laboratório #07-02

- Melhore a implementação da aplicação anterior, permitindo que a troca de mensagens seja **assíncrona**.
 - *Cliente e/ou servidor podem enviar uma ou mais mensagens sem ter que esperar a resposta do outro lado.*
 - *O que acontece quando **dois clientes** distintos tentam se comunicar com o server simultaneamente?*

Laboratório #07-03

- Adapte o Lab anterior para ser ***multi-client***.
 - *Ou seja, o servidor deve permitir a comunicação entre N clientes conectados na aplicação (estilo chat em grupo).*
 - *Utilize dicionário e serialização de objetos para poder enviar o nome e a mensagem de cada usuário do chat.*

Teste e Analise...

- *O que acontece quando um mesmo cliente tenta enviar muitas mensagens ***simultaneamente*** ao server?*
- *Como tratar?*

Referências

- VAN STEEN, Maarten; TANENBAUM, Andrew S. *Distributed systems.*
Leiden, The Netherlands: Maarten van Steen, 2017.
- MENDES, Eduardo. Lives de Python. YouTube Channel.
<https://github.com/dunossauro/live-de-python>
- GUEDES, Dorgival. Notas de aula, UFMG. YouTube Channel:
<https://www.youtube.com/channel/UCJQHsVoqmkygpOXtGfKECFw>