

Capítulo 2: Camada de aplicação

- ❑ 2.1 Princípios de aplicações de rede
- ❑ 2.2 A Web e o HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Correio eletrônico
 - ❖ SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 Aplicações P2P
- ❑ 2.7 Programação de sockets com UDP
- ❑ 2.8 Programação de sockets com TCP

Capítulo 2: Camada de aplicação

Objetivos do capítulo:

- ❑ aspectos conceituais, de implementação de protocolos de aplicação de rede
 - ❖ modelos de serviço da camada de transporte
 - ❖ paradigma cliente-servidor
 - ❖ paradigma *peer-to-peer*
- ❑ aprenda sobre protocolos examinando protocolos populares em nível de aplicação
 - ❖ HTTP
 - ❖ FTP
 - ❖ SMTP/POP3/IMAP
 - ❖ DNS
- ❑ programando aplicações de rede
 - ❖ API socket

Algumas aplicações de rede

- ☐ e-mail
- ☐ web
- ☐ mensagem instantânea
- ☐ login remoto
- ☐ compartilhamento de arquivos P2P
- ☐ jogos em rede multiusuários
- ☐ clipes de vídeo armazenados em fluxo contínuo
- ☐ redes sociais
- ☐ voice over IP
- ☐ vídeoconferência em tempo real
- ☐ computação em grade

Criando uma aplicação de rede

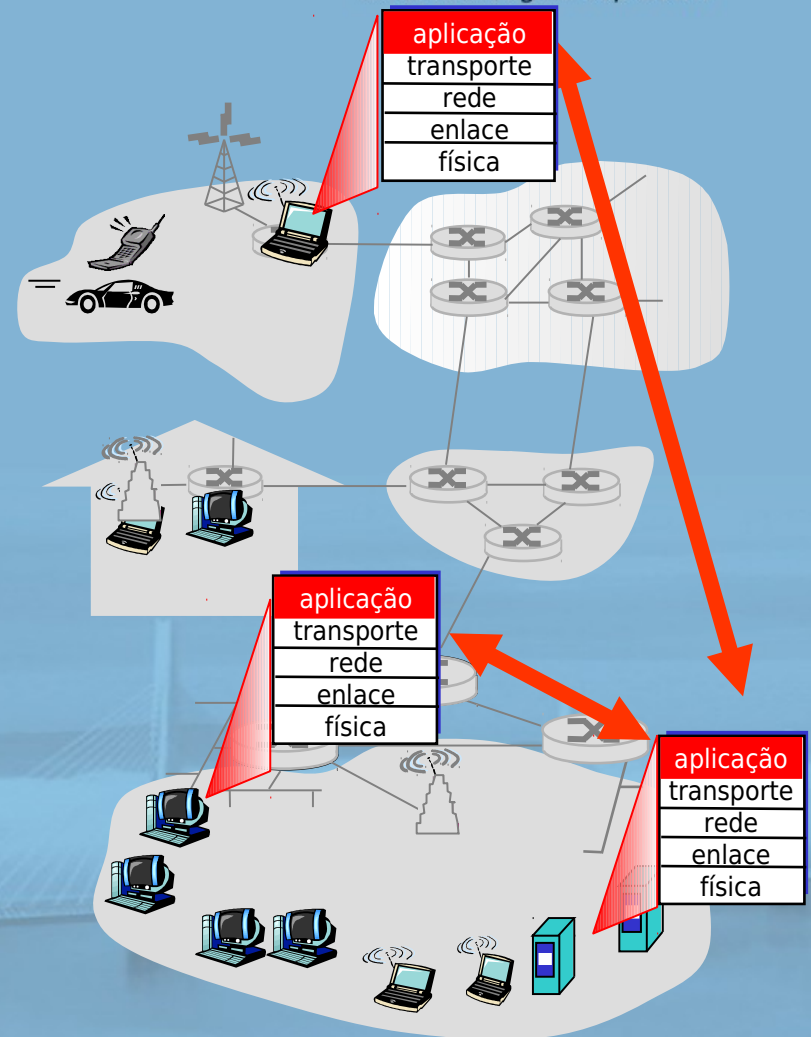
Escreva programas que

- ❖ executem em (diferentes) *sistemas finais*
- ❖ se comuniquem pela rede
- ❖ p. e., software de servidor Web se comunica com software de navegador Web

Não é preciso escrever software para dispositivos do núcleo da rede

- ❖ dispositivos do núcleo da rede não executam aplicações do usuário
- ❖ as aplicações nos sistemas finais permitem rápido desenvolvimento e propagação

Uma Abordagem Top-Down



Capítulo 2: Camada de aplicação

- ❑ 2.1 Princípios de aplicações de rede
- ❑ 2.2 A Web e o HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Correio eletrônico
 - ❖ SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 Aplicações P2P
- ❑ 2.7 Programação de sockets com UDP
- ❑ 2.8 Programação de sockets com TCP

Arquiteturas de aplicação

- ❑ Cliente-servidor
 - ❖ Incluindo centros de dados/cloud computing
- ❑ Peer-to-peer (P2P)
- ❑ Híbrida de cliente-servidor e P2P

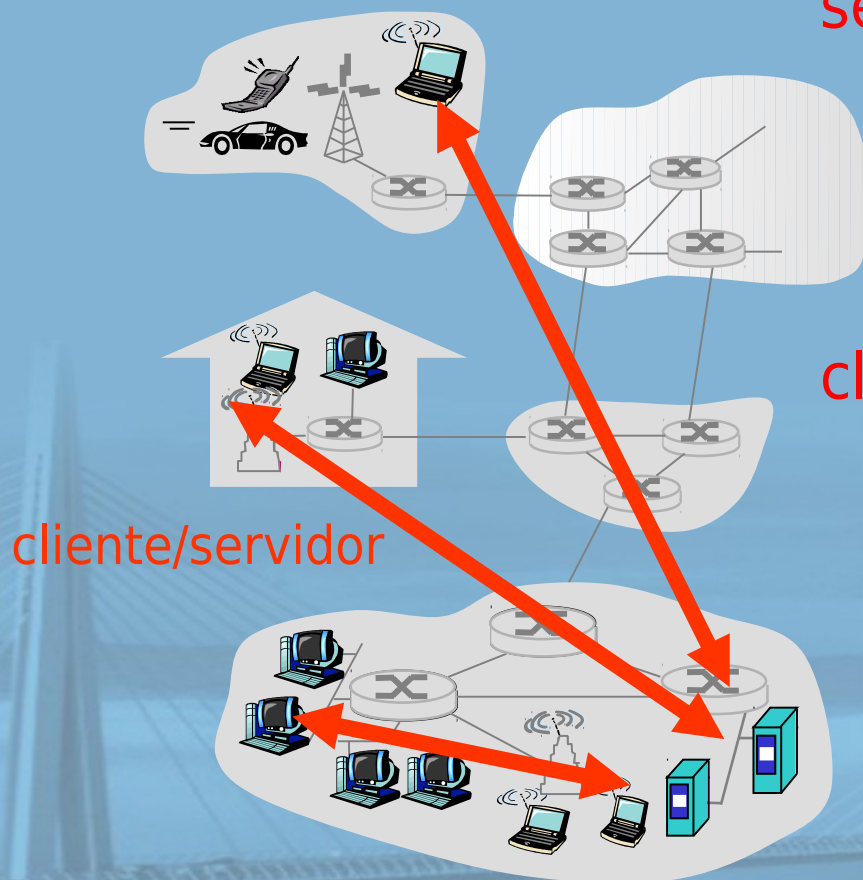
Arquitetura cliente-servidor

servidor:

- ❖ hospedeiro sempre ligado
- ❖ endereço IP permanente
- ❖ cluster de servidores

clientes:

- ❖ comunicam-se com o servidor
- ❖ podem estar conectados intermitentemente
- ❖ podem ter endereços IP dinâmicos
- ❖ não se comunicam diretamente entre si



Arquitetura P2P pura

- ❑ *nenhum* servidor sempre ligado
- ❑ sistemas finais arbitrários se comunicam diretamente
- ❑ pares são conectados intermitentemente e mudam endereços IP



altamente escalável, mas
difícil de administrar

Híbrido de cliente-servidor e P2P

Voice-over-IP

- ❖ aplicação P2P
- ❖ servidor centralizado: achando endereço da parte remota
- ❖ conexão cliente-cliente: direta (não através de servidor)

Mensagem instantânea

- ❖ bate-papo entre dois usuários é P2P
- ❖ serviço centralizado: detecção/localização da presença do cliente
 - usuário registra seu endereço IP com servidor central quando entra on-line
 - usuário contacta servidor central para descobrir endereços IP dos parceiros

Processos se comunicando

- processo:** programa rodando dentro de um hospedeiro
- no mesmo hospedeiro, dois processos se comunicam usando a **comunicação entre processos** (definida pelo SO).
 - processos em hospedeiros diferentes se comunicam trocando **mensagens**

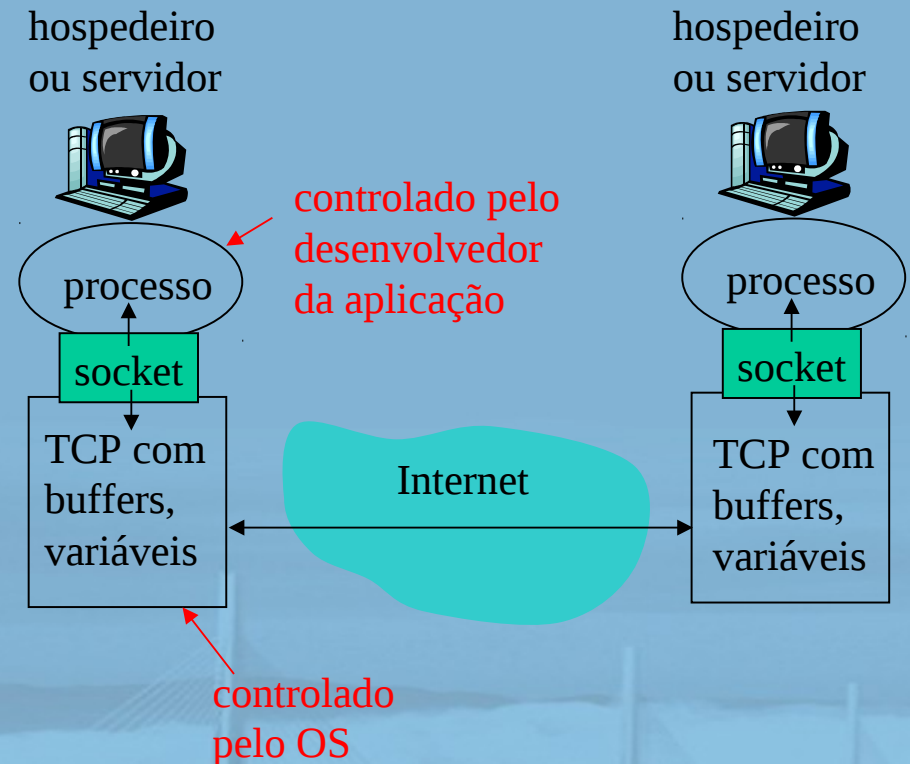
processo cliente: processo que inicia a comunicação

processo servidor: processo que espera para ser contactado

- Nota: aplicações com arquiteturas P2P têm processos clientes & processos servidores

Sockets

- ❑ processo envia/recebe mensagens de/para seu **socket**
- ❑ socket semelhante à porta
 - ❖ processo enviando empurra mensagem pela porta
 - ❖ processo enviando conta com infraestrutura de transporte no outro lado da porta, que leva a mensagem ao socket no processo receptor
- ❑ API: (1) escolha do protocolo de transporte; (2) determinar parâmetros de conexão



Endereçando processos

- ❑ para receber mensagens, processo deve ter *identificador*
- ❑ dispositivo hospedeiro tem endereço IP exclusivo de 32 bits
- ❑ exercício: use ifconfig do comando prompt para obter seu endereço IP (ipconfig no windows)
- ❑ P: Basta o endereço IP do hospedeiro em que o processo é executado para identificar o processo?
- ❖ R: Não, *muitos* processos podem estar rodando no mesmo hospedeiro
- ❑ *Identificador* inclui **endereço IP** e **números de porta** associados ao processo no hospedeiro.
- ❑ Exemplos de número de porta:
 - ❖ servidor HTTP: 80
 - ❖ servidor de correio: 25

Definições de protocolo da camada de aplicação

- tipos de mensagens trocadas,
 - ❖ p. e., requisição, resposta
- sintaxe da mensagem:
 - ❖ que campos nas mensagens & como os campos são delineados
- semântica da mensagem
 - ❖ significado da informação nos campos
- regras de quando e como processos enviam & respondem a mensagens

protocolos de domínio público:

- definidos em RFCs
- provê interoperabilidade
- p. e., HTTP, SMTP, BitTorrent

protocolos proprietários:

- p. e., Skype, ppstream

Que serviço de transporte uma aplicação precisa?

perda de dados

- algumas (p. e., áudio) podem tolerar alguma perda
- outras (p. e., transferência de arquivos, telnet) exigem transferência de dados 100% confiável

temporização

- algumas (p. e., telefonia na Internet, jogos interativos) exigem pouco atraso para serem “eficazes”

vazão

- algumas (p. e., multimídia) exigem um mínimo de vazão para serem “eficazes”
- outras (“elásticas”) utilizam qualquer vazão que receberem

segurança

- criptografia, integridade de dados,...

Requisitos de serviço de transporte das aplicações comuns

Aplicação	Perda de dados	Vazão	Sensível ao tempo
transf. arquivos	sem perda	elástica	não
e-mail	sem perda	elástica	não
documentos Web	sem perda	elástica	não
áudio/vídeo tempo real	tolerante a perda	áudio: 5 kbps-1 Mbps vídeo: 10 kbps-5 Mbps	sim, centenas de ms
áudio/vídeo armazenado	tolerante a perda	o mesmo que antes	sim, alguns seg
jogos interativos	tolerante a perda	poucos kbps ou mais	sim, centenas de ms
Mensagem instantânea	sem perda	elástica	centenas de ms a alguns seg

Serviços de protocolos de transporte da Internet

serviço TCP:

- ❑ *orientado a conexão:* preparação exigida entre processos cliente e servidor
- ❑ *transporte confiável* entre processo emissor e receptor
- ❑ *controle de fluxo:* emissor não sobrecarrega receptor
- ❑ *controle de congestionamento:* regula emissor quando a rede está sobrecarregada
- ❑ *não oferece:* temporização, garantias mínimas de vazão, segurança

serviço UDP:

- ❑ transferência de dados não confiável entre processo emissor e receptor
 - ❑ não oferece: preparação da conexão, confiabilidade, controle de fluxo, controle de congest., temporização, garantia de vazão ou segurança
- P: por que se incomodar? Por que existe um UDP?

Aplicações da Internet: aplicação, protocolos de transporte

Aplicação	Protocolo da camada de aplicação	Protocolo de transporte básico
e-mail	SMTP [RFC 2821]	TCP
acesso remoto	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
transf. arquivos	FTP [RFC 959]	TCP
multimídia com fluxo contínuo	HTTP (p. e., Youtube), RTP [RFC 1889]	TCP ou UDP
telefonia da Internet	SIP, RTP, proprietário (p. e., Skype)	normalmente UDP

Capítulo 2: Camada de aplicação

- ❑ 2.1 Princípios de aplicações de rede
- ❑ 2.2 A Web e o HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Correio eletrônico
 - ❖ SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 Aplicações P2P
- ❑ 2.7 Programação de sockets com UDP
- ❑ 2.8 Programação de sockets com TCP

Programação de sockets

Objetivo: aprender a criar aplicação cliente-servidor que se comunica usando sockets

API socket

- ❑ introduzida no BSD4.1 UNIX em 1981
- ❑ criada, usada e liberada explicitamente pelas apls.
- ❑ paradigma cliente-servidor
- ❑ dois tipos de serviços de transporte por meio da API socket:
 - ❖ UDP
 - ❖ TCP

socket

Uma interface *criada pela aplicação e controlada pelo SO* (uma “porta”) na qual o processo da aplicação pode *enviar e receber* mensagens para/de outro processo da aplicação

Fundamentos de programação de socket

- ❑ servidor deve estar rodando antes que o cliente possa lhe enviar algo
- ❑ servidor deve ter um socket (porta) pelo qual recebe e envia segmentos
- ❑ da mesma forma, o cliente precisa de um socket
- ❑ socket é identificado localmente com um número de porta
 - ❖ semelhante ao número de apartamento de um prédio
- ❑ cliente precisa saber o endereço IP do servidor e o número de porta do socket

Programação de socket com UDP

UDP: sem “conexão” entre
cliente e servidor

- sem “handshaking”
- emissor conecta de forma explícita endereço IP e porta do destino a cada segmento
- SO conecta endereço IP e porta do socket emissor a cada segmento
- Servidor pode extrair endereço IP, porta do emissor a partir do segmento recebido

ponto de vista da aplicação

UDP oferece transferência não confiável de grupos de bytes (“datagramas”) entre cliente e servidor

Nota: A terminologia oficial para um pacote UDP é “datagrama”. Nesta aula, usamos “segmento UDP” em seu lugar.

Exemplo em curso

❑ cliente:

- ❖ usuário digita linha de texto
- ❖ programa cliente envia linha ao servidor

❑ servidor:

- ❖ servidor recebe linha de texto
- ❖ coloca todas as letras em maiúsculas
- ❖ envia linha modificada ao cliente

❑ cliente:

- ❖ recebe linha de texto
- ❖ apresenta

Interação de socket cliente/servidor: UDP

servidor (rodando em **hostid**)

cliente

create socket,
port = x.
**serverSocket =
DatagramSocket(x)**

↓
lê datagrama de
serverSocket

↓
escreve resposta
em **serverSocket**
indicando endereço
do cliente, número de
porta

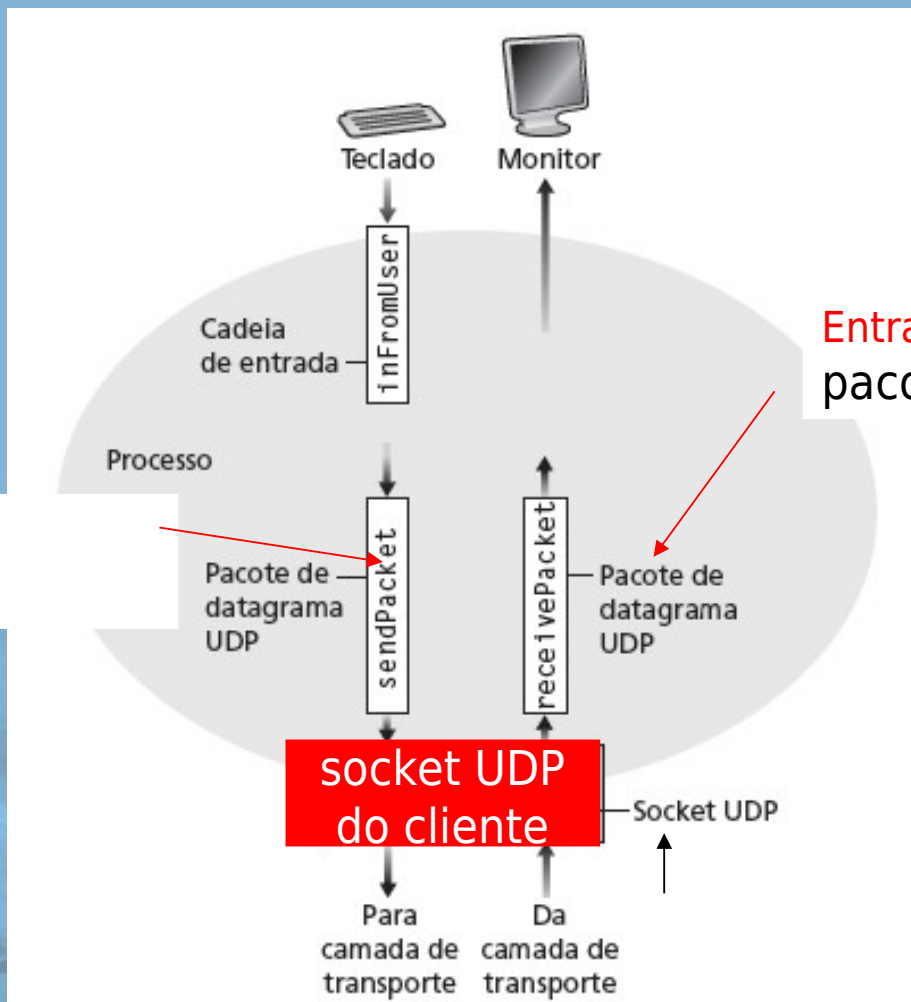
create socket,
**clientSocket =
DatagramSocket()**

↓
Cria datagrama com IP do
servidor e port = x; envia datagrama
por **clientSocket**

↓
lê datagrama de
clientSocket

↓
fecha
clientSocket

Exemplo: cliente Java (UDP)



Entrada: recebe
pacote

Saída: envia
pacote

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

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

cria cadeia
de entrada

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

cria socket
do cliente

```
        DatagramSocket clientSocket = new DatagramSocket();
```

traduz hostname
para endereço IP
usando DNS

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

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

```
        String sentence = inFromUser.readLine();  
        sendData = sentence.getBytes();
```

cria datagrama com
dados a enviar
tamanho, end. IP
porta

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

envia datagrama
ao servidor

```
clientSocket.send(sendPacket);
```

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

lê datagrama
do servidor

```
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  
    {
```

cria socket
de datagrama
na porta 9876

```
        DatagramSocket serverSocket = new DatagramSocket(9876);
```

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

```
        while(true)  
        {
```

cria espaço para
datagrama recebido

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

recebe
datagrama

```
            serverSocket.receive(receivePacket);
```

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

obtem end. IP
porta do
emissor

```
InetAddress IPAddress = receivePacket.getAddress();
```

```
int port = receivePacket.getPort();
```

```
String capitalizedSentence = sentence.toUpperCase();
```

```
sendData = capitalizedSentence.getBytes();
```

cria datagrama p/
enviar ao cliente

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

escreve
datagrama
no socket

```
serverSocket.send(sendPacket);
```

```
}  
}  
}
```

fim do loop while,
retorna e espera
outro datagrama

Observações e perguntas sobre UDP

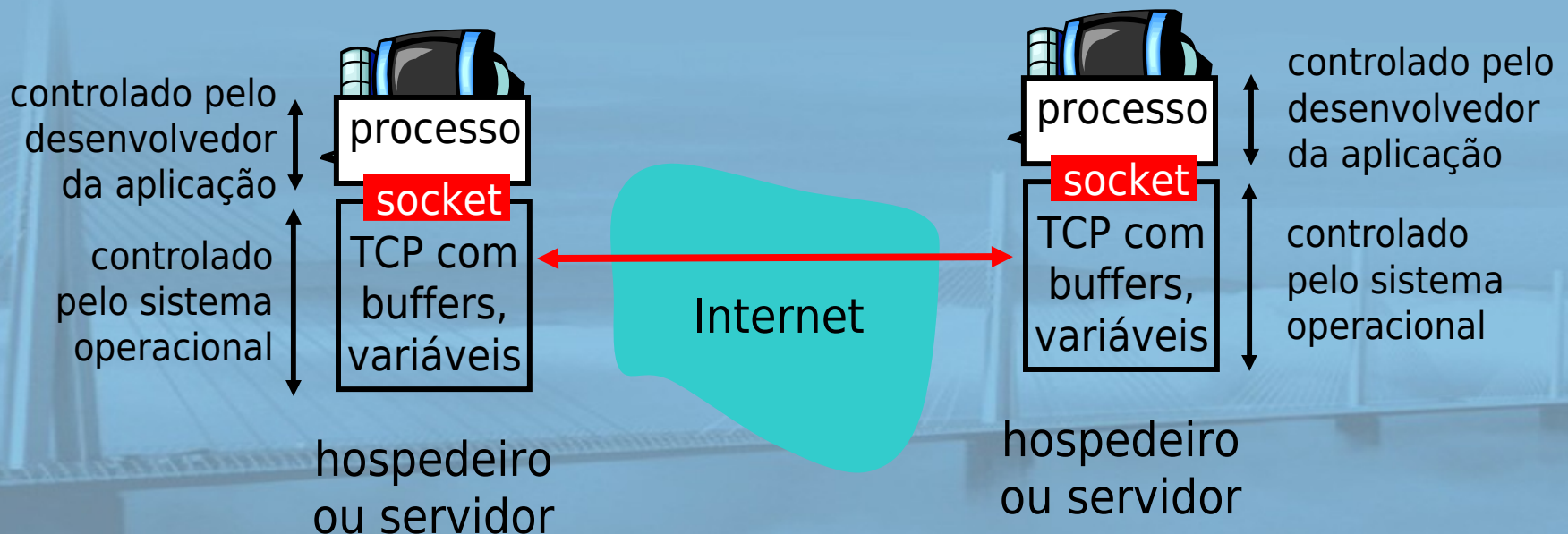
- ❑ cliente e servidor usam DatagramSocket
- ❑ IP e porta de destino são explicitamente conectados ao segmento.
- ❑ O cliente pode enviar um segmento ao servidor sem saber o endereço IP e/ou número de porta do servidor?
- ❑ Múltiplos clientes podem usar o servidor?

Capítulo 2: Camada de aplicação

- ❑ 2.1 Princípios de aplicações de rede
- ❑ 2.2 A Web e o HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Correio eletrônico
 - ❖ SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 Aplicações P2P
- ❑ 2.7 Programação de sockets com UDP
- ❑ 2.8 Programação de sockets com TCP

Programação de socket usando TCP

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



Programação de socket com TCP

cliente deve contactar servidor

- processo servidor primeiro deve estar rodando
- servidor deve ter criado socket (porta) que aceita contato do cliente

cliente contacta servidor:

- criando socket TCP local ao cliente
- especificando endereço IP, # porta do processo servidor
- quando **cliente cria socket**: cliente TCP estabelece conexão com servidor TCP

- quando contactado pelo cliente, **servidor TCP cria novo socket** para processo servidor se comunicar com cliente
 - ❖ permite que servidor fale com múltiplos clientes
 - ❖ números de porta de origem usados para distinguir clientes (mais no Cap. 3)

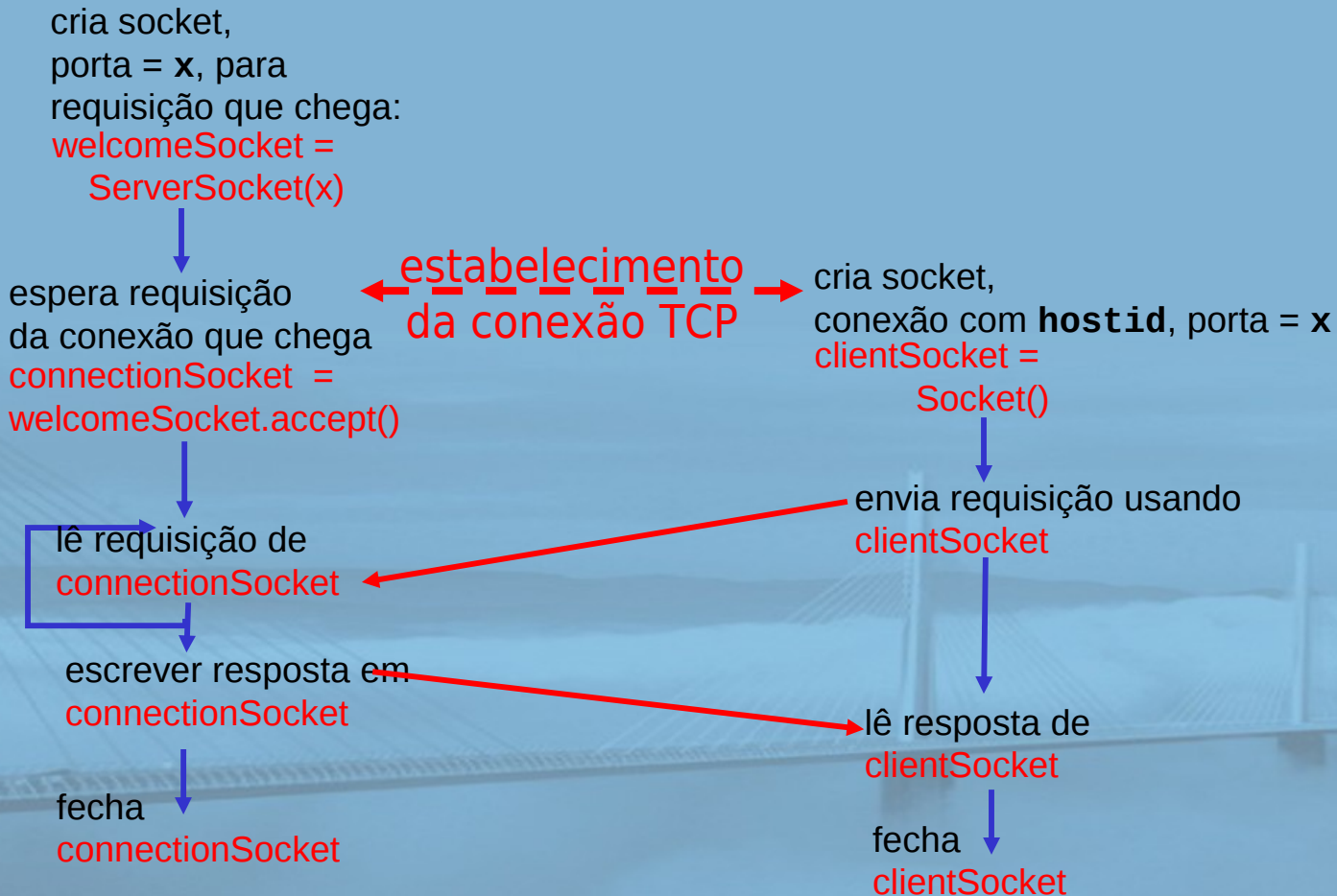
ponto de vista da aplicação

TCP oferece transferência de bytes confiável, em ordem ("pipe") entre cliente e servidor

Interação de socket cliente/servidor: TCP

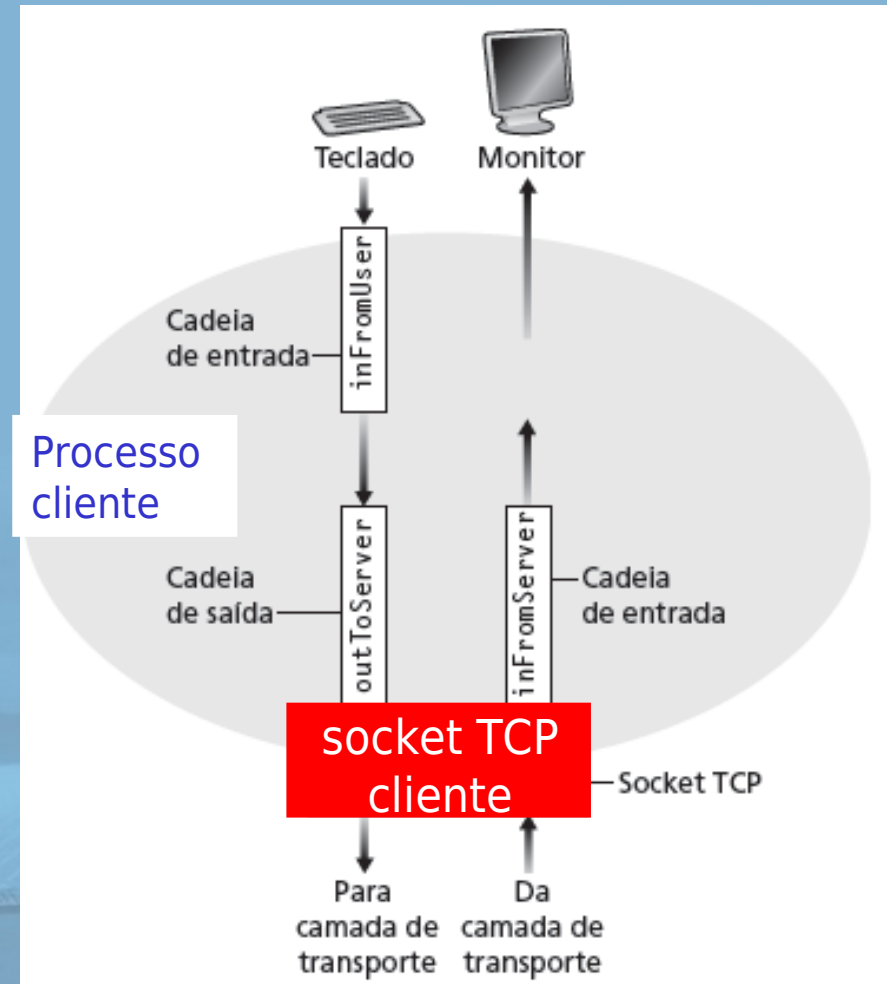
servidor (rodando em **hostid**)

Cliente



Jargão de cadeia

- uma **cadeia** é uma sequência de caracteres que flui para dentro ou fora de um processo.
- uma **cadeia de entrada** está conectada a uma fonte de entrada para o processo, p. e., teclado ou socket.
- uma **cadeia de saída** está conectada a uma fonte de saída, p. e., monitor ou socket.



Programação de socket com TCP

Exemplo de apl. cliente-servidor:

- 1) cliente lê linha da entrada padrão (cadeia **inFromUser**), envia ao servidor via socket (cadeia **outToServer**)
- 2) servidor lê linha do socket
- 3) servidor converte linha para maiúsculas, envia de volta ao cliente
- 4) cliente lê, imprime linha modificada do socket (cadeia **inFromServer**)

Exemplo: cliente Java (TCP)

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

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

cria cadeia
de entrada

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

cria socket
cliente, conexão
com servidor

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

cria cadeia de
entrada conectada
ao socket

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

cria cadeia de
saída conectada
ao socket

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

```
sentence = inFromUser.readLine();
```

envia linha
ao servidor

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

```
modifiedSentence = inFromServer.readLine();
```

lê linha
do servidor

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

```
clientSocket.close();
```

```
}  
}
```

Exemplo: servidor Java (TCP)

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

```
class TCPServer {
```

```
    public static void main(String argv[]) throws Exception  
    {
```

```
        String clientSentence;  
        String capitalizedSentence;
```

cria socket de
apresentação na
porta 6789

```
        ServerSocket welcomeSocket = new ServerSocket(6789);
```

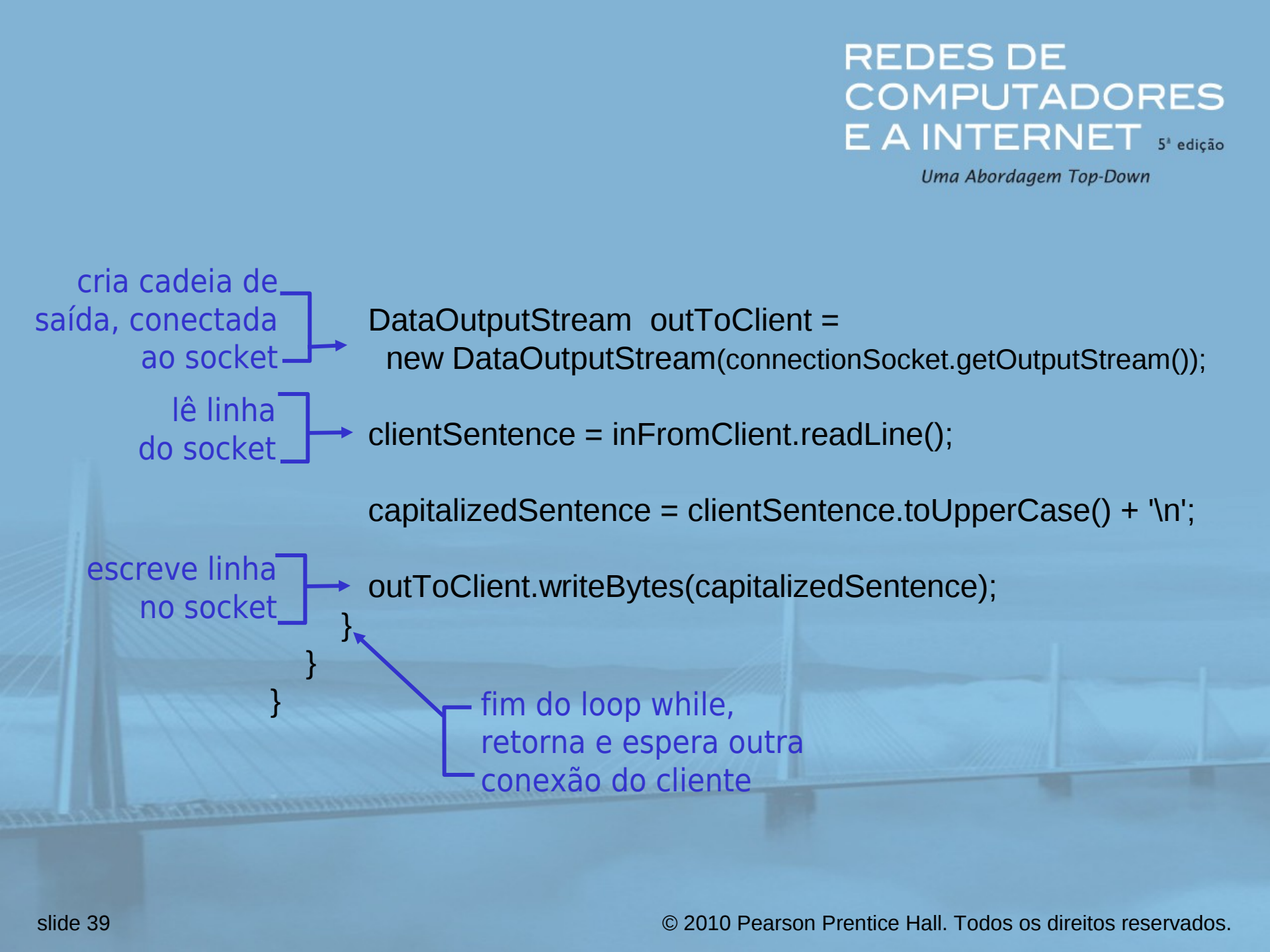
```
        while(true) {
```

espera no socket
de apresentação pelo
contato do cliente

```
            Socket connectionSocket = welcomeSocket.accept();
```

cria cadeia de
entrada, conectada
ao socket

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

```

    cria cadeia de
saída, conectada
ao socket      DataOutputStream outToClient =
                new DataOutputStream(connectionSocket.getOutputStream());

    lê linha
do socket      clientSentence = inFromClient.readLine();

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

    escreve linha
no socket      outToClient.writeBytes(capitalizedSentence);
                }
            }
        }
    fim do loop while,
    retorna e espera outra
    conexão do cliente

```

The diagram shows a bridge in the background. The code is annotated with blue text and arrows. The first annotation 'cria cadeia de saída, conectada ao socket' points to the `DataOutputStream` creation. The second annotation 'lê linha do socket' points to the `readLine()` call. The third annotation 'escreve linha no socket' points to the `writeBytes()` call. The fourth annotation 'fim do loop while, retorna e espera outra conexão do cliente' points to the closing brace of the `while` loop.

TCP – observações e perguntas

- ❑ servidor tem dois tipos de sockets:
 - ❖ ServerSocket e connectionSocket
- ❑ quando o cliente bate na “porta” de serverSocket, servidor cria connectionSocket e completa conexão TCP.
- ❑ IP de destino e porta não são explicitamente conectados ao segmento.
- ❑ Múltiplos clientes podem usar o servidor?

Capítulo 2: Camada de aplicação

- ❑ 2.1 Princípios de aplicações de rede
- ❑ 2.2 A Web e o HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Correio eletrônico
 - ❖ SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 Aplicações P2P
- ❑ 2.7 Programação de sockets com UDP
- ❑ 2.8 Programação de sockets com TCP

Web e HTTP

primeiro, algum jargão

- ❑ página Web consiste em objetos
- ❑ objeto pode ser arquivo HTML (Hypertext Markup Language), imagem JPEG, applet Java, arquivo de áudio,...
- ❑ página Web consiste em arquivo HTML básico que inclui vários objetos referenciados
- ❑ cada objeto é endereçável por um URL
- ❑ exemplo de URL (Uniform Resource Locator):

www.someschool.edu / someDept/pic.gif

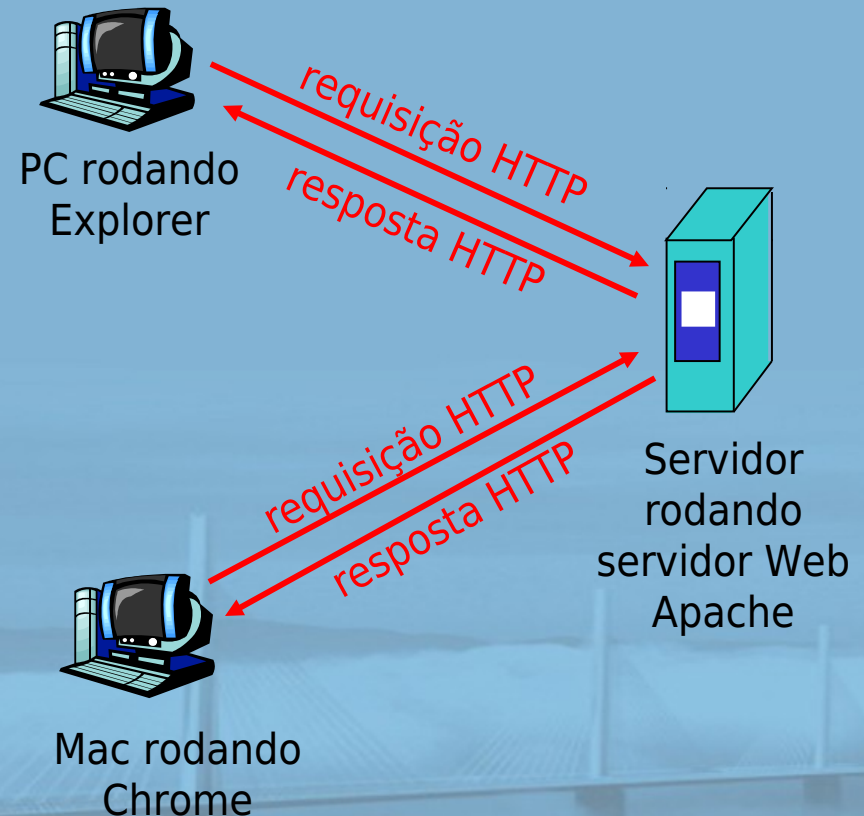
nome do hospedeiro

nome do caminho

Visão geral do HTTP

HTTP: HyperText Transfer Protocol

- protocolo da camada de aplicação da Web
- modelo cliente/servidor
 - ❖ *cliente*: navegador que requisita, recebe, “exibe” objetos Web
 - ❖ *servidor*: servidor Web envia objetos em resposta a requisições



usa TCP:

- ❑ cliente inicia conexão TCP (cria socket) com servidor, porta 80
- ❑ servidor aceita conexão TCP do cliente
- ❑ mensagens HTTP (do protocolo da camada de aplicação) trocadas entre navegador (cliente HTTP) e servidor Web (servidor HTTP)
- ❑ conexão TCP fechada

HTTP é “sem estado”

- ❑ servidor não guarda informações sobre requisições passadas do cliente

aparte

Protocolos que mantêm “estado” são complexos!

- ❑ história passada (estado) deve ser mantida
- ❑ se servidor/cliente falhar, suas visões do “estado” podem ser incoerentes, devem ser reconciliadas

Conexões HTTP

HTTP não persistente

- ❑ no máximo um objeto é enviado por uma conexão TCP.

HTTP persistente

- ❑ múltiplos objetos podem ser enviados por uma única conexão TCP entre cliente e servidor.

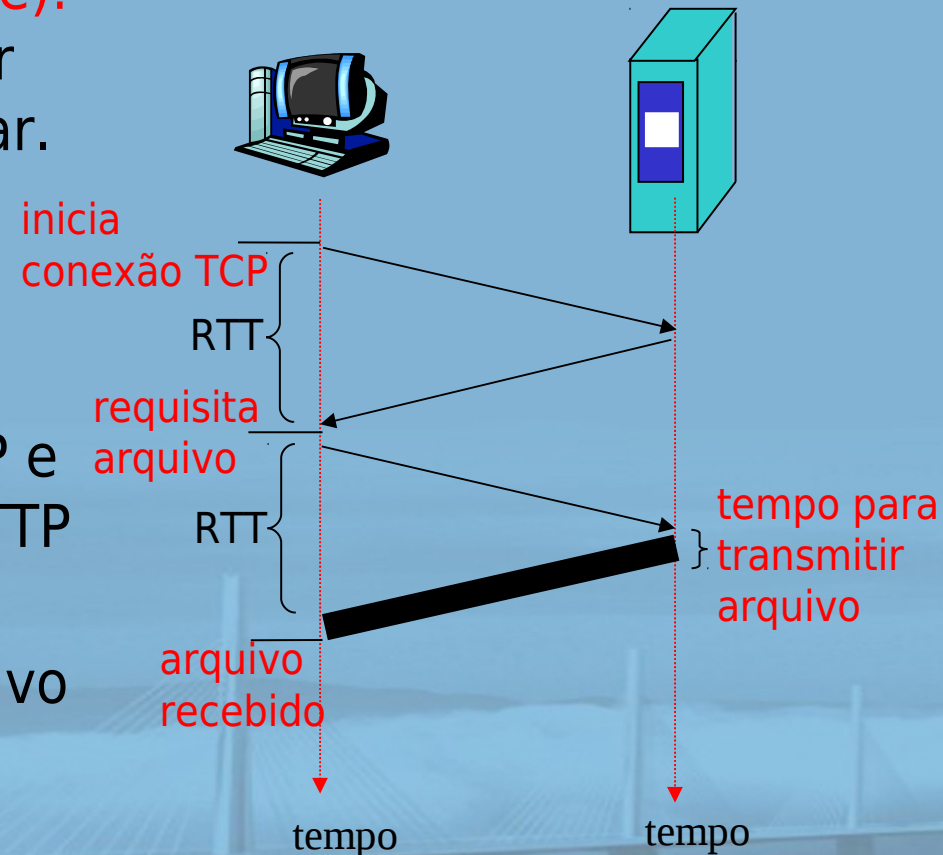
HTTP não persistente: tempo de resposta

definição de RTT (Round-Trip Time):
tempo para um pacote trafegar
do cliente ao servidor e retornar.

tempo de resposta:

- ❑ um RTT para iniciar a conexão TCP
- ❑ um RTT para a requisição HTTP e primeiros bytes da resposta HTTP retornarem
- ❑ tempo de transmissão de arquivo

total = 2RTT + tempo de transmissão



HTTP persistente

problemas do HTTP não persistente:

- ❑ requer 2 RTTs por objeto
- ❑ navegadores geralmente abrem conexões TCP paralelas para buscar objetos referenciados
- ❑ overhead do SO para *cada* conexão TCP

HTTP persistente:

- ❑ servidor deixa a conexão aberta depois de enviar a resposta
- ❑ mensagens HTTP seguintes entre cliente/servidor enviadas pela conexão aberta
- ❑ cliente envia requisições assim que encontra um objeto referenciado
- ❑ no mínimo um RTT para todos os objetos referenciados

Mensagem de requisição HTTP

- ❑ dois tipos de mensagens HTTP: *requisição, resposta*
- ❑ **mensagem de requisição HTTP:**
 - ❖ ASCII (formato de texto legível)

linha de requisição
(comandos GET,
POST, HEAD)

linhas de
cabeçalho

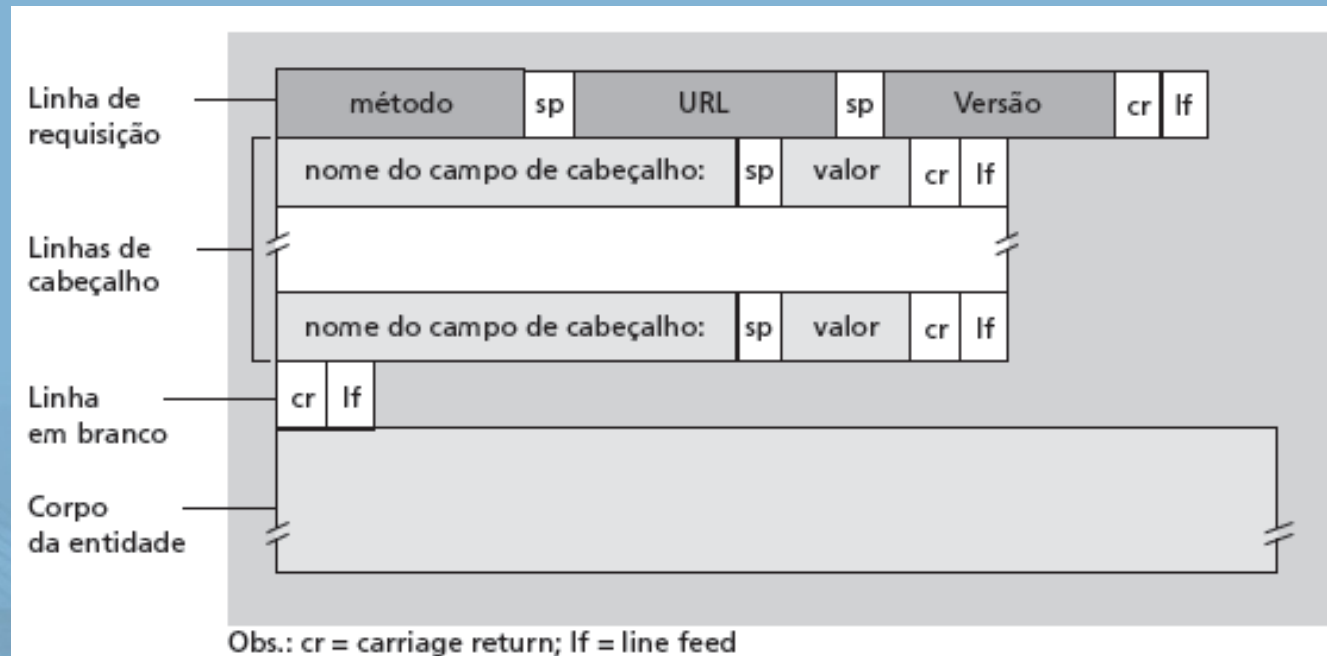
carriage return,
line feed
indica final
da mensagem

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

(carriage return, line feed extras)

Mensagem de requisição

HTTP: formato geral



Upload da entrada do formulário

método POST:

- ❑ página Web geralmente inclui entrada do formulário
- ❑ entrada é enviada ao servidor no corpo da entidade

método do URL:

- ❑ usa o método GET
- ❑ entrada é enviada no campo de URL da linha de requisição:

`www.umsite.com/buscaanimal?macacos&banana`

Tipos de método

HTTP/1.0

- ❑ GET
- ❑ POST
- ❑ HEAD
 - ❖ pede ao servidor para deixar objeto requisitado fora da resposta

HTTP/1.1

- ❑ Compactação
- ❑ PUT
 - ❖ envia arquivo no corpo da entidade ao caminho especificado no campo de URL
- ❑ DELETE
 - ❖ exclui arquivo especificado no campo de URL

Mensagem de resposta HTTP

linha de status
(protocolo
código de estado
frase de estado)

linhas de
cabeçalho

dados, p. e.,
arquivo HTML
requisitado

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 .....
Content-Length: 6821
Content-Type: text/html
```

dados dados dados dados dados ...

Códigos de estado da resposta HTTP

primeira linha da mensagem de resposta servidor->cliente
alguns exemplos de código:

200 OK

- ❖ requisição bem-sucedida, objeto requisitado mais adiante

301 Moved Permanently

- ❖ objeto requisitado movido, novo local especificado mais adiante na mensagem (Location:)

400 Bad Request

- ❖ mensagem de requisição não entendida pelo servidor

404 Not Found

- ❖ documento requisitado não localizado neste servidor

505 HTTP Version Not Supported

Testando o HTTP (lado cliente) você mesmo

1. Use Telnet para seu servidor Web favorito:

telnet www.each.usp.br 80

Abre conexão TCP com porta 80 (porta HTTP default do servidor) em cis.poly.edu. Qualquer coisa digitada é enviada à porta 80 em cis.poly.edu

2. Digite uma requisição HTTP GET:

GET /valdinei/ HTTP/1.1
Host: www.each.usp.br

Digitando isto (pressione carriage return duas vezes), você envia esta requisição GET mínima (mas completa) ao servidor HTTP

3. Veja a mensagem de resposta enviada pelo servidor HTTP!

Estado usuário-servidor: cookies

Muitos sites importantes
usam cookies

Quatro componentes:

- 1) linha de cabeçalho de cookie da mensagem de *resposta* HTTP
- 2) linha de cabeçalho de cookie na mensagem de *requisição* HTTP
- 3) arquivo de cookie na máquina do usuário, controlado pelo navegador do usuário
- 4) banco de dados de apoio no site Web

Exemplo:

- ❑ Susana sempre acessa a Internet pelo PC
- ❑ visita um site de comércio eletrônico pela primeira vez
- ❑ quando as primeiras requisições HTTP chegam ao site, este cria:
 - ❖ ID exclusivo
 - ❖ entrada no banco de dados de apoio para o ID

cliente

servidor



arq. de cookie



requisição http normal

resposta http normal
Set-cookie: 1678

requisição http normal
cookie: 1678

resposta http normal

requisição http normal
cookie: 1678

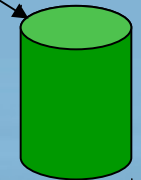
resposta http normal

servidor Amazon
cria ID 1678
para usuário

cria
entrada

acessa

acessa



banco de
dados de
apoio

ação
específica
do cookie

ação
específica
do cookie

O que os cookies podem ter:

- ☐ autorização
- ☐ carrinhos de compras
- ☐ recomendações
- ☐ estado da sessão do usuário (e-mail da Web)

Como manter o “estado”:

- ☐ extremidades do protocolo: mantêm estado no emissor/receptor por múltiplas transações
- ☐ cookies: mensagens HTTP transportam estado

aparte

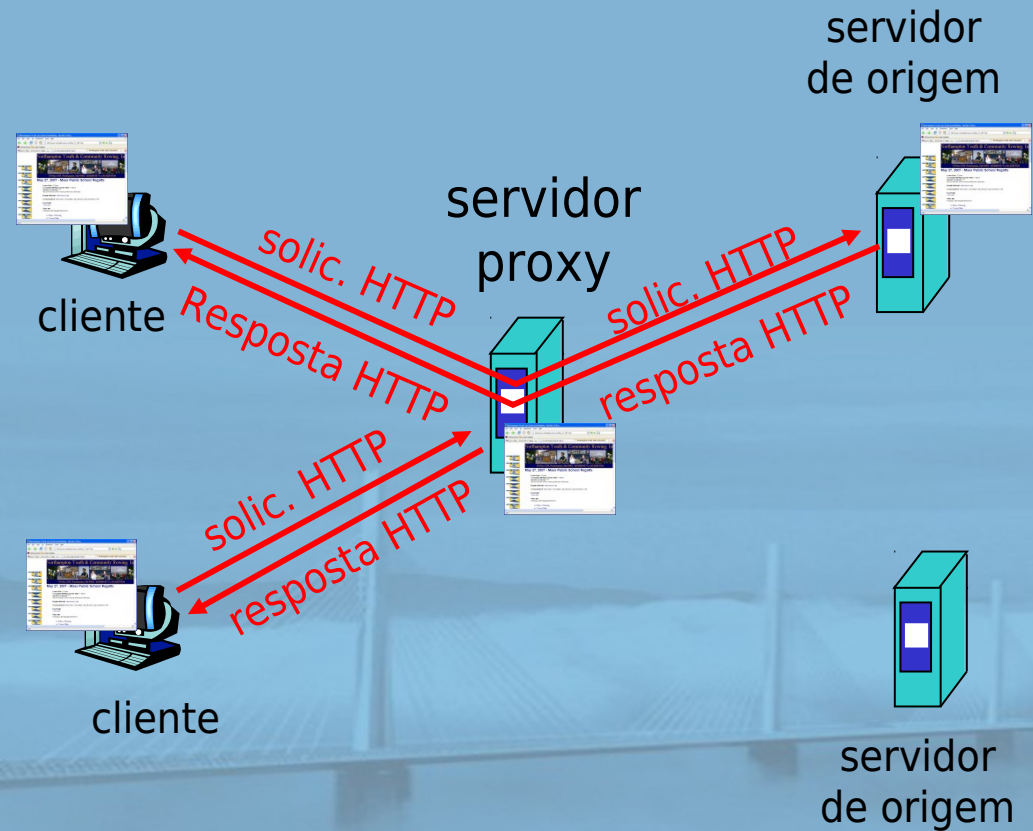
Cookies e privacidade:

- ☐ cookies permitem que os sites descubram muito sobre você
- ☐ Pode-se associar informações ao seu nome e seu e-mail

Caches Web (servidor proxy)

objetivo: satisfazer a requisição do cliente sem envolver servidor de origem

- ❑ usuário prepara navegador: acessos à Web via cache
- ❑ navegador envia todas as requisições HTTP ao cache
 - ❖ objeto no cache: cache retorna objeto
 - ❖ ou cache requisita objeto do servidor de origem, depois retorna objeto ao cliente



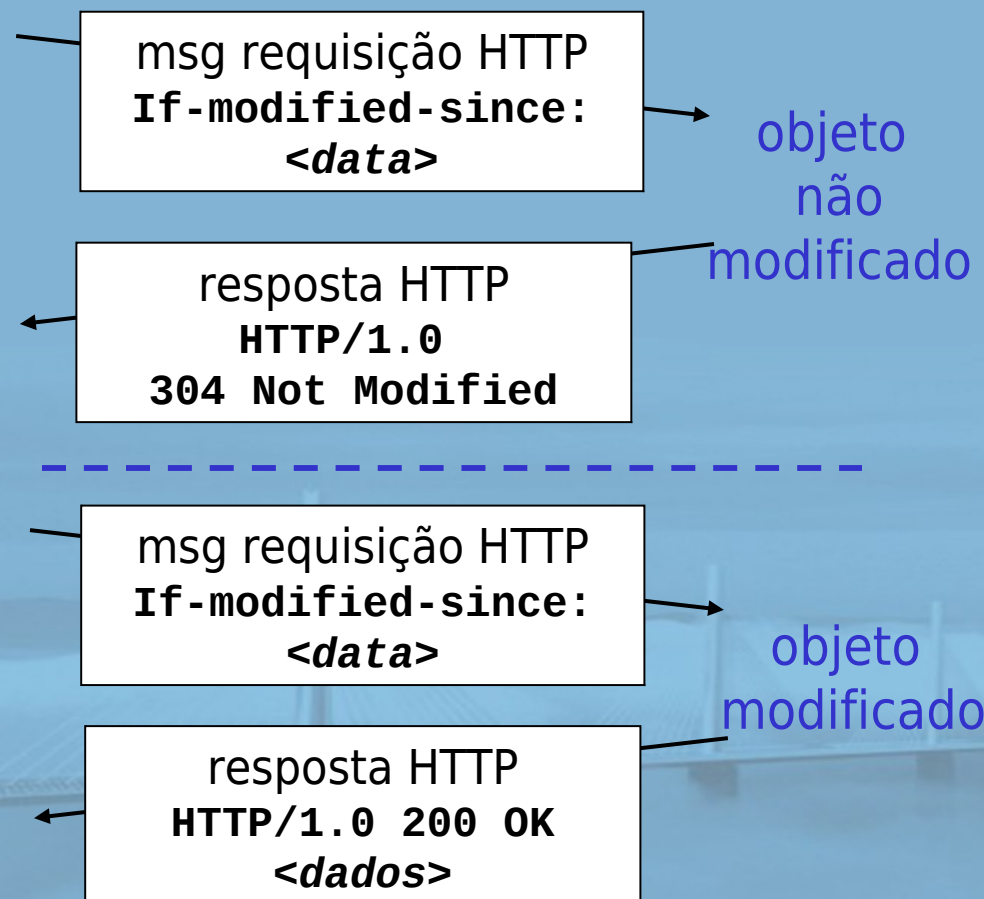
GET condicional

- ❑ **objetivo:** não enviar objeto se o cache tiver versão atualizada
- ❑ **cache:** especifica data da cópia em cache na requisição HTTP
If-modified-since:
<data>
- ❑ **servidor:** resposta não contém objeto se a cópia em cache estiver atualizada:

HTTP/1.0 304 Not Modified

cache

servidor

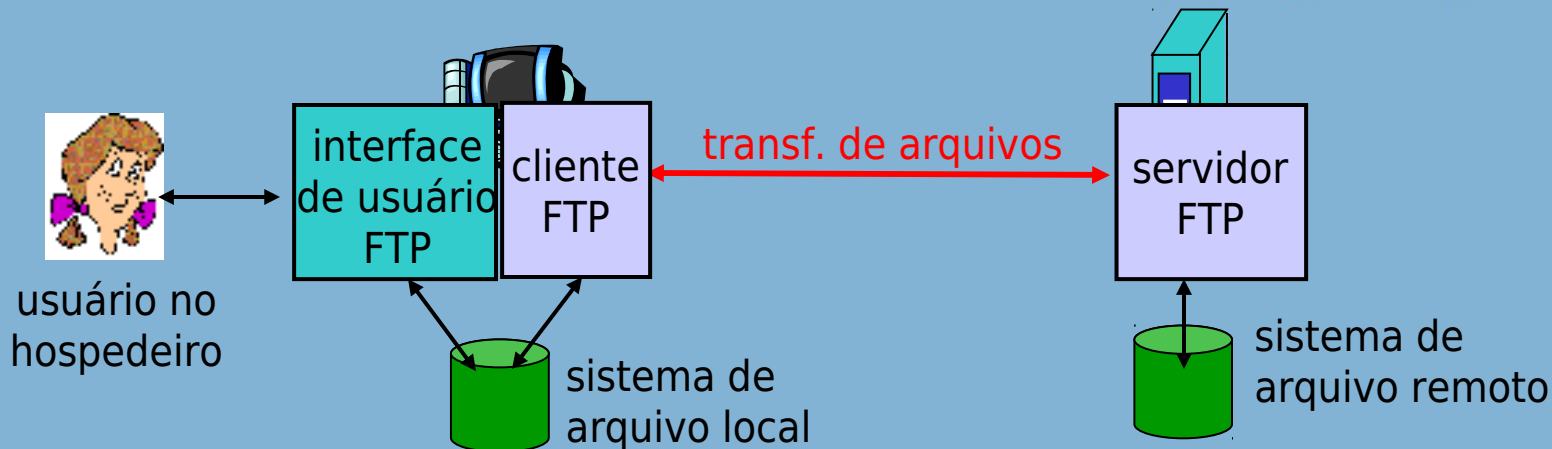


Capítulo 2: Camada de aplicação

- ❑ 2.1 Princípios de aplicações de rede
- ❑ 2.2 A Web e o HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Correio eletrônico
 - ❖ SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 Aplicações P2P
- ❑ 2.7 Programação de sockets com UDP
- ❑ 2.8 Programação de sockets com TCP

FTP: o protocolo de transferência de arquivos

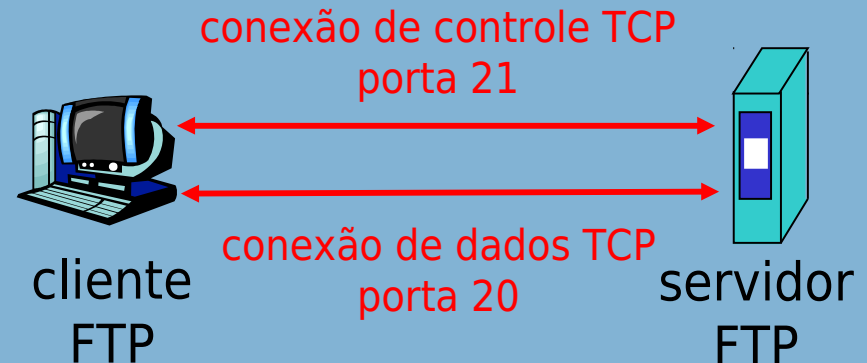
Uma Abordagem Top-Down



- ❑ transfere arquivo de/para hospedeiro remoto
- ❑ modelo cliente/servidor
 - ❖ *cliente*: lado que inicia transferência (de/para remoto)
 - ❖ *servidor*: hospedeiro remoto
- ❑ ftp: RFC 959
- ❑ servidor ftp: porta 21

FTP: conexões separadas para controle e dados

- ❑ cliente FTP contacta servidor FTP na porta 21, TCP é protocolo de transporte
- ❑ cliente autorizado por conexão de controle
- ❑ cliente navega por diretório remoto enviando comandos por conexão de controle
- ❑ quando servidor recebe comando de transferência de arquivo, abre 2ª conexão TCP (para arquivo) com cliente
- ❑ após transferir um arquivo, servidor fecha conexão de dados



- ❑ servidor abre outra conexão de dados TCP para transferir outro arquivo
- ❑ conexão de controle: “fora da banda”
- ❑ servidor FTP mantém “estado”: diretório atual, autenticação anterior

Comandos e respostas FTP

exemplos de comandos:

- ❑ enviado como texto ASCII pelo canal de controle
- ❑ **USER *nome-usuário***
- ❑ **PASS *senha***
- ❑ **LIST** retorna lista de arquivos no diretório atual
- ❑ **RETR *nome-arquivo*** recupera (apanha) arquivo
- ❑ **STOR *nome-arquivo*** armazena (coloca) arquivo no hospedeiro remoto

exemplos de códigos de retorno

- ❑ código e frase de estado (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**

Capítulo 2:

Camada de aplicação

- ❑ 2.1 Princípios de aplicações de rede
- ❑ 2.2 A Web e o HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Correio eletrônico
 - ❖ SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 Aplicações P2P
- ❑ 2.7 Programação de sockets com UDP
- ❑ 2.8 Programação de sockets com TCP

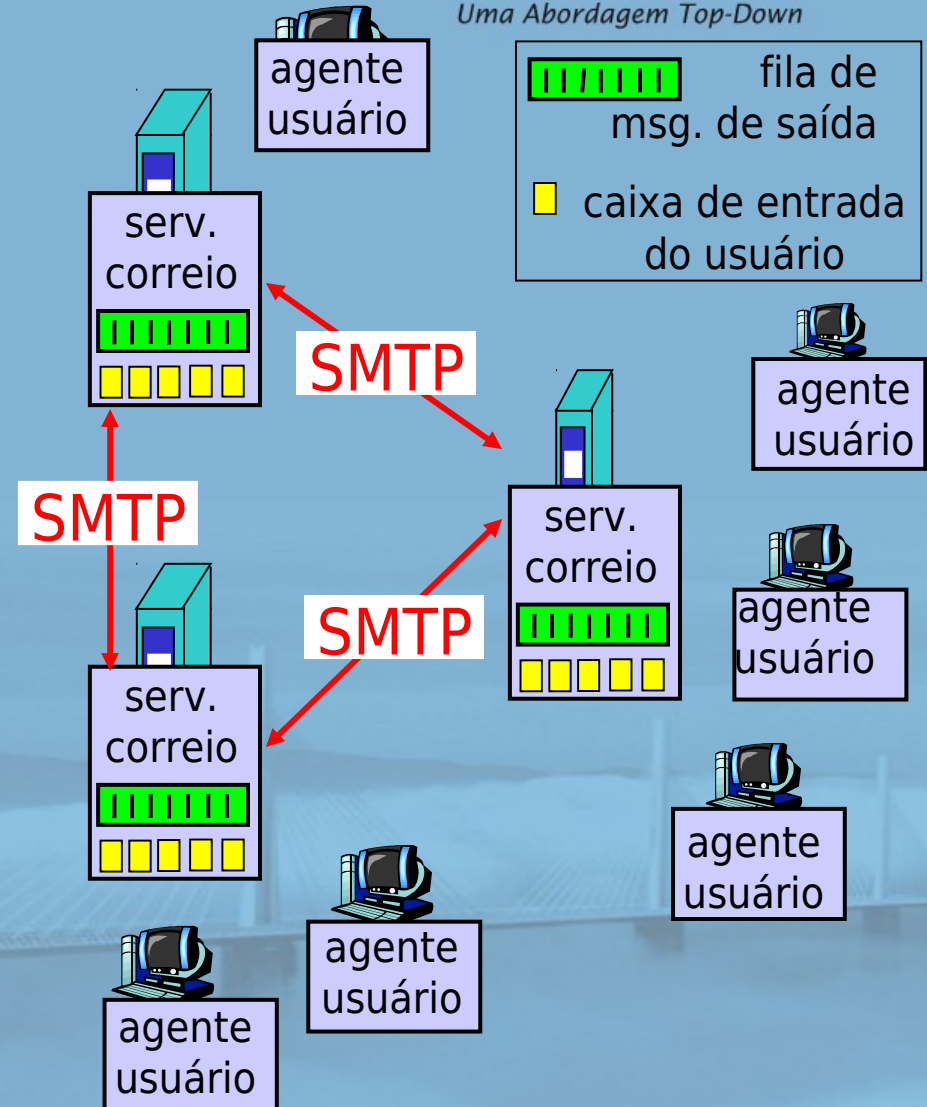
Correio eletrônico

Três componentes principais:

- ❑ agentes do usuário
- ❑ servidores de correio
- ❑ Simple Mail Transfer Protocol: SMTP

Agente do usuário

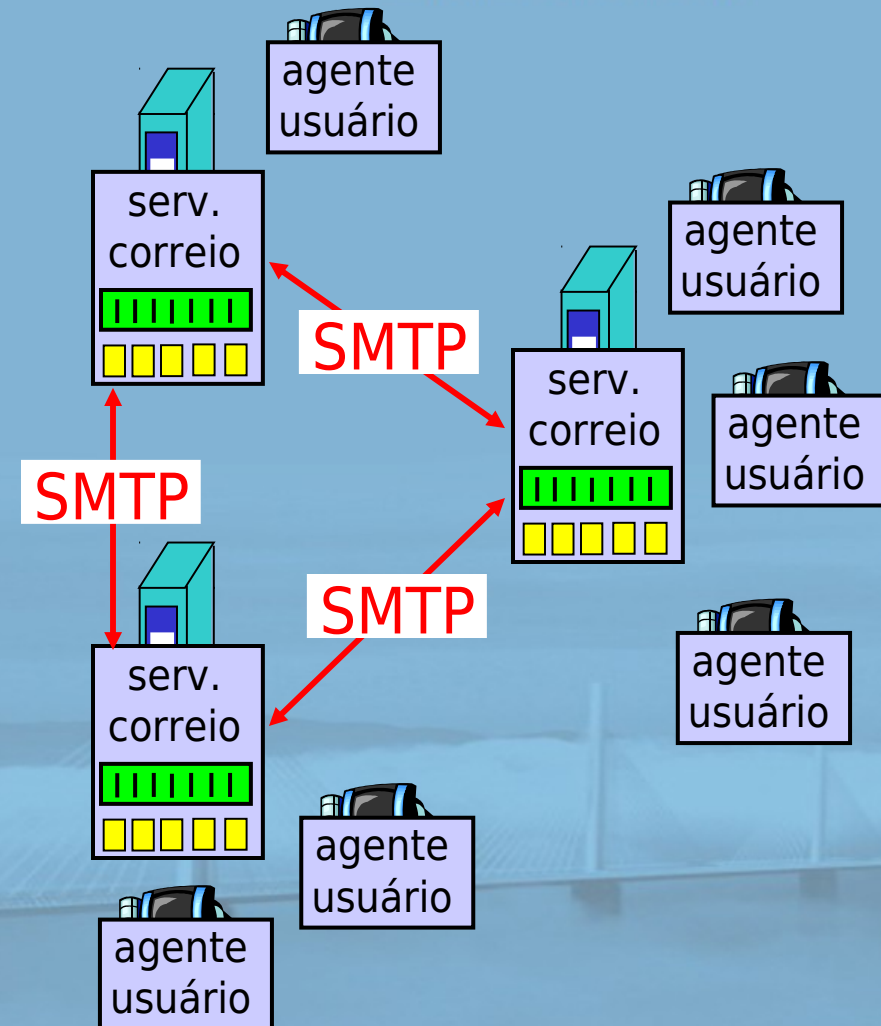
- ❑ também chamado “leitor de correio”
- ❑ redigir, editar, ler mensagens de correio eletrônico
- ❑ p. e., Outlook, Mozilla Thunderbird
- ❑ mensagens entrando e saindo armazenadas no servidor



Correio eletrônico

servidores de correio

- ❑ **caixa de correio** contém mensagens que chegam para o usuário
- ❑ **fila de mensagens** com mensagens de correio a serem enviadas
- ❑ **protocolo SMTP** entre servidores de correio para enviar mensagens de e-mail
 - ❖ cliente: servidor de envio de correio
 - ❖ “servidor”: servidor de recepção de correio

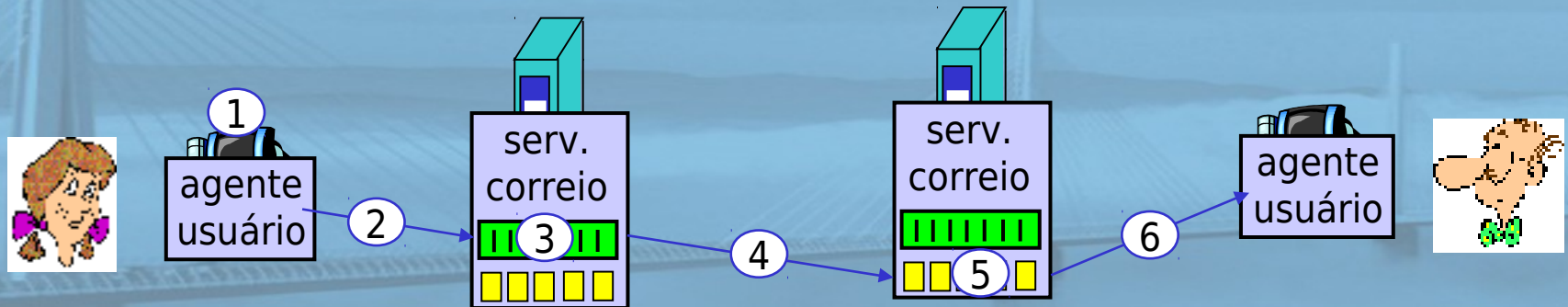


Correio eletrônico: SMTP [RFC 2821]

- ❑ usa TCP para transferir de modo confiável a mensagem de e-mail do cliente ao servidor, porta 25
- ❑ transferência direta: servidor de envio ao servidor de recepção
- ❑ interação comando/resposta
 - ❖ comandos: texto ASCII
 - ❖ resposta: código e frase de estado
- ❑ mensagens devem estar em ASCII de 7 bits
- ❑ MIME (Multipurpose Internet Mail Extensions)

Cenário: Alice envia mensagem a Bob

- 1) Alice usa AU para redigir mensagem “para” bob@algumaescola.edu
- 2) O AU de Alice envia mensagem ao seu servidor de correio, que é colocada na fila de mensagens
- 3) Lado cliente do SMTP abre conexão TCP com servidor de correio de Bob
- 4) Cliente SMTP envia mensagem de Alice pela conexão TCP
- 5) Servidor de correio de Bob coloca mensagem na caixa de correio de Bob
- 6) Bob chama seu agente do usuário para ler mensagem



Exemplo de interação SMTP

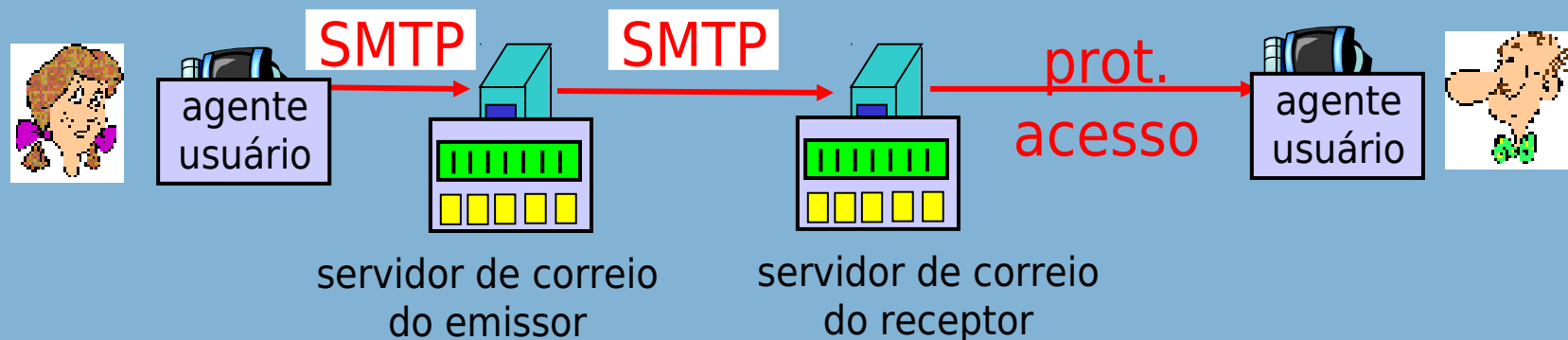
```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Você gosta de ketchup?
C: Que tal pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Teste a interação SMTP você mesmo:

- ❑ **telnet *nome-servidor* 25**
- ❑ veja resposta 220 do servidor
- ❑ digite comandos HELO, MAIL FROM, RCPT TO, DATA, QUIT

isso permite que você envie e-mail sem usar o cliente de e-mail (leitor)

Protocolos de acesso de correio



- ❑ SMTP: remessa/armazenamento no servidor do receptor
- ❑ protocolo de acesso ao correio: recuperação do servidor
 - ❖ POP: Post Office Protocol [RFC 1939]
 - autorização (agente <--> servidor) e download
 - ❖ IMAP: Internet Mail Access Protocol [RFC 1730]
 - mais recursos (mais complexo)
 - manipulação de msgs armazenadas no servidor
 - ❖ HTTP: gmail, Hotmail, Yahoo! Mail etc.

Capítulo 2: Camada de aplicação

- ❑ 2.1 Princípios de aplicações de rede
- ❑ 2.2 A Web e o HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Correio eletrônico
 - ❖ SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 Aplicações P2P
- ❑ 2.7 Programação de sockets com UDP
- ❑ 2.8 Programação de sockets com TCP

DNS: Domain Name System

hospedeiros da Internet, roteadores:

- ❖ endereço IP (32 bits) – usado para endereçar datagramas
- ❖ “nome”, p. e., www.yahoo.com – usado pelos humanos

P: Como mapear entre endereço IP e nome?

Domain Name System:

- ❑ *banco de dados distribuído* implementado na hierarquia de muitos *servidores de nomes*
- ❑ *protocolo em nível de aplicação* servidores de nomes se comunicam para *resolver* nomes (tradução endereço/nome)
- ❑ Função básica da Internet, implementada como protocolo em nível de aplicação

DNS

Serviços de DNS

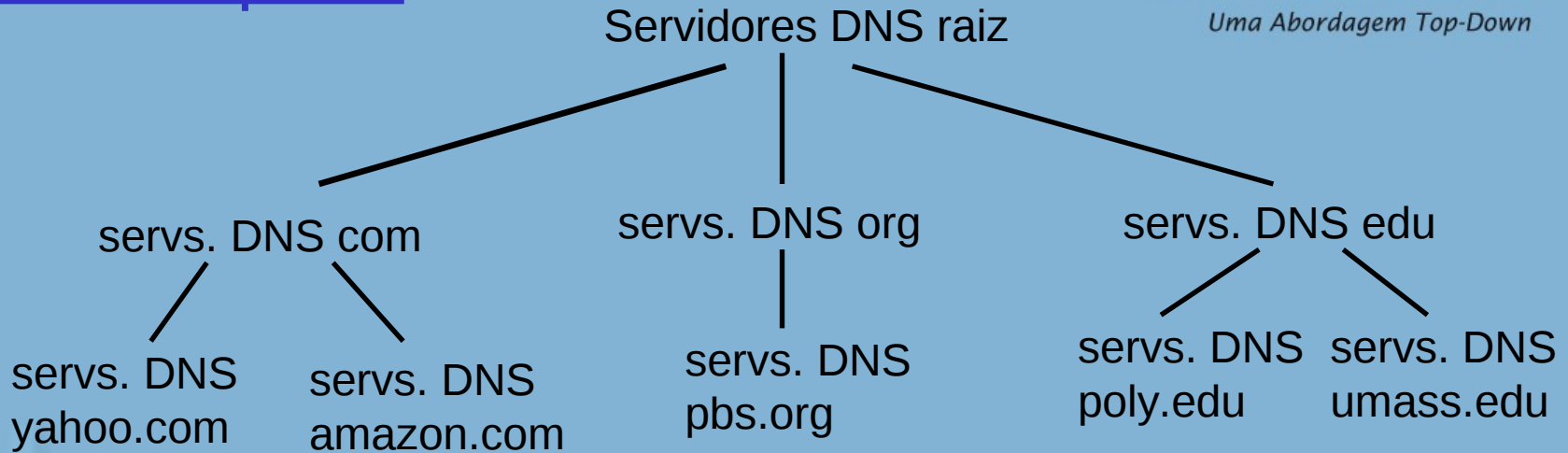
- ❑ tradução nome de hospedeiro -> endereço IP
- ❑ apelidos de hospedeiro
 - ❖ nomes canônicos
- ❑ apelidos de servidor de correio
- ❑ distribuição de carga
 - ❖ servidores Web replicados: conjunto de endereços IP para um nome canônico

Por que não centralizar o DNS?

- ❑ único ponto de falha
- ❑ volume de tráfego
- ❑ banco de dados centralizado distante
- ❑ manutenção

Não é escalável!

Banco de dados distribuído, hierárquico



Cliente quer IP para www.amazon.com:

- ❑ cliente consulta serv. **raiz** para achar servidor **DNS com**
- ❑ cliente consulta serv. **DNS com** para obter serv. **DNS amazon.com**
- ❑ cliente consulta serv. **DNS amazon.com** para obter endereço IP para **www.amazon.com**

DNS: Servidores de nomes raiz

- ❑ contactados por servidores de nomes locais que não conseguem traduzir nome
- ❑ servidores de nomes raiz:
 - ❖ contacta servidor de nomes com autoridade se o mapeamento não for conhecido
 - ❖ obtém mapeamento
 - ❖ retorna mapeamento ao servidor de nomes local



13 servidores de
nomes raiz no
mundo

TLD e servidores com autoridade

- ❑ servidores de domínio de alto nível (TLD – Top Level Domain) :
 - ❖ responsáveis por com, org, net, edu etc. e todos os domínios de país de alto nível: br, uk, fr, ca, jp.
 - ❖ A Network Solutions mantém servidores para TLD com
 - ❖ NIC (Núcleo de Informação e Coordenação do Ponto BR) – TLD br
- ❑ servidores DNS com autoridade:
 - ❖ servidores DNS da organização, provendo nome de hospedeiro com autoridade a mapeamentos IP para os servidores da organização (p. e., Web, correio).
 - ❖ podem ser mantidos pela organização ou provedor de serviços

Servidor de nomes local

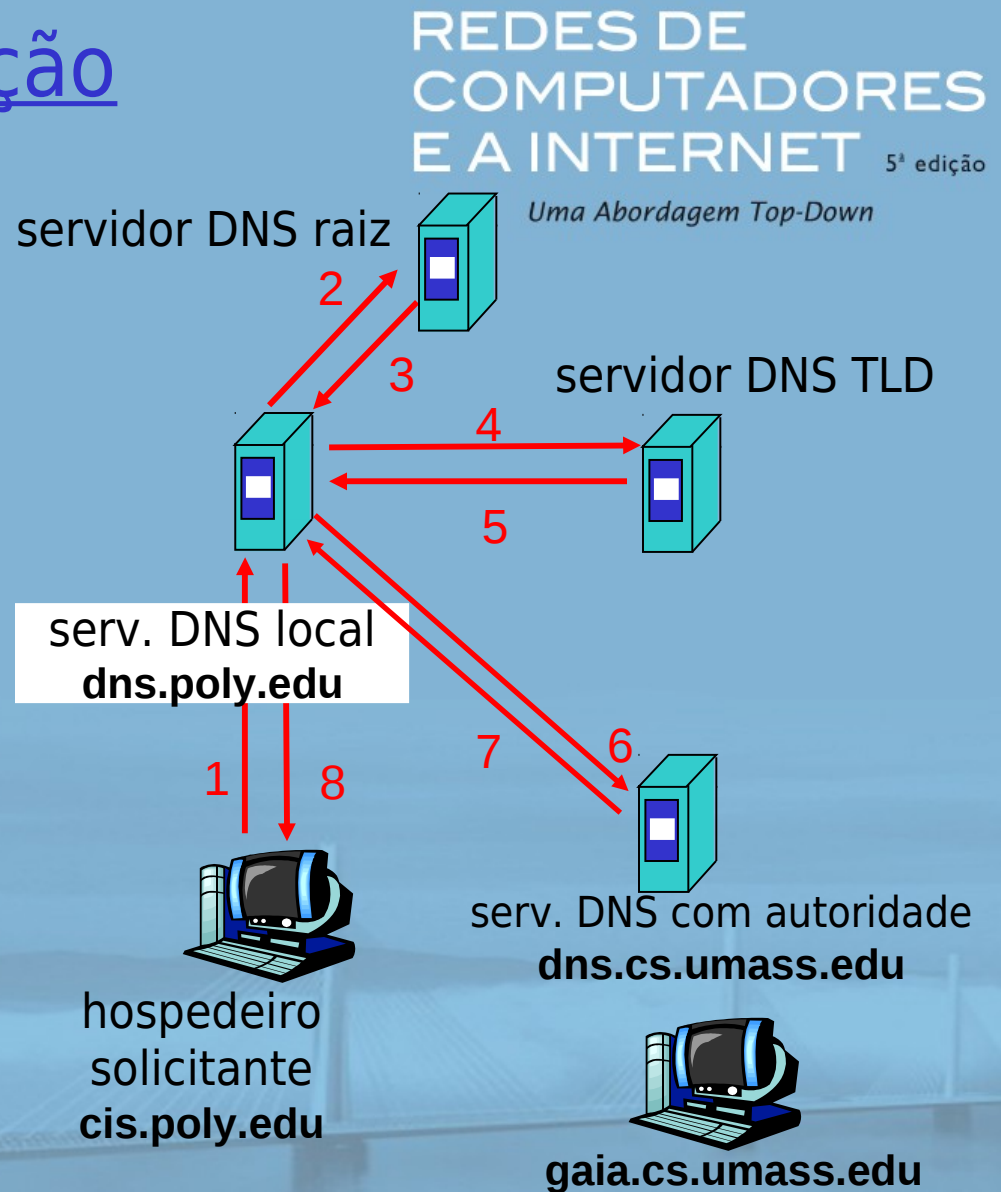
- ❑ não pertence estritamente à hierarquia
- ❑ cada ISP (ISP residencial, empresa, universidade) tem um.
 - ❖ também chamado “servidor de nomes default”
- ❑ quando hospedeiro faz consulta ao DNS, consulta é enviada ao seu servidor DNS local
 - ❖ atua como proxy, encaminha consulta para hierarquia

Exemplo de resolução de nome DNS

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

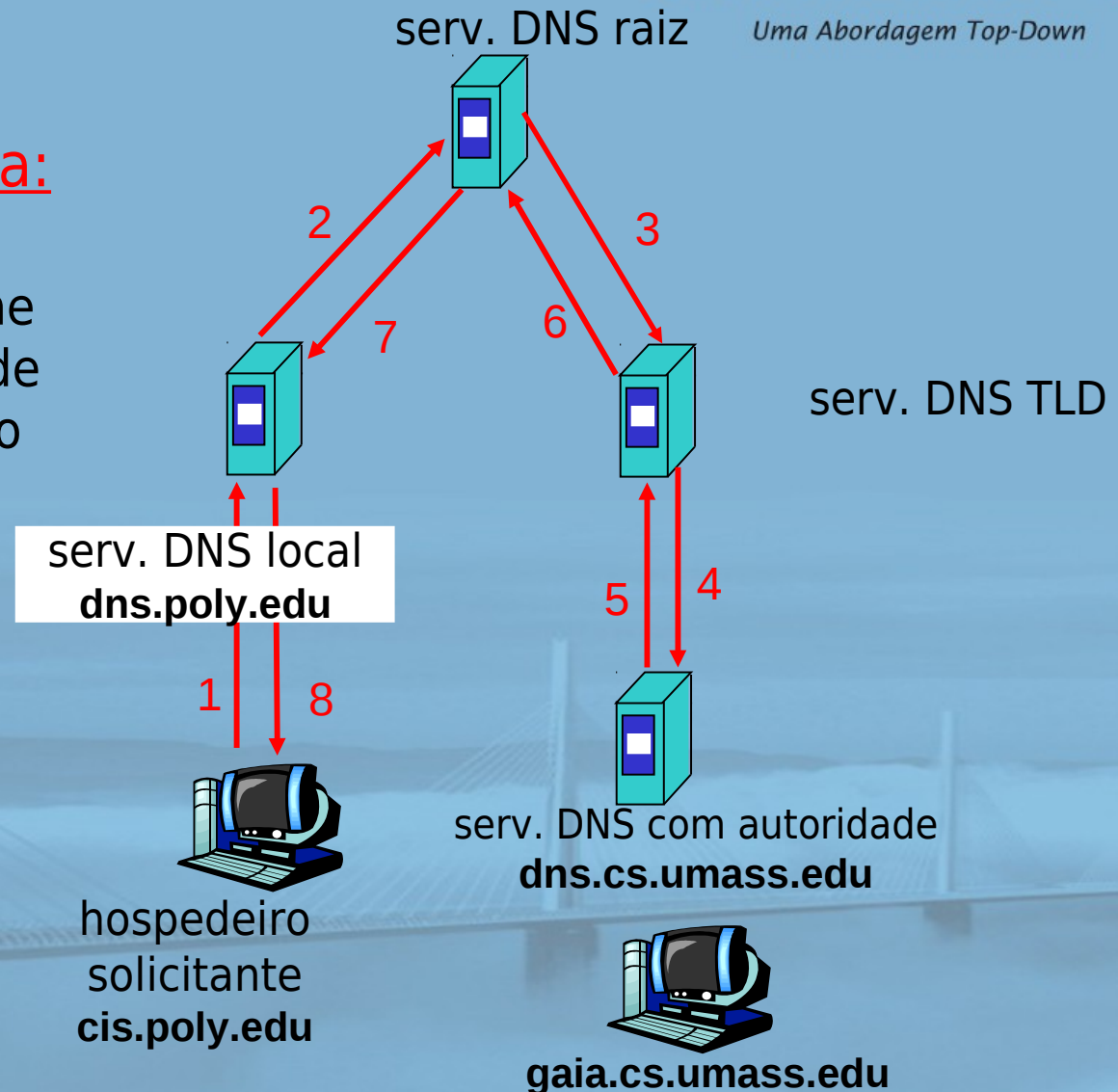
consulta repetida:

- servidor contactado responde com nome do servidor a contactar
- “não conheço esse nome, mas pergunte a este servidor”



consulta recursiva:

- coloca peso da resolução de nome sobre o servidor de nomes contactado
- carga pesada?



DNS: caching e atualização de registros

- ❑ quando (qualquer) servidores de nomes descobre o mapeamento, ele o mantém em *cache*
 - ❖ entradas de cache esgotam um tempo limite (desaparecem) após algum tempo
 - ❖ servidores TLD normalmente são mantidos em caches nos servidores de nomes locais
 - Assim, os servidores de nomes raiz não são consultados com frequência
- ❑ mecanismos de atualização/notificação em projeto na IETF

Registros de DNS

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

formato do RR: (**nome**, **valor**, **tipo**, **ttl**)

❑ Tipo = A

- ❖ **nome** é o “hostname”
- ❖ **valor** é o endereço IP

❑ Tipo = NS

- ❖ **nome** é o domínio (p. e. foo.com)
- ❖ **valor** é o “hostname” do servidor de nomes com autoridade tal domínio

❑ Tipo = SOA (start of authority)

- ❖ melhor fonte de informação para um domínio

❑ Tipo = CNAME

- ❖ **nome** é apelido para algum nome “canônico” (real)
www.ibm.com é na realidade
servereast.backup2.ibm.com
- ❖ **valor** é o nome canônico

❑ Tipo = MX

- ❖ **valor** é o nome do servidor de correio associado ao **nome**

Protocolo DNS, mensagens

protocolo DNS: mensagens de *consulta* e *resposta*, ambas com algum *formato de mensagem*

cabeçalho da mensagem

- ❑ **identificação**: # de 16 bits para consulta; resposta usa mesmo #
- ❑ **flags**:
 - ❖ consulta ou resposta
 - ❖ recursão desejada
 - ❖ recursão disponível
 - ❖ resposta é com autoridade

Identificação	Flags	12 bytes
Número de perguntas	Número de RRs de resposta	
Número de RRs com autoridade	Número de RRs adicionais	
Perguntas (número variável de perguntas)		
Respostas (número variável de registros de recursos)		
Autoridade (número variável de registros de recursos)		
Informação adicional (número variável de registros de recursos)		

Protocolo DNS, mensagens

Identificação	Flags	12 bytes
Número de perguntas	Número de RRs de resposta	
Número de RRs com autoridade	Número de RRs adicionais	
Perguntas (número variável de perguntas)		
Respostas (número variável de registros de recursos)		
Autoridade (número variável de registros de recursos)		
Informação adicional (número variável de registros de recursos)		

campos de nome e tipo
para uma consulta

RRs na resposta
à consulta

registros para servidores
com autoridade

informação adicional
“útil” que pode ser usada

Inserindo registros no DNS

- ❑ exemplo: nova empresa “Network Utopia”
- ❑ registre o nome networkutopia.com.br na *entidade registradora de DNS* (p. e., NIC)
 - ❖ oferece nomes, endereços IP do servidor de nomes com autoridade (primário e secundário)
 - ❖ entidade insere dois RRs no servidor TLD com:

(networkutopia.com.br, dns1.networkutopia.com.br, NS)
(dns1.networkutopia.com.br, 212.212.212.1, A)
- ❑ crie registro Tipo A do servidor com autoridade para www.networkutopia.com.br; registro Tipo MX para networkutopia.com.br

O que aprendemos sobre *protocolos*?

- ❑ troca de mensagem típica de requisição/resposta:
 - ❖ cliente solicita informação ou serviço
 - ❖ servidor responde com dados, código de estado
- ❑ formatos de mensagem:
 - ❖ cabeçalhos: campos dando informações sobre dados
 - ❖ dados: informações sendo comunicadas

Temas importantes:

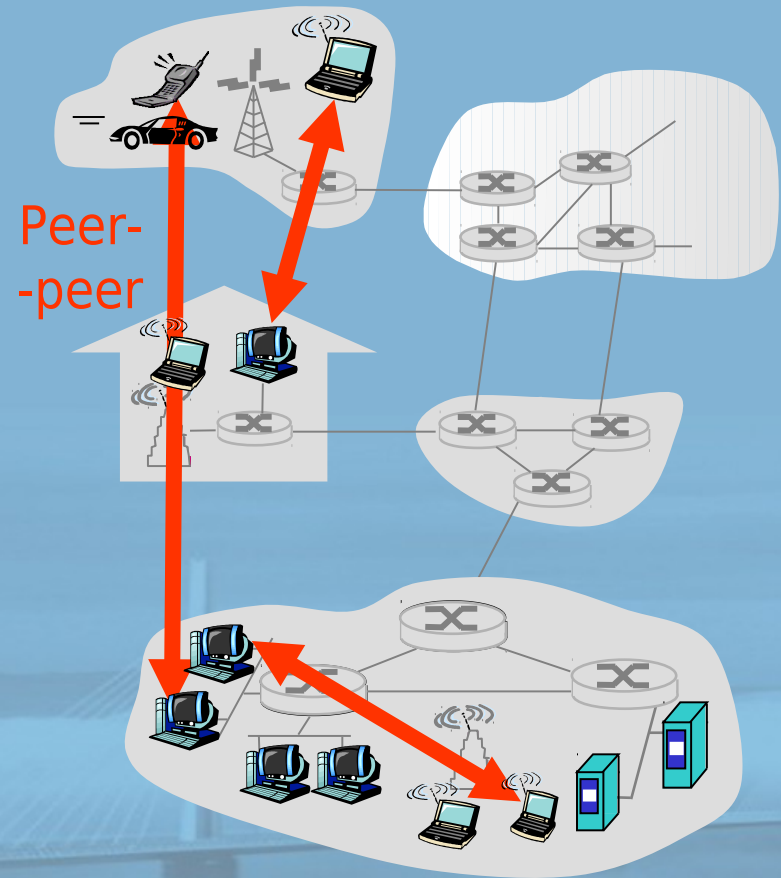
- ❑ msgs de controle e dados
 - ❖ na banda, fora da banda
- ❑ centralizado *versus* descentralizado
- ❑ sem estado *versus* com estado
- ❑ transf. de msg confiável *versus* não confiável
- ❑ “complexidade na borda da rede”

Capítulo 2: Camada de aplicação

- ❑ 2.1 Princípios de aplicações de rede
- ❑ 2.2 A Web e o HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Correio eletrônico
 - ❖ SMTP, POP3, IMAP
- ❑ 2.5 DNS
- ❑ 2.6 Aplicações P2P
- ❑ 2.7 Programação de sockets com UDP
- ❑ 2.8 Programação de sockets com TCP

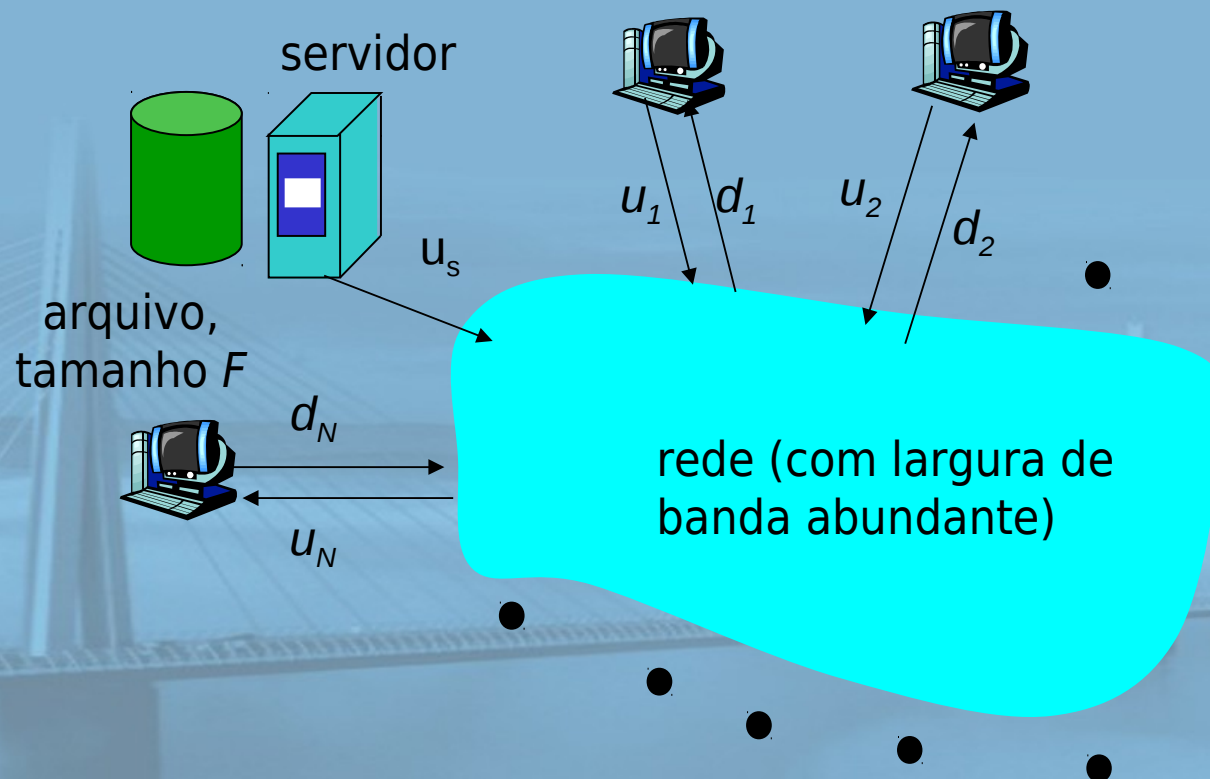
Arquitetura P2P pura

- ❑ sem servidor sempre ligado
- ❑ sistemas finais arbitrários se comunicam diretamente
- ❑ pares estão conectados intermitentemente e mudam de endereços IP
- ❑ Protocolos:
 - ❖ Torrent
 - ❖ Skype



Distribuição de arquivo: cliente-servidor versus P2P

Pergunta: Quanto tempo para distribuir arquivo de um servidor para N pares?



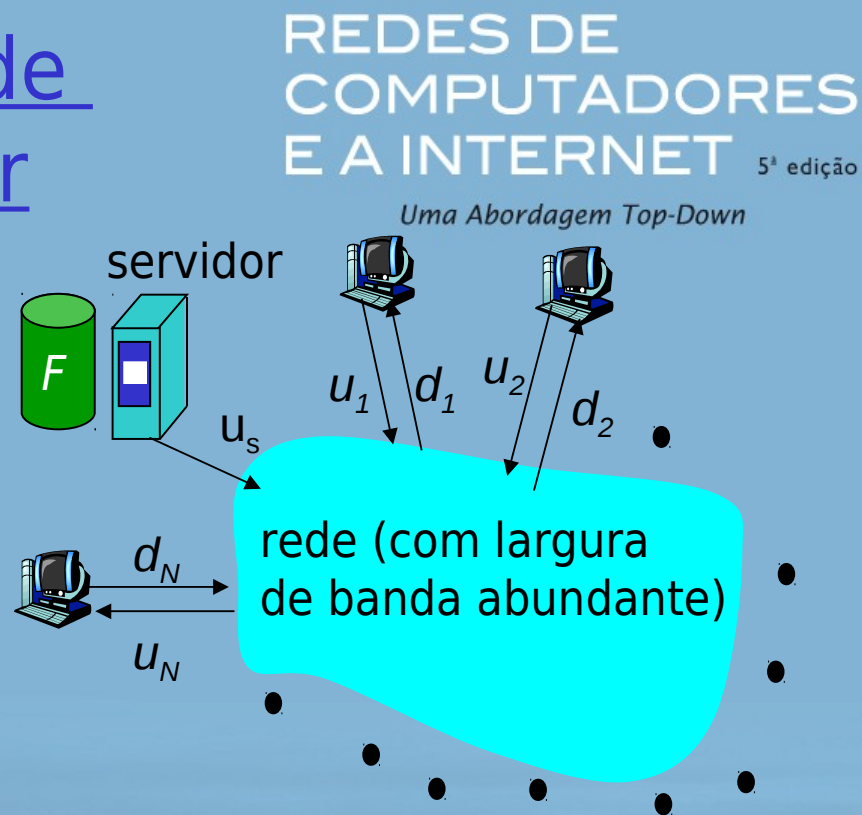
u_s : largura de banda de upload do servidor

u_i : largura de banda de upload do par i

d_i : largura de banda de download do par i

Tempo de distribuição de arquivo: cliente-servidor

- ❑ servidor envia N cópias sequencialmente:
 - ❖ tempo NF/u_s
- ❑ cliente i leva um tempo F/d_i para o download



tempo para distribuir F
a N clientes usando
técnica cliente/servidor

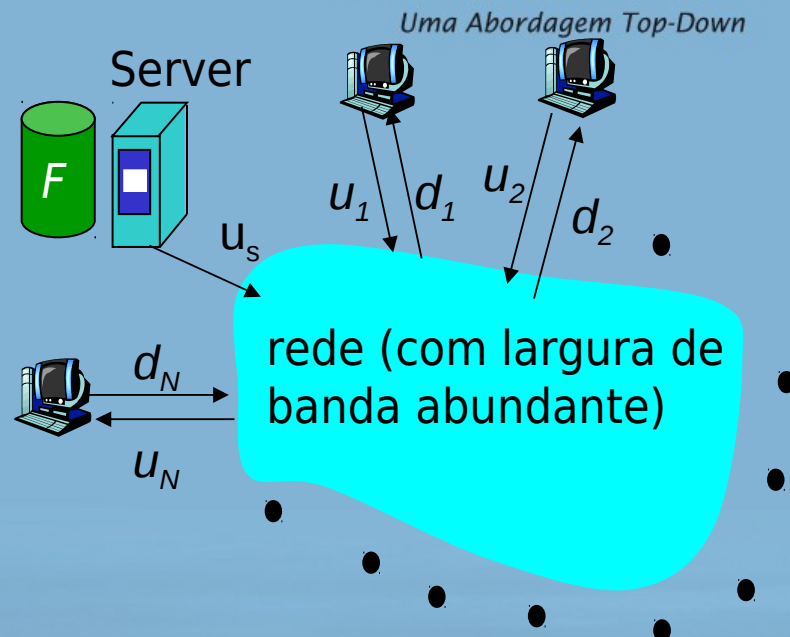
$$D_{CS} = \max \left\{ \frac{NF}{u_s}, \frac{F}{d_{\min}} \right\}$$

aumenta linearmente em N
(para N grande)

Tempo de distribuição de arquivo: P2P

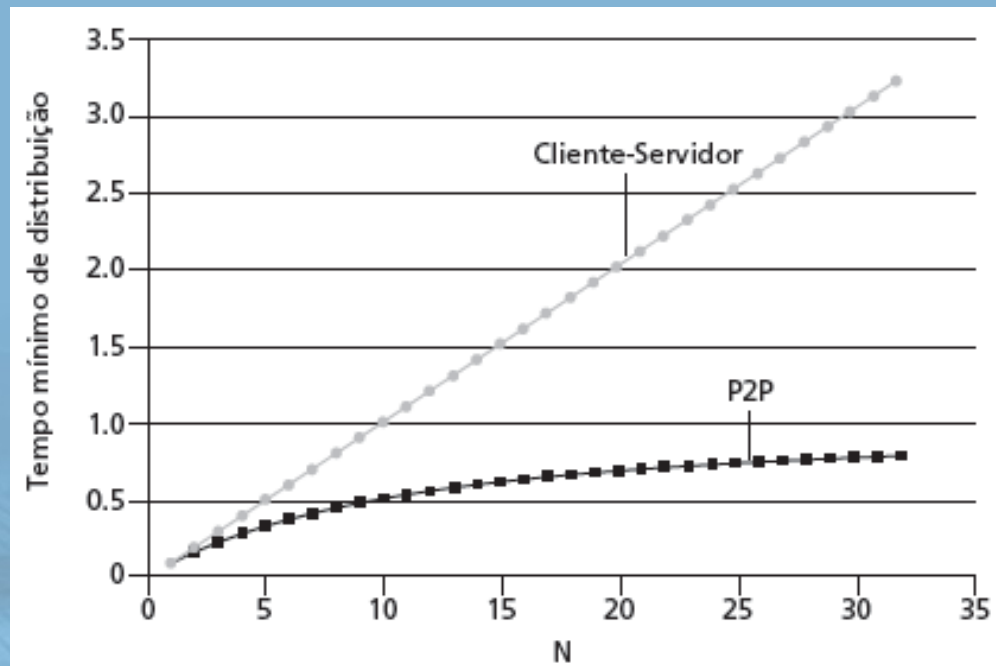
- ❑ servidor deve enviar uma cópia: tempo F/u_s
- ❑ cliente i leva tempo F/d_i para o download
- ❑ NF bits devem ser baixados (agregados)
- ❑ taxa de upload mais rápida possível: $u_s + \sum u_i$

$$D_{P2P} = \max \left\{ \frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum u_i} \right\}$$



Cliente-servidor versus P2P: exemplo

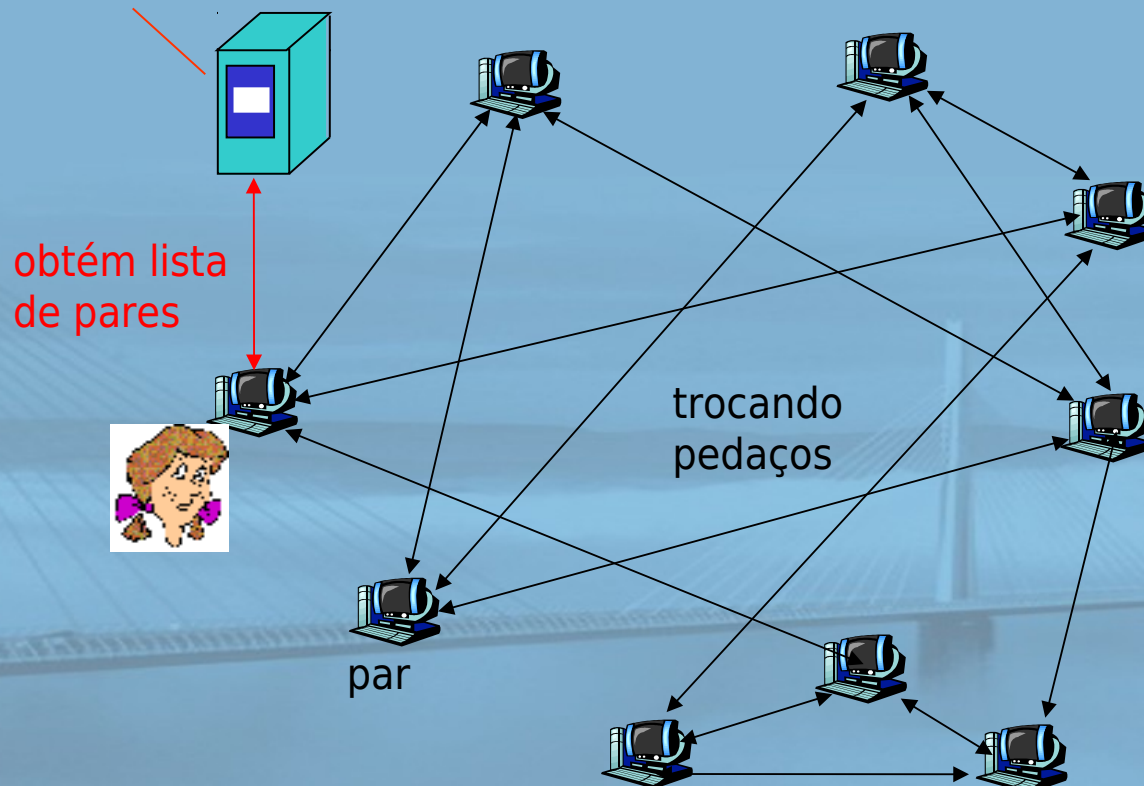
Taxa de upload cliente = u , $F/u = 1$ hora, $u_s = 10u$, $d_{\min} \geq u_s$



Distribuição de arquivos: BitTorrent

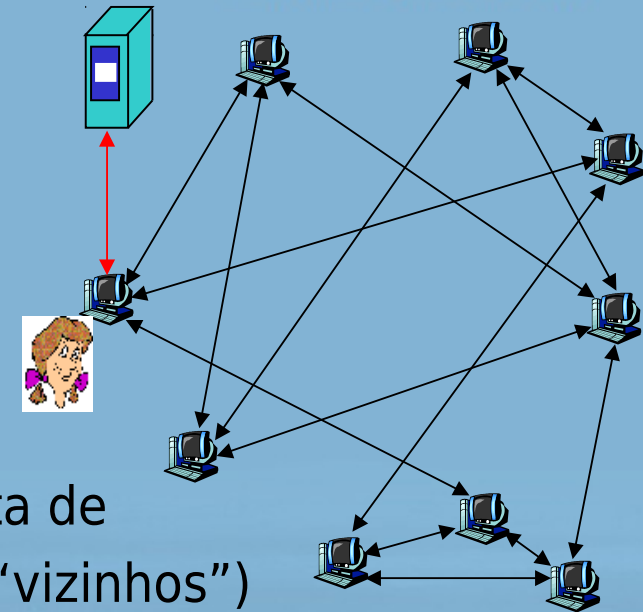
- distribuição de arquivos P2P

rastreador: verifica pares
que participam do torrent



BitTorrent

- ❑ arquivo dividido em *pedaços* de 256 KB.
- ❑ torrent de ajuntamento de pares:
 - ❖ não tem pedaços, mas os acumulará com o tempo
 - ❖ registra com rastreador para obter lista de pares, conecta a subconjunto de pares (“vizinhos”)
- ❑ ao fazer download, par faz upload de pedaços para outros pares
- ❑ pares podem ir e vir
- ❑ quando par tem arquivo inteiro, ele pode (de forma egoísta) sair ou (de forma altruísta) permanecer



Empurrando pedaços

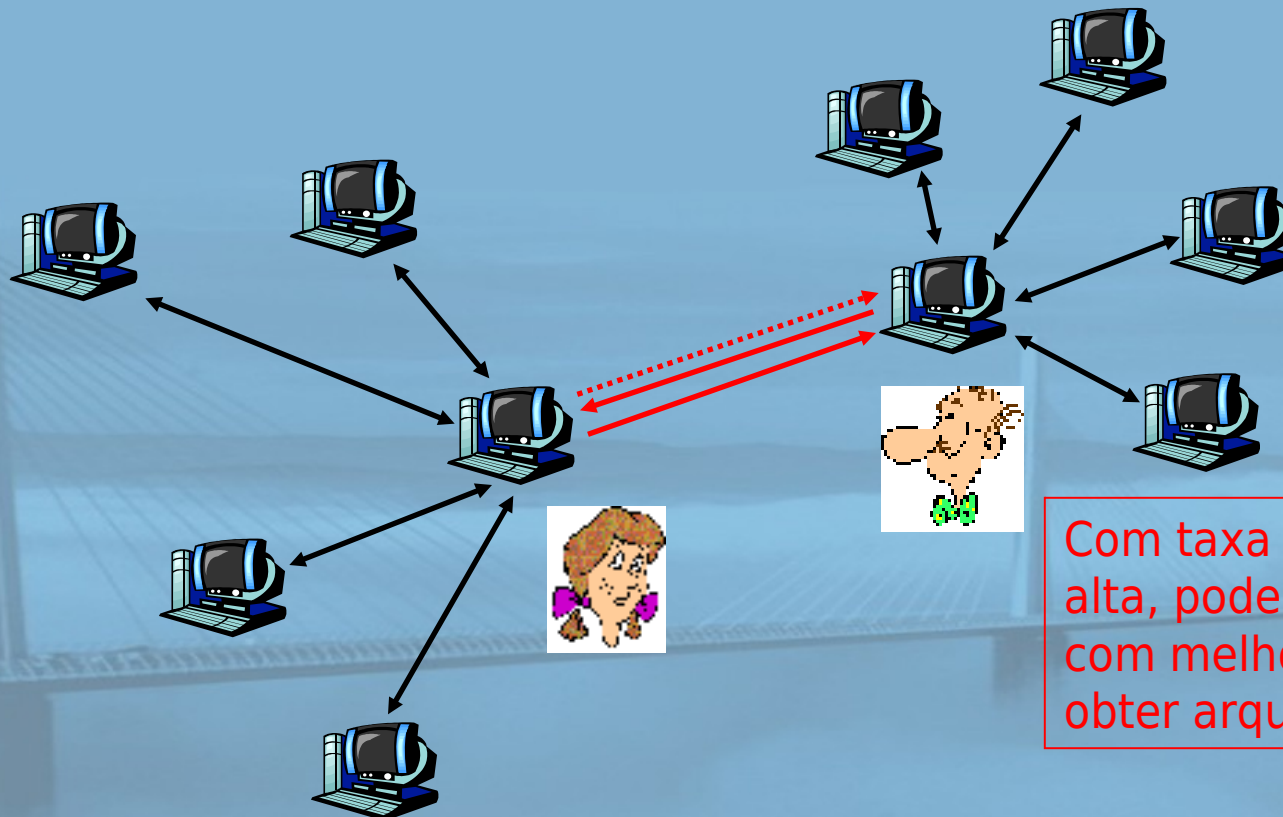
- a qualquer momento, diferentes pares têm diferentes subconjuntos de pedaços de arquivo
- periodicamente, um par (Alice) pede a cada vizinho a lista de pedaços que eles têm
- Alice envia requisições para seus pedaços que faltam
 - ❖ mais raros primeiro

Enviando pedaços: olho por olho

- Alice envia pedaços a quatro vizinhos atualmente enviando seus pedaços na *velocidade mais alta*
 - ❖ reavalia 4 maiores a cada 10 s
- a cada 30 s: seleciona outro par aleatoriamente, começa a enviar pedaços
 - ❖ par recém-escolhido pode se juntar aos 4 maiores
 - ❖ “desafoga” de forma otimista

BitTorrent: Olho por olho

- (1) Alice “desafoga” Bob de forma otimista
- (2) Alice um dos quatro maiores provedores de Bob; Bob recíproco
- (3) Bob torna-se um dos quatro maiores provedores de Alice



Com taxa de upload mais alta, pode achar parceiros com melhor negociação & obter arquivo mais rápido!

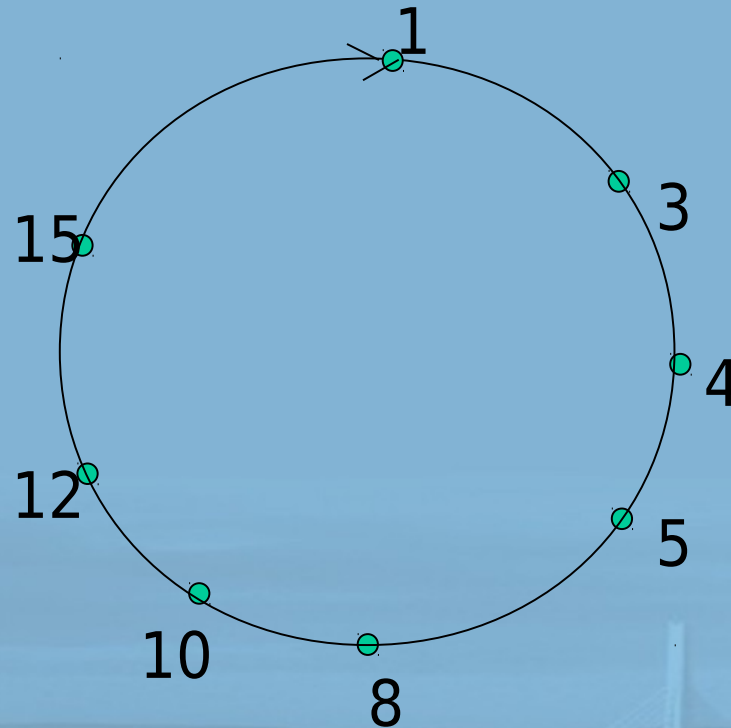
Distributed Hash Table (DHT)

- ❑ DHT = banco de dados P2P distribuído
- ❑ banco de dados tem duplas (**chave, valor**);
 - ❖ chave: hash arquivo; valor: conteúdo
 - ❖ chave: ID (aleatório); valor: endereço IP, porta
- ❑ pares **consultam** BD com chave
 - ❖ BD retorna valores que combinam com a chave
- ❑ pares também podem **inserir** duplas (chave, valor)

Identificadores DHT

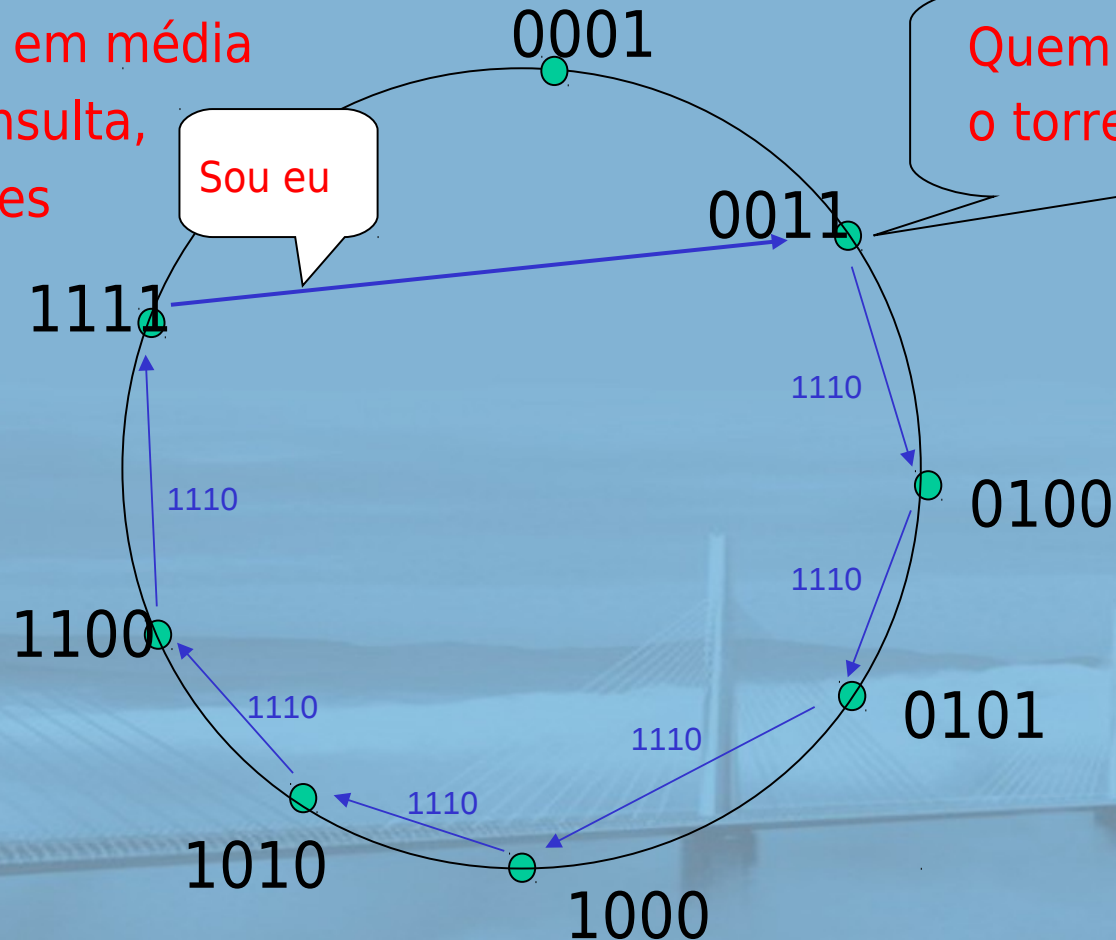
- ❑ atribuem identificador inteiro a cada par no intervalo $[0, 2^n - 1]$.
 - ❖ cada identificador pode ser representado por n bits.
- ❑ Distância entre identificadores.
 - ❖ exemplo: XOR (ou exclusivo)
- ❑ Banco de Dados Local
 - ❖ Gerencia (rastreador) torrents de arquivos próximos
 - ❖ Conhece sistemas finais próximos

DHT circular

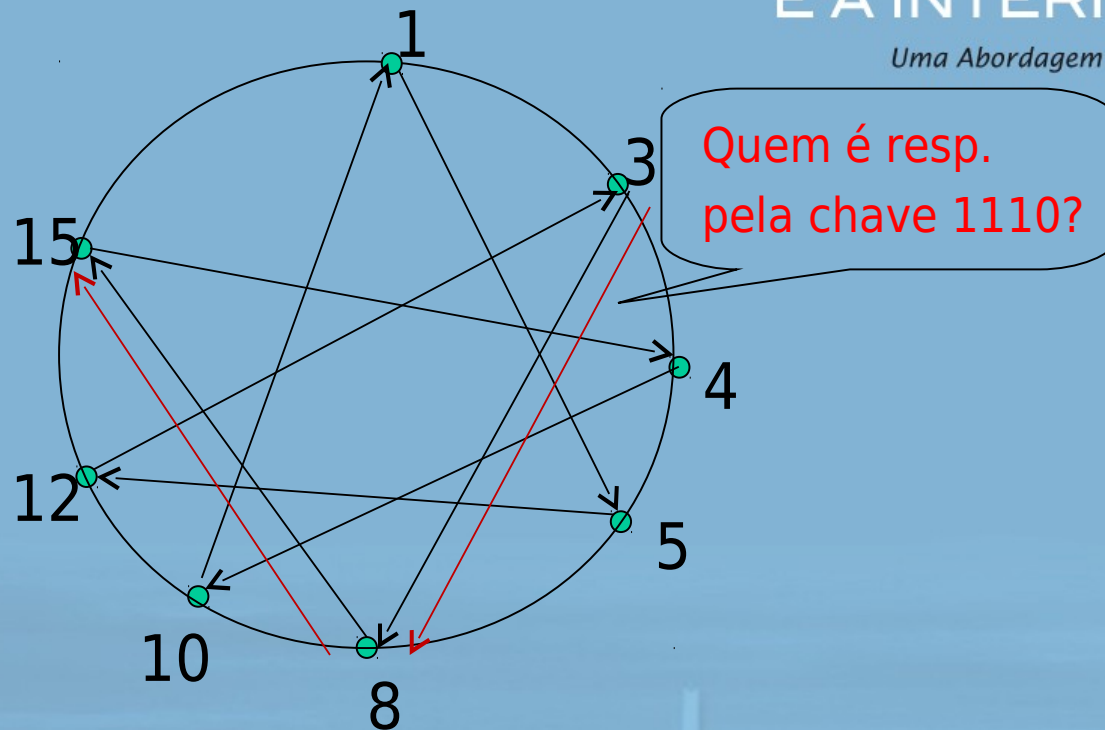


- ❑ cada par só conhece sucessor e predecessor imediato.
- ❑ “rede de sobreposição”

$O(N)$ mensagens em média
para resolver consulta,
quando há N pares



DHT circular com atalhos



- ❑ cada par registra endereços IP do predecessor, sucessor, atalhos
- ❑ reduzido de 6 para 2 mensagens
- ❑ possível criar atalhos de modo que $O(\log N)$ vizinhos, $O(\log N)$ mensagens na consulta

Protocolo Kademlia

- ❑ ID com 160 bits, operador de distância XOR
- ❑ Guarda lista de nós: <ID, IP, porta>
- ❑ Cada par divide o espaço de ID em 160 buckets
 - ❖ ID contíguos segundo a distância
 - ❖ tamanho do bucket relacionado com a distância
 - ❖ bucket i guarda ID com distâncias entre 2^i e 2^{i+1}
 - ❖ cada bucket armazena apenas k nós
 - ❖ ordenado pela última vez que foi visto
- ❑ Adicionando nós aos buckets
 - ❖ bucket possui menos que k nós
 - ❖ o nó mais antigo não responde

Protocolo Kademlia

□ Requisições

- ❖ FIND_NODE: o destino retorna triplas para os k nós mais próximos
- ❖ FIND_VALUE: se conhece o arquivo retorna o torrent, caso contrário funciona como FIND_NODE
- ❖ PING: verifica se um nó está vivo
- ❖ STORE: requisita a um nó para guardar uma dupla <key, value>

□ Encontrando um torrent

- ❖ Dentre os k nós mais próximos conhecidos do torrent, escolhe A nós para requisitar o valor
- ❖ Se não encontrou o torrent, continua com a busca com os novos nós conhecido

Protocolo Kademlia

□ Mantendo os torrents

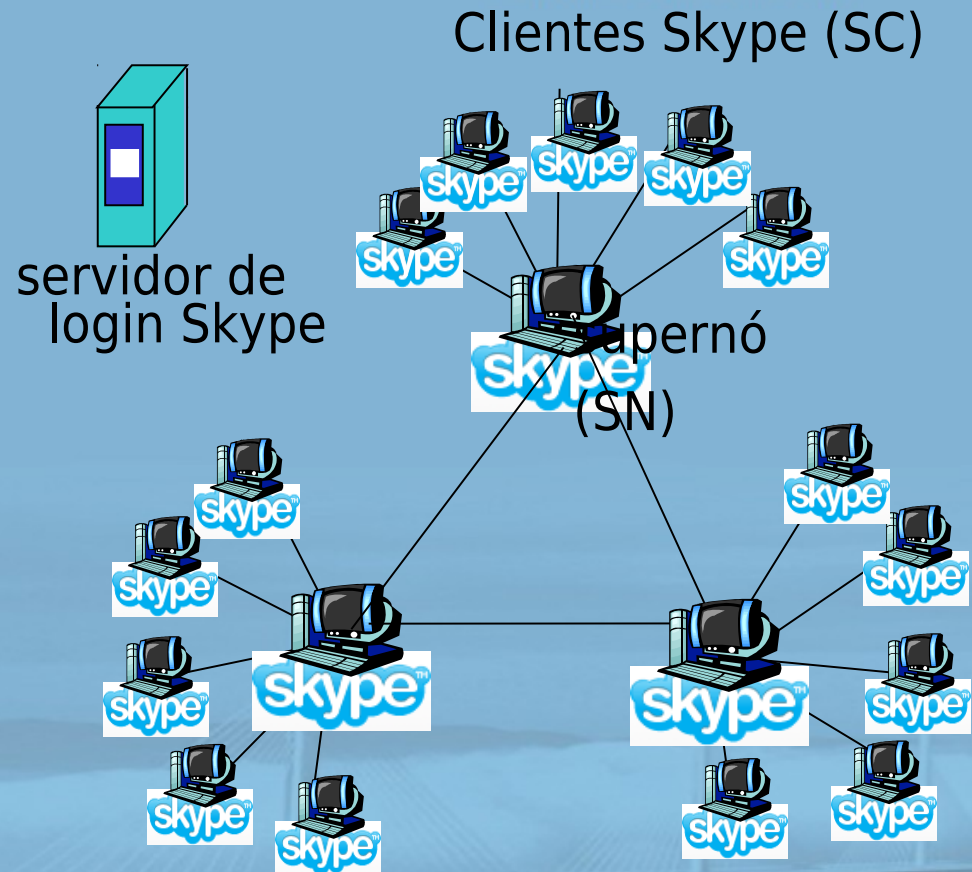
- ❖ o criador do torrent deve encontrar os k nós mais próximos e salvá-lo (STORE) nos mesmos
- ❖ quem possui os torrents deve republicá-lo aos k nós mais próximos
- ❖ O criador do torrent também deve republicá-lo
- ❖ quando um torrent é encontrado, deve-se republicá-lo aos k nós mais próximos

□ Juntando-se à rede

- ❖ deve-se ter ao menos um nó em seus buckets
- ❖ realiza-se então uma busca pelo próprio ID
- ❖ realiza-se busca para ID aleatório em cada bucket

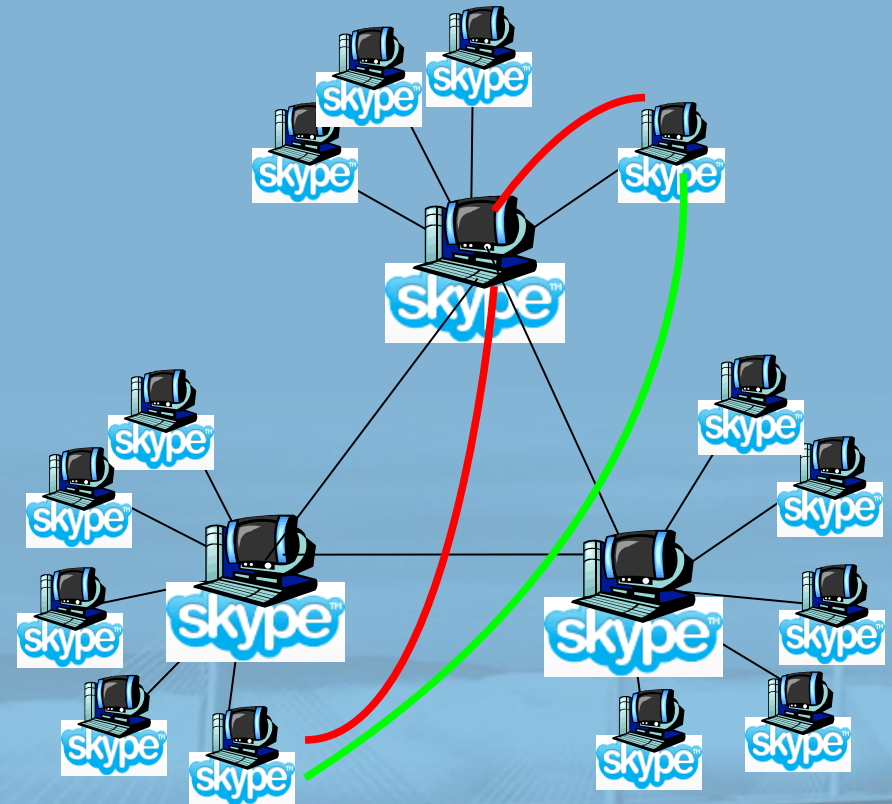
VoIP P2P: Skype

- ❑ inerentemente P2P: pares de usuários se comunicam.
- ❑ protocolo próprio da camada de aplicação (deduzido por engenharia reversa)
- ❑ sobreposição hierárquica com SNs
- ❑ índice compara usernames com endereços IP; distribuído por SNs



Pares como retransmissores

- ❑ problema quando Alice e Bob estão atrás de “NATs”
 - ❖ NAT impede que um par de fora inicie uma chamada para um par de dentro da rede
- ❑ **Retransmissão:**
 - ❖ usando os SNs de Alice e de Bob, o retransmissor é escolhido
 - ❖ cada par inicia a sessão com retransmissão.
- ❑ **Traversal**
 - ❖ Detecta o tipo de NAT
 - ❖ full-cone, restricted cone, restricted port



Capítulo 2: Resumo

terminamos nosso estudo das aplicações de rede!

- arquiteturas de aplicação
 - ❖ cliente-servidor
 - ❖ P2P
 - ❖ híbrido
- requisitos do servidor de aplicação:
 - ❖ confiabilidade, largura de banda, atraso
- modelo de serviço de transporte da Internet
 - ❖ orientado a conexão, confiável: TCP
 - ❖ não confiável, datagramas: UDP
- protocolos específicos:
 - ❖ HTTP
 - ❖ FTP
 - ❖ SMTP, POP, IMAP
 - ❖ DNS
 - ❖ P2P: BitTorrent, Skype
- programação de socket