RESUMOS CC

Contextualização do HTTP e da Web:
O HTTP é descrito como o protocolo de aplicação da Web e segue um modelo cliente-servidor. Por um lado, temos o programa cliente (geralmente um browser), que inicia as solicitações (requests); por outro, o servidor Web, que recebe as solicitações e envia as respostas (responses).

Páginas Web, objectos e URLs:
Uma página Web é vista como um documento que pode consistir num ficheiro HTML base (o "base HTML file") e vários objectos adicionais, tais como imagens JPEG, ficheiros CSS, ficheiros JavaScript ou vídeos. Cada objecto é identificável e recuperável através de um URL único, composto normalmente pelo nome do host (por ex., www.someSchool.edu) e pelo caminho do recurso no servidor (por ex., /someDepartment/picture.gif). Quando um browser obtém a página HTML base, este irá analisá-la (fazer o parsing) e detectar a referência a outros objectos, necessitando subsequentemente de obter todos esses objectos referenciados através de pedidos individuais ao servidor.

Arquitectura cliente-servidor e comunicação sobre TCP:
O HTTP assenta no protocolo de transporte TCP. Quando um cliente deseja aceder a uma página, o seu browser inicia uma ligação TCP para o servidor HTTP, tipicamente na porta 80. Após o estabelecimento da ligação TCP, o browser (cliente) envia uma mensagem de pedido HTTP (HTTP request). O servidor, ao receber o pedido, envia de volta uma mensagem de resposta (HTTP response) que inclui o objecto solicitado. Assim, cada mensagem HTTP é entregue intacta graças à camada de transporte confiável do TCP. A natureza confiável do TCP garante que os dados cheguem sem perda, a menos que a conexão seja interrompida, e o HTTP não necessita de se preocupar com problemas de retransmissão ou reordenação; estas são questões tratadas pelos protocolos subjacentes.

Propriedade "Stateless" do HTTP:
Um aspecto fundamental do HTTP é a sua característica "stateless", ou seja, o servidor não retém informação sobre clientes entre pedidos consecutivos. Cada pedido é tratado de forma independente. Por exemplo, se um cliente pede o mesmo ficheiro várias vezes num curto intervalo de tempo, o servidor envia o ficheiro como se não tivesse qualquer memória dos pedidos anteriores. Isto simplifica muito o desenho do servidor, mas transfere a responsabilidade de gerir estados, sessões ou preferências para outras camadas ou mecanismos (como cookies, gestão de sessão, etc.).

Conexões Não-Persistentes e Persistentes:
Há uma explicação pormenorizada sobre um tópico fundamental: a diferença entre conexões não-persistentes (non-persistent) e persistentes (persistent) no HTTP.

- **HTTP com Conexões Não-Persistentes (como no HTTP/1.0)**:

Neste modelo, cada pedido/resposta é trocado sobre uma nova ligação TCP. Ou seja, se uma página HTML base contém 10 imagens, para obter todos os 11 objectos (1 HTML + 10 imagens) serão estabelecidas 11 ligações TCP distintas. Cada ligação envolve o overhead de três fases: (1) estabelecimento da ligação TCP, (2) envio do pedido e recepção da resposta, e (3) encerramento da ligação. Isto torna o processo ineficiente, aumentando significativamente o tempo total necessário para carregar a página, pois cada ligação requer um "handshake" de 3 vias TCP, e uma eventual latência adicional (RTT – Round Trip Time) para cada objecto.

- **HTTP com Conexões Persistentes (como padrão no HTTP/1.1)**:

Com ligações persistentes, a mesma ligação TCP pode ser utilizada para transferir múltiplos objectos. Assim que o cliente estabelece uma única ligação TCP ao servidor e obtém a página HTML base, pode reutilizar a mesma ligação para pedir as imagens e outros objectos sem ter de fechar e reabrir novas ligações TCP. Isto reduz substancialmente o overhead, diminui a latência e melhora o desempenho global do carregamento de páginas complexas com múltiplos recursos.

HTTP não persistente:	**HTTP persistente:**
• Só pode ser enviado no máximo um objeto *web* por cada conexão estabelecida. • O HTTP/1.0 utiliza HTTP não persistente. • Mínimo de 2 RTT/objeto. • Exige mais recursos do S.O. • Alguns browsers abrem várias conexões TCP em paralelo para pedirem vários objetos referidos no mesmo objeto.	• Podem ser enviados múltiplos objetos *web* por cada ligação estabelecida entre o cliente e o servidor. • O HTTP/1.1 usa, por defeito, conexões persistentes. • Servidor mantém conexão TCP aberta. • Com ou sem estratégia de *pipelining*.

Impacto no Tempo de Carregamento (RTT):
O texto salienta a importância do round-trip time (RTT) para estimar o atraso na obtenção dos recursos. Com conexões não-persistentes, o RTT necessário acumula-se a cada objecto solicitado, enquanto com conexões persistentes, o impacto do RTT é amortizado, dado que a mesma ligação é mantida e reutilizada.

RTT – tempo necessário para que um pacote viaje do cliente até ao servidor e volte ao cliente.

$$RTT = T_{propagação} + T_{fila} + T_{processamento}$$

O comando ping já calcula o RTT.

Controlo de Paralelismo pelo Browser:
Embora o texto não entre em pormenores extensos, menciona a possibilidade do browser estabelecer múltiplas conexões TCP em paralelo para obter vários objectos simultaneamente. Esta abordagem é usada pelos browsers modernos para acelerar o carregamento da página, pois obtém múltiplos objectos em paralelo, reduzindo o tempo global até o utilizador ver a página completa.

Conexões Não-Persistentes vs. Persistentes:
Não-Persistentes: Cada objecto solicitado exige:

- ✓ Estabelecimento da ligação TCP (3-way handshake, consumindo cerca de 1 RTT).
- ✓ Envio do pedido e recepção da resposta (outro RTT para a resposta chegar após o pedido).

Consequentemente, cada objecto implica aproximadamente 2 RTTs mais o tempo de transmissão do ficheiro. Para múltiplos objectos, isso acumula um atraso considerável.
Persistentes: A partir do HTTP/1.1, as ligações persistentes permitem que múltiplos objectos sejam transferidos numa mesma conexão TCP. Assim, depois de estabelecida a ligação (1 RTT), os pedidos seguintes para objectos adicionais no mesmo servidor não necessitam de criar novas ligações, reduzindo drasticamente o atraso. Além disso, o HTTP/1.1 introduz o pipelining, possibilitando ao cliente enviar vários pedidos em sequência sem esperar pela resposta do primeiro, aumentando a eficiência.

Imagens e Ilustrações:

1. **Figura 2.6 (HTTP request-response behavior)**:

Esta figura mostra a interacção entre um PC (com Internet Explorer) e um servidor Apache, bem como um smartphone Android (com Google Chrome), ilustrando a relação de pedidos e respostas. As setas destacam como, após o estabelecimento da ligação, o cliente envia um pedido HTTP e o servidor responde com o objecto solicitado. Esta ilustração ajuda a entender a natureza cliente-servidor do HTTP, enfatizando que múltiplos clientes podem simultaneamente interagir com o mesmo servidor.

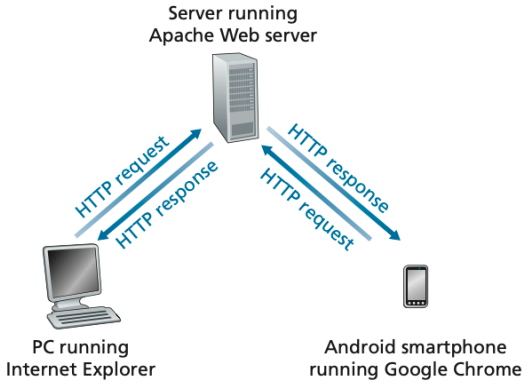

Figure 2.6 ♦ HTTP request-response behavior

2. **Figura 2.7 (Cálculo aproximado do tempo de resposta)**:

Esta figura apresenta uma linha temporal que destaca o tempo envolvido em:

- Estabelecer a ligação TCP (que requer um RTT devido ao "three-way handshake").
- Enviar o pedido para o ficheiro HTML (incluindo o pedido no terceiro segmento do handshake).
- Receber a resposta, que envolve mais um RTT para que o servidor responda e transfira o ficheiro.

A figura sublinha que o tempo total para obter o ficheiro HTML consiste em aproximadamente 2 RTTs (um para a ligação, outro para o pedido-resposta em si) mais o tempo de transmissão do ficheiro do servidor para o cliente.

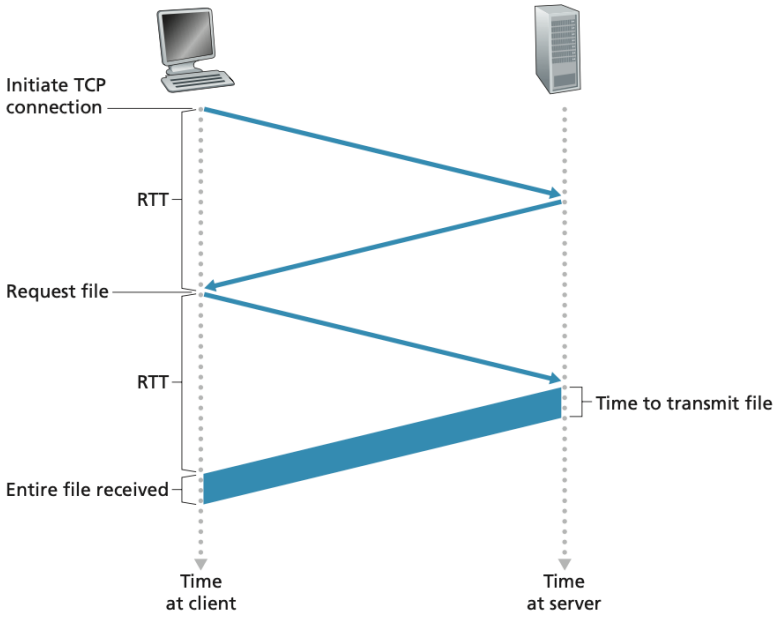

Figure 2.7 ♦ Back-of-the-envelope calculation for the time needed to request and receive an HTML file

3. **Figura 2.8 (Formato geral de uma mensagem HTTP de pedido)**:

Esta figura mostra a estrutura típica de um pedido HTTP:

- **Request line (linha de pedido)**: Contém o método (por ex., GET), o URL do objecto solicitado e a versão do HTTP.
- **Header lines (linhas de cabeçalho)**: Seguem a linha de pedido e fornecem informações adicionais, tais como o host, o tipo de cliente (User-agent), preferências linguísticas (Accept-language), ou se a ligação deve ser mantida aberta ou fechada (Connection).
- **Blank line (linha em branco)**: Separa os cabeçalhos do corpo da entidade.
- **Entity body (corpo da entidade)**: Em pedidos GET normalmente está vazio, mas em pedidos POST contém dados (por exemplo, dados introduzidos num formulário).

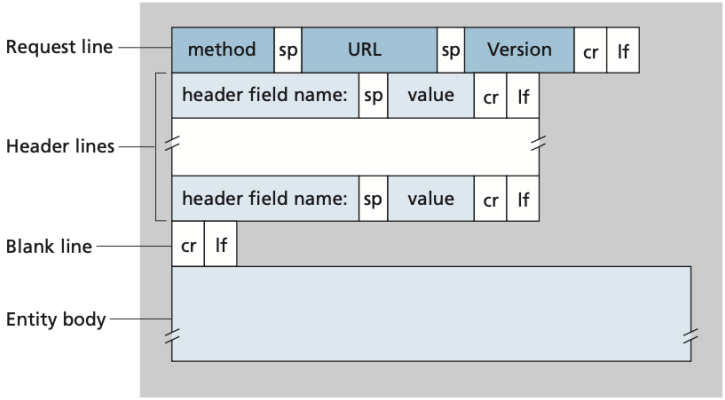

Figure 2.8 ♦ General format of an HTTP request message

Métodos HTTP (GET, POST, HEAD, PUT, DELETE):

- **GET**: Recupera dados do servidor, com o objecto identificado no URL.
- **POST**: Envia dados para o servidor no corpo do pedido (por exemplo, dados de um formulário).
- **HEAD**: Solicita o cabeçalho da resposta sem o corpo do objecto, útil para depuração ou verificar se um objecto existe.
- **PUT**: Permite enviar (fazer "upload") de um objecto para um servidor, armazenando-o num dado caminho.
- **DELETE**: Permite apagar um objecto armazenado no servidor.

Mensagens de Resposta HTTP:
Uma resposta HTTP inclui:

- Uma *status line* (linha de estado) com a versão do HTTP, um código numérico (por ex., 200 para OK, 404 para Not Found) e uma mensagem associada.
- Vários *header lines* com meta-informação (tipo de servidor, data, tamanho do objecto, tipo de conteúdo, etc.).
- O *entity body*, que contém o objecto solicitado, como o ficheiro HTML ou outro recurso.

A resposta típica seria algo como:

HTTP/1.1 200 OK

Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

(data data data data ...)

Esta mensagem mostra que o servidor encontrou o objecto, está a usar HTTP/1.1, o objecto foi modificado pela última vez em determinada data, o seu tamanho e tipo. Após os cabeçalhos e a linha em branco, surge o corpo da resposta com o próprio ficheiro (dados).

Formato Geral das Mensagens de Resposta (Figura 2.9):
A Figura 2.9 mostra o formato típico de uma mensagem de resposta HTTP e segue a mesma lógica estrutural das mensagens de pedido, porém adaptada ao contexto de resposta:

- **Status line (linha de estado)**: É a primeira linha da resposta HTTP e contém três elementos:
  1. A versão do HTTP (por ex. HTTP/1.1).
  2. O código de estado (um número inteiro, como 200, 301, 404, etc.).
  3. Uma frase ou mensagem textual explicando o significado do código de estado (por ex. “OK” para 200, “Not Found” para 404).

Este código e a frase fornecem informações importantes ao cliente sobre o resultado do pedido. Por exemplo:

- **200 OK**: O pedido foi bem-sucedido e o objecto solicitado é devolvido no corpo da mensagem.
- **301 Moved Permanently**: O objecto pedido mudou permanentemente de URL; o cliente deve usar o novo URL fornecido num cabeçalho *Location*.
- **400 Bad Request**: O pedido enviado pelo cliente não é compreensível pelo servidor.
- **404 Not Found**: O objecto solicitado não existe no servidor.
- **505 HTTP Version Not Supported**: A versão do HTTP usada pelo cliente não é suportada pelo servidor.
- **Header lines (linhas de cabeçalho)**: Após a linha de estado, surgem linhas opcionais de cabeçalho, que fornecem metainformações sobre a resposta e o objecto, tais como:
- **Connection**: Pode indicar se a conexão será fechada após enviar a resposta.
- **Date**: Data e hora em que o servidor enviou a resposta.
- **Server**: Identifica o software do servidor que gerou a resposta (similar ao *User-agent* do lado do cliente).
- **Last-Modified**: Indica quando o objecto no servidor foi criado ou alterado pela última vez, sendo crucial para cache e validação da versão do objecto.

- **Content-Length**: Tamanho do objecto em bytes, útil para o cliente saber quanto dados irá receber.
- **Content-Type**: Tipo MIME do objecto (por ex. text/html), indicando o formato do conteúdo a ser apresentado ou processado pelo cliente.
- **Blank line (linha em branco)**: Uma linha em branco separa as linhas de cabeçalho do corpo da mensagem.
- **Entity body (corpo da entidade)**: Contém os dados do objecto solicitado (por exemplo, o ficheiro HTML, uma imagem, etc.). É a parte "substantiva" da resposta, que o cliente irá interpretar ou apresentar ao utilizador.

A figura ajuda a visualizar essa estrutura hierárquica, mostrando a clara separação entre linha de estado, cabeçalhos, e corpo da entidade.

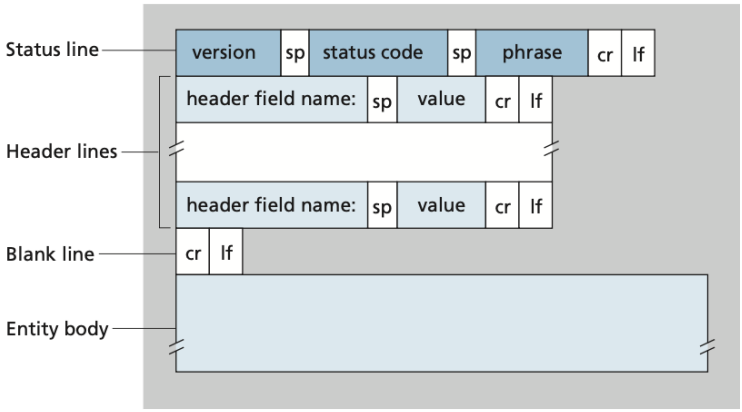

Figure 2.9 ♦ General format of an HTTP response message

Cookies e Manutenção de Estado (Figura 2.10):
A Figura 2.10 introduz o conceito de *cookies*, uma tecnologia fundamental que permite ao HTTP, originalmente sem estado, manter informações sobre o utilizador entre múltiplas interacções e sessões.

O funcionamento dos cookies envolve quatro componentes:

1. **Linha de cabeçalho Set-Cookie na resposta do servidor**:

Quando um utilizador (neste caso "Susan") acede a um site, o servidor pode enviar na resposta um cabeçalho "Set-Cookie" contendo um identificador único (por exemplo, "ID 1678"). Este ID funciona como um rótulo que permite ao servidor reconhecer o utilizador em acessos futuros.

2. **Linha de cabeçalho Cookie nos pedidos subsequentes do cliente**:

O browser, após receber o cookie do servidor, armazena-o localmente (num ficheiro de cookies associado ao domínio em questão). Em pedidos futuros ao mesmo site, o browser inclui um cabeçalho "Cookie" na mensagem de pedido, enviando o ID previamente fornecido. Assim, o servidor reconhece o utilizador. Na figura, mostra-se que depois de Susan receber um cookie do site "amazon" e outro

do “ebay”, nos acessos seguintes, o seu browser envia automaticamente o cabeçalho “Cookie: 1678” (para amazon) ou “Cookie: 8734” (para ebay).

3. **Ficheiro de cookies no lado do cliente**:

Este ficheiro é mantido pelo browser e guarda os cookies recebidos. O ficheiro é persistente e mantido no computador do utilizador. Na ilustração, em cada momento, a informação sobre a origem do cookie (amazon, ebay) e o seu valor (1678, 8734) é armazenada no computador do cliente.

4. **Base de dados no servidor**:

No lado do servidor existe uma base de dados que associa cada ID de cookie às preferências, histórico de navegação, ou sessão do utilizador. Assim, quando o servidor recebe um pedido com um certo cookie, procura o ID na base de dados e recupera a informação relevante, permitindo, por exemplo, personalizar a experiência, manter o utilizador autenticado, ou lembrar o seu carrinho de compras entre acessos sucessivos.

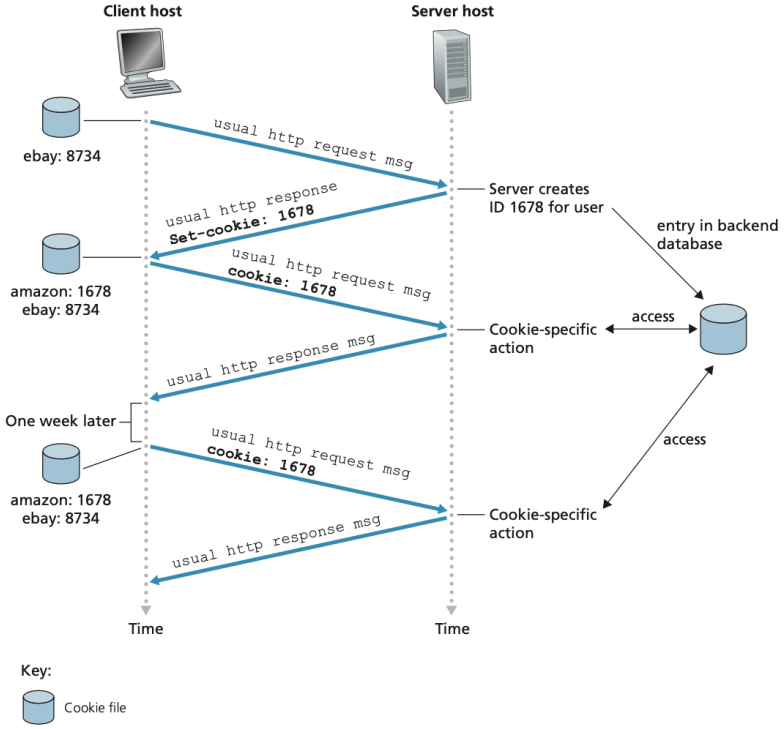

Figure 2.10 ♦ Keeping user state with cookies

A figura exemplifica como, ao longo do tempo (por exemplo, uma semana depois), quando Susan volta ao site, o browser envia novamente o cookie correspondente. O servidor reconhece o ID, recupera a informação do utilizador e mantém a continuidade da experiência, apesar do HTTP em si não guardar estado entre pedidos.

Cookies para Manter Estado do Utilizador:

As imagens e o texto exemplificam como um servidor pode atribuir um ID único ao utilizador através de um cabeçalho “Set-cookie” enviado na resposta HTTP. O browser, ao receber este cabeçalho, armazena o cookie no disco local (num ficheiro específico gerido pelo browser). Em visitas seguintes ao mesmo site, o browser envia o cookie de volta ao servidor num cabeçalho “Cookie” em cada pedido, permitindo ao servidor reconhecer o utilizador, lembrar preferências, manter carrinhos de compras e facilitar autenticações.

- Isto evita que o utilizador tenha de reintroduzir dados pessoais ou reconfigurar preferências a cada visita.
- Por outro lado, a capacidade dos cookies de rastrear a actividade do utilizador levanta questões de privacidade, já que, combinando cookies com dados fornecidos pelo próprio utilizador, um site pode recolher informação muito detalhada sobre os seus padrões de navegação.
- Cookies tornaram-se um elemento essencial do comércio electrónico e da personalização do conteúdo na Web.

Web Caching (ou Proxy Server):
A figura e o texto seguintes abordam a utilização de caches Web, também conhecidos como servidores proxy, para melhorar o desempenho e reduzir tráfego na Internet. Uma cache Web armazena localmente cópias de objectos Web solicitados recentemente pelos utilizadores. Quando um utilizador (browser) pede um objecto:

1. O pedido é enviado primeiro para a cache Web.
2. Se a cache tiver uma cópia atualizada do objecto, responde imediatamente, acelerando a entrega e reduzindo o tempo de resposta.
3. Caso contrário, a cache contacta o servidor de origem, obtém o objecto, armazena-o localmente e envia-o para o utilizador.

Esta aproximação traz vários benefícios:

- **Melhora do tempo de resposta**: Uma ligação entre o utilizador e a cache normalmente é mais rápida e com maior largura de banda do que a ligação até ao servidor de origem, especialmente se este estiver distante ou ligado por canais mais lentos.
- **Redução de tráfego e custos**: Ao entregar objectos populares a partir da cache, reduz-se a quantidade de tráfego que atravessa o link de acesso à Internet. Isso diminui a carga na rede da instituição (por exemplo, uma universidade ou uma empresa) e reduz o custo e a necessidade de aumentar a largura de banda disponível.
- **Escalabilidade da Web**: Quanto mais objectos podem ser servidos a partir das caches distribuídas, menor a carga nos servidores de origem e no backbone da Internet, melhorando a experiência global para todos os utilizadores.

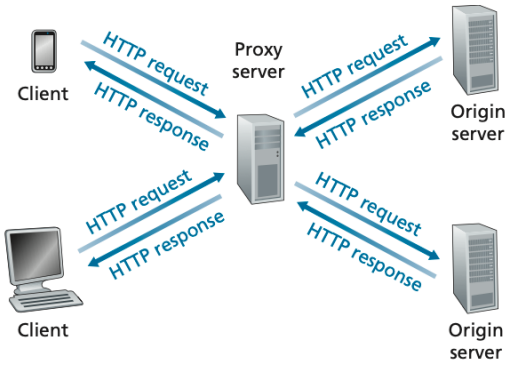

Figure 2.11 ♦ Clients requesting objects through a Web cache

Exemplo Prático (Figura 2.12):

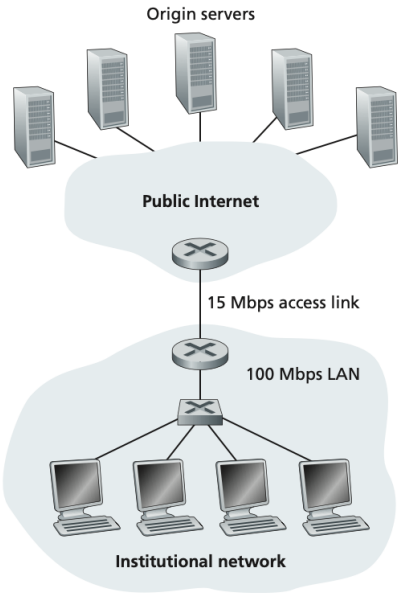

Figure 2.12 ♦ Bottleneck between an institutional network and the Internet

A imagem mostra um cenário típico em que uma instituição (como uma universidade) tem ligações internas de alta velocidade (LAN) e um link de acesso mais lento à Internet (por exemplo, 15 Mbps), enquanto os objectos Web têm um tamanho considerável (por exemplo, 1 Mbit). Quando o tráfego se intensifica, a ligação de acesso à Internet torna-se um estrangulamento (gargalo). Uma cache Web instalada no lado da instituição pode servir muitos pedidos de objectos populares sem precisar requisitá-los repetidamente ao servidor de origem. Isto reduz a intensidade de tráfego na ligação de acesso, evita congestionamentos e diminui o atraso no fornecimento de objectos aos utilizadores.

No geral, estas imagens e a explicação destacam como cookies e caches abordam dois aspectos críticos da Web moderna: a personalização e manutenção de estado do lado do cliente (cookies) e a optimização de desempenho e redução de tráfego através do armazenamento temporário de conteúdo próximo do utilizador (caches Web). Estas estratégias ajudam a contornar as limitações inerentes ao HTTP puro e à largura de banda disponível, garantindo uma experiência mais rápida, personalizada e eficiente.

Caching e Melhoria da Eficiência:
A figura 2.12 ilustra o cenário de uma rede institucional que acede à Internet por meio de uma ligação com determinada largura de banda (por exemplo, 15 Mbps). Dado um determinado padrão de pedidos (requests) – por exemplo, 15 pedidos por segundo, cada objecto com cerca de 1 Mbit –, é demonstrado como a "intensidade de tráfego" (ou razão entre a taxa de pedidos e a capacidade do link) pode aproximar-se de 1. Quando isso acontece, o atraso torna-se extremamente elevado, chegando a minutos, o que é inaceitável do ponto de vista do utilizador. É aqui que surgem duas soluções apresentadas:

1. **Aumentar a Largura de Banda do Acesso (de 15 Mbps para 100 Mbps)**:

Esta solução técnica reduz drasticamente a intensidade de tráfego no link de acesso, baixando-a de 1 para 0,15. Com isto, o atraso torna-se reduzido e praticamente insignificante face ao atraso inerente da própria Internet. Contudo, esta abordagem implica custos avultados, dado que significa melhorar a infraestrutura física, adquirir novas linhas e equipamentos, o que pode ser pouco viável economicamente.

2. **Instalar uma Cache Web Interna (Figura 2.13)**:

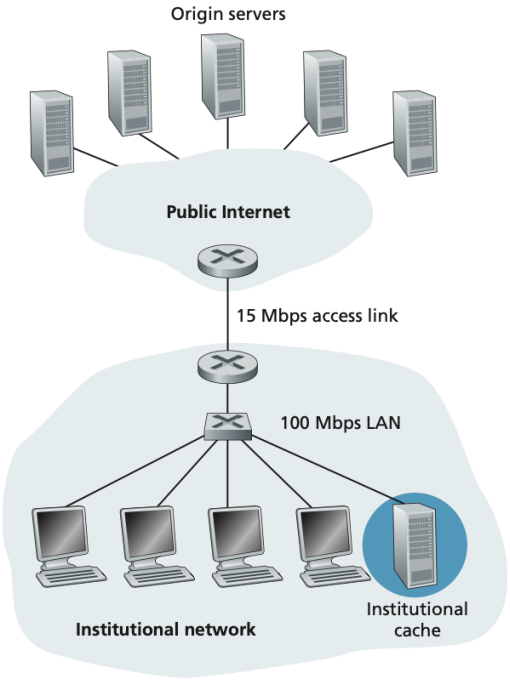

Figure 2.13 ♦ Adding a cache to the institutional network

A figura mostra que, em alternativa a um upgrade dispendioso, a instituição pode instalar uma cache Web no interior da sua rede. Supondo um *hit rate* (taxa de acertos) de 0,4, isto significa que 40% dos pedidos são satisfeitos imediatamente pela cache (em milissegundos), a partir da LAN de alta velocidade. Apenas 60% dos pedidos seguem para o servidor de origem pela ligação de 15 Mbps. Assim, o

tráfego que chega ao link de acesso é reduzido, por exemplo, de uma intensidade de 1 para 0,6, o que já é suficiente para diminuir consideravelmente os atrasos. Nesta abordagem, o custo é inferior (um simples cache pode ser um servidor com software de domínio público) e melhora significativamente a experiência do utilizador, sem o grande investimento de aumentar a largura de banda.
Desta forma, a cache age como um elemento intermediário:

- Do ponto de vista do cliente, a cache é um servidor rápido, oferecendo objectos populares de imediato.
- Do ponto de vista da origem (o servidor principal), a cache actua como um cliente que obtém objectos para depois distribuí-los localmente, reduzindo pedidos subsequentes à fonte original.

Distribuição de Conteúdo (CDNs):
O texto menciona também as redes de distribuição de conteúdo (CDNs), outro mecanismo para melhorar a escalabilidade e o desempenho. CDNs são constituídas por múltiplas caches distribuídas geograficamente que armazenam objectos populares. Deste modo, um pedido ao servidor "original" pode ser redireccionado para uma cache CDN próxima do utilizador, diminuindo atrasos e reduzindo tráfego na Internet global.

Conditional GET e Validação de Conteúdo Cacheado:
Mesmo com caches, surge um problema: o objecto armazenado na cache pode ter sido alterado no servidor de origem desde a última vez que foi guardado. Para resolver isto, o HTTP introduz o *Conditional GET*, um mecanismo para verificar se o objecto ainda está actual. A lógica é a seguinte:

1. **Armazenamento da Data de Última Modificação**:

Quando a cache obtém um objecto do servidor, recebe um cabeçalho *Last-Modified* com a data/hora da última modificação do objecto.

2. **Pedido Condicional (If-Modified-Since)**:

Quando a cache recebe um pedido subsequente para o mesmo objecto, em vez de enviar directamente a cópia que possui, pode verificar se está actualizada. Envia um pedido GET com o cabeçalho *If-Modified-Since*, datado da última modificação conhecida. Assim, a cache diz ao servidor: "Envie-me o objecto apenas se tiver sido modificado desde a data X".

3. **Resposta do Servidor**:

- Se o objecto não mudou, o servidor responde com "304 Not Modified" (sem enviar o objecto novamente). A cache pode então entregar ao cliente a sua cópia local.
- Se o objecto foi alterado, o servidor envia o objecto actualizado.

Esta estratégia evita a retransmissão desnecessária de grandes ficheiros não alterados, poupando largura de banda e reduzindo o tempo de resposta percebido pelo utilizador.

HTTP/2:
Por fim, o texto apresenta o HTTP/2 (RFC 7540), a primeira nova versão substancial do HTTP desde o HTTP/1.1. O HTTP/2 traz várias melhorias:

- **Multiplexação de Pedidos/Respostas numa Única Ligação TCP**:

Em vez de ter que gerir múltiplas ligações ou ter um pipeline complexo, o HTTP/2 permite que múltiplos pedidos/respostas sejam transmitidos simultaneamente na mesma ligação, resolvendo problemas de head-of-line blocking do HTTP/1.1.

- **Priorização de Pedidos e Push do Servidor**:

O servidor pode enviar objectos para o cliente antes mesmo de serem explicitamente solicitados (server push), antecipando as necessidades do cliente e reduzindo atrasos.

- **Compressão de Cabeçalhos**:

O HTTP/2 comprime os cabeçalhos, que se repetem frequentemente, reduzindo o volume de dados transferidos.

Essas inovações não alteram os métodos, códigos de estado ou campos de cabeçalho do HTTP, mas sim a forma de transporte dos dados, tornando a Web mais rápida, responsiva e eficiente.

HTTP/2 e as suas Inovações:
Nas imagens e secções sobre HTTP/2, é dado ênfase ao problema do “Head of Line (HOL) blocking”, que ocorre quando se usa HTTP/1.1 com uma única ligação TCP persistente para transferir múltiplos objectos de uma página Web. Neste cenário, um objecto volumoso (por exemplo, um vídeo) pode atrasar a entrega de objectos mais pequenos (como ícones ou miniaturas), pois estes ficam “atrás” do objecto de grande dimensão na fila de transmissão. Para mitigar isto, os browsers em HTTP/1.1 frequentemente abrem múltiplas ligações TCP paralelas, aumentando a complexidade e “enganando” os mecanismos de controlo de congestionamento do TCP, já que mais ligações implicam mais largura de banda agregada e prioridade sobre outros fluxos.

As melhorias do HTTP/2, ilustradas e descritas, concentram-se em:

1. **Fracionamento em Frames e Interleaving**:

Cada resposta ou pedido HTTP é dividido em múltiplos frames binários, menores, que podem ser intercalados (interleaved) na mesma ligação TCP. Isto permite enviar partes de objectos grandes e pequenos alternadamente, em vez de esperar o envio total do objecto grande para só depois enviar os menores. O resultado é uma perceção de maior rapidez por parte do utilizador, pois objectos menores chegam quase instantaneamente.

2. **Multiplexação na Mesma Ligação**:

Todos os pedidos/respostas de uma página são transmitidos na mesma ligação, em fluxos separados, evitando a abertura de múltiplas ligações TCP. Com isto, reduz-se o overhead de estabelecer/encerrar várias conexões e permite-se ao TCP fazer controlo de congestionamento de forma mais justa e eficiente.

3. **Prioritização e Server Push**:

O HTTP/2 permite ao cliente indicar a prioridade dos objectos pedidos, tornando possível que o servidor envie primeiro aqueles considerados mais importantes. Além disso, o mecanismo de *server push* permite ao servidor enviar recursos ao cliente antes mesmo de estes serem solicitados explicitamente, antecipando as necessidades com base no conteúdo HTML e evitando RTTs adicionais.

Estas imagens ilustram sobretudo a forma como as mensagens são divididas em frames e como a multiplexação e o interleaving são feitos para contornar os problemas de bloqueio em cascata, melhorando a velocidade e a eficiência global.

HTTP/3 (Breve Alusão):
No final desta parte, o texto menciona que o HTTP/3, baseado em QUIC (um protocolo sobre UDP), irá provavelmente incorporar características do HTTP/2, como multiplexação e compressão de cabeçalhos, mas numa implementação mais avançada e com ainda menor latência. Não há uma figura específica aqui, mas é mencionado como uma evolução natural, ainda em desenvolvimento.