Redes de Computadores

O Protocolo HTTP (Continuação)

Departamento de Informática da
FCT/UNL

Objetivo do Capítulo (continuação)

- HTTP (*Hyper Text Transfer Protocol*) é o principal protocolo que permite o acesso à WWW

- Devido à sua utilização massiva o protocolo tem sido muito expandido
  - a sua relação com as aplicações tem sido modificada
  - a sua relação com o TCP tem sido modificada
  - os cabeçalhos extensíveis têm sido usados intensivamente para acrescentar funcionalidades complementares

Anyone who has lost track of time when using a computer knows the propensity to dream, the urge to make dreams come true and the tendency to miss lunch.

– **Autor:** *Sir Tim Berners-Lee, inventor da Web*

HTTP e Transporte

- HTTP só usa TCP como transporte pois a maioria dos objetos são de dimensão razoável e a fiabilidade é muito importante
- O modelo inicial definido com o HTTP 1.0 era uma conexão TCP por cada interação pedido / resposta
- O modelo do HTTP versão 1.1 é a reutilização da conexão TCP por várias interações pedido / resposta
- A maioria dos browsers usam atualmente várias conexões em paralelo e reutilizam cada uma delas para várias interações pedido / resposta
- A versão HTTP 2.0 permite executar vários pedidos em paralelo dentro da mesma conexão

Modelo Base: Uma Conexão por Objecto

- Modelo muito simples
  - Obter sequencialmente um objecto de cada vez
- Necessidade de estabelecer múltiplas conexões TCP para o mesmo servidor
  - As páginas mais populares têm geralmente dezenas de componentes (imagens, figuras, código, ...) para além da página de base (as mais populares têm mais de 100 objectos)
- As conexões TCP curtas dão pouco rendimento
  - Problema da amortização da abertura da conexão e do *slow start*
- Elevada quantidade de conexões por unidade de tempo
  - O servidor é muito carregado com este aspecto

TCP e Pequenas Transferências

- Muitas páginas são constituídas por inúmeros objetos
  - Uma página inicial em HTML e depois inúmeras referências a outros objetos que são processados imediatamente (fotografias, botões, *frames*, código, etc.)
- Respostas relativamente curtas
  - Muitas mensagens curtas (e.g., alguns kilobytes)
- O *overhead* do TCP é elevado
  - *Three-way handshake* para estabelecer uma conexão (3 pacotes)
  - *Slow start (1, 2, 4, ... Pacotes por RTT)*
  - Até 4 pacotes para fechar a conexão

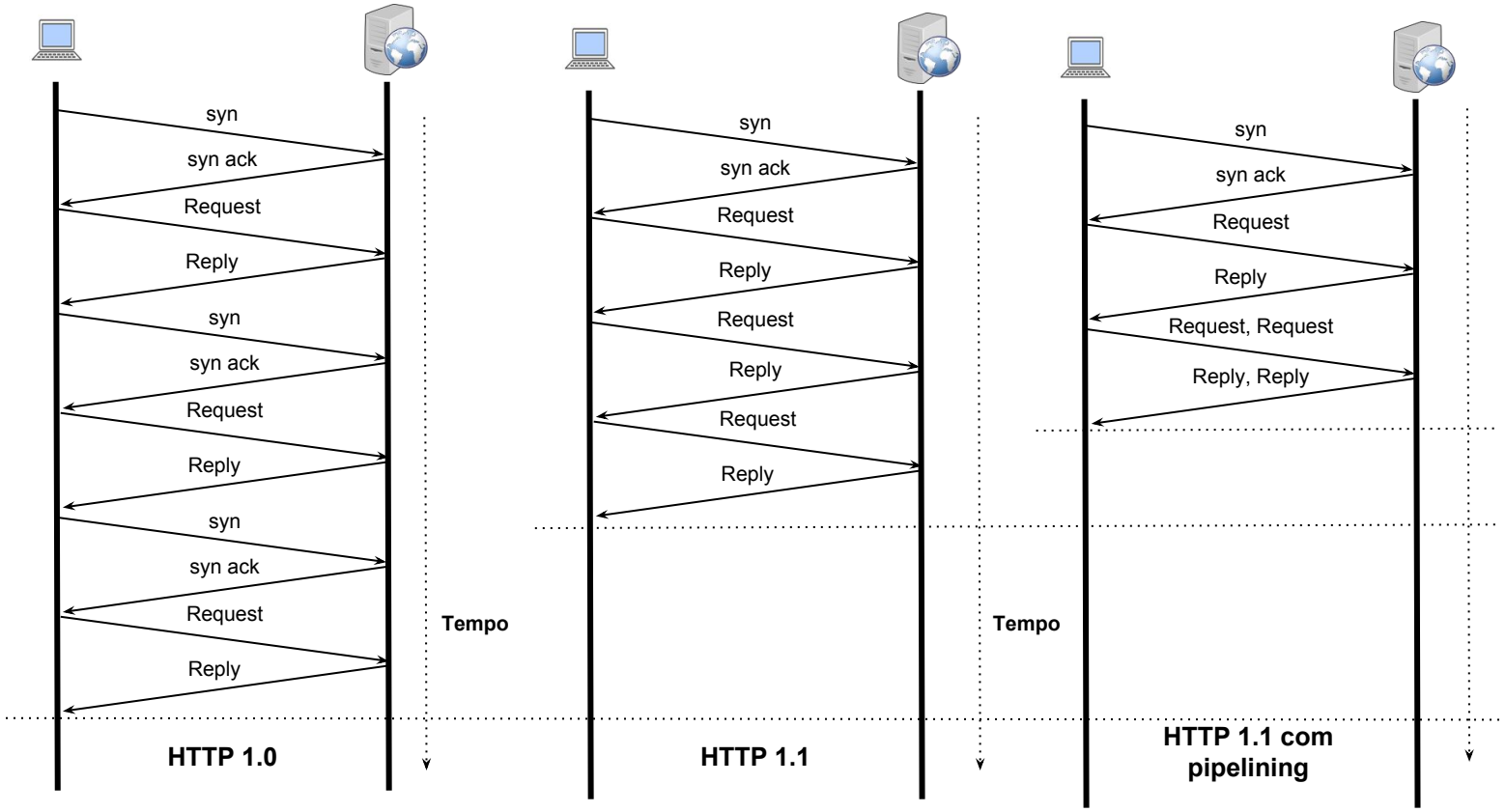
Transferências em Paralelo

- Como a maioria das páginas têm múltiplos objetos
  - Usar sequencialmente uma conexão para cada um não é muito eficaz e a obtenção da página completa torna-se muito lenta
- Alternativa: o browser abre muitas conexões paralelas para cada servidor
  - E obtém vários objetos em paralelo (e.g. 10)
- Alguns prós e vários contras de muitas conexões em paralelo
  - O cliente pode dar prioridades distintas a diferentes objetos
  - Várias transferências em paralelo podem não ser equitativas para outros utilizadores de canais partilhados
  - Várias transferências em paralelo não exploram muito melhor um canal dedicado mas usam melhor um canal partilhado
  - Servidor também tem de suportar mais conexões por unidade de tempo

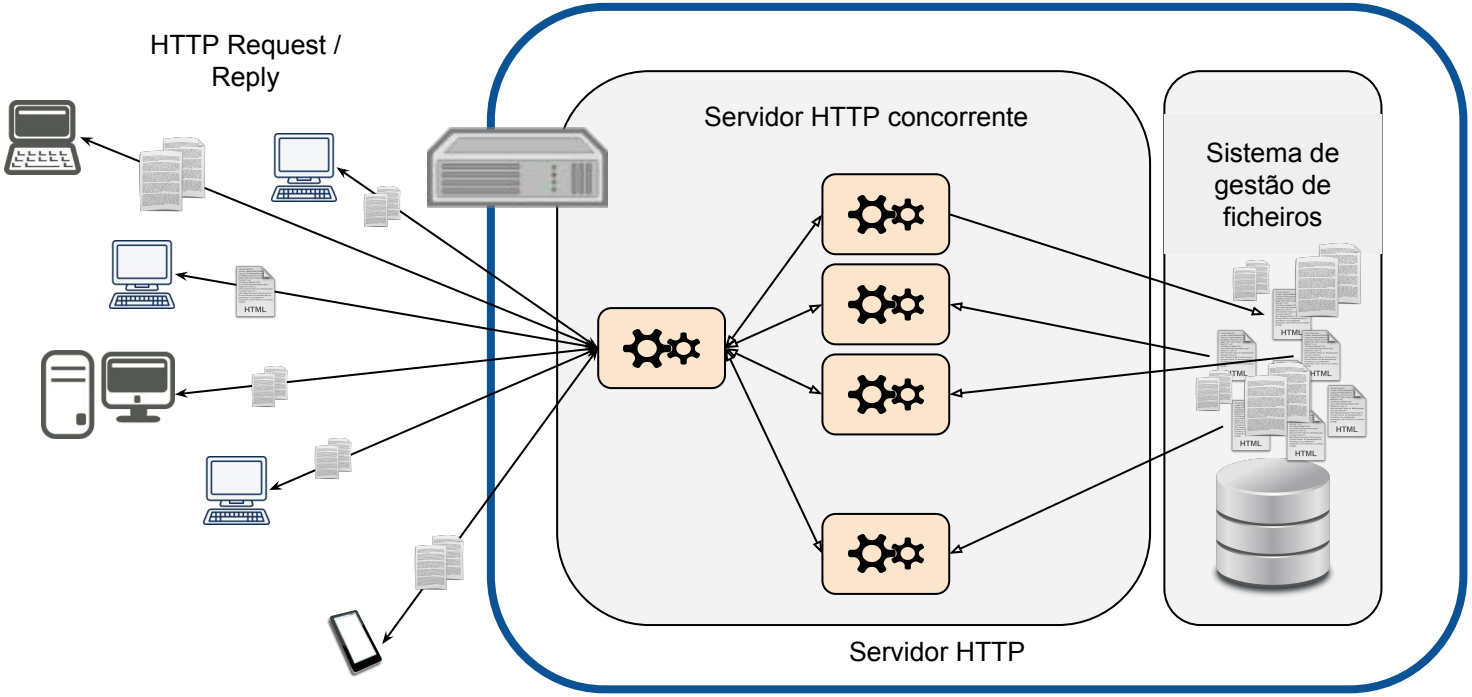
Outra alternativa: Conexões Persistentes

- Múltiplas transferências por conexão TCP
  - A conexão TCP não é fechada no fim e é reutilizada
  - O cliente e o servidor podem decidir fechar a conexão mais tarde
  - Opção introduzida com a versão HTTP 1.1
- Vantagens
  - Evita o custo suplementar da abertura e fecho da conexão
  - Permite ao TCP afinar a sua estratégia de controlo da conexão e melhorar o seu desempenho
  - As janelas de controlo da saturação podem ser maiores
- Hipótese suplementar: *pipelining*
  - Enviar múltiplos pedidos antes de receber uma resposta

HTTP 1.0, 1.1 e 1.1 com *pipelining*



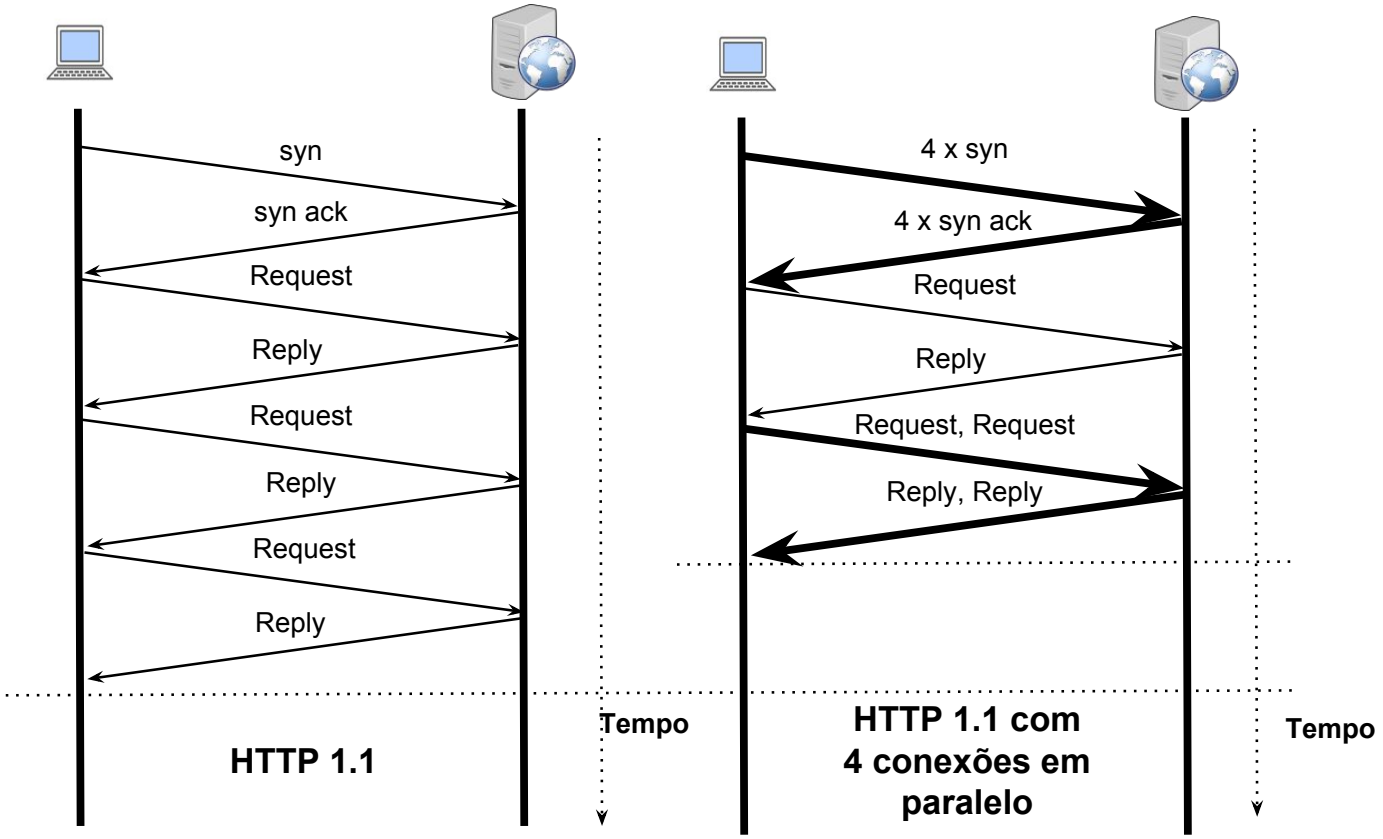
Do Lado do Servidor



Pipelining é pouco usado

- Quando é que é vantajoso usar *pipelining*?
  - Muitos pequenos objetos e tempo igual para servir cada um
  - Neste caso o *pipelining* pouparia o RTT correspondente ao pedido inicial
  - Mas só é possível receber um dado objeto depois dos outros, ora esse segundo objeto pode ser crítico (e.g. código JavaScript)
  - Por outro lado alguns objetos são volumosos e outros são gerados dinamicamente o que ainda leva mais tempo
  - Head-of-Line-Blocking
  - Utilizadores abandonam *sites* lentos
- Solução usada na prática
  - Múltiplas conexões paralelas sem *pipelining* e objetos distribuídos por vários servidores
  - A maioria dos browsers não usa *pipelining*

HTTP 1.1 e conexões paralelas



Caching: porquê e como

- Porque fazer *caching* ?
  - Muitos objetos não mudam (e.g. imagens, javascript, css)
  - Reduz o número de conexões e a carga do servidor
  - E portanto poupa a rede e aumenta a velocidade
- Mas o *caching* também é difícil pois muitos objetos HTTP não são *cachable*
  - Dados dinâmicos: Stock *prices*, resultados dinâmicos de comparações, vídeo em tempo real
  - Resultados de scripts baseados em parâmetros
  - O caching é contra o interesse dos "estudos de mercado"
- Controlo da validade geralmente baseado em estampilhas temporais
  - Dá uma perspetiva grosseira da validade

Caching pelas Aplicações

- As aplicações que usam o HTTP para obter objetos podem fazer caching destes para tornar mais céleres eventuais acessos futuros
- No entanto, como esse caching consome grande quantidade de memória e os objetos evoluem, o caching é delicado
- Principais candidatos a caching (javascript, css, fotografias, botões, ...)
- Principais mecanismos e header fields de suporte
  - If-Modified-Since e Last-Modified
  - Etag e If-None-Match (*e.g.*, ETag: 3d27f9, If-None-Match: 3d27f9)
  - Expires (e.g. Expires -1 == do not cache)
  - Cache-control

Exemplo: *Cache Check Request / Reply*

Pedido:

GET / HTTP/1.1

Accept-Language: en-us

If-Modified-Since: Mon, 29 Jan 2001 17:54:18 GMT

Host: www.example.com

Connection: Keep-Alive

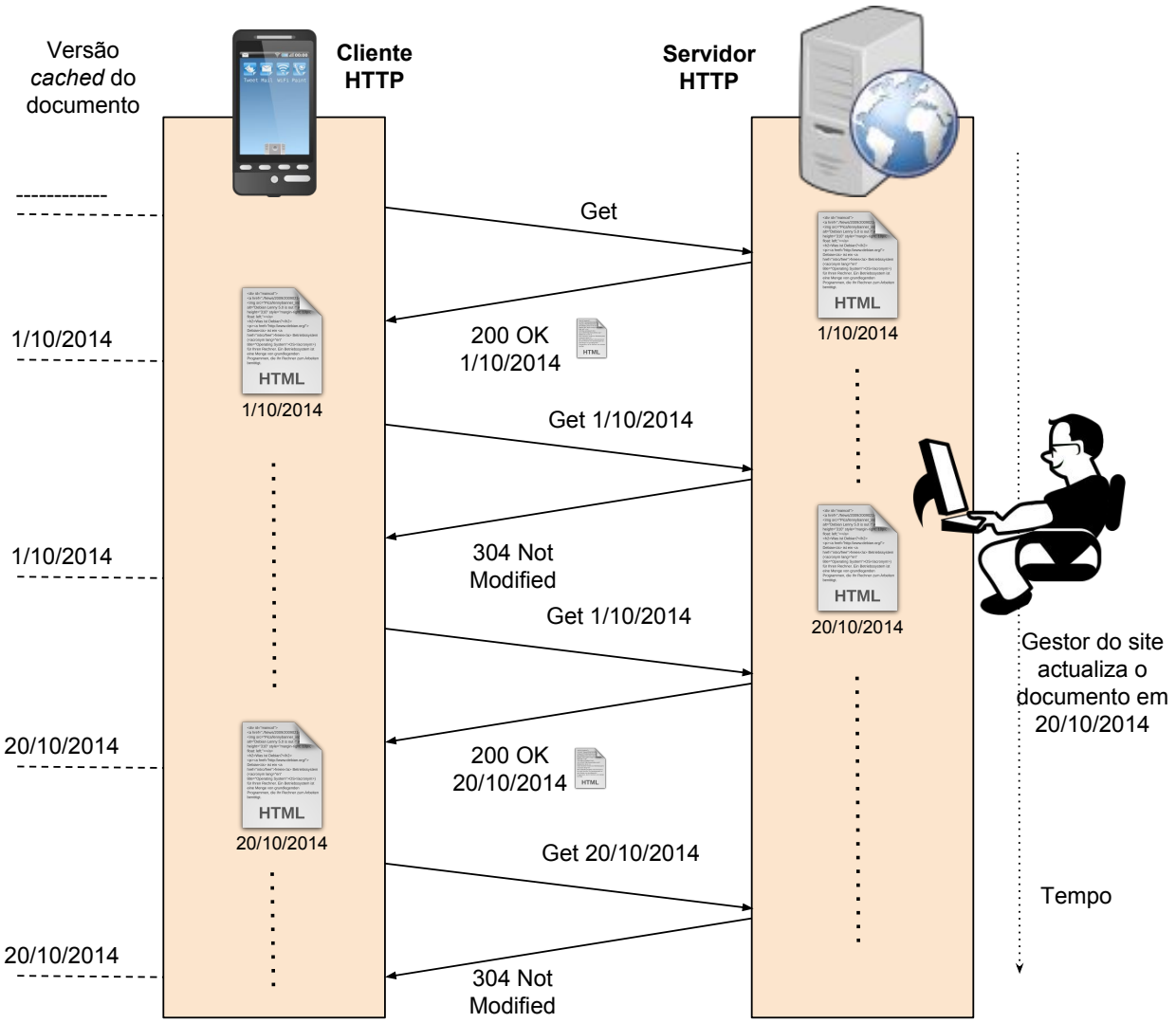
Resposta:

HTTP/1.1 304 Not Modified

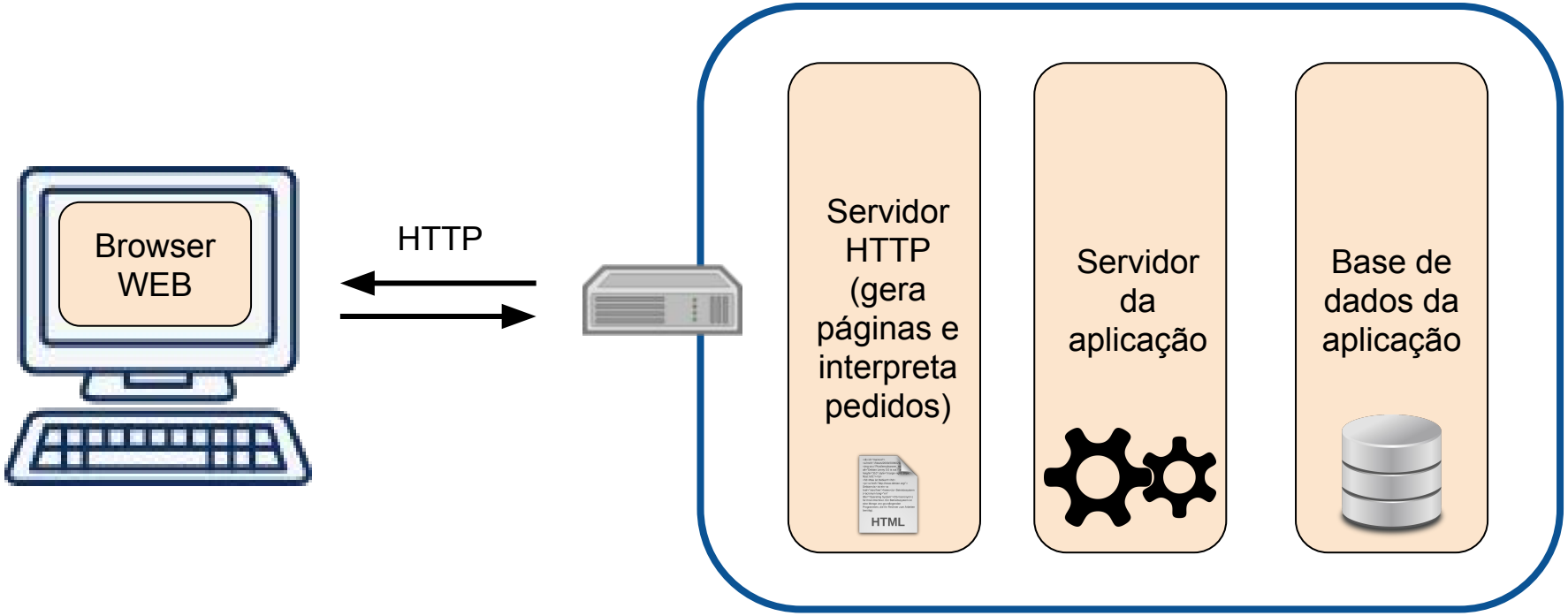
Date: Tue, 27 Mar 2001 03:50:51 GMT

Connection: Keep-Alive

Exemplo

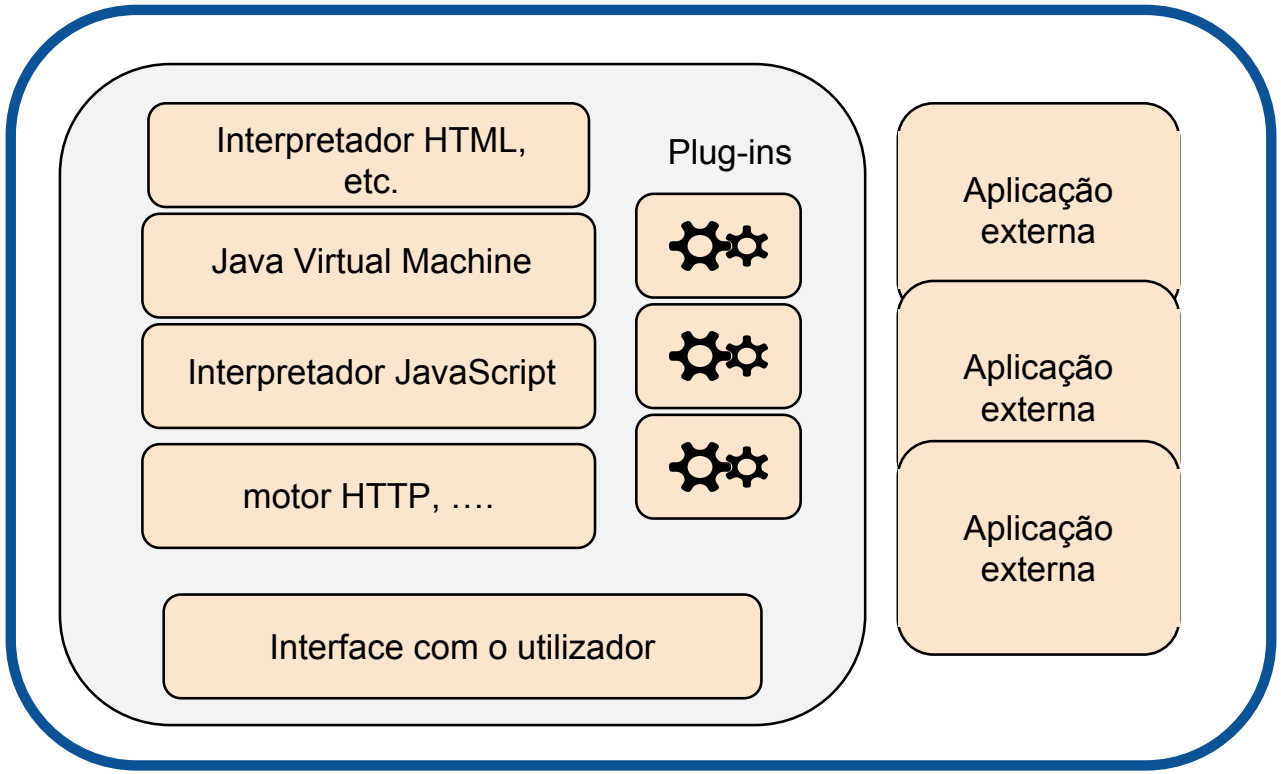

Versão *cached* do documento
Cliente HTTP
Servidor HTTP
Get
200 OK
1/10/2014
HTML
1/10/2014
Get 1/10/2014
304 Not Modified
Get 1/10/2014
200 OK
20/10/2014
HTML
20/10/2014
Get 20/10/2014
304 Not Modified
1/10/2014
1/10/2014
20/10/2014
20/10/2014
HTML
1/10/2014
HTML
20/10/2014
Gestor do site actualiza o documento em 20/10/2014
Tempo

O HTTP e a WEB Atual (Web 2, 3, 4, 5, ...☺)



Arquitetura *three tier*

Um Browser WEB



é uma peça de software muito complexa

Passagem de Parâmetros

- Método *POST*
  - Algumas vezes a página WEB contém um *form* para entrada de dados
  - O método *post* permite o envio das respostas do utilizador através do *body* da mensagem
- Através do URL
  - O recurso do lado do servidor pode identificar um recurso de código a executar pelo servidor (a resposta é portanto calculada e dinâmica)
  - O *path* desse recurso pode ser complementado com parâmetros a passar ao código executável
- Outros métodos (ver a seguir)

Passagem de Parâmetros

Sintaxe geral de um URL com o nome de uma operação e parâmetros:

https ou http://servidor[:porta]/operação[?parâmetros]

em que a parte parâmetros, quando presente, toma a forma de um ou mais pares (nome = valor) separados pelo caracter '&' e com a forma genérica:

parâmetro=valor{&parâmetro=valor}*

Exemplos:

http://www.search.com/search?language=english&q=http+tutorial
https://www.socialnet.com/profile.php?id=100014350098345

Autenticação

Pedido do cliente:

```
GET /secretpage HTTP/1.1 Host: www.httpspy.com
```

Resposta do servidor:

```
HTTP/1.1 401 Access Denied
WWW-Authenticate: Basic realm="Info for spies"
Content-Length: 0
```

O valor a seguir a realm é opcional e é apenas usado pelos servidores que pretendem indicar um domínio de proteção (*protection space*) ao cliente. Quando recebe a resposta, o *browser* Web abre uma caixa pedindo ao utilizador um nome de utilizador e uma palavra passe, e deve depois reenviar o pedido mas incluindo um *header field* Authorization.

Autenticação

Novo pedido do cliente:

GET /secretpage HTTP/1.1

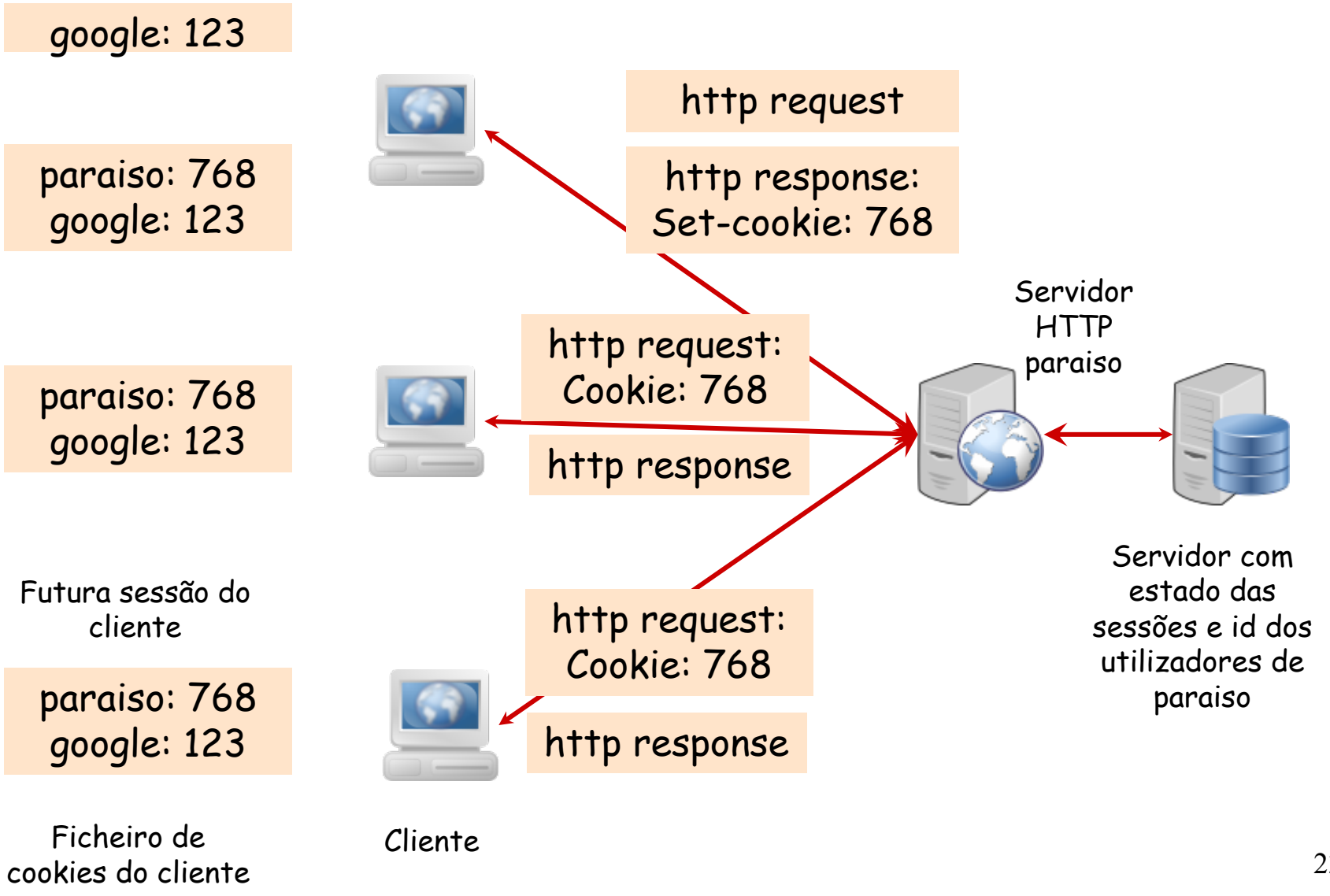
Host: www.httpspy.com

Authorization: Basic aHR0cHdhdGNoOmY=

onde "aHR0cHdhdGNoOmY=" representa "utilizador:palavra-passe" codificado num formato especial na base de uma sequência de caracteres US-ASCII e designado por Base64

Se a conexão não for cifrada (https:), o nome e a palavra passe estão em claro e podem ser copiados.

Acesso com Cookies



Funcionamento dos Cookies

Pedido do cliente:

GET /compras HTTP/1.1
Host: www.paraiso.pt

Resposta do servidor:

HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: session=1274126492323
Set-Cookie: userToken=user1412610421; Expires=Wed, 02 Jun 2021 22:18

.....

Pedido seguinte do cliente:

GET /compras HTTP/1.1
Host: www.paraiso.pt
Cookie: session=1274126492323 userToken=user1412610421

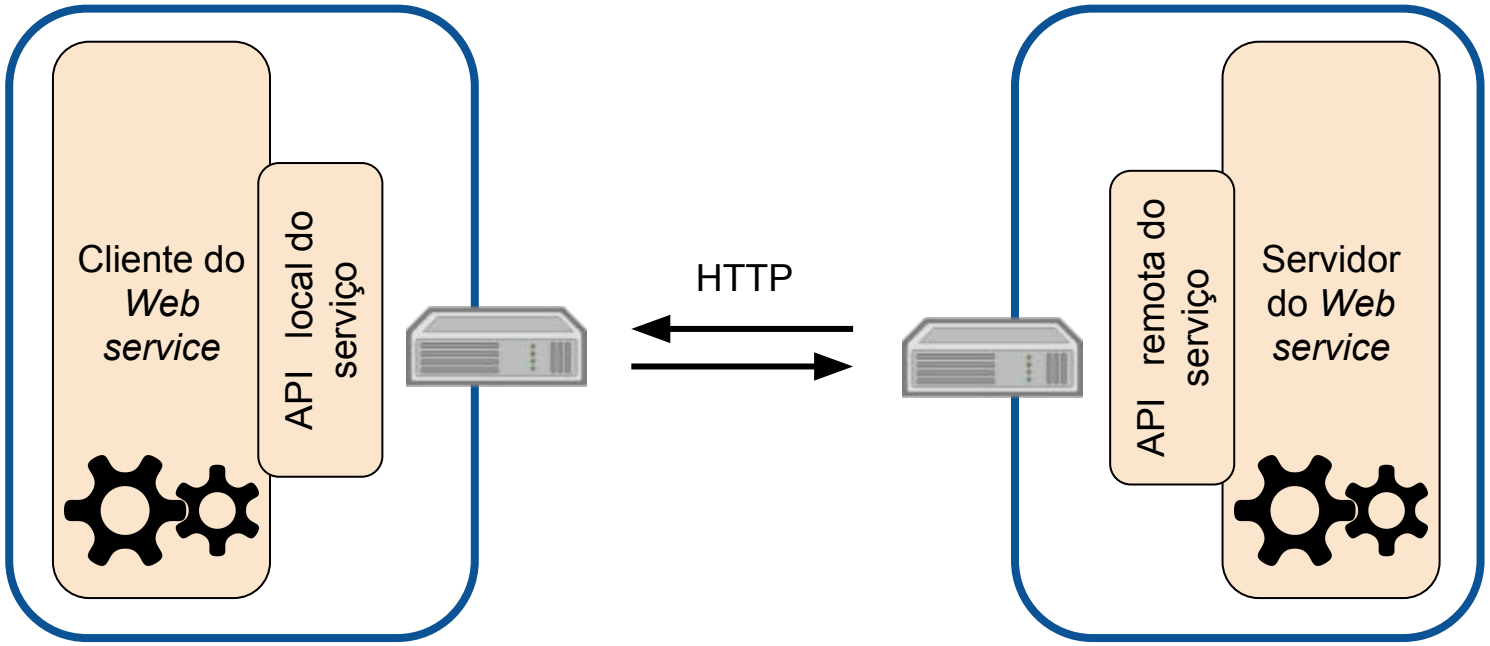
Cookies

- Os ***cookies*** podem ser usados para introduzir uma noção de sessão para seguimento dos utilizadores e para autenticação.

- A sua utilização para o desenvolvimento de aplicações complexas e com interações prolongadas via a Web, *i.e.*, com a noção de sessão (*e.g.*, aquisições e outros serviços envolvendo transações prolongadas via a Web) e suporte de autenticação tornou-se imprescindível.

- No entanto, devido a vários **problemas de segurança e privacidade têm** de ser utilizados de forma cuidadosa.

WEB Services



O HTTP como um transporte genérico

Passagem de Objetos em Parâmetro

- Existem várias linguagens de descrição de valores de objetos como por exemplo XML (independente da linguagem de interpretação) e JSON (JavaScript-based)

```
{
    "id": 1,
    "name": "Porta interior em carvalho",
    "price": 210,
    "tags": ["porta", "casa", "carvalho"]
}
```

Conclusões

- O protocolo HTTP é o principal suporte da WWW (*World Wide Web*)
- É um protocolo cliente / servidor relativamente simples mas muito extensível e que por isso tem sido utilizado para imensas funcionalidades complementares
- Dado ser idempotente ou sem estado (a operação pode repetir-se até ao infinito) e contempla facilmente *caching*, replicação, etc.
- Tem sido utilizado a fundo sendo hoje em dia o transporte mais comum entre aplicações distribuídas