



EACH

Escola de Artes, Ciências e Humanidades
da Universidade de São Paulo

ACH2026

Redes de Computadores

Sockets, FTP, Correio Eletrônico, P2P

Profa. Dra. Cíntia B. Margi

Setembro/2009

Atualizado pelo Prof. Dr. João Porto - Agosto/2010

ACH2026 - 2009



EACH

Escola de Artes, Ciências e Humanidades
da Universidade de São Paulo

Capítulo 2.3 - FTP



FTP (*File Transfer Protocol*)

Transferência de arquivos de e para o computador remoto.

Modelo cliente servidor:

Cliente: lado que inicia a transferência (seja de ou para o lado remoto)

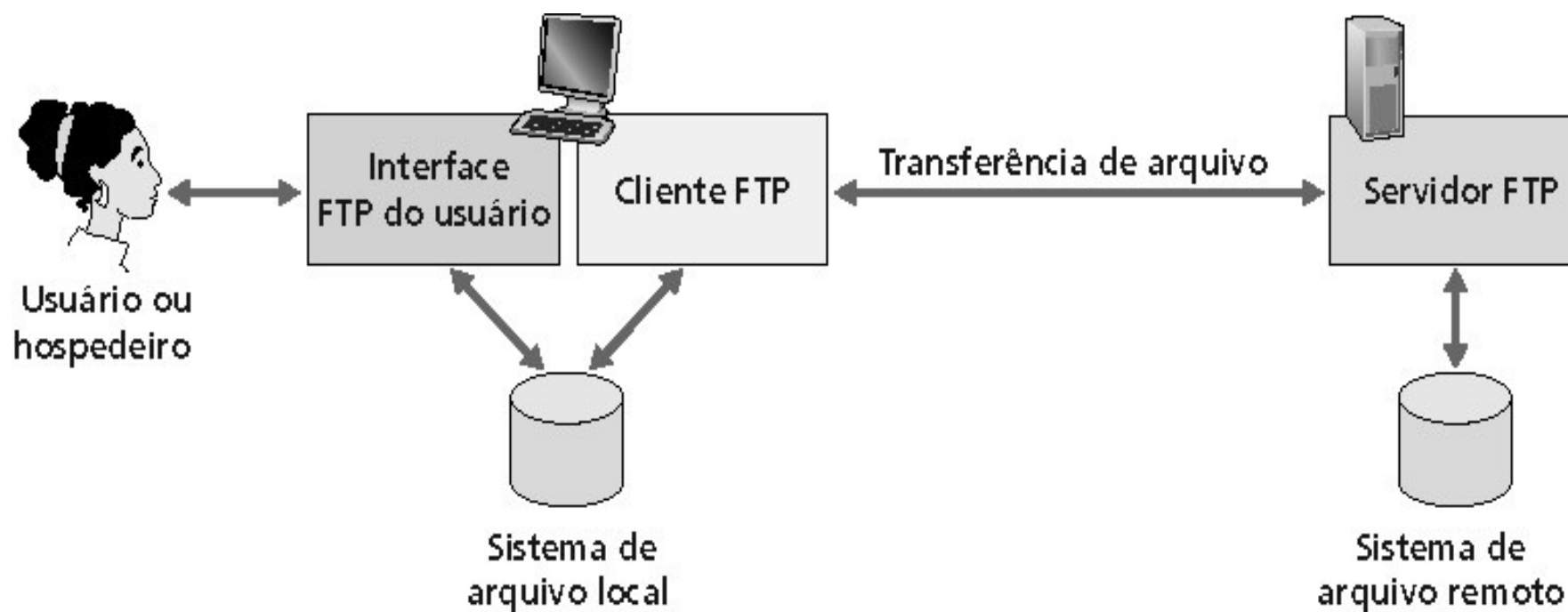
Servidor: hospedeiro remoto

RFC 959.

Servidor FTP: porta 21.



FTP

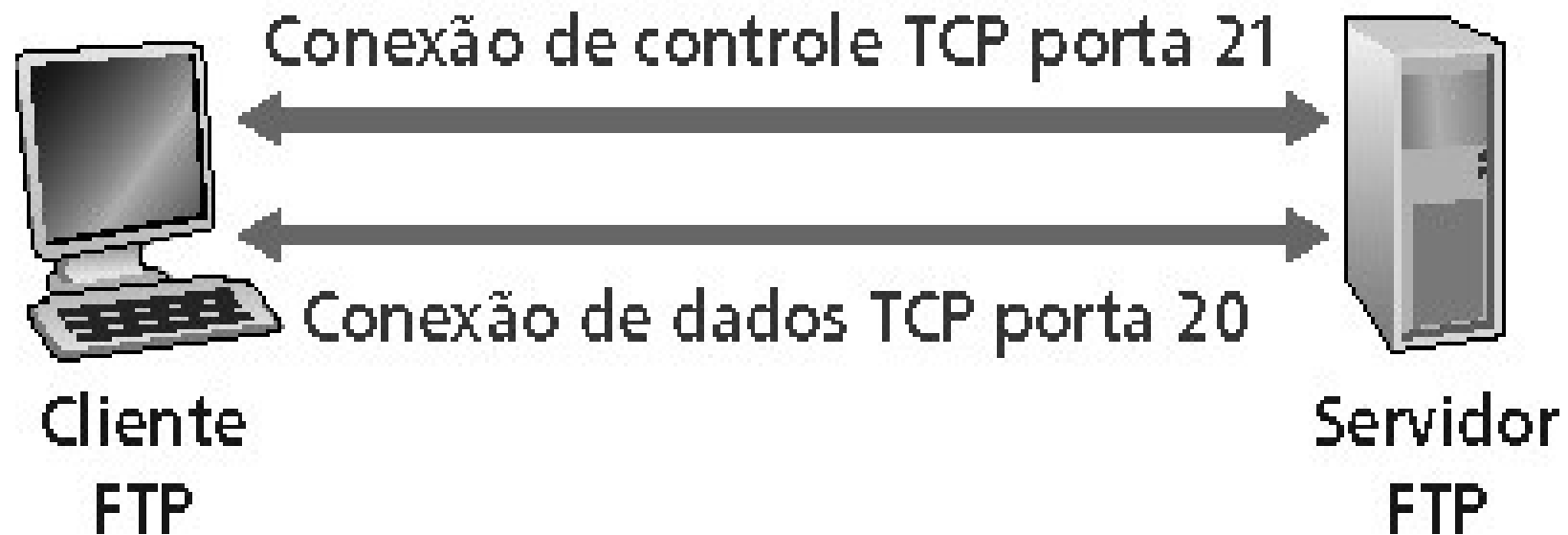




FTP: conexão de controle e de dados

Duas conexões:
controle;
dados.

Conexão de controle: “fora da banda”.



FTP: conexão de controle e de dados

Cliente FTP contata o servidor FTP na porta 21.

TCP como protocolo de transporte;
estabelece conexão de controle.

Conexão de Controle:
cliente obtém autorização;
cliente procura o diretório remoto;
cliente envia comando para transferência
de arquivo.

FTP: conexão de controle e de dados

Conexão de Dados:

servidor abre essa conexão TCP quando recebe um comando para transferência de arquivo;

após a transferência de um arquivo, o servidor fecha a conexão.

servidor abre uma segunda conexão de dados TCP para transferir outro arquivo;

Servidor FTP mantém “estado”: diretório atual, autenticação anterior.



FTP: Comandos

texto ASCII sobre canal de controle.

USER *username*

PASS *password*

LIST retorna listagem do arquivo no diretório atual.

RETR filename recupera (obtém) o arquivo.

STOR filename armazena o arquivo no hospedeiro remoto.



FTP: Códigos de retorno

Código de status e frase (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

Mas hoje....

FTP não é mais tão usado...

senhas passam em aberto na rede!

SCP:

somente transferência de arquivos.

SFTP (SSH File Transfer Protocol):

permite transferência e manipulação de arquivos;

normalmente usa SSH-2;

autenticação e segurança provida pelo protocolo abaixo (SSH).



EACH

Escola de Artes, Ciências e Humanidades
da Universidade de São Paulo

Capítulo 2.4 - Correio Eletrônico



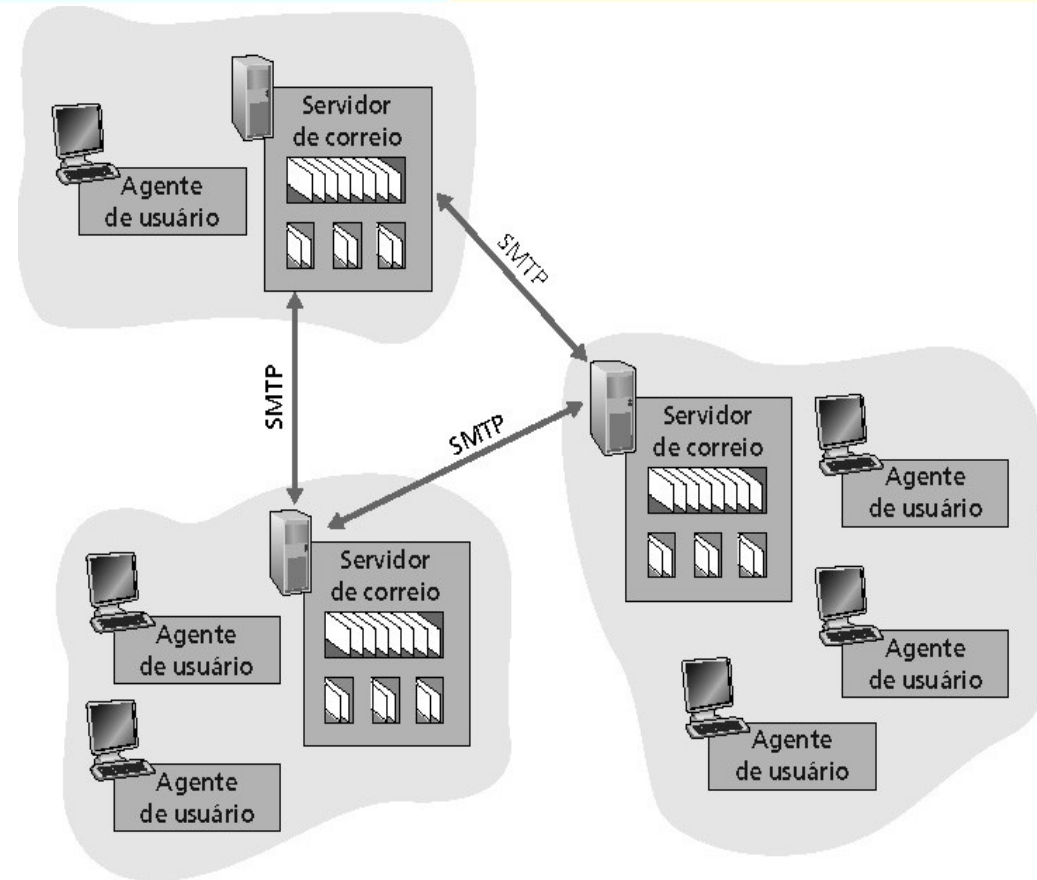
Correio Eletrônico

Três componentes principais:

Agentes de usuário

Servidores de correio

SMTP (Simple Mail Transfer Protocol)



Key:

 Outgoing message queue

 Caixa postal do usuário

Correio eletrônico: agentes de usuário

Composição, edição, leitura de mensagens de correio.

Ex.: Outlook, Thunderbird, Eudora, pine.

Mensagens de entrada e de saída são armazenadas no servidor.

alguns programas copiam as mensagens recebidas para o disco local (configuração).

Correio eletrônico: servidores de correio

Caixa postal contém mensagens que chegaram (ainda não lidas) para o usuário.

Fila de mensagens contém as mensagens de correio a serem enviadas.

Protocolo SMTP permite aos servidores de correio trocarem mensagens entre si:

cliente: servidor de correio que envia;

“servidor”: servidor de correio que recebe.

SMTP [RFC 821]

RFC é de 1982.

Usa TCP.

Servidor ouve na porta 25.

Conexões persistentes.

Transferência direta:

servidor que envia para o servidor que recebe.

Agentes de usuário usam o SMTP para enviar mensagens para o seu servidor.

SMTP [RFC 821]

Três fases de transferência:
handshaking (apresentação);
transferência de mensagens;
fechamento.

Interação comando/resposta:

Comandos: texto ASCII;

Resposta: código de *status* e frase.

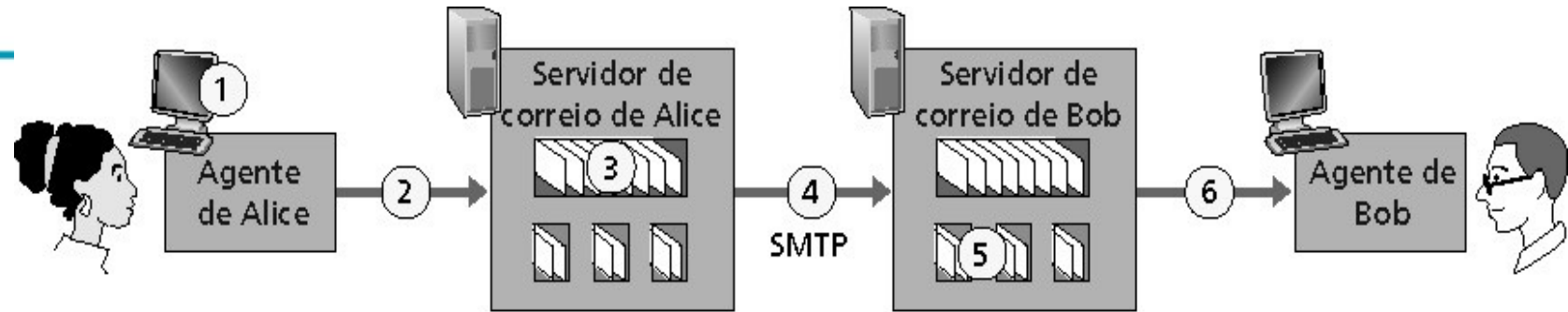
Mensagens (cabeçalho e corpo) devem ser formatadas em código ASCII de 7 bits.



EACH

Escola de Artes, Ciências e Humanidades
da Universidade de São Paulo

Alice envia email para Bob



Legenda:



Fila de mensagens



Caixa postal do usuário

- 1) Alice usa o agente de usuário para compor a mensagem para bob@school.edu
- 2) O agente de usuário dela envia a mensagem para o seu servidor de correio; a mensagem é colocada na fila de mensagens.
- 3) O lado cliente do SMTP abre uma conexão TCP com o servidor de correio do Bob.
- 4) O cliente SMTP envia a mensagem de Alice pela conexão TCP.
- 5) O servidor de correio de Bob coloca a mensagem na caixa de correio de Bob.
- 6) Bob invoca seu agente de usuário para ler a 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: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Tente você...

telnet nome-do-servidor 25

Veja resposta 220 do servidor.

Envie comandos HELO, MAIL FROM, RCPT TO, DATA, QUIT.

A sequência acima permite enviar um comando sem usar o agente de usuário do remetente.



```
cintia@PI12PROF093:~$ telnet smtp.usp.br 25
Trying 143.107.254.72...
Connected to colorado.uspnet.usp.br.
Escape character is '^]'.
220 colorado.uspnet.usp.br ESMTP
HELO margi.com
250 colorado.uspnet.usp.br
MAIL FROM: <cintia@margi.com>
250 ok
RCPT TO: <cintia@usp.br>
250 ok
DATA
354 go ahead punk, make my day
Subject: testando telnet
Exemplo para aula. Email enviado direto ma porta 25.
Obrigada!
.
250 ok 1188236528 qp 18103 by colorado.uspnet.usp.br
QUIT
221 colorado.uspnet.usp.br Goodbye.
Connection closed by foreign host.
cintia@PI12PROF093:~$
```



SMTP: comparação com HTTP

HTTP: protocolo de recuperação de informações (*pull protocol*).

SMTP: protocolo de envio de informações (*push protocol*).

Ambos usam comandos e respostas em ASCII, interação comando/resposta e códigos de *status*.

HTTP: cada objeto encapsulado na sua própria mensagem de resposta.

SMTP: múltiplos objetos são enviados numa mensagem multiparte.



Formato da Mensagem

RFC 822: padrão para
mensagens do tipo
texto:

linhas de cabeçalho:

To:

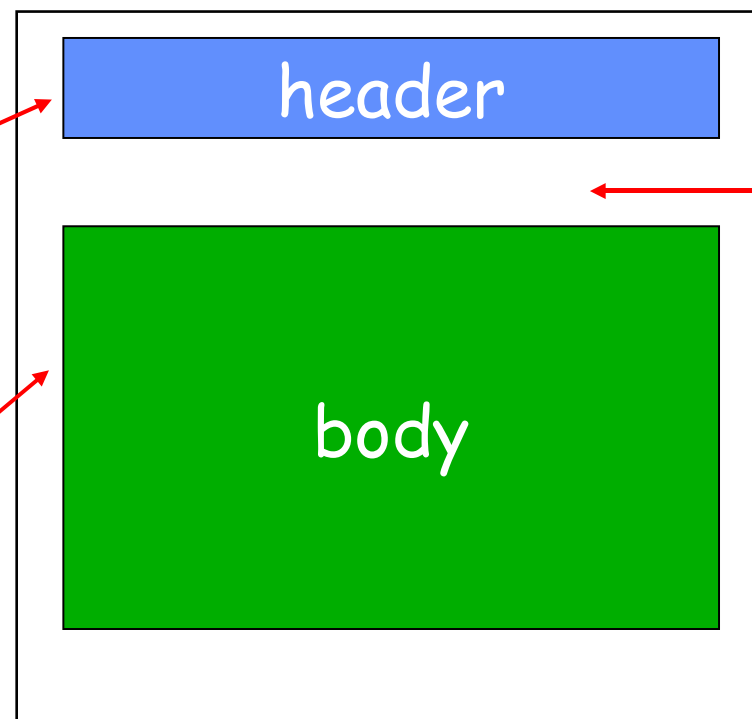
From:

Subject:

diferente dos
comandos *SMTP!*

corpo:

a “mensagem”, ASCII
somente com
caracteres.



linha em
branco

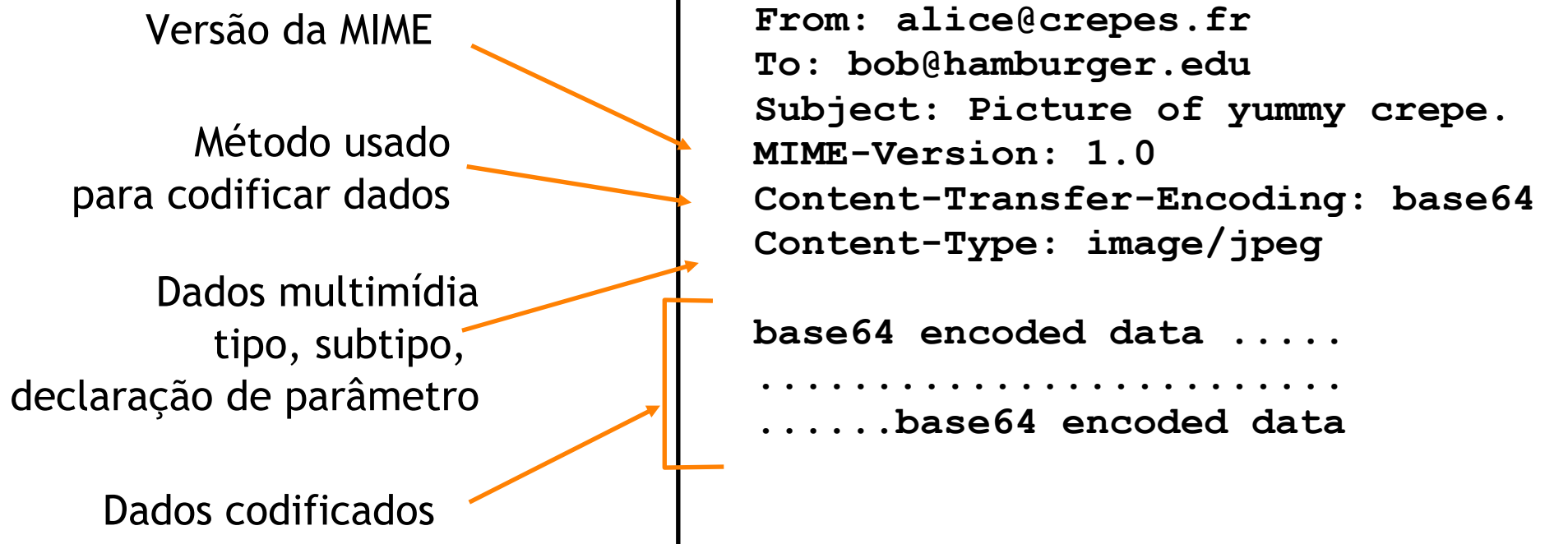


Formato da Mensagem: extensões multimídia

MIME: multimedia mail extension, RFC 2045, 2056.

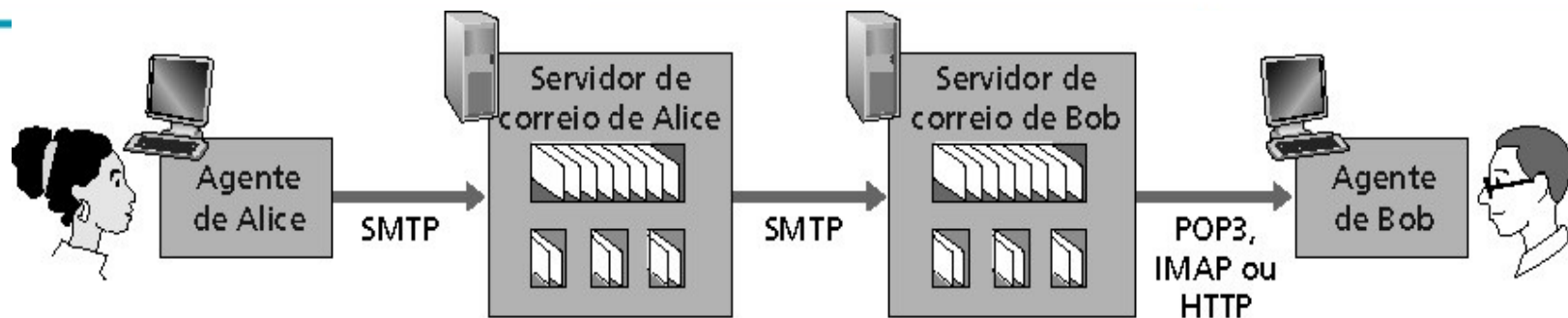
Linhas adicionais no cabeçalho declaram o tipo de conteúdo MIME.

Dados multimídia: codificados em ASCII de 7bits!





Protocolos de Acesso ao Correio



SMTP: entrega e armazena no servidor do destino.

Protocolo de acesso: recupera mensagens do servidor:

POP: Post Office Protocol [RFC 1939]

Autorização (agente<-->servidor) e download.

IMAP: Internet Message Access Protocol [RFC 2060]

mais recursos (mais complexo);

manipulação de mensagens armazenadas no servidor.

HTTP: Hotmail, Yahoo! Mail etc.

Protocolo POP3

Fase de autorização

comandos do cliente:

- **user**: declara nome do usuário

- **pass**: password

respostas do servidor

- **+OK**
- **-ERR**

Fase de transação, cliente:

- **list**: lista mensagens e tamanhos
- **retr**: recupera mensagem pelo número
- **dele**: apaga
- **quit**

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

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

POP3

O exemplo anterior usa o modo “ler e apagar” (*download-and-delete*).

O usuário não pode reler o e-mail se trocar o cliente.

“ler e guardar” (*download-and-keep*):
mantém a mensagem no servidor;
cópias das mensagens em clientes diferentes.

POP3 é não mantém estado através das sessões.

IMAP

Mantém todas as mensagens em um lugar:
o servidor.

Permite que o usuário:
crie pastas;

movam as mensagens do Inbox para pastas.

IMAP mantém o estado do usuário através
das sessões:

Nomes das pastas e mapeamentos entre os IDs
da mensagem e o nome da pasta.

Webmail

Agente de usuário: *browser*.

Protocolo entre agente de usuário e servidor: HTTP.

Algumas implementações utilizam um servidor IMAP:

scripts no servidor HTTP usam o protocolo IMAP para se comunicar com servidor IMAP;

vantagem: funcionalidades do IMAP.



EACH

Escola de Artes, Ciências e Humanidades
da Universidade de São Paulo

Cap. 2.6 - Compartilhamento de Arquivos P2P

Protocolos de compartilhamento de arquivos em redes *peer-to-peer*.

Outros aspectos não discutidos aqui, mas importantes:

segurança;

privacidade;

anonimato;

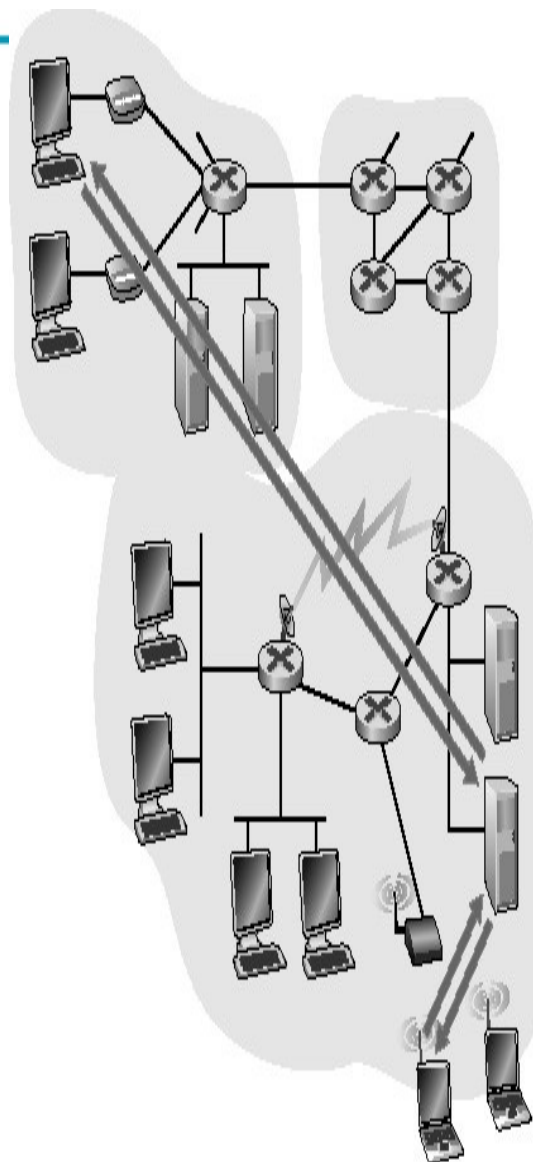
violação de direitos autorais e propriedade intelectual.



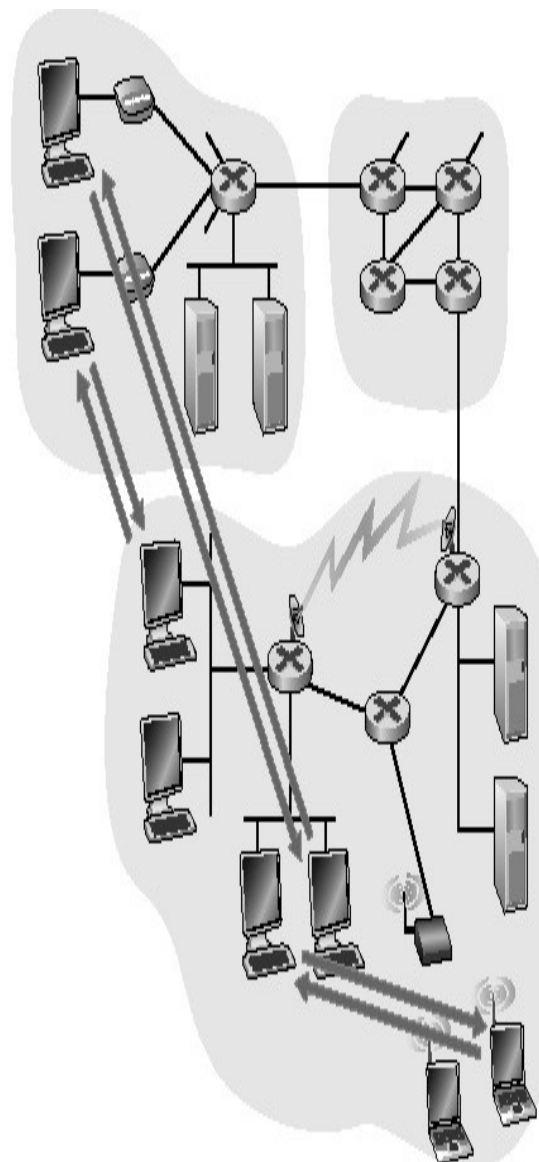
EACH

Escola de Artes, Ciências & Humanidades
da Universidade de São Paulo

Arquitetura P2P (pura)



a. Aplicação cliente-servidor



b. Aplicação P2P

Compartilhamento de Arquivos P2P

Alice executa aplicação P2P:
utiliza ADSL e obtém novos endereços IP para
cada conexão.

procura música “Hey Jude” (arquivo MP3);

a aplicação exibe outros pares que possuem uma
cópia de “Hey Jude”;

Alice escolhe um dos pares, Bob;

o arquivo é copiado de Bob para Alice.

Enquanto Alice faz o download, é possível
fazer upload de arquivos em Alice.

Compartilhamento de Arquivos P2P

Transferência de arquivos entre pares usa
HTTP!

Par que possui o arquivo é um **servidor
web transitório!**

Problema chave:
localização de arquivos!

P2P: Diretório centralizado

Localização de conteúdo através de diretório central.

Ex.: “Napster”

<http://pt.wikipedia.org/wiki/Napster>

Solução híbrida:

cliente/servidor: localização de conteúdo;

P2P: transferência de arquivos.



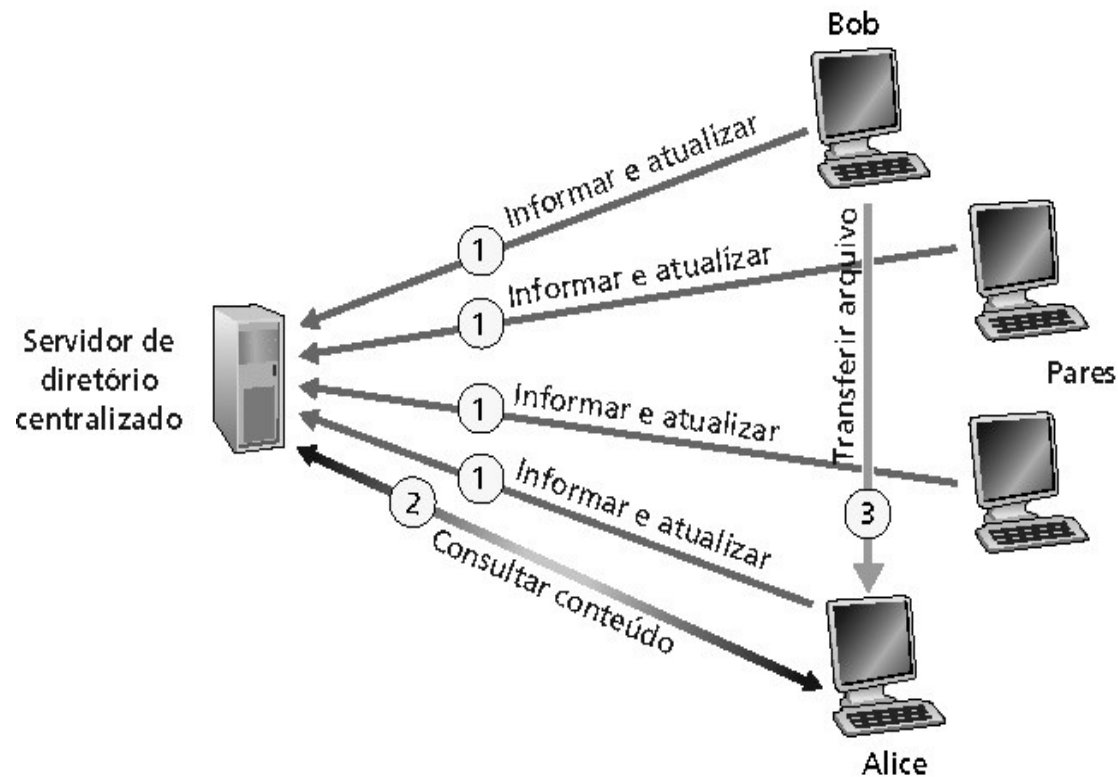
P2P: Diretório centralizado

Quando um par se conecta, ele informa ao servidor central: endereço IP e conteúdo.

Usuário procura arquivo no servidor.

Usuário requisita o arquivo do par que o contém.

Servidor precisa verificar se usuários estão conectados.



P2P: Diretório centralizado - Problemas

Ponto único de falhas.

Gargalo de desempenho.

Violação de direitos autorais:

sanções judiciais podem levar ao
desligamento dos servidores de diretório.

**Transferência de arquivo é
descentralizada, mas a localização de
conteúdo é altamente centralizada.**

P2P: Inundação de consultas - Gnutella

Totalmente distribuído:
Sem servidor central.

Protocolo de domínio público:
<http://rfc-gnutella.sourceforge.net/developer/stable/index.html>

Muitos clientes Gnutella implementam o protocolo.

Gnutella: Rede de sobreposição

Aresta entre X e o Y se há uma conexão
TCP entre X e Y.
aresta é diferente de enlace!

Rede de sobreposição: grafo que
contém todos os pares ativos e arestas.

Um determinado par (*peer*) está
tipicamente conectado a <10 vizinhos na
rede de sobreposição.

Gnutella: Protocolo

O par já pertence a rede de sobreposição.

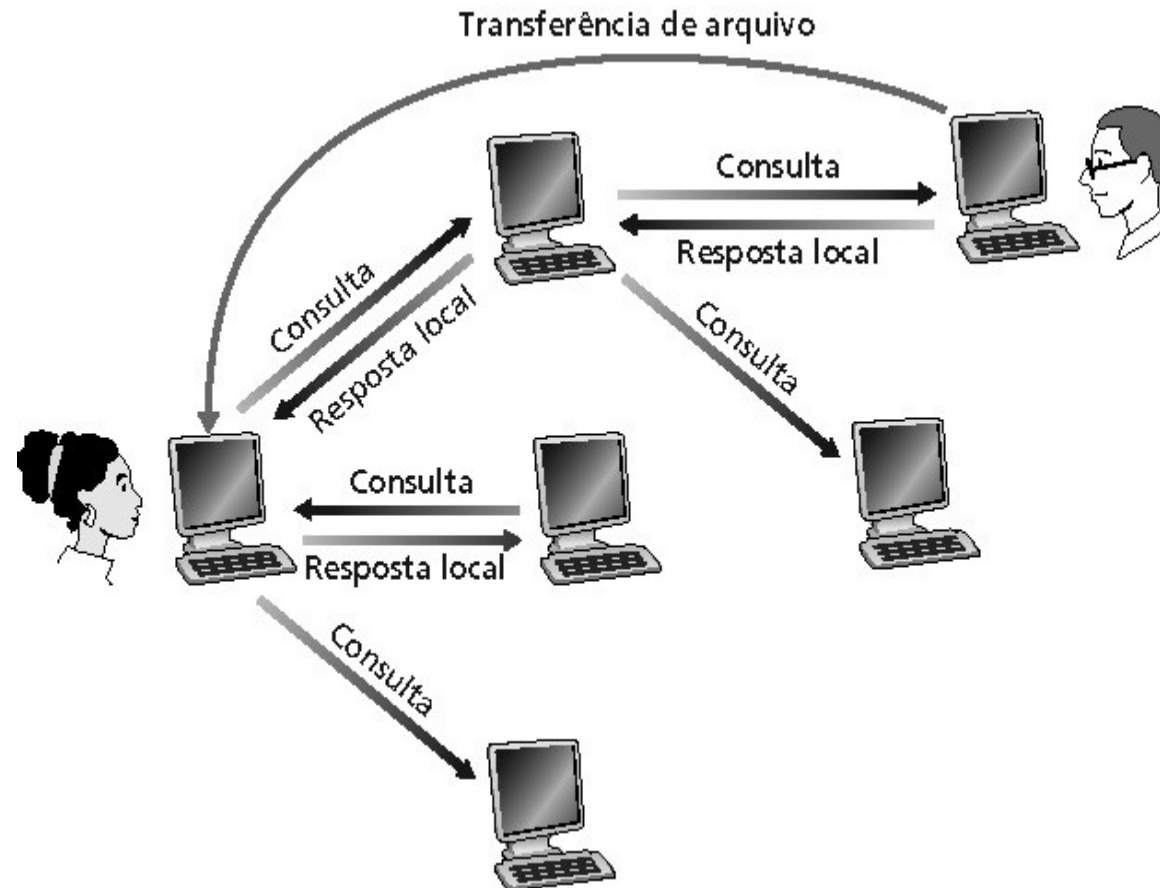
Mensagem de consulta (query) é enviada pelas conexões TCP existentes.

Os pares encaminham a mensagem de consulta através de suas arestas (inundação de consultas!).

Quando par possui arquivo contendo palavra chave, envia uma mensagem QueryHit (encontro) pelo caminho reverso.



Gnutella: funcionamento



Transferência de arquivo não usa rede de sobreposição!
Comandos HTTP (GET e resposta) diretamente entre pares.

Gnutella: otimizações

Inundação de consultas gera tráfego na Internet! -> não é escalável!

Solução: inundação de consultas de escopo limitado:

mensagem de consulta tem contador de pares máximo;

porém limita busca a pares próximos.

QueryHit: enviado via UDP.

Push request: para clientes bloqueados por firewall.



Gnutella: conectando pares

1. X precisa encontrar algum outro par na rede Gnutella:
cliente Gnutella possui uma lista de pares frequentemente ativos;
contata cache Gnutella que tenha lista.
2. X tenta estabelecer conexão TCP com os pares da lista:
atualiza lista quando não consegue estabelecer conexão.



Gnutella: conectando pares (cont.)

3. Quando estabelece conexão com Y, X envia mensagem de Ping.
Y encaminha a mensagem de Ping.
4. Pares que recebem a mensagem de Ping respondem com mensagens de Pong.
5. X recebe várias mensagens de Pong, e estabelece conexões TCP adicionais.

P2P: Napster & Gnutella

Abordagens opostas:

Napster: centralizado;

Gnutella: totalmente distribuído.

Como tirar vantagem de ambos?

KaZaA:

utiliza o protocolo FastTrack (proprietário);

criptografa o tráfego de controle;

não utiliza servidor (como o Gnutella);

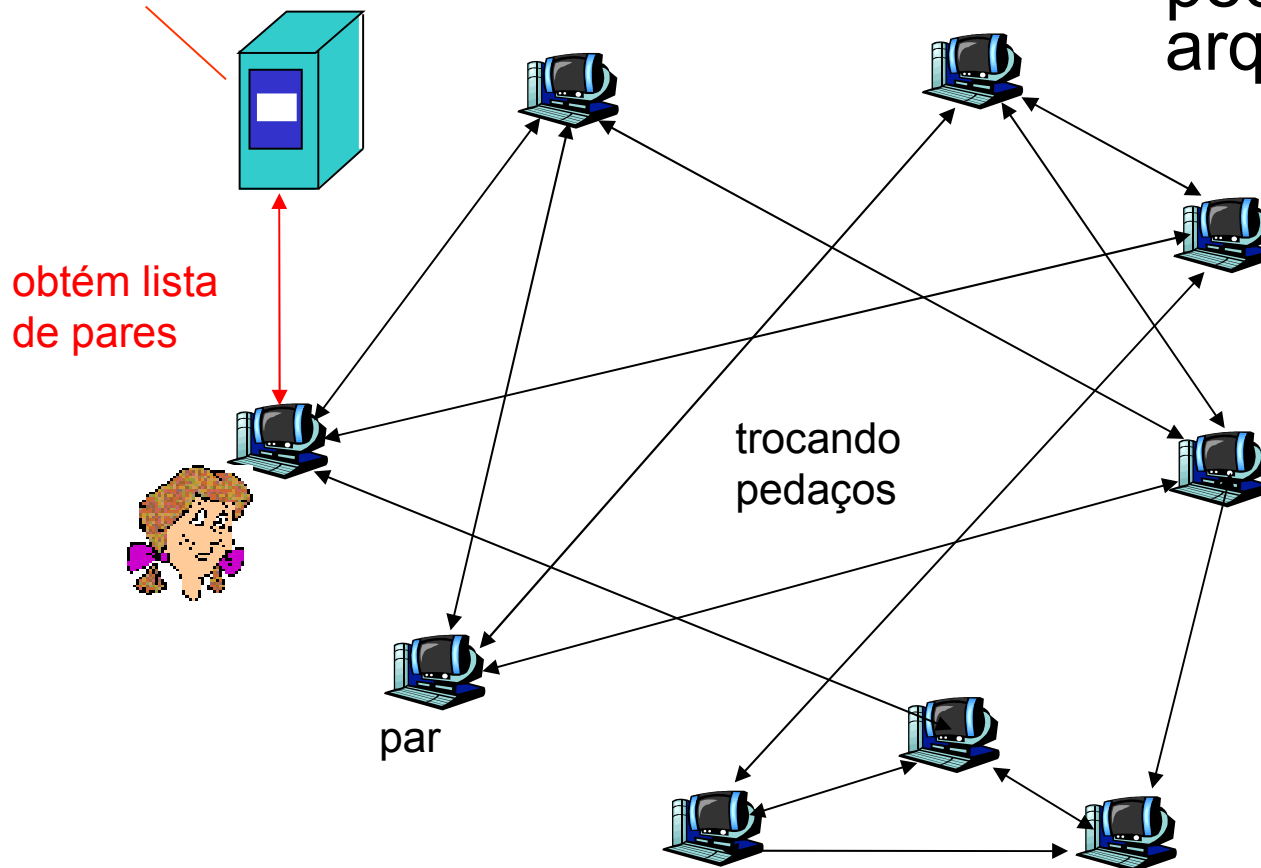
nem todos os pares são iguais!

Distribuição de arquivos: BitTorrent

- distribuição de arquivos P2P

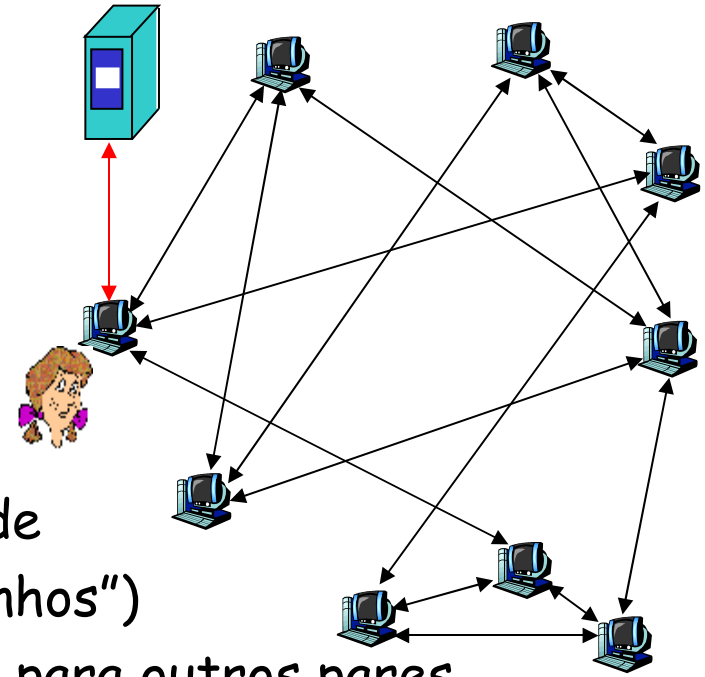
rastreador: verifica pares que participam do torrent

torrent: grupo de pares trocando pedaços de um arquivo



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



Bit Torrent (cont.)

Empurrando pedaços

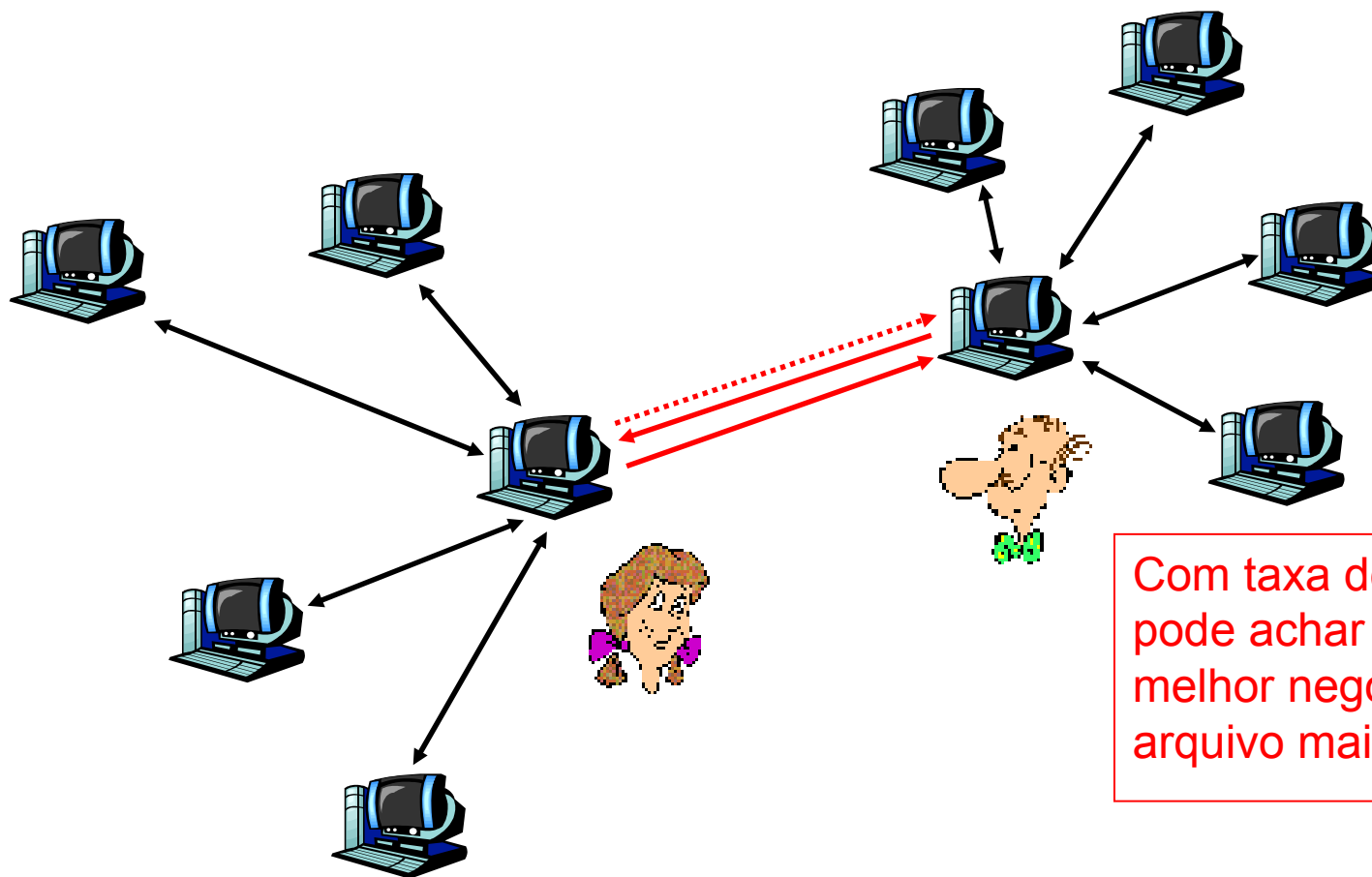
- a qualquer momento, diferentes pares têm diferentes subconjuntos de pedaços de arquivo
- periodicamente, um par (Alice) pede a cada vizinho a lista de pedaços que eles têm
- Alice envia requisições para seus pedaços que faltam
 - ❖ mais raros primeiro

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: número ss; valor: nome humano
 - ❖ chave: tipo conteúdo; valor: endereço IP
- pares consultam BD com chave
 - ❖ BD retorna valores que combinam com a chave
- pares também podem inserir duplas (chave, valor)

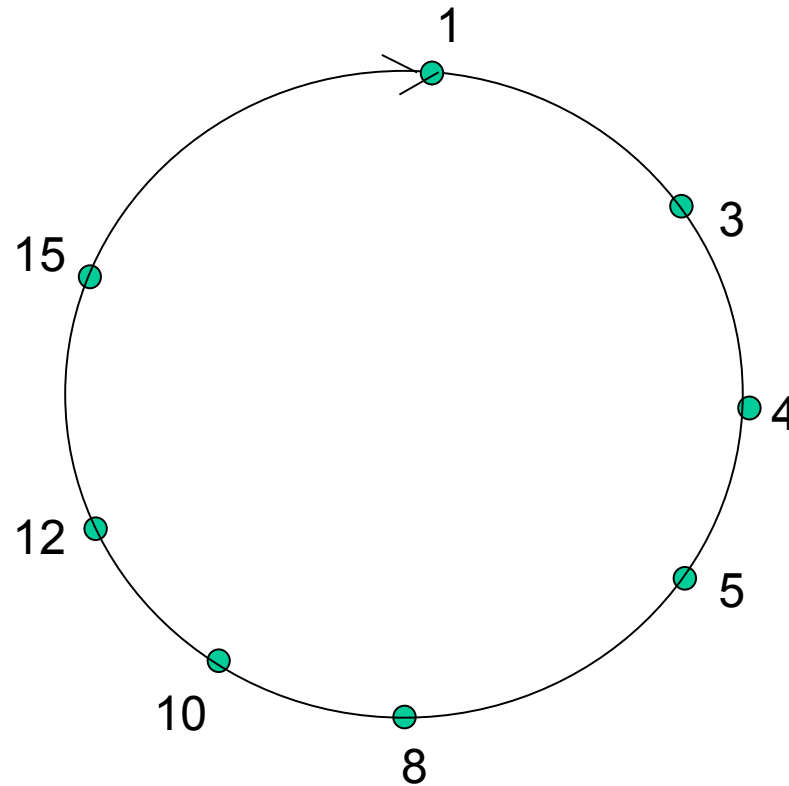
Identificadores DHT

- atribuem identificador inteiro a cada par no intervalo $[0, 2^n - 1]$.
 - ❖ cada identificador pode ser representado por n bits.
- exigem que cada chave seja um inteiro no mesmo intervalo.
- para obter chaves inteiras, misture chave original.
 - ❖ p. e., chave = $h(\text{"Led Zeppelin IV"})$
 - ❖ É por isso que a chamamos de tabela "hash" distribuída

Como atribuir chaves aos pares?

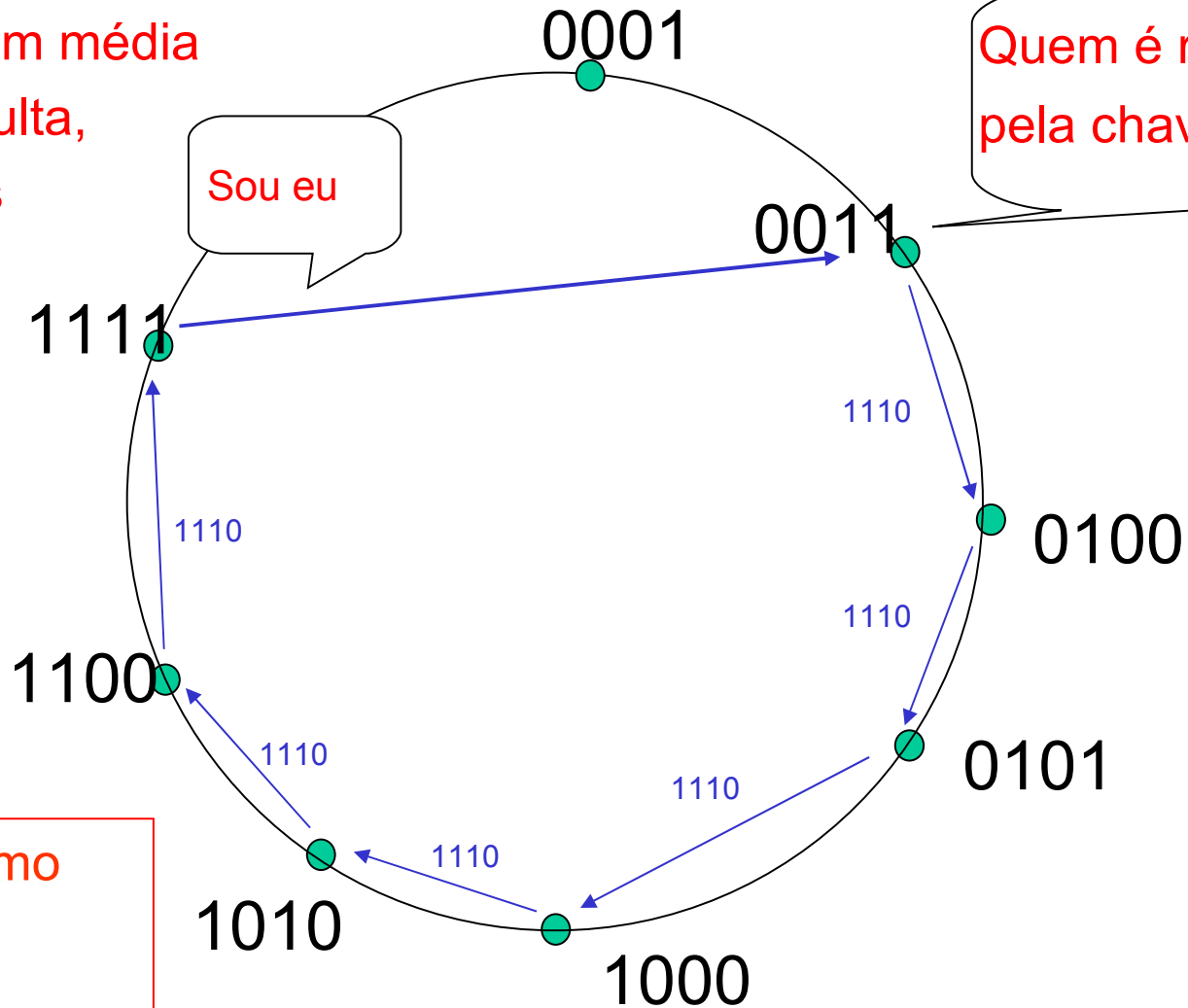
- ❑ questão central:
 - ❖ atribuir duplas (chave, valor) aos pares.
- ❑ regra: atribuir chave ao par que tem o ID **mais próximo**.
- ❑ convenção na aula: mais próximo é o **sucessor imediato** da chave.
- ❑ ex.: $n = 4$; pares: 1,3,4,5,8,10,12,14;
 - ❖ chave = 13, então par sucessor = 14
 - ❖ chave = 15, então par sucessor = 1

DHT circular



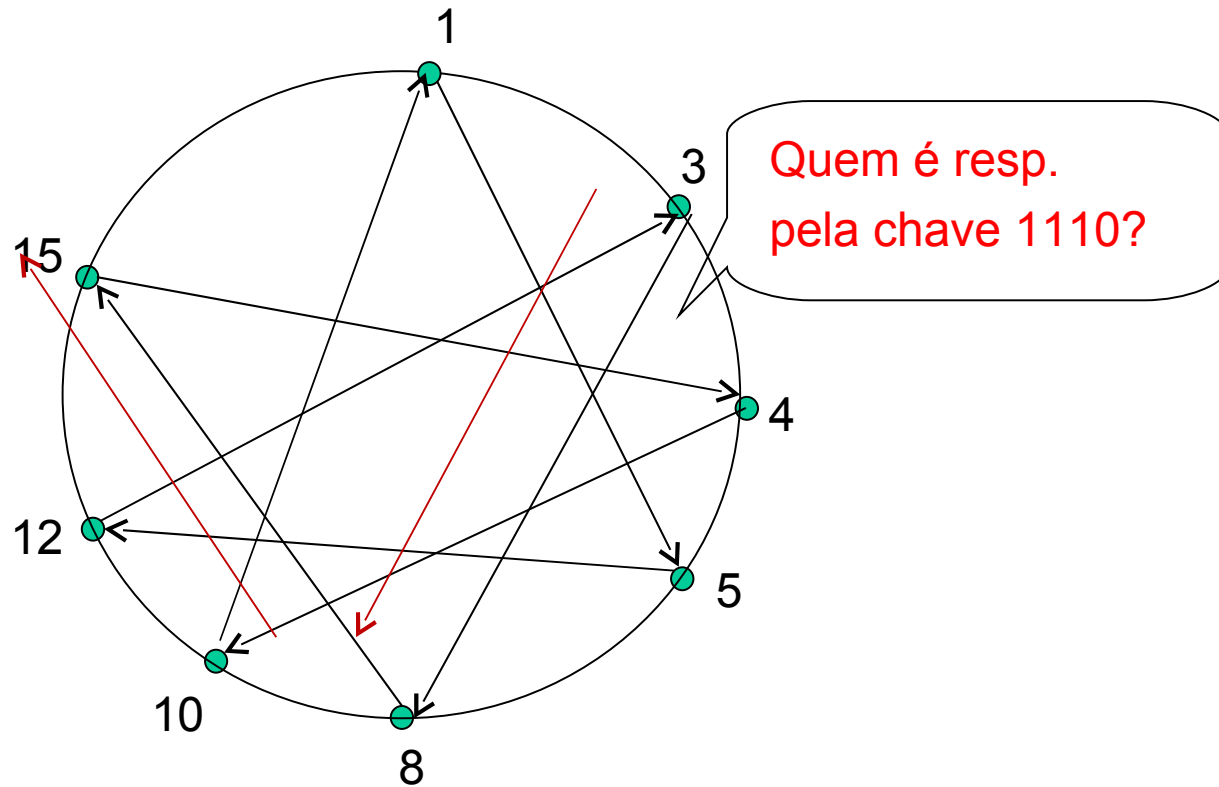
- ❑ cada par *só* conhece sucessor e predecessor imediato.
- ❑ "rede de sobreposição"

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



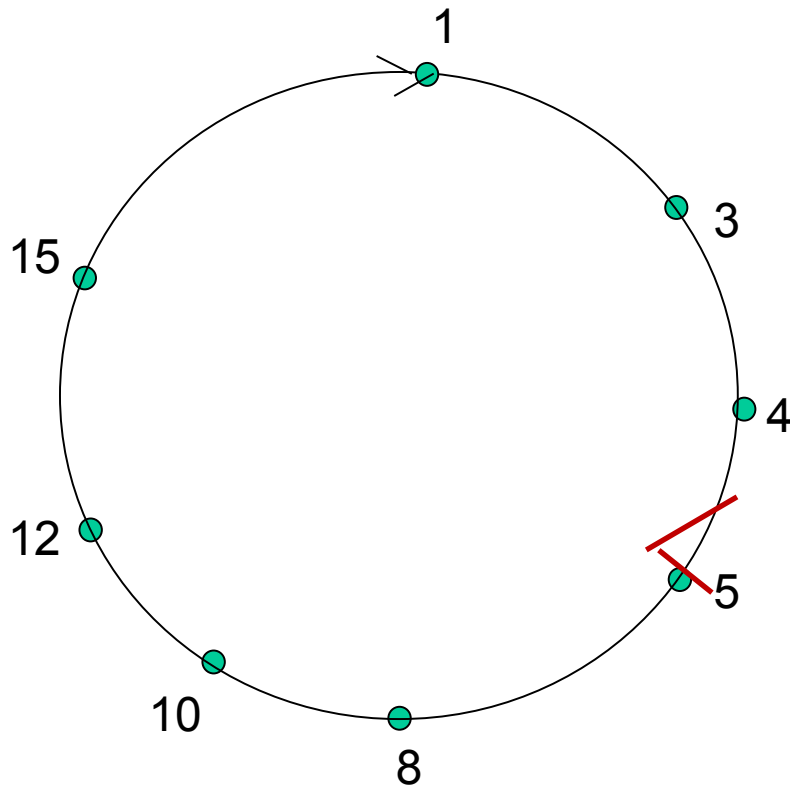
Define mais próximo
como sucessor
mais próximo

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

Peer Churn

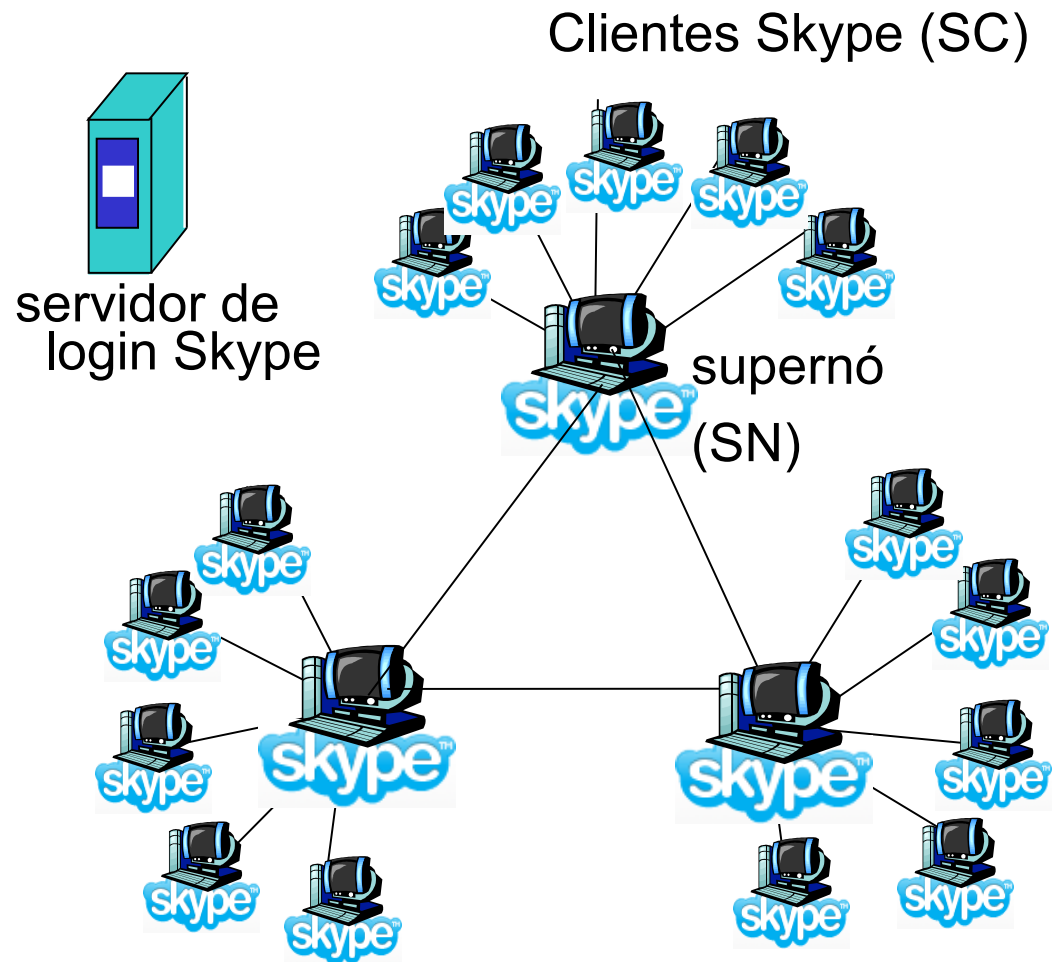


- para manejar o peer churn, é preciso que cada par conheça o endereço IP de seus dois sucessores.
- cada par periodicamente envia 'ping' aos seus dois sucessores para ver se eles ainda estão vivos.

- ❑ par 5 sai abruptamente
- ❑ par 4 detecta; torna 8 seu sucessor imediato; pergunta a 8 quem é seu sucessor imediato; torna o sucessor imediato de 8 seu segundo sucessor.
- ❑ e se o par 13 quiser se juntar?

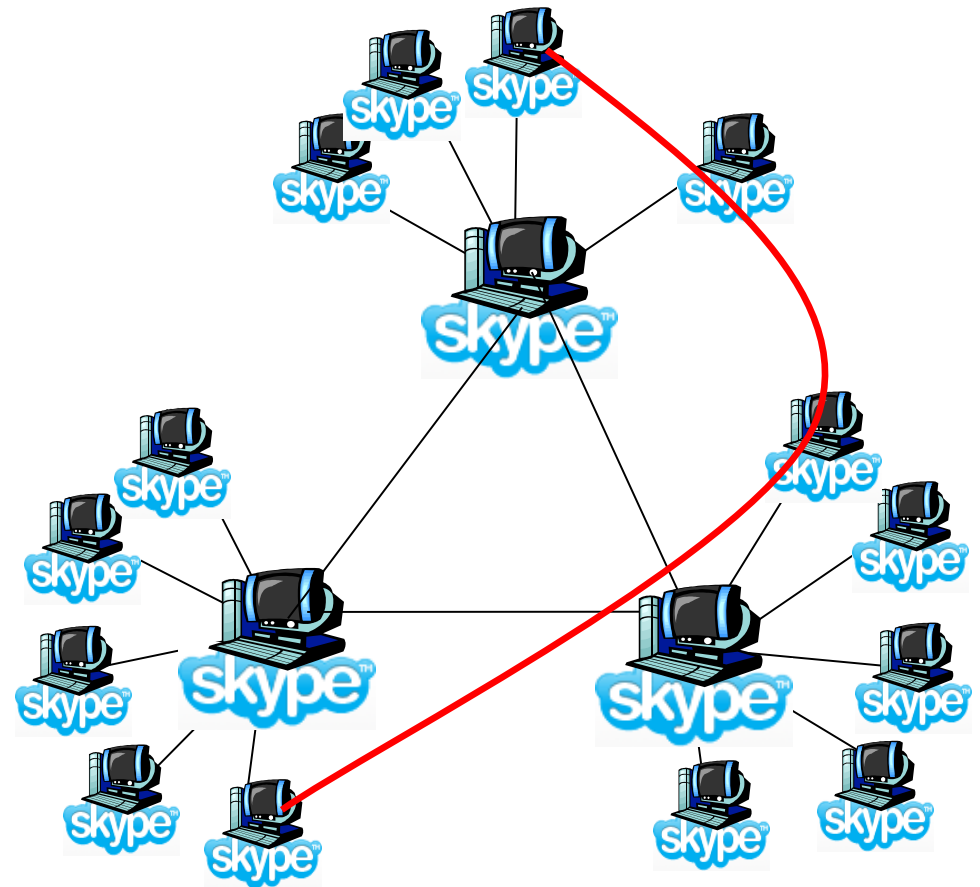
Estudo de caso do P2P: Skype

- ❑ 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
- solução:
 - ❖ usando os SNs de Alice e de Bob, o retransmissor é escolhido
 - ❖ cada par inicia a sessão com retransmissão.
 - ❖ pares agora podem se comunicar através de NATs com retransmissão





EACH

Escola de Artes, Ciências e Humanidades
da Universidade de São Paulo

Capítulo 2.7 e 2.8 - Camada de Aplicação

Programação de sockets

Programação de Sockets

Objetivo:

aprender a construir aplicações cliente-servidor que se comunicam usando sockets.

Socket:

interface local, criada por aplicações, controlada pelo OS (uma “porta”) na qual os processos de aplicação podem tanto enviar quanto receber mensagens de e para outro processo de aplicação (local ou remoto).

API Socket

Introduzida no BSD4.1 UNIX, 1981.

Sockets são explicitamente criados, usados e liberados pelas aplicações.

Implementam paradigma cliente-servidor.

Dois tipos de serviço de transporte via socket API:

datagrama não confiável (UDP);

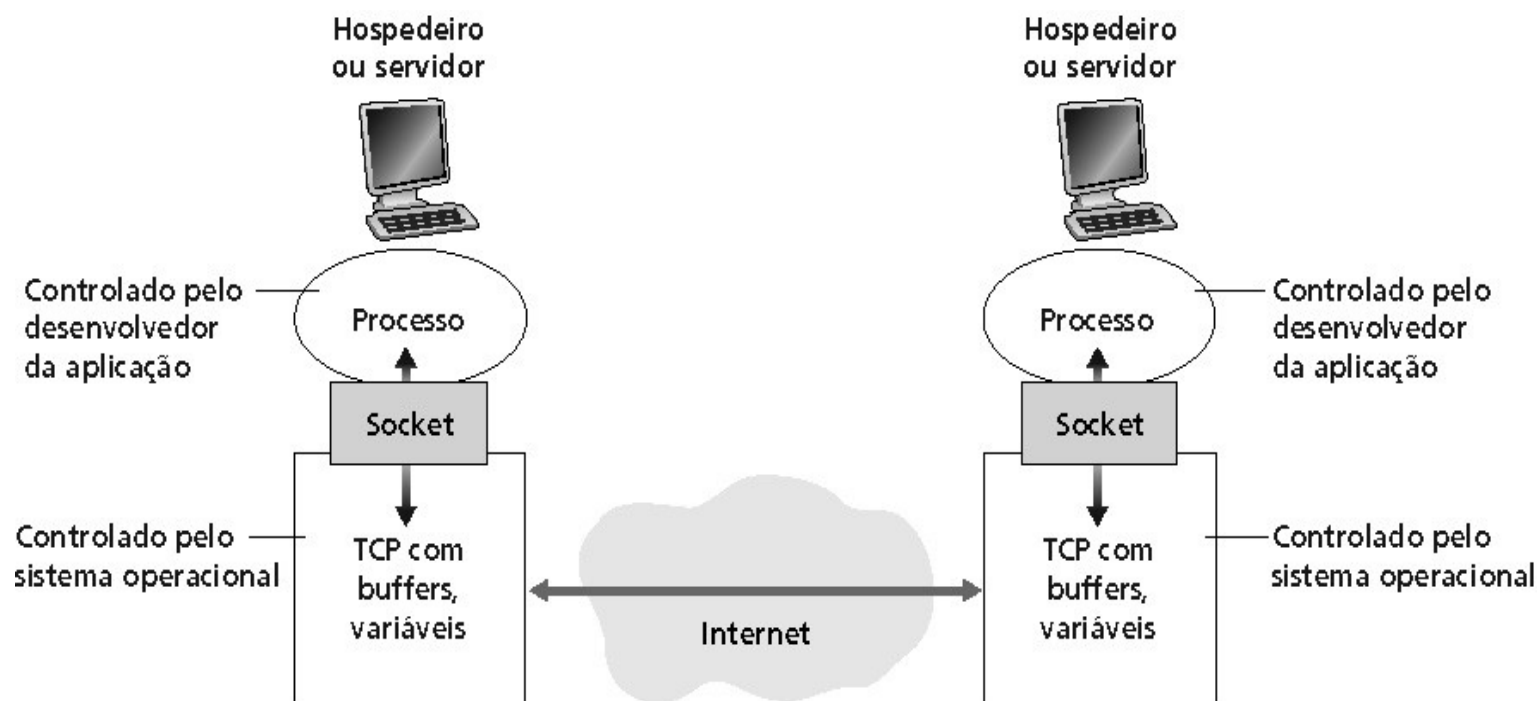
confiável, orientado a cadeias de bytes (TCP).



Programação de sockets com TCP

Socket: uma porta entre o processo de aplicação e o protocolo de transporte fim-a-fim.

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



Sockets TCP - Cliente

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.

Sockets TCP - Servidor

Quando contatado pelo cliente, o 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).



Terminologia: *stream* (cadeia)

Um *stream* (cadeia) é uma seqüência de caracteres que fluem para dentro ou para fora de um processo.

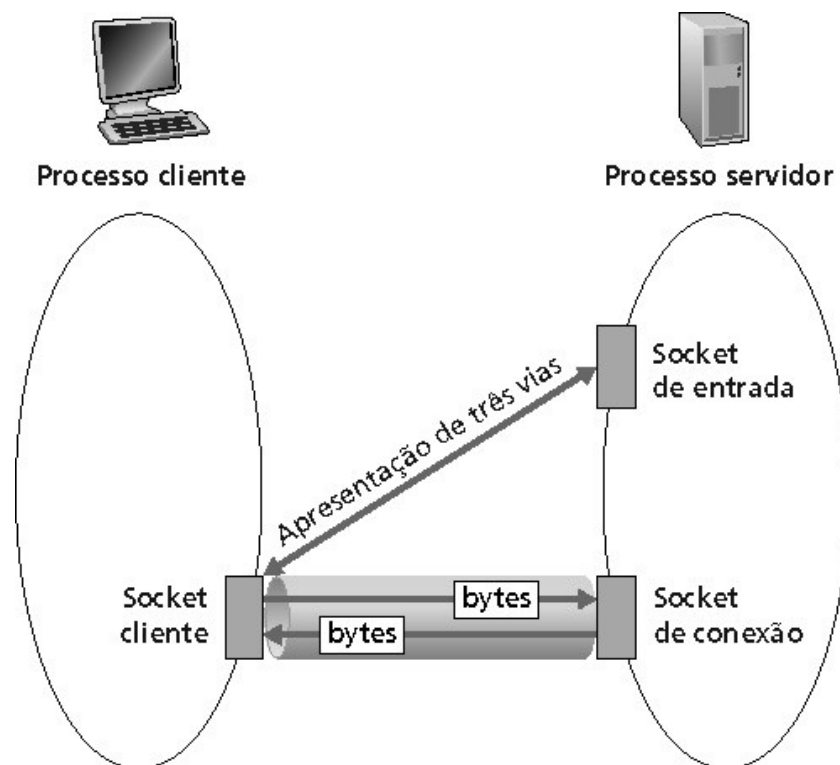
Um *stream* de entrada é agregado a alguma fonte de entrada para o processo, ex.: teclado ou socket.

Um *stream* de saída é agregado a uma fonte de saída, ex.: monitor ou socket.



Exemplo de aplicação cliente-servidor TCP

- 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).

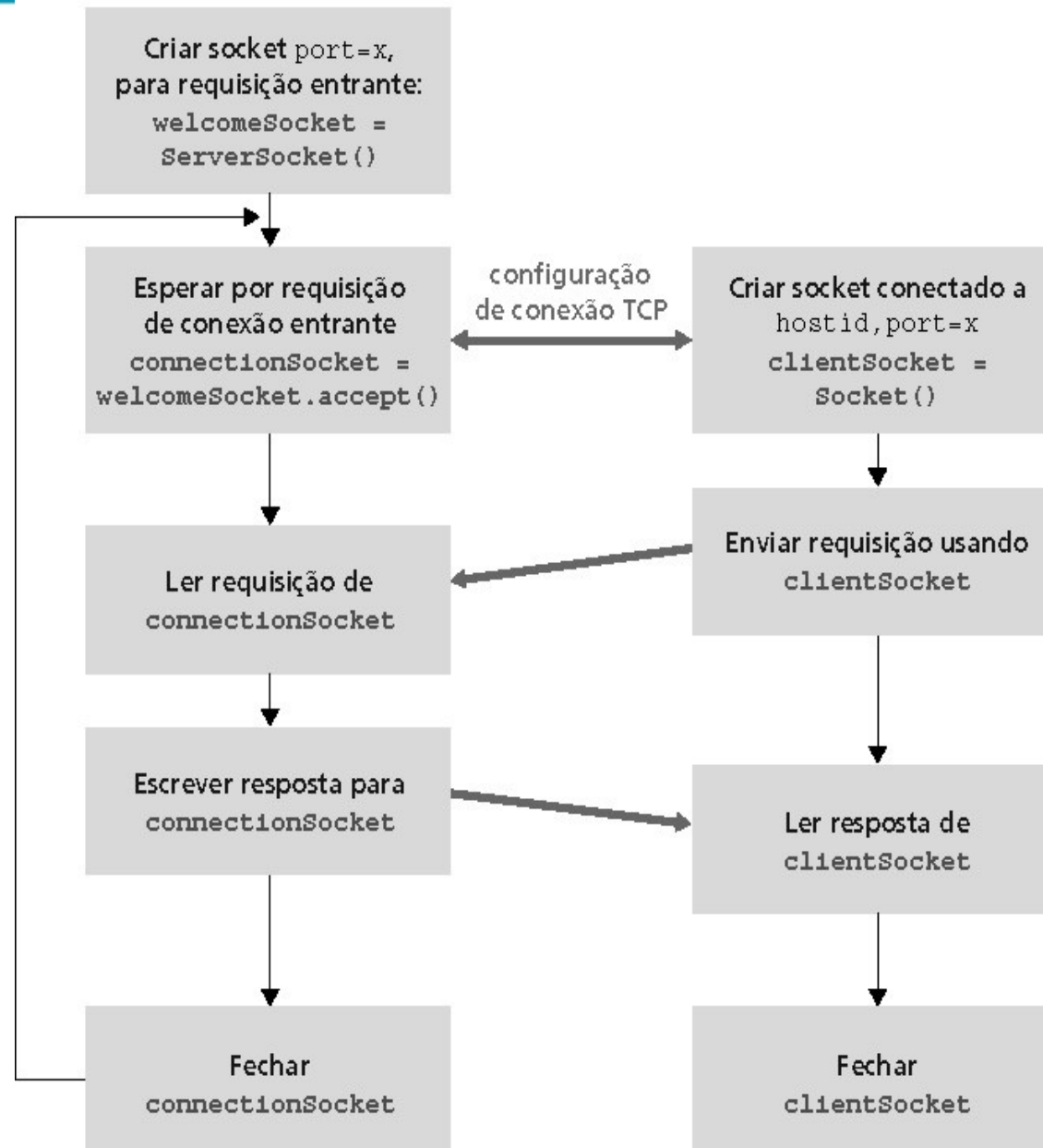




Interação cliente-servidor TCP

Servidor
(Rodando em hostid)

Cliente



Exemplo: cliente Java (TCP)

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

```
    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
saída conectada
ao socket

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

Exemplo: cliente Java (TCP)

cria cadeia de
entrada conectada
ao socket

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

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

envia linha
ao servidor

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

lê linha
do servidor

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

```
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()));
```

Exemplo: servidor Java (TCP)

```

    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);
            }
        }
    }

```

cria cadeia de saída, conectada ao socket

lê linha do socket

escreve linha no socket

fim do loop while, retorna e espera outra conexão do cliente

TCP - observações e perguntas

- ❑ servidor tem dois tipos de sockets:
 - ❖ ServerSocket e Socket
- ❑ 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?



Programação de sockets com UDP

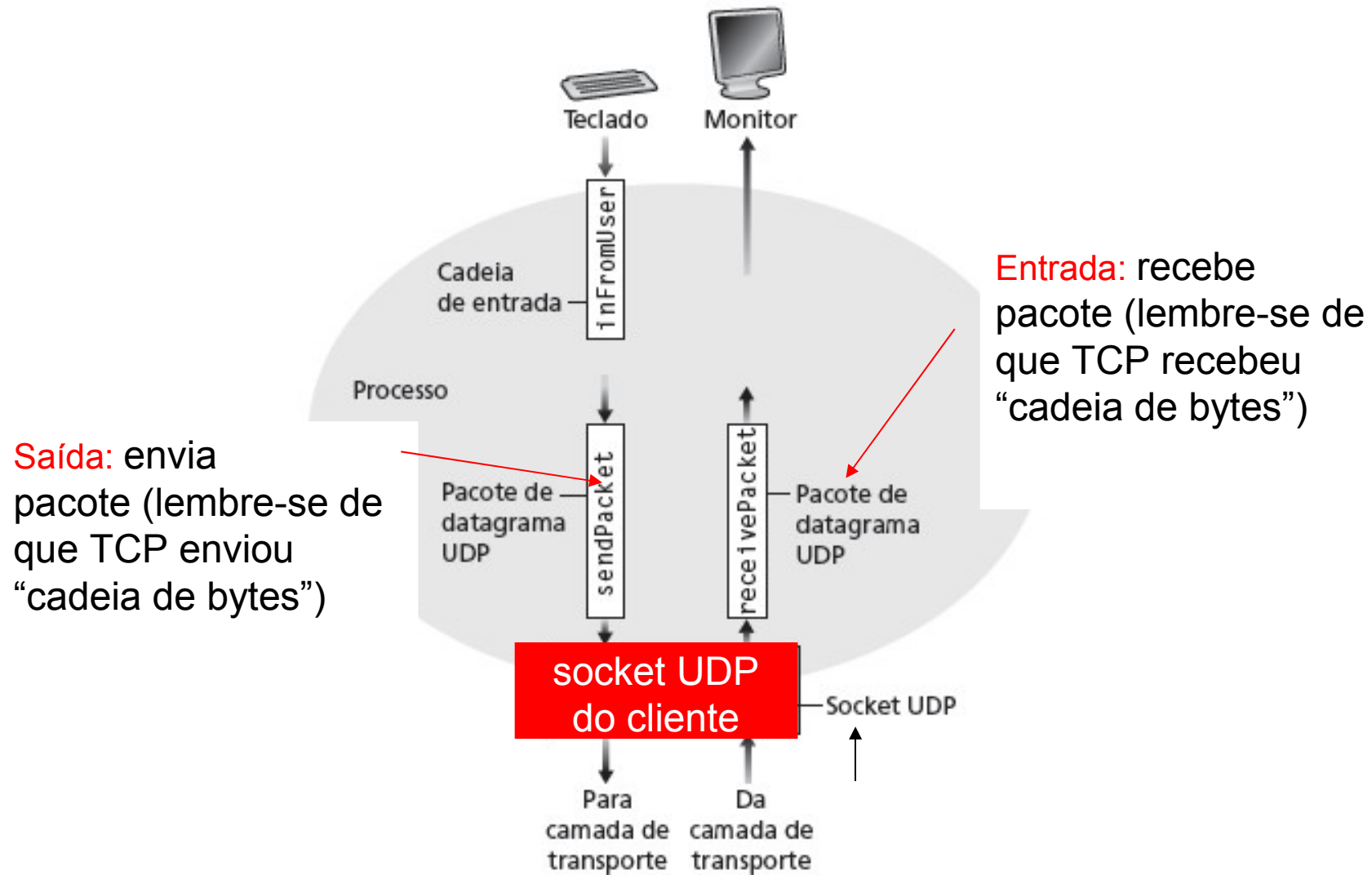
Não há conexão entre o cliente e o servidor.

Transmissor envia explicitamente endereço IP e porta de destino em cada mensagem.

Servidor deve extrair o endereço IP e porta do transmissor de cada datagrama recebido.

Dados transmitidos podem ser recebidos fora de ordem ou perdidos.

Exemplo: cliente Java (UDP)



Exemplo: cliente Java (UDP)

```
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();
```

Exemplo: cliente Java (UDP)

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);
```

lê datagrama
do servidor

```
DatagramPacket receivePacket =  
    new DatagramPacket(receiveData, receiveData.length);  
  
clientSocket.receive(receivePacket);  
  
String modifiedSentence =  
    new String(receivePacket.getData());  
  
System.out.println("FROM SERVER:" + modifiedSentence);  
clientSocket.close();  
}  
}
```

Exemplo: servidor Java (UDP)

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

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

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);
```

Exemplo: servidor Java (UDP)

```
String sentence = new String(receivePacket.getData());  
obtém end. IP  
# porta do } InetAddress IPAddress = receivePacket.getAddress();  
emissor    }  
           } int port = receivePacket.getPort();
```

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

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

```
cria datagrama p/ } DatagramPacket sendPacket =  
enviar ao cliente } new DatagramPacket(sendData, sendData.length, IPAddress,  
                  } port);
```

```
escreve }  
datagrama } serverSocket.send(sendPacket);  
no socket }  
          }
```

```
    }  
    } fim do loop while,  
      } retorna e espera  
      } outro datagrama
```



EACH

Escola de Artes, Ciências e Humanidades
da Universidade de São Paulo

EP: Implementação de um servidor Web

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



EACH

Escola de Artes, Ciências e Humanidades
da Universidade de São Paulo

Perguntas???
