Capítulo 2: Camada de Aplicação

Resumo

Até o momento:

- Visão geral da Internet
- O que é um protocolo?
- Estrutura da Internet/ISP
- Borda da rede, núcleo, rede de accesso
- Comutação de pacotes versus comutação de circuitos
- Desempenho: atraso, perda, vazão
- Arquitetura de camadas e modelos de serviços

Temos agora:

- Contexto, idéia geral, conceitos básicos de redes de computadores
- Iniciaremos agora o estudo detalhado de cada camada

Capítulo 2: Roteiro

- Princípios de aplicações de rede
- A Web e o HTTP
- Correio Eletrônico na Internet
- DNS: o serviço de diretório da Internet
- Aplicações P2P
- Streaming de vídeo e CDN

Capítulo 2: Camada de Aplicação

Metas do capítulo:

1. Discutir os aspectos conceituais e de implementação de protocolos de aplicação em redes
   - modelos de serviço fornecidos pela camada de transporte
   - modelo cliente-servidor
   - modelo *peer-to-peer (P2P)*
2. Aprender sobre protocolos através do estudo de protocolos populares da camada de aplicação:
   - HTTP
   - SMTP/ IMAP
   - DNS
3. Content Distribution Networks

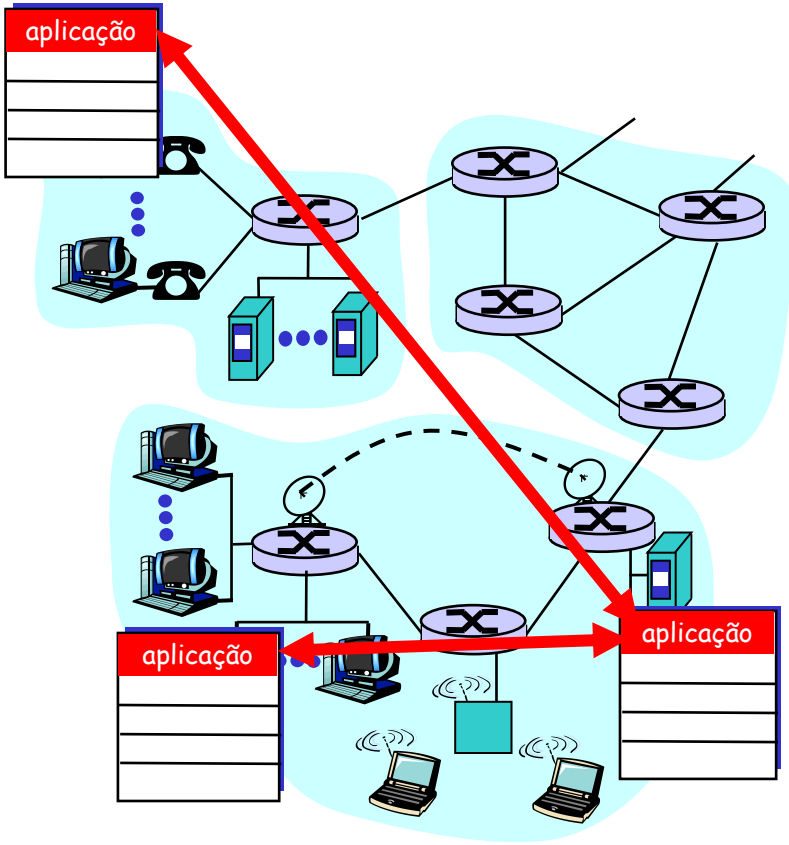
Algumas aplicações de rede

- Correio eletrônico
- Web
- Mensagens instantâneas
- Login em computador remoto como Telnet e SSH
- Compartilhamento de arquivos P2P
- Jogos multiusuários em rede
- *Streaming* de vídeos armazenados (YouTube, Netflix)
- Telefonia por IP (Skype)
- Videoconferência em tempo real
- Busca
- ...
- ...

Criando uma aplicação de rede

Escrever programas que

- Executam em (diferentes) sistemas finais
- Se comunicam através da rede
- p.ex., navegador se comunica com o navegador com um servidor Web



Obs: Esses programas não estão relacionados ao núcleo da rede

- ✓ Dispositivos do núcleo da rede não executam aplicações dos usuários
- ✓ Aplicações nos sistemas finais permite rápido desenvolvimento e disseminação

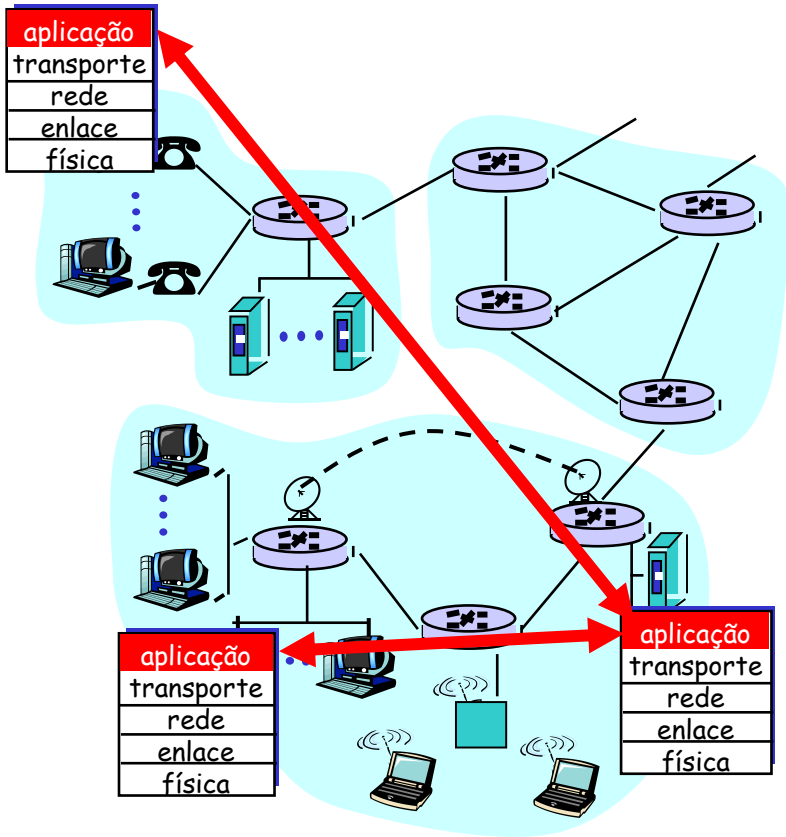
Aplicações e protocolos da camada de aplicação

Aplicação: processos distribuídos se comunicando pela rede

- executam em hospedeiros no "espaço de usuário"
- trocam mensagens para implementar a aplicação

Protocolos da camada de aplicação

- definem o formato das mensagens trocadas pelas apls e ações tomadas
- usam serviços providos por protocolos da camada inferior (TCP, UDP)



Arquiteturas das aplicações de rede

Arquiteturas possíveis das aplicações:

- Cliente-servidor
- Peer-to-peer (P2P)

Arquitetura Cliente-Servidor (C-S)

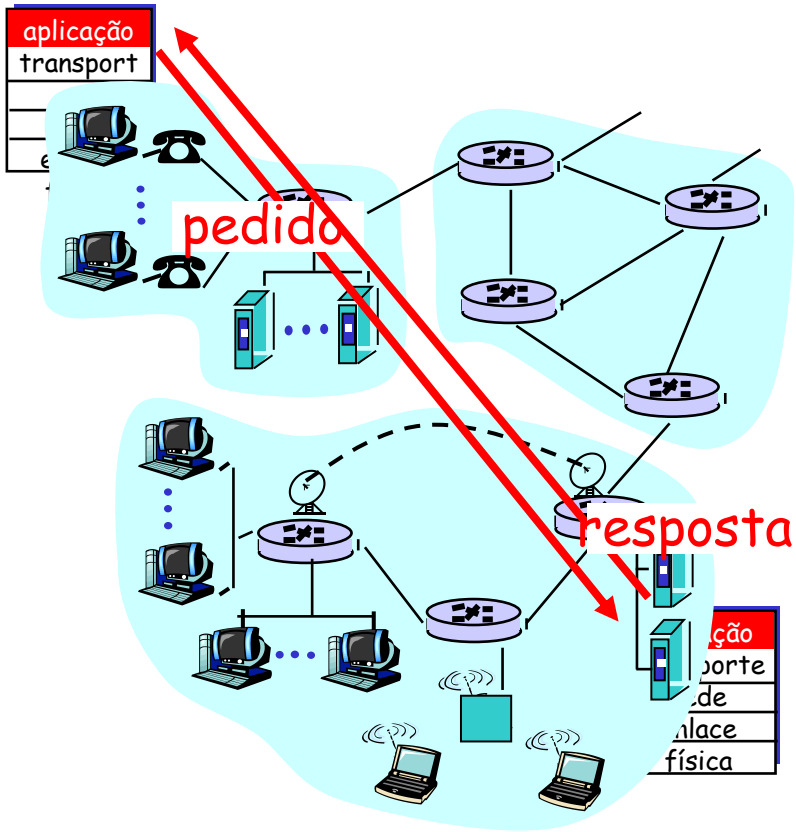
Aplicação de rede típica tem duas partes: ***cliente*** e ***servidor***

Cliente:

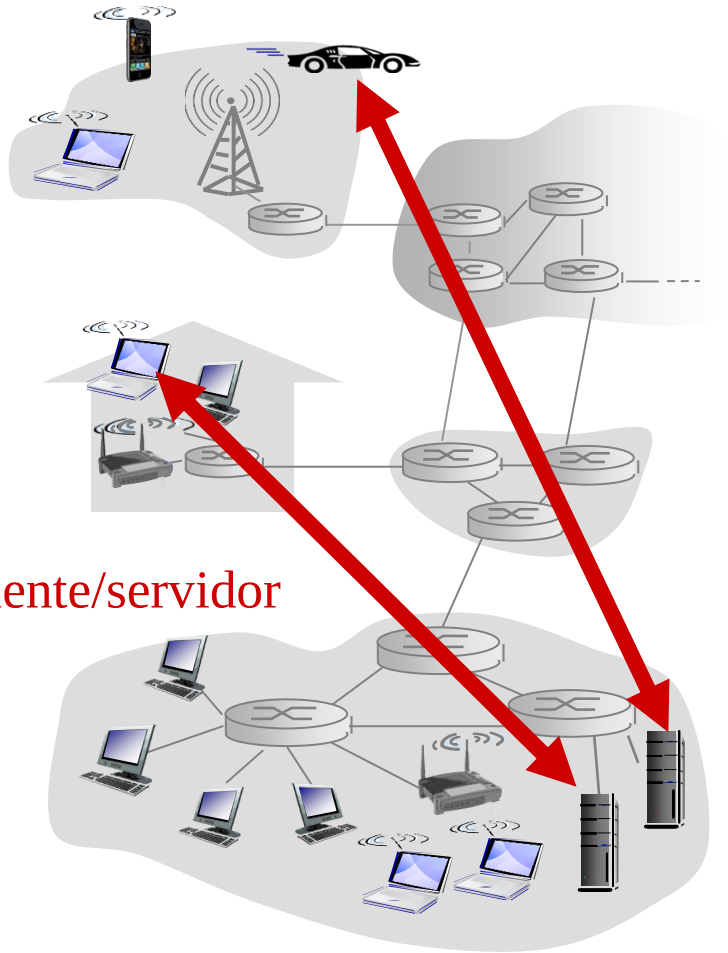
- inicia o contato com o servidor ("fala primeiro")
- tipicamente solicita um serviço ao servidor
- ex: para Web, cliente é implementado no navegador

Servidor:

- provê ao cliente o serviço solicitado
- ex: servidor Web envia página solicitada; servidor de correio entrega mensagens



Arquitetura Cliente-Servidor



Cliente/servidor

Servidor:

- Sempre ligado
- Possuem endereço IP permanente

Clientes:

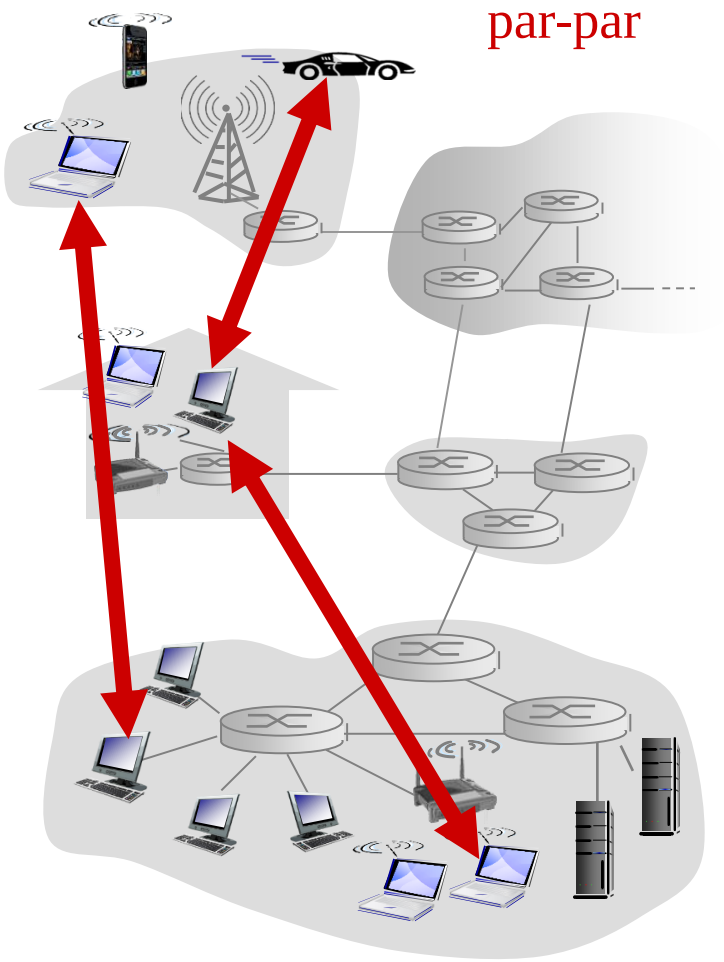
- Podem estar conectados ou não
- Podem ter endereços IP dinâmicos
- Não se comunicam diretamente uns com os outros

Arquitetura P2P

- Não há servidor sempre ligado
- Sistemas finais se comunicam diretamente
- Pares requisitam serviços de outros pares e, em contrapartida, provêem serviços para outros pares

Auto-escalabilidade:

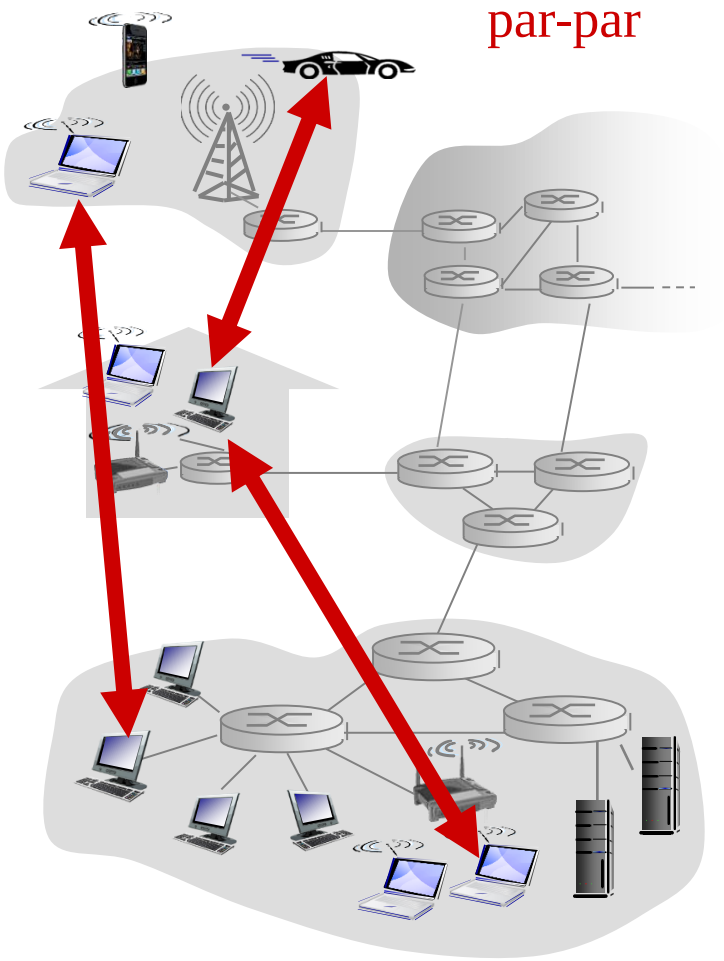
Novos pares trazem nova capacidade de serviço, assim como novas demandas por serviços



Arquitetura P2P

- ✓ Pares estão conectados intermitentemente (nem sempre estão online) e mudam seu endereço IP
- ✓ ⇒ **Gerenciamento mais complexo**
- ✓ As aplicações P2P têm 3 principais desafios:
  - ✓ Segurança
  - ✓ Desempenho
  - ✓ Confiabilidade

Por que? ***Estrutura altamente descentralizada***



Comunicação entre processos

- Um ***processo*** é um programa que executa num host.
- 2 processos no mesmo host se comunicam usando ***comunicação interprocessos*** (definida pelo sistema operacional).
- 2 processos em hosts distintos se comunicam trocando mensagens, usando um ***protocolo da camada de aplicação*****.**

Clientes, Servidores

processo cliente: processo que inicia a comunicação

processo servidor: processo que espera ser contatado

Nota:

aplicações com arquitetura P2P também possuem processos clientes e processos servidores

Comunicação entre processos

- Uma ***aplicação de rede*** consiste em pares de processos que enviam mensagens uns para os outros por meio de uma rede.

- Um processo envia mensagens para a rede e recebe mensagens dela através de uma ***interface de software*** denominada ***socket***.

- Para identificar o processo receptor, duas informações devem ser especificadas:

1. O ***endereço IP do hospedeiro***

2. Um ***identificador*** que especifica o processo receptor no hospedeiro de destino.

Sockets

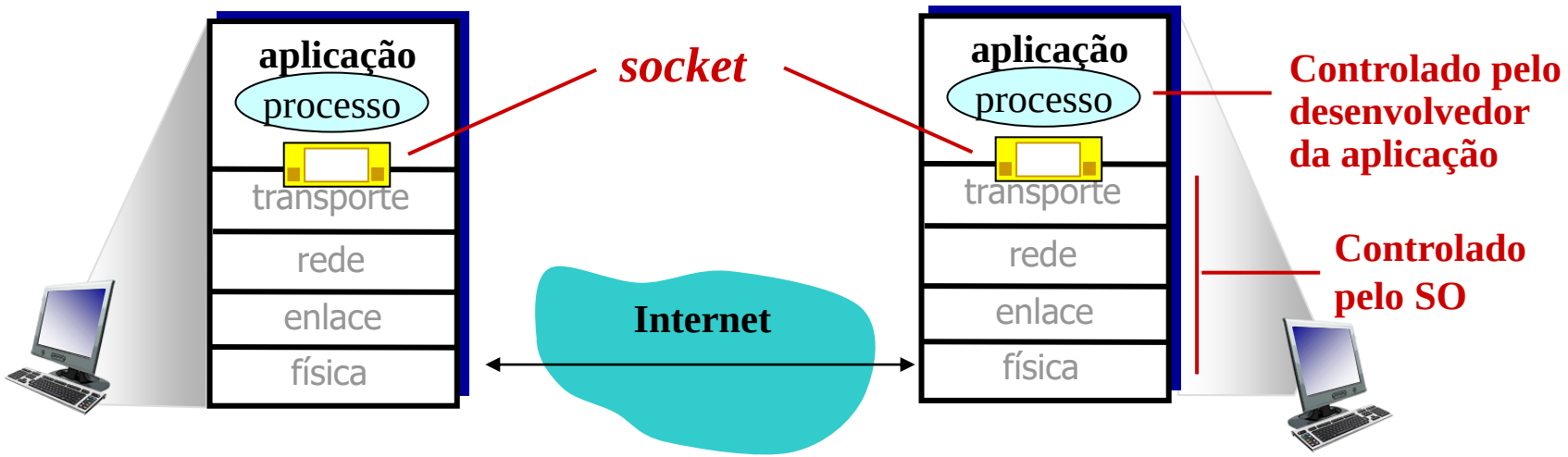
- Definem a interface entre a aplicação e a camada de transporte
- 2 processos se comunicam enviando dados para um socket ou lendo dados de um socket

Sockets

•um processo envia/recebe mensagens para/de seu ***socket*** (análogo a uma porta)

•processo que transmite envia a mensagem pela porta

•do lado do receptor, a infra-estrutura da camada de transporte distribui a mensagem para o socket do processo receptor correto



Endereçamento de processos

- Para receber mensagens, um processo deve ter um ***identificador***

- Cada host tem um único endereço IPv4 de 32 bits

Questão:

Ele é suficiente para identificar o processo?

Resposta:

Não!
Muitos processos podem rodar no mesmo hospedeiro

- *identificador* inclui tanto o endereço IP quanto os *números das portas associadas* com o processo no hospedeiro

- Exemplo de números de porta:
  - Servidor HTTP: 80
  - Servidor de correio: 25

- Para enviar uma msg HTTP para o servidor Web gaia.cs.umass.edu:

Endereço IP: **128.119.245.12**

Número da porta: **80**

Um protocolo de aplicação define:

- ***tipos de msgs trocadas***,

  ex., request, response

- ***Sintaxe da msg:***

  Quais campos existem nas msgs e como são delimitados

- ***Semântica da msg***

  Significado da informação nos campos

- ***Regras estipulando quando e como*** os processos enviam e respondem a msgs

Protocolos abertos:

- Definidos em RFCs (Request for Comments)
- Permitem interoperabilidade com outros protocolos

  ex.: HTTP, SMTP

Protocolos proprietários:

ex.: Skype

De que serviços uma aplicação precisa?

Transferência Confiável de dados

- algumas aplicações (p.ex., transferência de arquivos, telnet) requerem transferência 100% confiável
- outras (p.ex. áudio e vídeo) podem tolerar algumas perdas

Largura de banda

- algumas apls (p.ex., multimídia) requerem quantia mínima de banda para serem “viáveis”
- outras apls (“apls elásticas”) conseguem usar qq quantidade de banda disponível

Temporização (sensibilidade a atrasos)

- algumas aplicações (p.ex., telefonia Internet, jogos interativos) requerem baixo atraso para serem “viáveis”

Segurança

- Criptografia, integridade dos dados, ...

Requisitos de algumas aplicações de rede

Aplicação	Sensib. a perdas	Vazão	Sensib. a atrasos
transferência de arqs	sem perdas	elástica	não
correio	sem perdas	elástica	não
documentos Web	sem perdas	elástica	não
áudio/vídeo em tempo real	tolerante	áudio: 5kbps-1Mbps vídeo:10kbps-5Mbps	sim, 100's mseg
áudio/vídeo gravado	tolerante	Igual acima	sim, alguns segs
jogos interativos	tolerante	Alguns kbps-10Mbps	sim, 100's mseg
mensagem instantânea	sem perdas	elástica	---

Serviços providos por protocolos de transporte da Internet

Serviço TCP:

- ***orientado a conexão***
- ***estabelecimento de conexão*** necessário entre cliente e servidor
- ***transporte confiável*** entre processos remetente e receptor
- ***controle de fluxo:*** remetente não vai "afogar" receptor de dados
- ***controle de congestionamento:*** limitar a taxa de transmissão do remetente quando a rede estiver carregada

não provê:

garantias temporais, vazão mínima e segurança

Serviço UDP:

- ***transferência de dados não confiável*** entre processos remetente e receptor
- ***não existe estabelecimento de conexão, confiabilidade, controle de fluxo, controle de congestionamento, garantias temporais ou de vazão mínima, segurança***

- P: Qual é o interesse em ter um UDP? ***Velocidade!!!***

Aplicações da Internet: seus protocolos e seus protocolos de transporte

Aplicação	Protocolo da camada de apl.	Protocolo de transporte usado
correio eletrônico	**SMTP [RFC 2821]**	TCP
acesso terminal remoto	**Telnet [RFC 854]**	TCP
Web	**HTTP [RFC 2616]**	TCP
transferência de arquivos	**FTP [RFC 959]**	TCP
streaming multimídia	**HTTP (ex. Youtube)** **RTP [RFC 1889]**	TCP ou UDP
telefonia Internet	**SIP, RTP, proprietário (ex., Skype)**	TCP ou UDP

Provendo segurança ao TCP

TCP & UDP

Não fornecem criptografia!!!

Textos e senhas enviados para o socket atravessam a Internet abertamente

SSL

- Provê criptografia a uma conexão TCP
- Provê integridade dos dados
- Provê autenticação

SSL – camada de Aplicação

Aplicações usam as bibliotecas SSL, que "falam" com o TCP

Socket SSL

Textos abertos e senhas enviadas ao socket atravessam a rede criptografados

A Web e o *http*

Talvez o que mais atraia a maioria dos usuários da Web é que ela funciona por demanda.

- O ***http*** (*Protocolo de Transferência de Hipertexto*) é o protocolo da camada de aplicação da Web, está no coração da Web e é definido no [RFC 1945] e no [RFC 2616].

- O ***http*** é executado tanto no cliente quanto no servidor: necessita de dois processos

Web: algumas definições

Página **Web**:

- ✓ consiste de “objetos”
- ✓ Ex: um objeto pode ser um arquivo **html**, imagem **jpeg**, aplicativo **Java**, arquivo de **áudio**, **vídeo, etc**
- ✓ endereçada por uma **URL**

Uma URL tem duas partes:

- nome de hospedeiro
- nome de caminho + objeto

Ex: **www.ufrrj.br/depto/fig.gif**

Quase todas as págs Web consistem de uma página base HTML que inclui vários objetos referenciados.

O protocolo *http*

http: hypertext transfer protocol

- protocolo da camada de aplicação para Web
- modelo cliente/servidor
  - ***cliente:*** navegador que pede, recebe (usando o http) e "visualiza" objetos
  - ***servidor:*** servidor Web envia (usando o http) objetos em resposta a pedidos

Mais sobre o protocolo *http*

Protocolo ***http*** usa o serviço de ***transporte TCP***:

- cliente inicia conexão TCP (cria Socket) p/ servidor, porta 80
- servidor aceita pedido de conexão TCP do cliente
- mensagens http (mensagens do protocolo da camada de apl) trocadas entre browser (cliente http) e servidor Web (servidor http)
- conexão TCP é encerrada

http é **"sem estado"**

servidor não mantém informação sobre pedidos anteriores do cliente

Nota: **Protocolos que mantêm "estado" são complexos!**

- história passada (estado) tem que ser guardada
- caso servidor/cliente caia, suas visões do "estado" podem ser inconsistentes, devem ser "reconciliadas"

http: conexões TCP persistentes e não persistentes

Não persistente

- servidor analisa pedido, responde e encerra conexão TCP
  - Máx. 1 obj enviado pela conexão TCP antes desta ser fechada

    ⇒ download de múltiplos objetos requer múltiplas conexões TCP
- transferência de cada objeto sofre de "partida lenta"

Obs: Qdo este tipo é usado, a maioria dos browsers usa *conexões TCP paralelas*

Persistente

- na mesma conexão TCP: servidor analisa pedido, responde, analisa novo pedido,...

  ⇒ múltiplos objetos são enviados na mesma conexão TCP
- Cliente envia pedidos para todos objetos referenciados assim que recebe o HTML base

Exemplo de *http* não persistente

Suponha que usuário digita a URL
www.ufrrj.br/Departmento/inicial.index

(suponha que pág.contém texto e referências a 10 imagens jpeg)

1a. Cliente http inicia conexão TCP a servidor http (processo) a www.ufrrj.br. (porta 80)

1b. servidor http no hospedeiro www.ufrrj.br espera por conexão TCP na porta 80, "aceita" conexão e avisa ao cliente

2. cliente http envia *mensagem de pedido* de http (contendo URL) através do socket TCP. Msg indica que cliente quer o objeto Departmento/inicial.index

3. servidor http recebe mensagem de pedido, formula *mensagem de resposta* contendo objeto solicitado e envia mensagem para seu socket TCP

tempo

Exemplo de *http* (cont.)

4. servidor http encerra conexão TCP .

5. cliente http recebe mensagem de resposta contendo arquivo html, visualiza arquivo html. Analisando arquivo html, encontra 10 objetos jpeg referenciados

6. Passos 1 a 5 repetidos para cada um dos 10 objetos jpeg

tempo

http não persistente: tempo de resposta

RTT (definição): intervalo de tempo entre a ida e a volta de um pacote entre um cliente e um servidor

tempo de resposta HTTP:

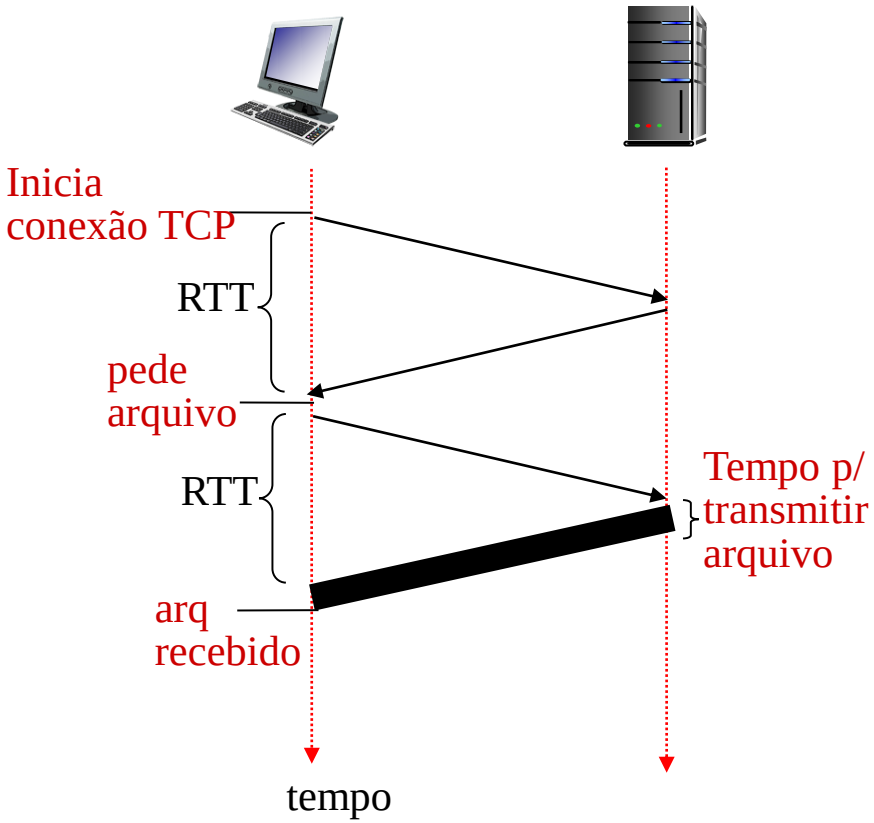
um RTT p/ iniciar a conexão TCP

um RTT p/ requisição HTTP e primeiros bytes da resposta HTTP retornarem

Tempo de transmissão do arquivo

Tempo de resposta http =

2RTT+ tempo de tx do arquivo



http persistente (http 1.1)

Problemas do HTTP não persistente:

- requer 2 RTTs por objeto
- SO aloca recursos do hospedeiro (overhead) p/ cada conexão TCP

⇒ browsers geralmente abrem **conexões TCP paralelas** (5-10) para carregar os objetos referenciados pelo arquivo html

HTTP persistente:

Servidor deixa a conexão aberta após enviar a resposta

Msgs HTTP subsequentes entre o mesmo par cliente/servidor são enviadas sobre a conexão aberta

cliente envia os pedidos assim que encontra objetos referenciados

Somente um RTT é necessário para todos os objetos referenciados

Mensagem de pedido *http*

Dois tipos de mensagem HTTP: *requisição, resposta*

mensagem de requisição HTTP:

ASCII (formato legível por pessoas)

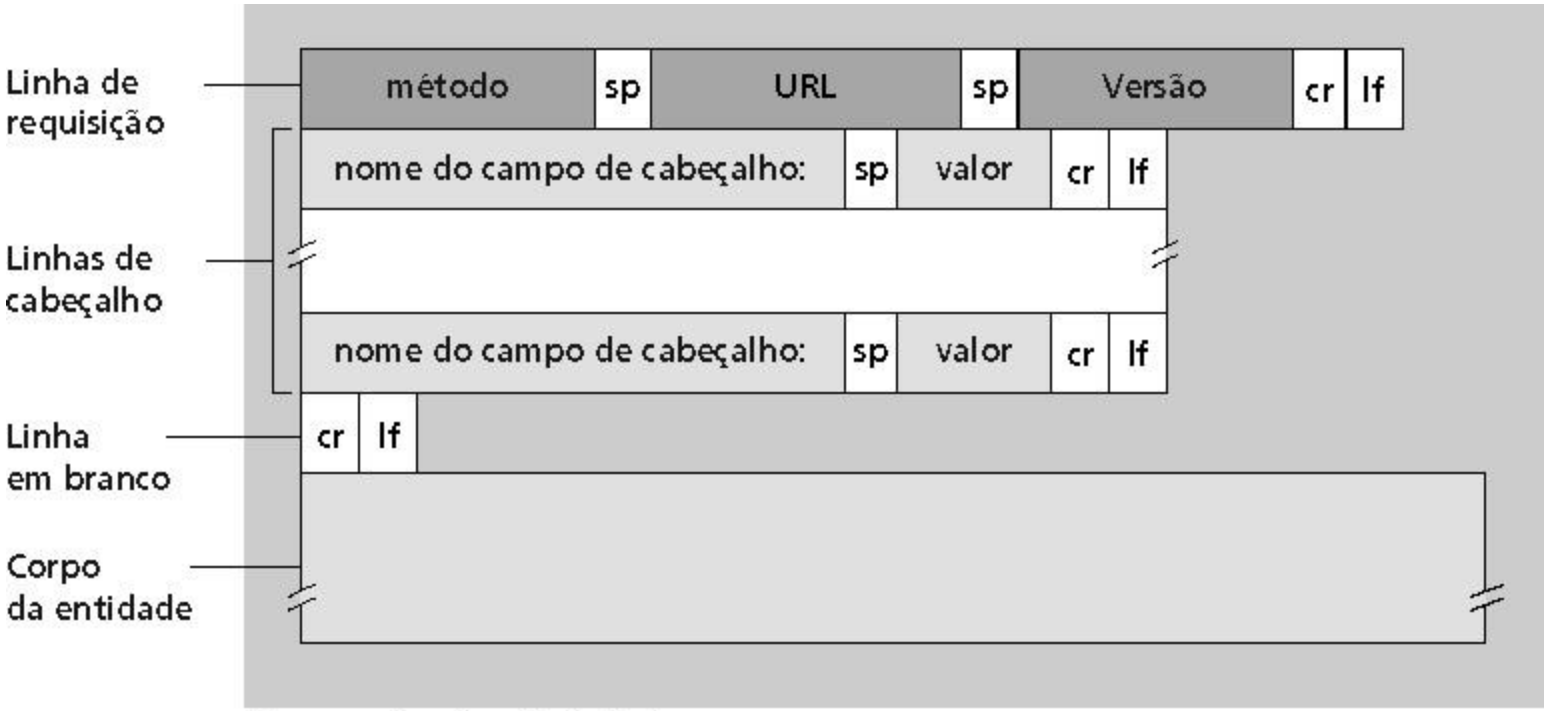
linha da requisição (comandos GET, POST, HEAD)

linhas de cabeçalho

Carriage return, line feed indicam fim de mensagem

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\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
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

Mensagem de pedido *http*: formato geral



Obs.: cr = carriage return; lf = line feed

Carregando dados em um formulário

Método POST:

- Página web geralmente inclui um formulário
- Conteúdo é enviado p/ o servidor no "corpo da entidade"

Método URL:

- Usa o comando GET
- Conteúdo é enviado p/ o servidor no campo URL da linha de requisição:

```
www.somesite.com/animalsearch?key=monkeys&bananas
```

Tipos de métodos mais usados

HTTP/1.0:

GET

POST

HTTP/1.1:

GET, POST

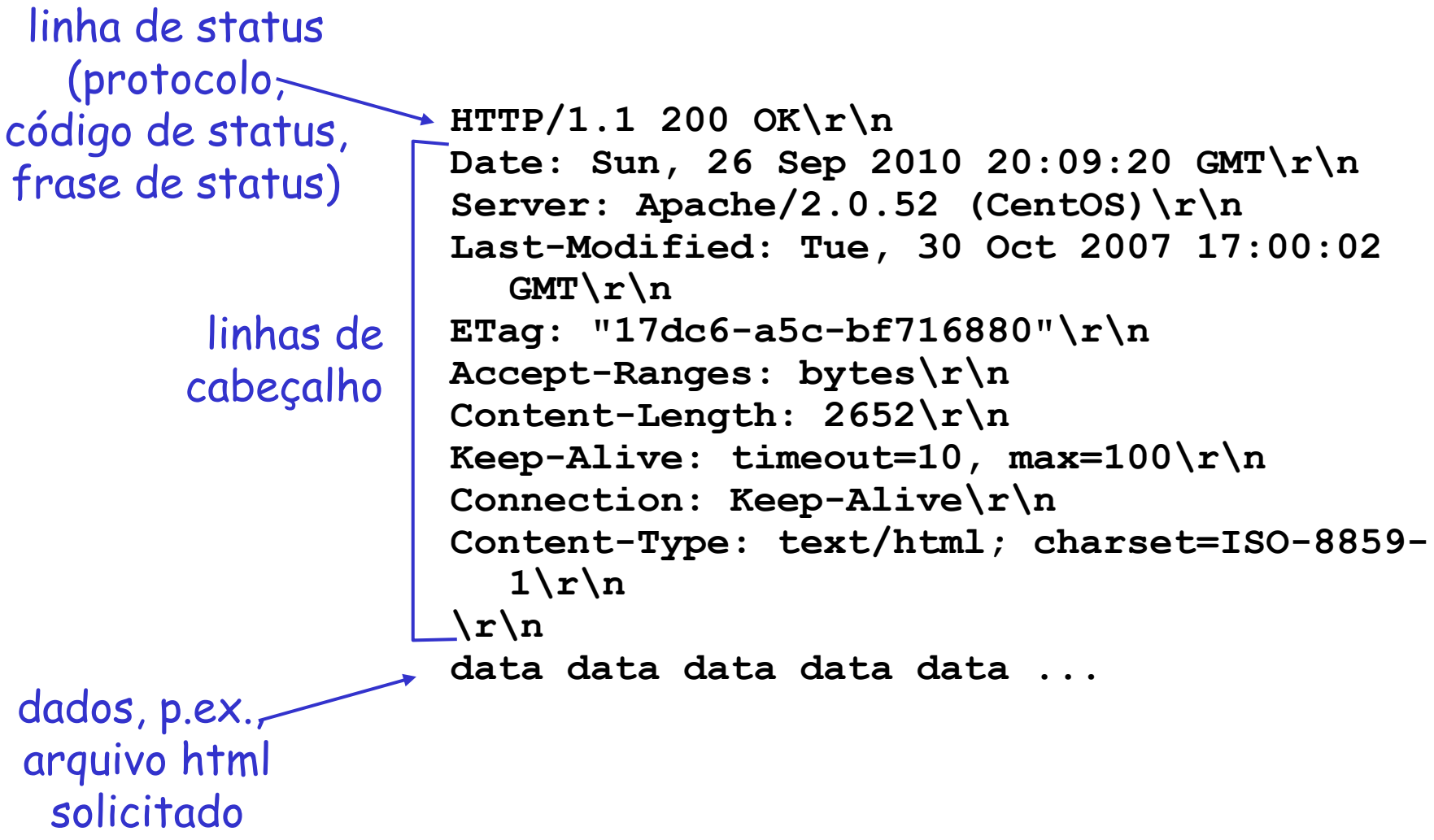
PUT

Faz upload do arq. no corpo da msg para o caminho/diretório especificado no campo URL

DELETE

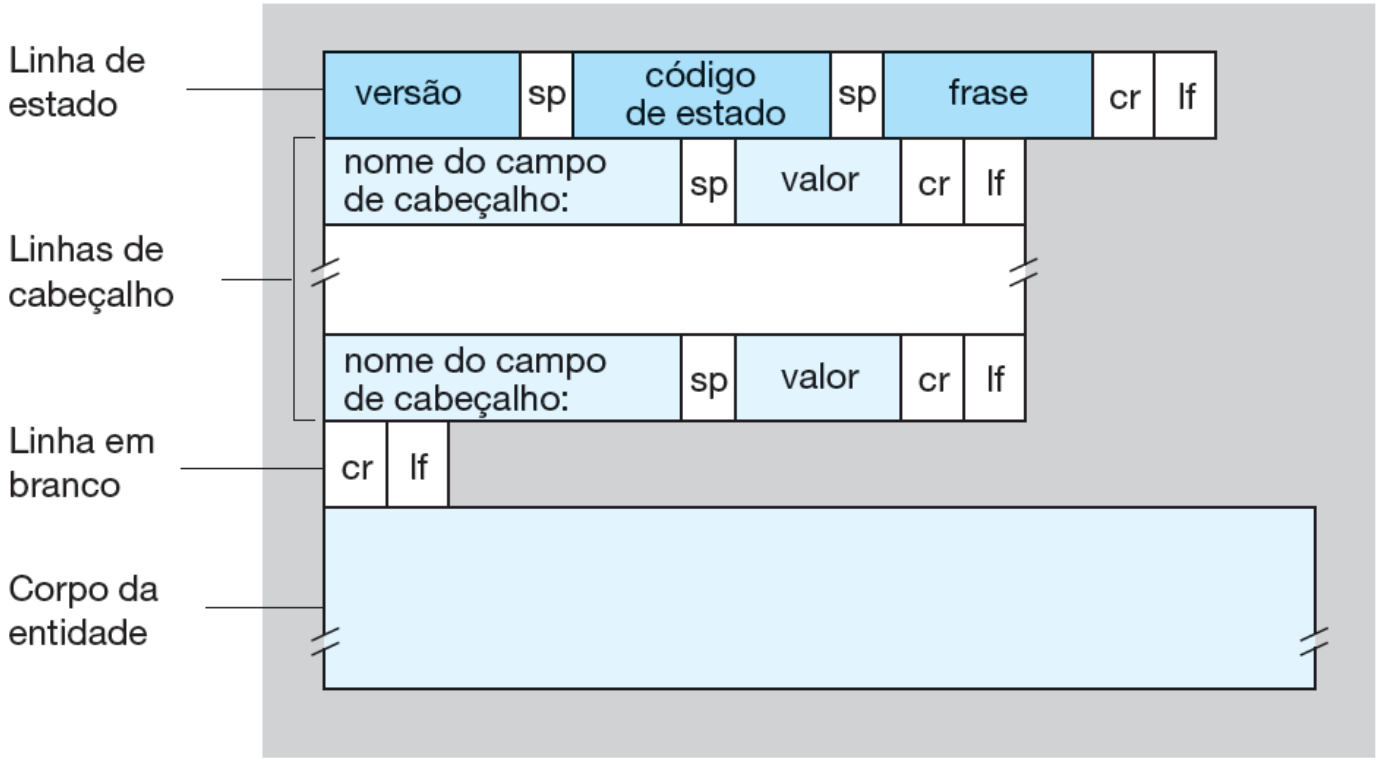
Deleta o arquivo especificado no campo URL

Mensagem de resposta HTTP



Mensagem de resposta *http*: formato geral

- Formato geral de uma mensagem de resposta HTTP



Alguns códigos de status da resposta HTTP

Na primeira linha da mensagem de resposta servidor->cliente. Alguns códigos típicos:

200 OK

sucesso, objeto pedido segue mais adiante nesta mensagem

301 Moved Permanently

objeto pedido mudou de lugar, nova localização especificado mais adiante nesta mensagem (Location:)

400 Bad Request

mensagem de pedido não entendida pelo servidor

404 Not Found

documento pedido não se encontra neste servidor

505 HTTP Version Not Supported

versão de http do pedido não usada por este servidor

Experimente você com http (do lado cliente)

1. Use cliente telnet para seu servidor Web favorito:

```
telnet www.ic.uff.br 80
```

Abre conexão TCP para a porta 80 (porta padrão do servidor http) a www.ic.uff.br. Qualquer coisa digitada é enviada para a porta 80 do www.ic.uff.br

2. Digite um pedido GET http:

```
GET /~michael/index.html HTTP/1.0
```

Digitando isto (deve teclar ENTER duas vezes), está enviando este pedido GET mínimo (porém completo) ao servidor http

3. Examine a mensagem de resposta enviada pelo servidor http !

Cookies: manutenção do "estado" da conexão

Muitos dos principais sites Web usam *cookies*

Quatro componentes:

1) linha de cabeçalho do *cookie* na mensagem de resposta HTTP

2) linha de cabeçalho do *cookie* na mensagem de pedido HTTP

3) arquivo do *cookie* mantido no host do usuário e gerenciado pelo browser do usuário

4) BD de retaguarda no site Web

Exemplo:

Suzana acessa a Internet sempre do mesmo PC

Ela visita um site específico de comércio eletrônico pela primeira vez

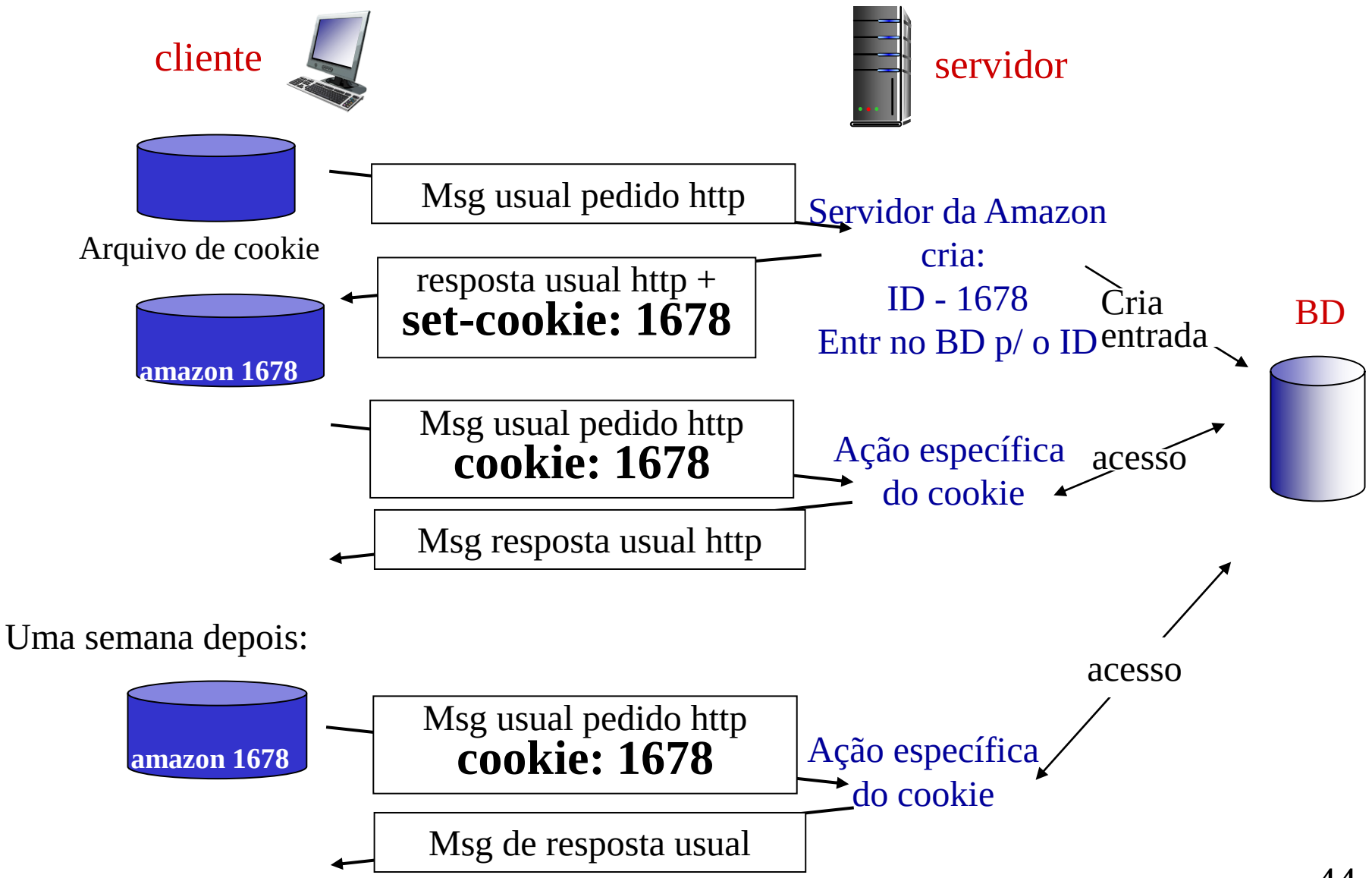
Quando os pedidos *http* chegam no site, o site cria

- *uma ID única*
- *uma entrada para a ID no BD de retaguarda*

Interação usuário-servidor: cookies

- Cliente faz um pedido comum ao servidor
- servidor envia linha de cabeçalho "cookie" na msg de resposta http
  - **`Set-cookie: 1678`** ***(identificação do cliente)***
- cliente apresenta cookie nas mensagens de pedidos posteriores (linha de cabeçalho de cookie)
  - **`cookie: 1678`**
- Servidor verifica cookie apresentado e obtém informações do cliente, guardadas em um banco de dados
  - autenticação
  - lembra preferências do usuário, opções anteriores

Cookies: mantendo "estado" - exemplo



Cookies (cont.):

Cookies podem ser usados para:

- autenticação
- compras com cartões de crédito
- estado da sessão do usuário (Webmail)

Nota

cookies e privaciaaae:

- Cookies permitem aos sites aprender muito sobre você
- O site pode fornecer nome e email para outro site, por ex.

Como manter o "estado":

- Protocolo nos sistemas finais: mantém o estado no transmissor/receptor para múltiplas transações
- Cookies: mensagens http transportam o "estado"