

PTC 3360

2. Introdução às camadas superiores

2.1 A camada de aplicação

(Kurose, Seções 2.1 e 2.2)

Agosto 2023

2. Introdução às camadas superiores

- Neste capítulo do curso, será feita uma introdução a aspectos das 3 camadas superiores da pilha de protocolos Internet (Aplicação, Transporte e Rede).
- A abordagem utilizada é a *top-down*, começando da camada de aplicação, mais próxima dos usuários, em direção à camada física.
- Esses assuntos são detalhados em cursos posteriores das ênfases de Telecomunicações e Computação.

Conteúdo

2.1 A camada de aplicação

2.2 Princípios da transferência confiável de dados (recorte da camada de transporte)

2.3 Camada de rede

Alguns aplicativos de rede

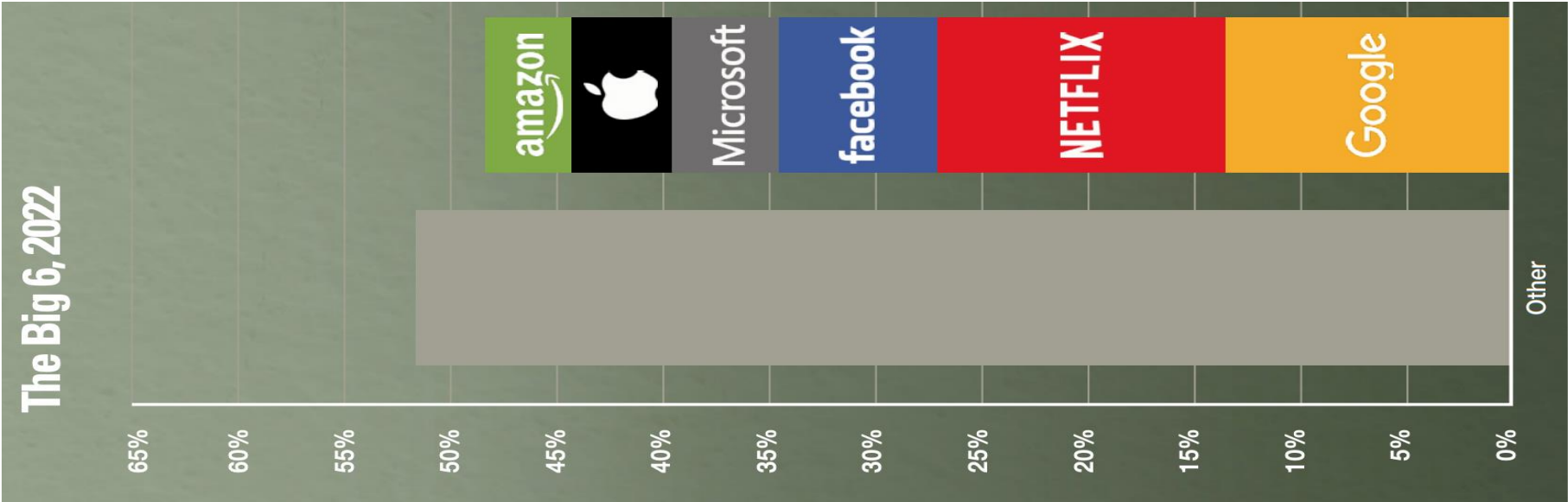
- ❖ *E-mail*
- ❖ *World Wide Web (WWW)*
- ❖ *Whatsapp*
- ❖ *Waze, Google Maps, Apple Plans*
- ❖ *BitTorrent*
- ❖ *Jogos em rede multiusuários*
- ❖ *Streaming de vídeo armazenado (YouTube, Disney+, Netflix)*
- ❖ *Voz-sobre-IP (e.g., Skype)*
- ❖ *Videoconferência em tempo real (Zoom, Google Meet)*
- ❖ *Redes sociais*
- ❖ *Busca*
- ❖ *...*

Tráfego Global - 2023

APP CATEGORY TOTAL VOLUME			TOP APPS 2022			TOP APPS 2022		
2022 Categories		Total Volume	DOWNSTREAM TRAFFIC			UPSTREAM TRAFFIC		
1	Video	65.93%	Application		Total Volume	Application		Total Volume
2	Marketplace	5.83%	1	Netflix	14.93%	1	Netflix	8.78%
3	Gaming	5.58%	2	YouTube	11.62%	2	HTTP Media Stream	6.89%
4	Social Networking	5.26%	3	Generic QUIC	5.88%	3	YouTube	5.90%
5	Cloud	4.98%	4	Disney+	4.49%	4	Generic Web Browsing	3.85%
6	Web Browsing	4.63%	5	TikTok	3.93%	5	HTTP download	3.70%
7	File Sharing	3.39%	6	HTTP Media Stream	3.72%	6	Generic QUIC	3.43%
8	Messaging	2.30%	7	Playstation Downloads	2.95%	7	Generic Messaging	3.17%
9	VPN	1.13%	8	Xbox Live	2.91%	8	Disney+	2.98%
10	Audio	0.95%	9	Facebook	2.87%	9	Hulu	2.79%
			10	Amazon Prime	2.83%	10	Facebook	2.67%

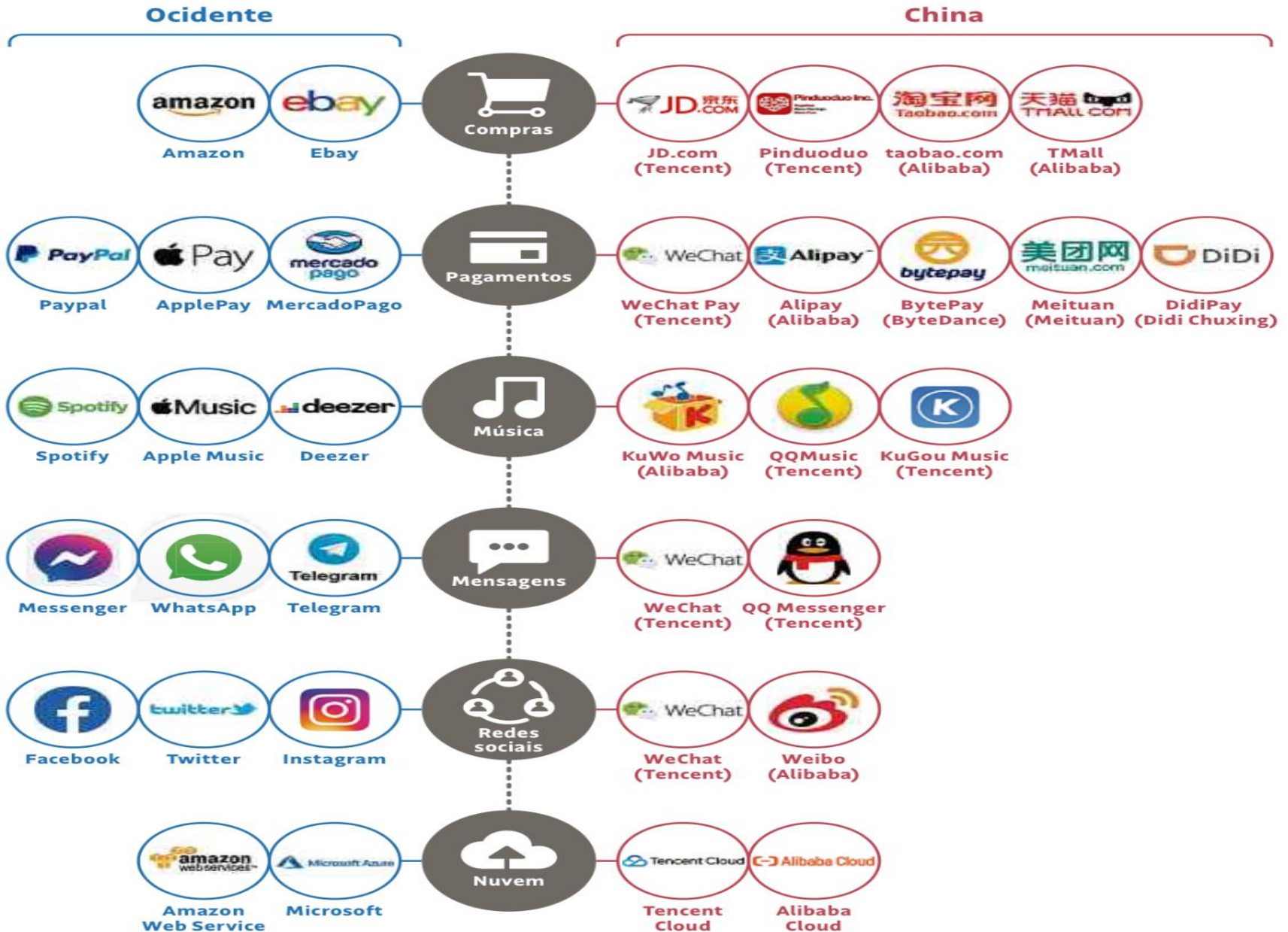
Tráfego Global - 2023

	Video	Games	Social	Messaging
1	Netflix	Playstation Downloads	Facebook	Generic Messaging
2	YouTube	Steam	Twitch	WhatsApp
3	Generic QUIC	ROBLOX	Instagram	Facebook
4	HTTP Media Stream	Epic Games Launcher	Snapchat	Discord Voice
5	Disney+	Nintendo Online	Reddit	Wattpad
6	Tik Tok	Xbox Live TLS	Wordpress	Telegram
7	Amazon Prime	Steam Client	Pinterest	Discord
8	Hulu	Kayo Sports	Twitter	Microsoft Teams
9	Facebook Video	Generic Gaming	VK	WeChat
10	Operator Content	League of Legends	LinkedIn	LINE



Enquanto isso, na China...

Como é o ecossistema das big techs chinesas

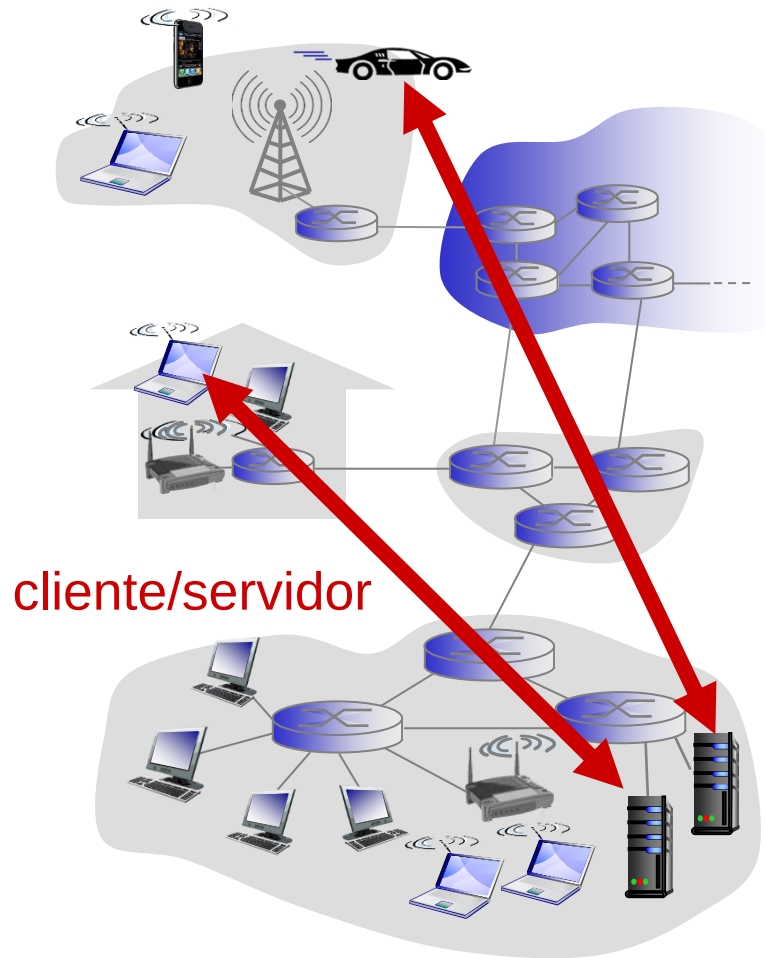


Arquiteturas de aplicativos

Possíveis estruturas de aplicativos:

- ❖ A. Cliente-servidor
- ❖ B. *Peer-to-peer* (P2P)

A. Arquitetura cliente-servidor



Servidor:

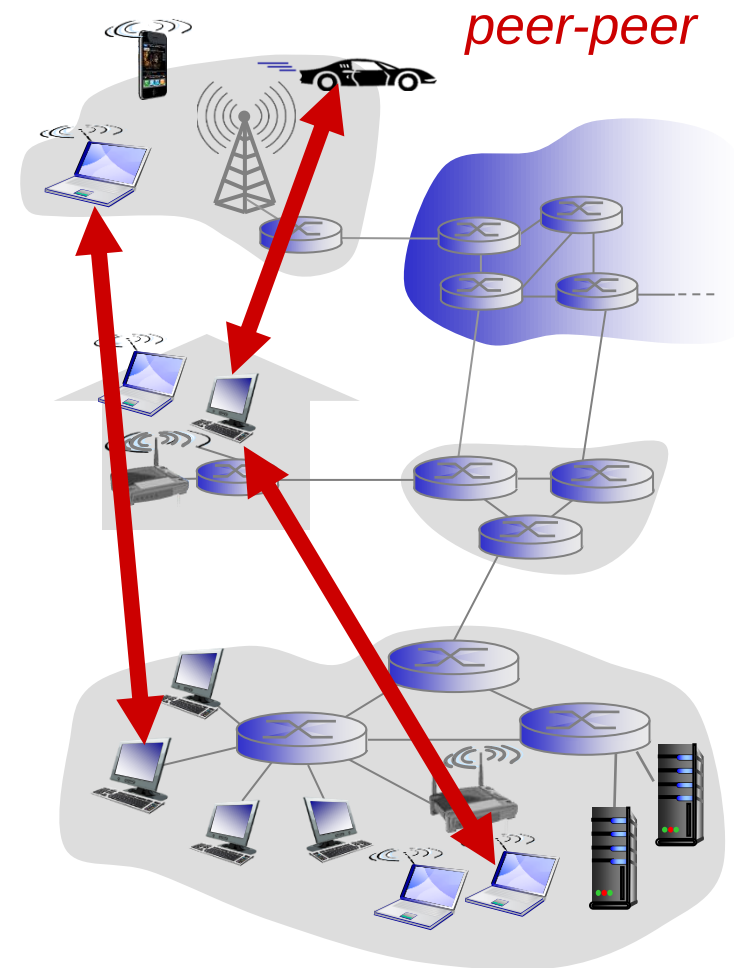
- ❖ em *host* sempre ligado
- ❖ endereço IP permanente
- ❖ *data centers* pensando em escala (servidor virtual)

Clientes:

- ❖ comunicam-se com servidor e não diretamente entre si
 - ❖ podem se conectar de forma intermitente
 - ❖ podem ter endereços IP dinâmicos
- Web, E-mail, *Streaming* de vídeo, ...

B. Arquitetura P2P

- ❖ Dependência mínima de servidores dedicados
- ❖ Comunicação direta entre sistemas finais (*peers*)
- ❖ *Peers* requerem serviço de outros *peers* – provêm serviço para outros *peers* em retorno
 - *Autoescalável* – novos *peers* trazem novas demandas de serviço, mas também fornecem serviço
- ❖ *Peers* conectam-se intermitentemente e mudam endereço IP
 - gerenciamento mais complexo
- ❖ Exemplo: BitTorrent, Bitcoin.

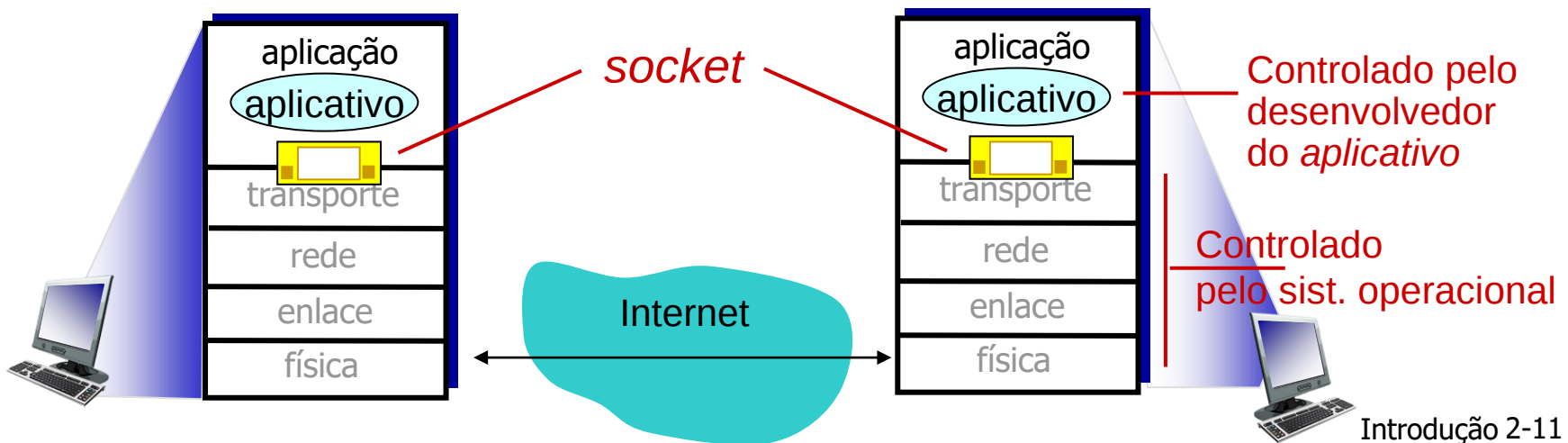


Discussão:

Whatsapp é cliente-servidor ou P2P? O que motiva essa escolha de arquitetura?

Sockets

- ❖ Aplicativo envia/recebe mensagens de/para uma interface de software (API - Application Programming Interface) : **socket**
- ❖ Aplicativo transmissor envia mensagem pela porta
- ❖ Ele confia na infraestrutura de transporte do outro lado da porta para entregar mensagem ao socket no aplicativo receptor
- ❖ Aplicativo receptor lê dados do socket



Endereçamento de processos

- ❖ Para receber mensagens, processo precisa ter *identificador*
- ❖ Dispositivo *host* tem endereço IP de 32 bits único (IPv4)
- ❖ Endereço IP do *host* em que o processo roda é suficiente para identificar o processo?
 - Resposta: Não! Muitos processos podem estar rodando no mesmo *host*!
- ❖ *Identificador* inclui tanto **endereço IP** (32 bits) quanto **número da porta** associado com processo no *host*.
- ❖ Exemplos de números de porta:
 - Servidor web (HTTP): 80
 - Servidor web (HTTPS): 443
 - Servidor de e-mail (SMTP): 25
 - Gerenciados pela *Internet Assigned Numbers Authority* (IANA)
 - <http://www.iana.org>
- ❖ Para enviar mensagem HTTP para servidor web www.usp.br:
 - **Endereço IP**: 200.144.248.41
 - **Número de porta**: 80
- ❖ Mais em breve...

Protocolo da camada de aplicação define

❖ Tipos de mensagens trocadas

- Por exemplo, requisição, resposta

❖ Sintaxe da mensagem

- Campos da mensagem e como eles são delineados

❖ Semântica das mensagens

- Significado das informações dos campos

❖ Regras para quando e como aplicativos enviam e respondem mensagens

Protocolos abertos:

- ❖ Definidos em RFCs
- ❖ Permitem interoperabilidade
- ❖ Exemplos:
 - HTTP [[RFC 9110](#)]
 - SMTP [[RFC 5321](#)]

Protocolos proprietários:

- ❖ Por exemplo, Skype

Exemplo de Aplicação: Web e HTTP I.I

Primeiro, alguns conceitos básicos...

- ❖ Uma *página web* consiste de *objetos (arquivos)*.
- ❖ Um objeto pode ser um arquivo HTML, uma imagem JPEG, um arquivo de áudio, um vídeo, etc.
- ❖ Uma página web consiste de “*programa*” *HTML base* que inclui *diversos objetos referenciados*.
- ❖ Cada objeto é endereçável por uma *URL (Uniform Resource Locator)*, por exemplo,

`www.lcs.poli.usp.br/~marcio/index_arquivos/image002.jpg`

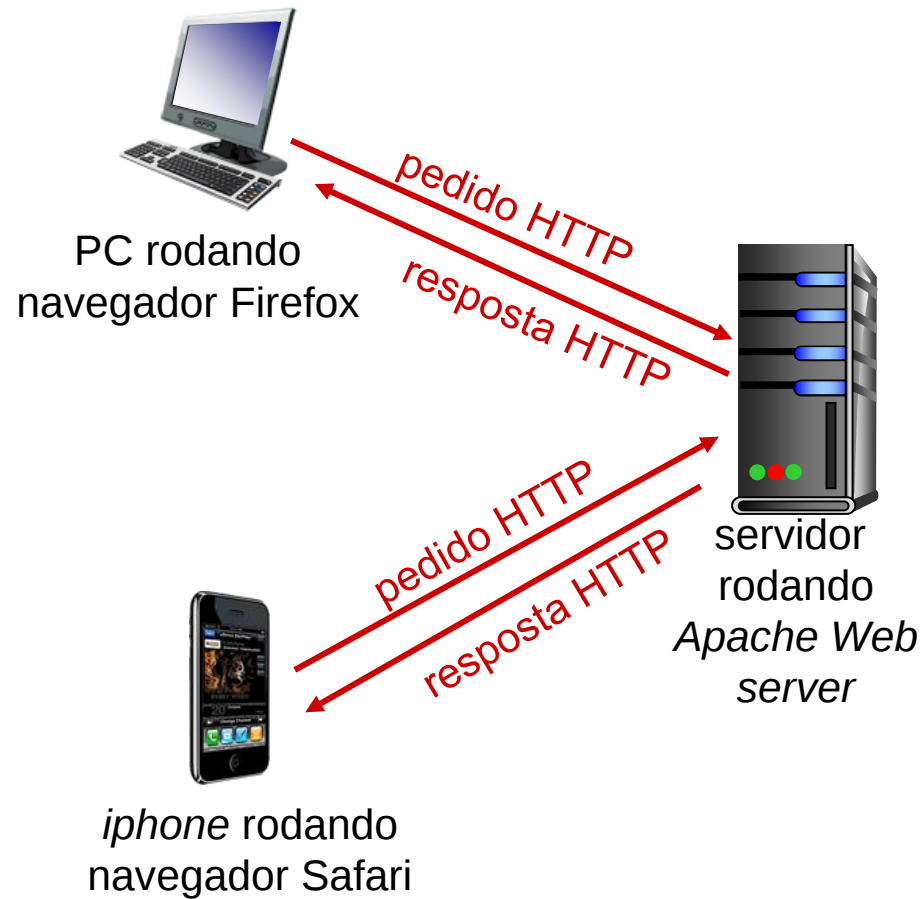
nome do *host*

local do objeto

Visão geral do HTTP

HTTP: *HyperText Transfer Protocol*

- ❖ HTTP 1.0 ([RFC 1945](#) - 1996)
 - ❖ HTTP 1.1 ([RFC 2068](#) - 1997)
 - ❖ HTTP/2 ([RFC 7540](#) - 2015)
 - ❖ HTTP/3 ([RFC 9114](#) - junho/2022)
-
- ❖ **Protocolo associado à aplicação World Wide Web**
 - ❖ **Modelo cliente/servidor**
 - **Cliente:** navegador que pede, recebe e apresenta objetos Web (Microsoft Edge, Firefox, Chrome)
 - **Servidor:** servidor Web envia objetos em resposta a requisições (Apache, Microsoft Internet Information Server)



Visão geral do HTTP 1.1 (continuação)

Usa TCP como protocolo da camada de transporte:

- ❖ Cliente inicia **conexão TCP** (cria socket) para o servidor, porta 80
- ❖ Servidor aceita conexão TCP do cliente
- ❖ Mensagens HTTP trocadas entre navegador (**cliente HTTP**) e servidor Web (**servidor HTTP**)
- ❖ Conexão TCP fechada

HTTP é “sem memória”

- ❖ Servidor não mantém informação sobre pedidos anteriores do cliente

nota

Protocolos que mantêm “memória” são complexos!

- ❖ História passada (estado) precisa ser mantido
- ❖ Se cliente/servidor cai, suas visões do “estado” podem ser inconsistentes e precisam ser reconciliadas

Tipos de Conexões HTTP 1.1

A. *HTTP não persistente*

- ❖ No máximo um objeto enviado sobre uma conexão TCP
 - conexão então fechada
- ❖ Fazer *download* de múltiplos objetos requer múltiplas conexões

B. *HTTP persistente*

- ❖ Múltiplos objetos podem ser enviados sobre única conexão TCP entre cliente, servidor
- ❖ Padrão para HTTP/1.1

A. HTTP não persistente

Suponha que usuário digita URL:

<http://www.lcs.poli.usp.br/contato.html>

(contém texto e referências a 10 imagens jpeg)

1a. HTTP cliente inicia conexão TCP ao (aplicativo) servidor HTTP em `www.lcs.poli.usp.br` na porta 80

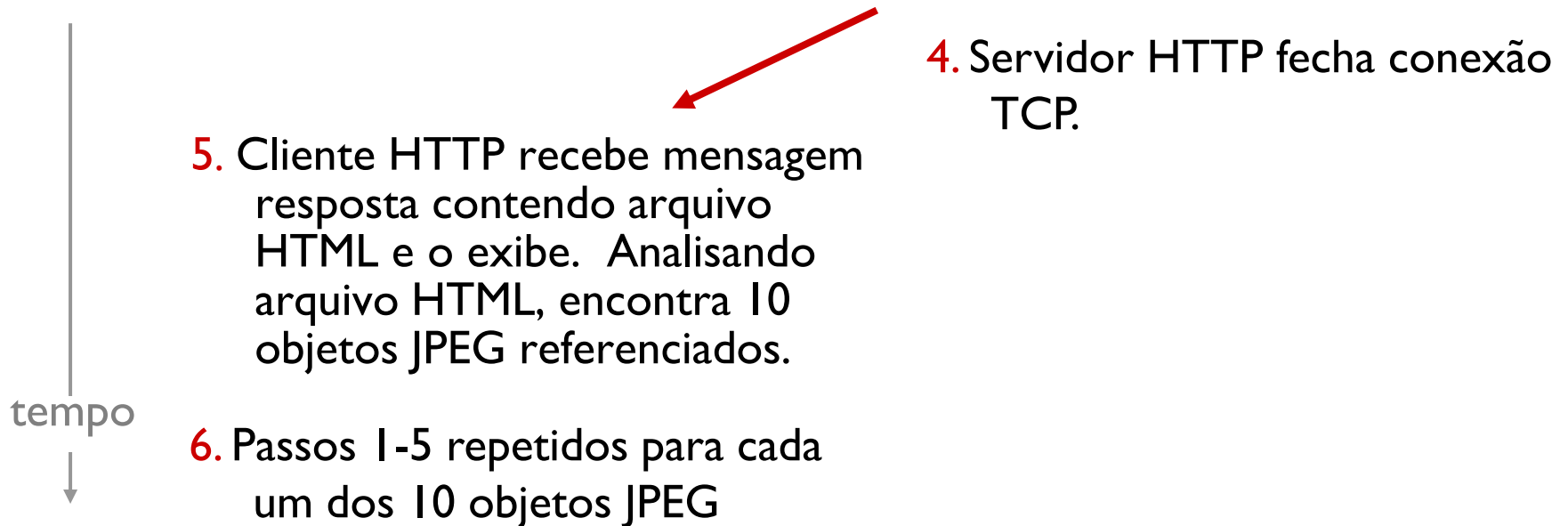
1b. Servidor HTTP no *host* `www.lcs.poli.usp.br` espera por conexão TCP na porta 80. Aceita conexão, notificando cliente

2. Cliente HTTP envia *mensagem pedido* HTTP (contendo URL) para o *socket* de conexão TCP. Mensagem indica que o cliente quer objeto `/contato.html`

3. Servidor HTTP recebe mensagem pedido, forma *mensagem resposta* contendo objeto solicitado, e envia mensagem pelo seu *socket*

tempo
↓

A. HTTP não persistente (cont.)



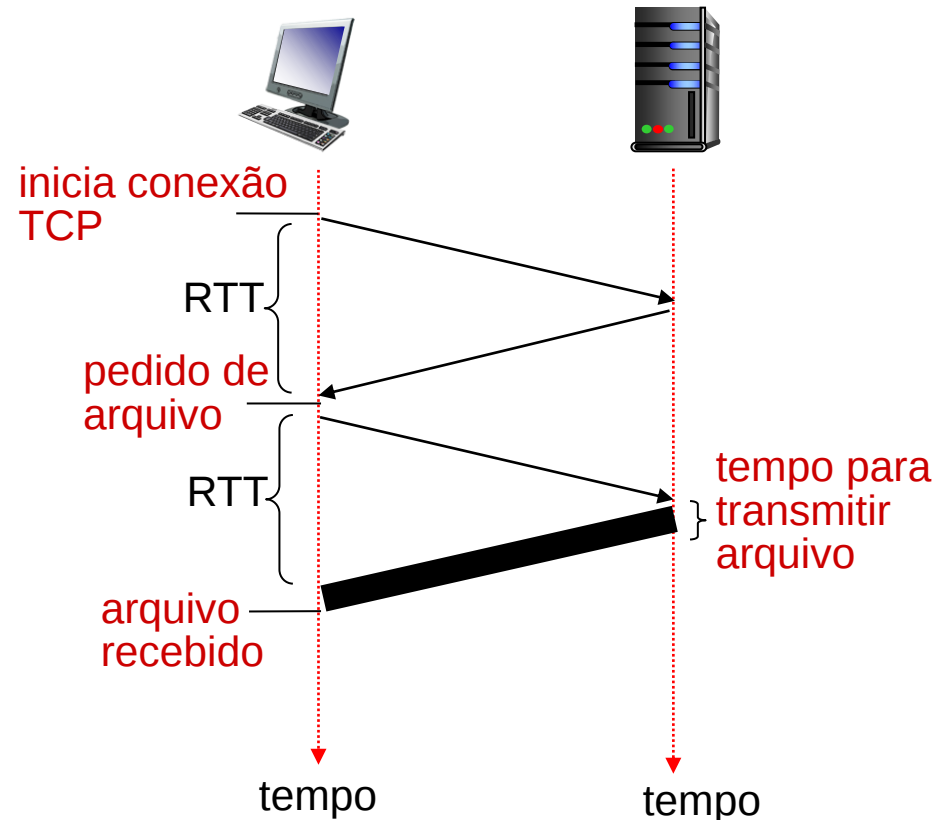
A. HTTP não persistente: tempo de resposta

RTT (Round-Trip Time) : tempo para pequeno pacote viajar do cliente ao servidor e voltar

tempo de resposta HTTP:

- ❖ 1 RTT para iniciar conexão TCP
- ❖ 1 RTT para pedido HTTP e primeiros bytes da resposta HTTP retornar
- ❖ Tempo de transmissão do arquivo
- ❖ Tempo de resposta para HTTP não persistente =

2RTT + tempo de transmissão do arquivo



B. HTTP Persistente

Problemas do HTTP não persistente :

- ❖ Requer 2 RTTs por objeto, aumentando a latência do sistema

HTTP persistente:

- ❖ Servidor deixa conexão aberta depois de enviar resposta
- ❖ Mensagens HTTP subsequentes entre mesmo cliente/servidor enviadas sobre a conexão aberta
- ❖ Cliente envia pedido assim que encontra objeto referenciado
- ❖ Perto de 1 RTT para todos os objetos referenciados

Mensagem pedido HTTP 1.1

- ❖ 2 tipos de mensagens HTTP: *pedido (request)*, *resposta*
- ❖ **Mensagem pedido HTTP:**
 - ASCII (formato que permite leitura por humanos)

linha de requisição
(comandos

GET, POST, HEAD,...)

linhas de
cabeçalho (opcionais)

http://en.wikipedia.org/wiki/List_of_HTTP_header_fields

*carriage return,
line feed* no início
de linha indica

fim de linhas de cabeçalho

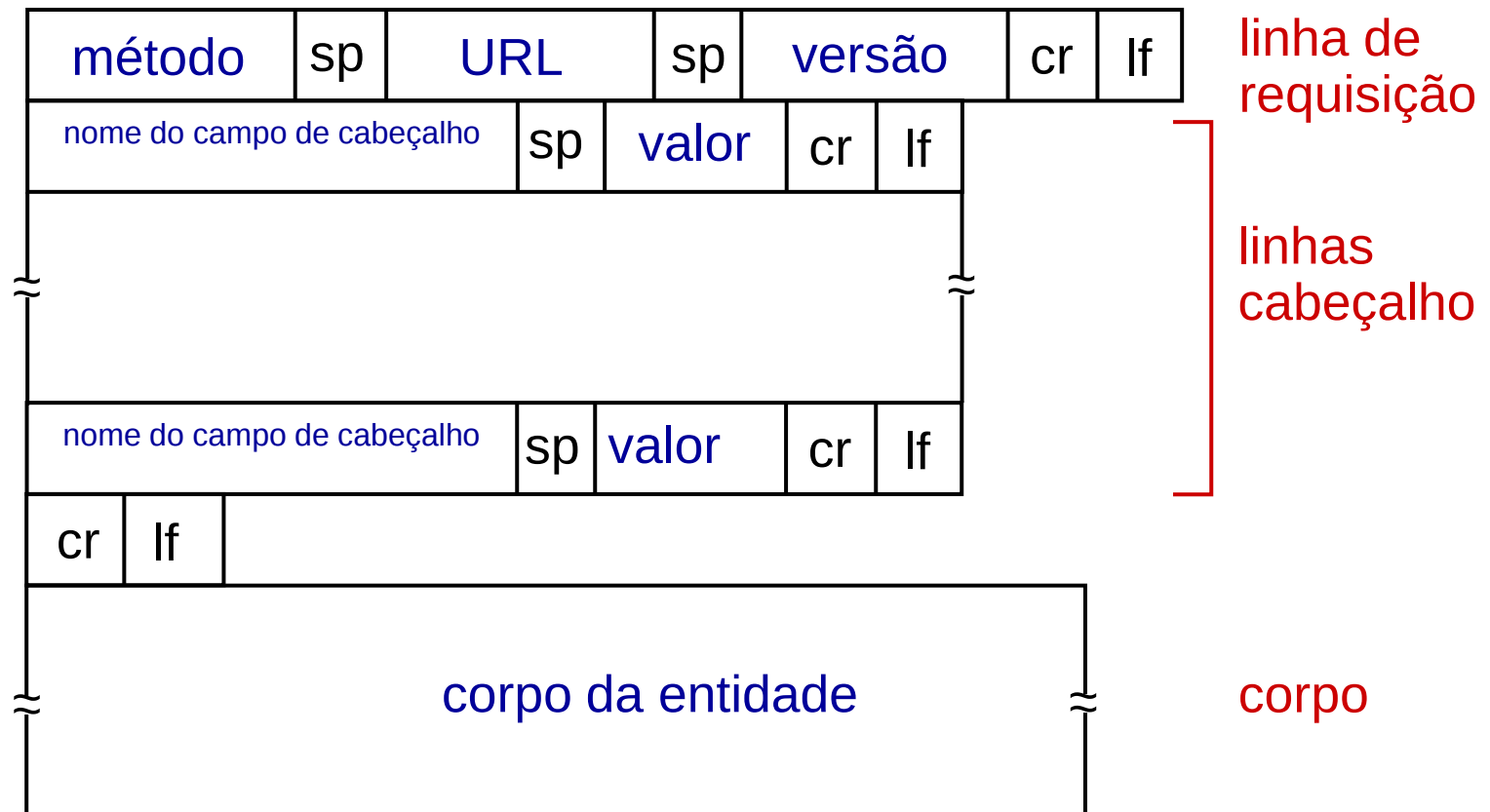
```
GET /~marcio/index.htm HTTP/1.1\r\n
Host: www.lcs.poli.usp.br\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: pt-br,en-us;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

caractere *carriage return* ASCII 13

caractere *line-feed* ASCII 10

close para conexão não persistente

Mensagem pedido HTTP 1.1: formato geral



Obs.: sp=caracter de espaço; cr=carriage return; lf=line feed

Mensagem resposta HTTP 1.1

linha de estado
(código e frase
de estado do
protocolo)

linhas
de
cabeçalho

dados, e.g.,
arquivo HTML
requisitado

```
HTTP/1.1 200 OK\r\n
Date: Tue, 25 Feb 2014 18:24:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 18 Feb 2014 17:00:02
      GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
      1\r\n
\r\n
data data data data data ...
```

Códigos de estado da resposta HTTP

- ❖ Código de estado aparece na 1a linha da mensagem resposta servidor-cliente
- ❖ Alguns códigos exemplos:

200 OK

- Atendido com sucesso, objeto pedido mais para frente na msg

301 Moved Permanently

- Objeto pedido foi movido, nova localização especificada mais a frente nessa msg (Location:)

400 Bad Request

- Mensagem pedido não entendida pelo servidor

404 Not Found

- Documento pedido não encontrado nesse servidor

505 HTTP Version Not Supported

Experimentando o HTTP 1.1 (lado cliente)

Usando o Wireshark

1. Abra o navegador
2. Abra o Wireshark e inicialize a varredura de pacotes no enlace usado para acesso à rede (e.g., WiFi)
3. Acesse pelo navegador o endereço <http://nginx.org/>
4. Siga (*follow* no Wireshark) a troca HTTP para este pedido

Experimentando o HTTP 1.1 (lado cliente)

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http

No.	Time	Source	Destination	Protocol	Length	Info
223	2.442385	192.168.0.187	142.250.218.14	HTTP	166	GET /generate_204 HTTP/1.1
225	2.455666	142.250.218.14	192.168.0.187	HTTP	137	HTTP/1.1 204 No Content
526	7.292843	192.168.0.187	104.122.253.121	HTTP	267	GET /pt-BR/livetile/preinstall?region=BR&appid=...
530	7.301091	104.122.253.121	192.168.0.187	HTTP/X...	337	HTTP/1.1 200 OK
1237	20.298822	192.168.0.187	3.125.197.172	HTTP	520	GET / HTTP/1.1
1249	20.522593	3.125.197.172	192.168.0.187	HTTP	626	HTTP/1.1 200 OK (text/html)
1254	20.559793	192.168.0.187	3.125.197.172	HTTP	456	GET /nginx.png HTTP/1.1
1295	20.795418	3.125.197.172	192.168.0.187	HTTP	981	HTTP/1.1 200 OK (PNG)
1362	20.850452	192.168.0.187	3.125.197.172	HTTP	458	GET /favicon.ico HTTP/1.1
1449	21.062083	192.168.0.187	13.227.113.74	HTTP	429	GET /js/build/bf-munchkin.min.js HTTP/1.1
1457	21.069289	13.227.113.74	192.168.0.187	HTTP	671	HTTP/1.1 301 Moved Permanently (text/html)
1549	21.298532	3.125.197.172	192.168.0.187	HTTP	1329	HTTP/1.1 200 OK (image/x-icon)

> Frame 1237: 520 bytes on wire (4160 bits), 520 bytes captured (4160 bits) on interface \Device\NPF_{64163EDF-21F6-4590-AD06...}

> Ethernet II, Src: IntelCor_bc:74:5b (5c:cd:5b:bc:74:5b), Dst: Tp-LinkT_ab:e4:be (68:ff:7b:ab:e4:be)

> Internet Protocol Version 4, Src: 192.168.0.187, Dst: 3.125.197.172

> Transmission Control Protocol, Src Port: 53345, Dst Port: 80, Seq: 1, Ack: 1, Len: 466

> Hypertext Transfer Protocol

> GET / HTTP/1.1\r\n

Host: nginx.org\r\n

Connection: keep-alive\r\n

DNT: 1\r\n

Upgrade-Insecure-Requests: 1\r\n

0000 68 ff 7b ab e4 be 5c cd 5b bc 74 5b 08 00 45 00 h-{\...\ [t[...E-

0010 01 fa 0e 49 40 00 80 06 00 00 c0 a8 00 bb 03 7d ...I@...

0020 c5 ac d0 61 00 50 46 a8 e4 a7 a3 a5 02 69 50 18 ...a PF- ...iP-

0030 01 02 8c 79 00 00 47 45 54 20 2f 20 48 54 54 50 ...y...GE T / HTTP

0040 2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 6e 67 69 6e /1.1..Ho st: ngin

0050 78 2e 6f 72 67 0d 0a 43 6f 6e 6e 65 63 74 69 6f x.org..C onnectio

0060 6e 3a 20 6b 65 65 70 2d 61 6c 69 76 65 0d 0a 44 n: keep- alive..D

0070 4e 54 3a 20 31 0d 0a 55 70 67 72 61 64 65 2d 49 NT: 1..U pgrade-I

0080 6e 73 65 63 75 72 65 2d 52 65 71 75 65 73 74 73 nsecure- Requests

0090 3a 20 31 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a : 1..Use r-Agent:

00a0 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28 57 69 Mozilla /5.0 (Wi

00b0 6e 64 6f 77 73 20 4e 54 20 31 30 2e 30 3b 20 57 ndows NT 10.0; W

00c0 69 6e 36 34 3b 20 78 36 34 29 20 41 70 70 6c 65 in64; x6 4.0) Apple

00d0 57 65 62 4b 69 74 2f 35 33 37 2e 33 36 20 28 4b WebKit/5 37.36 (K

Wireshark · Follow HTTP Stream (tcp.stream eq 52) · Wi-Fi

GET / HTTP/1.1

Host: nginx.org

Connection: keep-alive

DNT: 1

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

Accept-Encoding: gzip, deflate

Accept-Language: fr-FR,fr;q=0.9,pt;q=0.8,en-US;q=0.7,en;q=0.6

HTTP/1.1 200 OK

Server: nginx/1.19.10

Date: Fri, 20 Aug 2021 18:53:42 GMT

Content-Type: text/html; charset=utf-8

Content-Length: 3171

Last-Modified: Fri, 20 Aug 2021 05:28:43 GMT

Connection: keep-alive

Keep-Alive: timeout=15

ETag: "611f3d8b-c63"

Content-Encoding: gzip

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html><head><meta http-equiv="Content-Type" content="text/html; charset=utf-8"><link rel="alternate" type="application/rss+xml" title="nginx news" href="http://nginx.org/index.rss"><title>nginx news</title><style type="text/css">body { background: white; color: black; font-family: sans-serif; line-height: 1.4em; text-align: center; margin: 0; padding: 0; } #banner { background: black; color: #F2F2F2; line-height: 1.2em; padding: .3em 0; box-shadow: 0 5px 10px black; } #banner a { color: #00B140; } #main {

3 client pkt(s), 3 server pkt(s), 5 turn(s).

Entire conversation (30kB)

Show data as ASCII

Find:

Filter Out This Stream Print Save as... Back Close Help

Observações - HTTPS

- O HTTPS (*Hyper Text Transfer Protocol Secure* - protocolo de transferência de hipertexto seguro) é uma implementação do protocolo HTTP que tem se tornado o padrão na web.
- Possui uma camada adicional de segurança que utiliza o [protocolo SSL/TLS](#).
- Essa camada adicional permite que os dados sejam transmitidos por meio de uma conexão criptografada e que se verifique a autenticidade do servidor e do cliente por meio de certificados digitais. Passou a ser o padrão a partir do HTTP/2.
- A porta usada para o protocolo HTTPS é a 443.

Observações - HTTP/3 e QUIC

- Diferentemente das versões anteriores, o HTTP/3 não é baseado no TCP mas sim no [QUIC \(RFC 9000\)](#), desenvolvido inicialmente pelo Google.
- Em 2022, cerca de 25% do tráfego HTTP já é realizado sobre o QUIC. A principal vantagem é a diminuição significativa do tempo de resposta.
- Nesta disciplina, tomamos por base a versão 1.1 do HTTP por questões didáticas. Dessa forma, fica mais fácil se concentrar nos princípios da camada de aplicação, deixando o problema da comunicação confiável para a camada de transporte que será vista a partir da próxima aula.
- Um vídeo inicial sobre as diferenças do HTTP/3 em relação às versões anteriores pode ser visto [aqui](#).
- Mais detalhes sobre o HTTP/3 e o QUIC são deixados para cursos mais avançados.

(Kurose, p. 125) Considere o seguinte *string* de caracteres ASCII que foram capturados pelo *Wireshark* quando o navegador enviou uma mensagem HTTP GET. Os caracteres `<cr>` `<lf>` são caracteres *carriage return* e *line-feed*. Responda as seguintes questões, indicando onde na mensagem HTTP GET abaixo você encontra a sua resposta.

```
GET /cs453/index.html HTTP/1.1<cr><lf>Host: gai
a.cs.umass.edu<cr><lf>User-Agent: Mozilla/5.0 (
Windows;U; Windows NT 5.1; en-US; rv:1.7.2) Gec
ko/20040804 Netscape/7.2 (ax) <cr><lf>Accept:ex
t/xml, application/xml, application/xhtml+xml, text
/html;q=0.9, text/plain;q=0.8,image/png,*/*;q=0.5
<cr><lf>Accept-Language: en-us,en;q=0.5<cr><lf>Accept-
Encoding: zip,deflate<cr><lf>Accept-Charset: ISO
-8859-1,utf-8;q=0.7,*;q=0.7<cr><lf>Keep-Alive: 300<cr>
<lf>Connection:keep-alive<cr><lf><cr><lf>
```

- (a) Qual o URL do documento requisitado pelo navegador?
- (b) Qual a versão de HTTP o navegador está rodando?
- (c) O navegador requisitou uma conexão persistente ou não persistente?
- (d) Qual é o endereço IP do *host* no qual o navegador está rodando?
- (e) Que tipo de navegador iniciou a mensagem? Por que é necessário o tipo de navegador numa mensagem de pedido HTTP?