



# 90293 - Tecnologias de Redes de Computadores

Aula 13 – HTTP

*Pedro Gonçalves – pasg@ua.pt*

# Sumário

- Funcionalidades da camada de aplicação. Exemplos de aplicações de rede e identificação dos protocolos da camada de aplicação utilizados.
- World Wide Web: protocolo HTTP
  - Apresentação geral do HTTP.
  - Ligações persistentes e não persistentes.
  - Formato da dos pacotes HTTP.
  - Métodos de acesso HTTP GET e POST.
  - Clientes WWW.



## Proxies HTTP

- Motivação
- Princípio de funcionamento

# Algumas aplicações de rede

- social networking
- Web
- text messaging
- e-mail
- Jogos de rede
- streaming video (YouTube, Hulu, Netflix)
- file sharing P2P
- voice over IP (e.g., Skype)
- real-time video conferencing (e.g., Zoom)
- Pesquisas na Internet
- remote login
- ...



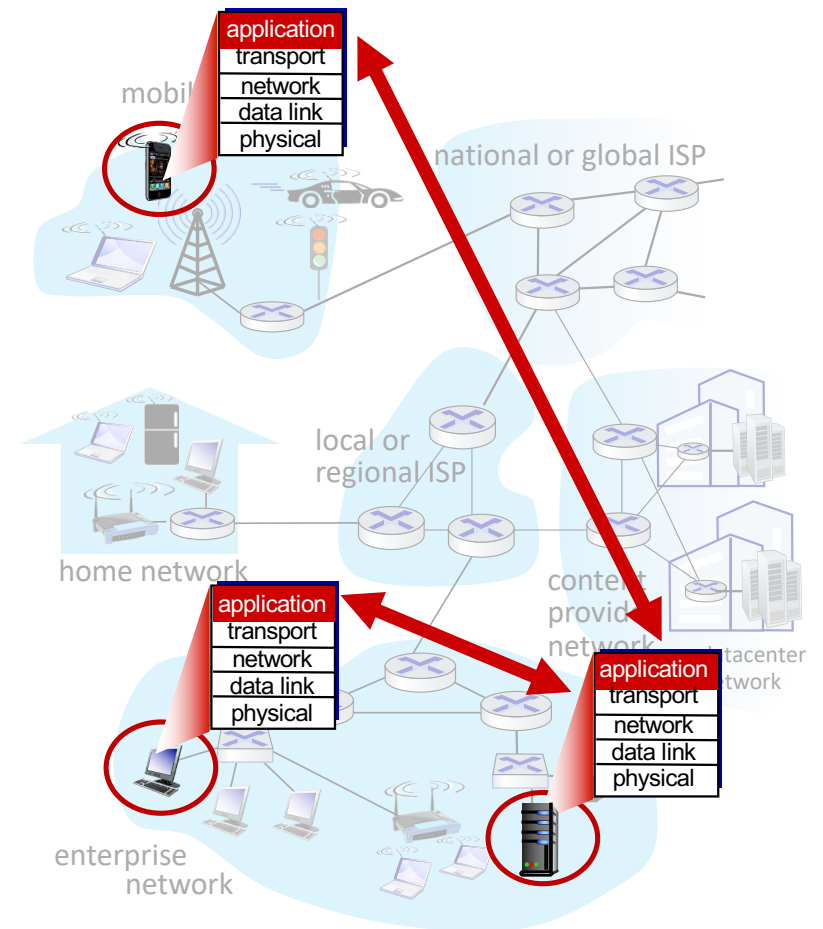
# Criação aplicações de rede

## programas que:

- Correm em diferentes OSs
- comuniquem over sobre a rede
  - e.g., aplicação web server comunica com aplicação browser

## Sem necessidade de escrever software para dispositivos de rede

- Dispositivos de rede não correm aplicações dos utilizadores



# Paradigma Cliente-servidor

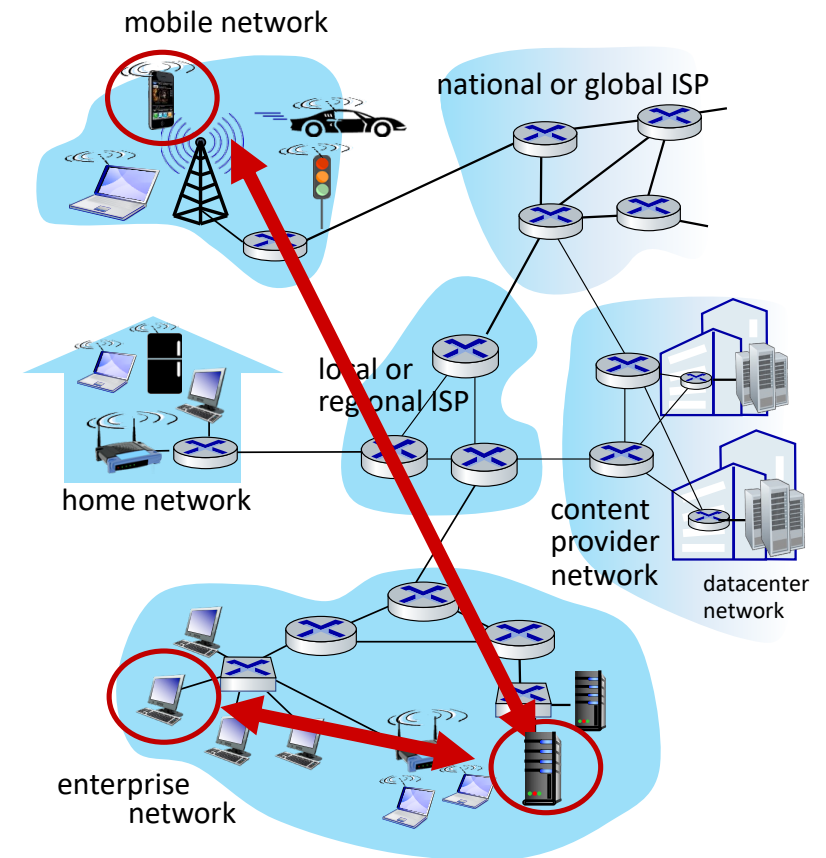
- server:

- Host sempre ligado
- Endereço IP permanente
- Normalmente nos data centers, para poder ser **escalado**

- clients:

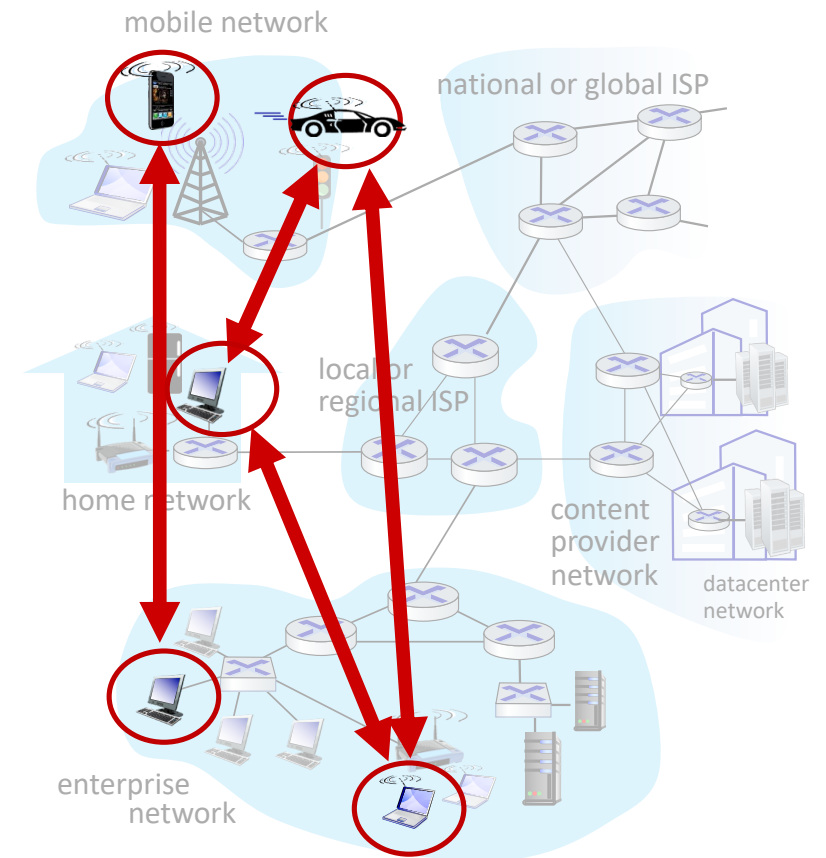
- contacta, comunica com o server
- Pode estar intermitentemente ligado
- Pode ter endereço IP atribuído dinamicamente
- Não comunicam directamente entre si

- examples: HTTP, IMAP, FTP



# Arquitetura *peer-peer*

- Sem host sempre ligado
- Arbitrariamente os terminais comunicam entre si
- *peers* pedem serviço a outros *peers*, fornecem serviço a outros *peers*
  - Auto escalável – novos peers trazem nova capacidade de serviço, como novos requisitos de serviço
- Peers são ligados intermitentemente mudam de endereços IP
  - Gestão complexa
- example: P2P file sharing



# Processos comunicantes

*processo*: programa que corre num host

- No mesmo host, dois processos comunicam usando **inter-process communication** (definido pelo OS)
- processos em diferentes hosts comunicam enviando **mensagens**



clients, servers

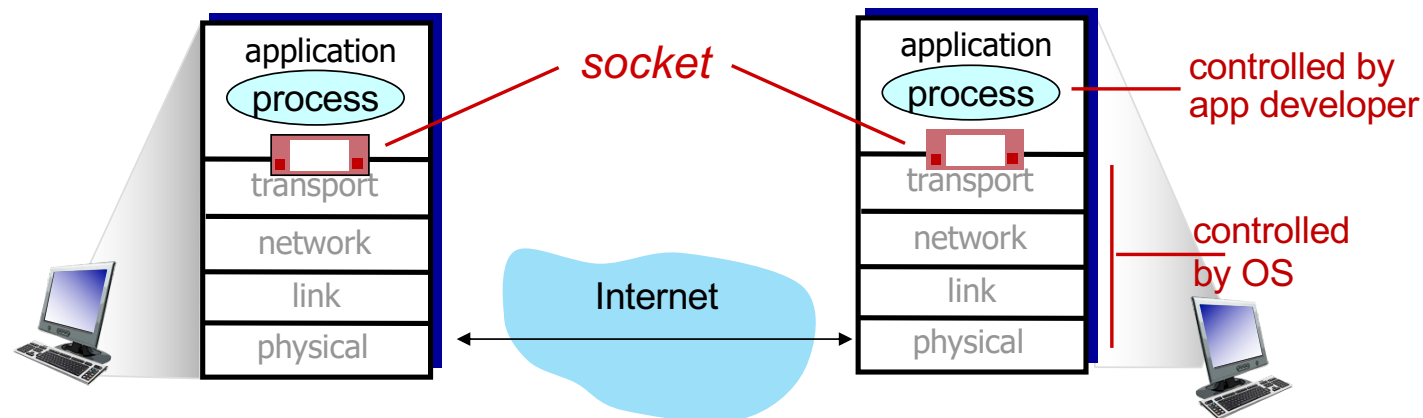
*client process*: processo que inicia comunicação

*server process*: processo que espera para ser contactado

- nota: aplicações com arquitecturas P2P têm processos clientes & servidores

# Sockets

- processos enviam/recebem mensagens de/para o seu **socket**
- socket é com uma porta
  - processo emissor empurra porta
  - o processo de envio depende da infraestrutura de transporte do outro lado da porta para entregar a mensagem ao socket no processo de recepção
  - 2 sockets envolvidos: um de cada lado





# Endereçamento de processos

- para receber mensagens, o processo deve ter identificador
- dispositivo host tem endereço IP exclusivo de 32 bits
- P: o endereço IP do host em que o processo é executado é suficiente para identificar o processo?
- A: não, muitos processos podem ser executados no mesmo host
- *O identificador* inclui o *endereço IP e os números* de porta associados ao processo no host.
- exemplos de números de porta:
  - Servidor HTTP: 80
  - servidor de correio: 25
- para enviar mensagem HTTP para o servidor da web gaia.cs.umass.edu:
  - Endereço IP: 128.119.245.12
  - número da porta: 80
- *mais em breve ...*



# Protocolos da camada de aplicação definem:

- **Tipos de mensagens trocadas,**
  - e.g., pedido, resposta
- **Sintaxe das mensagens:**
  - Quais os campos que existem nas mensagens e como são colocados
- **Semântica das mensagens**
  - Significados de cada campo
- **rules** para quem e quando cada um dos processos enviam mensagens e para quem respondem



## Protocolos abertos:

- Definidos em RFCs, toda a gente tem acesso À definição do protocol
- permite interoperabilidade
- e.g., HTTP, SMTP

## Protocolos proprietários :

- e.g., Skype, Zoom

# Que serviço de transporte um aplicativo precisa?

## data integrity

- algumas aplicações (por exemplo, transferência de arquivos, transações da web) exigem transferência de dados 100% confiável
- outras (por exemplo, áudio) podem tolerar alguma perda



## timing

- algumas (por exemplo, telefonia pela Internet, jogos interativos) requerem baixo atraso para serem "eficazes"

## throughput

- algumas aplicações (por exemplo, multimédia) exigem uma quantidade mínima de taxa de transferência para serem "eficazes"
- outros aplicativos ("aplicativos elásticos") fazem uso de qualquer taxa de transferência que obtêm

## security

- criptografia, integridade de dados, ...

# Requisitos de serviço de transporte: aplicativos comuns

application	data loss	throughput	time sensitive?
-------------	-----------	------------	-----------------

file transfer/download	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5Kbps-1Mbps video:10Kbps-5Mbps	yes, 10's msec
streaming audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	Kbps+	yes, 10's msec
text messaging	no loss	elastic	yes and no

# Serviços de protocolos de transporte da Internet

## *TCP service:*

- *transporte confiável* entre o processo de envio e recebimento
- *controle de fluxo*: o remetente não sobrecarrega o receptor
- *controle de congestionamento*: acelerador do remetente quando a rede estiver sobrecarregada
- *orientado para conexão*: configuração necessária entre os processos do cliente e do servidor
- *não fornece*: tempo, garantia de rendimento mínimo, segurança



## *UDP service:*

- *transferência de dados não confiável* entre o processo de envio e recebimento
- *não fornece*: confiabilidade, controle de fluxo, controle de congestionamento, tempo, garantia de taxa de transferência, segurança ou configuração de conexão.

Q: *Porquê pensar em UDP? Para quê?*

# Internet applications, and transport protocols

application		
application	application layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP 1.1 [RFC 7320]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or	TCP or UDP
streaming audio/video	proprietary HTTP [RFC 7320], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

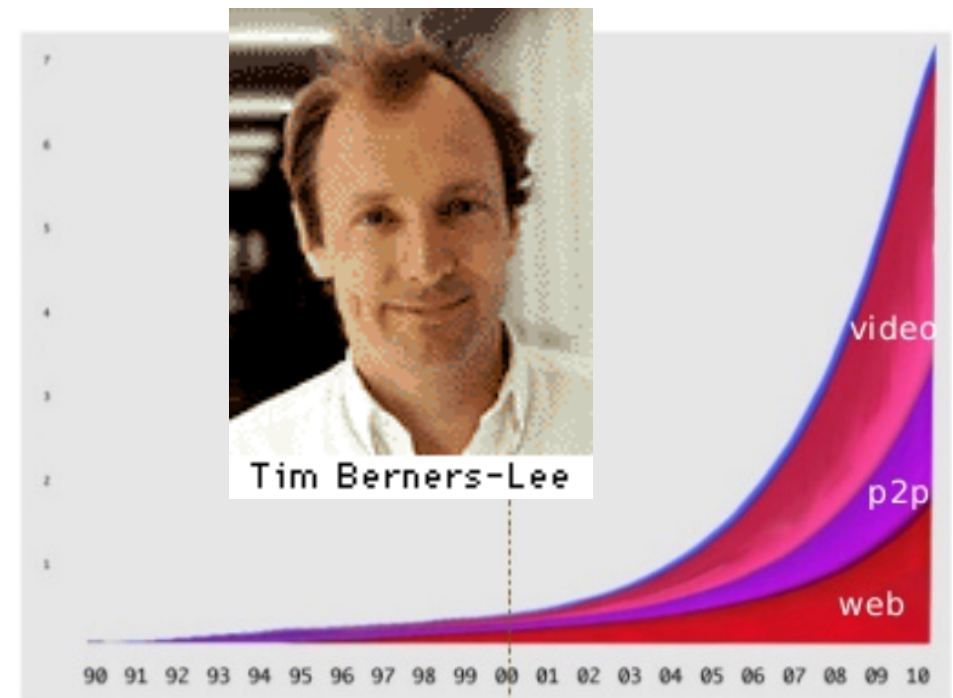




## World Wide Web e Hypertext Markup Language

# World Wide Web

- HTTP tornou-se predominante depois de 95.
  - Investigador do CERN (Tim Berners-Lee) propõe HTML que torna Web muito apelativa 1989.
  - HTML ( HyperText Markup Language) junta texto com imagens, sons e videos.
- Existe um crescimento exponencial do numero de servidores, de conteúdos e cibernautas.
- Web é serviço da Internet, não são a mesma coisa!





# WWW

- Cada página possui um URL (Universal Resource Locator).
- <http://www.ua.pt/uaonline/default.asp>
- Formato de um URL é:
- `http://hostname: [porto]/path [;paremetros] [?query]`



# HTML

- Formado por um conjunto de tags - `<HTML>`, `<FORM>`, `<TD>` `</TD>`, normalmente criadas aos pares.
- Define como browser deve mostrar aspecto de cada página.
- HTML inclui tags para incluir imagens, videos, sons e hiperligações.
- Cada um dos componentes são pedidos ao servidor em separado.



# TAGS HTML



Tag	Description
<code>&lt;html&gt; ... &lt;/html&gt;</code>	Declares the Web page to be written in HTML
<code>&lt;head&gt; ... &lt;/head&gt;</code>	Delimits the page's head
<code>&lt;title&gt; ... &lt;/title&gt;</code>	Defines the title (not displayed on the page)
<code>&lt;body&gt; ... &lt;/body&gt;</code>	Delimits the page's body
<code>&lt;h n&gt; ... &lt;/h n&gt;</code>	Delimits a level <i>n</i> heading
<code>&lt;b&gt; ... &lt;/b&gt;</code>	Set ... in boldface
<code>&lt;i&gt; ... &lt;/i&gt;</code>	Set ... in italics
<code>&lt;center&gt; ... &lt;/center&gt;</code>	Center ... on the page horizontally
<code>&lt;ul&gt; ... &lt;/ul&gt;</code>	Brackets an unordered (bulleted) list
<code>&lt;ol&gt; ... &lt;/ol&gt;</code>	Brackets a numbered list
<code>&lt;li&gt; ... &lt;/li&gt;</code>	Brackets an item in an ordered or numbered list
<code>&lt;br&gt;</code>	Forces a line break here
<code>&lt;p&gt;</code>	Starts a paragraph
<code>&lt;hr&gt;</code>	Inserts a horizontal rule
<code>&lt;img src="..."&gt;</code>	Displays an image here
<code>&lt;a href="..."&gt; ... &lt;/a&gt;</code>	Defines a hyperlink

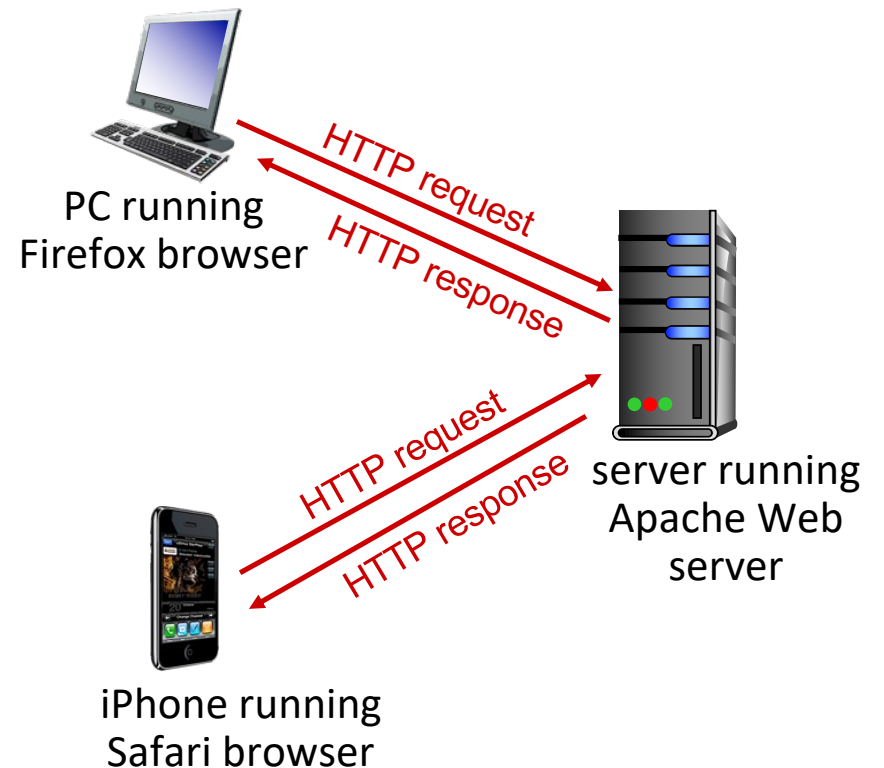


## Hypertext Transport Protocol

# HTTP overview

## HTTP: hypertext transfer protocol

- Web's application-layer protocol
- Modelo client/server:
  - *client*: browser que pede, recebe, (usando o protocolo HTTP) e “mostra” conteúdo Web
  - *server*: Web server envia (usando o protocolo HTTP) objectos em resposta a pedidos



# HTTP overview (continued)

## *HTTP usa TCP:*

- client inicia conexão TCP (cria socket) para server, no port 80
- server aceita conexão TCP do cliente
- Mensagens HTTP (application-layer protocol messages) trocada entre browser (HTTP client) e Web server (HTTP server)
- Ligação TCP é fechada



## *HTTP é “stateless”*

- server não mantém *no* informação acerca dos pedidos anteriores

### *detalhe*

protocolos que mantêm  
“estado” são complexos!

- histórico (estado) tem que ser mantido
- se server ou client crasha, a sua visão do estado tem que ser reconciliada

# Ligações HTTP: dois tipos

## *HTTP Não-persistentes*

Ligações TCP abertas

1. Pelo menos um object enviado sobre a ligação TCP
2. ligação TCP fechada

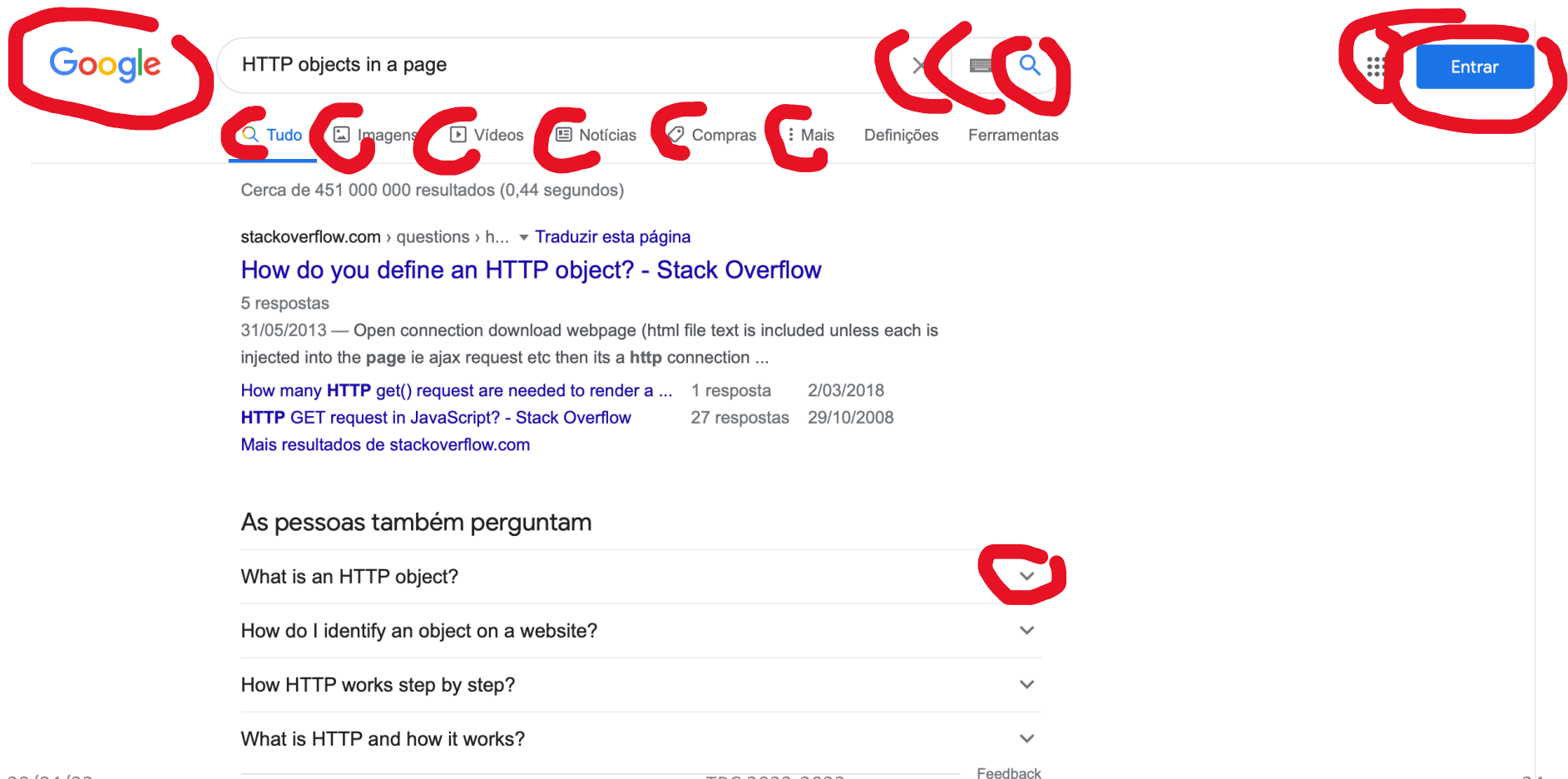
Descarga de múltiplos objectos obriga a múltiplas ligações



## *HTTP Persistente*

- Ligações TCP abertas para o server
- múltiplos objectos podem ser enviados numa *simples* ligação TCP entre o client, e esse server
- Ligação TCP é fechada

# Objectos HTTP



Google

HTTP objects in a page

Tudo Imagens Vídeos Notícias Compras Mais Definições Ferramentas

Cerca de 451 000 000 resultados (0,44 segundos)

stackoverflow.com › questions › h... Traduzir esta página

## How do you define an HTTP object? - Stack Overflow

5 respostas

31/05/2013 — Open connection download webpage (html file text is included unless each is injected into the **page** ie ajax request etc then its a **http** connection ...

How many **HTTP** get() request are needed to render a ... 1 resposta 2/03/2018

**HTTP** GET request in JavaScript? - Stack Overflow 27 respostas 29/10/2008

Mais resultados de stackoverflow.com

### As pessoas também perguntam

- What is an HTTP object? ▾
- How do I identify an object on a website? ▾
- How HTTP works step by step? ▾
- What is HTTP and how it works? ▾

Feedback



# HTTP Não-persistente : exemplo

User digita o URL: `www.someSchool.edu/someDepartment/home.index`  
(containing text, references to 10 jpeg images)



**1a.** Cliente HTTP inicia ligação TCP para HTTP server (processo) em `www.someSchool.edu` no porto 80



**1b.** HTTP server at host `www.someSchool.edu` waiting for TCP connection at port 80 “accepts” connection, notifying client

**2.** HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object `someDepartment/home.index`

**3.** HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

20/04/23

TRC 2022-2023

25

# HTTP Não-persistente: exemplo (cont.)

User enters URL: `www.someSchool.edu/someDepartment/home.index`  
(containing text, references to 10 jpeg images)



4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

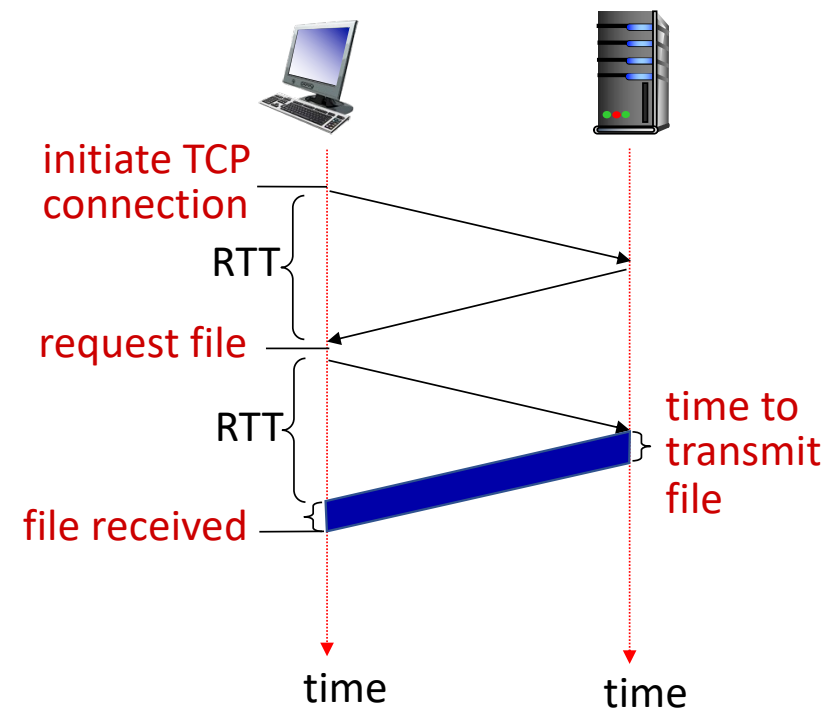
time

# Non-persistent HTTP: response time

**RTT (definition):** tempo de envio de pedido ao servidor e recepção de resposta

**Tempo de resposta de HTTP (por objecto):**

- um RTT para iniciar ligação TCP
- um RTT para pedido HTTP e primeiros bytes de devolução da resposta HTTP
- objectos/file transmission time



*Non-persistent HTTP response time =  $2RTT + \text{file transmission time}$*

# HTTP persistente (HTTP 1.1)

## *Problemas HTTP não-persistente:*

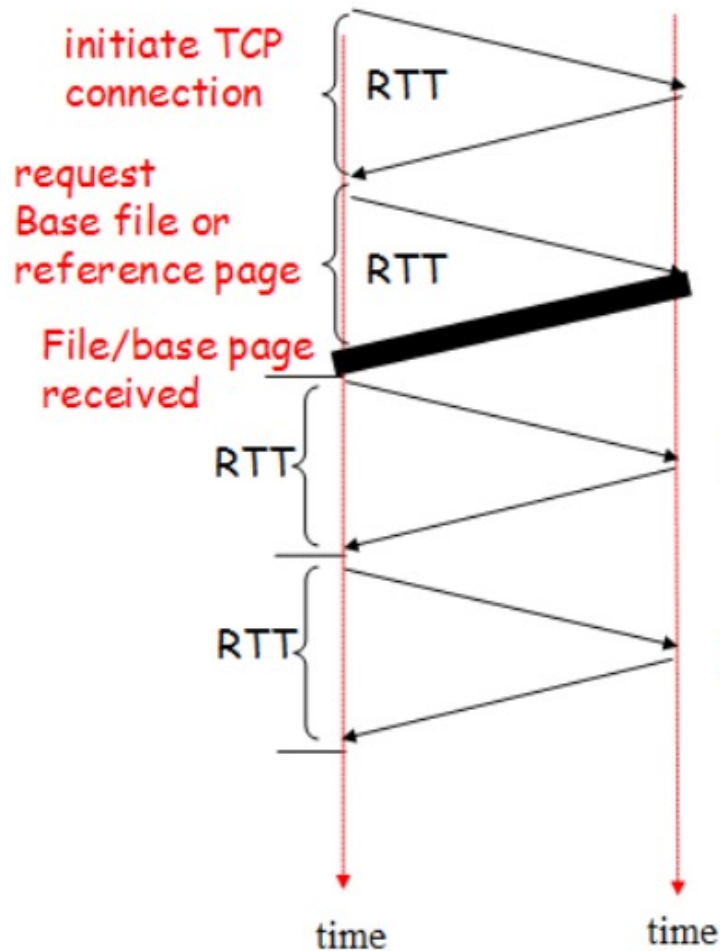
- requer 2 RTTs por objeto
- Sobrecarga de *cada* ligação TCP
- browsers abrem frequentemente múltiplas ligações para retirar objetos em paralelo



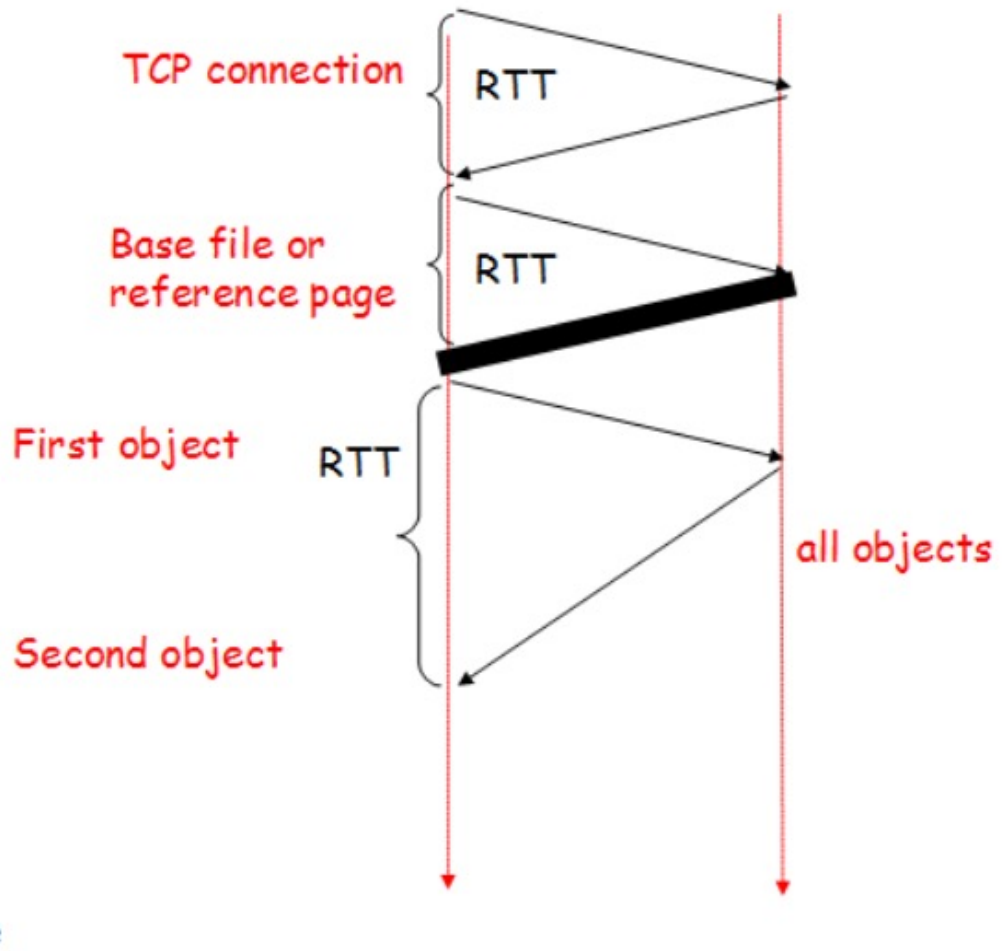
## *Persistent HTTP (HTTP1.1):*

- server deixa ligação aberta depois de enviar resposta
- mensagens HTTP subsequentes entre o mesmo client/server enviadas na mesma ligação
- cliente envia pedidos assim que encontra o objeto referenciado
- Só um pequeno para todos os objetos referenciados (cortando para metade o tempo de resposta)

## Persistent & Pipelined/non-pipelined connections



□ Persistent without pipelining



□ Persistent with pipelining

2: Application Layer 13

# HTTP request message

\* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

- 2 tipos de HTTP messages: *request, response*
- **HTTP request message:**
  - ASCII (human-readable format)

request line (GET, POST, HEAD commands)

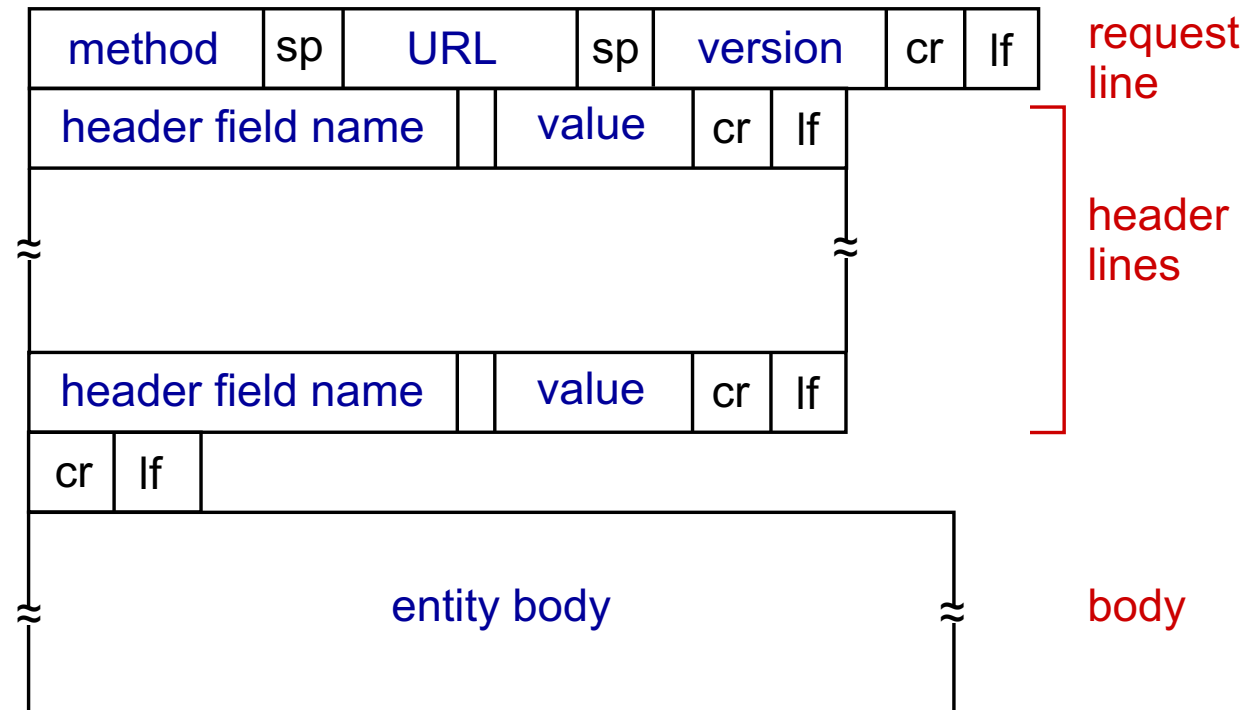
header lines

carriage return, line feed at start of line indicates end of header lines

carriage return character  
line-feed character

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
10.15; rv:80.0) Gecko/20100101 Firefox/80.0 \r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Connection: keep-alive\r\n
\r\n
```

# HTTP request message: general format



# Outras mensagens de pedido HTTP

## POST method:

- web page often includes form input
- user input sent from client to server in entity body of HTTP POST request message



## GET method (for sending data to server):

- include user data in URL field of HTTP GET request message (following a '?'):

`www.somesite.com/animalsearch?monkeys&banana`

## HEAD method:


- requests headers (only) that would be returned *if* specified URL were requested with an HTTP GET method.

## PUT method:

- uploads new file (object) to server
- completely replaces file that exists at specified URL with content in entity body of POST HTTP request message



# HTTP response message

status line (protocol  status code status phrase) **HTTP/1.1 200 OK**



# HTTP response status codes

- status code appears in 1st line in server-to-client response message.
- some sample codes:

## 200 OK

- request succeeded, requested object later in this message

## 301 Moved Permanently

- requested object moved, new location specified later in this message (in Location: field)

## 400 Bad Request

- request msg not understood by server

## 404 Not Found

- requested document not found on this server

## 505 HTTP Version Not Supported



# Trying out HTTP (client side) for yourself

## 1. netcat to your favorite Web server:

```
% nc -c -v gaia.cs.umass.edu 80
```

- opens TCP connection to port 80 (default HTTP server port) at gaia.cs.umass.edu.
- anything typed in will be sent to port 80 at gaia.cs.umass.edu

## 2. type in a GET HTTP request:

```
GET /kurose_ross/interactive/index.php HTTP/1.1  
Host: gaia.cs.umass.edu
```

- by typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

## 3. look at response message sent by HTTP server!

(or use Wireshark to look at captured HTTP request/response)



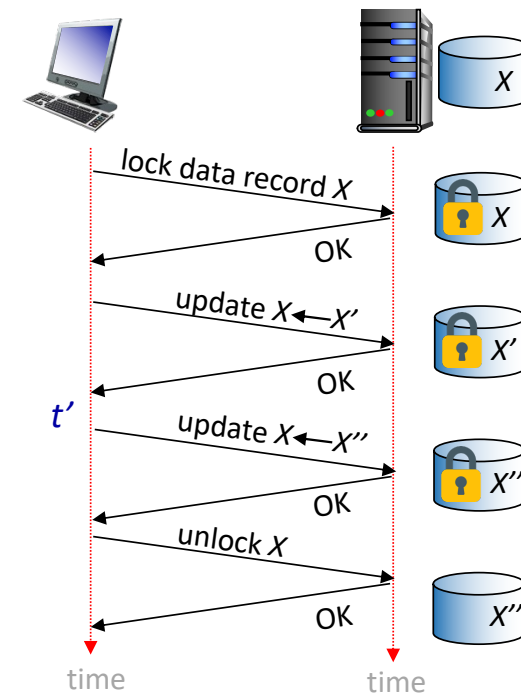
# Maintaining user/server state: cookies

Recall: HTTP GET/response interaction is *stateless*

- no notion of multi-step exchanges of HTTP messages to complete a Web “transaction”

- no need for client/server to track “state” of multi-step exchange
- all HTTP requests are independent of each other
- no need for client/server to “recover” from a partially-completed-but-never-completely-completed transaction

a **stateful protocol**: client makes two changes to  $X$ , or none at all



**Q:** what happens if network connection or client crashes at  $t'$  ?

# Maintaining user/server state: cookies

Web sites and client browser use *cookies* to maintain some state between transactions

*four components:*

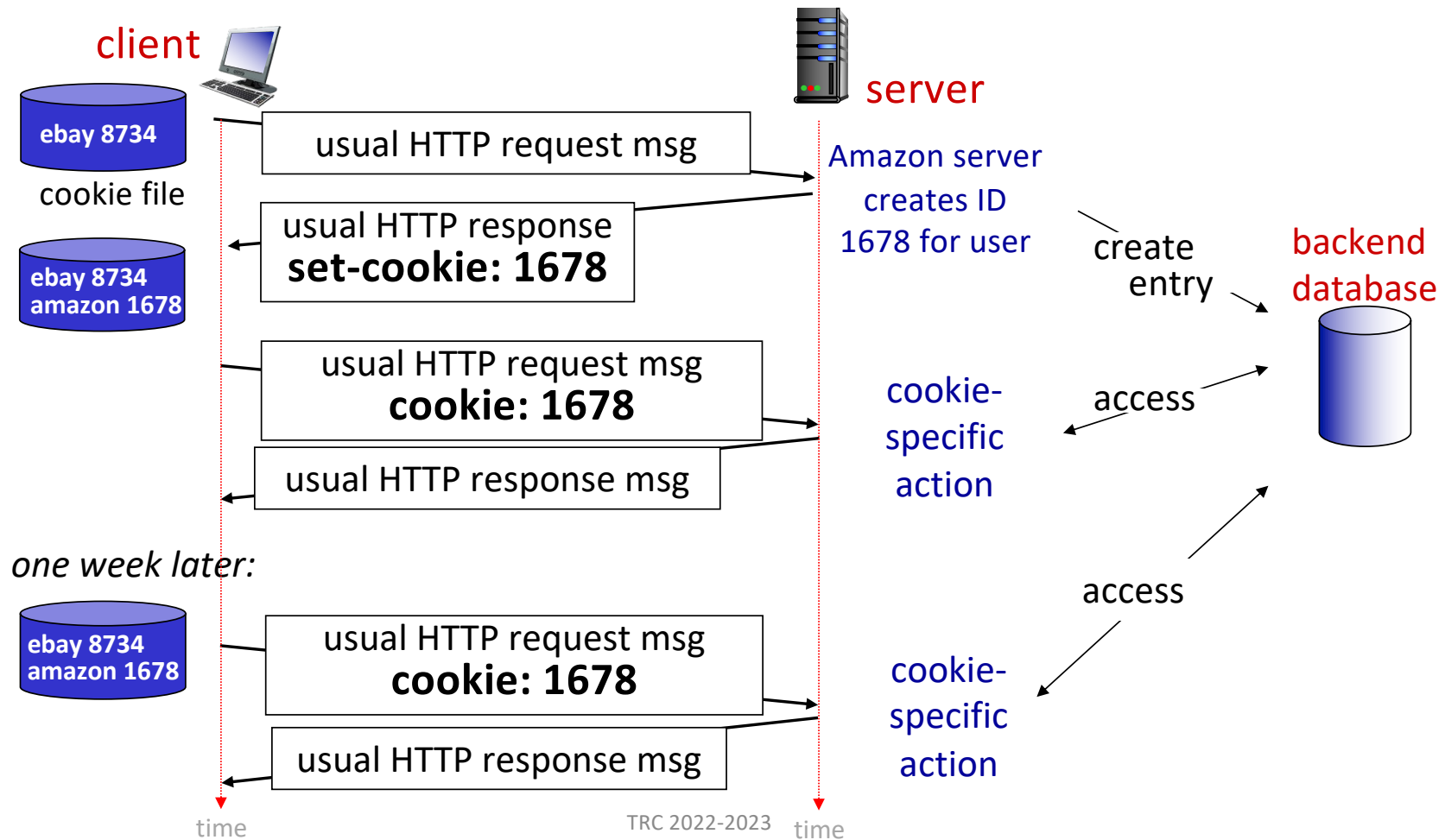
- 1) cookie header line of HTTP *response* message
- 2) cookie header line in next HTTP *request* message
- 3) cookie file kept on user's host, managed by user's browser
- 4) back-end database at Web site



**Example:**

- Susan uses browser on laptop, visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID (aka “cookie”)
  - entry in backend database for ID
- subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to “identify” Susan

# Maintaining user/server state: cookies



# cookies HTTP : comentários

## *What cookies can be used for:*

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

## *Challenge: How to keep state?*

- *at protocol endpoints:* maintain state at sender/receiver over multiple transactions
- *in messages:* cookies in HTTP messages carry state



## *Privacidade e cookies:* aside

- cookies permit sites to *learn* a lot about you on their site.
- third party persistent cookies (tracking cookies) allow common identity (cookie value) to be tracked across multiple web sites

# Exemplo de resposta de um servidor HTTP

- Trying 4.17.168.6...
- Connected to www.ietf.org.
- Escape character is '^']'.
- HTTP/1.1 200 OK
- Date: Wed, 08 May 2002 22:54:22 GMT
- Server: Apache/1.3.20 (Unix) mod3ssl/2.8.4 OpenSSL/0.9.5a
- Last-Modified: Mon, 11 Sep 2000 13:56:29 GMT
- ETag: "2a79d-c8b-39bce48d"
- Accept-Ranges: bytes
- Content-Length: 3211
- Content-Type: text/html
- X-Pad: avoid browser bug



- <html>
- <head>
- <title>IETF RFC Page</title>
- <script language= javascript >
- function url() {
- var x = document.form1.number.value
- if (x.length == 1) {x = "000" + x }
- if (x.length == 2) {x = "00" + x }
- if (x.length == 3) {x = "0" + x }
- document.form1.action = "/rfc/rfc" + x + ".txt"
- document.form1.submit
- }
- </script>
- </head>





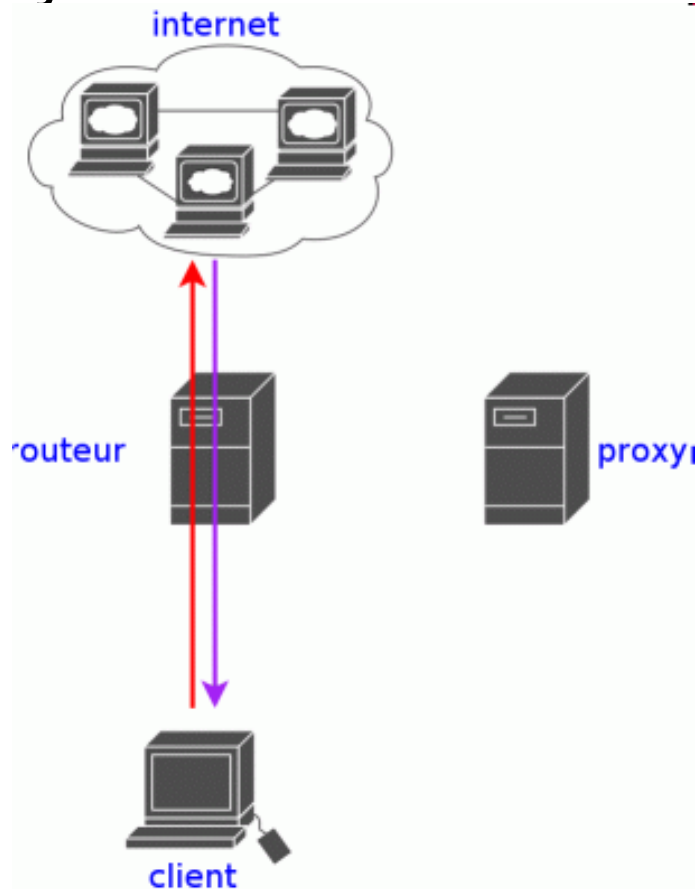
## Web proxies

# Suporte de Proxy servers

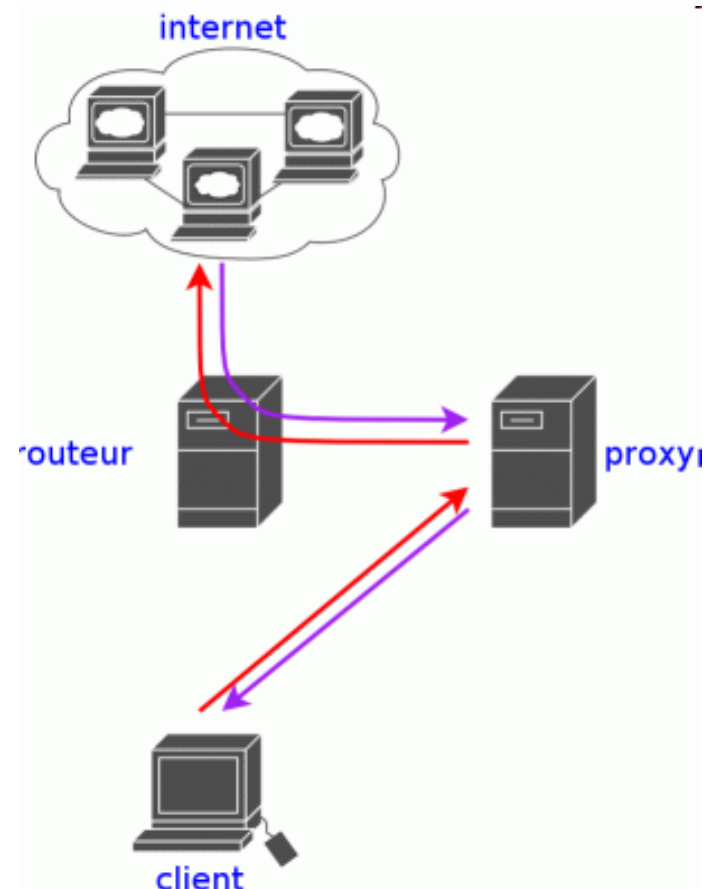
- Proxies são intermediários dos pedidos HTTP.
- Fazem cache das respostas aos pedidos que são feitos reutilizando essa informação por muitos pedidos.
- Proxy contacta periodicamente servidor HTTP.
- Browser são configurados para fazerem pedidos à proxy.
- Proxy é também integrada com firewall frequentemente.



# Objectivo



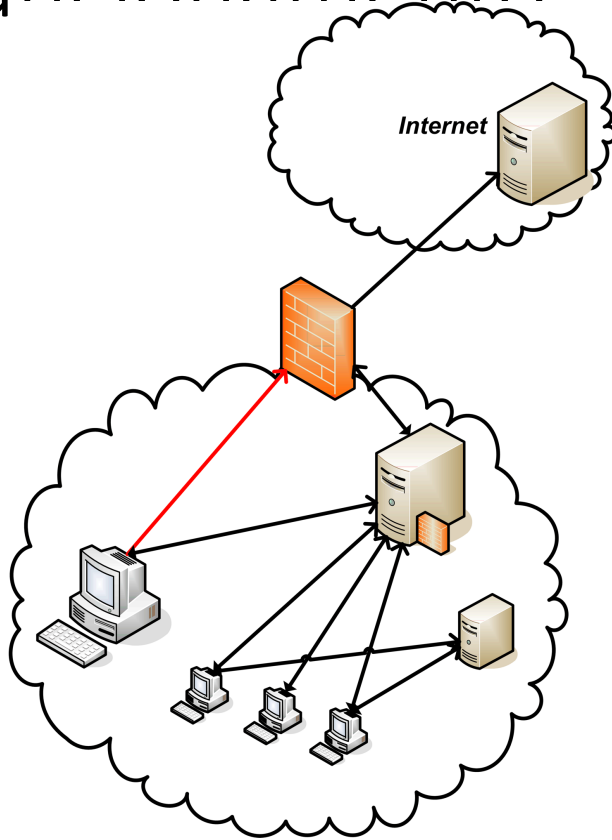
20/04/23



TRC 2022-2023

43

# Funcionamento



- Firewall barra acessos directos ao exterior
- Pedidos são efectuados à proxy;
- Proxy pede ao exterior se não tiver.
- Faz cache da informação por algum tempo;
- Sempre que tiver informação em cache devolve-a sem pedir ao exterior;
- Acesso ao interior é efectuado sem pedir à proxy.
- Clientes tem que ser configurados.

# Exemplos de Web Proxies

- Squid
- Microsoft Internet Security and Acceleration Server (ISA Server)
- WinProxy
- Winconnection
- BlueCoat
- Sonicwall
- Polipo
- OMNE smartWEB





## HTTP/2 e HTTP/3

# HTTP/2

*Key goal:* decreased delay in multi-object HTTP requests

HTTP1.1: introduced **multiple, pipelined GETs** over single TCP connection

- server responds *in-order* (FCFS: first-come-first-served scheduling) to GET requests
- with FCFS, small object may have to wait for transmission (**head-of-line (HOL) blocking**) behind large object(s)
- loss recovery (retransmitting lost TCP segments) stalls object transmission



# HTTP/2

*Objetivo principal:* diminuição do atraso em solicitações HTTP multi-objeto

*HTTP/2:* [RFC 7540, 2015] maior flexibilidade no servidor no envio de objetos ao cliente:

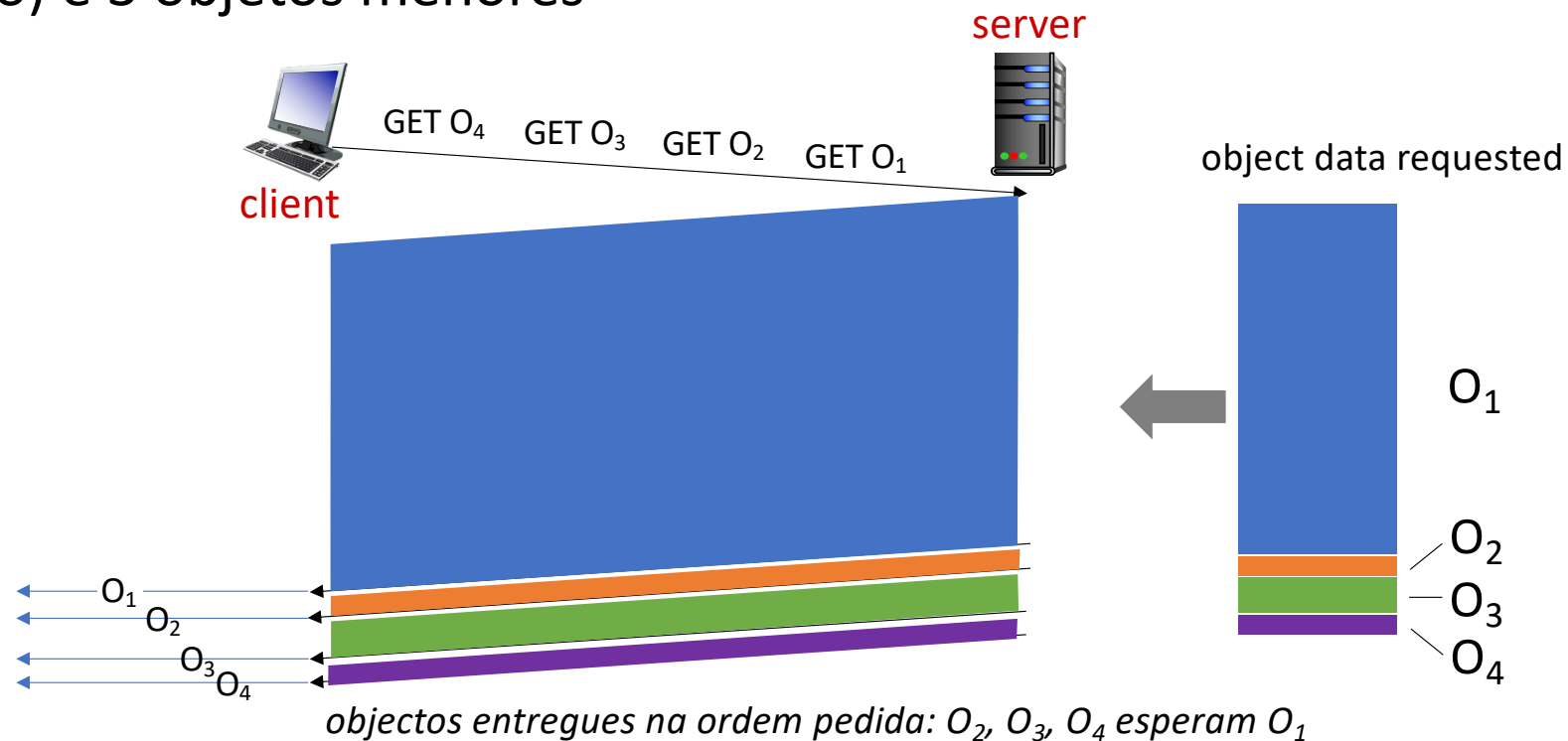
- métodos, códigos de status, a maioria dos campos de cabeçalho inalterados em relação ao HTTP 1.1
- ordem de transmissão de objetos solicitados com base na prioridade de objeto especificada pelo cliente (não necessariamente FCFS)
- enviar objetos não solicitados para o cliente
- dividir objetos em *frames*, agende quadros para mitigar o bloqueio HOL





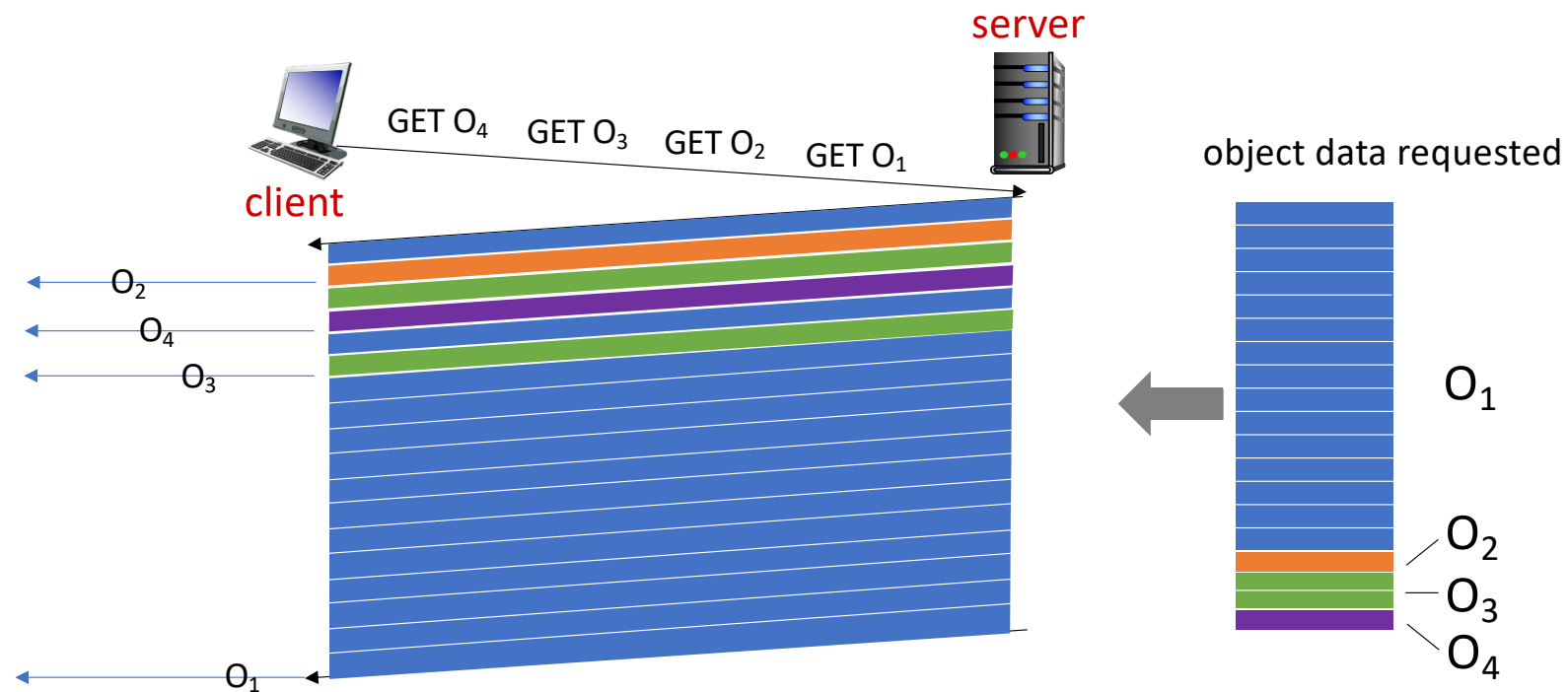
# HTTP/2: mitigating HOL blocking

HTTP 1.1: o cliente solicita 1 objeto grande (por exemplo, arquivo de vídeo) e 3 objetos menores



# HTTP/2: mitigating HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved



*`O2`, `O3`, `O4` delivered quickly, `O1` slightly delayed*

# HTTP/2 to HTTP/3

HTTP / 2 em uma única conexão TCP significa:

- a recuperação da perda de pacotes ainda bloqueia todas as transmissões de objetos
- como no HTTP 1.1, os navegadores têm incentivos para abrir várias conexões TCP paralelas para reduzir a paralisação e aumentar a taxa de transferência geral
- sem segurança na conexão TCP vanilla
- **HTTP/3:** adiciona segurança, por erro de objeto e controle de congestionamento (mais pipelining) sobre UDP
  - mais sobre HTTP / 3 na camada de transporte



# Mais informação

- “Computer Networking: A Top-Down Approach”, 8th edition, Jim Kurose, Keith Ross  
Pearson, 2020
- "Computer Networks", Andrew Tanenbaum, 3rd ed. Prentice Hall, 1996.
- “Internetworking with TCP-IP”, [Douglas E. Comer](#)
- “Data and Computer Communications”, William Stallings  
<http://www.w3.org/People/Berners-Lee/>



# E é tudo...

- Questões?
- Comentários?

