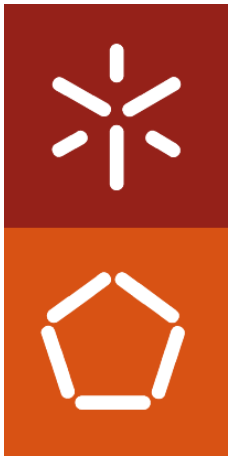
HTTP

Comunicações por Computador EI

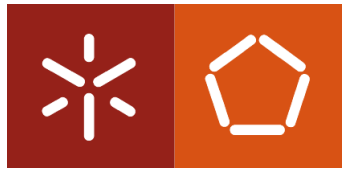
Mestrado Integrado em Engenharia Informática

3º ano/2º semestre

2015/2016



HTTP
Hypertext Transfer Protocol



Conceitos básicos

- **Uma página Web consiste numa colecção de objectos**
- **Um objecto pode ser um ficheiro HTML, uma imagem JPEG image, um applet Java, um ficheiro audio…**
- **Uma página Web consiste num ficheiro de base HTML que inclui várias referências a outros objectos**
- **Cada objecto é endereçado por uma URL (Uniform Resource Locator)**
- **URL exemplo**

`http://www.di.uminho.pt/cursos/miei.jpeg`

host name — path name

HTTP

Como funciona?

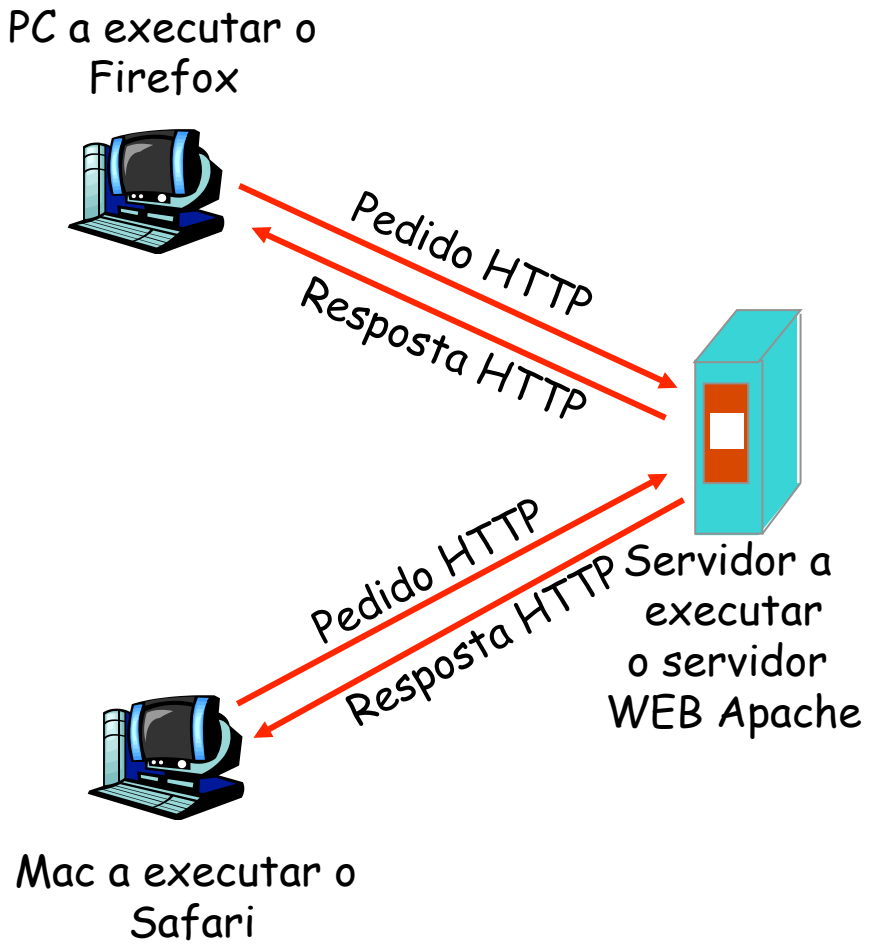
HTTP: hypertext transfer protocol

- **Protocolo do nível da aplicação**
- **Modelo cliente/servidor**
  - *cliente:* browser pede, recebe e mostra objectos Web
  - *servidor:* servidor envia objectos como resposta a pedidos

HTTP 1.0: RFC 1945

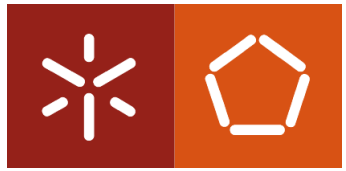
HTTP 1.1: RFC 2068

HTTP 2.0: RFC 7540 (maio 2015)



HTTP

Como funciona?



Utiliza o TCP:

- O cliente inicia uma conexão TCP (cria um socket) com um servidor HTTP (porta 80).
- O servidor TCP aceita o pedido de conexão do cliente
- São trocadas mensagens HTTP (mensagens de protocolo de nível de aplicação) entre o browser (cliente HTTP) e o servidor Web (servidor HTTP)
- A ligação TCP é terminada

O HTTP não tem estado

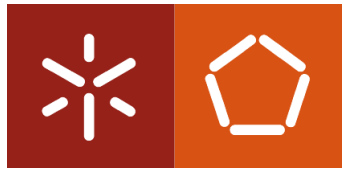
- O servidor não mantém estado acerca dos pedidos anteriores dos clientes

Os protocolos orientados ao estado são muito complexos!

- O passado tem que ser armazenado
- Se o servidor/cliente falham a sua visão do estado pode ficar inconsistente e terá que ser sincronizada

HTTP

Tipos de ligações



HTTP não persistente

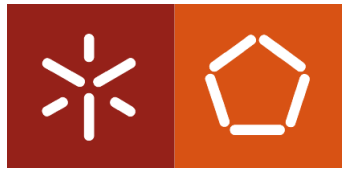
- **Só pode ser enviado no máximo um objecto Web por cada conexão estabelecida**
- **O HTTP/1.0 utiliza HTTP não persistente**

HTTP persistente

- **Podem ser enviados múltiplos objectos Web por cada ligação estabelecida entre o cliente e o servidor.**
- **O HTTP/1.1 usa por defeito conexões persistentes**

HTTP

Não persistente



Supondo que o utilizador introduziu a url `www.uminho.pt/DI/index.html`

(contém texto e referência para imagens jpeg)

1a. O cliente HTTP inicia uma conexão TCP com o servidor HTTP que está a ser executado no sistema www.uminho.pt e está à escuta na porta 80

1b. O servidor HTTP que está a ser executado no sistema www.uminho.pt e está à escuta na porta 80 aceita o pedido de conexão e avisa o cliente

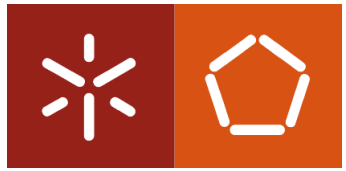
2. O cliente HTTP envia uma mensagem HTTP do tipo *request message* (contendo a URL) através de um novo socket TCP. A mensagem indica que o cliente deseja o objecto Web DI/ index.html

3. O servidor HTTP recebe a *request message* e constroi uma *response message* que contém o objecto Web requerido, enviando depois essa mensagem através do socket TCP estabelecido

tempo

HTTP

Não persistente



4. O servidor HTTP pede para terminar a conexão, mas a ligação só é terminada quando o cliente receber a response message

5. O cliente HTTP recebe a response message que contem o ficheiro html, mostra o ficheiro e faz o parsing do seu conteúdo encontrando a referência a vários objectos jpeg

6. Repete os passos 1-5 para cada objecto referenciado

tempo

HTTP

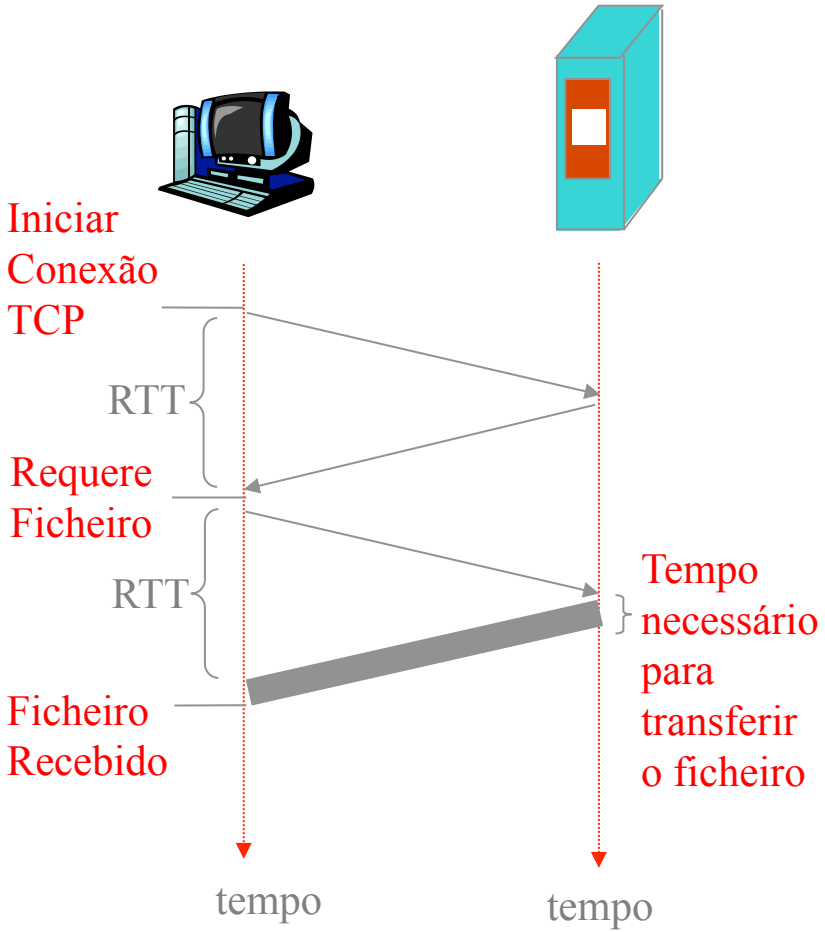
Modelo do Tempo de Resposta

Definição de RTT: tempo que o sinal (1 bit) demora a ir do cliente para o servidor e voltar (2*TempoPropagação +n*tempoEsperanasQueues +n*tempoProcessamento)

Tempo de Resposta

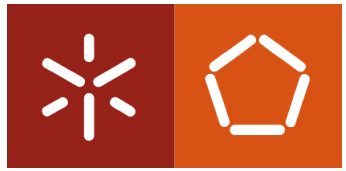
- um RTT para inciar uma conexão TCP
- um RTT para enviar a *request message* e receber o primeiro bit da *response message*
- tempo de transmissão do ficheiro

total = 2RTT+ tempo_transmissão



HTTP

Persistente



HTTP não persistente:

- exige 2 RTTs por objecto
- O Sistema Operativo tem que reservar recursos para cada ligação TCP estabelecida
- Muitos browsers abrem ligações TCP paralelas para irem buscar os objectos referidos

HTTP persistente:

- O servidor deixa a ligação aberta depois de enviar a mensagem de resposta
- Os pedidos HTTP posteriores são enviados através da mesma ligação

Persistente sem pipelining:

- O cliente envia um novo pedido apenas quando recebe a resposta ao anterior
- Um RTT por cada objecto referido

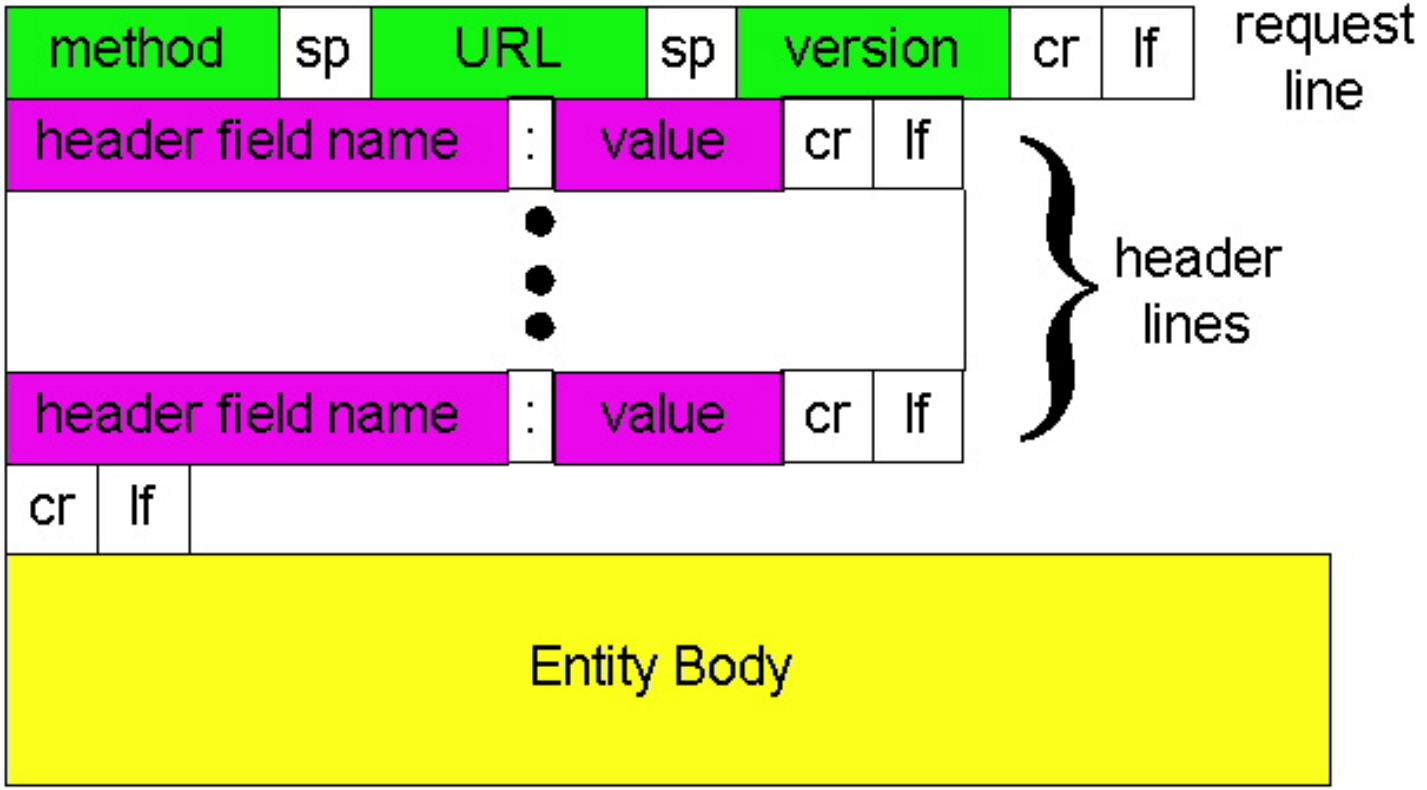
Persistente com pipelining:

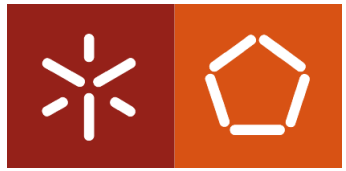
- Modo por defeito no HTTP/1.1
- O cliente envia os pedido assim que os encontra no objecto referenciador
- No mínimo é consumido um RTT por todos os objectos referenciados

Aplicações de rede

Exemplo: HTTP

Formato dos PDU





Exemplo de uma *HTTP Request Message*

GET	/directoria/pagina.html	HTTP/1.1
Host: www.sitio.pt		
Connection: close		
User-Agent: Mozilla/4.0		
Accept-Language: PT		

<new line>

Corpo da mensagem (*vazio no caso do GET*)

Linha do pedido

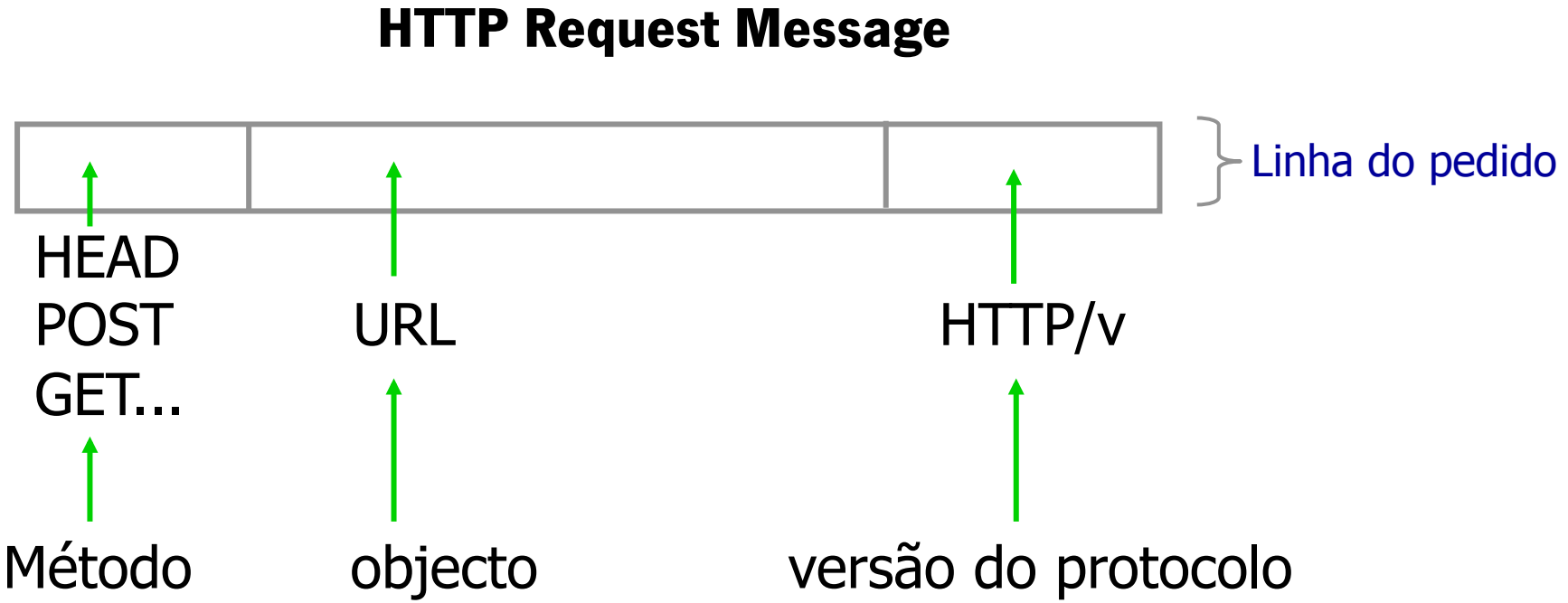
Linhas do cabeçalho

Dados da mensagem

Aplicações de rede

Exemplo: HTTP

Formato dos PDU

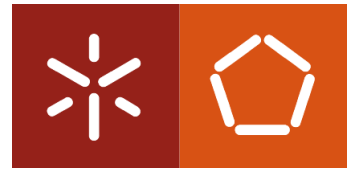


HTTP/1.0 usa conexões TCP não-persistentes:

a conexão é terminada após o envio de cada mensagem

HTTP/1.1 usa conexões TCP persistentes, por defeito

Tipos de Métodos



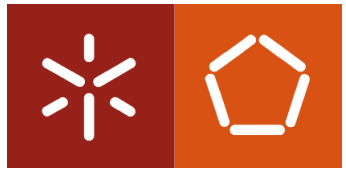
HTTP/1.0

- **GET**
- **POST**
- **HEAD**
  - pede ao servidor para não incluir o objecto requerido na resposta

HTTP/1.1

- **GET, POST, HEAD**
- **PUT**
  - faz o upload do objecto contido no corpo da mensagem na localização especificada no campo URL da mesma mensagem
- **DELETE**
  - apaga o ficheiro especificado no campo URL

Intput de dados através de formulários



Método Post:

- É frequente as páginas Web incluírem um formulário para introdução de dados.
- Nesse caso pode utilizar-se o método POST em vez do método GET.
- O método POST é muito semelhante ao método GET, mas o objecto requerido depende do *input* introduzido pelo utilizador através de um formulário.
- O *Input* introduzido pelo utilizador é enviado para o servidor HTTP no corpo da *HTTP Request Message*, utilizando o método POST.

Método URL:

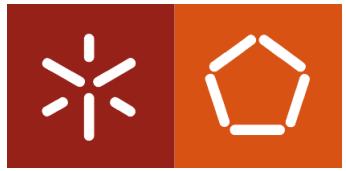
- Utiliza o método GET
- O Input é enviado para o servidor HTTP utilizando o campo URL da *HTTP Request Message*, com o método GET.

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

Aplicações de rede

Exemplo: HTTP

Formato dos PDU



HTTP Response Message

HTTP/1.1	200	OK
Connection: close		
Date: 07 Mai 2003 11:35:15 UTC+1		
Server: Apache/1.3.0 (Unix)		
Last-Modified: 05 Mai 2003 09:23:45 UTC+1		
Content-Length: 6825		
Content-Type: text/html		

<new line>

Corpo da mensagem (objecto)

Linha do tipo da resposta

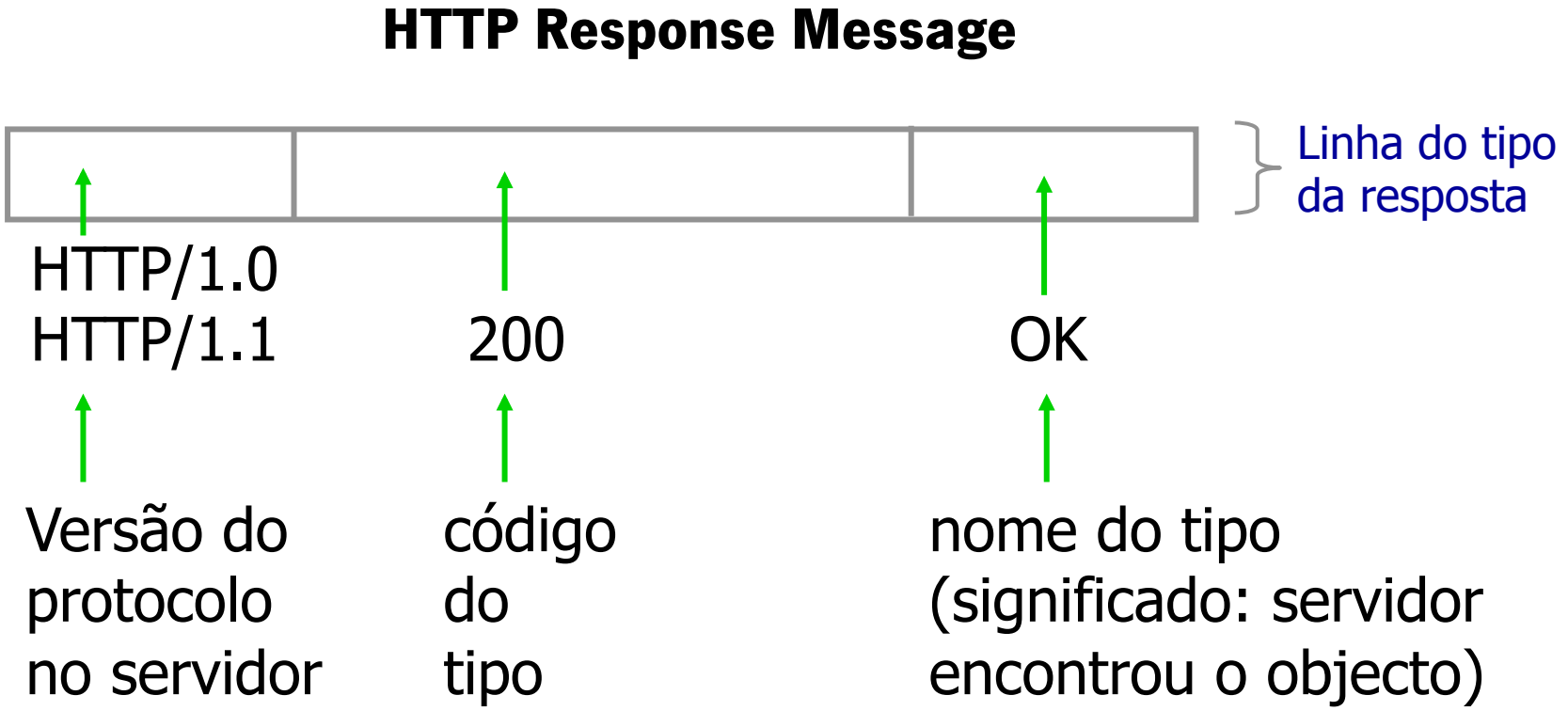
Linhas do cabeçalho

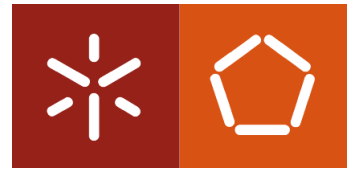
Dados da mensagem

Aplicações de rede

Exemplo: HTTP

Formato dos PDU





Alguns códigos de tipo e seu significado

200 OK

301 Moved permanently, location: xyz

304 Not modified

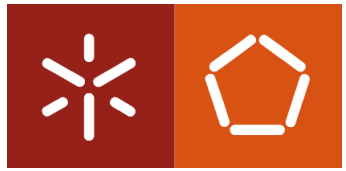
400 Bad request (pedido não entendido)

401 Authorization required

404 Not found (objecto não encontrado)

505 HTTP version not supported

Testar o HTTP (do lado do cliente)



1. Telnet para o Web Server

```
telnet marco.uminho.pt 80
```

Estabelecer uma conexão TCP com a porta 80 da máquina onde está o servidor Web (marco.uminho.pt). Qualquer input introduzido é enviado para a porta 80 da maquina marco.uminho.pt

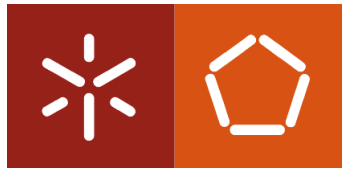
2. Digitar uma *HTTP Request Message:*

```
GET /~costa/ HTTP/1.1
Host: marco.uminho.pt
```

Ao digitar esta mensagem (seguido de dois carriage returns), é enviada uma mensagem mínima, mas completa, ao servidor

3. Analisar a mensagem enviada pelo servidor HTTP!

Cookies: informação de estado



A maioria dos sites Web usa cookies

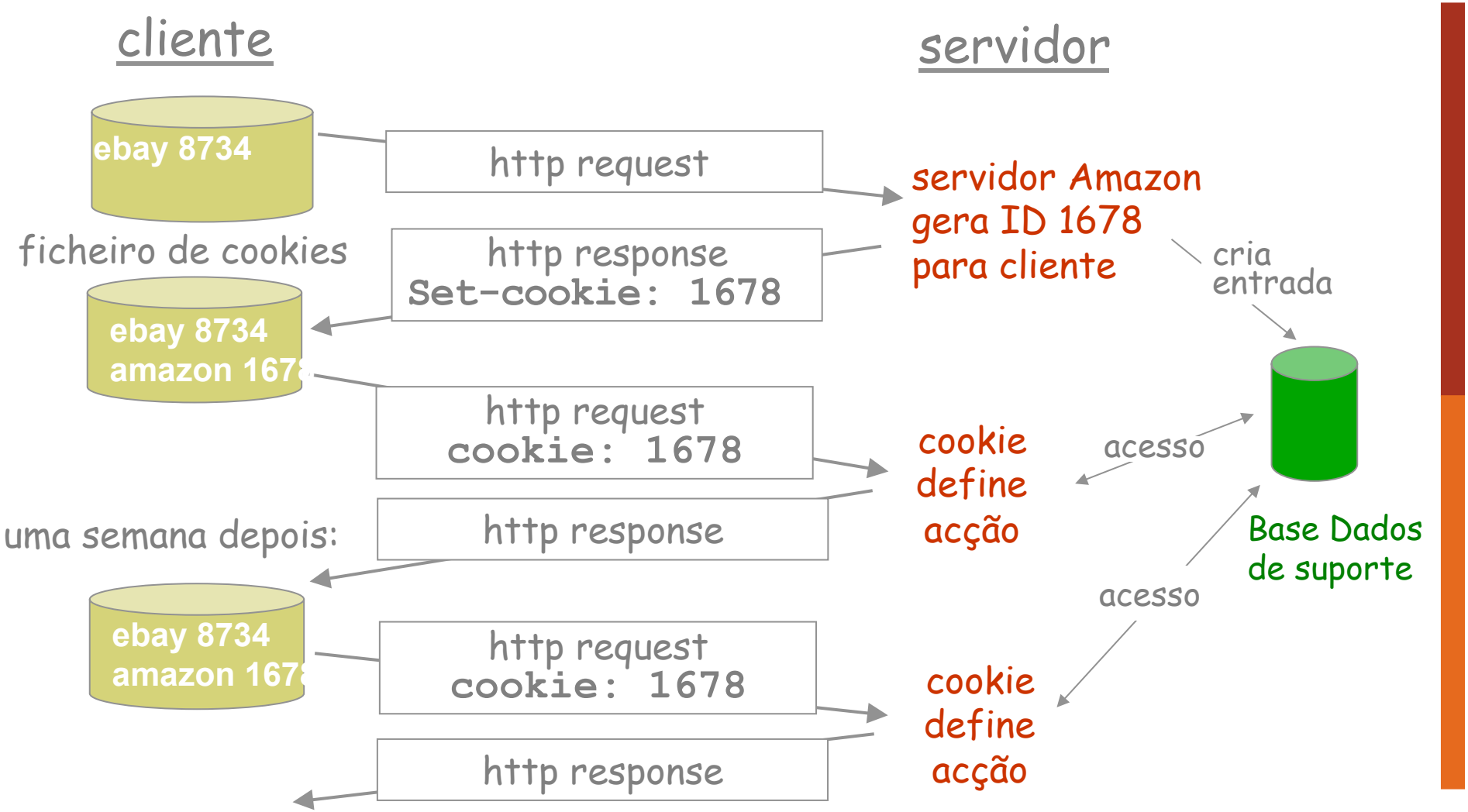
Quatro componentes:

1) Linha com *cookie* no cabeçalho da mensagem *HTTP response*
2) Linha com *cookie* no cabeçalho da mensagem HTTP *request*
3) Ficheiro com *cookies* mantido na máquina do utilizador, gerido pelo seu browser
4) Uma base de dados de suporte do lado servidor Web

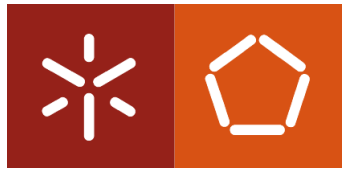
Exemplo:

- **Susana acede sempre à Internet a partir do seu PC**
- **visita um site de comércio electrónico pela primeira vez**
- **quando o primeiro pedido chega ao servidor Web, o servidor gera:**
  - Um Identificador (ID) único
  - Uma entrada na base de dados de suporte para esse ID

Cookies: informação de estado



Cookies: informação de estado



O que os cookies permitem:

- autorização
- cabaz de compras
- sugestões ao utilizador
- informação de sessão por utilizador (ex: Web e-mail)

efeitos colaterias

Os Cookies e a privacidade:

- os cookies ensinam muito aos servidores a respeito dos utilizadores e seus hábitos
- o utilizador pode fornecer nome e e-mail ao servidor

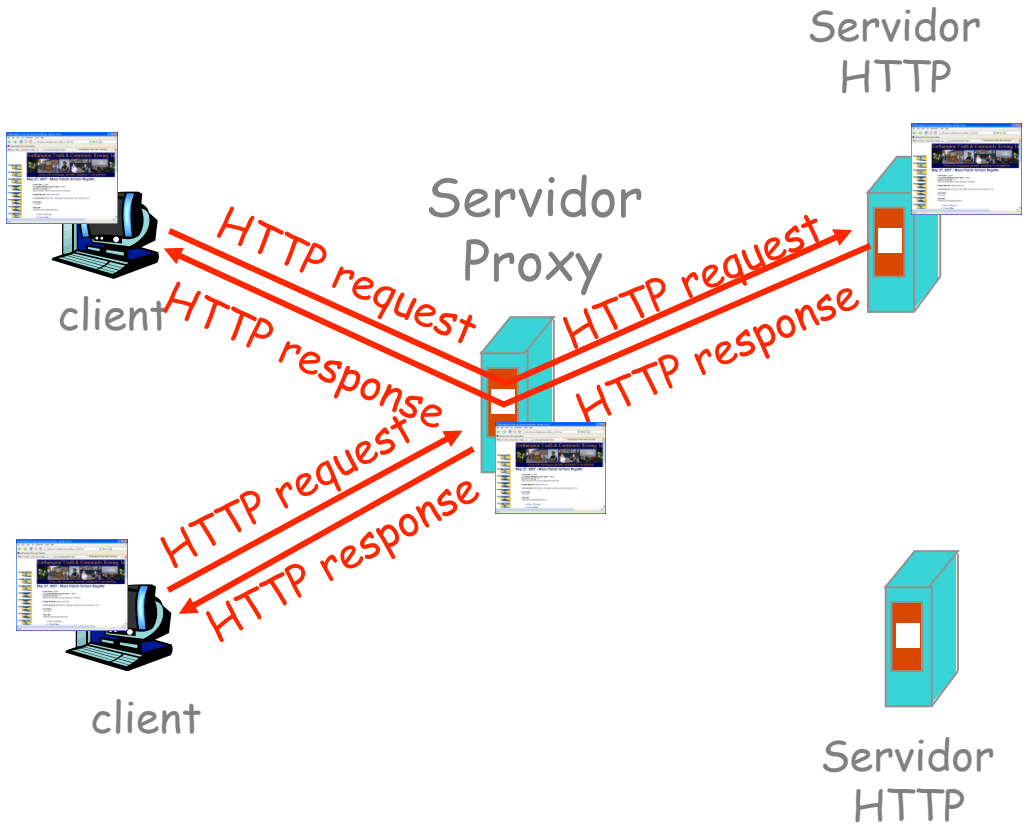
Como manter "estado":

- entidades protocolares: guardam estado por emissor/receptor entre transacções distintas
- cookies: forma como as mensagens http transportam a informação de estado

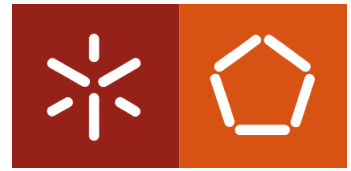
Web caches (servidor proxy)

Objectivo: satisfazer o pedido do cliente sem envolver o servidor HTTP alvo

- **O utilizador configura o cliente HTTP (browser) para aceder à Web através de um servidor proxy**
- **O browser enviar todas as *HTTP request messages* para o servidor proxy**
  - Se o objecto requerido está na cache do proxy o servidor proxy retorna o objecto
  - Senão o servidor proxy contacta o servidor HTTP alvo, reenvia-lhe a *HTTP request message*, aguarda a resposta que retorna ao browser



Web caching



- o servidor proxy/cache tem de actuar simultaneamente como cliente e como servidor
- são tipicamente instalados pelos ISP ou pelas próprias instituições (universidades, empresas, ISP residenciais, etc)

Porquê *Web caching*?

- reduz o tempo de resposta para os pedidos dos clientes
- reduz o tráfego nos links de acesso ao exterior (os mais problemáticos para a instituição).
- Internet está povoada de *caches*: permitem que fornecedores de conteúdos mais "pobres" disponibilizem efectivamente os seus conteúdos (mas isso também as redes de partilha de ficheiros P2P)

Exemplo de Caching

Pressupostos

Tamanho médio dos objectos = 100,000 bits

- Taxa média de pedidos efectuados pelos *browsers* da instituição para servidores HTTP = 15/sec
- Tempo médio de atraso desde o pedido HTTP até à chegada da resposta = 2 sec

Consequências

- Utilização da LAN = 15%

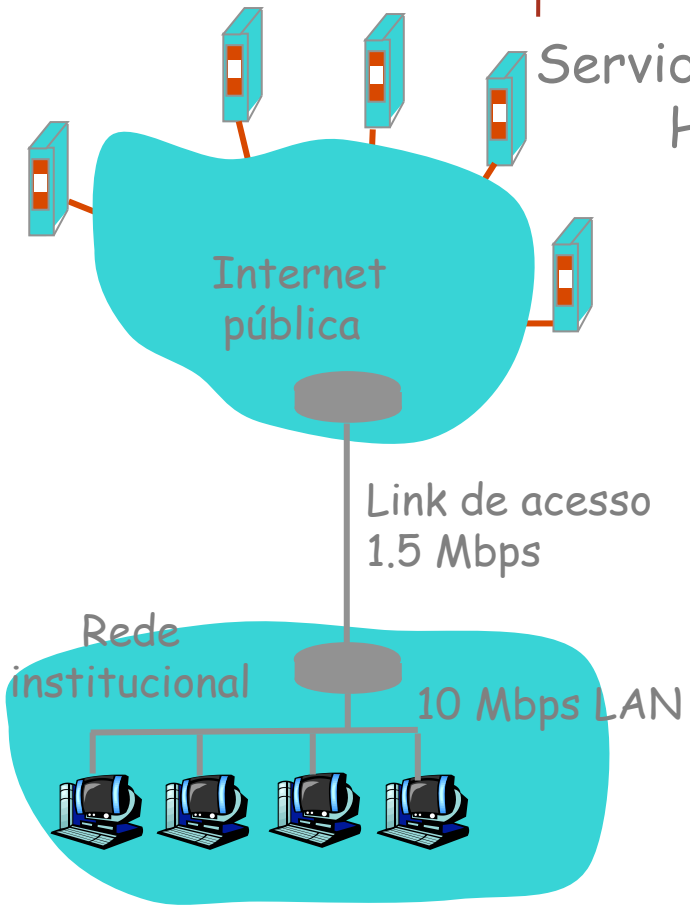
(15 pedidos/sec).(100Kbits/pedido)/(10Mbps)

- Utilização do Link de acesso = 100%

(15 pedidos/sec).(100Kbits/pedido)/(1.5Mbps)

- Total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + milliseconds



Exemplo de Caching (cont)

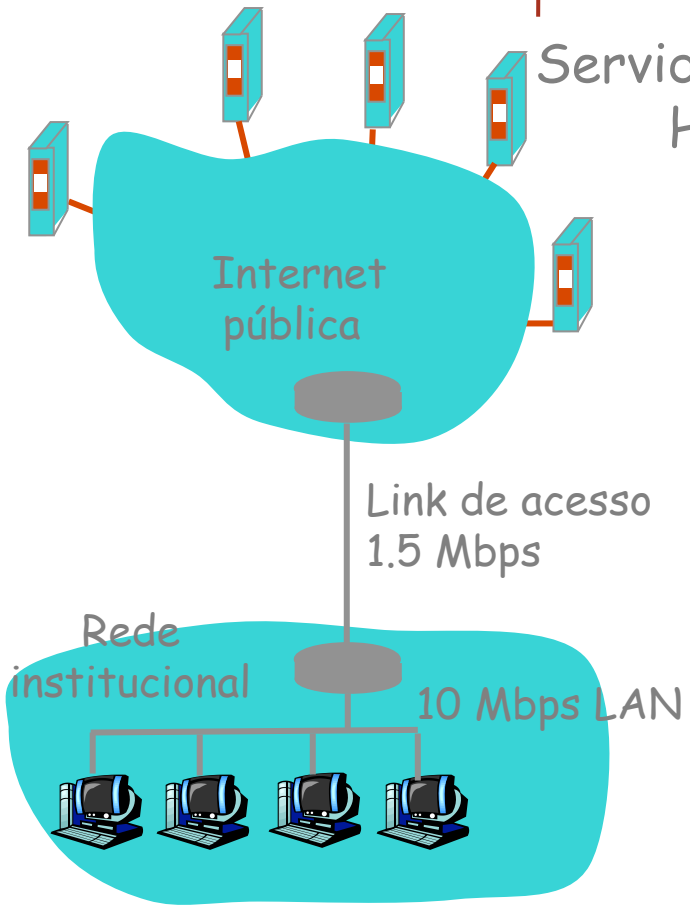
Solução possivel

- Aumentar a largura de banda do link de acesso para 10 Mbps

Consequência

- Utilização da LAN = 15%
- Utilização do Link de Acesso = 15%
- Total delay = Internet delay + access delay + LAN delay

  = 2 sec + msecs + msecs
- É habitualmente muito dispendioso fazer o upgrade do link de acesso de uma instituição



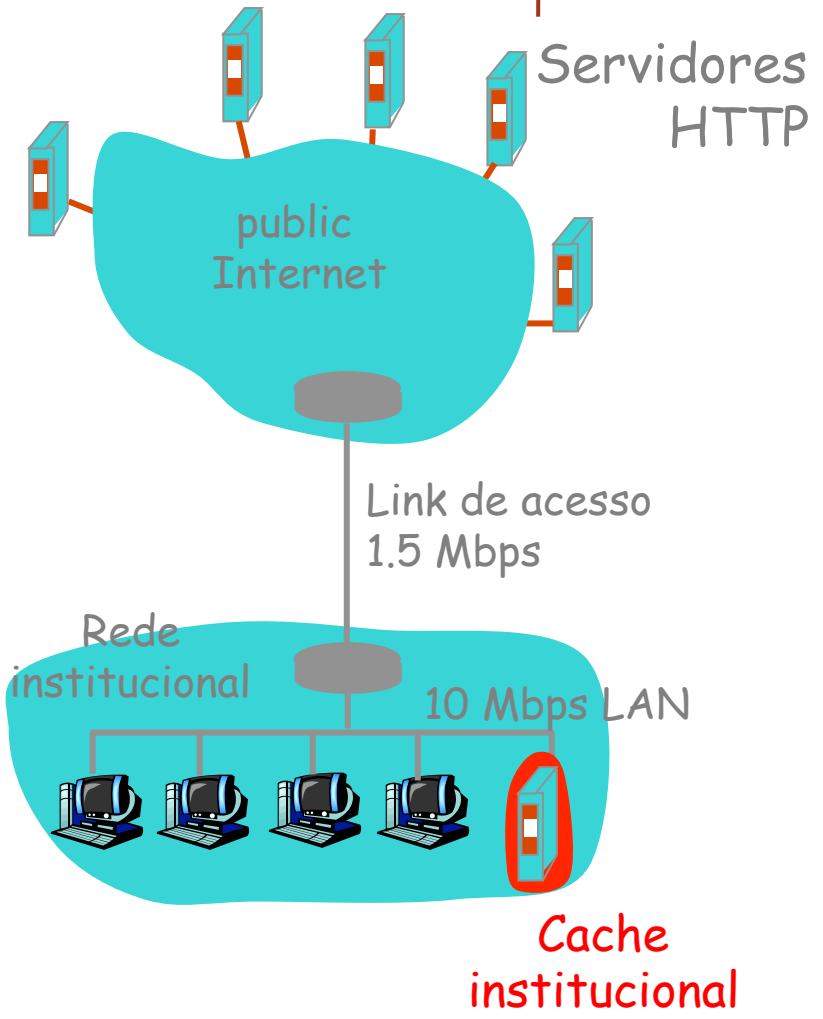
Exemplo de Caching (cont)

Solução possivel: instalar o Web Proxy

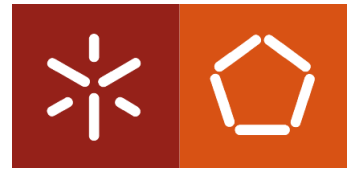
- Se a taxa de acerto for de 0.4

Consequências

- 40% dos pedidos serão satisfeitos imediatamente
- 60% dos pedidos terão que ser redireccionados para o servidor HTTP respectivo
- A utilização do link de acesso será reduzida para 60% resultando em atrasos negligenciáveis (10 msec)
- total avg delay = Internet delay + access delay + LAN delay = .6*(2.01) secs + .4*10msec < 1.4 secs



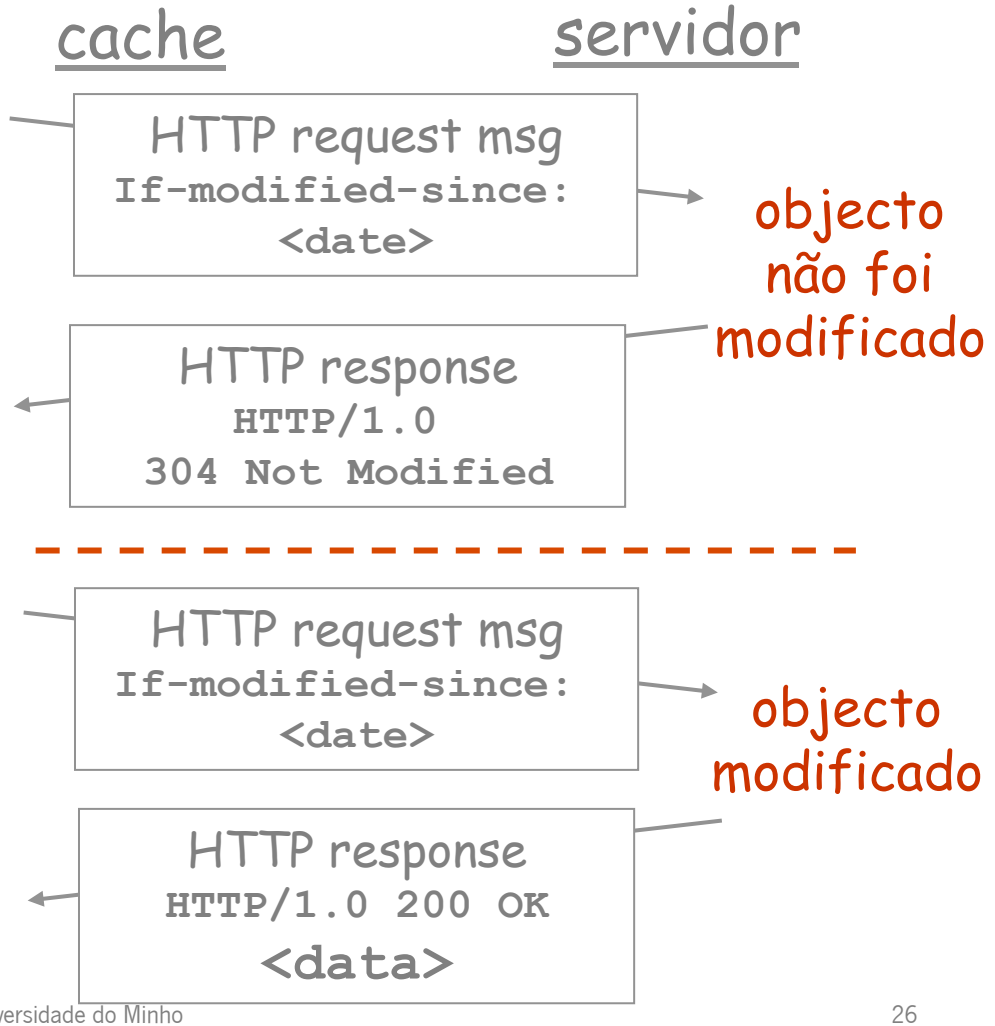
GET Condicional



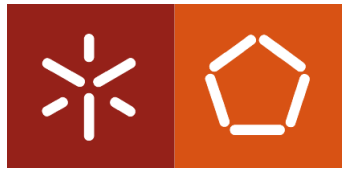
- **Objectivo:** não enviar o objecto se a cópia mantida em cache está actualizada
- *cache*: inclui no cabeçalho do pedido HTTP, a data da cópia guardada na cache

  `If-modified-since: <date>`
- *servidor*: resposta não contém nenhum objecto se a cópia mantida em cache estiver actualizada:

  `HTTP/1.0 304 Not Modified`

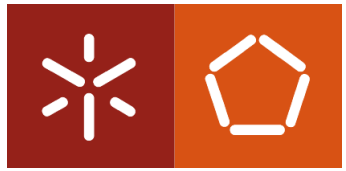


Exercício



- **Pretende-se estimar o atraso na recepção de um documento Web usando o protocolo HTTP. Sabemos que o atraso de ida-e-volta entre cliente e servidor é 4 ms, que o débito do caminho que une o cliente ao servidor é 1024 Kbps e que cada segmento TCP contém no máximo 128 *bytes* de dados. Desprezam-se os tempos de transmissão dos cabeçalhos; em particular, despreza-se o tempo de transmissão dos segmentos que não contêm dados pertencentes ao documento Web. As respostas às alíneas seguintes devem ser ilustradas com diagramas espaço-tempo**
  - Se o documento consistir num único objecto base com 2048 *bytes*, a memória de recepção TCP for ilimitada e o TCP utilizar o mecanismo de arranque lento ("slow-start"), mudando para a fase de "congestion avoidance" quando a janela atinge os 4 segmentos, determine o atraso na recepção do documento, desde o instante em que o cliente estabelece contacto com o servidor até que o documento é recebido na totalidade.
  - Assuma, agora, que o documento Web contém 4 imagens que são referenciadas no objecto base. Cada imagem contém 1024 *bytes* e a versão de HTTP usada é não-persistente (1.0) suportando um máximo de 2 sessões paralelas. Determine o atraso até à recepção do documento, considerando que a largura de banda disponível é repartida equitativamente entre sessões paralelas.
  - Considere agora que usa a versão 1.1 do protocolo HTTP primeiro sem possibilidade de pedidos em sequência (“pipelining”) e depois com pipelining.

Exercício



- **Pretende-se estimar o tempo mínimo necessário para obter um documento da Web. O documento é constituído por 6 objectos: o objecto base HTML e cinco imagens referenciadas no objecto base. O *browser* está ligado ao servidor HTTP por uma única linha com RTT de 20 ms. O tempo mínimo de transmissão na linha do objecto base HTML é de 8 ms e o tempo mínimo de transmissão na linha de cada imagem é de 80 ms. Admita que o *browser* só pode pedir as imagens quando receber completamente o objecto base. Admita que o utilizador o utilizador sabe o endereço IP do servidor, indicando-o no *browser*. A dimensão dos pacotes de estabelecimento de ligação, de confirmação de estabelecimento de ligação e de envio dos pedidos HTTP é desprezável. Os tempos de processamento dos pacotes são também desprezáveis. Não há mais tráfego nenhum na rede.**
  - Ilustrando a situação com um diagrama temporal, qual o tempo necessário para obter o documento (todos os objectos) se utilizar HTTP não persistente com um máximo de 4 ligações paralelas?
  - Ilustrando a situação com um diagrama temporal, qual o tempo necessário para obter o documento (todos os objectos) se utilizar HTTP/1.1 com *pipelining* em todos os pedidos?