Capítulo 2: Camada de Aplicação

Metas do capítulo:

- aspectos conceituais e de implementação de protocolos de aplicação em redes
  - paradigma cliente servidor
  - modelos de serviço
- aprenda sobre protocolos através do estudo de protocolos populares do nível da aplicação

Mais metas do capítulo

- protocolos específicos:
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- a programação de aplicações de rede
  - programação usando sockets

Aplicações de rede: algum jargão

- Um processo é um programa que executa num hospedeiro (host).
- 2 processos no mesmo hospedeiro se comunicam usando comunicação entre processos definida pelo sistema operacional (SO).
- 2 processos em hospedeiros distintos se comunicam usando um protocolo da camada de aplicação.

- Um agente de usuário (UA) é uma interface entre o usuário e a aplicação de rede.
  - WWW: browser
  - Correio: leitor/compositor de mensagens
  - streaming audio/video: tocador de mídia

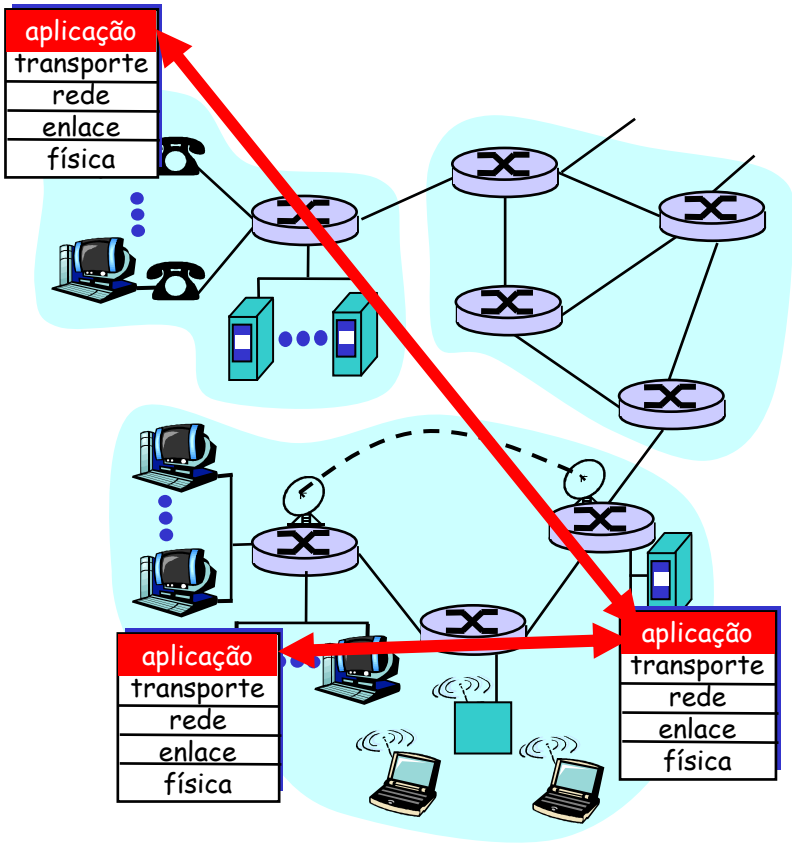
Aplicações e protocolos da camada de aplicação

Aplicação: processos distribuídos em comunicação

- executam em hospedeiros no "espaço de usuário"
- trocam mensagens para implementar a aplicação
- p.ex., correio, transf. de arquivo, WWW

Protocolos da camada de aplicação

- uma "parte" da aplicação
- define mensagens trocadas por apls e ações tomadas
- usam serviços providos por protocolos de camadas inferiores (TCP, UDP)



Camada de aplicação define:

- Tipo das mensagens trocadas: ex, mensagens de requisição & resposta
- Sintaxe das mensagens: quais os campos de uma mensagem & como estes são delineados;
- Semântica dos campos: qual o significado das informações nos campos;
- Regras: definem quando e como os processos enviam & respondem mensagens;

Protocolos de domínio público:

- Definidos por RFCs
- Garante interoperabilidade
- ex, HTTP, SMTP

Protocolos proprietários:

- ex, KaZaA

Paradigma cliente-servidor (C-S)

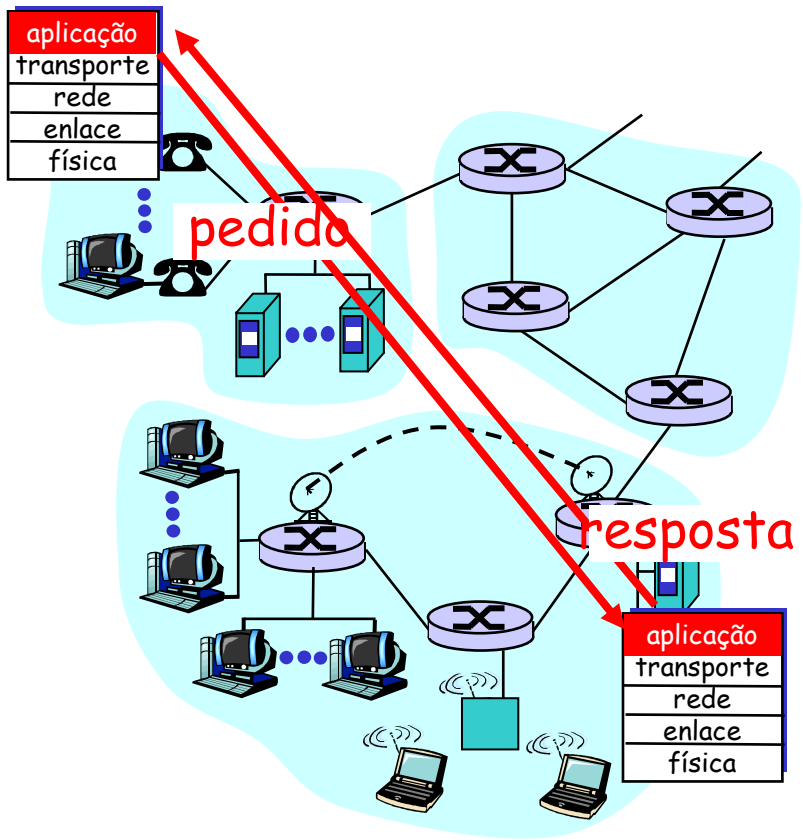
Apl. de rede típica tem duas partes: *cliente* e *servidor*

Cliente:

- inicia contato com o servidor ("fala primeiro")
- tipicamente solicita serviço do servidor
- para WWW, cliente implementado no browser; para correio no leitor de mensagens

Servidor:

- provê ao cliente o serviço requisitado
- p.ex., servidor WWW envia página solicitada; servidor de correio entrega mensagens



Arquiteturas de aplicação

- Cliente-servidor
- **Peer-to-peer** (P2P)
- Híbrida de cliente-servidor e P2P

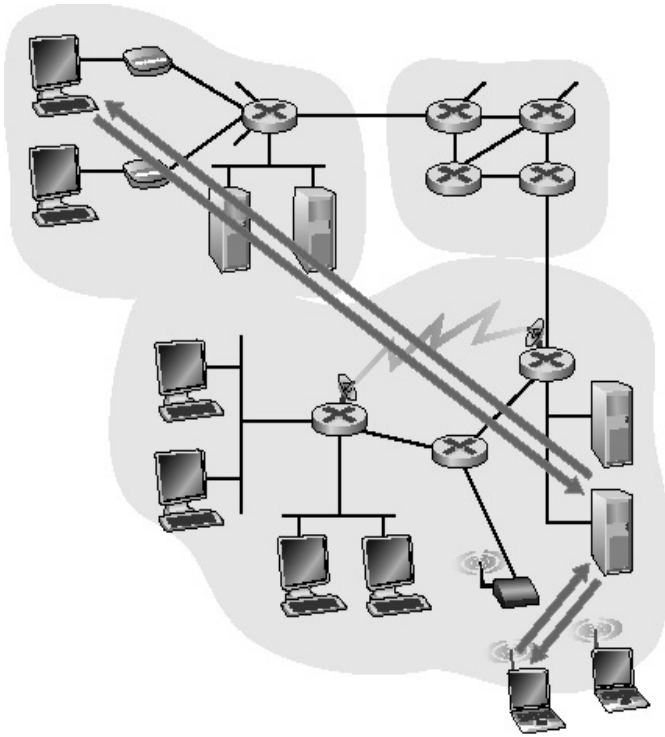
Arquitetura cliente-servidor

Servidor:

- Hospedeiro sempre ativo
- Endereço IP permanente
- Fornece serviços solicitados pelo cliente

Clientes:

- Comunicam-se com o servidor
- Pode ser conectado intermitentemente
- Pode ter endereço IP dinâmico
- Não se comunicam diretamente uns com os outros

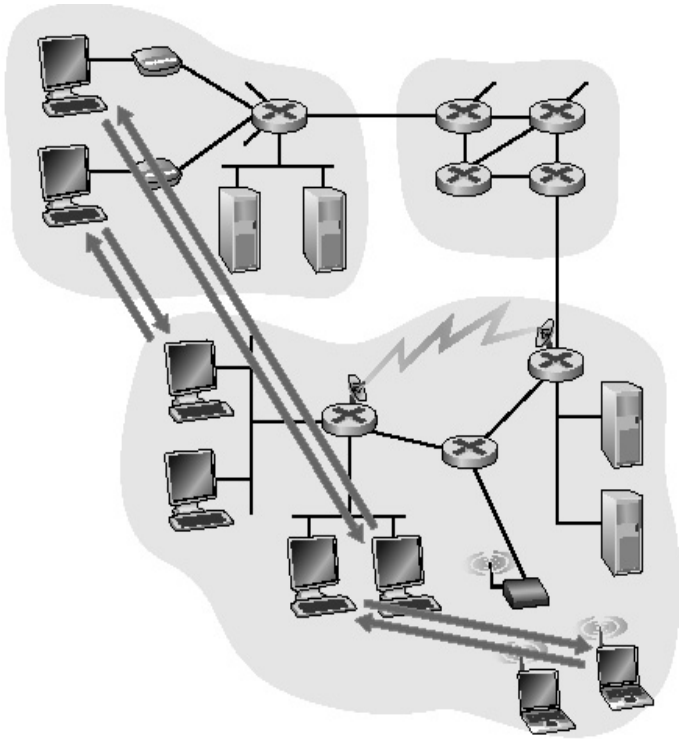


a. Aplicação cliente-servidor

Arquitetura P2P pura

- Nem sempre no servidor
- Sistemas finais arbitrários comunicam-se diretamente
- Pares são intermitentemente conectados e trocam endereços IP
- Ex.: Gnutella

Altamente escaláveis mas difíceis de gerenciar



b. Aplicação P2P

Híbrida de cliente-servidor e P2P

Napster

- Transferência de arquivo P2P
- Busca centralizada de arquivos:
  - Conteúdo de registro dos pares no servidor central
  - Consulta de pares no mesmo servidor central para localizar o conteúdo

Instant messaging

- Bate-papo entre dois usuários é P2P
- Detecção/localização centralizada de presença:
  - Usuário registra seu endereço IP com o servidor central quando fica on-line
  - Usuário contata o servidor central para encontrar endereços IP dos vizinhos

Comunicação de processos

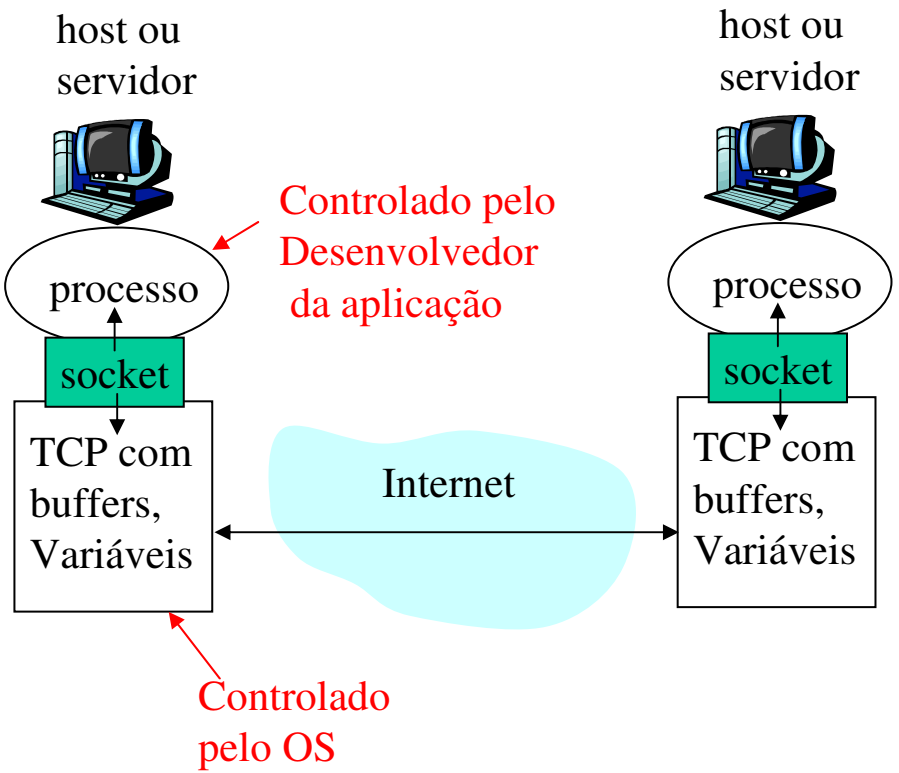
Processo: programa executando num hospedeiro

- Dentro do mesmo hospedeiro: dois processos se comunicam usando comunicação interprocesso (definido pelo OS)
- Processos em diferentes hospedeiros se comunicam por meio de troca de mensagens
- Processo cliente: processo que inicia a comunicação
- Processo servidor: processo que espera para ser contatado

Nota: aplicações com arquiteturas P2P possuem processos cliente e processos servidor

Comunicação entre processos na rede

- processos se comunicam enviando ou recebendo mensagens através de um socket;
- socket
  - O processo emissor joga a mensagem por seu socket;
  - O processo emissor assume que há uma infra-estrutura de transporte no lado oposto do socket que irá transmitir a mensagem até o socket do processor receptor;



- API: (1) escolhe do protocolo de transporte; (2) abilidade para fixar alguns parâmetros (voltamos mais tarde a este assunto)

Identificando processos:

- Para que um processo possa receber mensagens, ele precisa ter um identificador;
- Cada host tem um endereço único de 32 bits – endereço IP;
- Q: O endereço IP de um host no qual um processo está executando é suficiente para identificar este processo?
- Resposta: Não, muitos processos podem estar em execução em um mesmo host

- O identificador inclue tanto o endereço IP como também o número de porta associado com o processo no host;
- Exemplo de número de portas:
  - Servidor HTTP: 80
  - Servidor de Correio: 25
- Voltaremos a este assunto mais tarde

De que serviço de transporte uma aplicação precisa?

Perda de dados

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

Temporização

- algumas apls (p.ex., telefonia Internet, jogos interativos) requerem baixo retardo para serem "viáveis"

Largura de banda

- algumas apls (p.ex., multimídia) requerem quantia mínima de banda para serem "viáveis"
- outras apls ("apls elásticas") conseguem usar qualquer quantia de banda disponível

Requisitos do serviço de transporte de apls comuns

Aplicação	Perdas	Banda	Sensibilidade temporal
transferência de arqs	sem perdas	elástica	não
correio	sem perdas	elástica	não
documentos WWW	sem perdas	elástica	não
áudio/vídeo de tempo real	tolerante	áudio: 5Kb-1Mb vídeo:10Kb-5Mb	sim, 100's mseg
áudio/vídeo gravado	tolerante	como anterior	sim, alguns segs
jogos interativos	tolerante	> alguns Kbps	sim, 100's mseg
apls financeiras	sem perdas	elástica	sim e não

Serviços providos por protocolos de transporte Internet

serviço TCP:

- *orientado a conexão:* negociação e definição da conexão (setup) requerida entre cliente, servidor
- *transporte confiável* entre processos remetente e receptor
- *controle de fluxo:* remetente não vai sobrecarregar o receptor
- *controle de congestionamento:* estrangular remetente quando a rede está sobrecarregada
- *não provê:* garantias temporais ou de banda mínima

serviço UDP:

- transferência de dados não confiável entre processos remetente e receptor
- não provê: setup da conexão, confiabilidade, controle de fluxo, controle de congestionamento, garantias temporais ou de banda mínima

P: Qual é o interesse em ter um UDP?

Apls Internet: seus protocolos e seus protocolos de transporte

Aplicação	Protocolo da camada de apl	Protocolo de transporte usado
correio eletrônico	smtp [RFC 821]	TCP
accesso terminal remoto	telnet [RFC 854]	TCP
WWW	http [RFC 2068]	TCP
transferência de arquivos	ftp [RFC 959]	TCP
streaming multimídia	proprietário (p.ex. RealNetworks)	TCP ou UDP
servidor de arquivo remoto	NSF	TCP ou UDP
telefonia Internet	proprietário (p.ex., Vocaltec)	tipicamente UDP

WWW e HTTP: algum jargão

- Página WWW:
  - consiste de "objetos"
  - endereçada por uma URL
- Quase todas as páginas WWW consistem de:
  - página base HTML, e
  - vários objetos referenciados.
- URL tem duas partes: nome de hospedeiro, e nome de caminho:

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

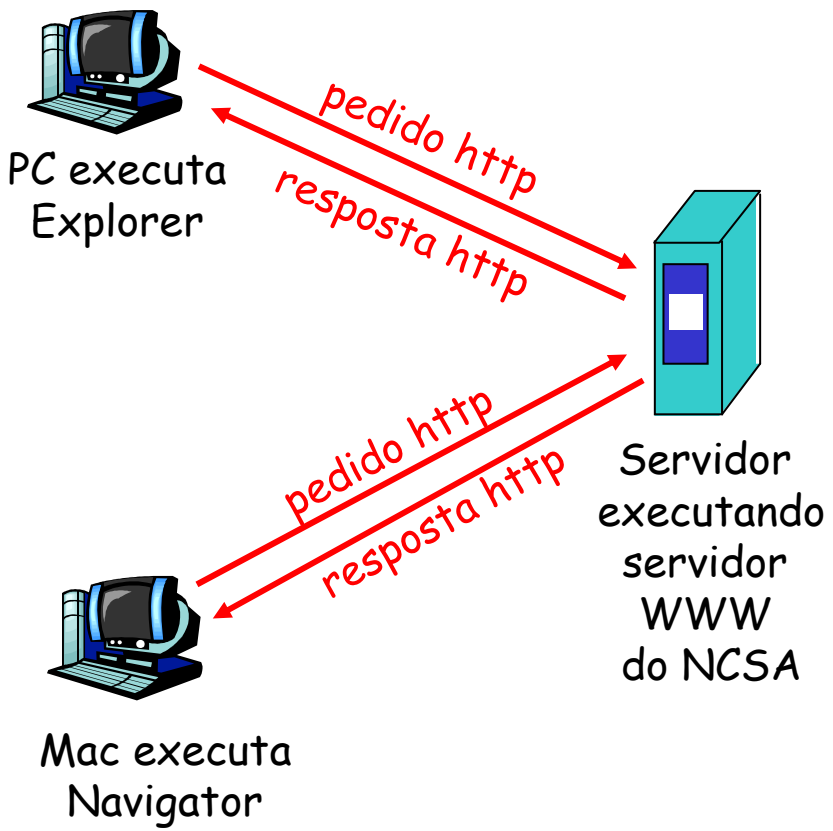
nome do host nome do caminho

- Agente de usuário para WWW se chama de browser:
  - MS Internet Explorer
  - Netscape Communicator
- Servidor para WWW se chama "servidor WWW":
  - Apache (domínio público)
  - MS Internet Information Server (IIS)

Protocolo HTTP: visão geral

HTTP: hypertext transfer protocol

- protocolo da camada de aplicação para WWW
- modelo cliente/servidor
  - *cliente:* browser que pede, recebe, "visualiza" objetos WWW
  - *servidor:* servidor WWW envia objetos em resposta a pedidos
- http1.0: RFC 1945
- http1.1: RFC 2068



Mais sobre o protocolo HTTP

HTTP: serviço de transporte TCP:

- cliente inicia conexão TCP (cria socket) ao servidor, porta 80
- servidor aceita conexão TCP do cliente
- mensagens HTTP (mensagens do protocolo da camada de apl) trocadas entre browser (cliente HTTP) e servidor e WWW (servidor HTTP)
- encerra conexão TCP

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 parem de executar, suas visões do "estado" podem ser inconsistentes, devendo então ser reconciliadas

Conexões HTTP

HTTP: não persistente

- No máximo um objeto é enviado em uma conexão TCP;
- HTTP/1.0 usa conexões não persistentes

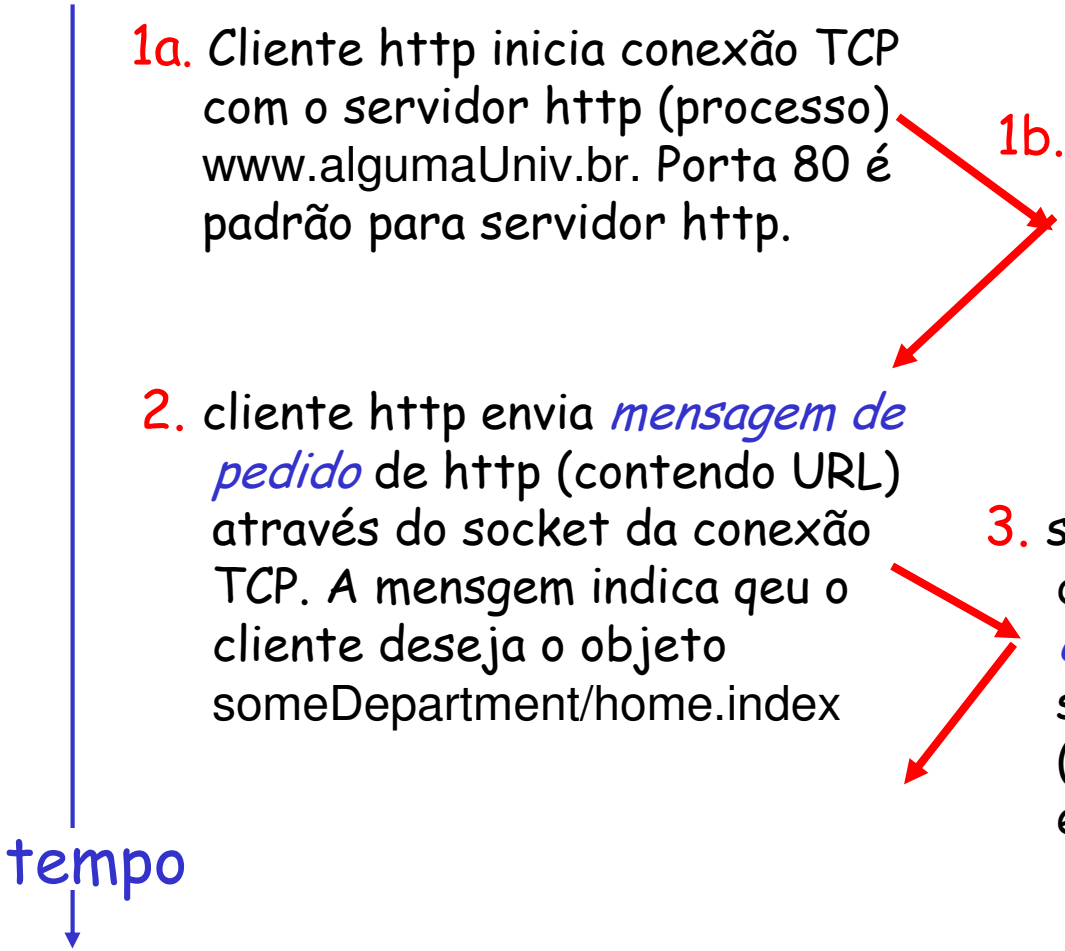
HTTP: persistente

- Múltiplos objetos podem ser enviados numa única conexão TCP entre o servidor e o cliente;
- HTTP/1.1 usa conexões persistentes no modo default;

Ex: HTTP não-persistente

Supomos que usuário digita a URL
www.algumaUniv.br/algumDepartmento/inicial.index

(contém texto, referências a 10 imagens jpeg)



Ex: HTTP não-persistente (cont.)

4. servidor http encerra conexão TCP .

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

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

tempo

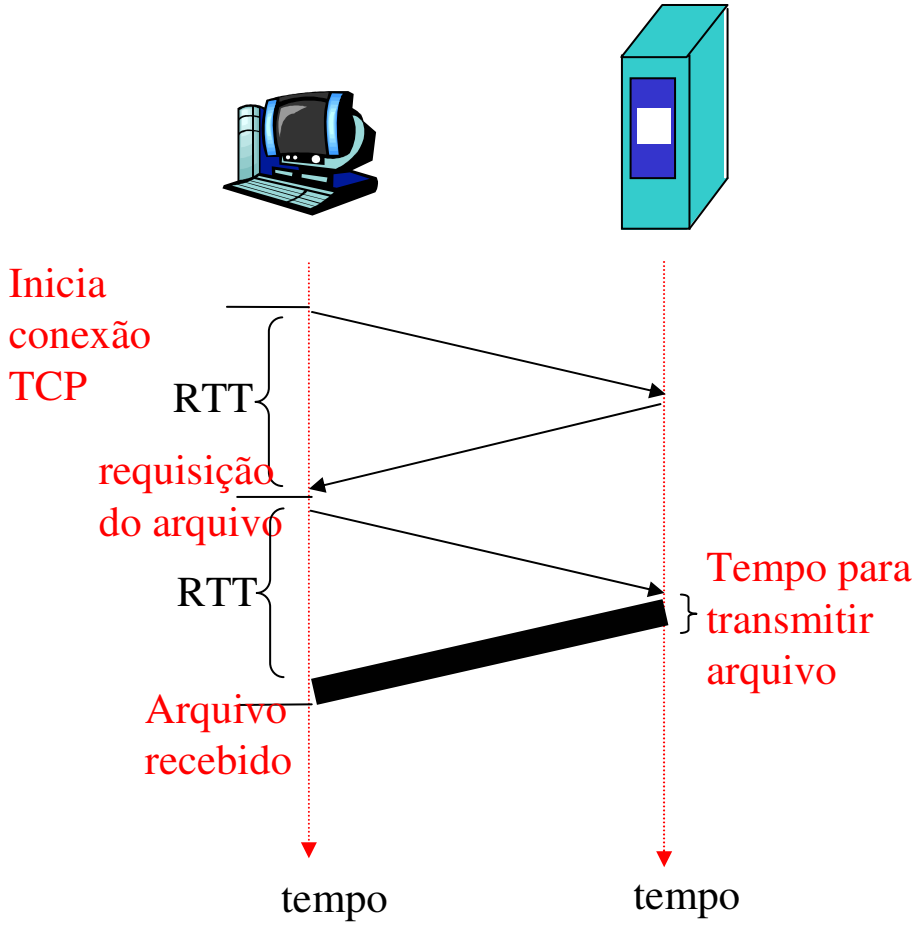
Tempo de Resposta

Definição de RTT: tempo para enviar um pequeno pacote para viajar do cliente para o servidor e retornar;

Tempo de resposta:

- um RTT para iniciar a conexão TCP
- um RTT para a requisição HTTP e para que alguns bytes da resposta HTTP sejam recebidos
- tempo de transmissão do arquivo

total = 2RTT+tempo de transmissão



HTTP persistente

HTTP não-persistente:

- servidor analisa pedido, responde, e encerra conexão TCP
- requer 2 RTTs para trazer cada objeto
- mas os browsers geralmente abrem conexões TCP paralelas para trazer cada objeto

HTTP- persistente

- servidor mantém conexão aberta depois de enviar a resposta;
- mensagens HTTP subsequentes entre o o mesmos cliente/servidor são enviadas por esta conexão;
- na mesma conexão TCP: servidor analisa pedido, responde, analisa novo pedido e assim por diante

Persistente sem pipelining:

- Cliente só faz nova requisição quando a resposta de uma requisição anterior foi recebida;
- um RTT para cada objeto

Persistente com pipelining:

- default in HTTP/1.1
- O cliente envia a requisição assim que encontra um objeto;
- Um pouco mais de um RTT para trazer todos os objetos

HTTP persistente

Características do HTTP persistente:
- Requer 2 RTTs por objeto
- OS deve manipular e alocar recursos do hospedeiro para cada conexão TCP
  Mas os browsers freqüentemente abrem conexões TCP paralelas para buscar objetos referenciados

HTTP persistente
- Servidor deixa a conexão aberta após enviar uma resposta
- Mensagens HTTP subseqüentes entre o mesmo cliente/servidor são enviadas pela conexão

Persistente sem **pipelining**:
- O cliente emite novas requisições apenas quando a resposta anterior for recebida
- Um RTT para cada objeto referenciado

Persistente com **pipelining**:
- Padrão no HTTP/1.1
- O cliente envia requisições assim que encontra um objeto referenciado
- Tão pequeno como um RTT para todos os objetos referenciados

Formato de mensagem HTTP: pedido

- Dois tipos de mensagem HTTP: *pedido, resposta*
- mensagem de pedido HTTP:
  - ASCII (formato legível por pessoas)

linha do pedido (comandos GET, POST, HEAD)

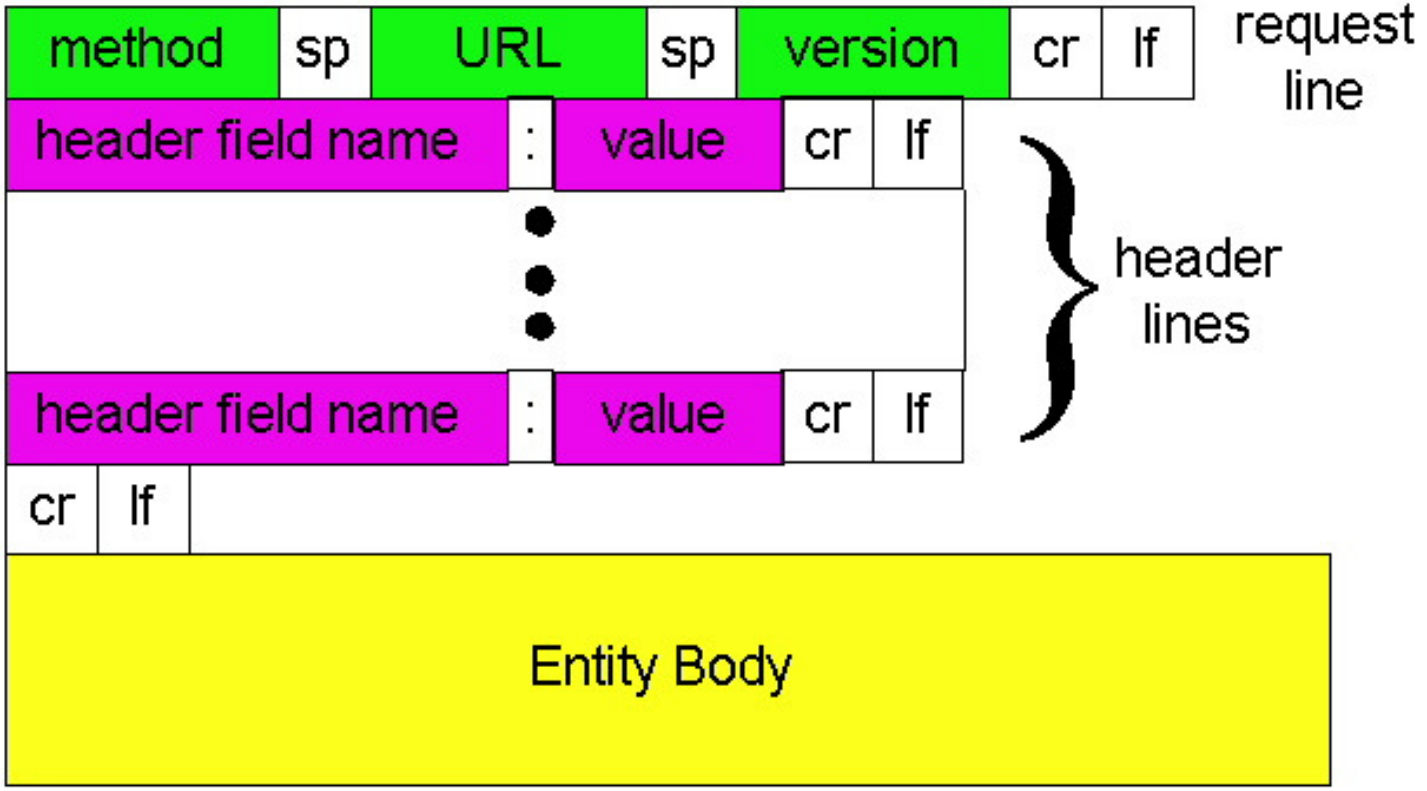
linhas do cabeçalho

```
GET /somedir/page.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr
```

(carriage return (CR), line feed(LF) adicionais)

Carriage return, line feed indica fim de mensagem

Mensagem de pedido HTTP: formato geral



Tipos de Requisição

Método Post:

- A página Web geralmente inclue um formulário para entrada de dados;
- A requisição é enviada para o servidor no corpo da entidade;

Método URL:

- Usa método GET
- A requisição é enviada para o servidor no campo URL da linha de requisição;

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

Tipos de Métodos

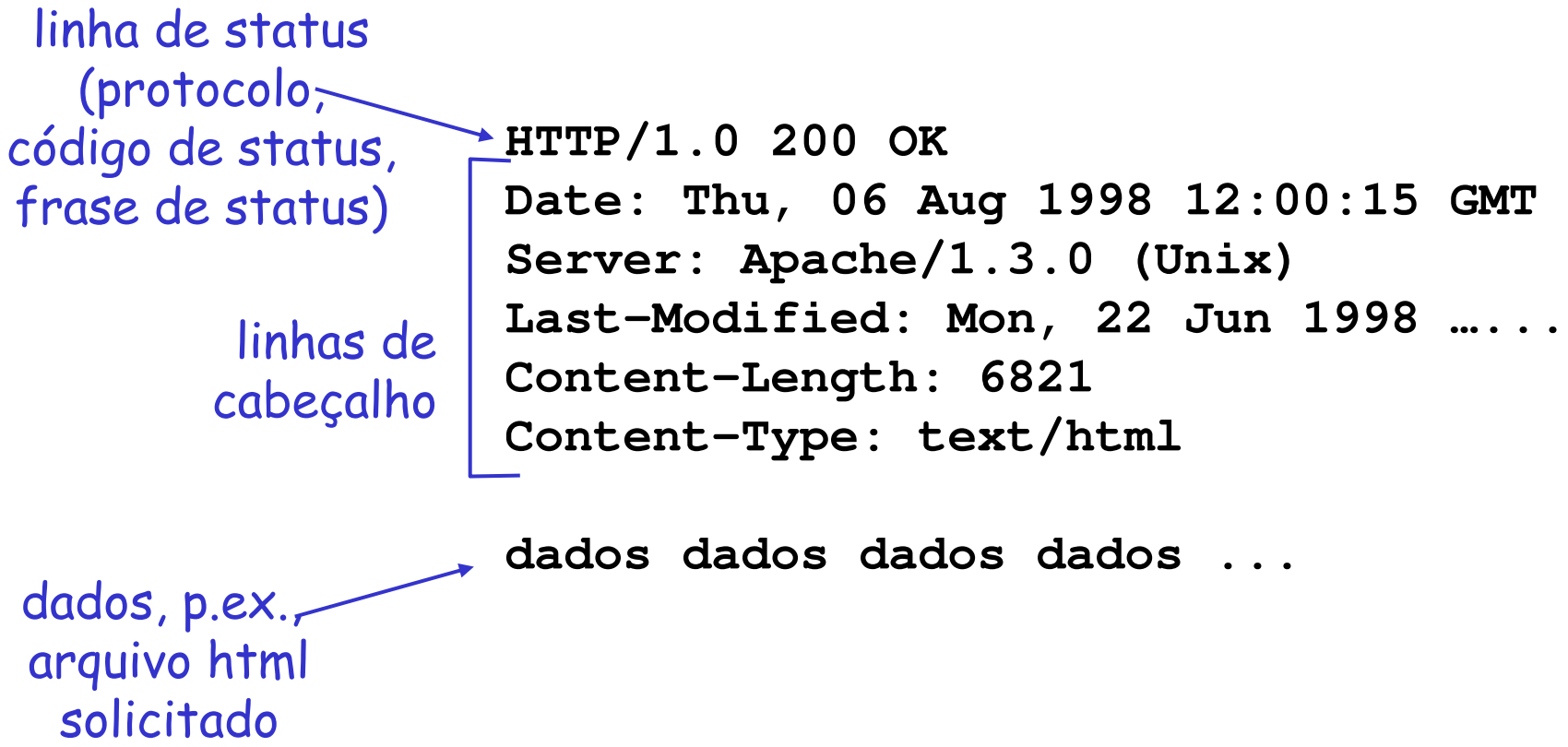
HTTP/1.0

- GET
- POST
- HEAD
  - Pede ao servidor que deixe de fora da resposta o objeto solicitado; geralmente é usado para depuração;

HTTP/1.1

- GET, POST, HEAD
- PUT
- DELETE
  - Remove o arquivo especificado no campo URL;

Formato de mensagem HTTP: resposta



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 WWW 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 enviado pelo servidor http !

HTML (HyperText Markup Language)

- HTML: uma linguagem simples para hipertexto
  - começou como versão simples de SGML
  - construção básica: cadéias de texto anotadas
- Construtores de formato operam sobre cadéias
  - .. *bold* (negrito)
  - <H1 ALIGN=CENTER> ..título centrado .. </H1>
  - <BODY bgcolor=white text=black link=red ..> .. </BODY>
- vários formatos
  - listas de *bullets*, listas ordenadas, listas de definição
  - tabelas
  - *frames*

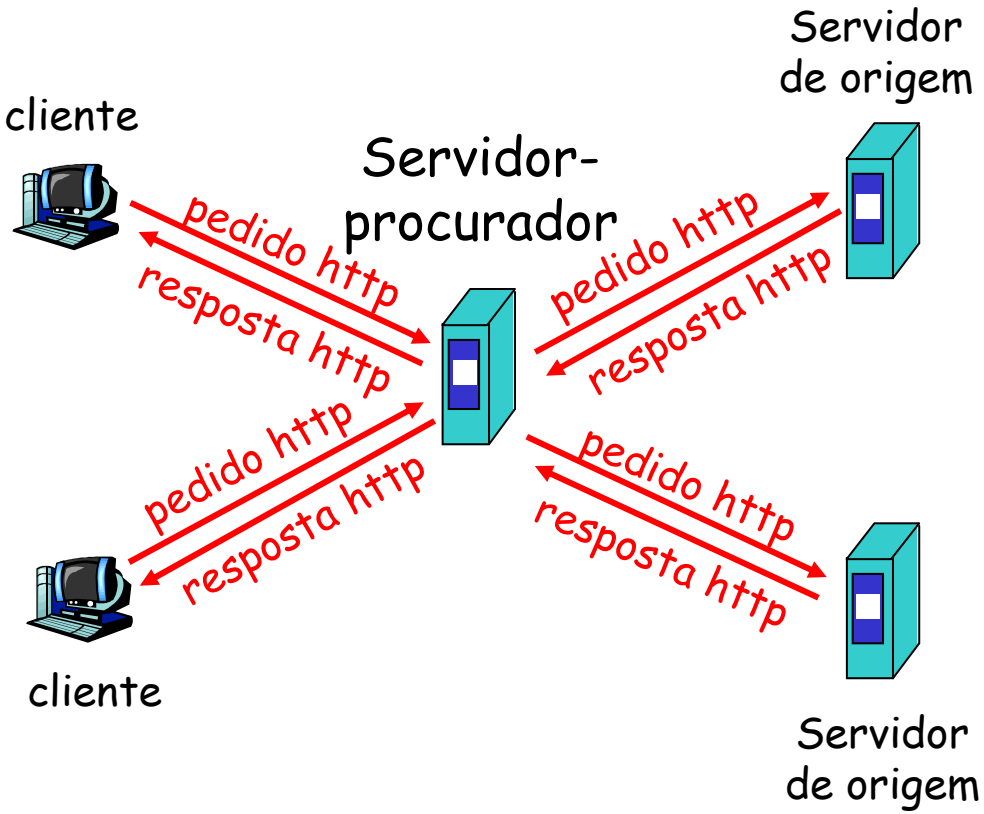
Encadeamento de referências

- Referências <A HREF=LinkRef> ... </A>
  - a componentes do documento local
    <A HREF="importante"> clique para uma dica </A>
  - a documentos no servidor local
    <A HREF="../index.htm"> voltar ao sumário </A>
  - a documentos em outros servidores
    <A HREF="http://www.uff.br"> saiba sobre a UFF </A>
- Multimídia
  - imagem embutida: <IMG SRC="eclipse">
  - imagem externa: <A HREF="eclipse.gif"> imagem maior </A>
  - vídeo Mpeg <A HREF="ByeByeBrasil.mpg"> um bom filme </A>
  - som <A HREF="http://www.sons.br/aniv.au"> feliz niver </A>

Cache WWW (servidor-procurador)

Meta: atender pedido do cliente sem envolver servidor de origem

- usuário configura browser: acessos WWW via procurador
- cliente envia todos pedidos http ao procurador
  - se objeto no cache do procurador, este o devolve imediatamente na resposta http
  - senão, solicita objeto do servidor de origem, depois devolve resposta http ao cliente



Mais sobre Web cache

- Cache atua tanto como cliente como servidor;
- Cache pode fazer ferificação no cabeçalho HTTP usando o campo `If-modified-since` :
  - Questão: a cache deve correr o risco e enviar objetos solicitados sem verificação?
  - São usadas heurísticas;
- Tipicamente os caches web são instalados em ISPs (universidades, companhias, ISP residencial)

Por quê usar cache WWW?

- tempo de resposta menor: cache "mais próximo" do cliente
- diminui tráfego aos servidores distantes
  - muitas vezes é um gargalo o enlace que liga a rede da instituição ou do provedor à Internet

Exemplo de Cache (1)

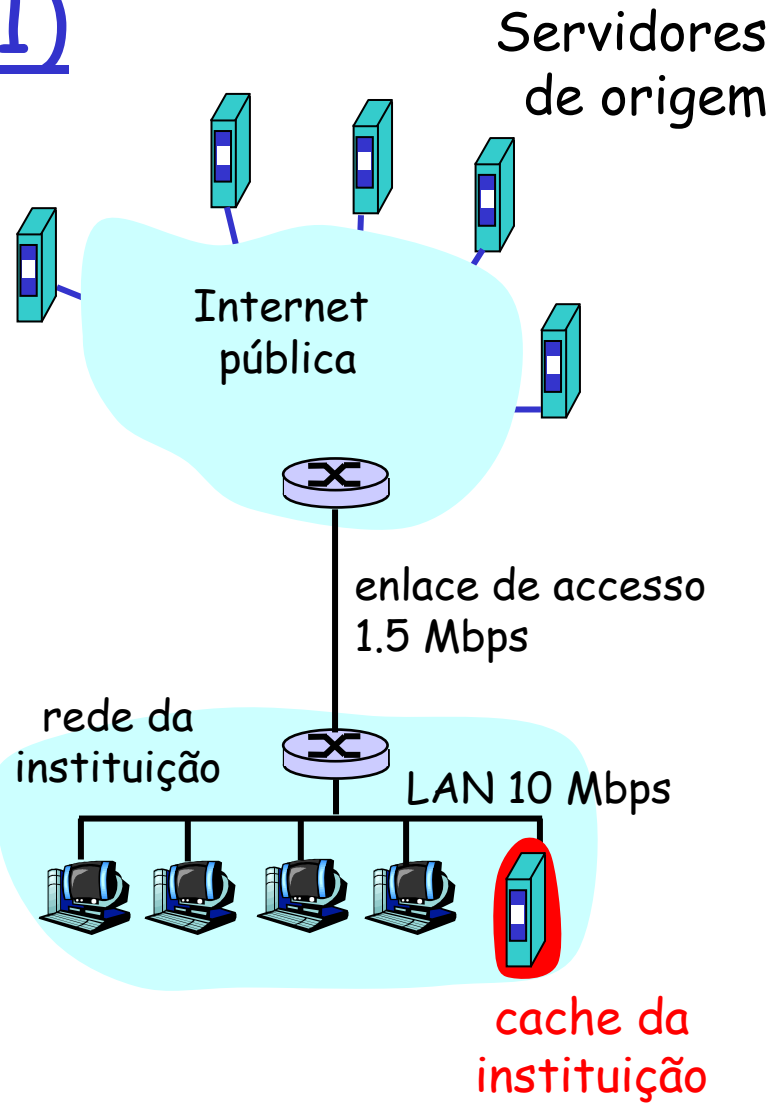
Assumptions

- Tamanho médio do objeto = 100,000 bits
- Taxa média de requisição do browser da instituição para os servidores de origem = 15/seg
- Atraso do roteador da instituição para qualquer servidor de origem e de volta para o roteador = 2 seg

Conseqüências

- Utilização da LAN = 15%
- Utilização do enlace de acesso = 100%
- Atraso total = atraso Internet + atraso de acesso + atraso LAN

  = 2 seg + minutos + milisegundos



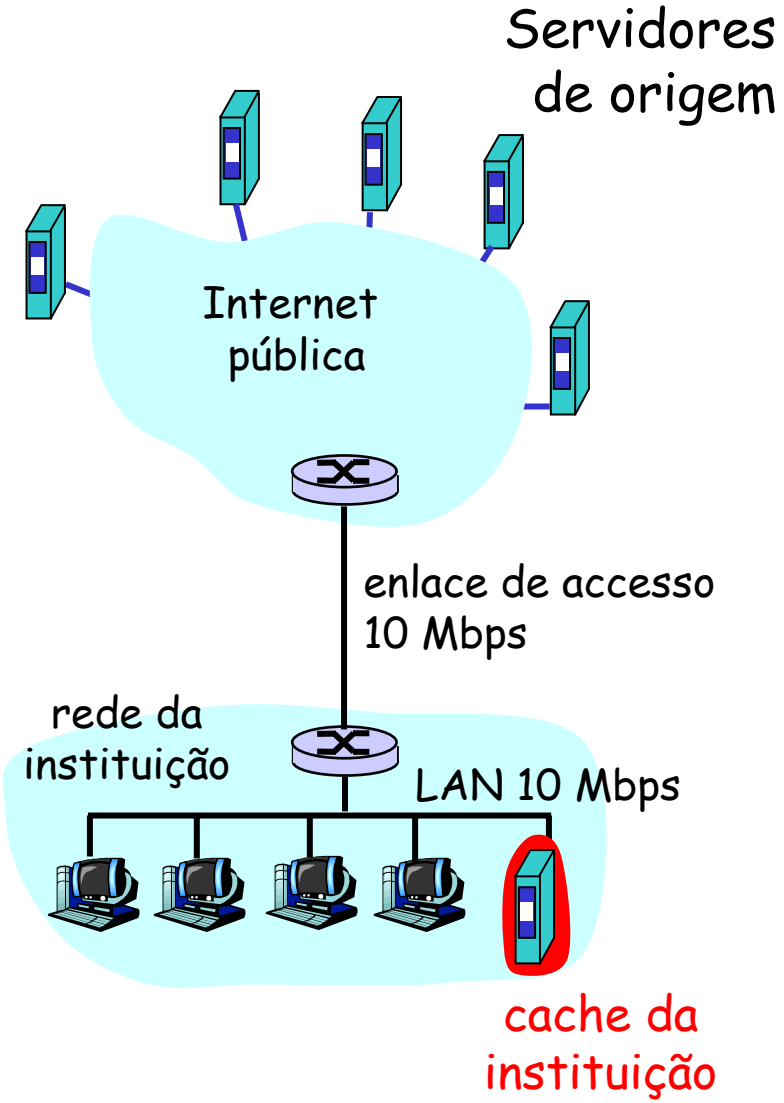
Exemplo cache (2)

Solução possível

- Aumentar a banda do enlace de acesso para 10 Mbps

Conseqüências

- utilização LAN = 15%
- Utilização do enlace de acesso = 15%
- Atraso total = atraso Internet + atraso de acesso + atraso LAN = 2 sec + msecs + msecs
- Geralmente um upgrade caro



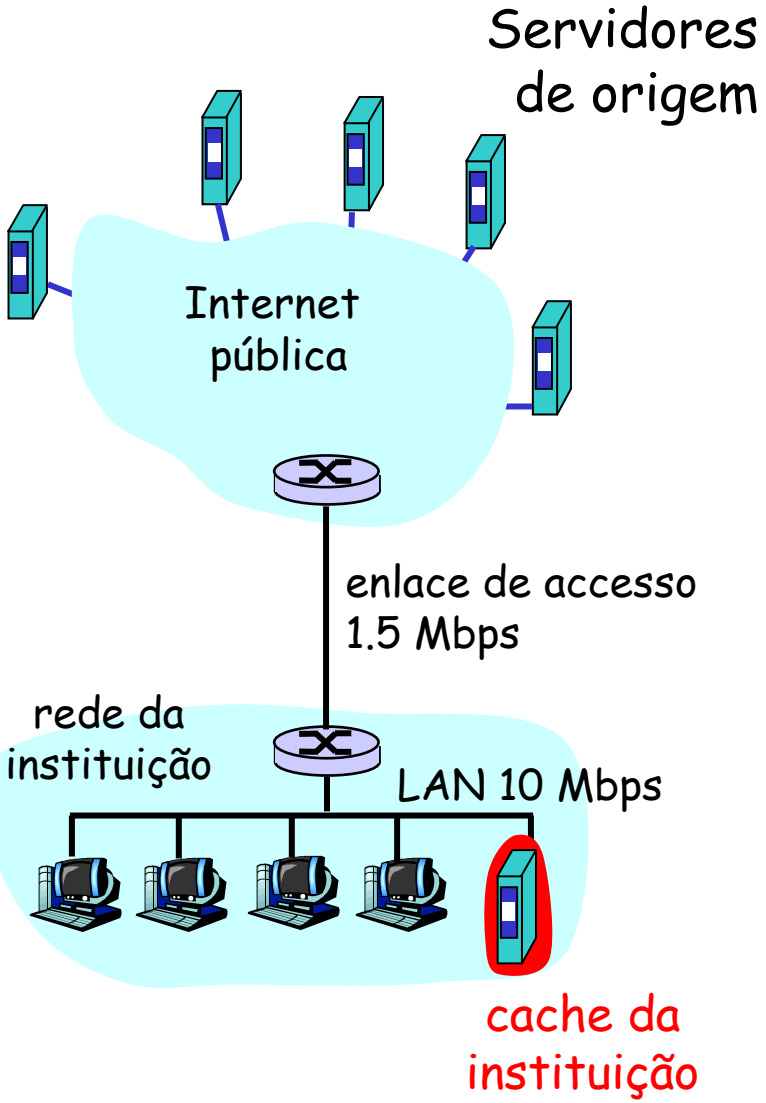
Exemplo cache(3)

Instala cache

- Suponha que a taxa de hits é .4

Conseqüência

- 40% das requisições são satisfeitas quase que imediatamente;
- 60% das requisições são satisfeitas pelo servidor;
- Utilização do enlace de acesso deduzido para 60%, resultando resulting em atrasos desprezíveis (digamos 10 mseg)
- Atraso total = atraso Internet + atraso de acesso + atraso = .6*2 sec + .6*.01 seg + millisegundos < 1.3 sg



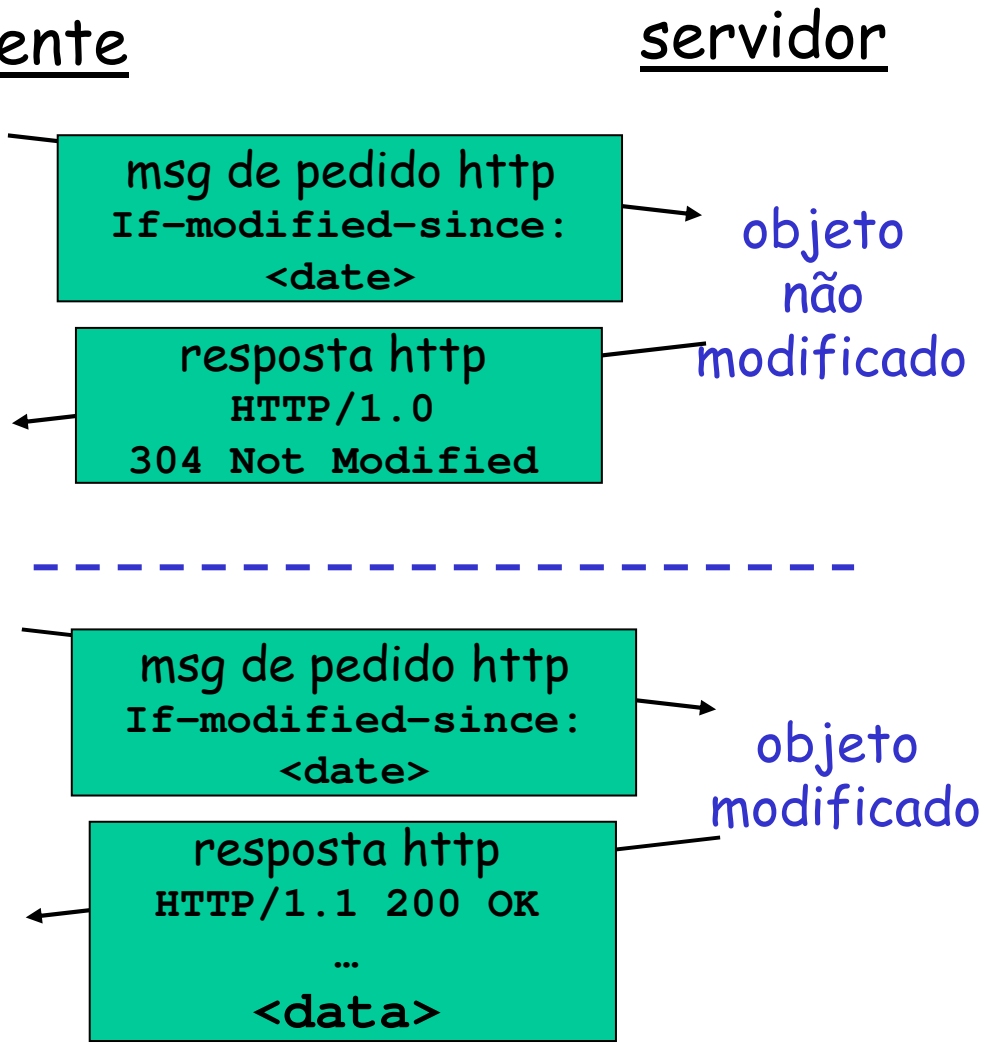
Interação usuário-servidor: GET condicional

- Meta: não enviar objeto se cliente já tem (no cache) versão atual
- cliente: especifica data da cópia no cache no pedido http

```
If-modified-since:
   <date>
```

- servidor: resposta não contém objeto se cópia no cache é atual:

```
HTTP/1.0 304 Not
   Modified
```



Formulários e interação bidirecional

- Formulários transmitem informação do cliente ao servidor
- HTTP permite enviar formulários ao servidor
- Resposta enviada como página HTML ***dinâmica***

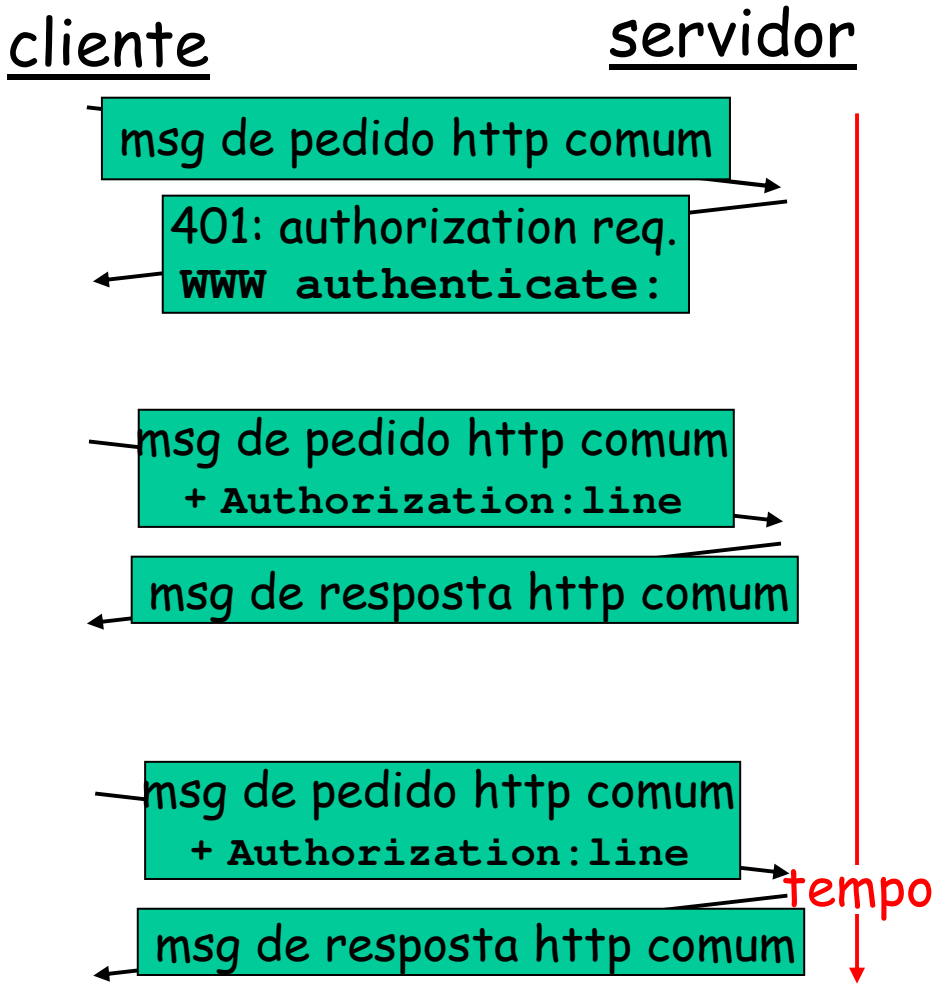
- Formulários processados usando *scripts* CGI (programas que executam no servidor WWW)
  - CGI - Common Gateway Interface
  - *scripts* CGI escondem acesso a diferentes serviços
  - servidor WWW atua como *gateway* universal

Interação usuário-servidor: autenticação

Meta da autenticação: controle de acesso aos documentos do servidor

- sem estado: cliente deve apresentar autorização com cada pedido
- autorização: tipicamente nome, senha
  - `authorization`: linha de cabeçalho no pedido
  - se não for apresentada autorização, servidor nega acesso, e coloca no cabeçalho da resposta

    `WWW authenticate:`



Browser guarda nome e senha para evitar que sejam pedidos ao usuário a cada acesso.

Interação usuário-servidor: cookies, mantendo o "estado"

Exemplo:

- Susan acessa a Internet sempre usando o mesmo PC;
- Ela visita um site de comércio eletrônico pela primeira vez;
- Quando a requisição HTTP inicial chega ao site, é criado um ID único e uma entrada no bando de dados para este ID;
- servidor envia "cookie" ao cliente na msg de resposta
- cliente apresenta cookie nos pedidos posteriores
- servidor casa cookie- apresentado com a info guardada no servidor

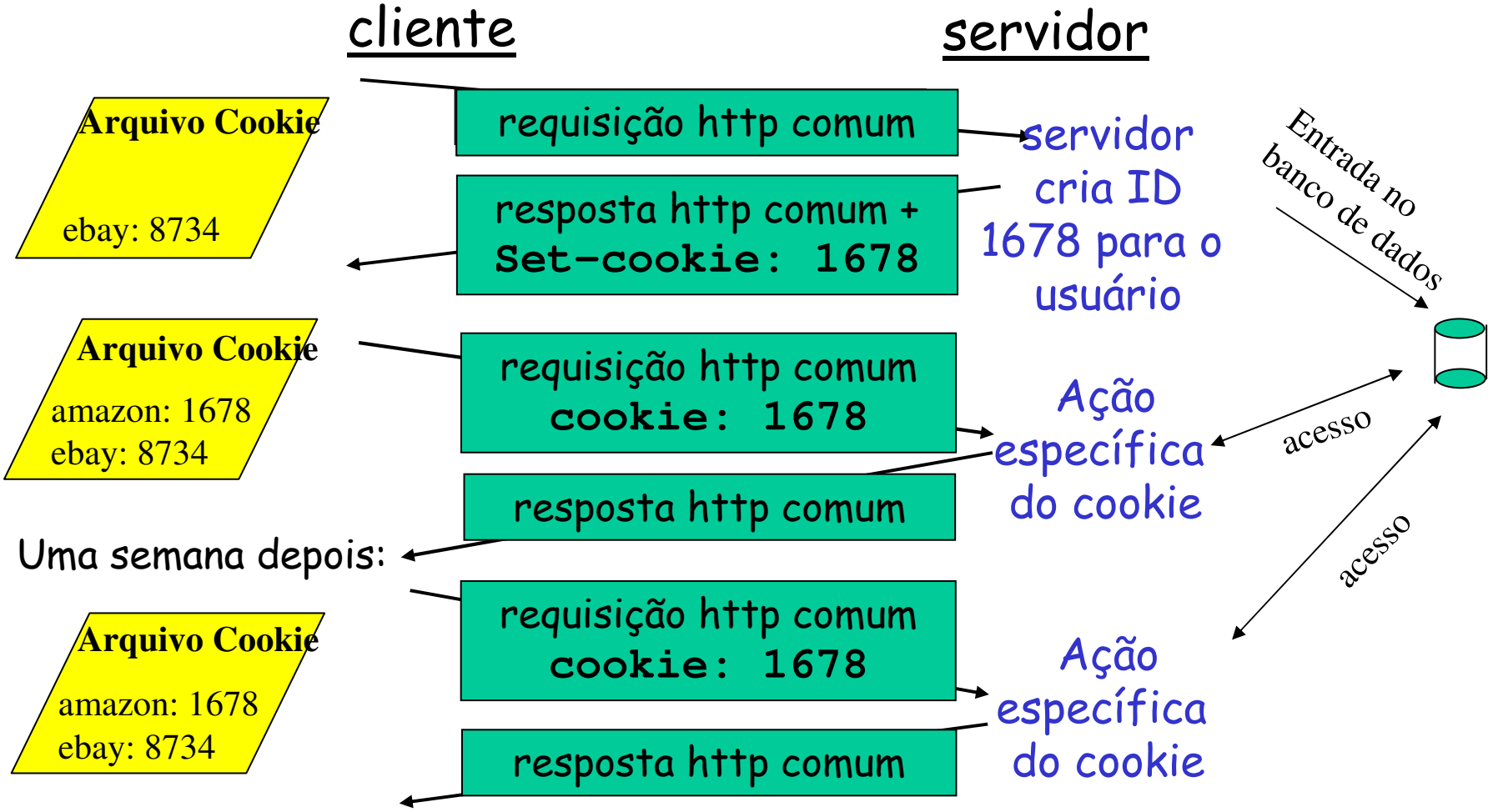
Interação usuário-servidor: cookies, mantendo o "estado"

A grande maioria dos sites Web usa cookies

Quatro componentes:

1) linha de cabeçalho do cookie na mensagem de resposta HTTP;
2) linha de cabeçalho do cookie na mensagem de requisição HTTP
3) Arquivo de cookie mantido na máquina do usuário e gerenciado por seu browser;
4) Banco de dados no site Web

Interação usuário-servidor: cookies, mantendo o "estado"



Interação usuário-servidor: cookies, mantendo o "estado"

O que cookie pode trazer?

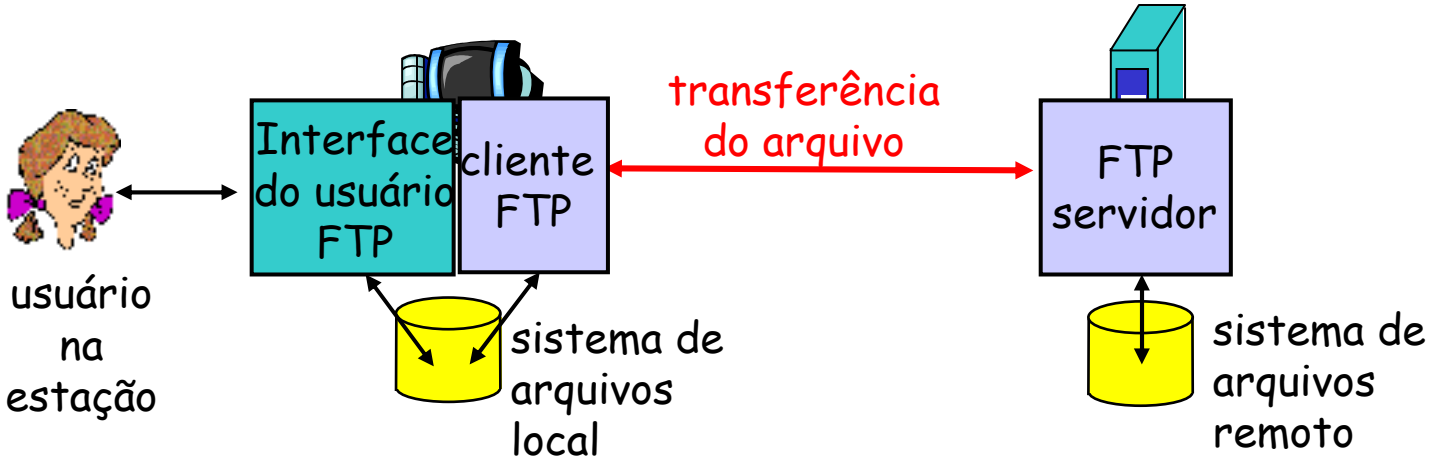
- autorização
- shopping carts
- recomendações
- Estado de sessões de usuários (Web e-mail)

Nota

Cookies e privacidade:

- O uso de cookies permite que o site "aprenda" muita coisa sobre você
- Você deve fornecer nome e e-mail para os sites;
- Ferramentas de buscas usam redirecionamento & cookies para aprender ainda mais;
- Agências de publicidade obtém suas informações através dos sites;

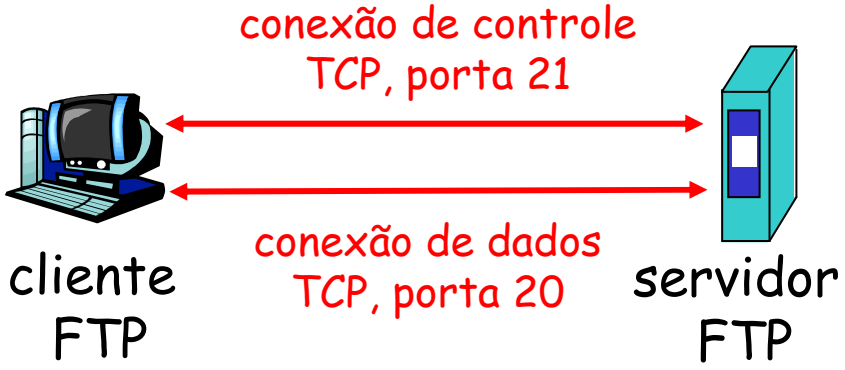
FTP: o protocolo de transferência de arquivos



- transferir arquivo de/para hospedeiro remoto
- modelo cliente/servidor
  - *cliente:* lado que inicia transferência (pode ser de ou para o sistema remoto)
  - *servidor:* hospedeiro remoto
- ftp: RFC 959
- servidor ftp: porta 21

FTP: conexões separadas p/ controle, dados

- Cliente FTP contacta servidor ftp na porta 21, especificando TCP como protocolo de transporte
- Cliente obtem autorização através da conexão de controle;
- O cliente acessa o diretório remoto através do envio de comandos pela conexão de controle;
- Quando o servidor recebe um comando para transferência de arquivo, o servidor abre uma conexão TCP com o cliente;
- Depois de transferir o arquivo a conexão é finalizada;



- são abertas duas conexões TCP paralelas:
  - controle: troca comandos, respostas entre cliente, servidor.

    "controle fora da banda"
  - dados: dados de arquivo de/para servidor

FTP: comandos, respostas

Comandos típicos:

- enviados em texto ASCII pelo canal de controle
- **USER *nome***
- **PASS *senha***
- **LIST** devolve lista de arquivos no directório corrente
- **RETR arquivo** recupera (lê) arquivo remoto
- **STOR arquivo** armazena (escreve) arquivo no hospedeiro remoto

Códigos de retorno típicos

- código e frase de status (como para http)
- **331 Username OK, password required**
- **125 data connection already open; transfer starting**
- **425 Can't open data connection**
- **452 Error writing file**

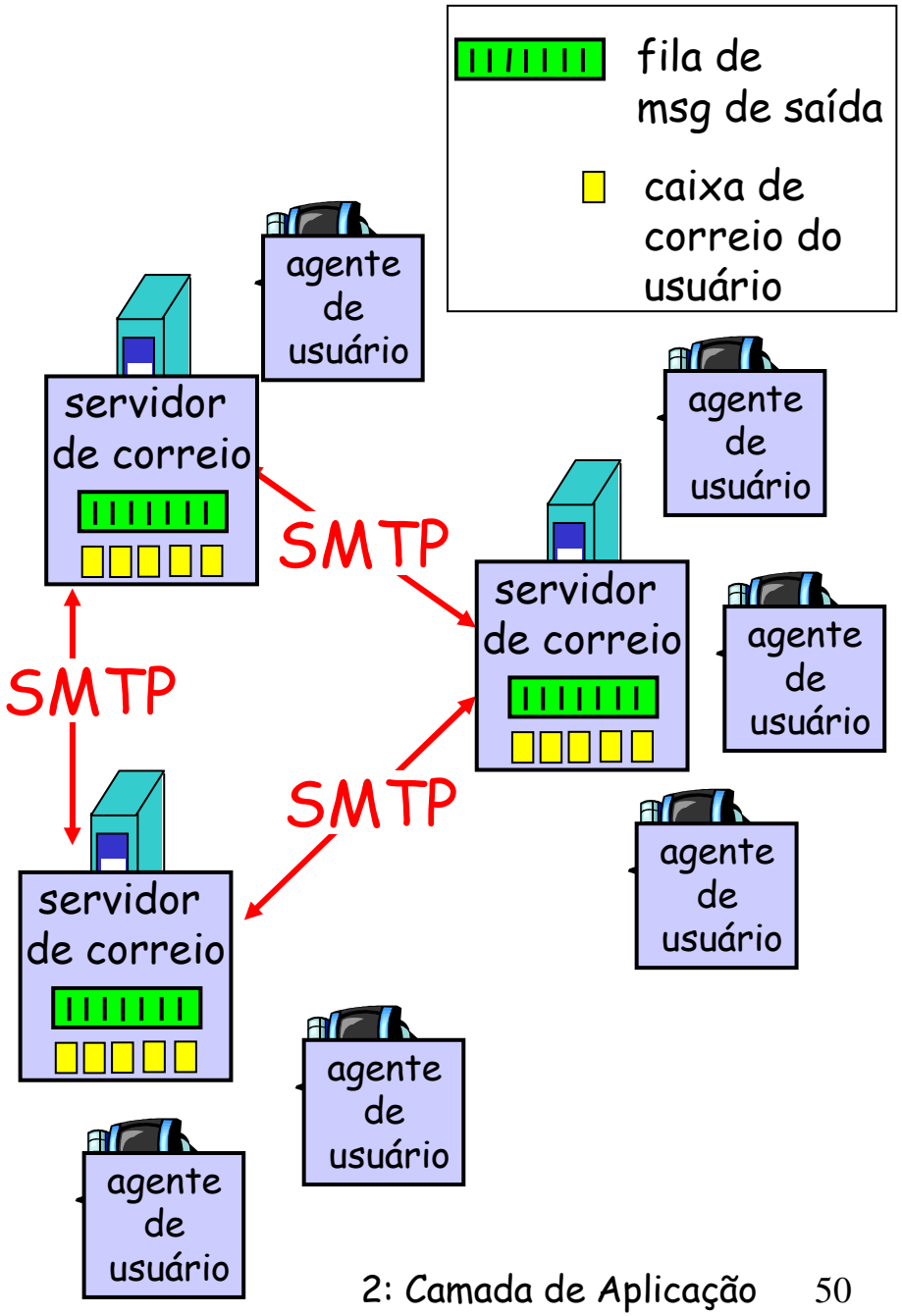
Correio Eletrônico

Três grandes componentes:

- agentes de usuário (UA)
- servidores de correio
- SMTP: simple mail transfer protocol

Agente de Usuário

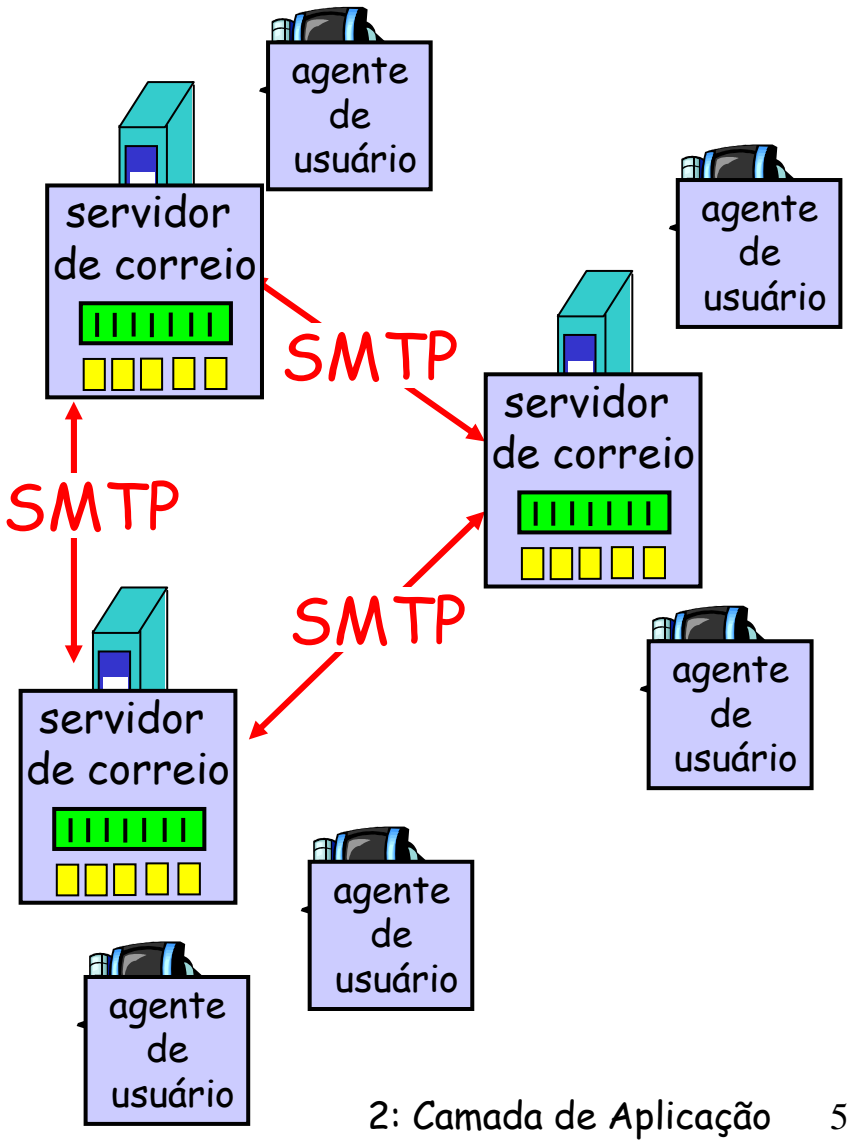
- a.k.a. "leitor de correio"
- compor, editar, ler mensagens de correio
- p.ex., Eudora, Outlook, elm, Netscape Messenger
- mensagens de saída e chegada são armazenadas no servidor



Correio Eletrônico: servidores de correio

Servidores de correio

- caixa de correio contém mensagens de chegada (ainda não lidas) p/ usuário
- fila de mensagens contém mensagens de saída (a serem enviadas)
- protocolo SMTP entre servidores de correio para transferir mensagens de correio
  - cliente: servidor de correio que envia
  - "servidor": servidor de correio que recebe

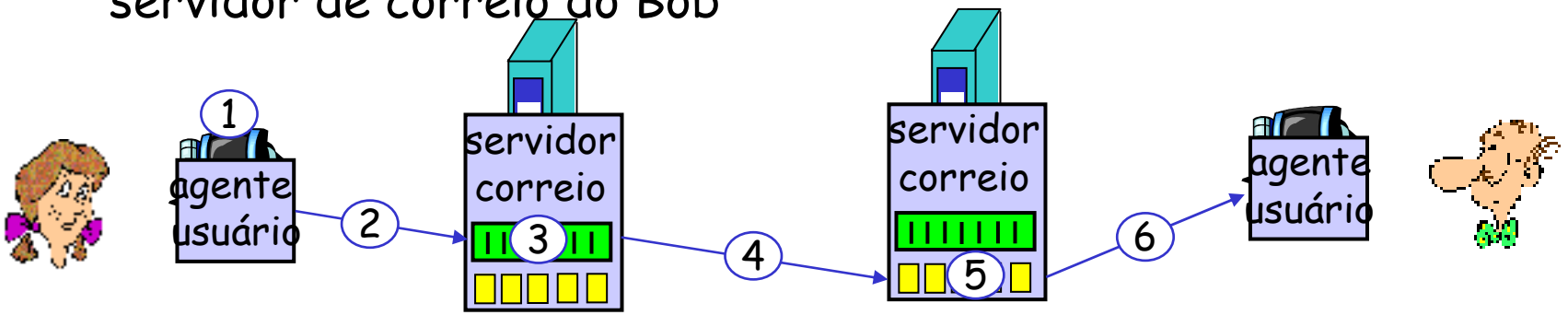


Correio Eletrônico: SMTP [RFC 821]

- usa TCP para a transferência confiável de msgs do correio do cliente ao servidor, porta 25
- transferência direta: servidor remetente ao servidor receptor
- três fases da transferência
  - handshaking (cumprimento)
  - transferência das mensagens
  - encerramento
- interação comando/resposta
  - comandos: texto ASCII
  - resposta: código e frase de status
- mensagens precisam ser em ASCII de 7-bits

Cenário: Alice envia msg para Bob

1) Alice usa UA para compor a mensagem e enviá-la para `bob@someschool.edu`
2) O UA da Alice envia a mensagem para o seu servidor de correio; a msg é colocada na fila de mensagens;
3) O cliente SMTP abre uma conexão TCP com o servidor de correio do Bob
4) SMTP cliente envia a msg da Alice através da conexão TCP;
5) Servidor de correio de Bob coloca a msg na caixa de correio de Bob;
6) Bob invoca o seu UA para ler a sua msg;



Interação SMTP típica

```
S: 220 doces.br
C: HELO consumidor.br
S: 250  Hello consumidor.br, pleased to meet you
C: MAIL FROM: <ana@consumidor.br>
S: 250 ana@consumidor.br... Sender ok
C: RCPT TO: <bernardo@doces.br>
S: 250 bernardo@doces.br ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Voce gosta de chocolate?
C:   Que tal sorvete?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 doces.br closing connection
```

Experimente você uma interação SMTP

⋮

- `telnet nomedoservidor 25`
- veja resposta 220 do servidor
- entre comandos HELO, MAIL FROM, RCPT TO, DATA, QUIT

estes comandos permite que você envie correio sem usar um cliente (leitor de correio)

SMTP: últimas palavras

- SMTP usa conexões persistentes
- smtp requerque a mensagem (cabeçalho e corpo) sejam em ASCII de 7-bits
- algumas cadeias de caracteres não são permitidas numa mensagem (p.ex., `CRLF.CRLF`). Logo a mensagem pode ter que ser codificada (normalmente em base-64 ou "quoted printable")
- servidor SMTP usa `CRLF.CRLF` para reconhecer o final da mensagem

Comparação com http

- HTTP : pull (puxar)
- email: push (empurrar)
- ambos tem interação comando/resposta, códigos de status em ASCII
- HTTP: cada objeto é encapsulado em sua própria mensagem de resposta
- SMTP: múltiplos objetos de mensagem enviados numa mensagem de múltiplas partes

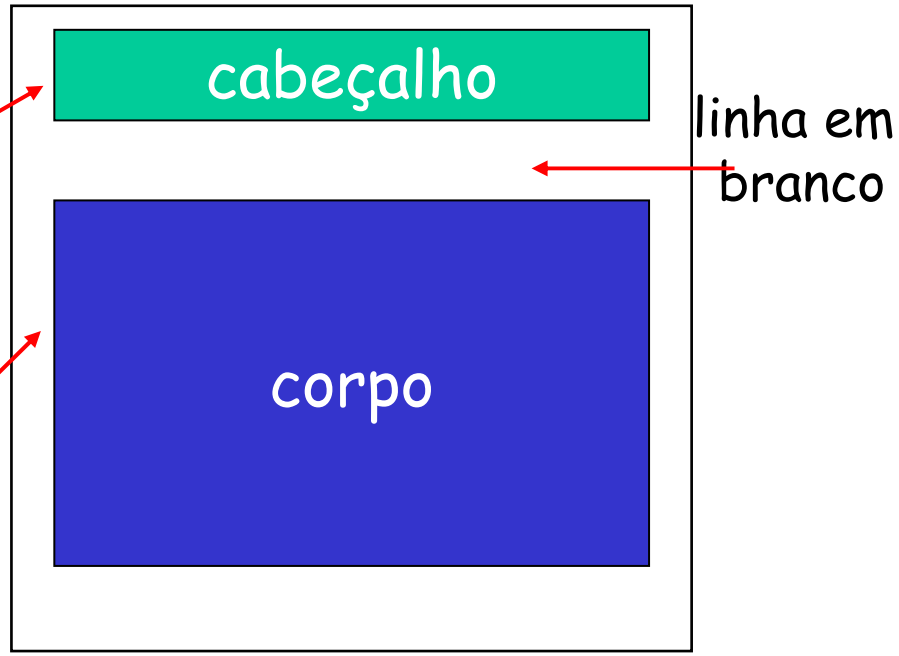
Formato de uma mensagem

SMTP: protocolo para trocar msgs de correio

RFC 822: padrão para formato de mensagem de texto:

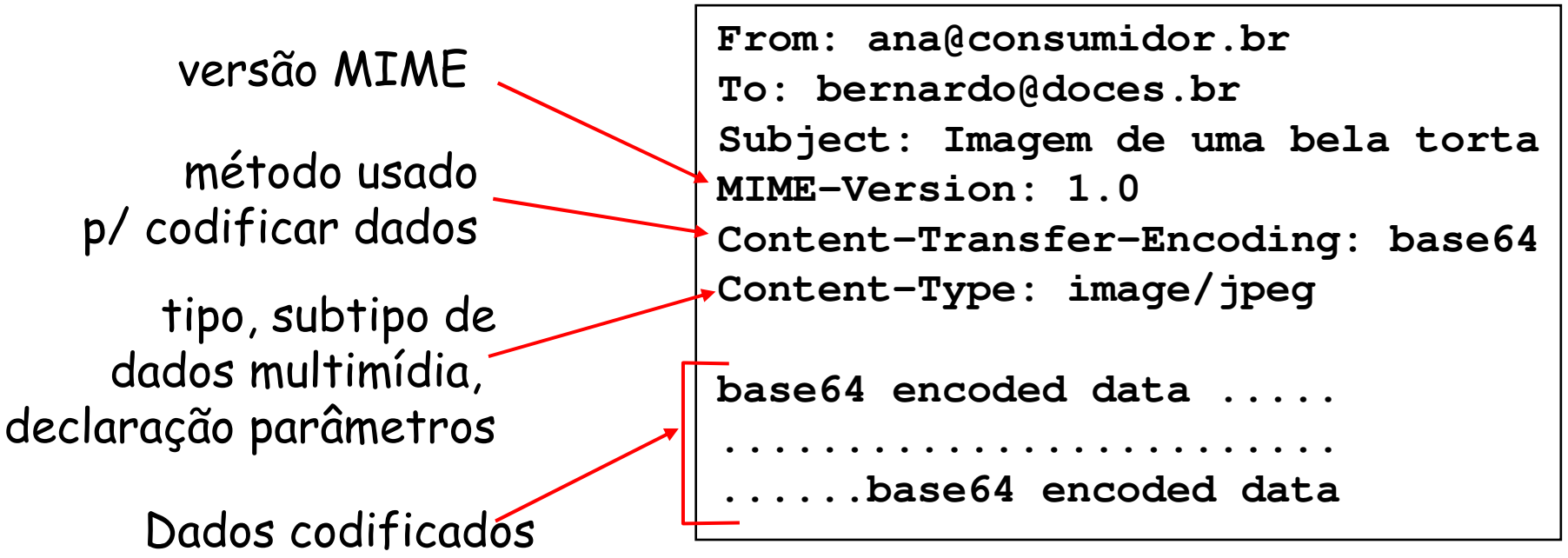
- linhas de cabeçalho, p.ex.,
  - To:
  - From:
  - Subject:

  diferentes dos comandos de SMTP!
- corpo
  - a "mensagem", somente de caracteres ASCII



Formato de uma mensagem: extensões para multimídia

- MIME: multimedia mail extension, RFC 2045, 2056
- linhas adicionais no cabeçalho da msg declaram tipo do conteúdo MIME



Tipos MIME

Content-Type: tipo/subtipo; parâmetros

Text

- subtipos exemplos: **plain, html**
- **charset="iso-8859-1", ascii**

Image

- subtipos exemplos : **jpeg, gif**

Video

- subtipos exemplos : **mpeg, quicktime**

Audio

- subtipos exemplos : **basic** (8-bit codificado mu-law), **32kadpcm** (codificação 32 kbps)

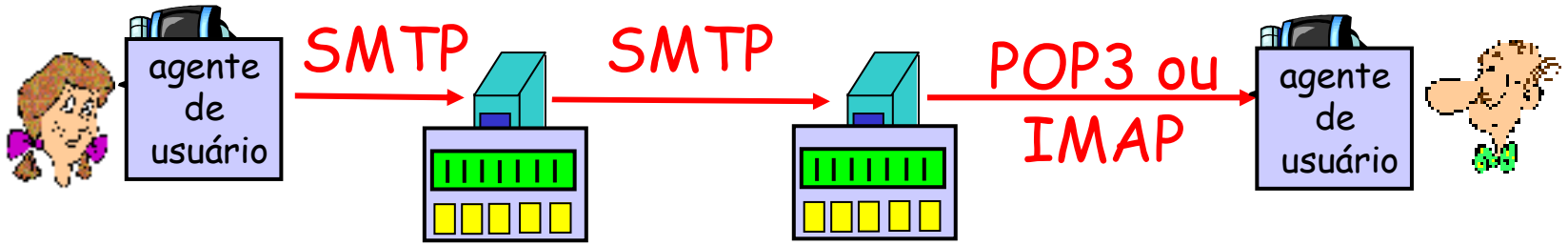
Application

- outros dados que precisam ser processados por um leitor para serem "visualizados"
- subtipos exemplos : **msword, octet-stream**

Tipo Multipart

```
From: ana@consumidor.br
To: bernardo@doces.br
Subject: Imagem de uma bela torta
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

--98766789
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain

caro Bernardo,
Anexa a imagem de uma torta deliciosa.
--98766789
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
......base64 encoded data
--98766789--
```

Protocolos de accesso ao correio



- SMTP: entrega/armazenamento no servidor do receptor
- protocolo de accesso ao correio: recupera do servidor
  - POP: Post Office Protocol [RFC 1939]
    - autorização (agente <-->servidor) e transferência
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - mais comandos (mais complexo)
    - manuseio de msgs armazenadas no servidor
  - HTTP: Hotmail , Yahoo! Mail, Webmail, etc.

Protocolo POP3

fase de autorização

- comandos do cliente:
  - **user**: declara nome
  - **pass**: senha
- servidor responde
  - **+OK**
  - **-ERR**

fase de transação, cliente:

- **list**: lista números das msgs
- **retr**: recupera msg por número
- **dele**: apaga msg
- **quit**

```
S: +OK POP3 server ready
C: user ana
S: +OK
C: pass faminta
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 "ler-e-apagar".
- Bob não pode reler suas msgs se ele mudar de cliente;
- POP3 não mantém estado;

IMAP

- Usa o modo: "ler-e-guardar" que posibilita acessar mensagens de vários clientes;
- Mantém todas as mensagens em um único lugar: servidor;
- Permite que o usuário organize suas msgs em pastas remotas como se fosse locais;
- IMAP mantém estado dos usuários durante as sessões:
  - Nomes e pastas e mapeia os IDs das msgs e o nome das pastas;

DNS: Domain Name System

Pessoas: muitos identificadores:
- CPF, nome, no. da Identidade

hospedeiros, roteadores Internet :
- endereço IP (32 bit) - usado p/ endereçar datagramas
- "nome", ex., jambo.ic.uff.br - usado por gente

P: como mapear entre nome e endereço IP?

Domain Name System:
- *base de dados distribuída* implementada na hierarquia de muitos *servidores de nomes*
- *protocolo de camada de aplicação* permite que hospedeiros, roteadores, servidores de nomes se comuniquem para *resolver* nomes (tradução endereço/nome)
  - note: função imprescindível da Internet implementada como protocolo de camada de aplicação
  - complexidade na borda da rede

DNS

- Roda sobre UDP e usa a porta 53
- Especificado nas RFCs 1034 e 1035 e atualizado em outras RFCs.
- Outros serviços:
  - apelidos para hospedeiros (aliasing)
  - apelido para o servidor de mails
  - distribuição da carga

DNS: Domain Name System

Pessoas: muitos identificadores:
- RG, nome, passaporte

Internet hospedeiros, roteadores:
- Endereços IP (32 bits) - usados para endereçar datagramas
- "nome", ex.: gaia.cs.umass.edu - usados por humanos

P.: Relacionar nomes com endereços IP?

Domain Name System:
- **Base de dados distribuída** implementada numa hierarquia de muitos **servidores de nomes**
- **Protocolo de camada de aplicação** hospedeiro, roteadores se comunicam com servidores de nomes para **resolver** nomes (translação nome/endereço)
  - Nota: função interna da Internet, implementada como protocolo da camada de aplicação
  - Complexidade na "borda" da rede

Servidores de nomes DNS

Por que não centralizar o DNS?

- ponto único de falha
- volume de tráfego
- base de dados centralizada e distante
- manutenção (da BD)

Não é escalável!

- Nenhum servidor mantém todos os mapeamento nome-para-endereço IP

servidor de nomes local:

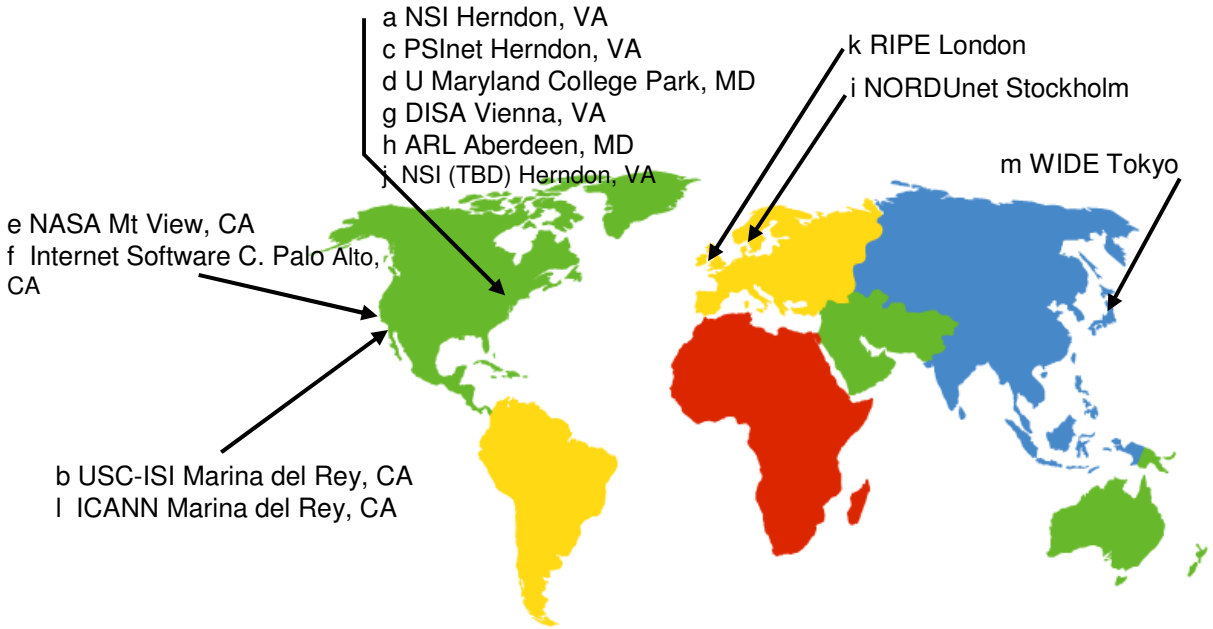
- cada provedor, empresa tem *servidor de nomes local (default)*
- pedido DNS de hospedeiro vai primeiro ao servidor de nomes local

servidor de nomes oficial:

- p/ hospedeiro: guarda nome, endereço IP dele
- pode realizar tradução nome/endereço para este nome

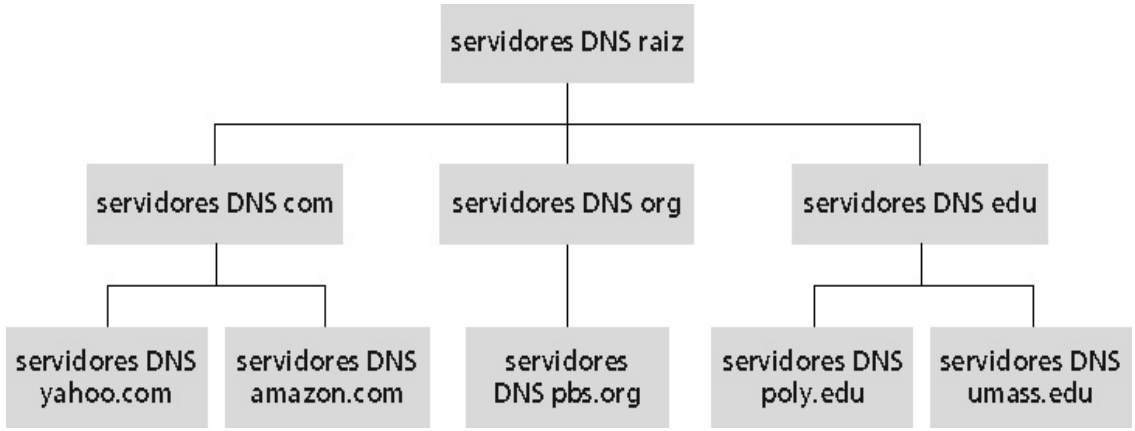
DNS: Servidores raiz

- procurado por servidor local que não consegue resolver o nome
- servidor raiz:
  - procura servidor oficial se mapeamento é desconhecido
  - obtém tradução
  - devolve mapeamento ao servidor local



13 servidores raíz no mundo

Base de dados distribuída, hierárquica



Cliente quer o IP para www.amazon.com; 1ª aprox.:

- Cliente consulta um servidor de raiz para encontrar o servidor DNS com
- Cliente consulta o servidor DNS com para obter o servidor DNS amazon.com
- Cliente consulta o servidor DNS amazon.com para obter o endereço IP para www.amazon.com

Servidores TLD e autoritários

Servidores top-level domain (TLD): responsáveis pelos domínios com, org, net, edu etc e todos os domínios **top-level** nacionais uk, fr, ca, jp.

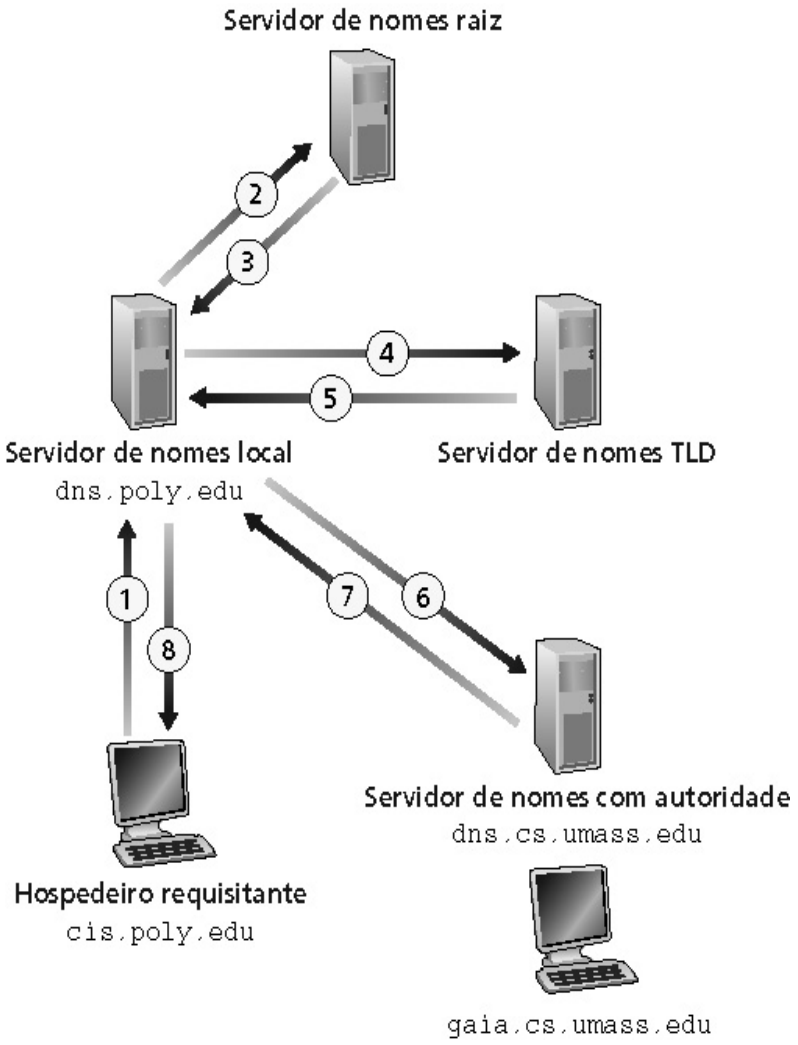
- Network Solutions mantém servidores para o TLD "com" TLD
- Educause para o TLD "edu"

Servidores DNS autorizados: servidores DNS de organizações, provêm nome de hospedeiro autorizado para mapeamentos IP para servidores de organizações (ex.: Web e mail).

- Podem ser mantidos por uma organização ou provedor de serviços

Exemplo

- O hospedeiro em cis.poly.edu quer o endereço IP para gaia.cs.umass.edu



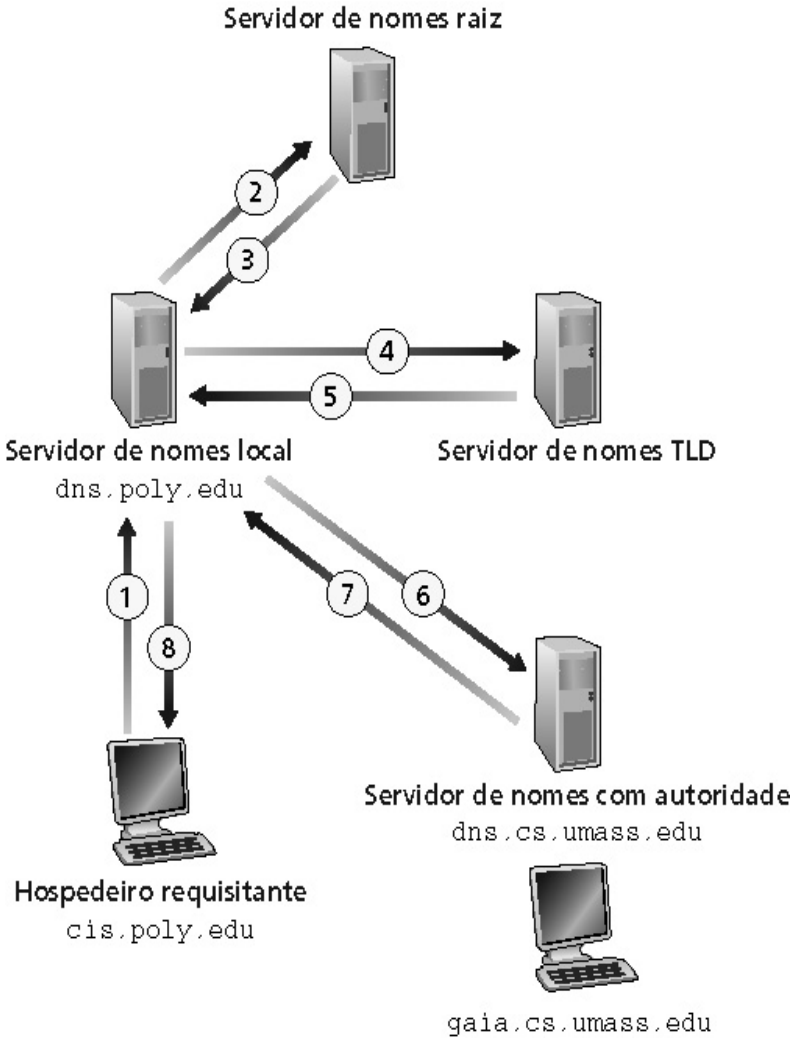
Consultas recursivas

Consulta recursiva:

- Transfere a tarefa de resolução do nome para o servidor de nomes consultado
- Carga pesada?

Consulta encadeada:

- Servidor contatado responde com o nome de outro servidor de nomes para contato
- "eu não sei isto, mas pergunte a este servidor"



DNS: armazenando e atualizando registros

Uma vez que um servidor de nomes apreende um mapeamento, ele armazena o mapeamento num registro do tipo **cache**

- Registro do cache tornam-se obsoletos (desaparecem) depois de um certo tempo
- Servidores TLD são tipicamente armazenados em cache nos servidores de nome locais

Mecanismos de atualização e notificação estão sendo projetados pelo IETF

- RFC 2136
- http://www.ietf.org/html.charters/dnsind-charter.html

Registros do DNS

DNS: base de dados distribuída que armazena registros de recursos (RR)

formato dos RR: **(name, value, type,ttl)**

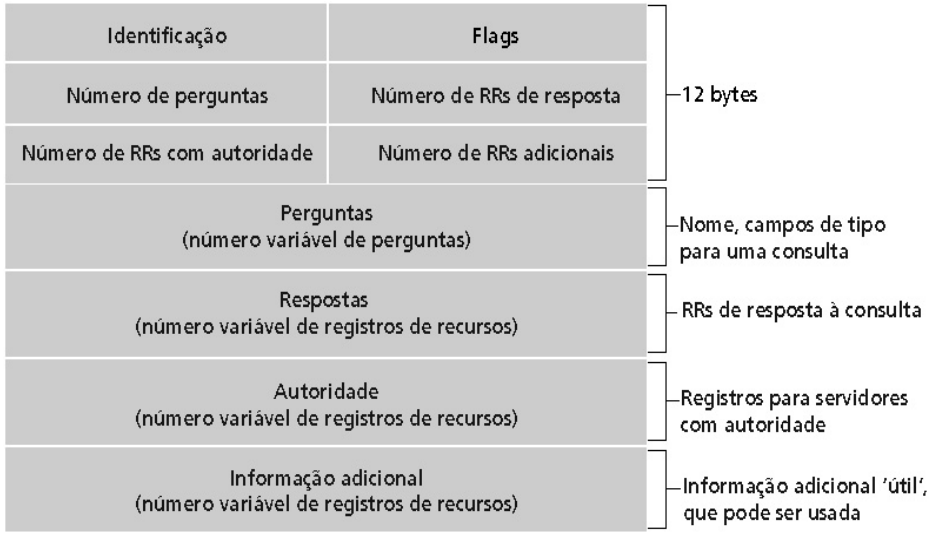
- Type = A
  - **name** é o nome do computador
  - **value** é o endereço IP
- Type = NS
  - **name** é um domínio (ex.: foo.com)
  - **value** é o endereço IP do servidor de nomes autorizados para este domínio
- Type = CNAME
  - **name** é um "apelido" para algum nome "canônico" (o nome real) www.ibm.com é realmente servereast.backup2.ibm.com
  - **value** é o nome canônico
- Type = MX
  - **value** é o nome do servidor de correio associado com **name**

DNS: protocolo e mensagem

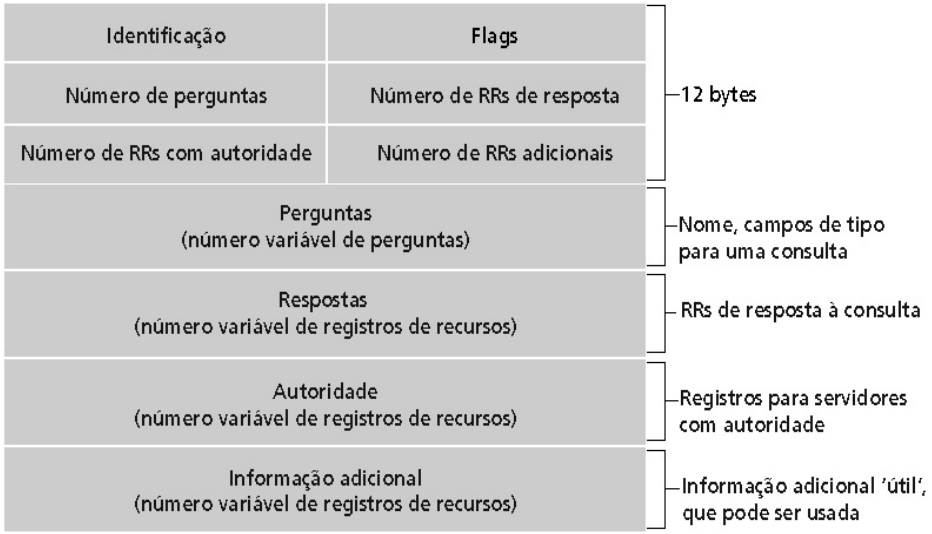
Protocolo DNS: mensagem de **consulta** e **resposta** , ambas com o mesmo **formato de mensagem**

Cabeçalho da msg

- Identificação: número de 16 bits para consulta, resposta usa o mesmo número
- Flags:
  - Consulta ou resposta
  - Recursão desejada
  - Recursão disponível
  - Resposta é autorizada



Camada de aplicação



DNS: protocolo e mensagens

Camada de aplicação

- Exemplo: empresa recém-criada "Network Utopia"
- Registrar o nome networkuptopia.com num "registrar" (ex.: Network Solutions)
  - É necessário fornecer ao registrar os nomes e endereços IP do seu servidor nomes autorizados (primário e secundário)
  - Registrar insere dois RRs no servidor TLD do domínio com:

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

- No servidor autorizado, inserir um registro Tipo A para www.networkuptopia.com e um registro Tipo MX para networkutopia.com
- Como as pessoas obtêm o endereço IP do seu Web site?

Inserindo registros no DNS

DNS: uso de cache, atualização de dados

- uma vez que um servidor qualquer aprende um mapeamento, ele o coloca numa *cache* local
  - futuras consultas são resolvidas usando dados da cache
  - entradas na cache são sujeitas a temporização (desaparecem depois de um certo tempo)
    ttl = time to live (sobrevida)
- estão sendo projetados pela IETF mecanismos de atualização/notificação dos dados
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

Registros DNS

DNS: BD distribuído contendo *registros de recursos* (RR)

formato RR: **(nome, valor, tipo, sobrevida)**

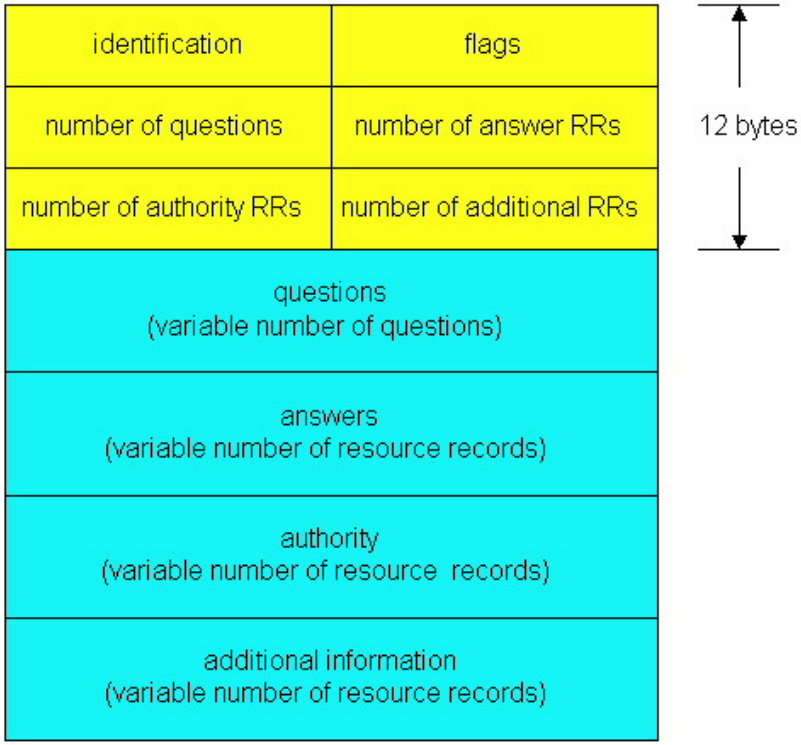
- Tipo=A
  - **nome** é nome de hospedeiro
  - **valor** é o seu endereço IP
- Tipo=NS
  - **nome** é domínio (p.ex. foo.com.br)
  - **valor** é endereço IP de servidor oficial de nomes para este domínio
- Tipo=CNAME
  - **nome** é nome alternativo (alias) para algum nome "canônico" (verdadeiro)
  - **valor** é o nome canônico
- Tipo=MX
  - **nome** é domínio
  - **valor** é nome do servidor de correio para este domínio

DNS: protocolo e mensagens

protocolo DNS: mensagens de *pedido* e *resposta*, ambas com o mesmo *formato de mensagem*

cabeçalho de msg

- identificação: ID de 16 bit para pedido, resposta ao pedido usa mesmo ID
- flags:
  - pedido ou resposta
  - recursão desejada
  - recursão permitida
  - resposta é oficial



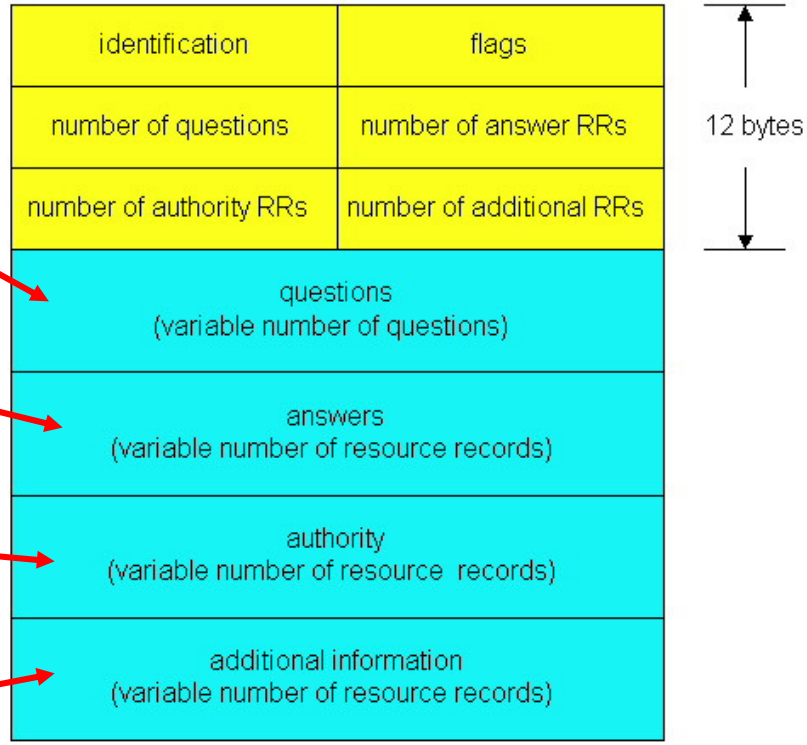
DNS: protocolo e mensagens

campos de nome, e de tipo num pedido

RRs em resposta ao pedido

registros para outros servidores oficiais

info adicional "relevante" que pode ser usada



Programação com sockets

Meta: aprender a construir aplicações cliente/servidor que se comunicam usando sockets

API Sockets

- apareceu no BSD4.1 UNIX em 1981
- são explicitamente criados, usados e liberados por apls
- paradigma cliente/servidor
- dois tipos de serviço de transporte via API Sockets
  - datagrama não confiável
  - fluxo de bytes, confiável

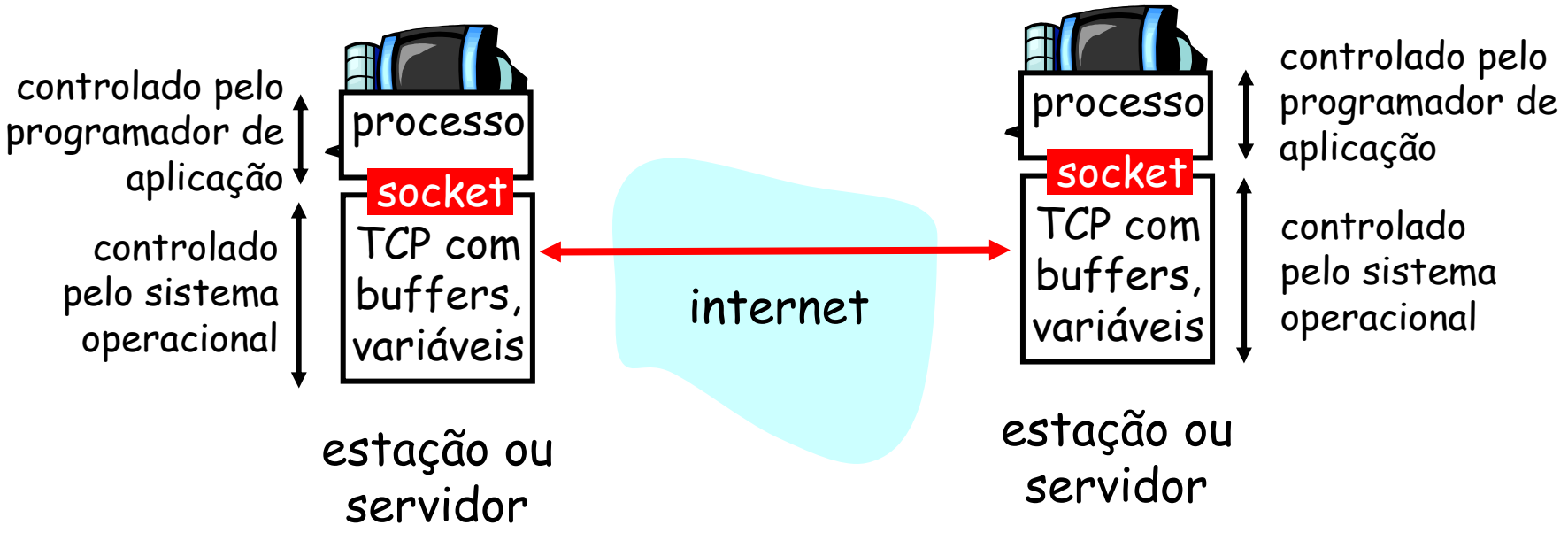
socket

uma interface (uma "porta"), local ao hospedeiro, criada por e pertencente à aplicação, e controlado pelo SO, através da qual um processo de aplicação pode tanto enviar como receber mensagens para/de outro processo de aplicação (remoto ou local)

Programação com sockets usando TCP

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

Serviço TCP: transferência confiável de bytes de um processo para outro



Programação de sockets com TCP

Cliente deve contatar o servidor
- Processo servidor já deve estar em execução
- Servidor deve ter criado socket (porta) que aceita o contato do cliente

Cliente contata o servidor
- Criando um socket TCP local
- Especificando endereço IP e número da porta do processo servidor
- Quando o cliente cria o socket: cliente TCP estabelece conexão com o TCP do servidor

Quando contatado pelo cliente, o TCP do servidor cria um novo socket para o processo servidor comunicar-se com o cliente
- Permite ao servidor conversar com múltiplos clientes
- Números da porta de origem são usados para distinguir o cliente (mais no capítulo 3)

Ponto de vista da aplicação
TCP fornece a transferência confiável, em ordem de bytes ("pipe") entre o cliente e o servidor

Jargão stream

- Um stream é uma seqüência de caracteres que fluem para dentro ou para fora de um processo
- Um stream de entrada é agregado a alguma fonte de entrada para o processo, ex.: teclado ou socket
- Um stream de saída é agregado a uma fonte de saída, ex.: monitor ou socket

Programação de sockets com TCP

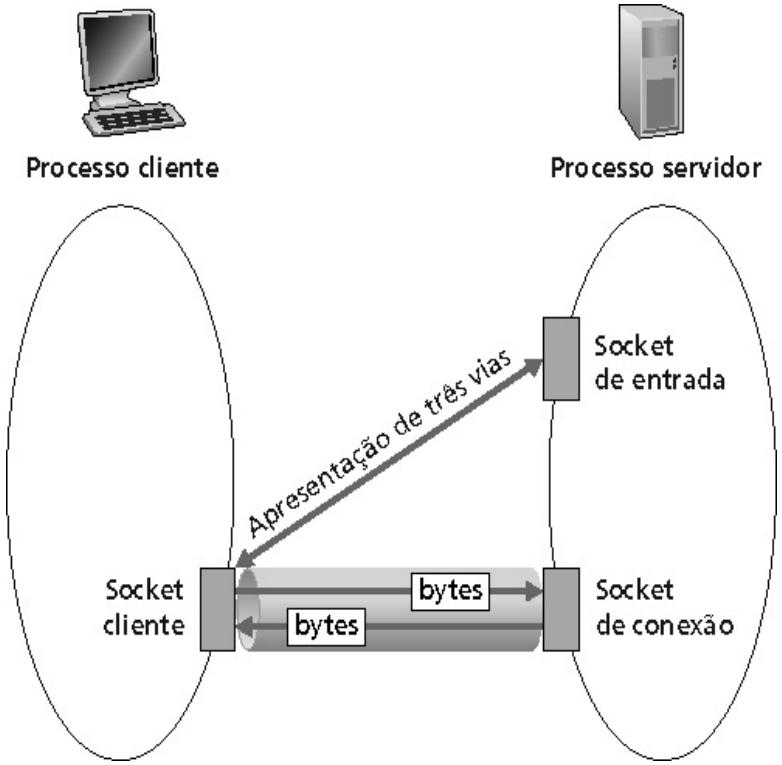
Exemplo de aplicação cliente-servidor:

1) Cliente lê linha da entrada-padrão do sistema (**inFromUser** stream), envia para o servidor via socket (**outToServer** stream)

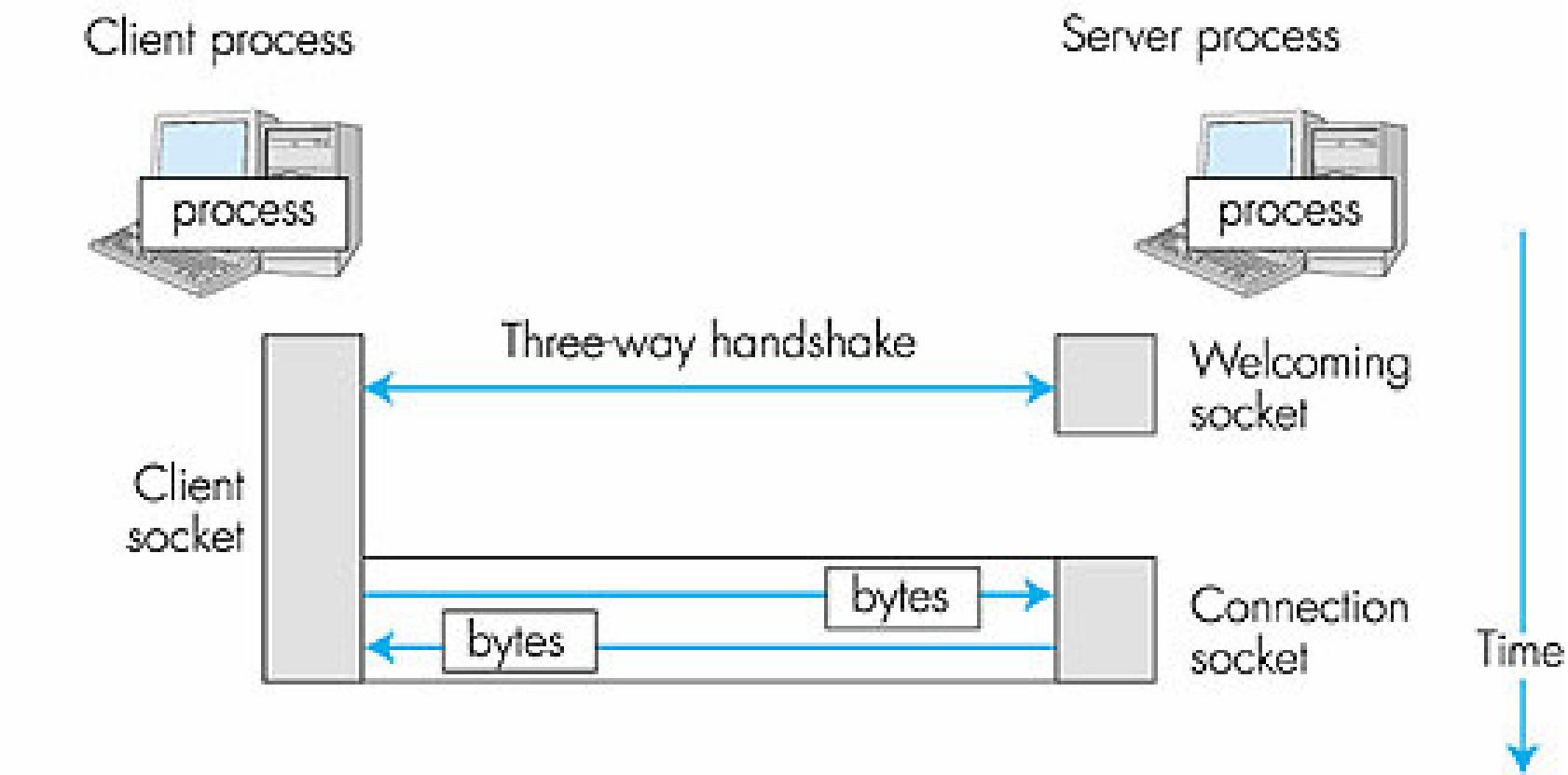
2) Servidor lê linha do socket

3) Servidor converte linha para letras maiúsculas e envia de volta ao cliente

4) Cliente lê a linha modificada através do (**inFromServer** stream)

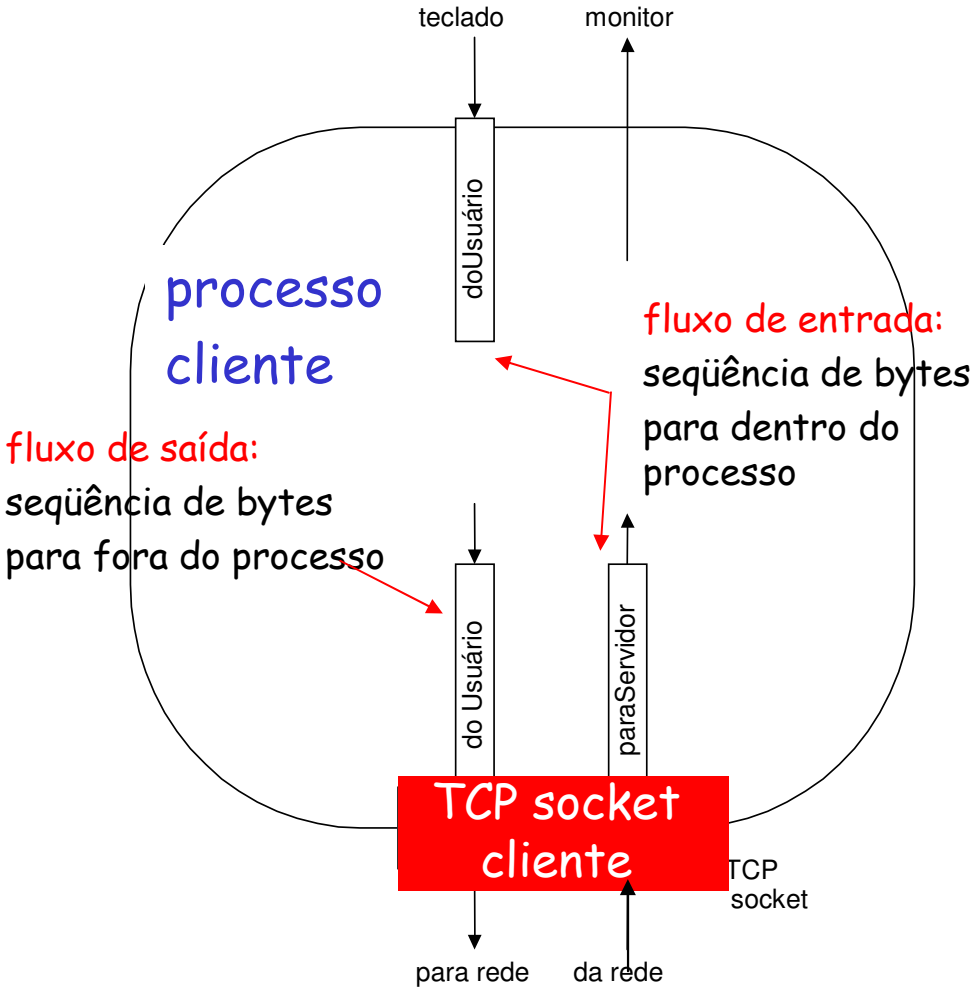


Comunicação entre sockets

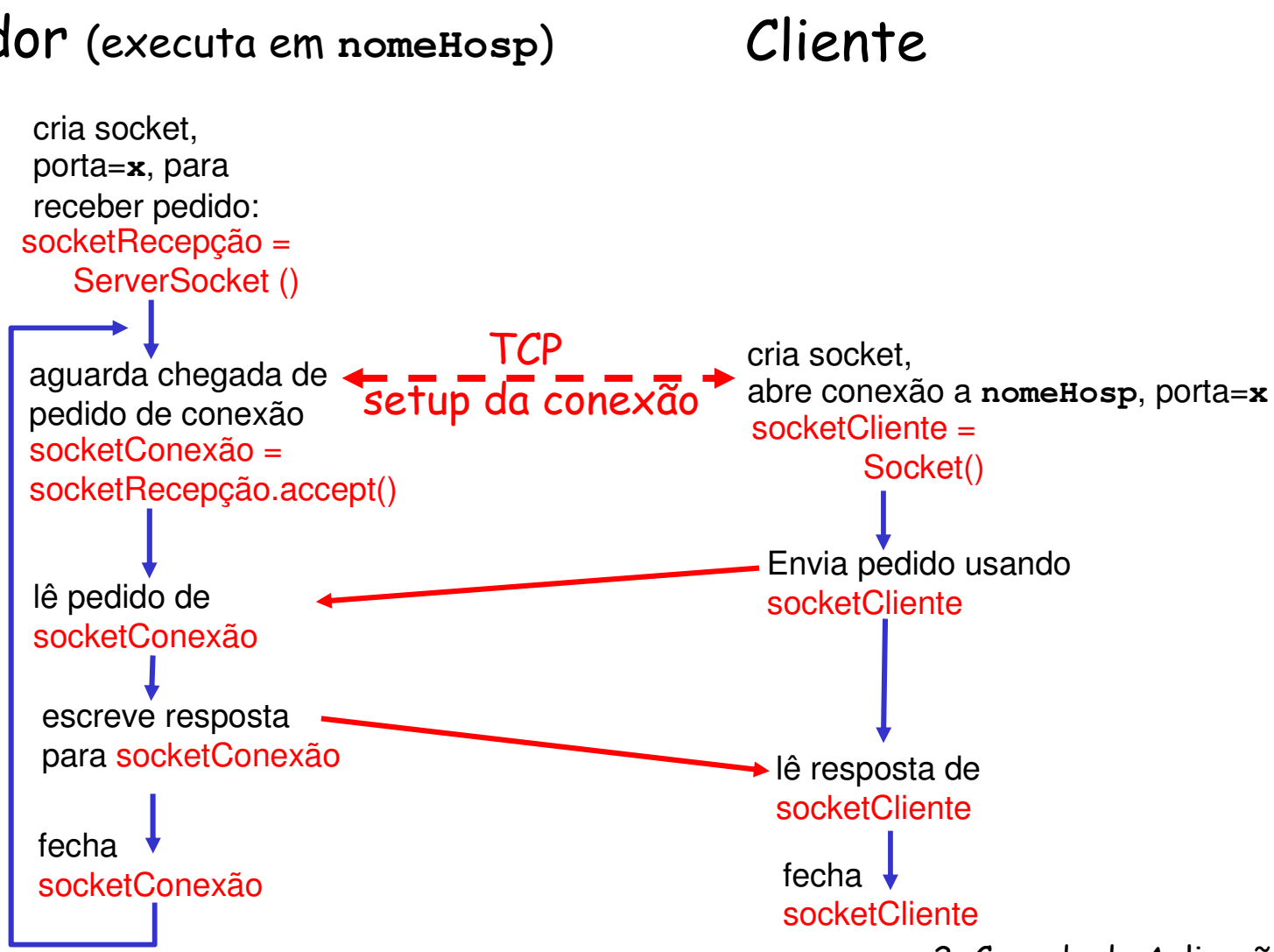


Exemplo de aplicação cliente-servidor

- cliente lê linha da entrada padrão (fluxo **`doUsuário`**), envia para servidor via socket (fluxo **`paraServidor`**)
- servidor lê linha do socket
- servidor converte linha para letras maiúsculas, devolve para o cliente
- cliente lê linha modificada do socket (fluxo **`doServidor`**), imprime-a



Interações cliente/servidor usando o TCP



Exemplo: cliente Java (TCP)

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

    public static void main(String argv[]) throws Exception
    {
        String frase;
        String fraseModificada;

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

        Socket socketCliente = new Socket("nomeHosp", 6789);

        DataOutputStream paraServidor =
          new DataOutputStream(socketCliente.getOutputStream());
```

Cria fluxo de entrada

Cria socket de cliente, conexão ao servidor

Cria fluxo de saída ligado ao socket

Exemplo: cliente Java (TCP), cont.

Cria fluxo de entrada ligado ao socket

```
BufferedReader doServidor =
  new BufferedReader(new
  InputStreamReader(socketCliente.getInputStream()));

frase = doUsuario.readLine();

paraServidor.writeBytes(frase + '\n');

fraseModificada = doServidor.readLine();

System.out.println("Do Servidor: " + fraseModificada);

socketCliente.close();

    }
}
```

Envia linha ao servidor

Lê linha do servidor

Exemplo: servidor Java (TCP)

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

class servidorTCP {

  public static void main(String argv[]) throws Exception
    {
      String fraseCliente;
      StringfFraseMaiusculas;

      ServerSocket socketRecepcao = new ServerSocket(6789);

      while(true) {

        Socket socketConexao = socketRecepcao.accept();

       BufferedReader doCliente =
          new BufferedReader(new
          InputStreamReader(socketConexao.getInputStream()));
```

Cria socket para recepção na porta 6789

Aguarda, no socket para recepção, o contato do cliente

Cria fluxo de entrada, ligado ao socket

Exemplo: servidor Java (TCP), cont

Cria fluxo de saída, ligado ao socket

Lê linha do socket

Escreve linha ao socket

```
DataOutputStream  paraCliente =
  new DataOutputStream(socketConexão.getOutputStream());

fraseCliente= doCliente.readLine();

fraseEmMaiusculas= fraseCliente.toUpperCase() + '\n';

paraClient.writeBytes(fraseEmMaiusculas);
            }
        }
    }
```

Final do laço while, volta ao início e aguarda conexão de outro cliente

Programação com sockets usando UDP

UDP: não tem "conexão" entre cliente e servidor

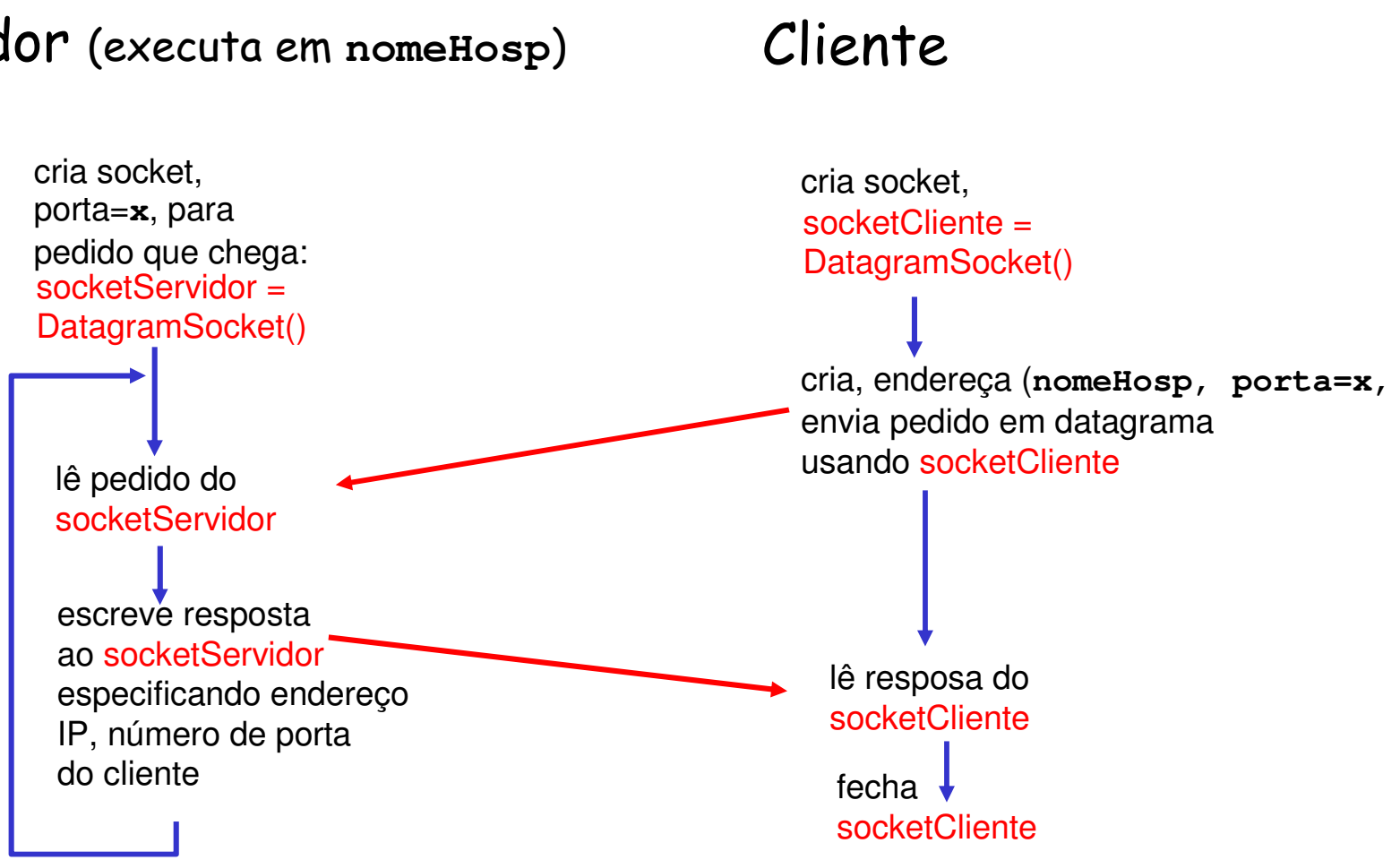
- não tem "handshaking"
- remetente coloca explicitamente endereço IP e porta do destino
- servidor deve extrair endereço IP, porta do remetente do datagrama recebido

UDP: dados transmitidos podem ser recebidos fora de ordem, ou perdidos

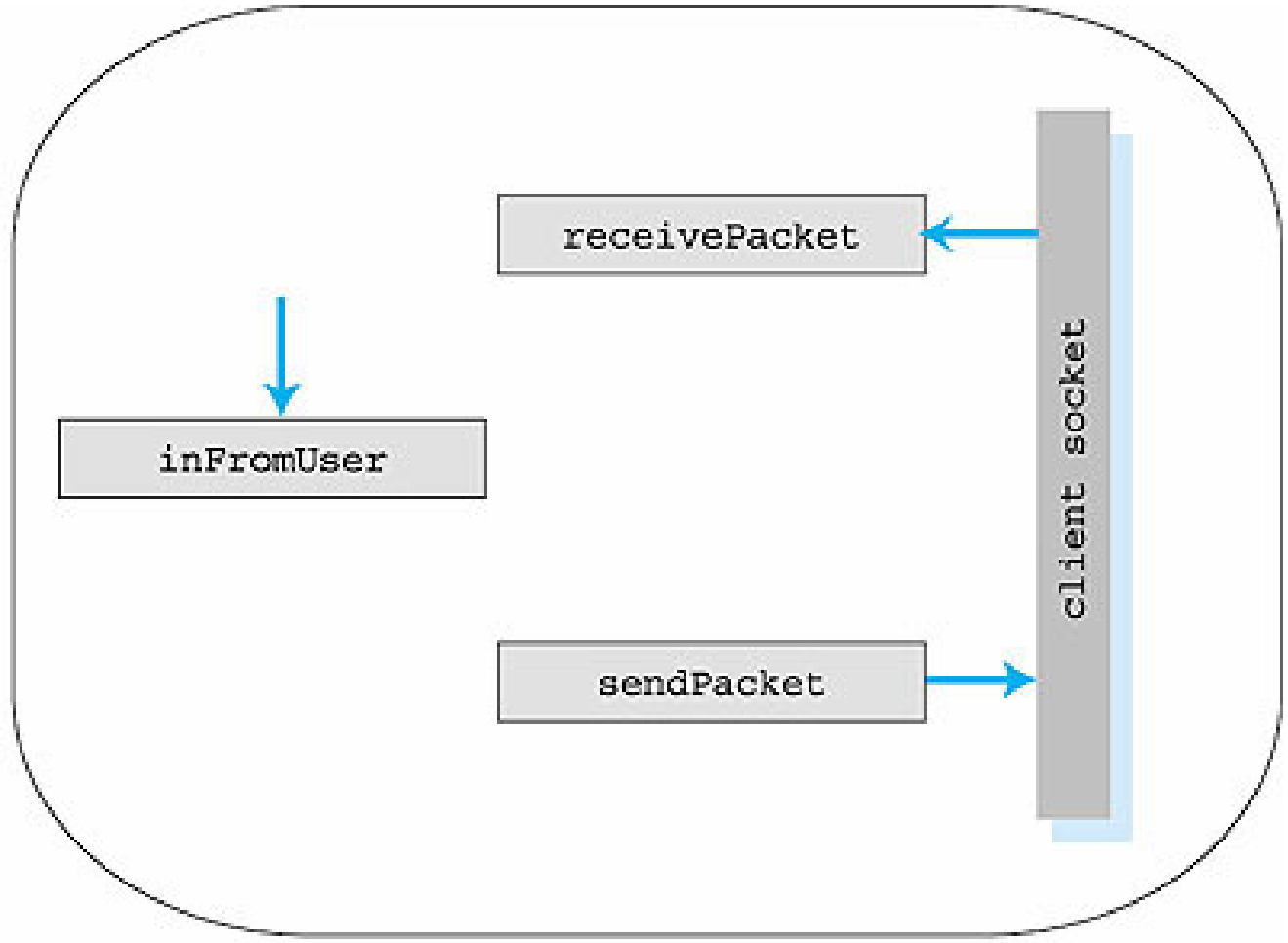
ponto de vista da aplicação

UDP provê transferência não confiável de grupos de bytes ("datagramas") entre cliente e servidor

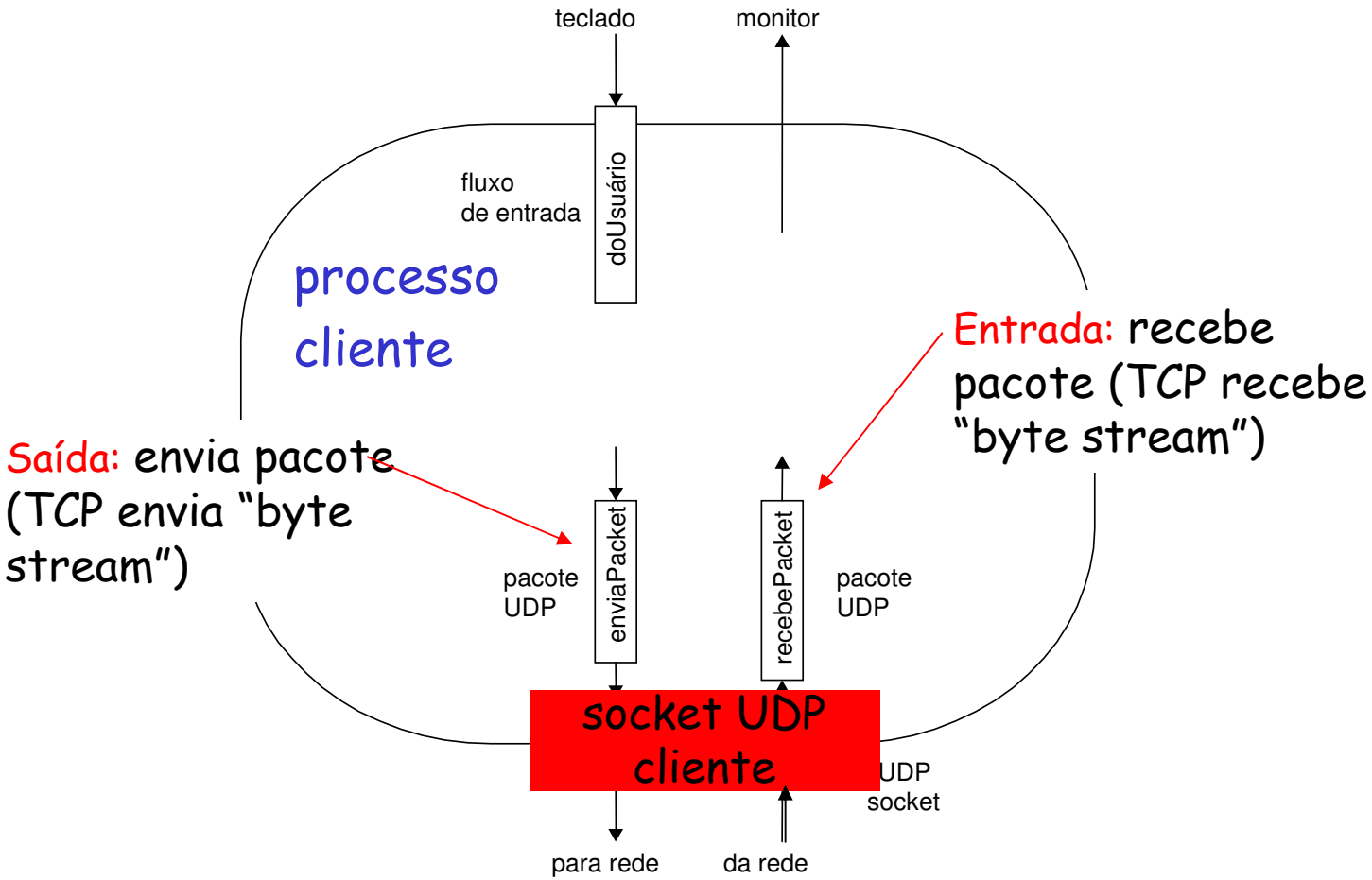
Interações cliente/servidor usando o UDP



Cliente UDP



Exemplo: cliente Java (UDP)



Exemplo: cliente Java (UDP)

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

class clienteUDP {
    public static void main(String args[]) throws Exception
    {
      BufferedReader do Usuario=
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket socketCliente = new DatagramSocket();

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

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

      String frase = doUsuario.readLine();

      sendData = frase.getBytes();
```

Cria fluxo de entrada

Cria socket de cliente

Traduz nome de hospedeiro ao endereço IP usando DNS

Exemplo: cliente Java (UDP) cont.

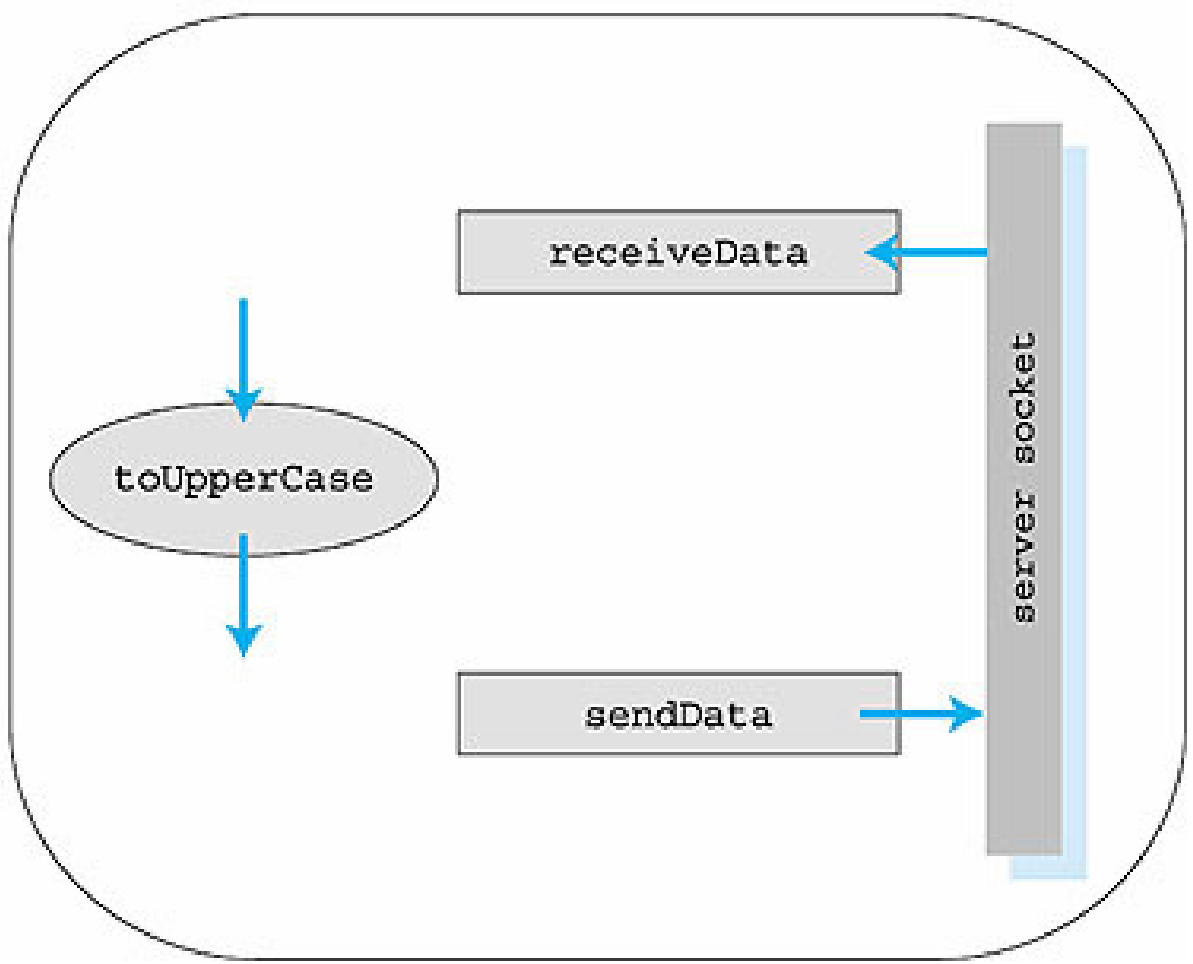
Cria datagrama com dados para enviar, comprimento, endereço IP, porta

Envia datagrama ao servidor

Lê datagrama do servidor

```
                DatagramPacket pacoteEnviado =
                  new DatagramPacket(dadosEnvio, dadosEnvio.length,
                        IPAddress, 9876);

                socketCliente.send(pacoteEnviado);

                DatagramPacket pacoteRecebido =
                   new DatagramPacket(dadosRecebidos, dadosRecebidos.length);

                socketCliente.receive(pacoteRecebido);

                String fraseModificada =
                    new String(pacoteRecebido.getData());

                System.out.println("Do Servidor:" + fraseModificada);
                socketCliente.close();
                }
        }
```

Servidor UDP



Exemplo: servidor Java (UDP)

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

class servidorUDP  {
  public static void main(String args[]) throws Exception
    {
      DatagramSocket socketServidor = new DatagramSocket(9876);

      byte[] dadosRecebidos = new byte[1024];
      byte[] dadosEnviados  = new byte[1024];

      while(true)
       {
          DatagramPacket pacoteRecebido =
           new DatagramPacket(dadosRecebidos,
                              dadosRecebidos.length);

          socketServidor.receive(pacoteRecebido);
```

Cria socket para datagramas na porta 9876

Aloca memória para receber datagrama

Recebe datagrama

Exemplo: servidor Java (UDP), cont

```
String frase = new String(pacoteRecebido.getData());

InetAddress IPAddress = pacoteRecebido.getAddress();

int porta = pacoteRecebido.getPort();

String fraseEmMaiusculas = frase.toUpperCase();

dadosEnviados = fraseEmMaiusculas.getBytes();

DatagramPacket pacoteEnviado =
   new DatagramPacket(dadosEnviados,
            dadosEnviados.length, IPAddress, porta);

socketServidor.send(pacoteEnviado);
    }
  }
}
```

Obtém endereço IP, no. de porta do remetente

Cria datagrama p/ enviar ao cliente

Escreve datagrama no socket

Fim do laço while, volta ao início e aguarda chegar outro datagrama

Servidor Web Simples

- Funções do servidor Web:
  - Trata apenas um pedido HTTP por vez
  - Aceita e examina o pedido HTTP
  - Recupera o arquivo pedido do sistema de arquivos do servidor
  - Cria uma mensagem de resposta HTTP consistindo do arquivo solicitado precedido por linhas de cabeçalho
  - Envia a resposta diretamente ao cliente
  - Depois de criado o servidor, pode-se requisitar um arquivo utilizando um browser;

Servidor Web Simples

Contém a classe StringTokenizer que é usada para examinar o pedido

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

class WebServer {
   public static void main(String argv[]) throws Exception
   {
      String requestMessageLine;
      String fileName;

      ServerSocket listenSocket = new ServerSocket(6789);
      Socket connectionSocket = listenSocket.accept();

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

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

Primeira linha da mensagem de pedido HTTP e Nome do arquivo solicitado

Aguarda conexão do cliente

Cria fluxo de Entrada

Cria fluxo de Saída

Servidor Web Simples, cont

Lê a primeira linha do pedido HTTP que deveria ter o seguinte formato: GET file_name HTTP/1.0

```
requestMessageLine = inFromClient.readLine();
```

Examina a primeira linha da mensagem para extrair o nome do arquivo

```
StringTokenizer tokenizedLine =
   new StringTokenizer(requestMessageLine);
if (tokenizedLine.nextToken().equals("GET")){
   fileName = tokenizedLine.nextToken();
   if (fileName.startsWith("/") == true )
         fileName = fileName.substring(1);

   File file = new File(fileName);
   int numOfBytes = (int) file.length();
```

Associa o fluxo inFile ao arquivo fileName

```
   FileInputStream inFile = new FileInputStream (
         fileName);
```

Determina o tamanho do arquivo e constrói um vetor de bytes do mesmo tamanho

```
   byte[] fileInBytes = new byte[];
   inFile.read(fileInBytes);
```

Servidor Web Simples, cont

Inicia a construção da mensagem de resposta

Transmissão do cabeçalho da resposta HTTP.

```
outToClient.writeBytes(
        "HTTP/1.0 200 Document Follows\r\n");

if (fileName.endsWith(".jpg"))
   outToClient.writeBytes("Content-Type: image/jpeg\r\n");
if (fileName.endsWith(".gif"))
   outToClient.writeBytes("Content-Type:
         image/gif\r\n");
outToClient.writeBytes("Content-Length: " + numOfBytes +
         "\r\n");
outToClient.writeBytes("\r\n");

outToClient.write(fileInBytes, 0, numOfBytes);
connectionSocket.close();
}

else System.out.println("Bad Request Message");
}
}
```

Programação de Sockets: referências

Tutorial sobre linguagem C (audio/slides):

- "Unix Network Programming" (J. Kurose),

http://manic.cs.umass.edu.

Tutoriais sobre Java:

- "Socket Programming in Java: a tutorial," http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html

P2P compartilhamento de arquivos

Exemplo

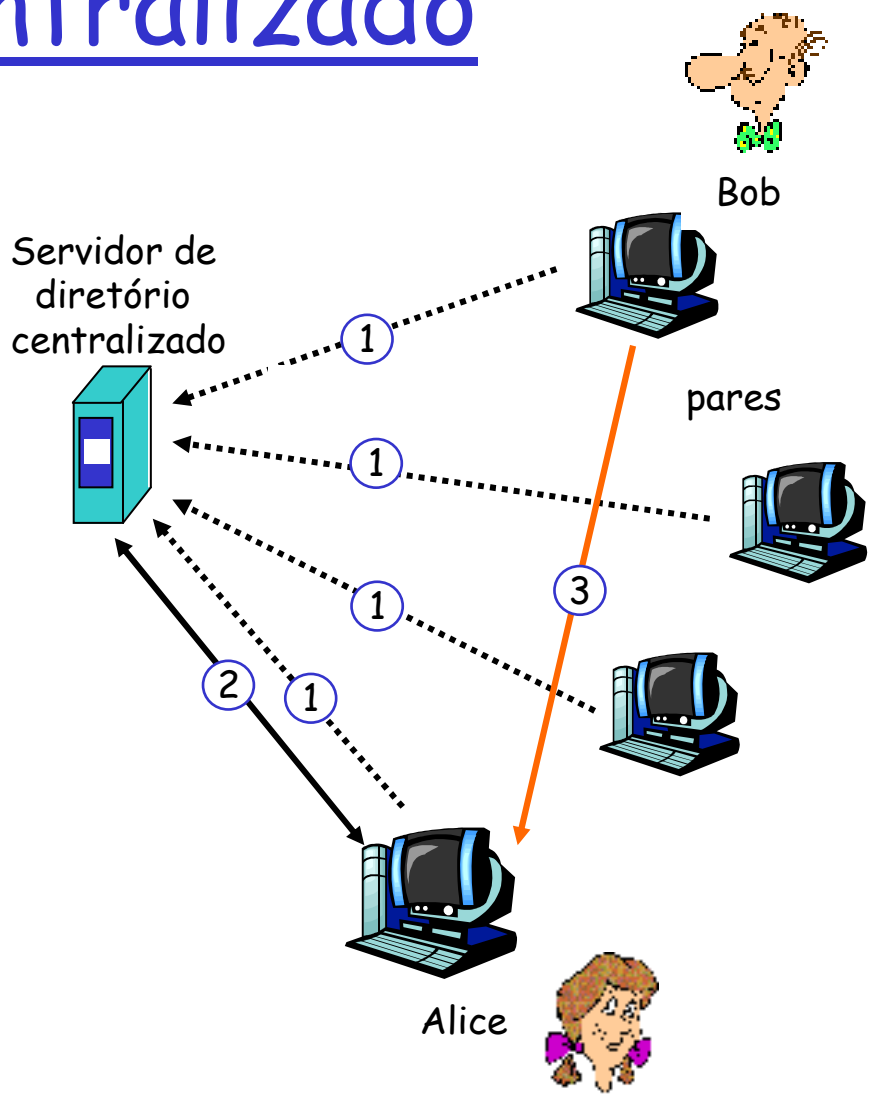
- Alice executa a aplicação cliente P2P no seu notebook
- Interminentemente conecta com a Internet; adquire um endereço IP para cada conexão;
- Requisita "Hey Jude"
- A aplicação apresenta vários nós que possuem uma cópia de "Hey Jude.
- Alice escolhe um dos nós, Bob.
- Arquivo é copiado do nó do Bob para o nó (notebook) da Alice: HTTP
- Enquanto Alice copia o arquivo do nó de Bob, outros usuários copiam os arquivos do nó da Alice;
- O nó daAlice é um cliente web como também um servidor web temporário.

Todos os nós são servidores = extremamente escalável!

P2P: diretório centralizado

"Napster" projeto original

1) Quando um dos pares se conecta, ele informa ao servidor central :
   - Endereço IP
   - conteúdo
2) Alice procura por "Hey Jude"
3) Alice requisita o arquivo de Bob



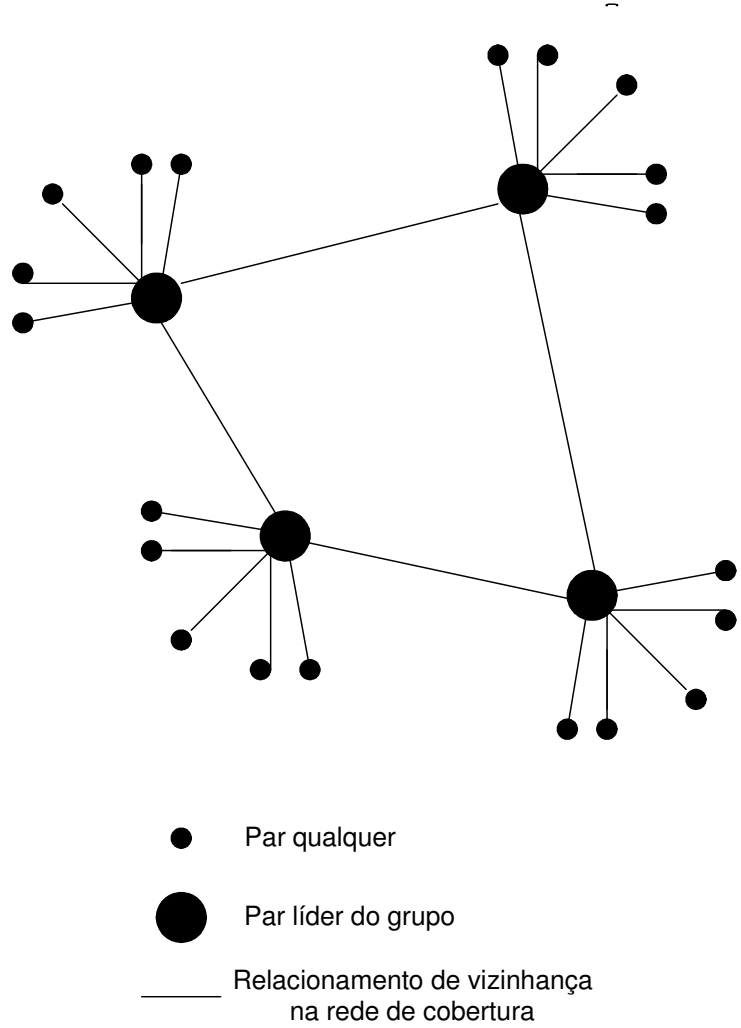
P2P: problemas com diretórios centralizados

- Único ponto de falha
- Gargalo de desempenho
- Infringe-se Copyright

transferência de arquivo é descentralizada, mas localizar conteúdo é totalmente descentralizada

P2P: diretório descentralizado

- Cada par ou é um líder de grupo ou pertence ao grupo de um líder;
- O líder do grupo localiza o conteúdo em todos os seus filhos;
- Os pares consultam o líder do grupo; o par líder pode consultar outros nós pares que também são líder;



Mais sobre diretório descentralizado

Rede de cobertura

- Os pares são nós
- Arestas entre os pares e o seu líder;
- Arestas entre alguns nós pares líderes de grupos;
- Vizinhos virtuais

Nó bootstrap

- O par conectado ou faz parte de um grupo de um líder ou é um par líder de grupo;

Vantagens da abordagem

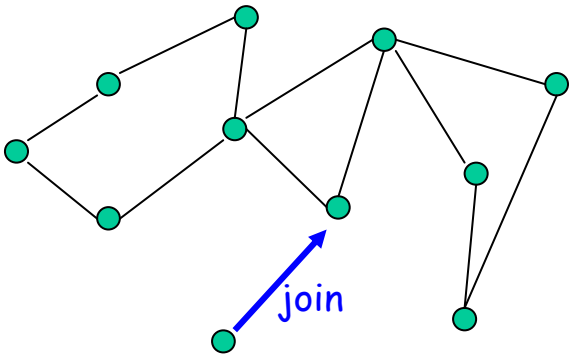
- Nenhum servidor centralizado;
  - O serviço de localização é distribuído entre os pares
  - Mais dificuldade de se ter falhas;

Desvantagem da abordagem

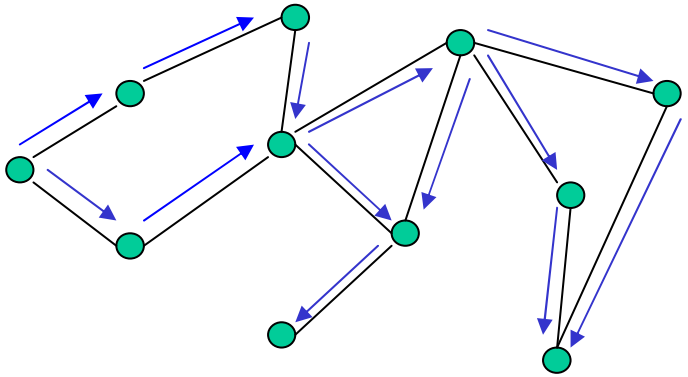
- Necessário nó bootstrap
- O líder do grupo pode ficar sobrecarregado;

P2P: fluxo de consultas (query flooding)

- Gnutella
- Sem hierarquia
- Mensagem *join*
- Usa o nó bootstrap para aprender sobre os outros



- Envia a "pergunta ou consulta"para os vizinhos;
- Vizinhos reencaminham as mensagens;
- Se o par consultado possui o objeto, envia uma mensagem de volta para o par originador da consulta;

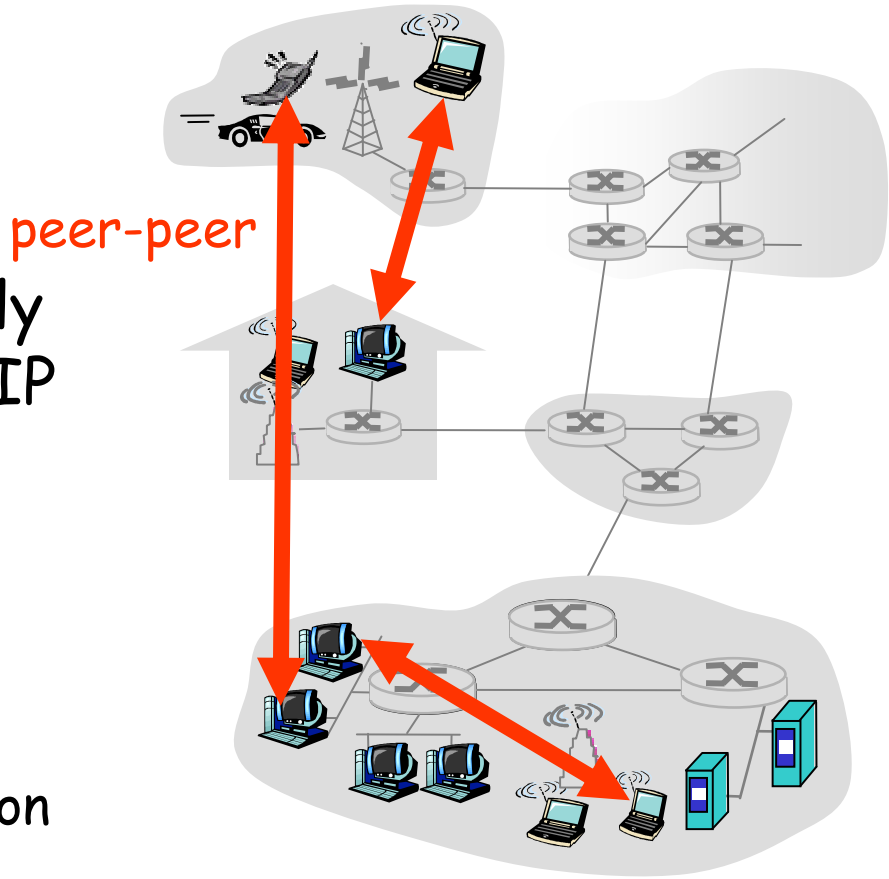


Chapter 2: Application layer

- 2.1 Principles of network applications
  - app architectures
  - app requirements
- 2.2 Web and HTTP
- 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

Pure P2P architecture

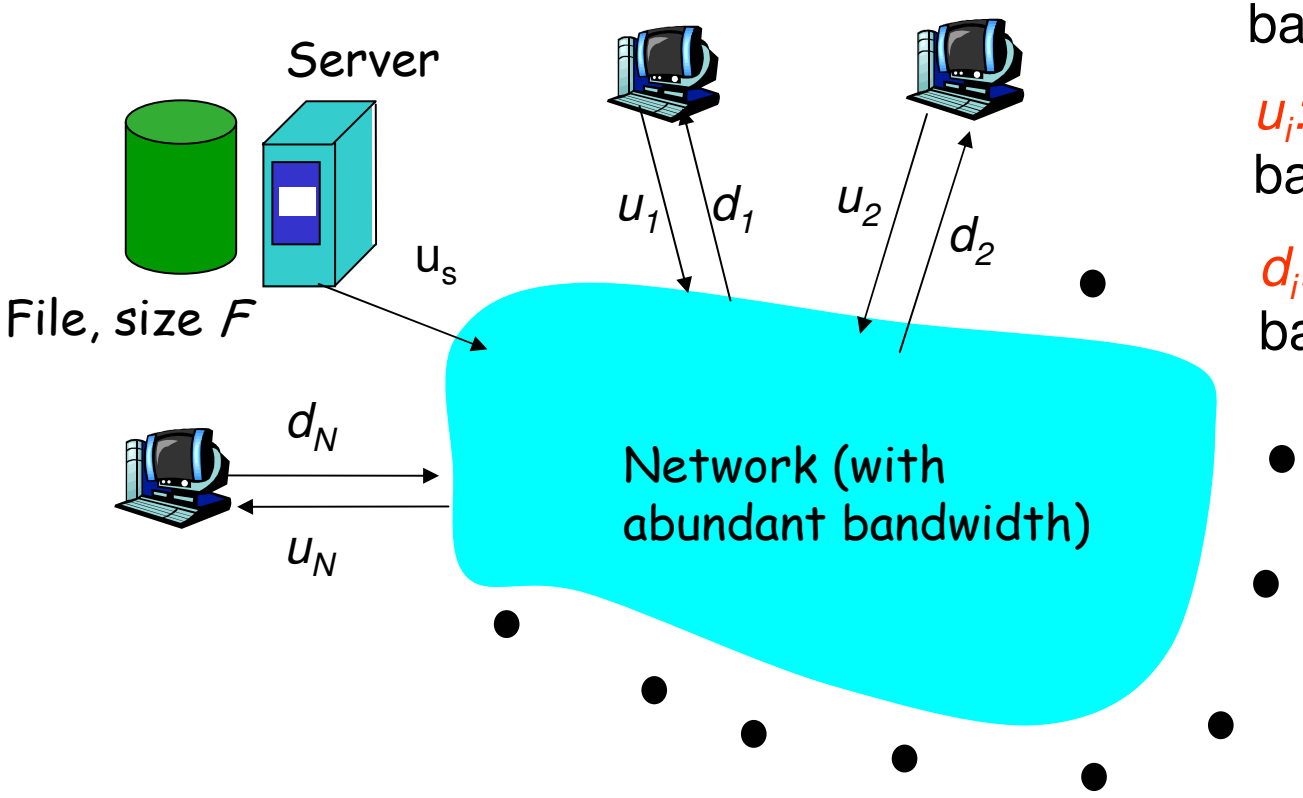
- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses



- Three topics:
  - File distribution
  - Searching for information
  - Case Study: Skype

File Distribution: Server-Client vs P2P

Question : How much time to distribute file from one server to *N peers*?



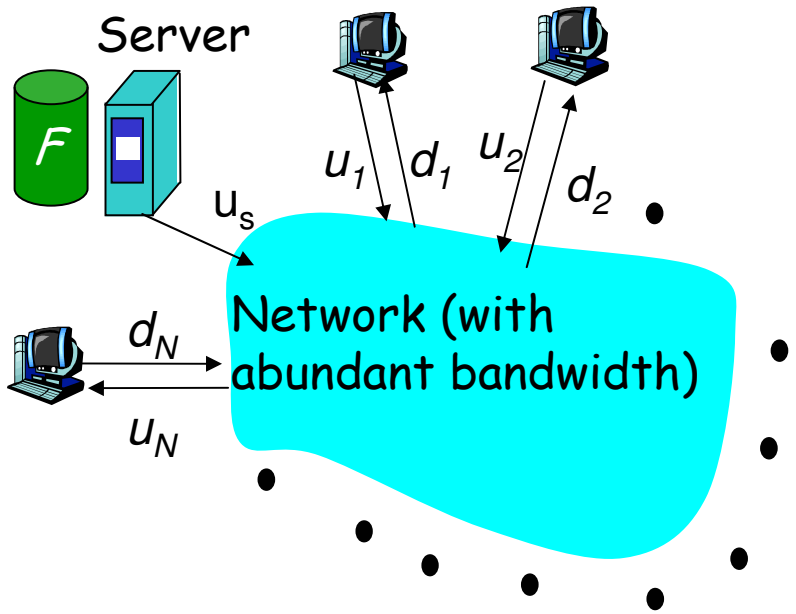
u_s: server upload bandwidth

u_i: peer i upload bandwidth

d_i: peer i download bandwidth

File distribution time: server-client

- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download

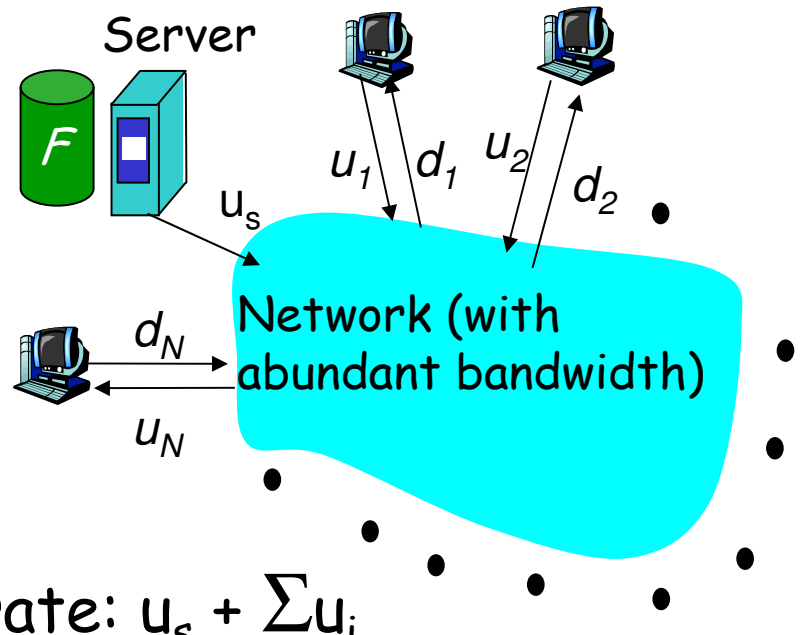


Time to distribute F to N clients using client/server approach $= d_{cs} = \max\{ NF/u_s, F/\min(d_i) \}$

increases linearly in N (for large N)

File distribution time: P2P

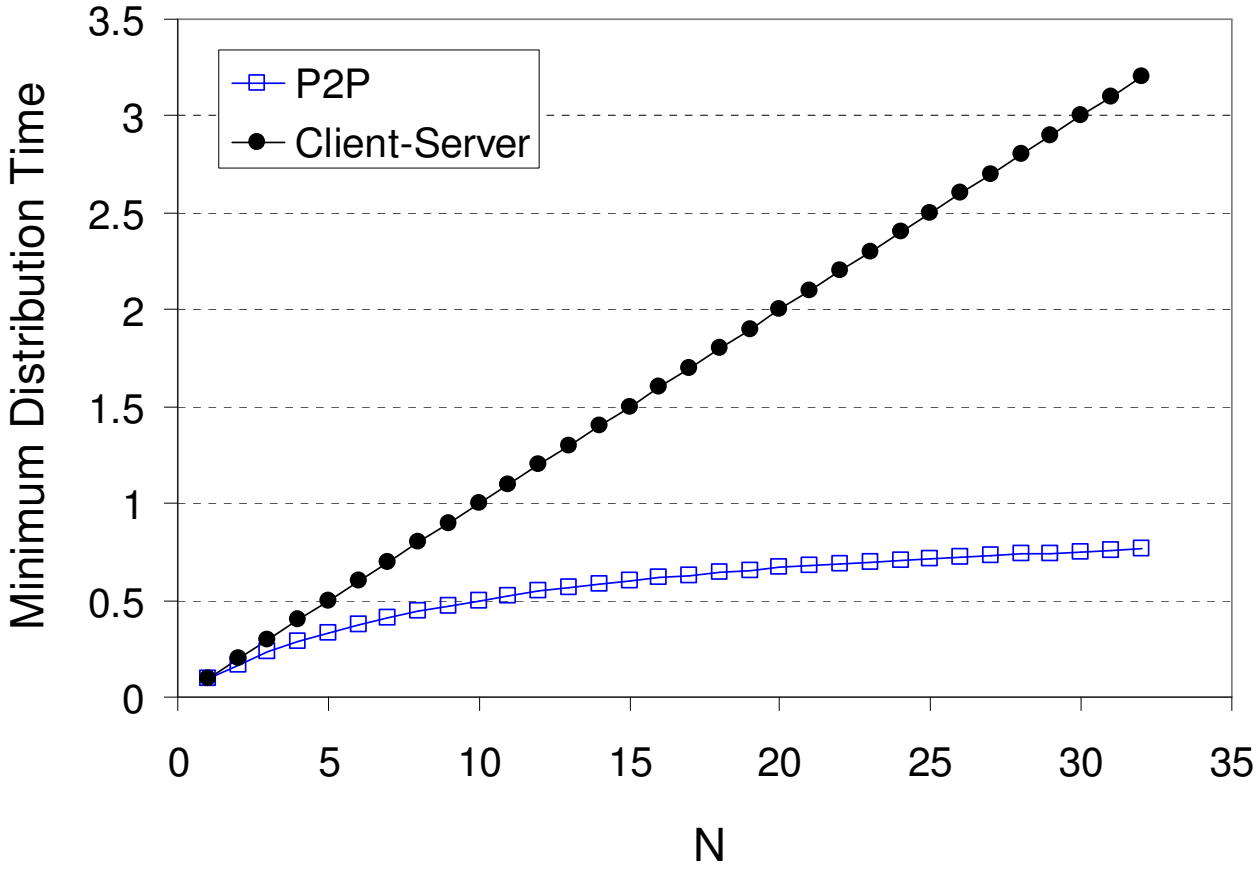
- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - fastest possible upload rate: $u_s + \Sigma u_i$



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

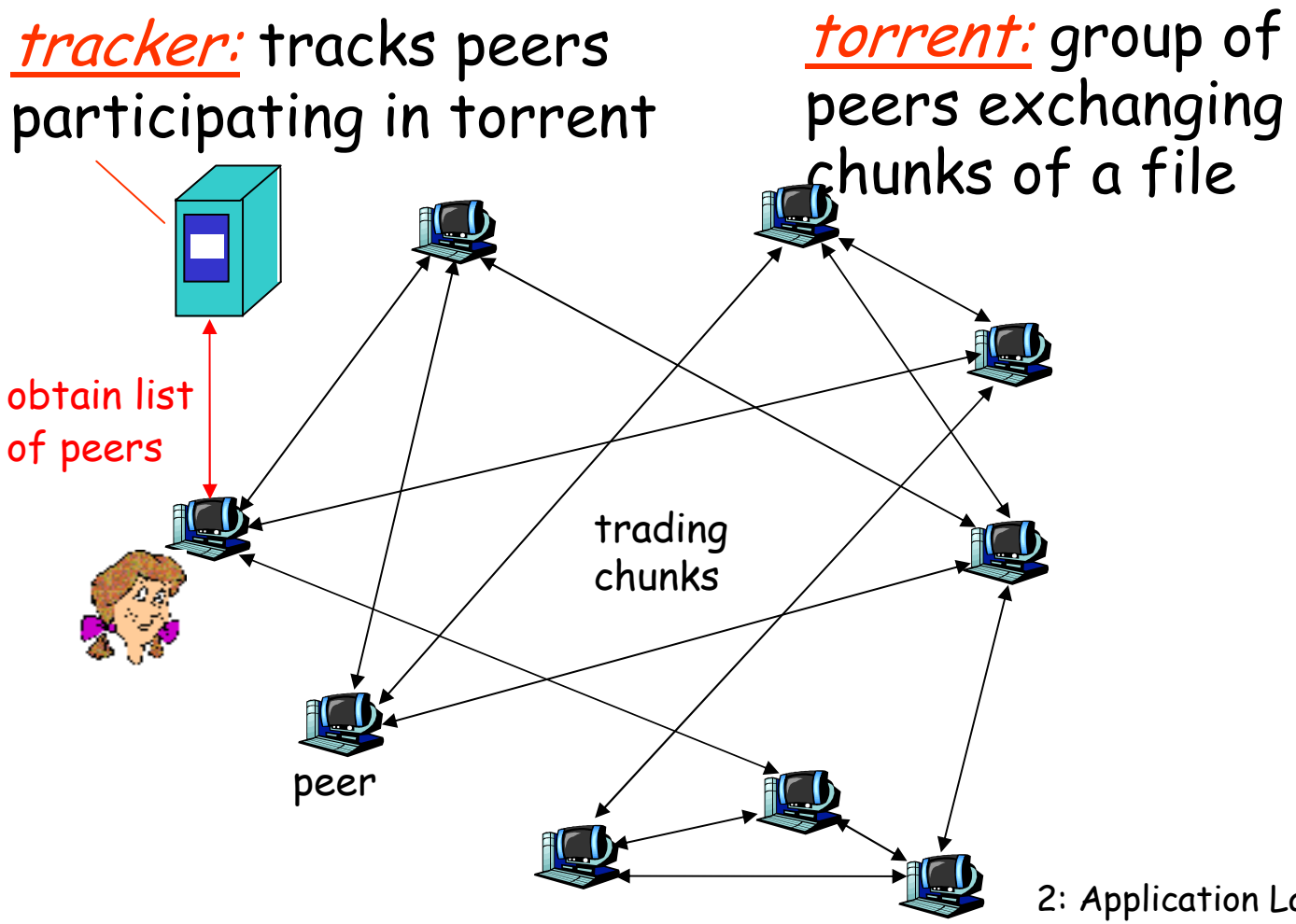
Server-client vs. P2P: example

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

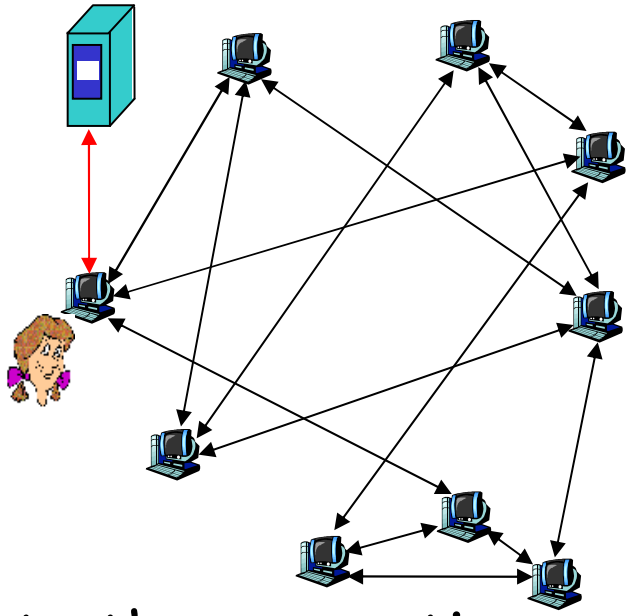


File distribution: BitTorrent

- P2P file distribution



BitTorrent (1)



- file divided into 256KB *chunks*.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers.
- peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain

BitTorrent (2)

Pulling Chunks

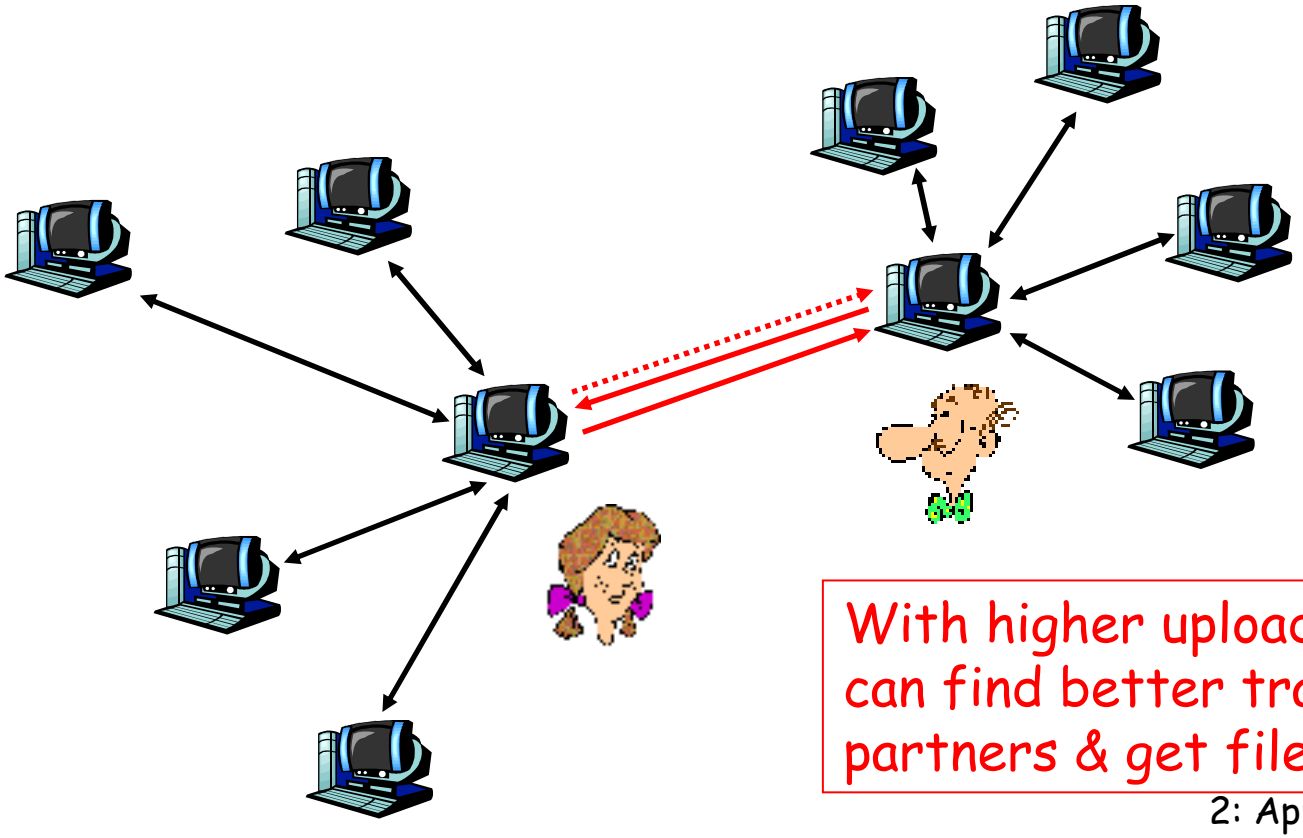
- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - rarest first

Sending Chunks: tit-for-tat

- Alice sends chunks to four neighbors currently sending her chunks *at the highest rate*
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - newly chosen peer may join top 4
  - "optimistically unchoke"

BitTorrent: Tit-for-tat

(1) Alice "optimistically unchokes" Bob
(2) Alice becomes one of Bob's top-four providers; Bob reciprocates
(3) Bob becomes one of Alice's top-four providers



With higher upload rate, can find better trading partners & get file faster!

Distributed Hash Table (DHT)

- DHT = distributed P2P database
- Database has (key, value) pairs;
  - key: ss number; value: human name
  - key: content type; value: IP address
- Peers query DB with key
  - DB returns values that match the key
- Peers can also insert (key, value) peers

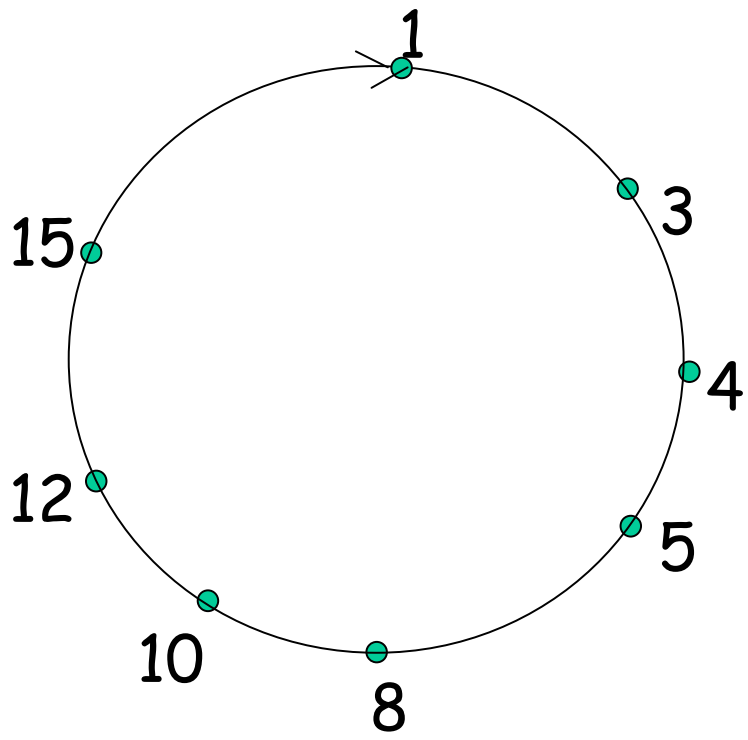
DHT Identifiers

- Assign integer identifier to each peer in range $[0,2^n-1]$.
  - Each identifier can be represented by n bits.
- Require each key to be an integer in same range.
- To get integer keys, hash original key.
  - eg, key = h("Led Zeppelin IV")
  - This is why they call it a distributed "hash" table

How to assign keys to peers?

- Central issue:
  - Assigning (key, value) pairs to peers.
- Rule: assign key to the peer that has the closest ID.
- Convention in lecture: closest is the immediate successor of the key.
- Ex: n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1

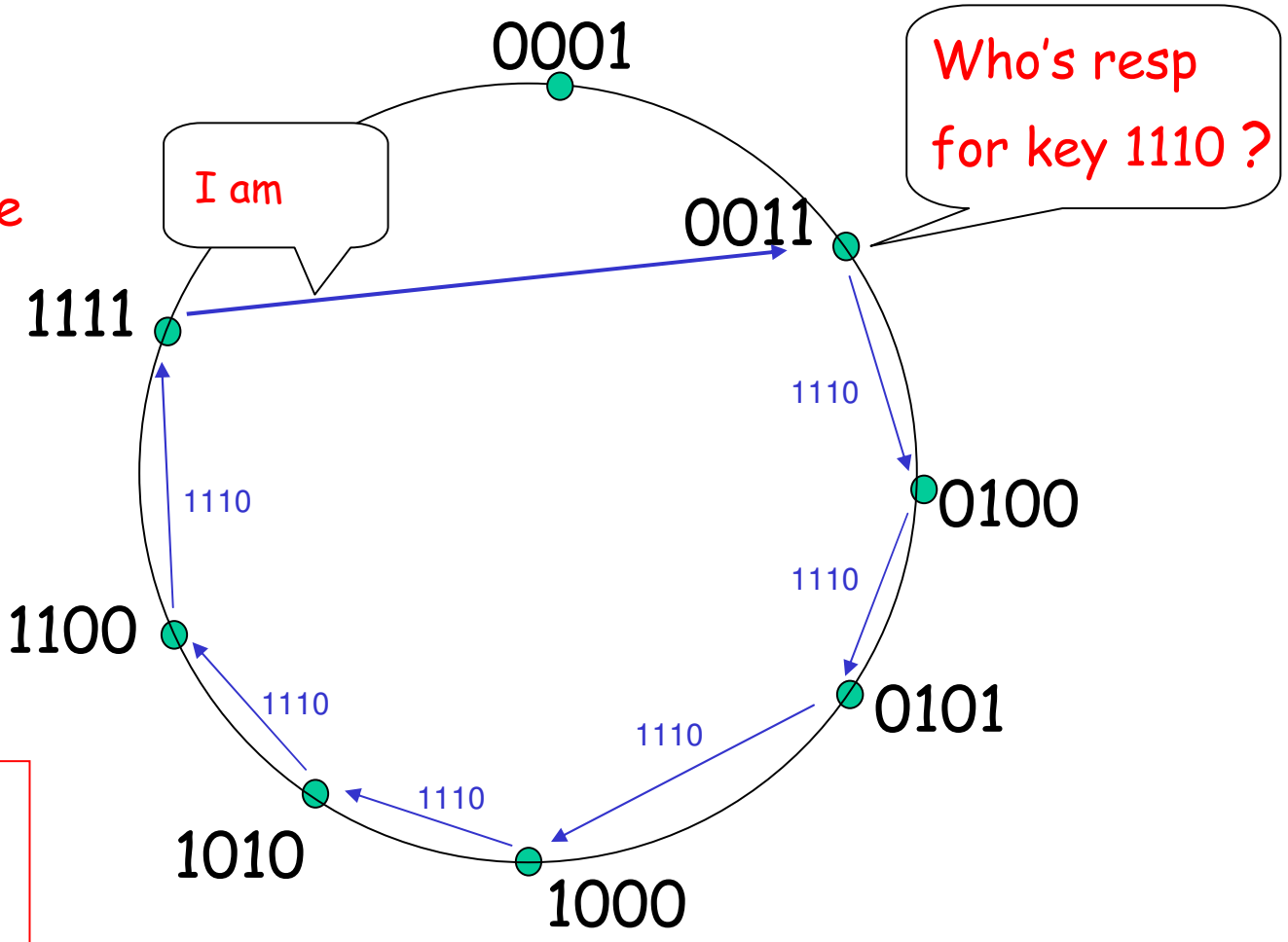
Circular DHT (1)



- Each peer *only* aware of immediate successor and predecessor.
- "Overlay network"

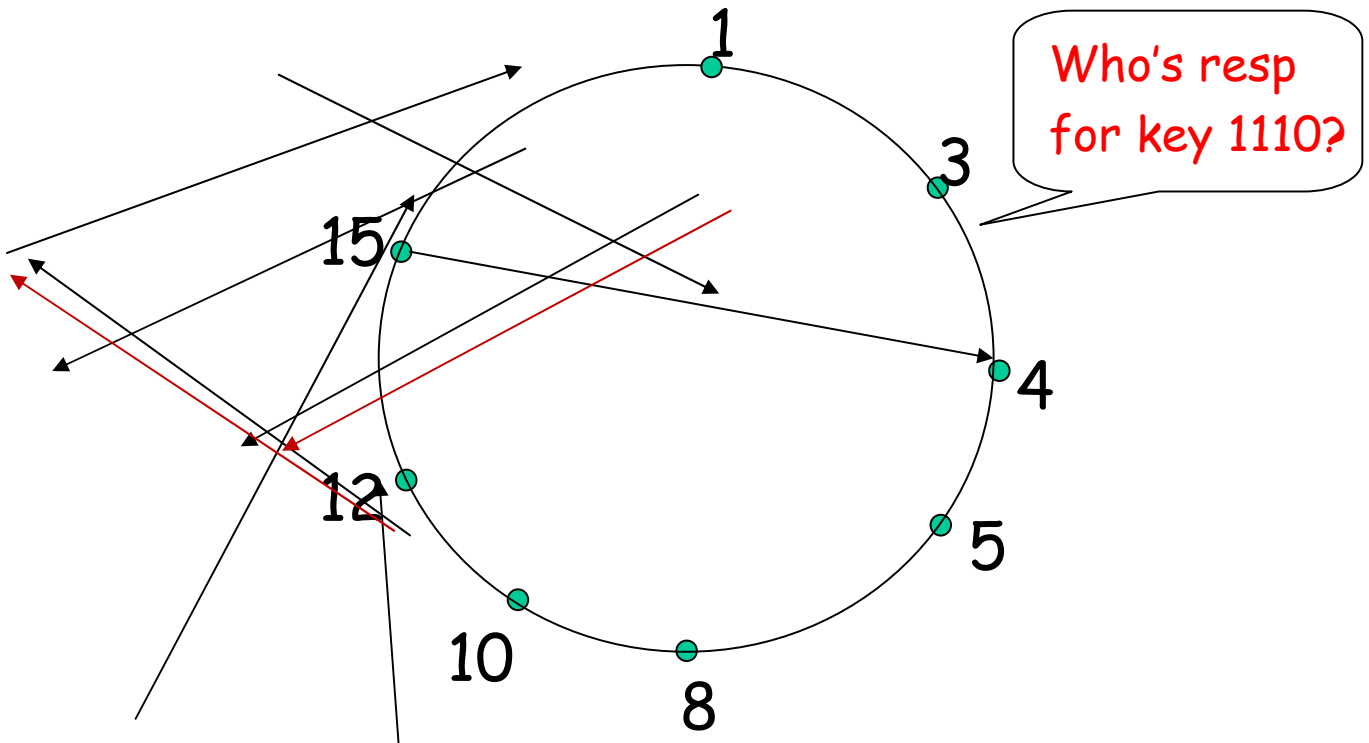
Circle DHT (2)

O(N) messages on avg to resolve query, when there are N peers



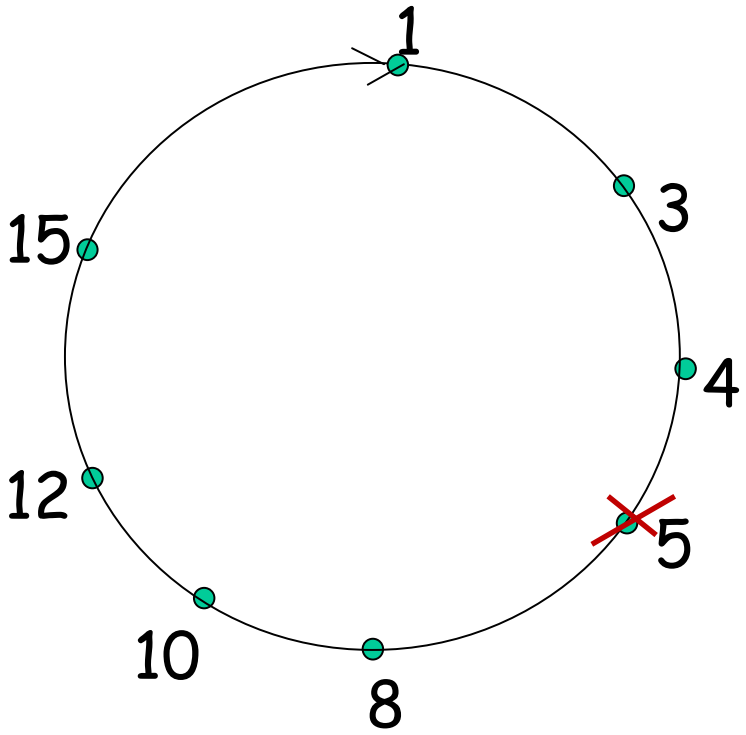
Define closest as closest successor

Circular DHT with Shortcuts



- Each peer keeps track of IP addresses of predecessor, successor, short cuts.
- Reduced from 6 to 2 messages.
- Possible to design shortcuts so O(log N) neighbors, O(log N) messages in query

Peer Churn

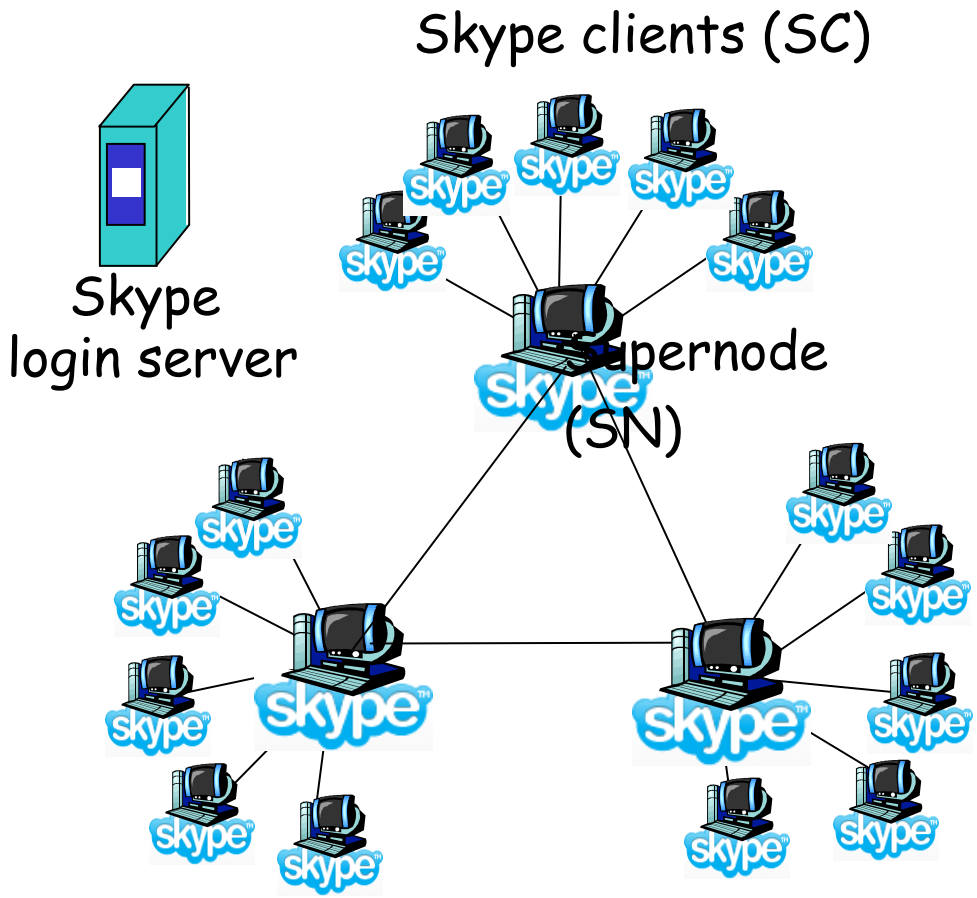


- To handle peer churn, require each peer to know the IP address of its two successors.
- Each peer periodically pings its two successors to see if they are still alive.

- Peer 5 abruptly leaves
- Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor.
- What if peer 13 wants to join?

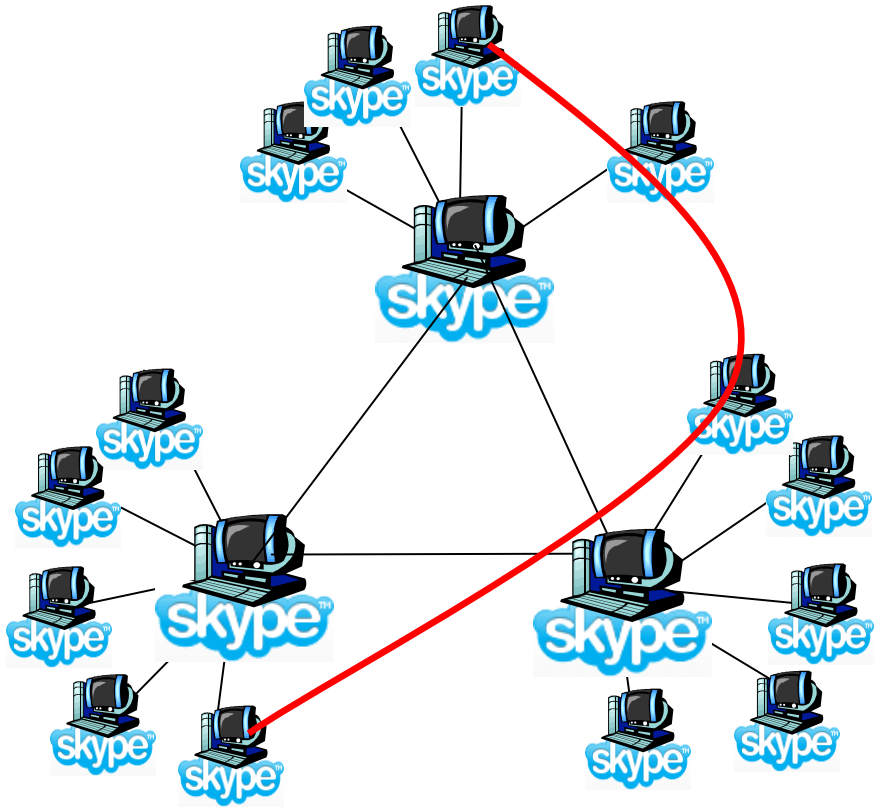
P2P Case study: Skype

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SNs
- Index maps usernames to IP addresses; distributed over SNs



Peers as relays

- Problem when both Alice and Bob are behind "NATs".
  - NAT prevents an outside peer from initiating a call to insider peer
- Solution:
  - Using Alice's and Bob's SNs, Relay is chosen
  - Each peer initiates session with relay.
  - Peers can now communicate through NATs via relay



P2P: mais sobre fluxo de consultas

Prós

- pares possuem responsabilidades semelhantes: não existem líderes de grupo;
- Extremamente descentralizado;
- Nenhum par mantem informações de diretório;

Contras

- Tráfico excessivo de consultas
- Raio da consulta: pode não ser o suficiente para obter o conteúdo, quando este existir;
- Manutenção de uma rede de cobertura;
- Necessário nó bootstrap

Capítulo 2: Resumo

Terminamos nosso estudo de aplicações de rede!

- Requisitos do serviço de aplicação:
  - confiabilidade, banda, retardo
- paradigma cliente-servidor
- modelo de serviço do transporte
  - orientado a conexão, confiável da Internet: TCP
  - não confiável, datagramas: UDP
- Protocolos específicos:
  - http
  - ftp
  - smtp, pop3, imap
  - dns
- programação c/ sockets
  - implementação cliente/servidor
  - usando sockets tcp, udp
- Distribuição de conteúdo:
  - caches, CDNs
  - P2P

Capítulo 2: Resumo

Mais importante: aprendemos sobre *protocolos*

- troca típica de mensagens pedido/resposta:
  - cliente solicita info ou serviço
  - servidor responde com dados, código de *status*
- formatos de mensagens:
  - cabeçalhos: campos com info sobre dados (metadados)
  - dados: info sendo comunicada

- msgs de controle X dados
  - na banda, fora da banda
- centralizado X descentralizado
- s/ estado X c/ estado
- transferência de msgs confiável X não confiável
- "complexidade na borda da rede"
- segurança: autenticação