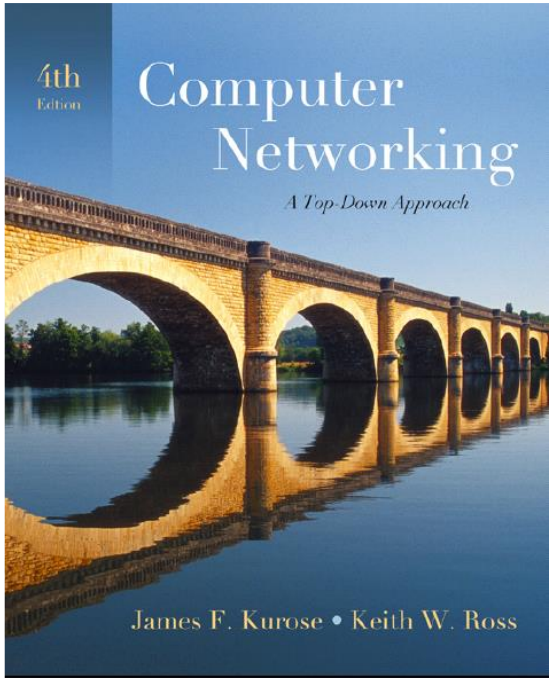
Nível de Aplicação



Computer Networking: A Top Down Approach, 4th edition.
Jim Kurose, Keith Ross
Addison-Wesley, July 2007.

Nota sobre a utilização destes slides:

We're making these slides freely available to all (faculty, students, readers).. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Versão em português 2010 Teles Rodrigues
EST/IPS

Nível de Aplicação

- 2.1 Aplicações de Rede
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Correio Electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de Socket com TCP
- 2.8 Programação de Socket com UDP

Nível de Aplicação

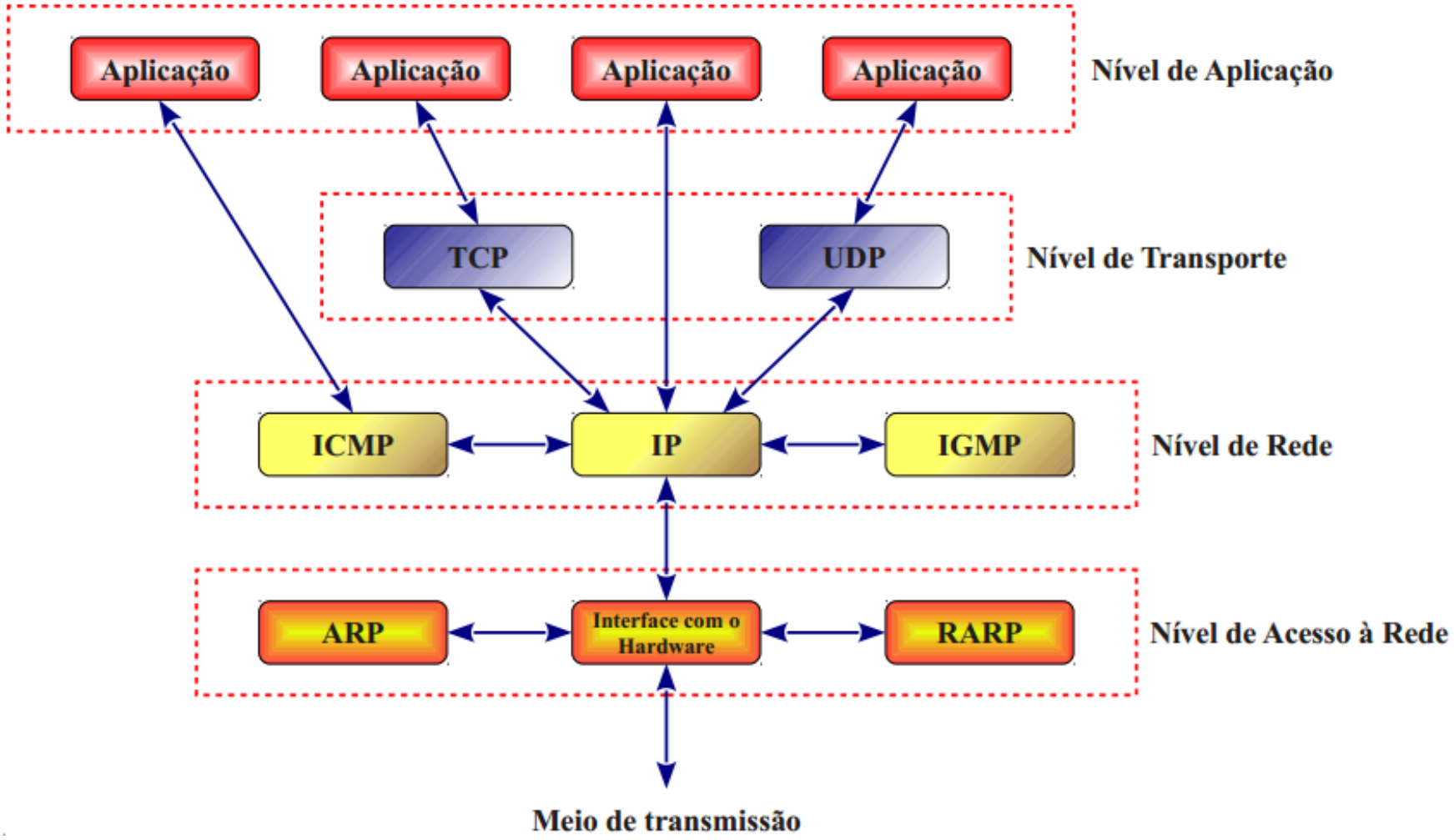
Objectivos:

- conceitos, aspectos de realização dos protocolos de aplicação em rede
  - Modelos de serviço da camada de transporte
  - Paradigma cliente servidor
  - Paradigma peer to peer (p2p)
- Aprender os protocolos examinando protocolos populares da camada de aplicação
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- Programação de aplicações de rede
  - API socket

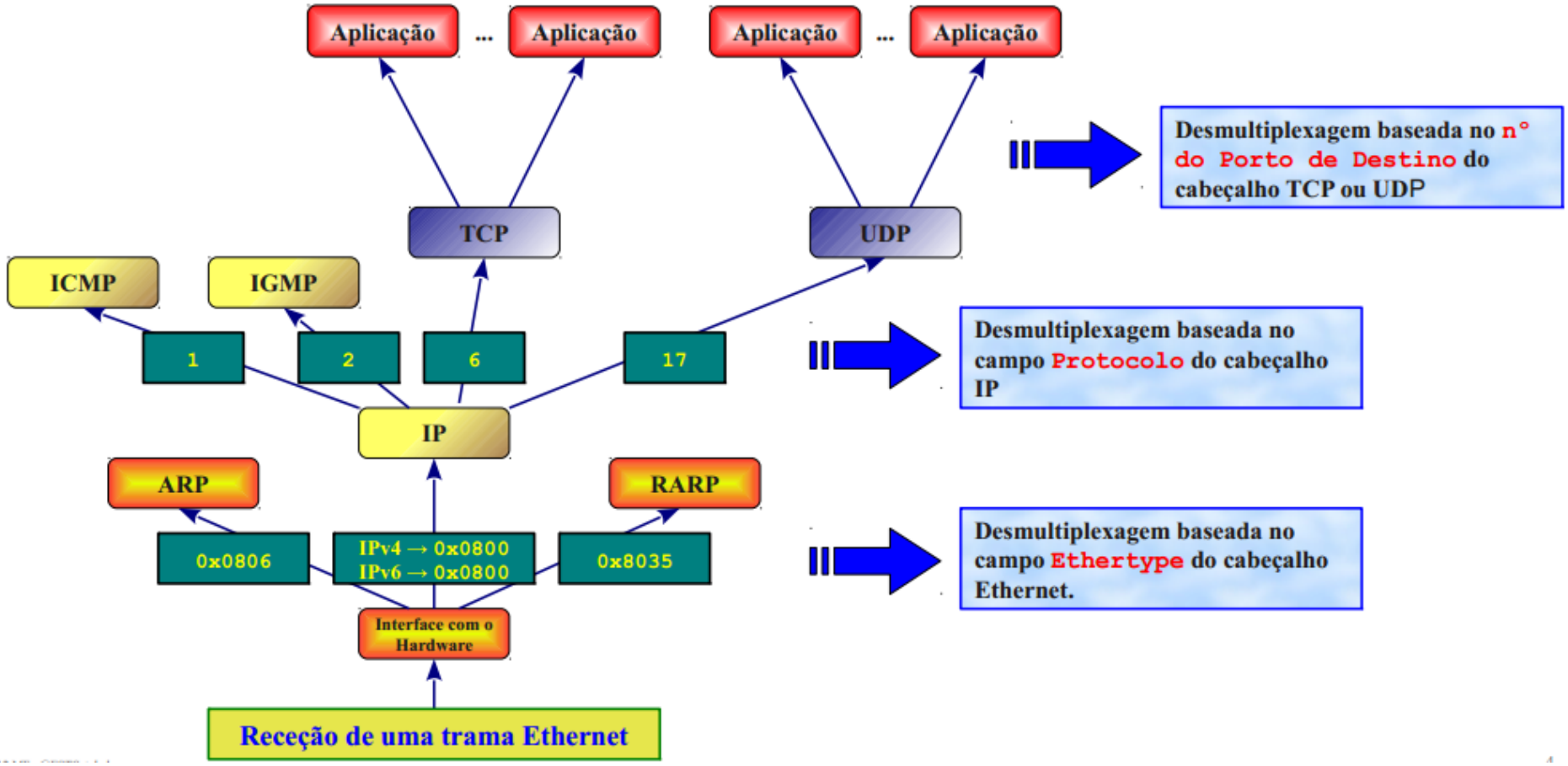
Algumas Aplicações de Rede

- e-mail
- web
- instant messaging
- Login remoto
- Partilha de ficheiros P2P
- Jogos multiutilizadores em rede
- streaming video clips
- voice over IP
- Conferência em tempo real
- grid computing
-
-
-

Modelo TCP/IP



Modelo TCP/IP - Multiplexagem



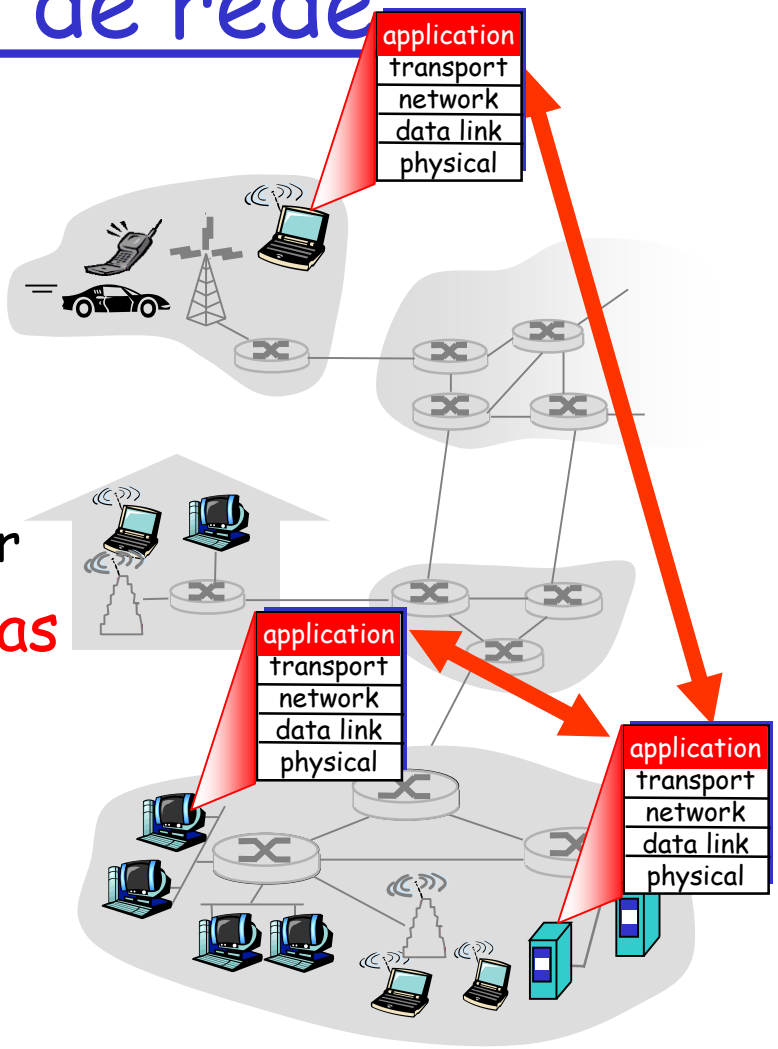
Criação de Aplicações de rede

Escrever um programa que:

- É executado em diferentes sistemas terminais
- Comunica pela rede
- e.g., programa de servidor web comunica com programa browser

Não é necessário criar programas para os dispositivos de rede

- Os dispositivos de rede não executam programas dos utilizadores
- Aplicações apenas nos sistemas terminais permitem desenvolvimento rápido de aplicações



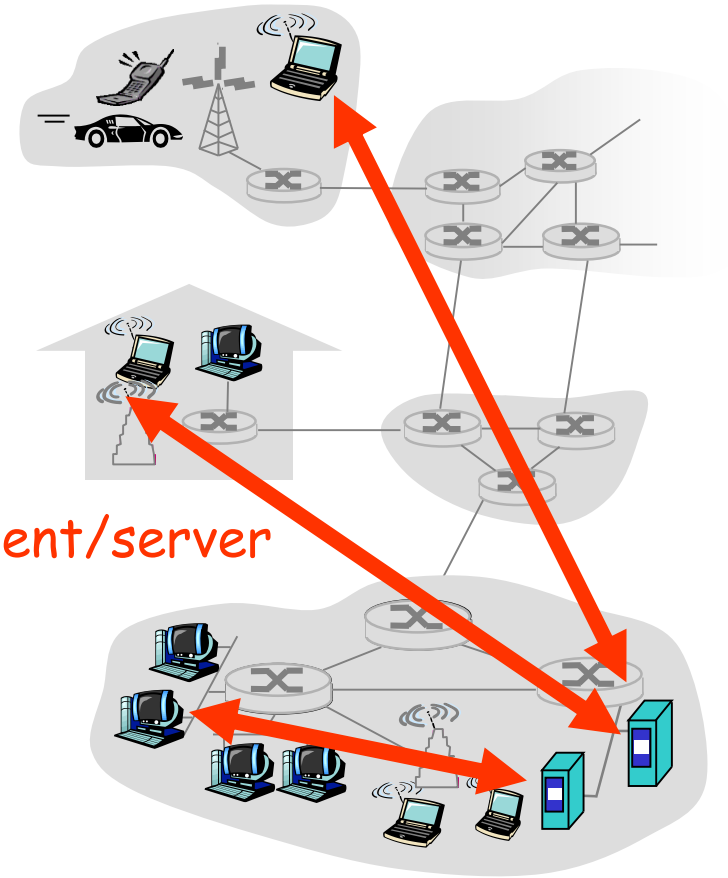
Nível de Aplicação

- 2.1 Aplicações de Rede
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Correio Electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de Socket com TCP
- 2.8 Programação de Socket com UDP

Arquitectura das Aplicações

- Cliente-servidor
- Peer-to-peer (P2P)
- Híbrido client-server e P2P

Arquitectura Cliente-Servidor



servidor:

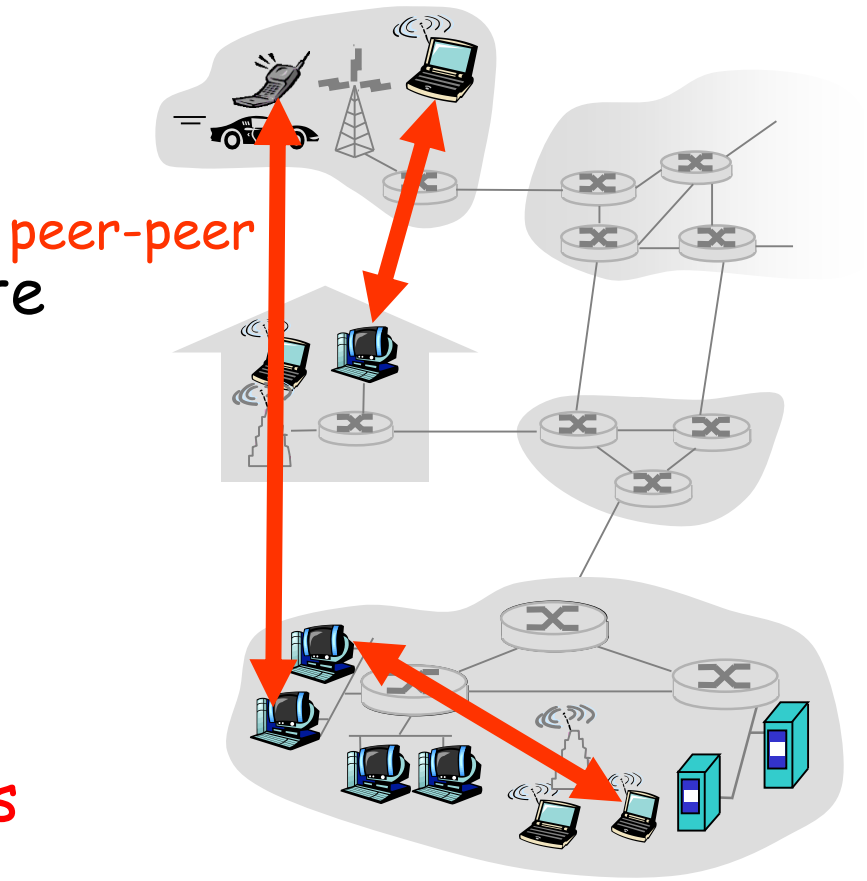
- Sempre ligado
- Endereço IP permanente
- server farms para expansão
- Os programas que implementam os serviços são demoninados DAEMON

clients:

- Comunica com o servidor
- Ligação intermitente
- Pode ter endereço IP dinâmico
- Não comunica directamente com outros clientes

Arquitectura P2P pura

- *Servidores não estão sempre ligados*
- Terminais arbitrários comunicam directamente
- Os pares estão intermitentemente ligados e mudam de endereço IP



Grande escalabilidade mas gestão difícil

Híbrido client-server e P2P

Skype

- Aplicação VoIP (Voice-over-IP) P2P
- servidor centralizado: encontrar endereços dos pares remotos
- Ligação cliente-cliente : directa (não passa pelo servidor)

Instant messaging

- Conversa entre 2 utilizadores é P2P
- serviço centralizado: detecção e localização do cliente
  - user regista o seu endereço IP no servidor central quando se liga
  - user contacta servidor central para encontrar os endereços IP dos pareceiros

Comunicação de processos

Processo: programa em execução num computador.

- Num mesmo computador, 2 processos comunicam usando comunicação entre processos (definido pelo SO: socket, memória partilhada, …).
- processos em diferentes computadores comunicam trocando mensagens

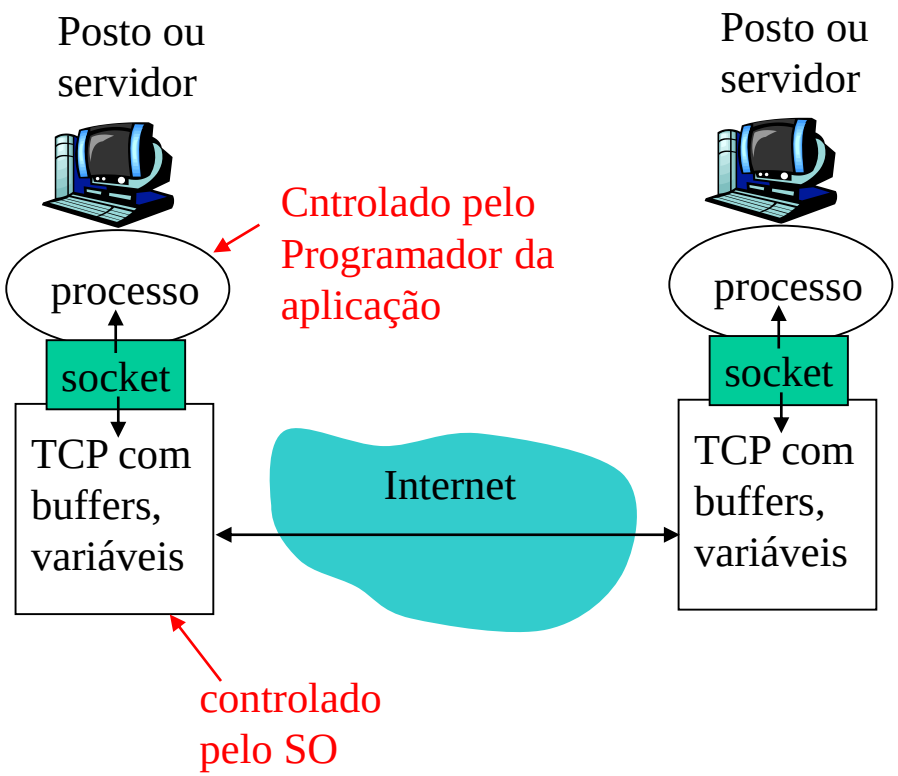
Processo Cliente:
processo que inicia a comunicação

Processo Servidor :
processo que espera ser contactado

- Nota: aplicações com arquitectura P2P têm processos cliente e processos servidor

Sockets

- processos enviam/recebem mensagens de/para os seus socket
- socket análogo a portas
  - processo de envio empurra a mensagem para a porta
  - Processo de envio confia na infraestrutura de transporte para enviar a mensagem ao outro processo



Endereçamento de Processos

- Para receber mensagens, os processos têm de ter um *identificador*
- Computadores têm um endereço único de 32 bits
- *P:* o endereço IP é suficiente para identificar o processo?

Endereçamento de Processos

- Para receber mensagens, os processos têm de ter um *identificador*
- Computadores têm um endereço único de 32 bits
- *P:* o endereço IP é suficiente para identificar o processo?
  - *R:* Não, poderão existir vários processos em execução no mesmo computador
- *identificador* inclui quer endereço IP e número do porto associado com o processo no computador.
- Exemplo de número de portos :
  - HTTP server: 80
  - Mail server: 25
- mensagem HTTP para gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - Port number: 80

Protocolos de Aplicação definem

- Tipos de mensagens trocadas
  - e.g., request, response
- Sintaxe das Mensagens:
  - Que campos nas mensagens e como os campos são delimitados
- Semântica das Mensagens
  - Significado da informação nos campos
- Regras para quando e como enviar e responder a mensagens

Protocolos de domínio público:

- definidos em RFCs
- Permitem interoperabilidade
- e.g., HTTP, SMTP

Protocolos Proprietarios :

- e.g., Skype, Oracle, …

Requisitos de transporte necessários para aplicações

Perda de dados

- Algumas aplicações (e.g., audio) podem tolerar algumas perdas
- Outras aplicações (e.g., file transfer, telnet) requerem transferências 100% confiáveis

Timing/atrasos

- Algumas aplicações (e.g., Internet telephony, jogos interactivos) requerem pouco atraso para "funcionarem"

Largura de banda

- Algumas aplicações (e.g., multimedia) requerem uma quantidade mínima de largura de banda para "funcionarem"
- Outras aplicações ("aplicações elásticas") usam qualquer largura de banda

Segurança

- Encriptação, integridade dos dados, …

Requisitos do serviço de transporte para as aplicações comuns

Aplicação	Perda de dados	Largura de banda	Sensibilidade ao atraso
file transfer	não	elástica	não
e-mail	não	elástica	não
Documentos Web	não	elástica	não
audio/video tempo real	tolerante	audio: 5kbps-1Mbps video:10kbps-5Mbps	sim, 100's mseg
audio/video gravados	tolerante	Como anterior	sim, alguns segs
Jogos interactivos	tolerante	Poucos Kbps	sim, 100's mseg
instant messaging	não	elástica	Sim e não

Serviços de Transporte

serviços TCP :

- *Orientados à ligação:* necessário estabelecimento de ligação entre os processos Cliente e Servidor
- *Transporte fiável* entre o processo emissor e o processo receptor
- *Controlo de fluxo:* o emissor não sobrecarrega o receptor
- *Controlo de congestão:* emissor reduz o envio quando a rede está sobrecarregada
- *Não fornece:* garantia de atraso nem de largura de banda

serviços UDP :

- Transferência de dados não fiável entre processo emissor e processo receptor
- Não estabelece: ligação, fiabilidade, controlo de fluxo, controlo de congestão, nem garantia de atraso ou largura de banda

P: Porque existe?

Para que serve o UDP?

Aplicações Internet: protocolos de aplicação e de transporte

Aplicação	Protocolo de nível aplicação	Protocolo de transporte
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (eg Youtube), RTP [RFC 1889]	TCP ou UDP
Internet telephony	SIP, RTP, proprietario (e.g., Skype)	Tipicamente UDP

Nível de Aplicação

- 2.1 Aplicações de Rede
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Correio Electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de Socket com TCP
- 2.8 Programação de Socket com UDP

Web e HTTP

Alguma terminologia

- Uma págima web consiste num conjunto de objectos
- Objectos podem ser: ficheiros HTML, imagens JPEG, applet Java, ficheiros áudio,…
- Páginas Web consistem num ficheiro baseado em HTML que inclui várias referências a objectos
- Cada objecto é endereçavel por um URL
- Exemplo de URL:

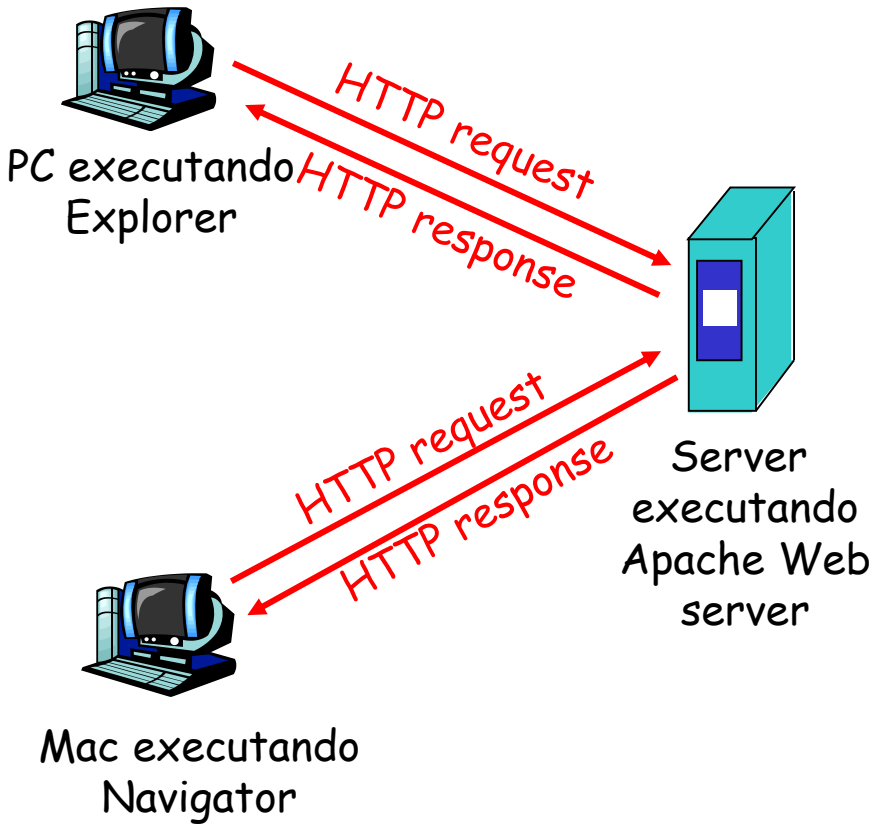
`www.someschool.edu/someDept/pic.gif`

host name | path name

Sumário HTTP

HTTP: hypertext transfer protocol

- Protocolo Web Nível de Aplicação
- Modelo client/server
  - *client:* browser que faz pedidos, recebe e afixa objectos Web
  - *server:* Web server envia respostas aos pedidos



Sumário HTTP (continuação)

Utiliza TCP:

- client inicia ligação TCP (cria socket) para servidor, port 80
- Servidor aceita ligação TCP do cliente
- Mensagens HTTP (messagens de protocolo do nível aplicação) trocadas entre browser (HTTP client) e Web server (HTTP server)
- Fecho da ligação TCP

HTTP não tem memória "stateless"

- O servidor não mantem informação sobre os pedidos anteriores dos clientes

À parte

Os Protocolos que mantêm o estado são complex!os

- História passada (estado) deve ser mantida
- Se o servidor/cliente falharem, a sua visão do estado pode ficar inconsistente, e tem de ser conciliada

Ligações HTTP

HTTP Não persistente

- No máximo é enviado um objecto por cada ligação TCP.
- HTTP 1.0 HTTP não persistente

HTTP Persistente

- Multiplos objectos podem ser enviados numa única ligação TCP entre o servidor e o cliente.
- HTTP 1.1 usa ligações persistentes por omissão

HTTP não persistente

Supondo que um utilizador introduz o URL

`www.someSchool.edu/someDepartment/home.index`

(contem texto, referencia a 10 Imagens jpeg)

1a. Cliente HTTP inicia a ligação TCP para o servidor HTTP (processo) em www.someSchool.edu no port 80

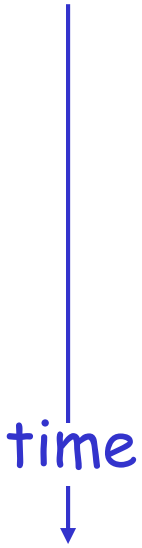
1b. servidor HTTP na máquina www.someSchool.edu
- Espera ligação TCP no port 80.
- "aceita" ligação,
- Notifica cliente

2. Cliente HTTP envia mensagem de pedido (contendo URL) para o socket da ligação TCP. A Mensagem indica que o cliente quer o objecto someDepartment/home.index

3. Servidor HTTP recebe pedido, constroi a mensagem de resposta contendo o objecto pedido, e envia a mensagem para o socket

tempo

HTTP não persistente (cont.)



4. Servidor HTTP fecha a ligação TCP.

5. Cliente HTTP recebe a mensagem de resposta contendo o ficheiro html, mostra o ficheiro html. Executa o ficheiro html, encontra referência a 10 objectos jpeg e fecha a ligação

6. Repete os passos de 1 a 5 para cada um dos 10 objectos jpeg

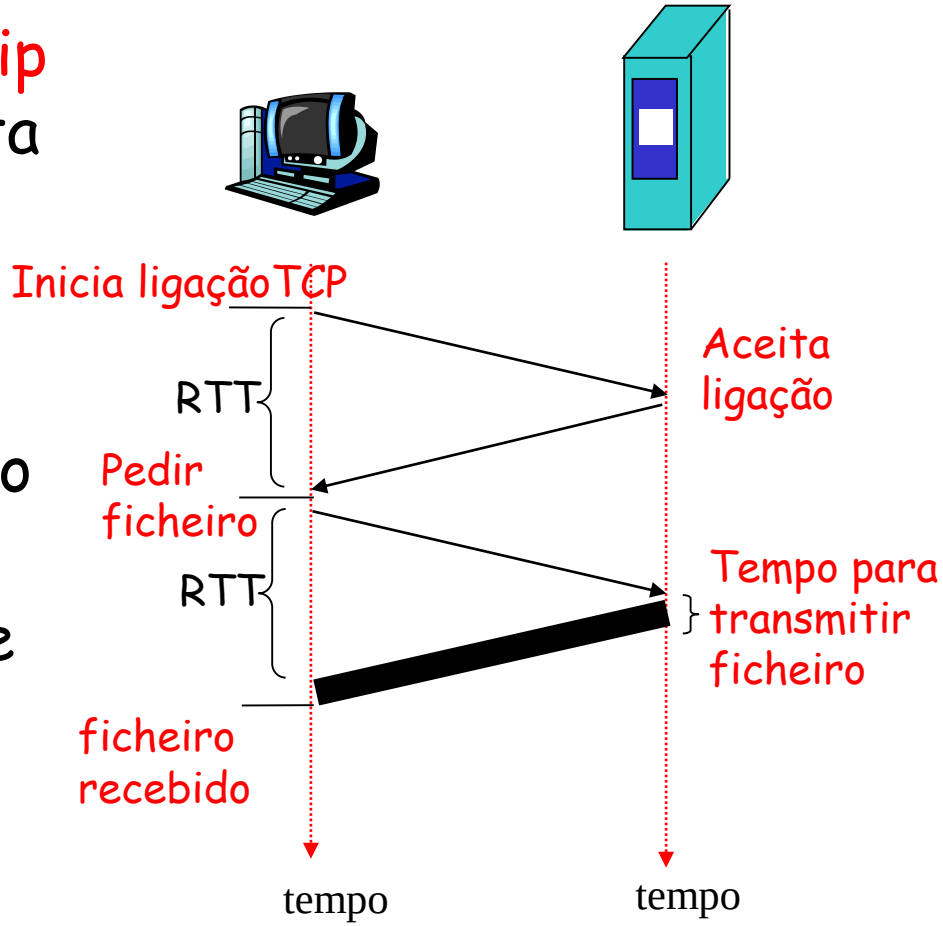
HTTP não persistente: tempo de resposta

Definição de RTT (Round Trip Time): tempo de ida e volta de um pacote entre o cliente e o servidor

Tempo de Resposta:

- 1 RTT para iniciar a ligação TCP
- 1 RTT para pedido HTTP e resposta HTTP começar a chegar
- Tempo de transmissão do ficheiro

total = 2RTT+tempo de transmissão



HTTP persistente

HTTP não persistente:

- requere 2 RTTs por objecto
- Sobrecarga do SO para cada ligação TCP
- Muitos browsers abrem ligações em paralelo para ir buscar os objectos mais rapidamente.

HTTP persistente

- Servidor deixa a ligação aberta depois de responder
- Mensagens HTTP seguintes do mesmo cliente vão pela mesma ligação

HTTP persistente sem pipelining

- O cliente envia novo pedido apenas quando recebeu a resposta do anterior
- Um RTT para cada objecto

HTTP persistente com pipelining

- O cliente envia novo pedido assim que encontra a sua referência no código html
- Um RTT para todos os objectos referidos
- Por omissão para HTTP 1.1

Formato das mensagens de pedido HTTP

- Dois tipos de mensagens HTTP: *pedido, resposta*
- Mensagem de pedido HTTP:
  - ASCII (formato legível por humanos)

Linha do pedido (commandos GET, POST, HEAD)

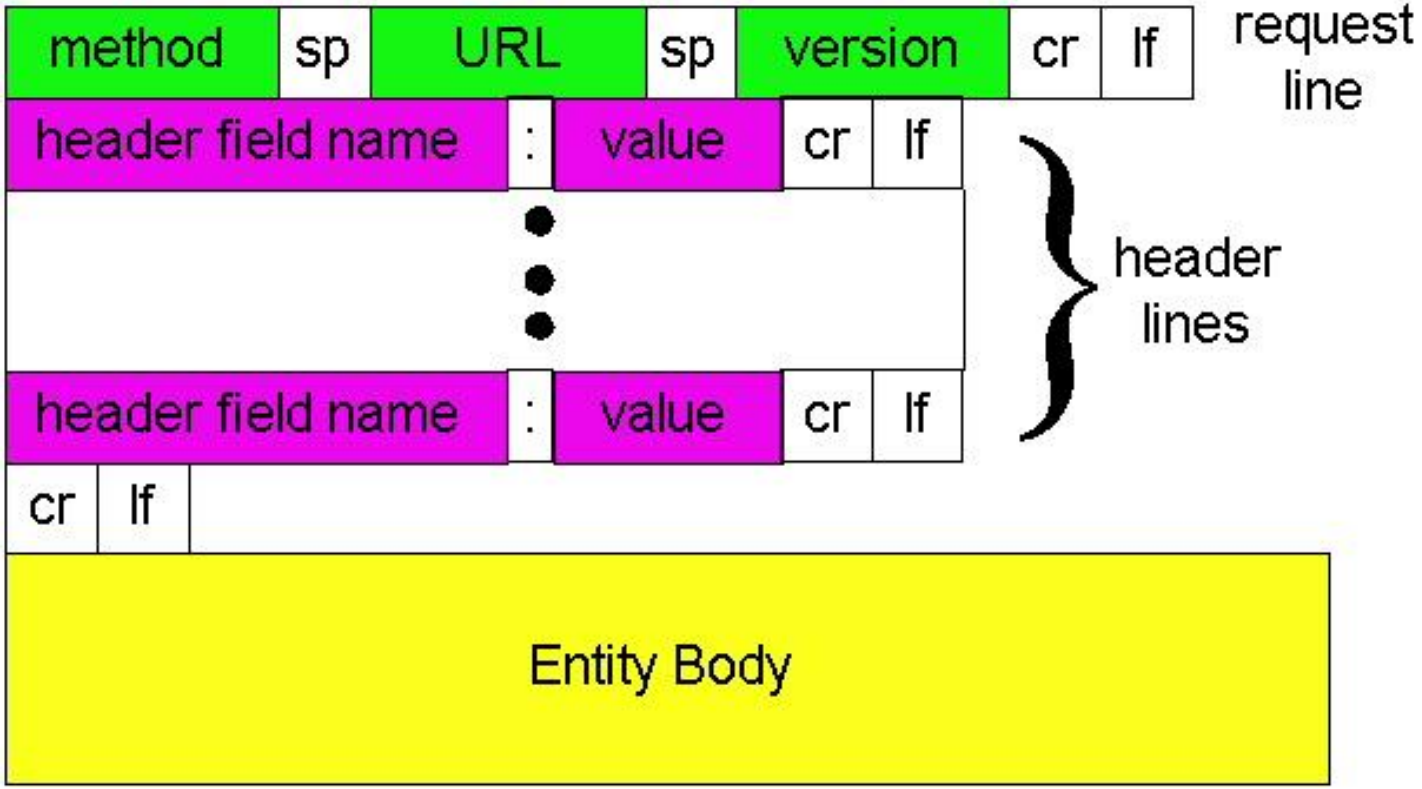
Linha de cabeçalho

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

Linha em branco (Carriage return, line feed) indica fim da mensagem

(extra carriage return, line feed)

Mensagem de pedido HTTP: formato geral



Envio de dados para formulário

Método Post:

- Na Web é frequente as páginas incluirem entrada de dados para formulários
- Os dados são enviados para o servidor no corpo do pedido (entity body)

Método GET:

- Os dados são enviados no campo URL da linha de pedido:

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

Tipos de métodos

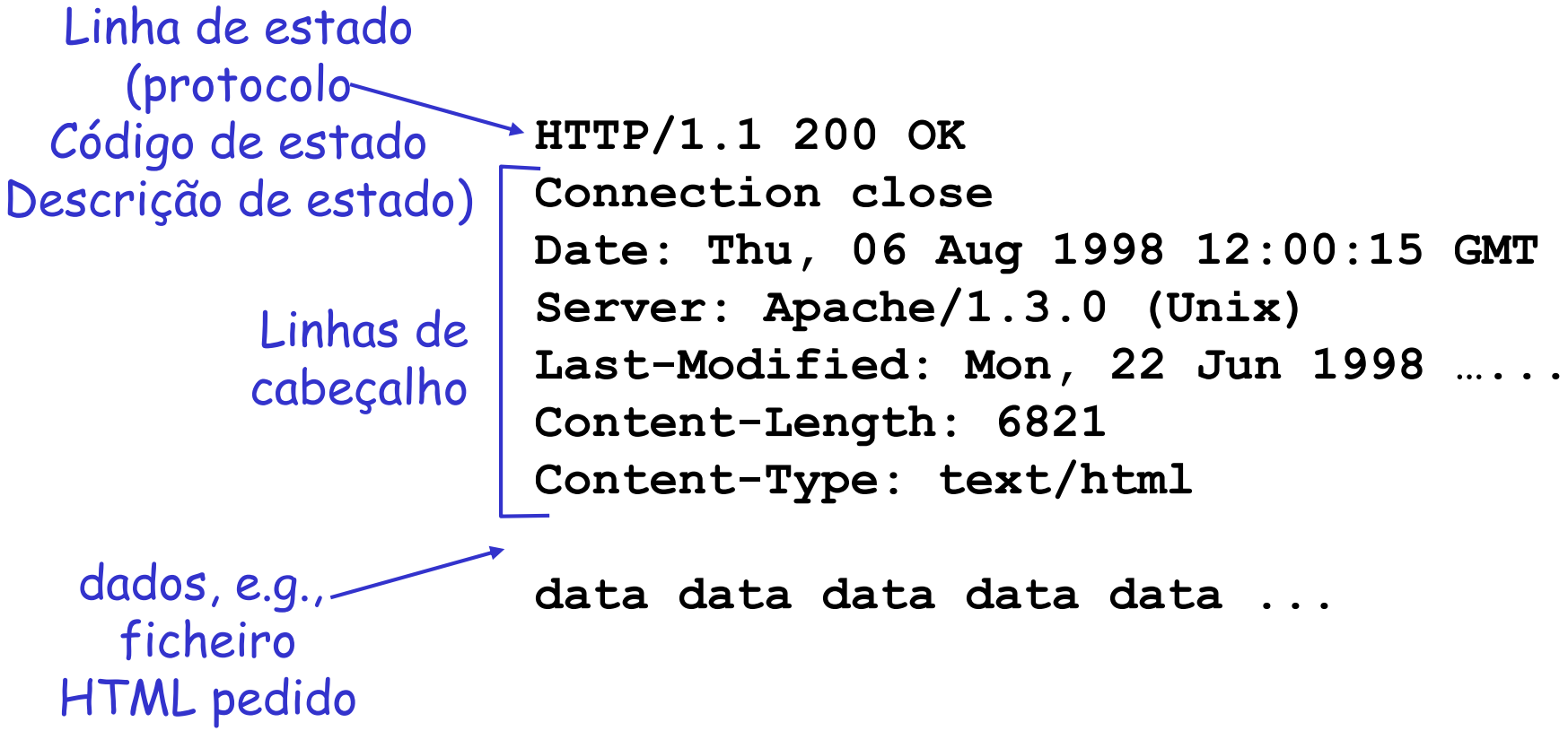
HTTP/1.0

- GET
- POST
- HEAD
  - asks server to leave pede ao servidor para não incluir o objecto na resposta

HTTP/1.1

- GET, POST, HEAD
- PUT
  - Envia o ficheiro no corpo do pedido para o caminho especificado no campo URL
- DELETE
  - Apaga o ficheiro especificado no campo URL

Formato da mensagem HTTP: resposta



Códigos de estado HTTP

Na primeira linha da mensagem servidor->cliente.

Alguns exemplos de códigos:

200 OK

- Pedido teve sucesso, objecto pedido mais tarde nesta mensagem

301 Moved Permanently

- Objecto pedido foi movido, nova localização especificada mais tarde nesta mensagem

400 Bad Request

- Mensagem de pedido não entendida pelo servidor

404 Not Found

- Objecto pedido não foi encontrado no servidor

505 HTTP Version Not Supported

Teste o HTTP (lado cliente)

1. Telnet para site web :

```
telnet cis.poly.edu 80
```

Abre ligação TCP para o porto 80 (porto default HTTP) em cis.poly.edu. O que for digitado é enviado para o porto 80 em cis.poly.edu

2. Digitar pedido HTTP GET:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

Ao digitar isto (introduzindo RETURN 2 x), é enviado um pedido GET mínimo, mas completo, para o servidor HTTP

3. Analise as respostas enviadas pelo servidor HTTP!

Interacção utilizador-servidor: autenticação

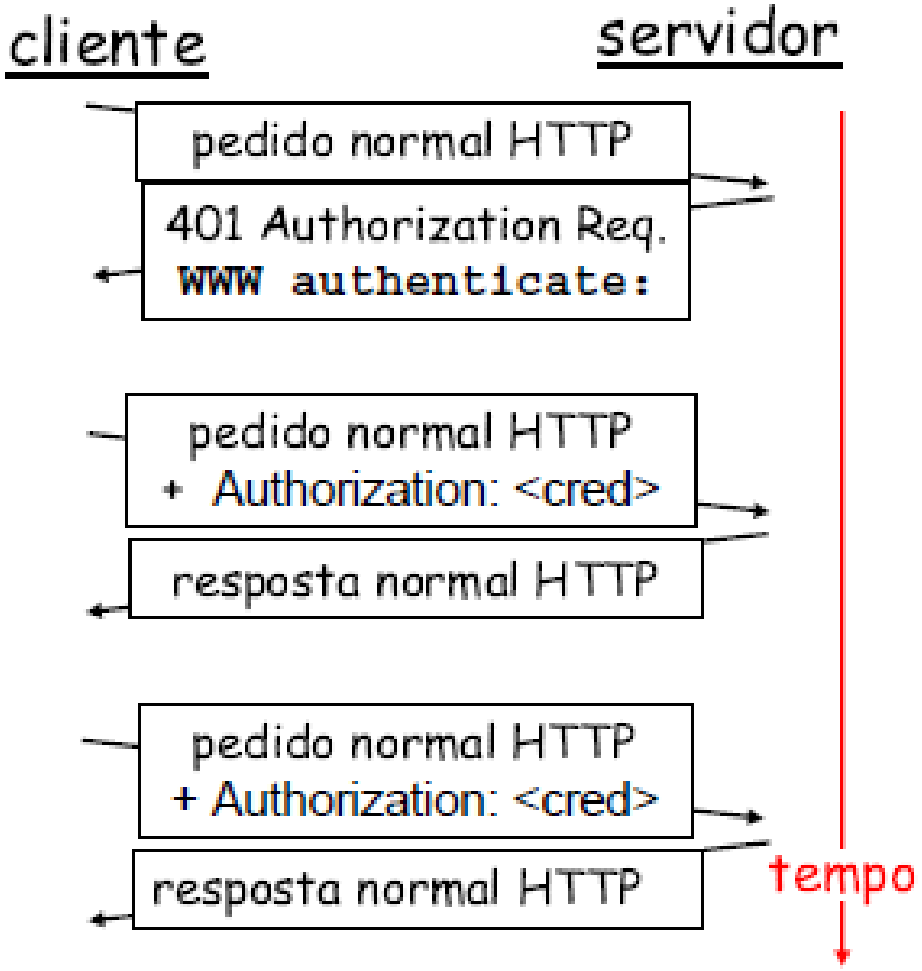
Autenticação: controlo de acessos ao conteúdo do servidor:

Credenciais de autorização: tipicamente nome e password.

Sem estado: o cliente tem de apresentar a autorização em cada pedido

- Linha de cabeçalho Autorization em cada pedido
- Sem cabeçalho Autorization o servidor recusa acesso e responde com cabeçalho WWW authenticate

O browser memoriza a autenticação e repete-a a cada pedido



Mantendo o estado: cookies

Muitos servidores Web usam cookies

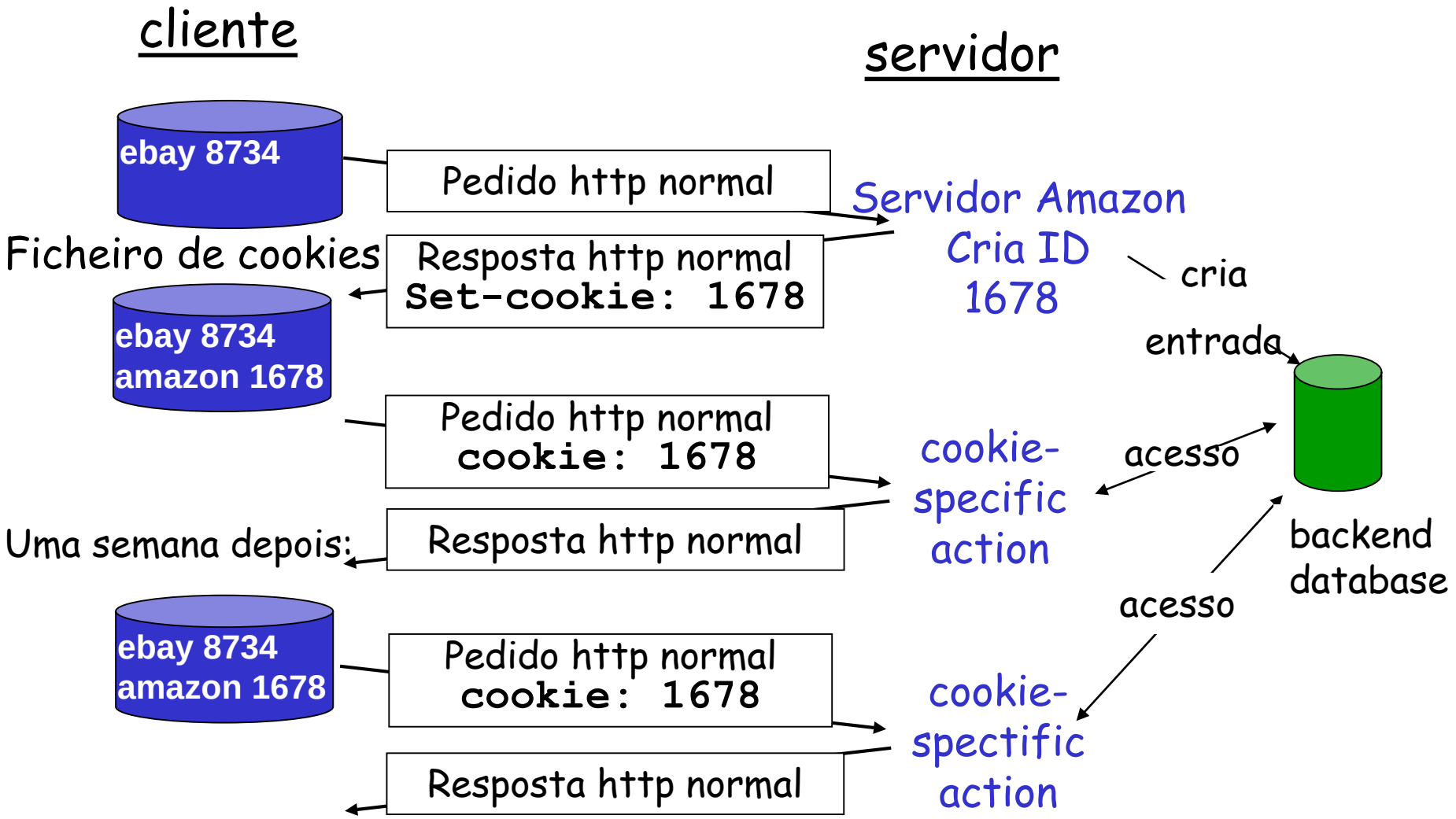
Quatro componentes

1) Linha de cabeçalho cookie na resposta HTTP
2) Linha de cabeçalho cookie no pedido HTTP
3) Ficheiro de cookies mantido na máquina do utilizador e gerido pelo browser
4) Base de dados no servidor web

Exemplo:

- O João acede à Internet sempre do mesmo PC
- Visita um servidor de comércio electrónico pela 1ª vez
- Quando o 1º pedido HTTP chega ao servidor, este cria um identificador (ID) único e cria uma entrada na sua base de dados para o ID

Mantendo o estado: cookies (cont.)



Cookies (continuação)

O que os cookies podem oferecer:

- autorização
- Carrinho de compras
- recomendações
- Estado da sessão do utilizador (Web e-mail)

À parte

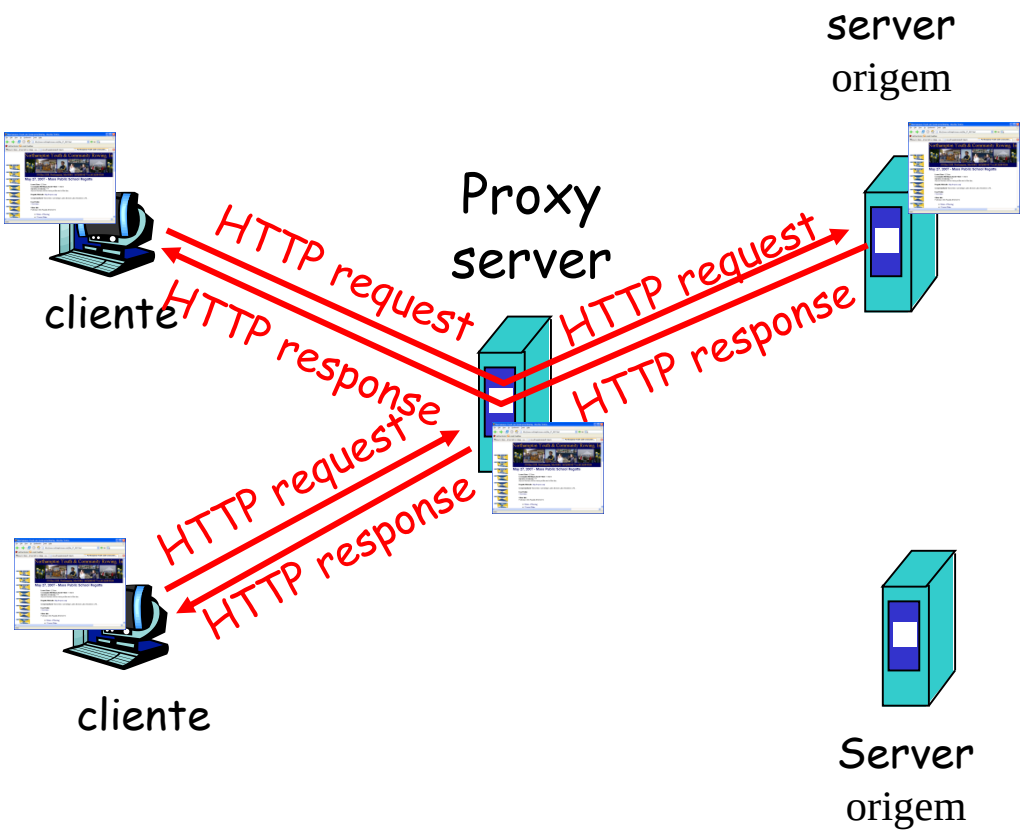
Cookies e privacidade:

- Os cookies permitem que os servidores aprendam muito sobre os utilizadores
- Os utilizadores podem dar o nome e e-mail ao servidor
- Motores de pesquisa usam redirecção e cookies para aprender ainda mais
- Empresas de publicidade obtêm informação de vários servidores

Web caches (proxy server)

Objectivo: servir o cliente sem envolver o servidor de origem

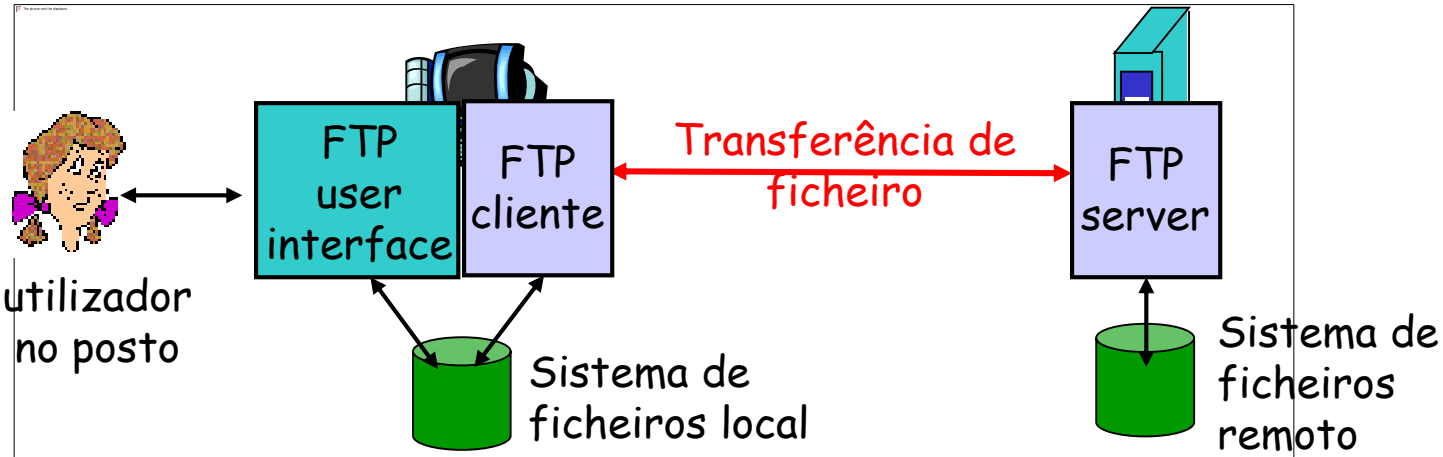
- Utilizador configura browser para acesso através do proxy
- browser envia todos os pedidos ao proxy
  - objectos na cache: cache devolve objecto
  - De outra forma, cache pede objecto à origem, e devolve objecto ao cliente



Nível de Aplicação

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP
- 2.9 Building a Web server

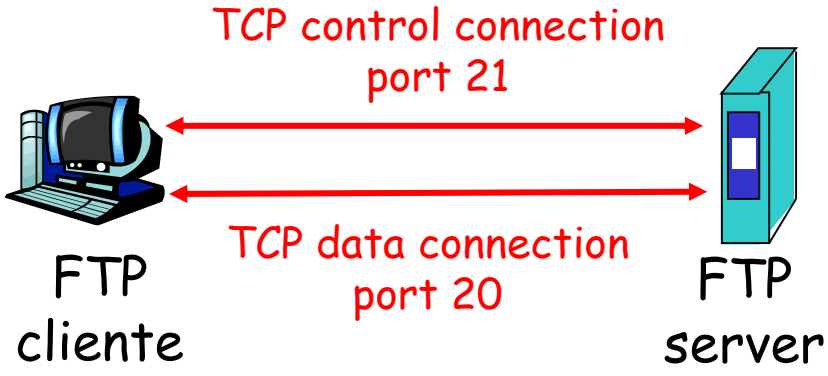
FTP: file transfer protocol



- Transferência de ficheiro de/para o sistema remoto
- Modelo cliente - servidor
  - *cliente:* inicia a transferência (de/para o sistema remoto)
  - *servidor:* sistema remoto
- ftp: RFC 959
- ftp server: porto 21

FTP: ligações de controlo e dados separadas

- O cliente contacta o servidor no porto 21 especificando o TCP como protocolo de transporte
- O cliente obtem autorização pela ligação de controlo
- O cliente lista os ficheiros remotos enviando comandos pela ligação de controlo.
- Quando o servidor recebe comando para transferência de ficheiro, abre ligação TCP porto 20 para dados com o cliente
- Depois de transferir o ficheiro, o sevidor fecha a ligação.



- O servidor abre outra ligação TCP porto 20 para transferir outro ficheiro
- Ligação de controlo: "out of band"
- Servidor FTP mantem o estado: directório actual, autenticação anterior

Comandos FTP, respostas

Exemplos de comandos:

- Enviados como textoASCII no canal de controlo
- **USER *username***
- **PASS *password***
- **LIST** devolve a lista dos ficheiros no directório corrente
- **RETR filename** devolve (obtem) ficheiro
- **STOR filename** guarda (põe) ficheiro no sistema remoto
- **HELP** [comando]

Exempolos de códigos de estado

- Códigos de estado e descrição (como no HTTP)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

Nível de Aplicação

- 2.1 Aplicações de Rede
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Correio Electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de Socket com TCP
- 2.8 Programação de Socket com UDP

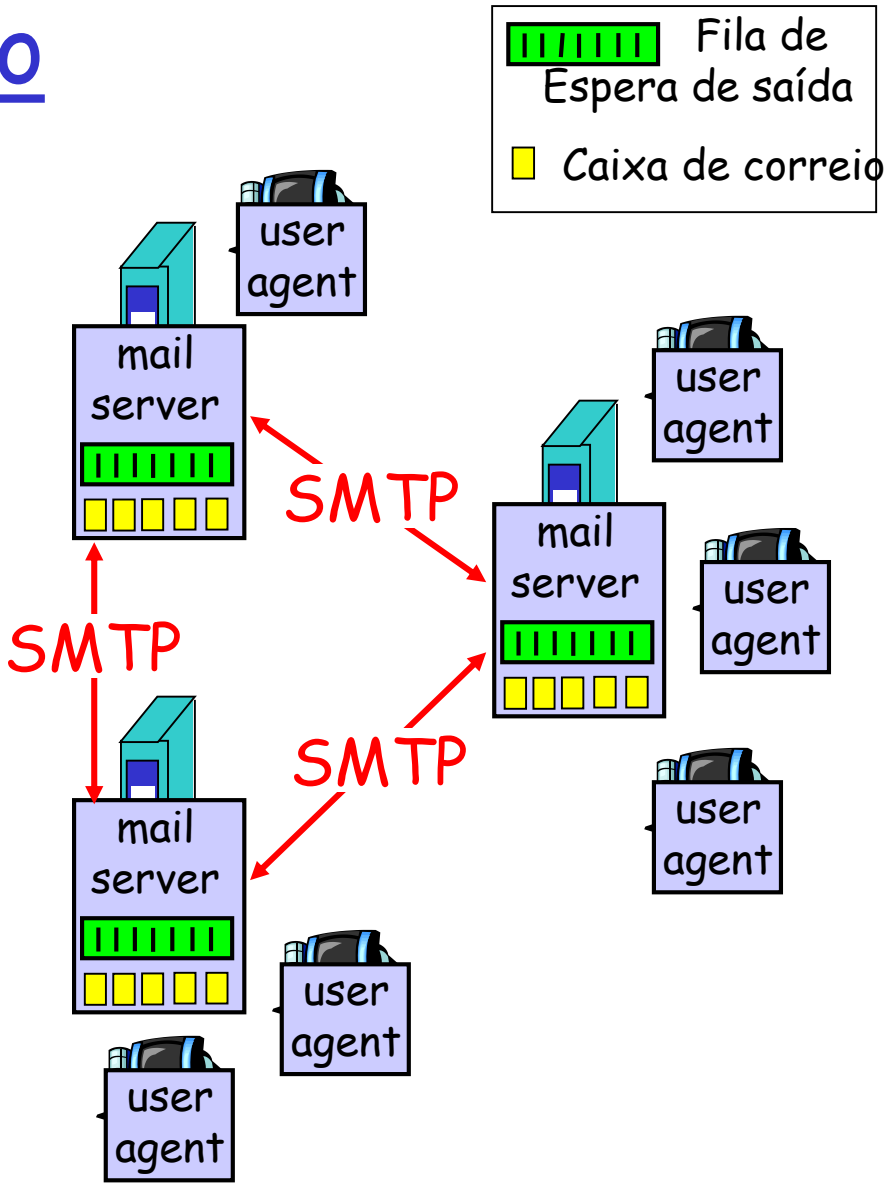
Correio Electrónico

Três componentes fundamentais:

- Agentes de utilizador
- Servidores de correio
- simple mail transfer protocol: SMTP

Agentes de utilizador

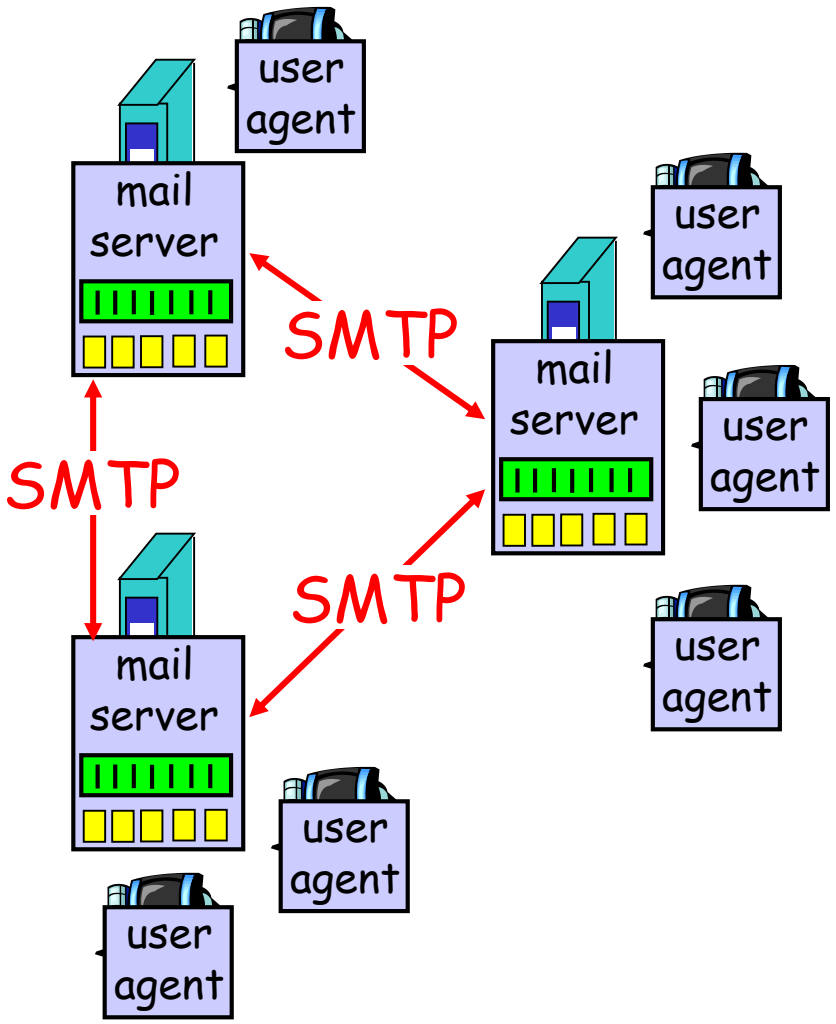
- a.k.a. "programa de correio"
- Compôr, editar e ler mensagens de correio
- e.g., Eudora, Outlook, elm, Mozilla Thunderbird
- Mensagens a enviar ou a receber são armazenadas no servidor



Correio electrónico: servidor de correio

Servidor de correio

- Caixas de correio contém mensagens de entrada para o utilizador (ainda não lidas)
- Fila de espera de mensagens que o utilizador quer enviar
- protocolo SMTP entre servidores de correio para enviar as mensagens
  - cliente: servidor de envio
  - "servidor": servidor de recepção

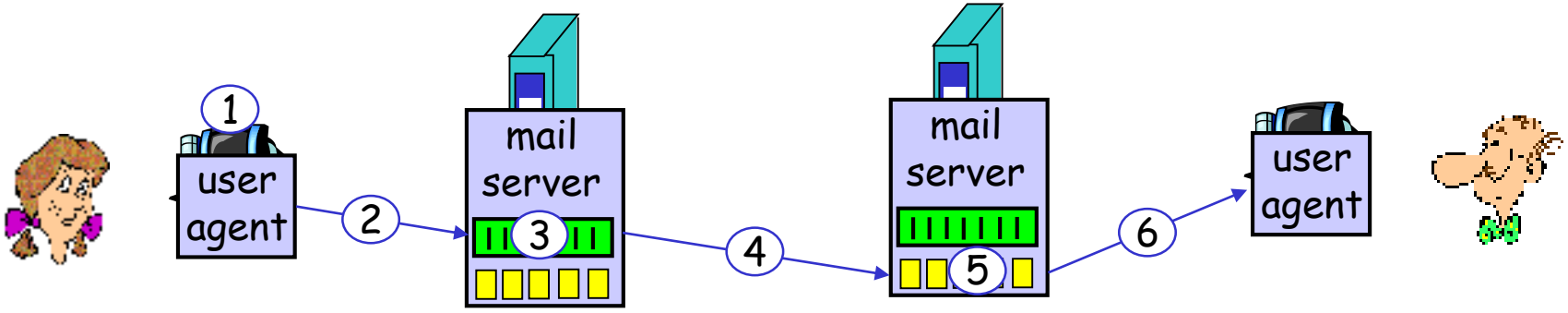


Correio electrónico: SMTP [RFC 2821]

- Usa TCP para transferir mensagens de correio do cliente para o servidor, de forma fiável, através do porto 25
- Transferência directa: servidor de envio para servidor de recepção
- Três fases de transferência:
  - handshaking (apresentação)
  - Transferência de mensagens
  - fecho
- Interacção comando-resposta
  - commandos: texto ASCII
  - resposta: código e descrição de estado
- Mensagens codificadas em 7-bit ASCII

Cenário: Alice envia messagem a Bob

1) Alice usa UA (User Agent) para compor mensagem para `bob@someschool.edu`
2) O UA da Alice envia a mensagem para o seuservidor de correio: a mensagem é colocada em fila de espera
3) O lado cliente do SMTP abre uma ligação com o servidor de correio do Bob
4) O cliente SMTP envia a mensagem pela ligação SMTP
5) O servidor de correio do Bob coloca a mensagem na caixa de correio
6) Bob invoca o seu UA para ler a mensagem



Exemplo de interacção SMTP

```
c: telnet hamburger.edu 25 Ligação TCP Porto 25
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Experimentem SMTP :

- **Digitar** `telnet nome_do_servidor 25`
- Observar 220 resposta do servidor
- Digitar comandos HELO, MAIL FROM, RCPT TO, DATA, QUIT

Os comandos anteriores permitem enviar correio sem usar um programa cliente

SMTP: palavras finais

- SMTP usa ligação persistente
- SMTP requer que a mensagem seja codificada em ASCII 7 bits (cabeçalho e corpo)I
- Servidor SMTP usa `CRLF.CRLF` para determinaro fim da mensagem
- Alguns caracteres não são permitidos na mensagem. Então a mensagem tem de ser codificada (ex base 64 ou quoted printable)

Comparação com HTTP:

- HTTP: puxa (pull)
- SMTP: empurra (push)
- Ambos têm interacção comando/resposta em ASCII, código de estado
- HTTP: cada objecto encapsula a sua própria mensagem de resposta
- SMTP: multiplos objectos enviados em múltiplas partes (multipart message)

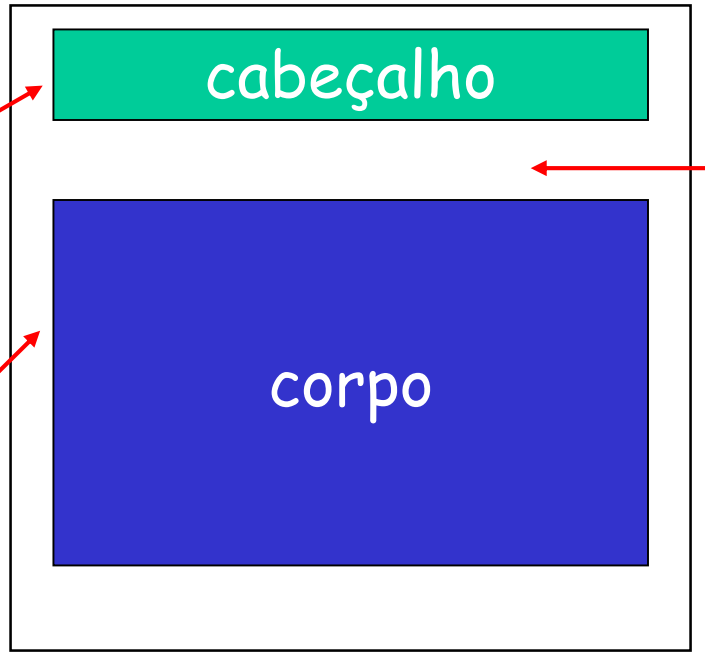
Formato da mensagem de Correio

SMTP: protocolo para transferir mensagens de correio

RFC 822: formato standard para mensagens de texto:

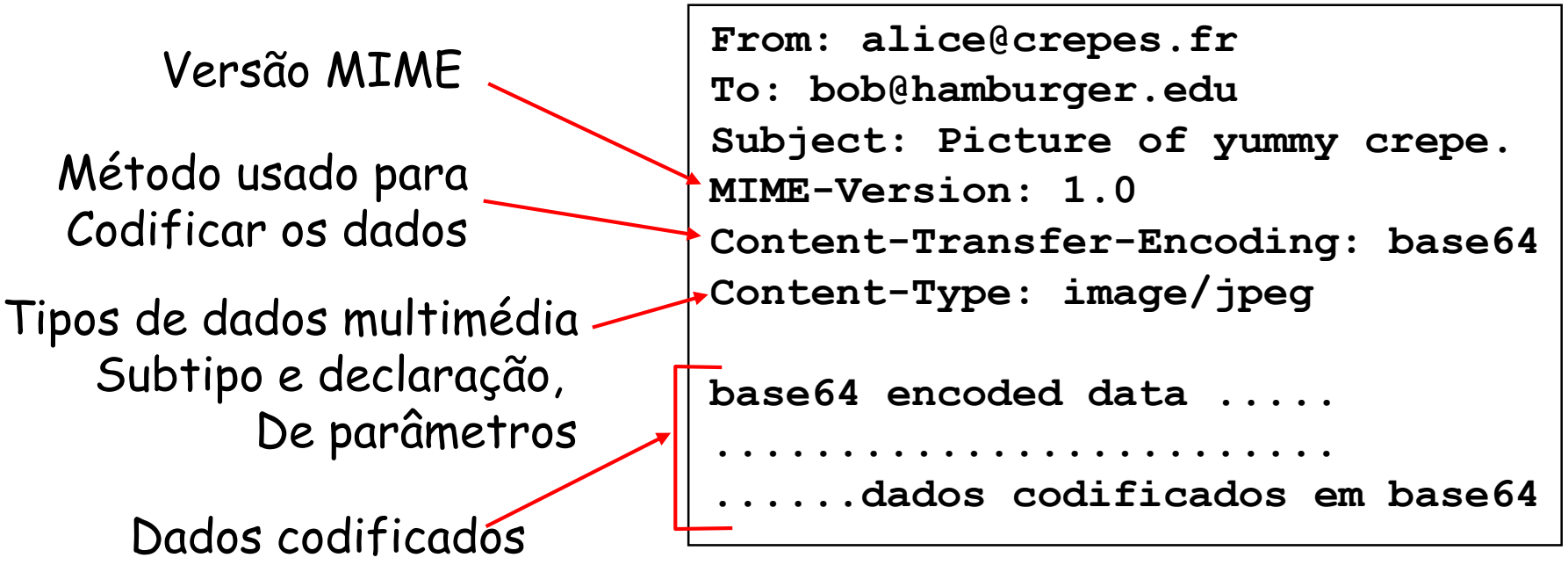
- Linhas de cabeçalho, e.g.,
  - To:
  - From:
  - Subject:

  diferente dos comandos SMTP
- corpo
  - A mensagem com apenas caracteres ASCII



Formato das mensagens: extensões multimédia

- MIME: multimedia mail extension, RFC 2045, 2056
- Linhas adicionais no cabeçalho declaram o tipo de conteúdo MIME



Tipos MIME

Content-type: tipo/subtipo; parâmetros

- Texto (text)

Exemplo de subtipo:

Plain, html

- Imagem (image)

Exemplo de subtipo:

Jpg, gif

- Áudio (audio)

Exemplo de subtipo:

Basic(codificação 8 bits mu-law), 32 kbps pcm

- Vídeo (video)

Exemplo de subtipo:

mpeg, quicktime

- Aplicação (application)

Outros dados que têm de ser processado pelo leitor antes de serem visíveis

Exemplo de subtipo:

Msword, octet-stream

Tipo Multipart

```
From: alice@crepes.fr
To: bruno@hamburger.edu
Subject: Imagem de saboroso doce.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=StartOfNextPart

--StartOfNextPart
Caro Bruno, junto envio imagem de um doce.
--StartOfNextPart
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
dados codificados em base64 .....
.................................
......dados codificados em base64
--StartOfNextPart
Queres a receita?
```

Tipo Multipart - recepção

```
Received: from crepes.fr by hamburger.edu; 12 Oct 98
From: alice@crepes.fr
To: bruno@hamburger.edu
Subject: Imagem de saboroso doce.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=StartOfNextPart

--StartOfNextPart
Caro Bruno, junto envio imagem de um doce.
--StartOfNextPart
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
dados codificados em base64 .....
.................................
......dados codificados em base64
--StartOfNextPart
Queres a receita?
```

Tipo Multipart - encaminhamento

```
Received: from hamburger.edu by sushi.jp; 12 Oct 98 15:30:01 GMT
Received: from crepes.fr by hamburger.edu; 12 Oct 98 15:27:39 GMT
From: alice@crepes.fr
To: bruno@hamburger.edu
Subject: Imagem de saboroso doce.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=StartOfNextPart

--StartOfNextPart
Caro Bruno, junto envio imagem de um doce.
--StartOfNextPart
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
dados codificados em base64 .....
.................................
......dados codificados em base64
--StartOfNextPart
Queres a receita?
```

Estudo de um caso: Vírus Klez - Cabeçalho

```
From biomedic@brturbo.com.br Wed May  8 21:58:05 2002
Received: from smtp.brturbo.com ([200.199.201.47])
        by inesc.inesc.pt (8.9.3/8.9.3) with ESMTP id VAA96840
        for <webmaster@mariel.inesc.pt>; Wed, 8 May 2002 21:57:12 +0100 (WEST)
        (envelope-from biomedic@brturbo.com.br)
Received: from Kdccjtcr (unknown [200.163.62.20])
        by smtp.brturbo.com (Postfix) with SMTP id 006D611E620
        for <webmaster@mariel.inesc.pt>; Wed,  8 May 2002 17:50:17 -0300 (BRT)
From: kit <kit@microcontrolador.com>
To: webmaster
Subject: Japanese girl VS playboy
Mime-Version: 1.0
Content-Type: multipart/alternative;
        boundary=AHdz7JR6BD7Tk7N1IvZU9l1q90A5tX90j3
Message-Id: <20020508205018.006D611E620@smtp.brturbo.com>
Date: Wed,  8 May 2002 17:50:18 -0300 (BRT)
```

Destino obtido numa página WEB

Origem forjada

Utilizador infectado

Assunto supostamente apelativo

Verme (Worm) - vírus que se propaga por correio electrónico

Fazer vírus é punível com prisão!

Estudo de um caso: Vírus Klez - Mensagem

```
--AHdz7JR6BD7Tk7N1IvZU9l1q90A5tX90j3
Content-Type: text/html;
Content-Transfer-Encoding: quoted-printable

<HTML><HEAD></HEAD><BODY>
<iframe src=3Dcid:Rh5Y20734J height=3D0 width=3D0>
</iframe>
<FONT></FONT></BODY></HTML>

--AHdz7JR6BD7Tk7N1IvZU9l1q90A5tX90j3
Content-Type: audio/x-midi;
        name=kitty.pif
Content-Transfer-Encoding: base64
Content-ID: <Rh5Y20734J>

TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAA2AAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaW4g
(…)
--AHdz7JR6BD7Tk7N1IvZU9l1q90A5tX90j3--
```

HTML com suposto som para música de fundo (extensão da Microsoft)

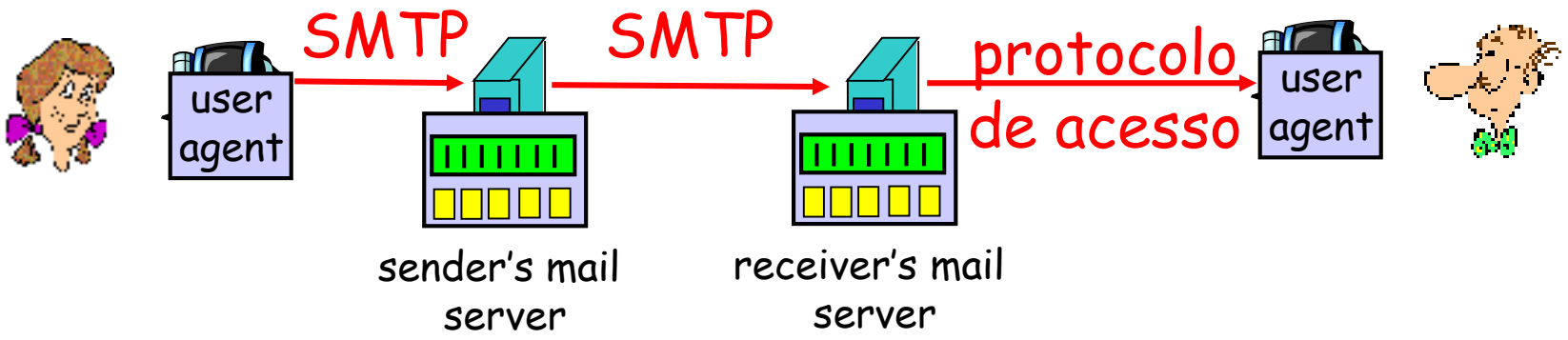
Programa executável com o vírus

Bug: ao tocar a "música", corre o vírus!
Basta ver a mensagem para ficar infectado!

Estudo de um caso: Vírus Klez
Moral da História

- Programadores:
  - verificar cuidadosamente toda a informação que chega da rede
    - se os dados fazem sentido, bytes a mais ou a menos
- Administradores:
  - instalar antivírus, actualizar as definições de vírus c/frequência
  - aplicar periodicamente "patches" de segurança
  - monitorizar explosões de tráfego na rede
- Utilizadores
  - não pôr o endereço de correio electrónico directamente em páginas web (usar imagem, Javascript)
  - desconfiar de anexos no correio com programas executáveis, com dupla extensão, ou com nomes que parecem ligações web:
    - Ex: Veja as fotos fantásticas em: www.myparty.yahoo.com

Protocolos de acesso ao correio



- SMTP: entrega/armazena correio no servidor
- Protocolo de acesso ao correio: obter correio do servidor do receptor
  - POP: Post Office Protocol [RFC 1939]
    - Autorização (agent <-->server) e download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - Mais funcionalidades (mais complexo)
    - Manipulação das mensagens guardadas no servidor
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

Protocolo POP3

Fase de autorização

- Comandos do cliente:
  - `user`: declara username
  - `pass`: password
- Servidor responde
  - `+OK`
  - `-ERR`

Fase de transacção, cliente:

- `list`: lista nº das mensagens
- `retr`: obtem mensagem através do nº
- `dele`: apaga
- `Quit`: termina

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

POP3 e IMAP

Mais sobre POP3

- O exemplo anterior usa o modo "descarrega e apaga".
- Bob não pode voltar a ler o correio se mudar de programa
- "Download-and-keep": cópias de mensagens em clientes diferentes
- POP3 não mantem estado entre sessões

IMAP

- Mantem as mensagens num único lugar: o servidor
- Permite ao utilizador organizar as mensagens em pastas
- IMAP mantem o estado entre sessões:
  - Nome das pastas e mapeamento entre IDs das mensagens
- Utilizador pode obter apenas componentes específicos da mensagem

Nível de Aplicação

- 2.1 Aplicações de Rede
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Correio Electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de Socket com TCP
- 2.8 Programação de Socket com UDP

DNS: Domain Name System

pessoas: muitos identificadores:

- #BI, nome, #passaporte

Internet hosts, routers:

- Endereço IP (32 bit) – usado para endereçar datagramas
- "nome", e.g., www.ips.pt- usado por humanos

P: mapeamento entre endereços IP e nome?

Domain Name System:

- *Base de dados distribuida implementada numa hierarquia de muitos servidores de nome*
- *Protocolo de nível Aplicação* sistemas terminais, routers, servidores de nomes comunicam para resolver nomes (tradução endereço/IP)
  - nota: função do núcleo da Internet, implementada como protocolo de aplicação
  - Complexidade na periferia da rede"

DNS

Serviços DNS

- Tradução nome/endereço IP
- host aliasing
  - Canonical, alias names
- mail server aliasing
- Distribuição de carga
  - replica servidores Web : conjunto de endereços IP para apenas um nome canónico

Porque não centralizar DNS?

- Um só ponto de falha
- Volume de tráfego
- Base de dados centralizada distante
- manutenção

Não escala!

Nenhum servidor tem todos os mapeamentos nome/IP

DNS

Servidore de nomes locais (Local name server)

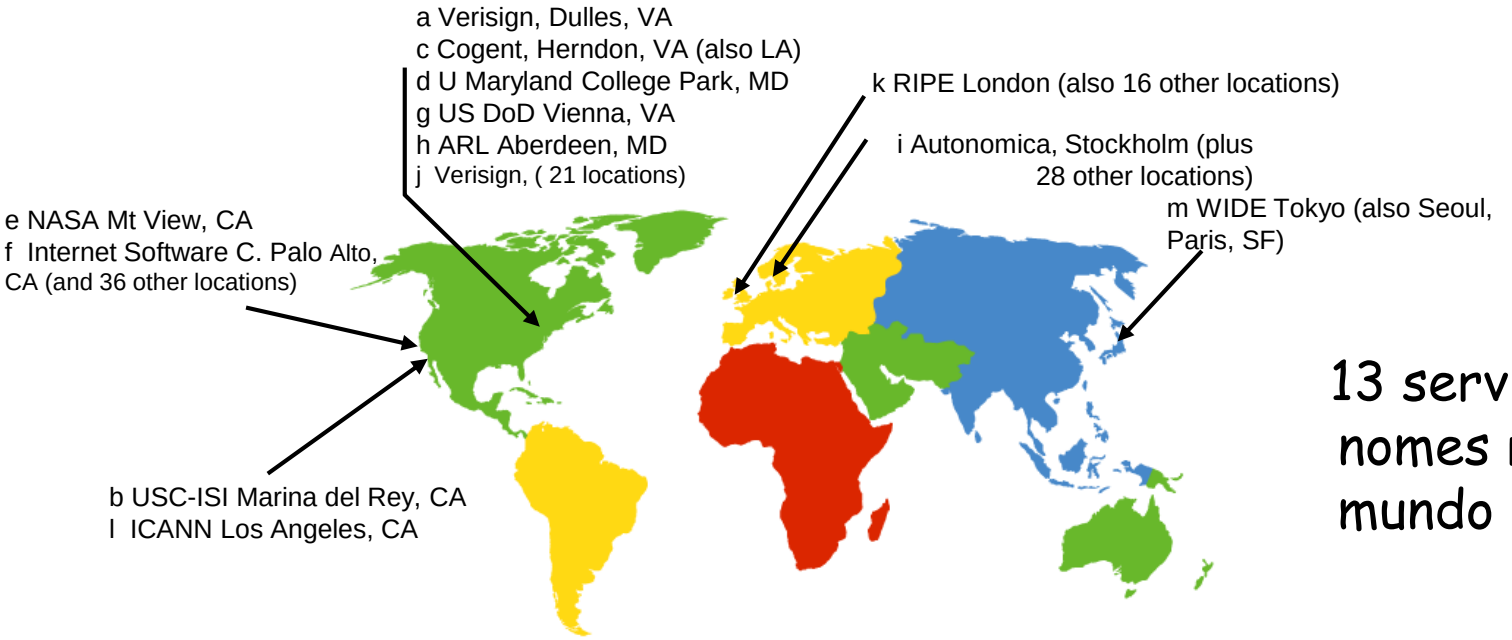
- Cada ISP ou instituição tem um servidor de nomes local (default)
- Sistemas terminais interrogam os servidor de nomes local
- Caso este não consiga resolver o nome, o servidor local questiona o seu superior hierárquico

Servidor de nomes oficial (authoritative name server)

- Para um sistema terminal: guarda o seu nome e endereço IP
- Pode executar a tradução nome/endereço para esse sistema terminal

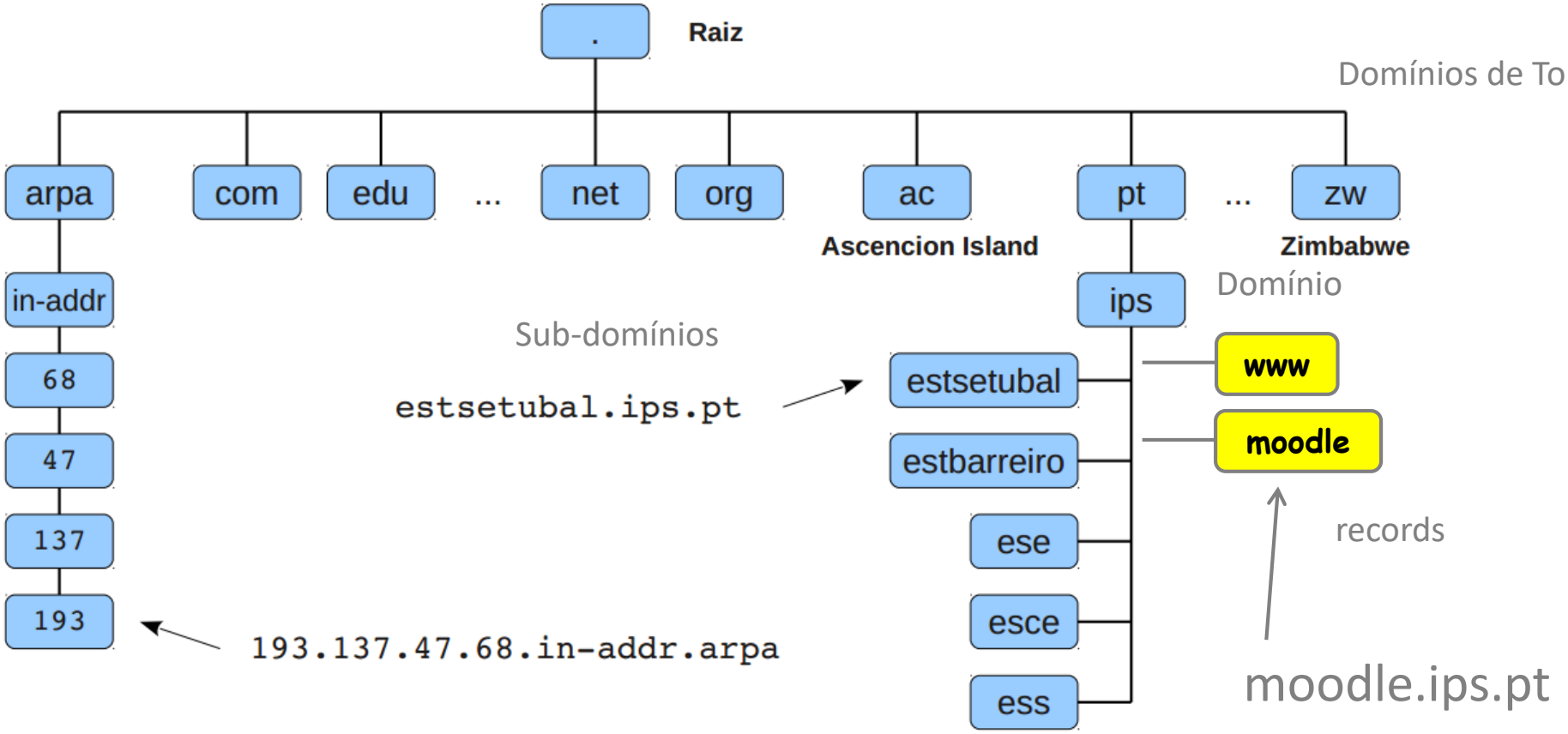
DNS: servidores de nomes raiz (Root Name Servers)

- Contactados por servidores de nomes locais que não sabem resolver nomes
- root name server:
  - Contacta o servidor de nomes oficial quando não sabe o mapeamento do nome
  - Obtem o mapeamento e Devolve o mapeamento ao servidor de nomes local



13 servidores de nomes raiz no mundo

DNS: Hierarquia



TLD e servidores oficiais

- Servidores Top-Level Domain (TLD) :
  - responsáveis por com, org, net, edu, etc, e todos os top-level domínios dos países uk, fr, ca, jp.
  - Network Solutions mantem os servidores para o domínio com TLD
  - Educause para domínio edu TLD
- Servidores Authoritative DNS :
  - Servidores DNS das organizações, fornecendo mapeamento entre IP e nomes dos terminais locais(e.g., Web, mail).
  - Podem ser mantidos pelas organizações ou pelos ISP's

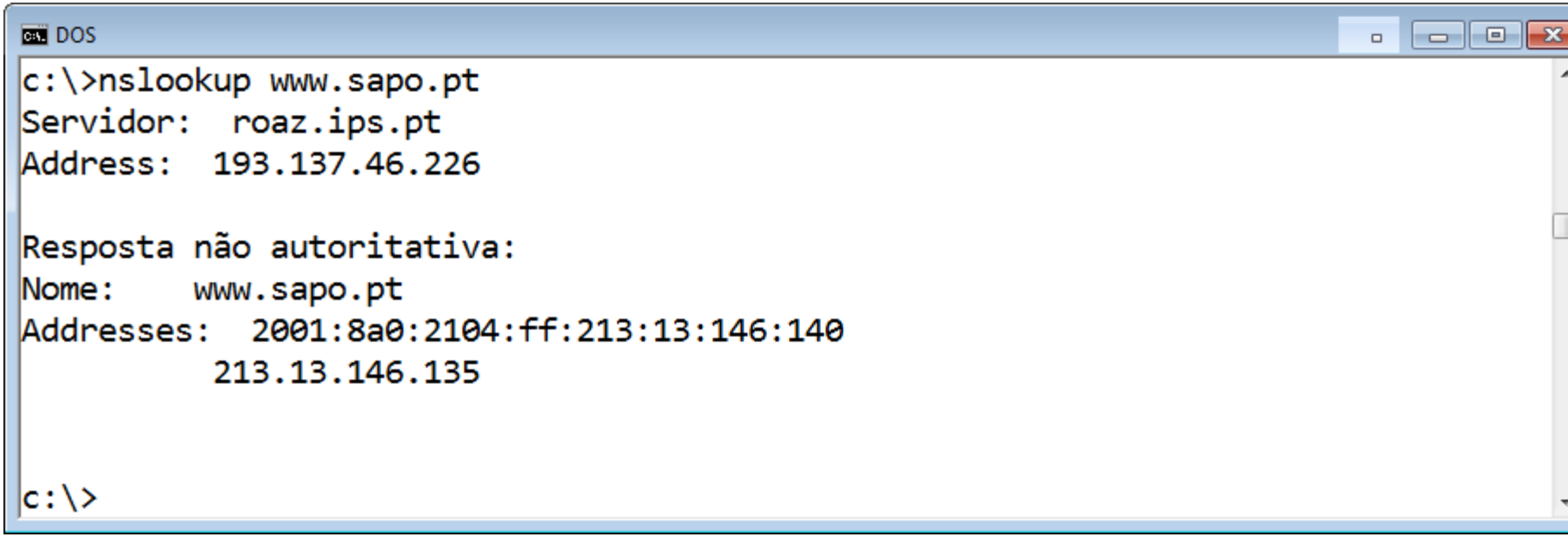
TLD e servidores oficiais

www.sapo.pt

Domínio de Topo (TLP - *Top Level Domain*)

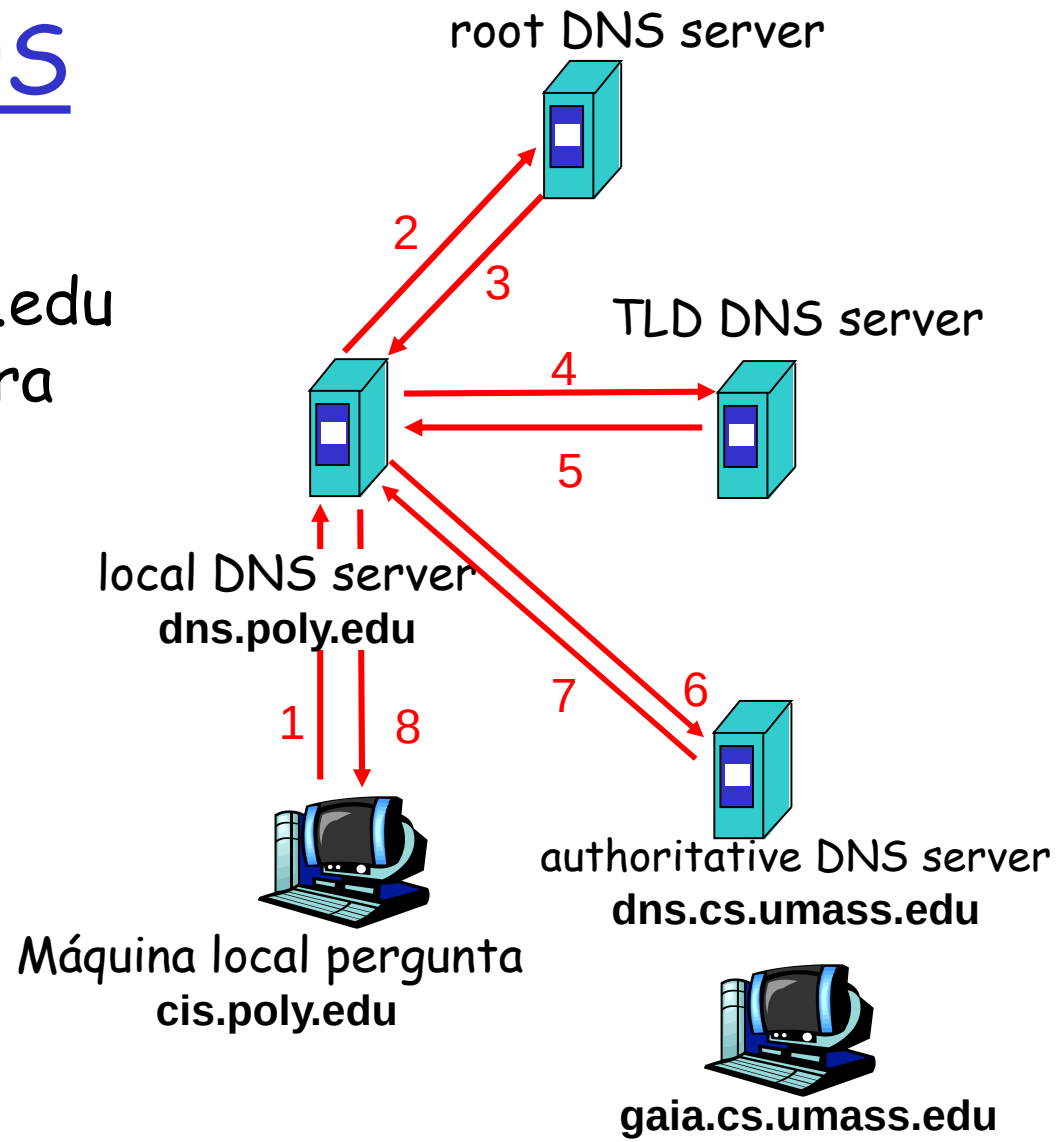
Domínio da Instituição

Serviço



Exemplo DNS

- Terminal em cis.poly.edu quer endereço IP para gaia.cs.umass.edu

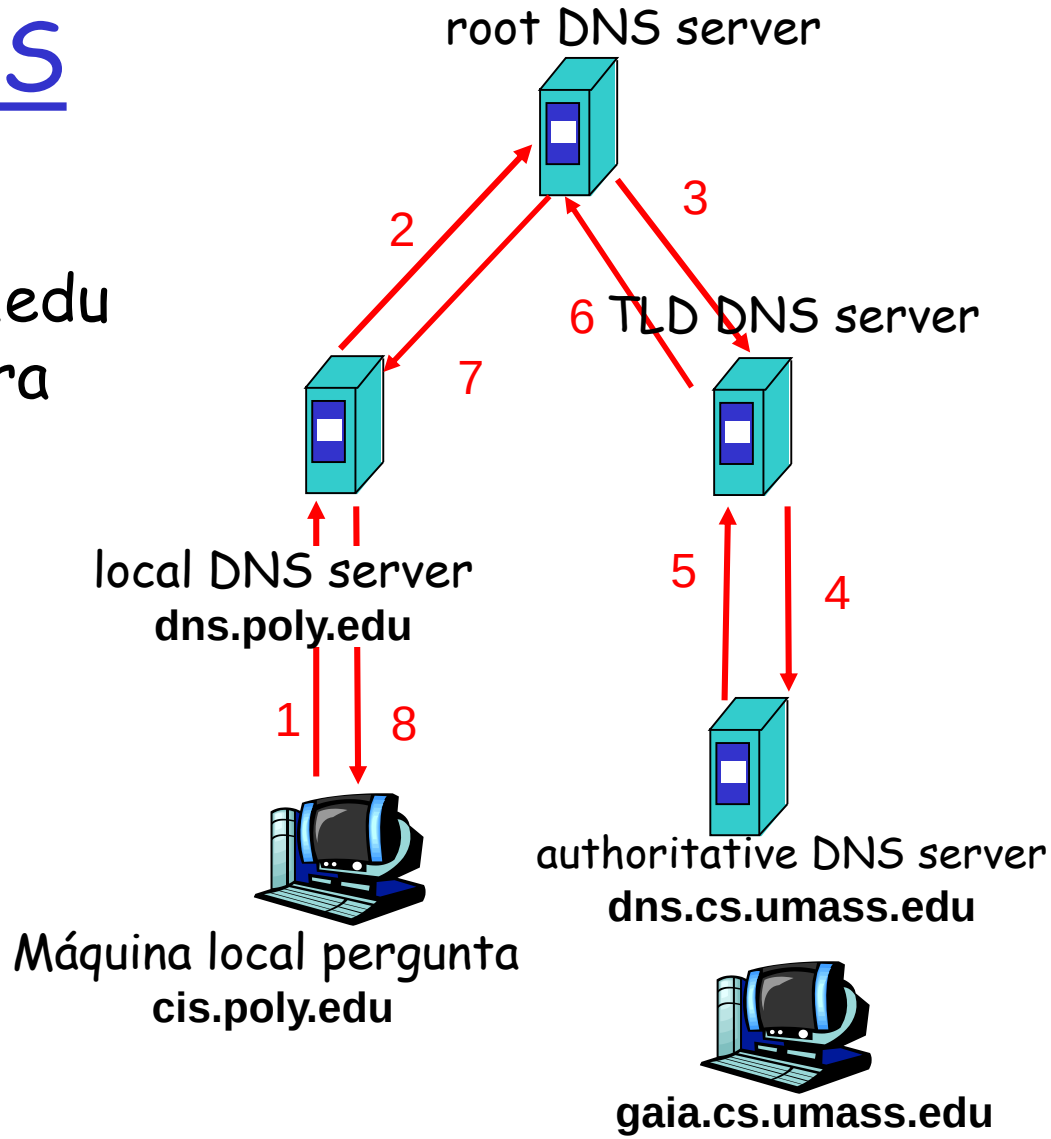


Pergunta iterativa:

- O servidor contactado responde com o nome do outro servidor a contactar
- "Não conheço este nome, mas pergunta a este servidor. Pode ser que ele saiba..."

Exemplo DNS

- Terminal em cis.poly.edu quer endereço IP para gaia.cs.umass.edu



Pergunta recursiva:

- coloca o peso da resolução de nomes no servidor de nomes contactado
- Carga pesada?

DNS: caches e actualização de registos

- Cada vez que um (qualquer) servidor de nomes aprende os mapeamentos, armazena-os na cache
  - Entradas na cache desaparecem ao fim de um certo tempo de vida (TTL)
  - Os servidores TLD ficam armazenados na cach do servidor local de nomes
    - Desta forma, os servidores root não são visitados com frequência
- Mecanismos de actualização e notificação em estudo pelo IETF
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

Registos DNS

DNS: Registo de Recurso (Resource Records, RR) armazenados numa base de dados distribuida

RR format: (**name**, **value**, **type**, **ttl**)

- Type=*A* - endereço IPv4 de domínio
  Type=*AAAA* - endereço IPv6 de domínio
  Type=CNAME - Nomes alternativos para os campos *A* ou *AAAA*
  Type=MX - endereço IP do servidor de E-mail do domínio
  Type=PTR - mapeia endereço IP para o nome do servidor
  Type=NS - endereço IP do servidor de DNS do domínio
  Type=SOA - indica o servidor DNS com autoridade sobre o domínio

Registos DNS Exemplo

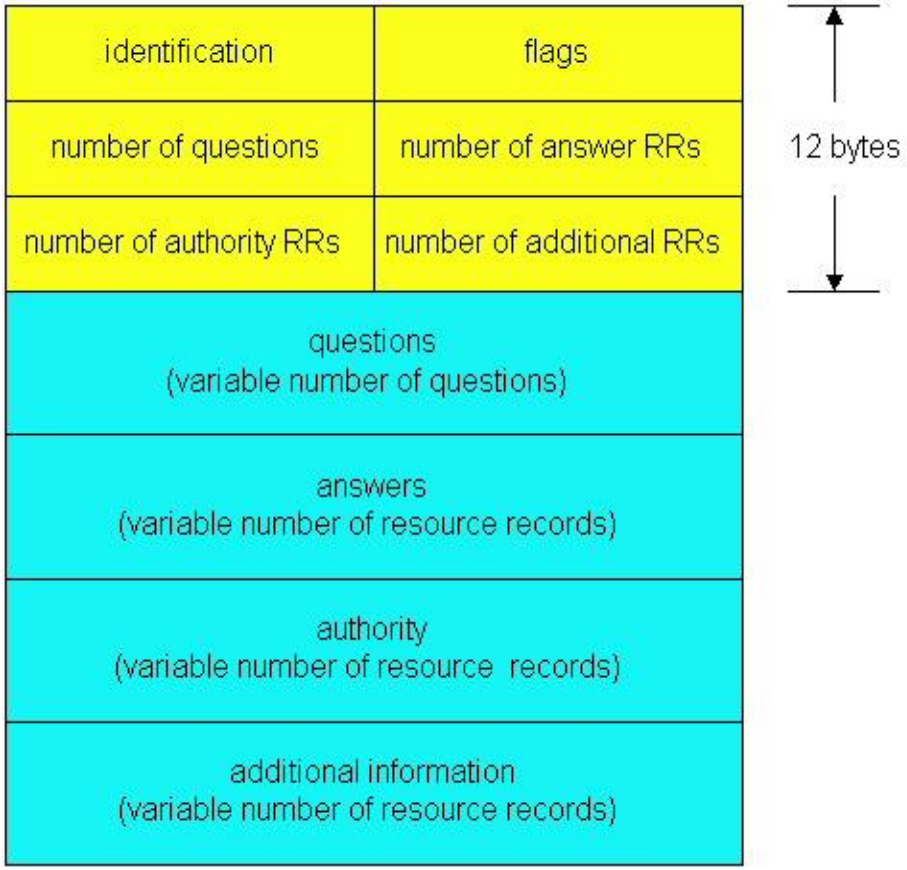
```
$TTL    86400 ; 24 hours could have been written as 24h or 1d
; $TTL used for all RRs without explicit TTL value
$ORIGIN example.com.
@  1D  IN  SOA ns1.example.com. hostmaster.example.com. (
                                2002022401 ; serial
                                3H ; refresh
                                15 ; retry
                                1w ; expire
                                3h ; minimum
                               )
       IN  NS     ns1.example.com. ; in the domain
       IN  NS     ns2.smokeyjoe.com. ; external to domain
       IN  MX  10 mail.another.com. ; external mail provider
; server host definitions
ns1    IN  A      192.168.0.1  ;name server definition
www    IN  A      192.168.0.2  ;web server definition
ftp    IN  CNAME  www.example.com.  ;ftp server definition
; non server domain hosts
bill   IN  A      192.168.0.3
fred   IN  A      192.168.0.4
```

Protocolo DNS: mensagens

Protocolo DNS : mensagens de *perguntas* e *respostas*, ambas com o mesmo formato de mensagem

Cabeçalho da msg

- identificação: 16 bit # para pedidos e respostas a pedidos
- flags:
  - Pedido ou resposta
  - recursividade desejada
  - recursividade disponível
  - Resposta é oficial



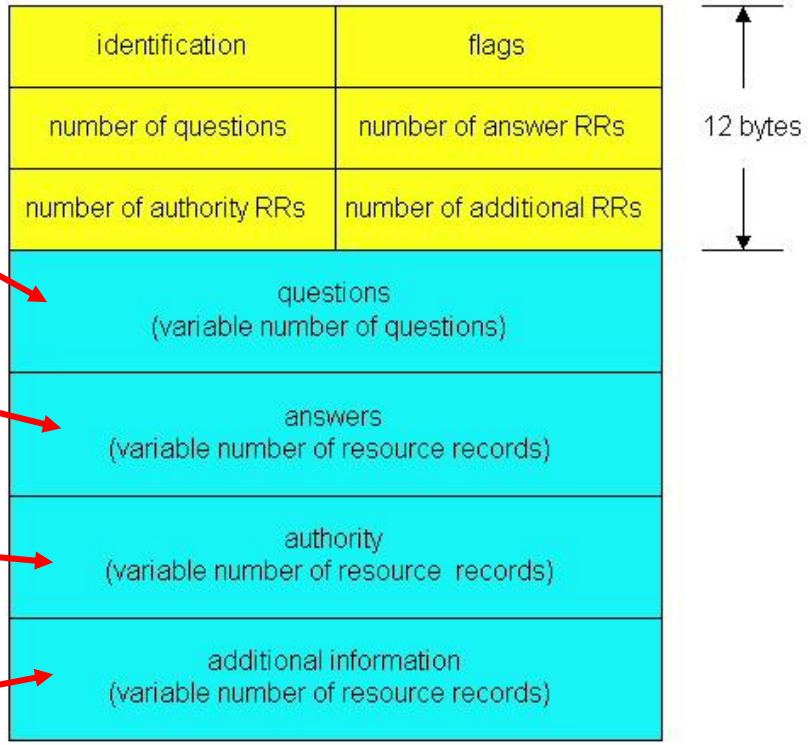
protocolo DNS: mensagens

Nome, tipo de campo para um padido

RRs em resposta a pedido

Registo para Servidores oficiais

Informação adicional Que pode ser útil



Inserindo Records no DNS

- exemplo: new startup "Network Utopia"
- Nome de registo networkuptopia.com no *DNS registrar* (e.g., Network Solutions)
  - fornece nomes, endereços IP do servidor de nomes oficial (primario e secundario)
  - registrar insere dois RRs no servidor TLD .com:

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```

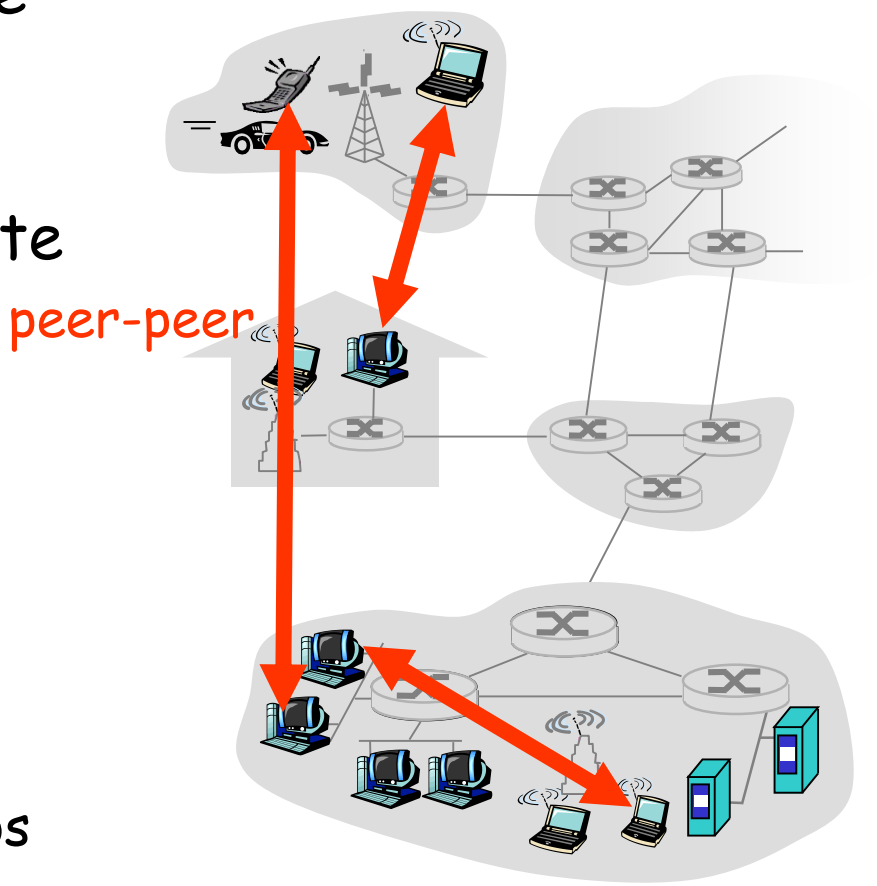
- Cria no servidor oficial um registo Type A para www.networkuptopia.com; registo Type MX for networkutopia.com
- Como é que as pessoas conhecem o IP do seu Web site?

Nível de Aplicação

- 2.1 Aplicações de Rede
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Correio Electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de Socket com TCP
- 2.8 Programação de Socket com UDP

Arquitectura P2P pura

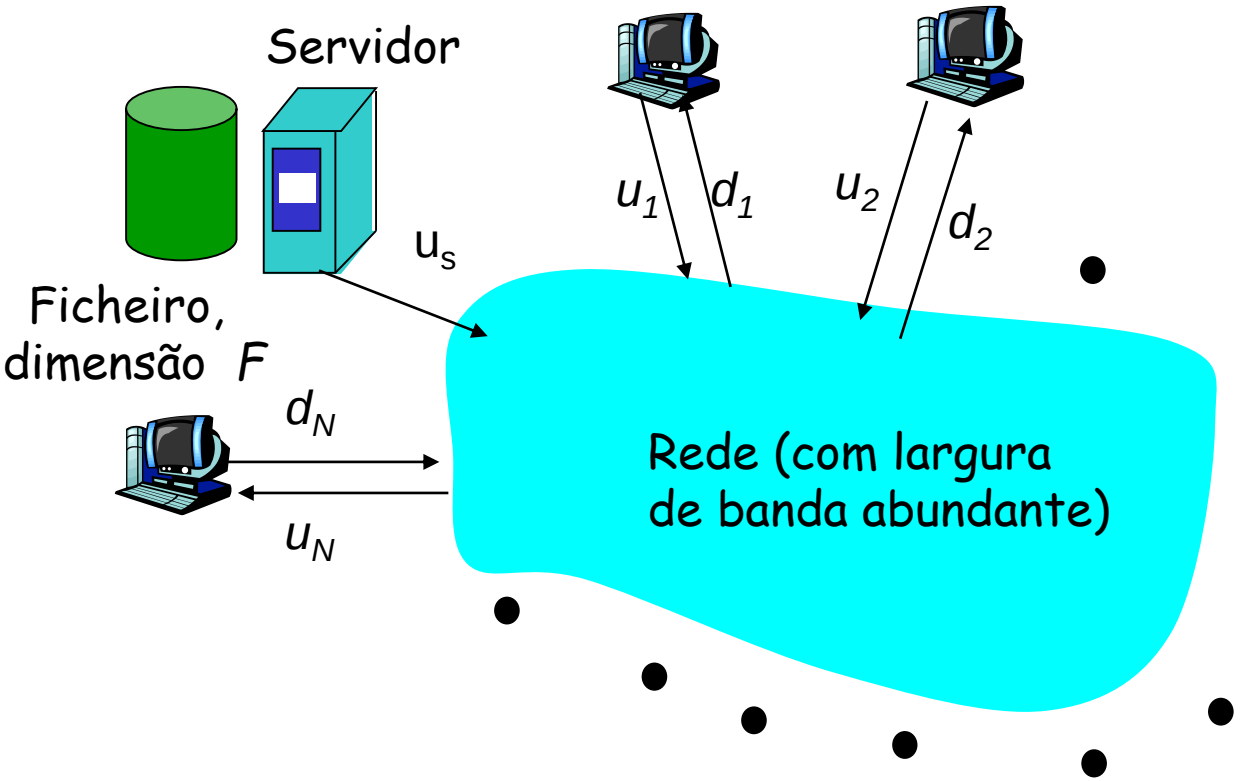
- *Servidores nem sempre ligados*
- Sistemas terminais comunicam directamente
- Sistemas terminais estão ligados intermitentemente e mudam de endereço IP



- Três tópicos:
  - Distribuição de ficheiros
  - Procura de informação
  - Case Study: Skype

Duistribuição de ficheiros: Servidor-Cliente vs P2P

Questão : Quanto tempo leva a distribuição de um ficheiro de um servidor para N terminais?



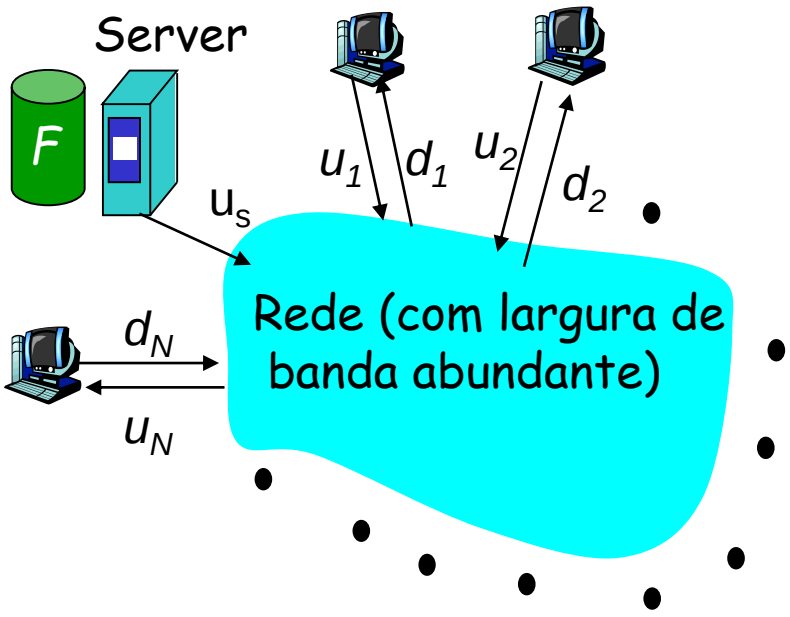
u_s: largura de banda de upload do servidor

u_i: largura de banda de upload do terminal I

d_i: largura de banda de download do terminal i

Tempo de distribuição de ficheiro: servidor-cliente

- Servidor envia sequencialmente N cópias:
  - NF/u_s
- cliente i leva F/d_i para o download



Tempo para distribuir *F*

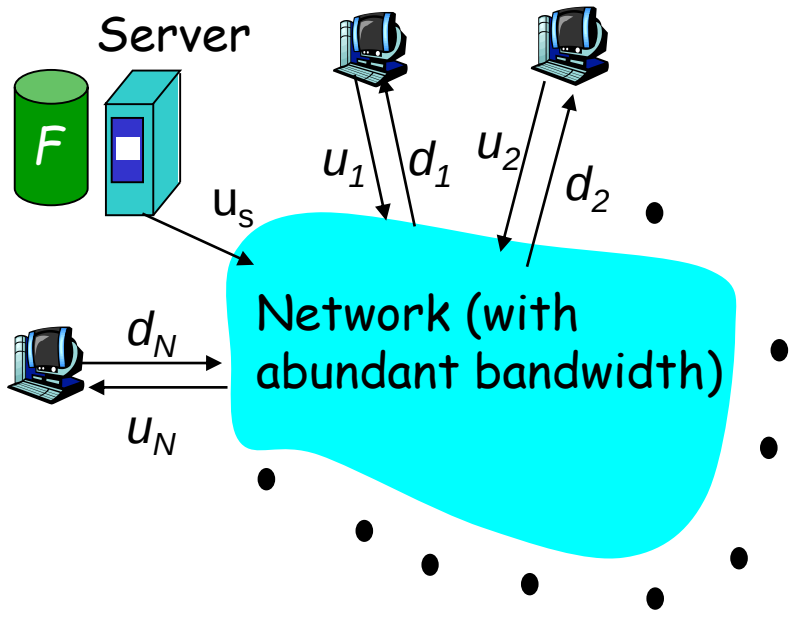
Para os *N* clientes com = $d_{cs} = \max \{ NF/u_s, F/\min_i(d_i) \}$

aproximaçãoclient/server

Aumento linear em N (para grandes N)

Tempo de distribuição de ficheiro: P2P

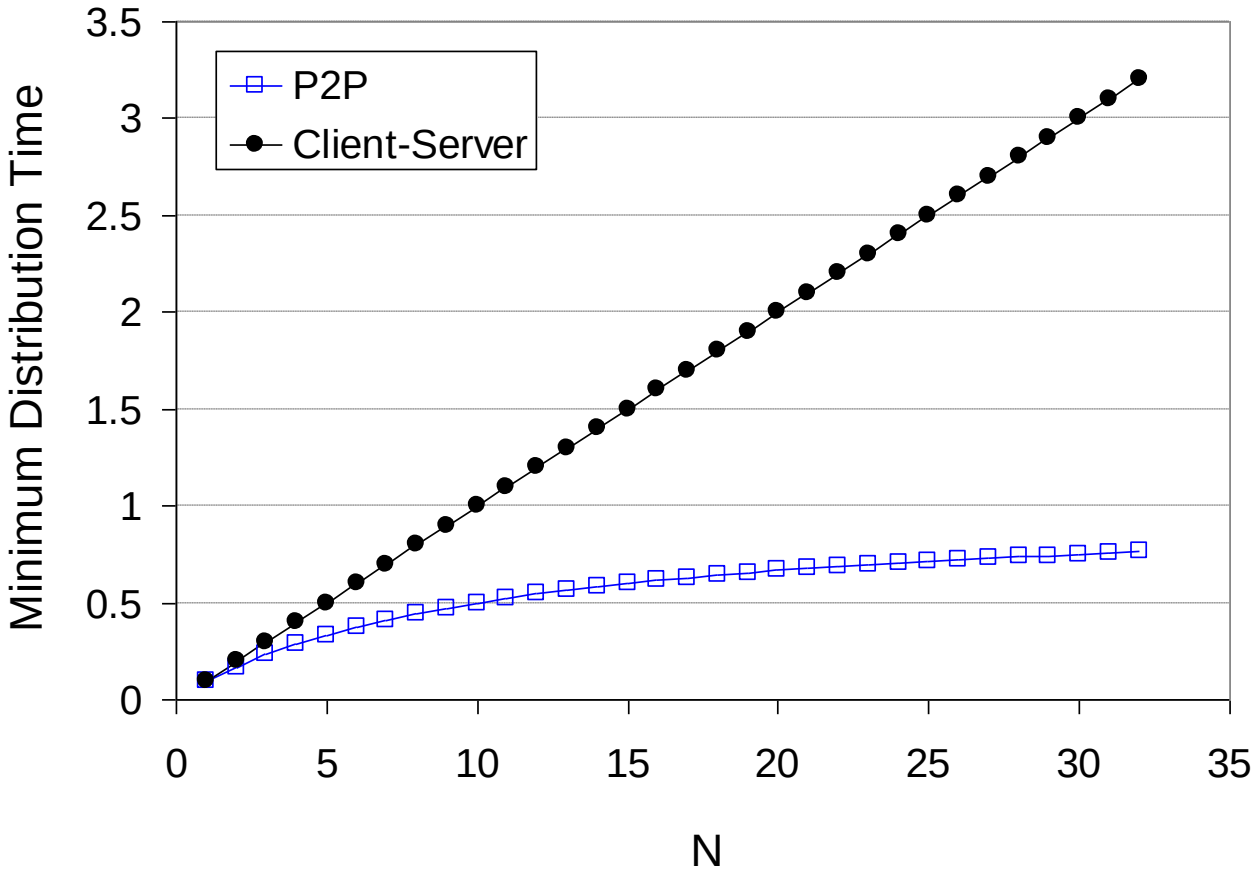
- servidor deve enviar uma cópia: F/u_s
- cliente i leva F/d_i para download
- NF bits devem ser descarregados (agregados)
  - Máxima taxa possível de upload : $u_s + \Sigma u_i$



$$d_{P2P} = \max \{ F/u_s, F/\min(d_i)_i, NF/(u_s + \Sigma u_i) \}$$

Servidor-cliente vs. P2P: exemplo

Client upload rate = u, F/u = 1 hora, u_s = 10u, $d_{min} \geq u_s$

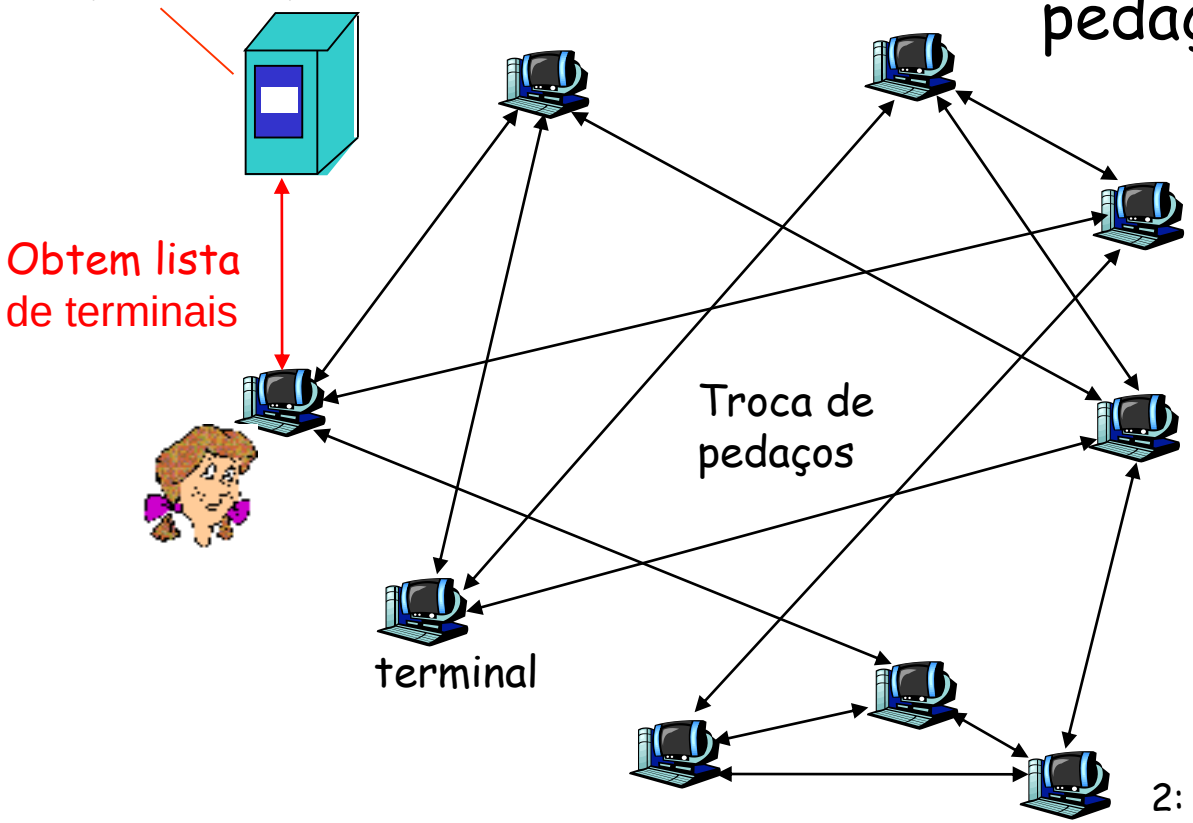


Distribuição de ficheiros: BitTorrent

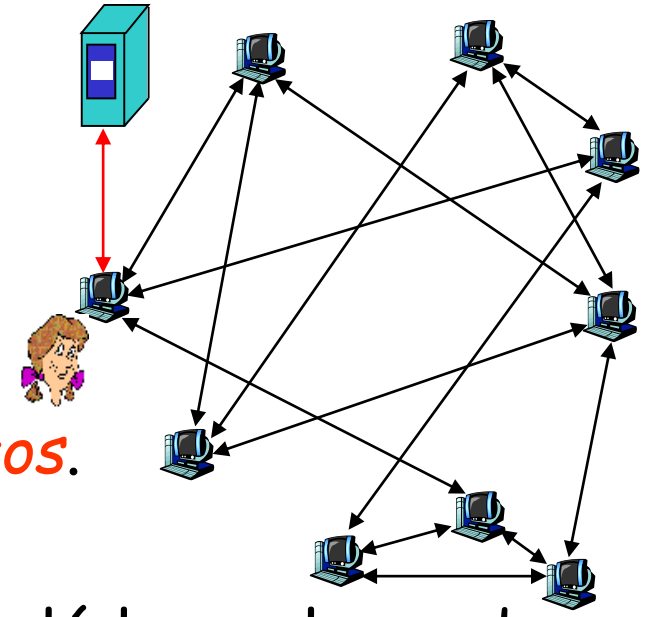
- Distribuição de ficheiros P2P

tracker: regista terminais participantes no torrent

torrent: grupo de terminais trocando pedaços de ficheiro



BitTorrent (1)



- Ficheiro dividido em 256KB *pedaços*.
- pares que se juntam ao torrent:
  - Não têm pedaços, mas irá acumulá-los ao longo do tempo
- Enquanto faz download, o par envia pedaços para outros pares
- Pares podem chegar e partir
- Uma vez que o par tem o ficheiro completo, pode (egoista) sair ou (altruista) ficar

BitTorrent (2)

Enviando pedaços

- Num determinado momento, diferentes pares têm diferentes partes do ficheiro
- periodicamente, um par (Alice) pergunta a cada vizinho por uma lista de pedaços que ele tenha
- Alice envia pedido para os seus pedaços faltantes
  - Mais raros primeiro

Enviando pedaços: tit-for-tat

- Alice envia pedaços para 4 vizinhos enviando os seus pedaços à taxa mais elevada
  - Reavalia top 4 a cada 10 secs
- A todos os 30 secs: aleatoriamente selecciona outros pares, começando a eniar pedaços
  - Os novos pares podem chegar ao top 4

P2P: procurando informação

Indexação em sistemas P2P: mapeia informação com a localização dos pares (local = endereço IP & port number)

Partilha de ficheiro (eg e-mule)

- Índice recolhe dinamicamente o local dos ficheiros partilhados pelos pares.
- Pares comunicam os ficheiros que possuem.
- Pares procuram índice para saber onde estão os ficheiros

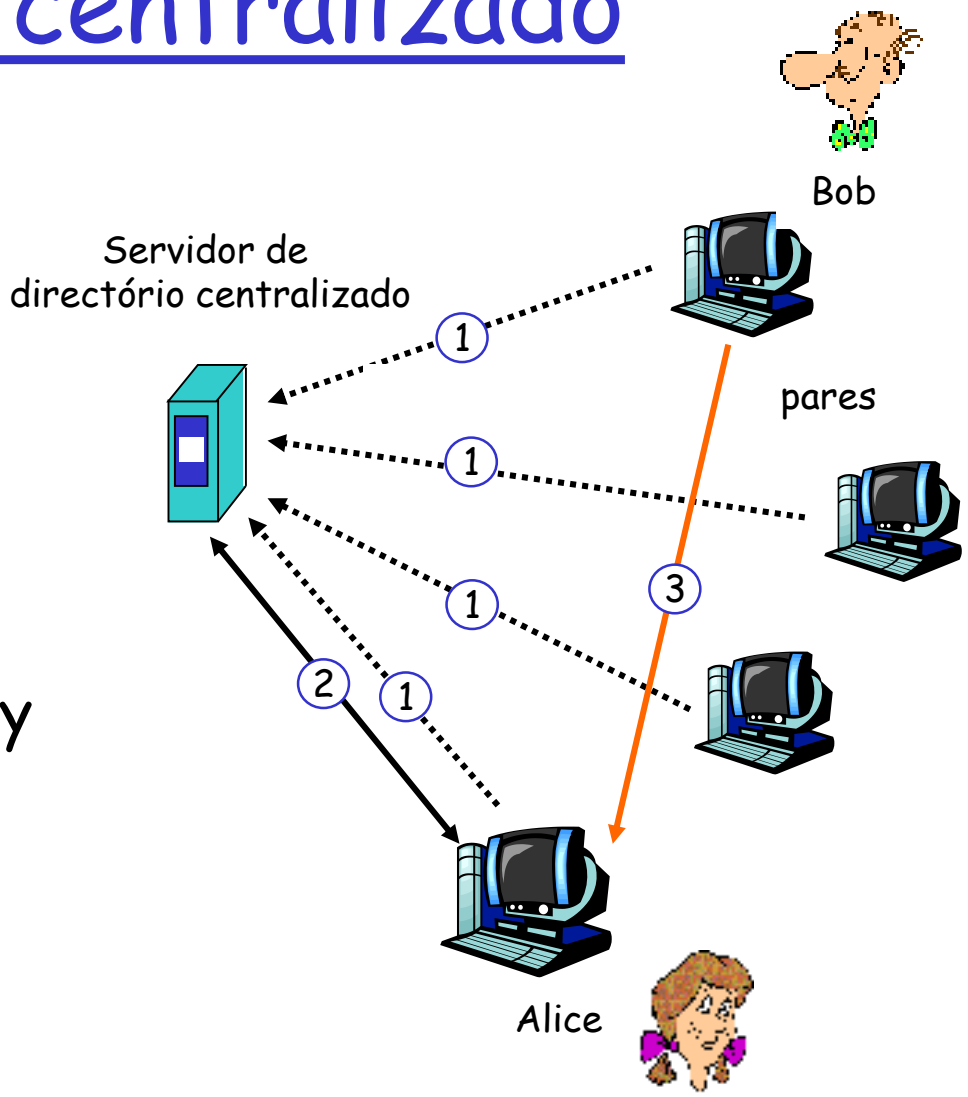
Instant messaging

- Índice mapeia os utilizadores aos locais
- Quando o utilizador inicia aplicação IM necesita informar o Índice da sua localização
- Pares procuram índice para saber endereço IP do outro par.

P2P: índice centralizado

original "Napster"

1) Quando o par se liga, informa o servidor central::
   - Endereço IP
   - conteúdo
2) Alice pergunta por "Hey Jude"
3) Alice pede ficheiro ao Bob



P2P: problemas com directórios centralizados

- Ponto único de falha
- Garrafão para o tráfego
- Violação de copyright : "alvo" da perseguição policial e jurídica é óbvia

transferência de ficheiros é descentralizada, mas a localização dos conteúdos é altamente centralizada.

Inundação: Query flooding

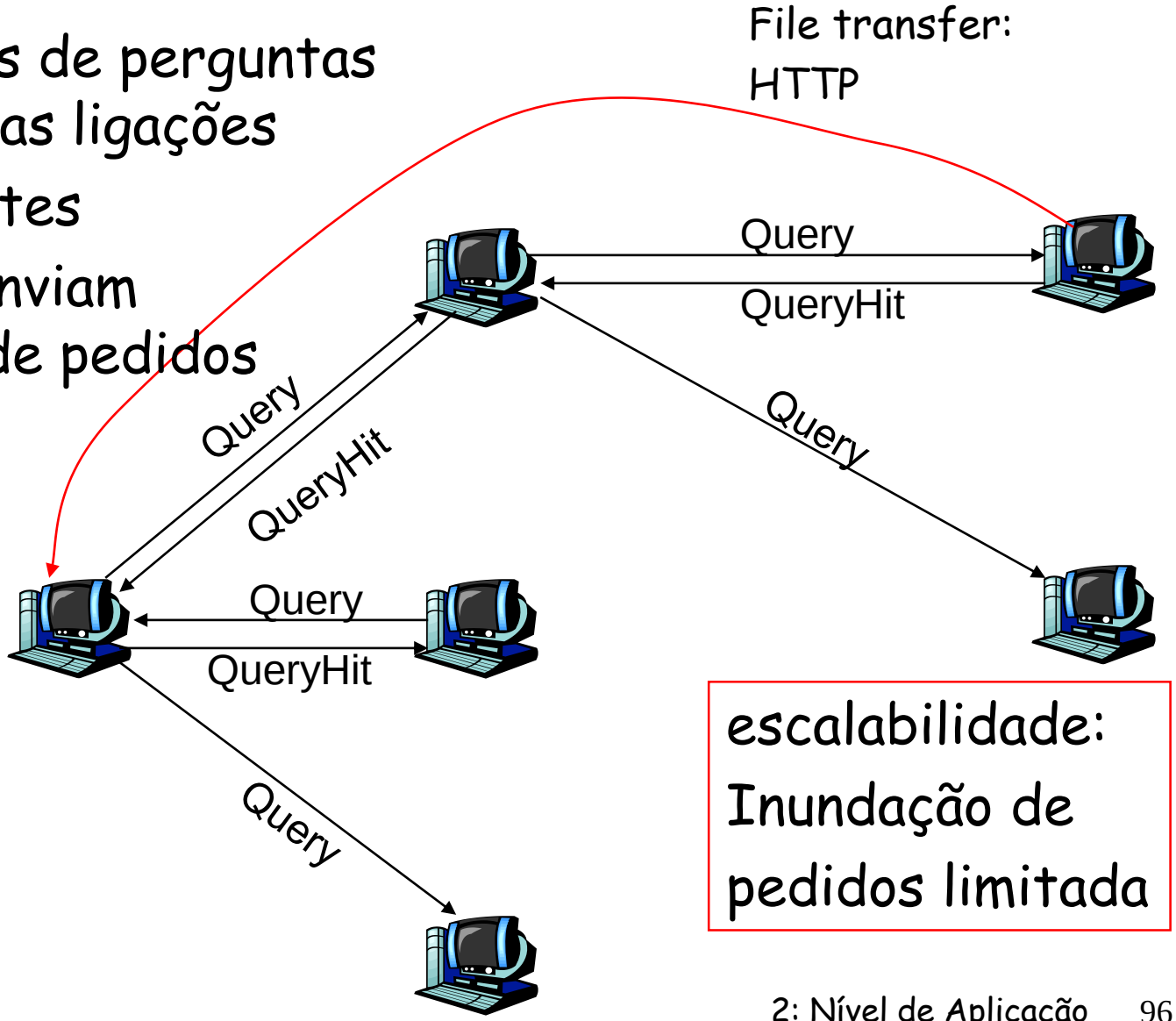
- Completamente distribuido
  - Sem servidor central
- Usado por Gnutella
- Cada par indexa os ficheiros de disponibiliza para partilha (e apenas estes)

Rede sobreposta: grafo

- Fronteira entre par X e par Y se houver ligação TCP.
- Todos os pares activos e fronteiras formam uma rede sobreposta.
- fronteira: ligação virtual (não física)
- Par tipicamente ligado a 10 redes vizinhas sobrepostas

Query flooding

- mensagens de perguntas enviadas pelas ligações TCP existentes
- pares reenviam mensagens de pedidos
- QueryHit (sucesso) enviado pelo caminho de retorno



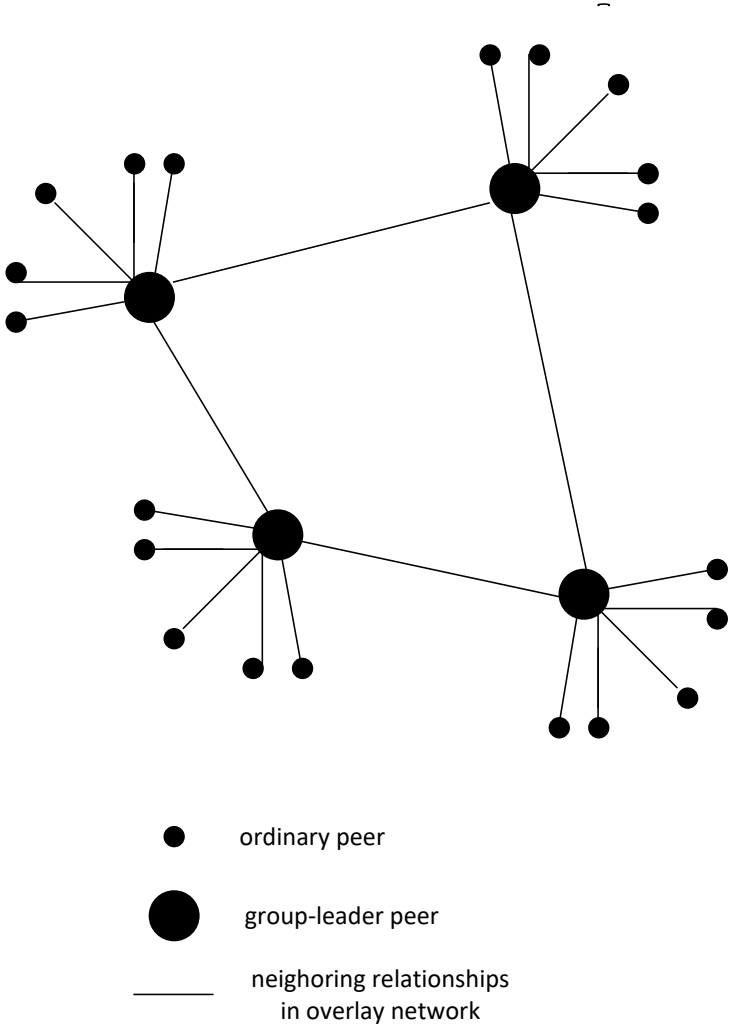
escalabilidade: Inundação de pedidos limitada

Gnutella: adição de pares

1. O par Alice deve encontrar um novo par na rede Gnutella : usa lista de pares candidatos
2. Alice tenta sequencialmente ligações TCP com candidatos a par até estabelecer até estabelecer ligação com Bob
3. *Flooding:* Alice envia mensagens Ping para Bob; Bob reenvia mensagens Ping para a sua rede vizinha (que por sua vez enviará para os seus vizinhos ….)
   - Os pares que recebem a mensagem de Ping respondem à Alice com mensagem Pong
4. Alice recebe muitas mensagens Pong, e pode criar novas ligações TCP

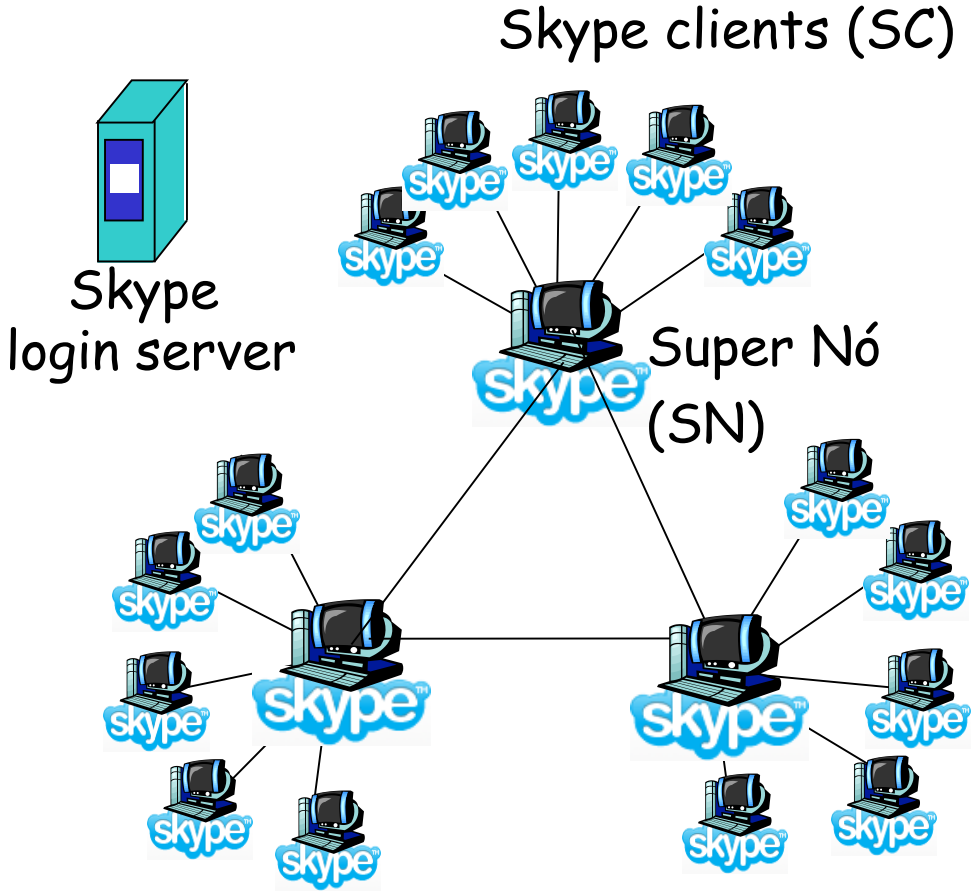
Sobreposição hierárquica

- Entre indíces centralizados: aproximação de flooding de perguntas
- Cada par ou é um super nó, ou está ligado a um
  - Ligação TCP connection entre pares e o seu super nó.
  - Ligação TCP entre pares de super nós.
- Super nós localiza conteúdos nos seus "filhos"



P2P Case study: Skype

- inerentemente P2P: pares de utilizadores comunicam
- Protocolo proprietário de aplicação proprietary
- Sobreposição hierárquica com SN
- Mapas de índice com pares user / endereço IP distribuidos nos SN



Nível de Aplicação

- 2.1 Aplicações de Rede
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Correio Electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de Socket com TCP
- 2.8 Programação de Socket com UDP

Programação de Socket

objectivo: aprender a construir aplicações cliente/servidor que comunicam através de sockets

API de Socket

- introduzido no BSD4.1 UNIX, 1981
- Criado explicitamente, utilizado e libertado pelas aplicações
- Paradigma cliente/servidor
- Dois tipos de serviço de transporte através da API de sockets:
  - Datagramas não fiáveis
  - fiável, orientado para byte stream

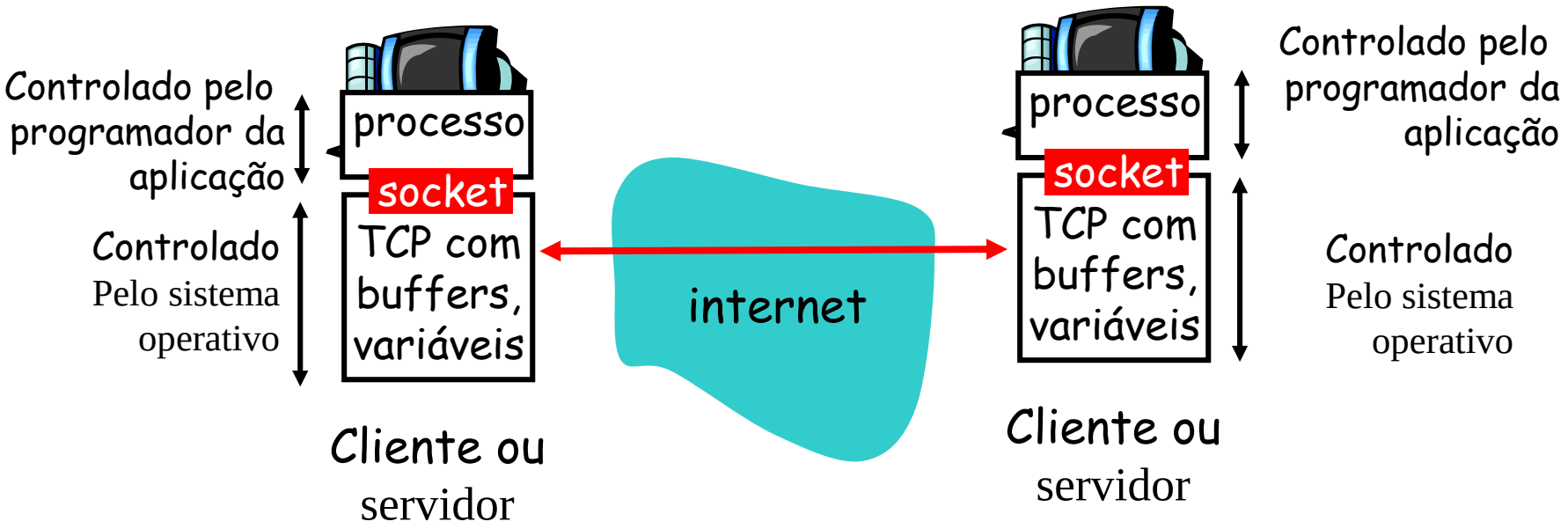
socket

Uma interface (porta) local da máquina, *criada pela aplicação*, *controlada pelo SO* para a qual o processo de aplicação tanto pode enviar como receber mensagens de/para outros processos de aplicação

Programação em socket usando TCP

Socket: uma porta entre o processo de aplicação e o protocolo de transporte extremo a extremo (UDP ou TCP)

Serviço TCP : transferência fiável de **bytes** de um processo para outro



Programação com Sockets comTCP

Cliente contacta servidor

- Processo servidor está a correr
- Servidor deve ter previsto socket para aceitar contactos dos clientes

Cliente contacta servidor :

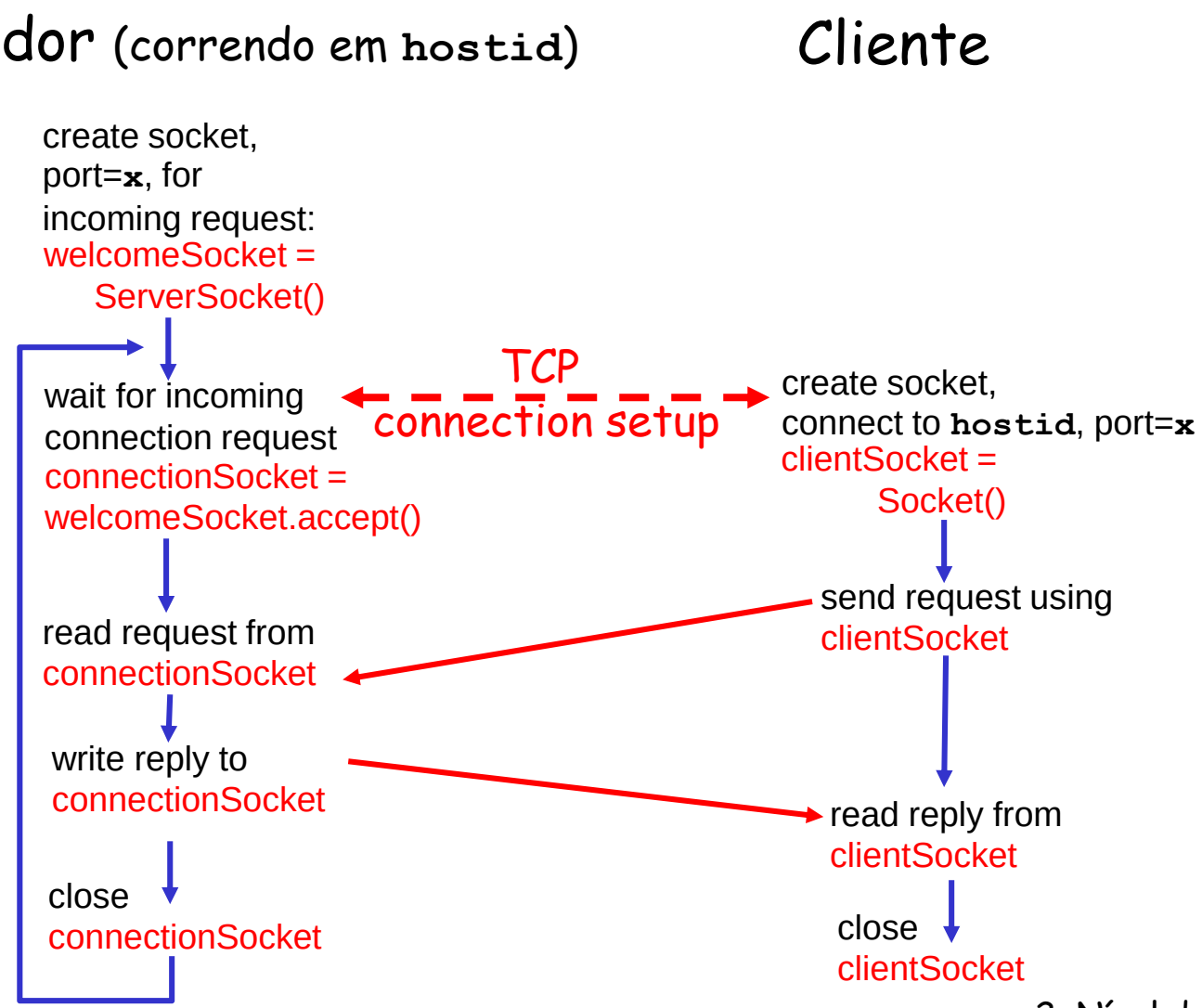
- Criando um socket local ao cliente
- Especificando o endereço IP, nº de porto do processo servidor
- Estabelecendo ligação TCP com o servidor

- Quando contactado pelo cliente, servidor TCP cria novo porto para o processo servidor comunicar com o cliente
  - Permite que o servidor fale com múltiplos clientes
  - Nº de porto de origem para distinguir clientes

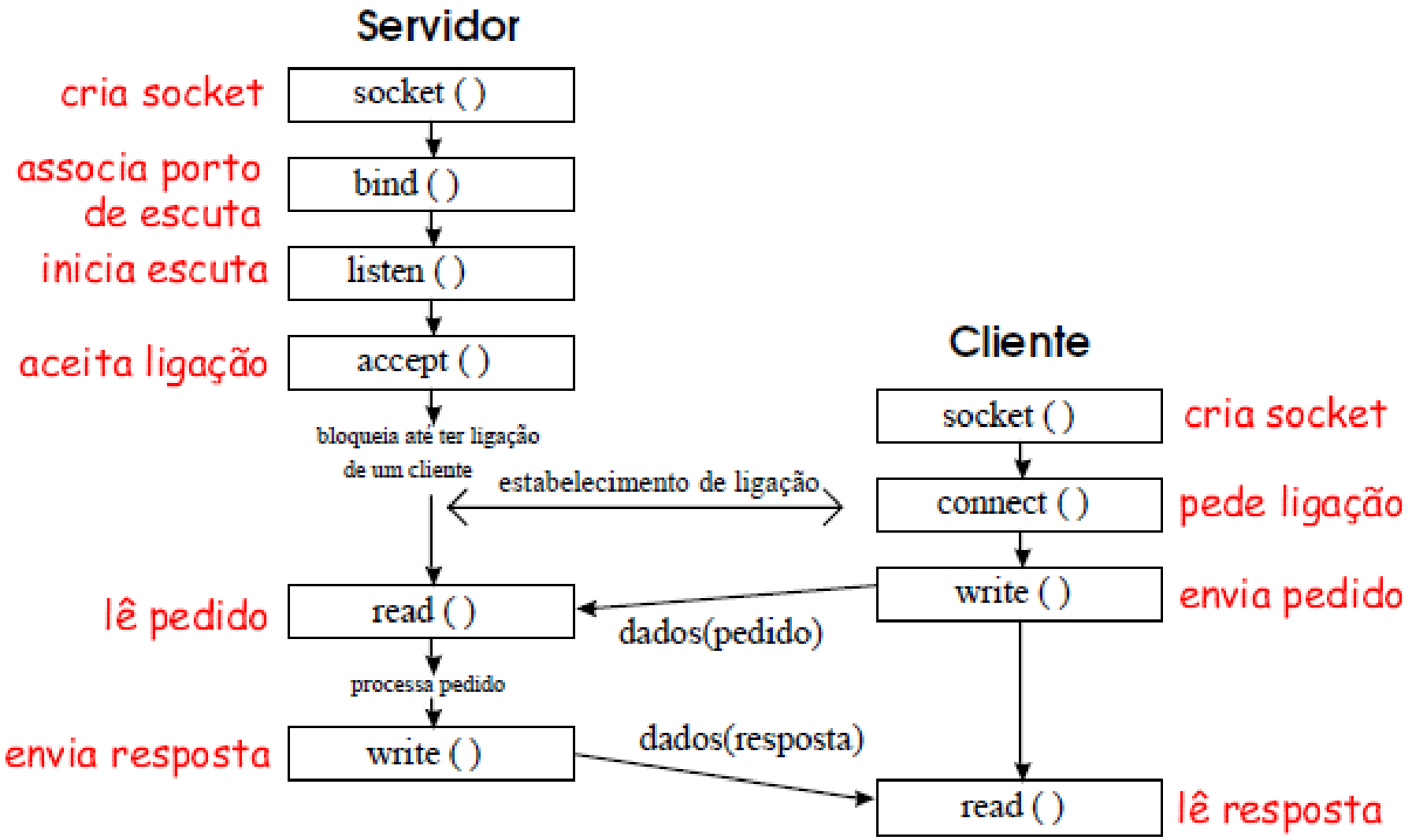
Ponto de vista da aplicação

O TCP oferece transferência fiável de bytes ("pipe") entre cliente e servidor

Interacção cliente servidor com socket: TCP



Interacção cliente servidor com socket: TCP



Programação com Sockets *comTCP*

Examplo aplicação cliente-servidor:

1) cliente lê linhas da entrada standard (stdio) (inFromUser stream) , envia para o servidor através do socket (outToServer stream)

2) Servidor lê linha do socket

3) servidor converte linha para maiúsculas, e envia de volta ao cliente

4) cliente lê e imprime a linha que vem do socket (**`inFromServer`** stream)

Exemplo: cliente Java (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("hostname", 6789);

        DataOutputStream outToServer =
          new DataOutputStream(clientSocket.getOutputStream());
```

cria input stream

cria client socket, Ligado ao servidor

cria output stream Ligado ao socket

Exemplo: cliente Java (TCP), cont.

cria input stream Ligado aosocket

```
BufferedReader inFromServer =
  new BufferedReader(new
  InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();

outToServer.writeBytes(sentence + '\n');

modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close();

    }
}
```

Envia linha para servidor

Lê linha Vinda do servidor

Exemplo: servidor Java (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

       Socket connectionSocket = welcomeSocket.accept();

      BufferedReader inFromClient =
        new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
```

cria Socket de entrada no port 6789

espera, no socket de entrada o contacto do cliente

Cria stream de entrada, ligado ao socket

Exemplo: servidor Java (TCP), cont

Cria stream de saida ligado ao socket

```
DataOutputStream  outToClient =
  new DataOutputStream(connectionSocket.getOutputStream());
```

Lê linha do socket

```
clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';
```

Escreve linha para socket

```
outToClient.writeBytes(capitalizedSentence);
            }
        }
    }
```

Fim do ciclo while, volta atrás e espera por Ligação de outro cliente

Nível de Aplicação

- 2.1 Aplicações de Rede
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Correio Electrónico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Aplicações P2P
- 2.7 Programação de Socket com TCP
- 2.8 Programação de Socket com UDP

Programação com sockets com UDP

UDP: sem "ligação" entre cliente e servidor

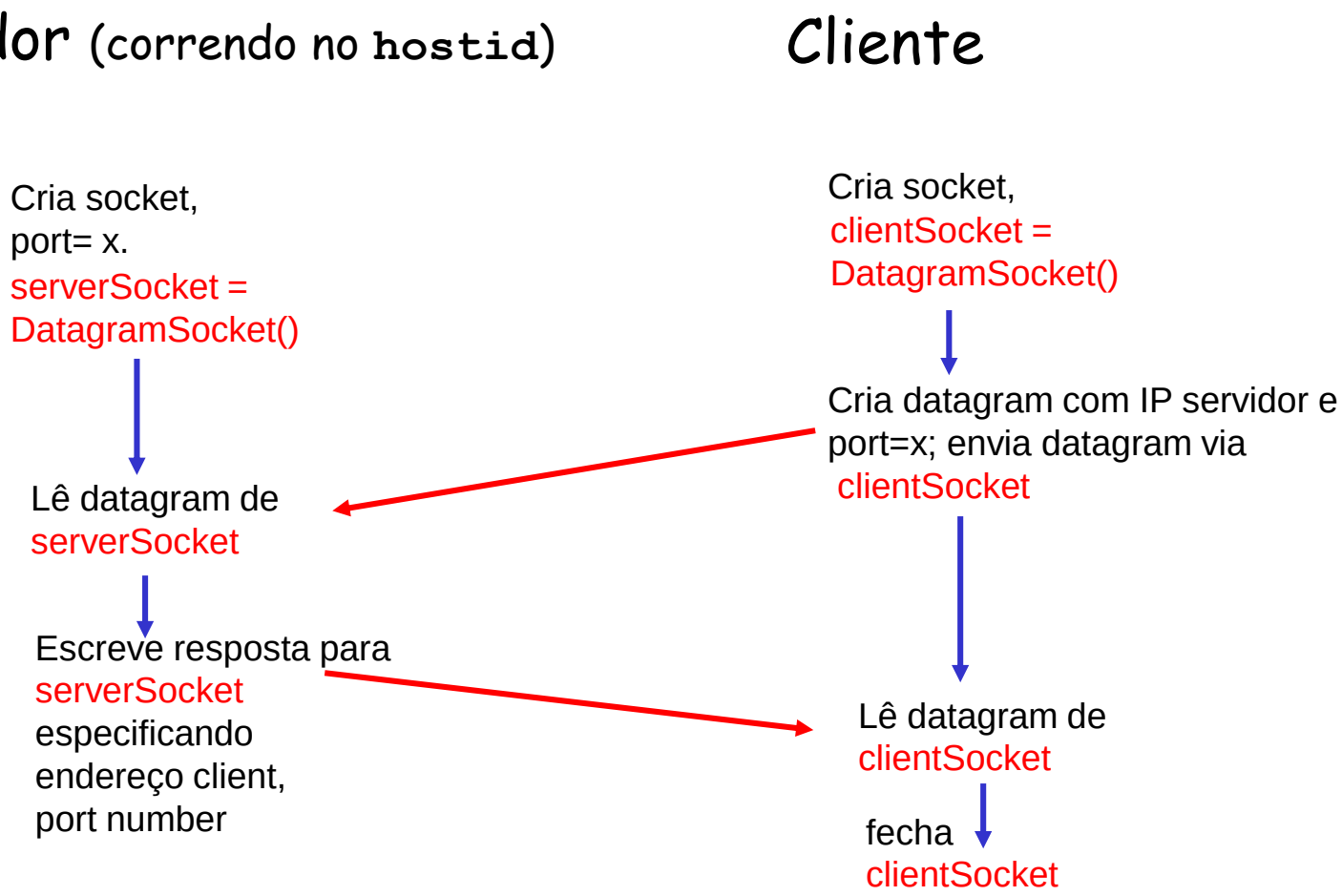
- Sem handshaking
- Emissor explicitamente coloca endereço IP e porto em cada pacote
- Servidor tem de extrair do pacote recebido o endereço IP e porto emissor

UDP: dados transmitidos podem ser recebidos for a de ordem ou perdidos

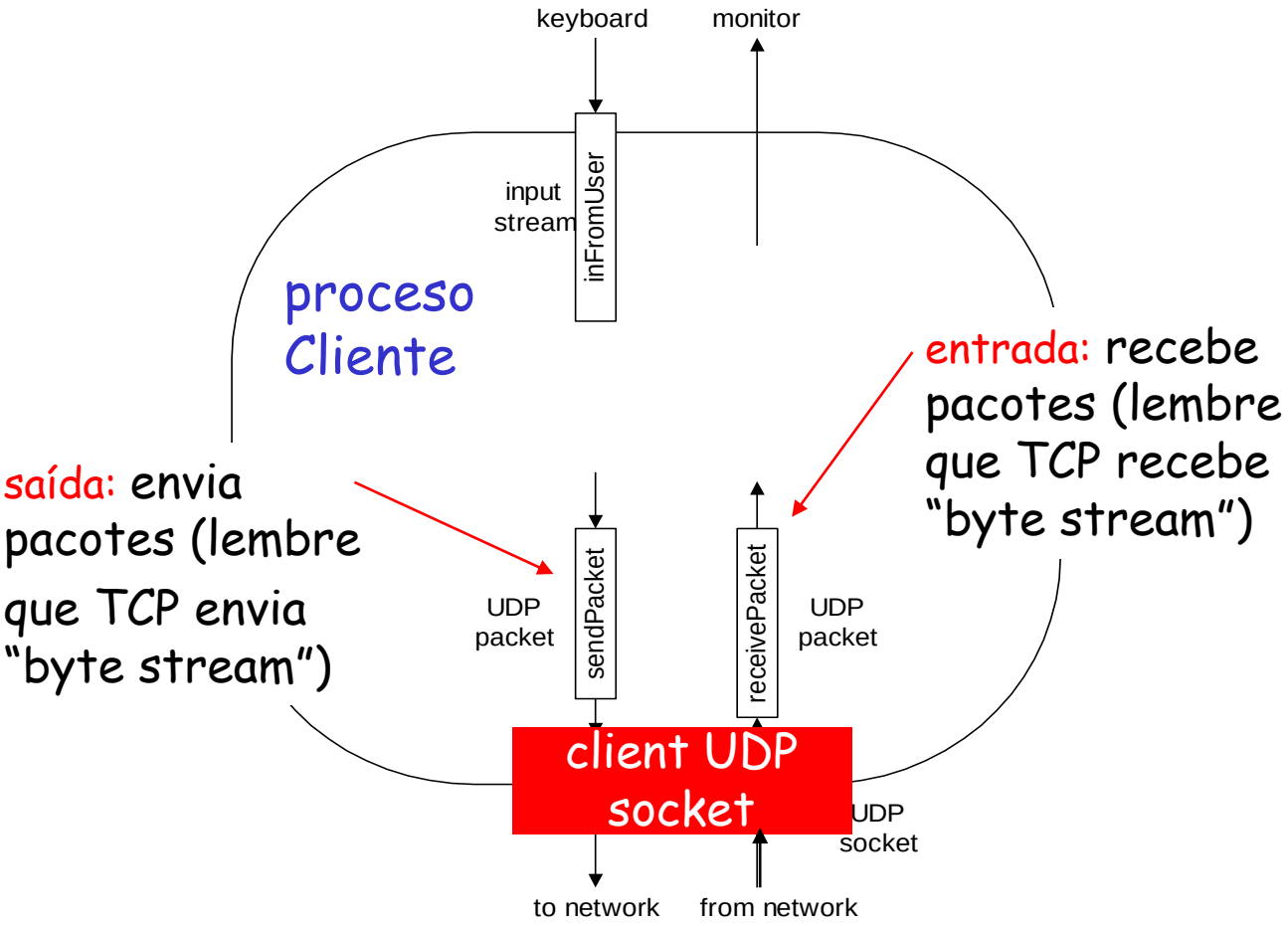
ponto de vista da aplicação

UDP oferece transferência não fiável de grupo de bytes ("datagrams") entre cliente e servidor

Interacção cliente/servidor com socket: UDP



Exemplo: cliente Java (UDP)



Exemplo: cliente Java (UDP)

```
import java.io.*;
import java.net.*;

class UDPClient {
   public static void main(String args[]) throws Exception
   {

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("hostname");

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      String sentence = inFromUser.readLine();

      sendData = sentence.getBytes();
```

cria Stream entrada

cria Socket cliente

traduz hostname para IP address usando DNS

Exemplo: cliente Java (UDP), cont.

Cria datagram com data-to-send, length, IP addr, port

Envia datagram Para server

Lê datagram Do server

```
                DatagramPacket sendPacket =
                  new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

                clientSocket.send(sendPacket);

                DatagramPacket receivePacket =
                   new DatagramPacket(receiveData, receiveData.length);

                clientSocket.receive(receivePacket);

                String modifiedSentence =
                    new String(receivePacket.getData());

                System.out.println("FROM SERVER:" + modifiedSentence);
                clientSocket.close();
                }

            }
```

Exemplo: servidor Java (UDP)

```
import java.io.*;
import java.net.*;

class UDPServer {
  public static void main(String args[]) throws Exception
   {

      DatagramSocket serverSocket = new DatagramSocket(9876);

      byte[] receiveData = new byte[1024];
      byte[] sendData  = new byte[1024];

      while(true)
       {

        DatagramPacket receivePacket =
           new DatagramPacket(receiveData, receiveData.length);

        serverSocket.receive(receivePacket);
```

cria Socket datagram no port 9876

Cria espaço para datagram recebidos

Datagram recebido

Exemplo: servidor Java (UDP), cont

```
                String sentence = new String(receivePacket.getData());

Lê endreço IP
port #, do
emissor         InetAddress IPAddress = receivePacket.getAddress();

                int port = receivePacket.getPort();

                        String capitalizedSentence = sentence.toUpperCase();

                sendData = capitalizedSentence.getBytes();

Cria datagram
para enviar ao
cliente         DatagramPacket sendPacket =
                   new DatagramPacket(sendData, sendData.length, IPAddress,
                                      port);

escreve
datagram
no socket       serverSocket.send(sendPacket);
              }
            }

          }
```

Fim do ciclo while,
Volta atrás e espera
por outro datagram

Sumário

Mais importante: aprenderam-se protocolos

- Tipicamente transferência de mensagens de pedido/resposta:
  - Cliente pede informação ou serviço
  - Servidor responde com dados, código de estados
- Formato das mensagens:
  - Cabeçalho (header): campos que fornecem informação sobre os dados
  - Dados: informação a ser comunicada

Temas Importantes :

- controlo vs. dados
  - Em banda (in-band), fora de banda (out-of-band)
- centralizado vs. decentralizado
- Sem estado vs. com estado (stateless vs. stateful)
- Transferência fiável vs não fiável de mensagens
- "complexidade na periferia da rede"
- Segurança: autenticação