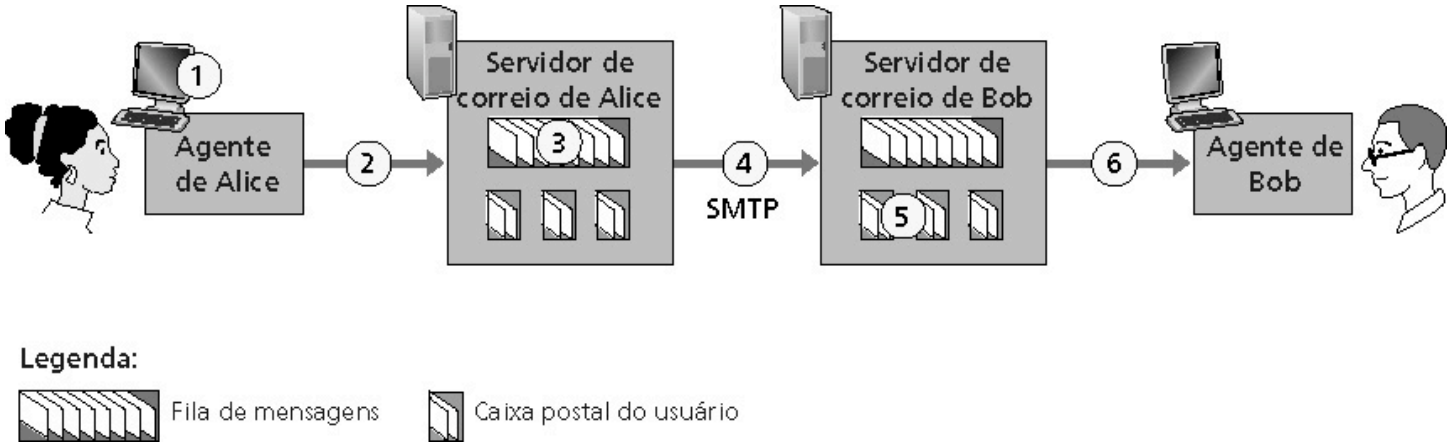
2 Correio eletrônico: SMTP [RFC 821]

- Usa TCP para transferência confiável de mensagens de correio do cliente ao servidor, porta 25
- Transferência direta: servidor que envia para o servidor que recebe
- Três fases de transferência
  - Handshaking (apresentação)
  - Transferência de mensagens
  - Fechamento
- Interação comando/resposta
  - Comandos: texto ASCII
  - Resposta: código de *status* e frase
- Mensagens devem ser formatadas em código ASCII de 7 bits

2 Cenário: Alice envia mensagem para Bob

1) Alice usa o agente de usuário (UA) para compor a mensagem e “para” bob@someschool.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 Bo
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.



2 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
```

2 Tente o SMTP você mesmo

- **`telnet nome do servidor 25`**
- Veja resposta 220 do servidor
- Envie comandos HELO, MAIL FROM, RCPT TO, DATA, QUIT.

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

2 SMTP: palavras finais

- SMTP usa conexões persistentes
- SMTP exige que as mensagens (cabeçalho e corpo) estejam em ASCII de 7 bits
- Servidor SMTP usa CRLF.CRLF para indicar o final da mensagem

Comparação com HTTP:

- HTTP: pull
- E-mail: push
- 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

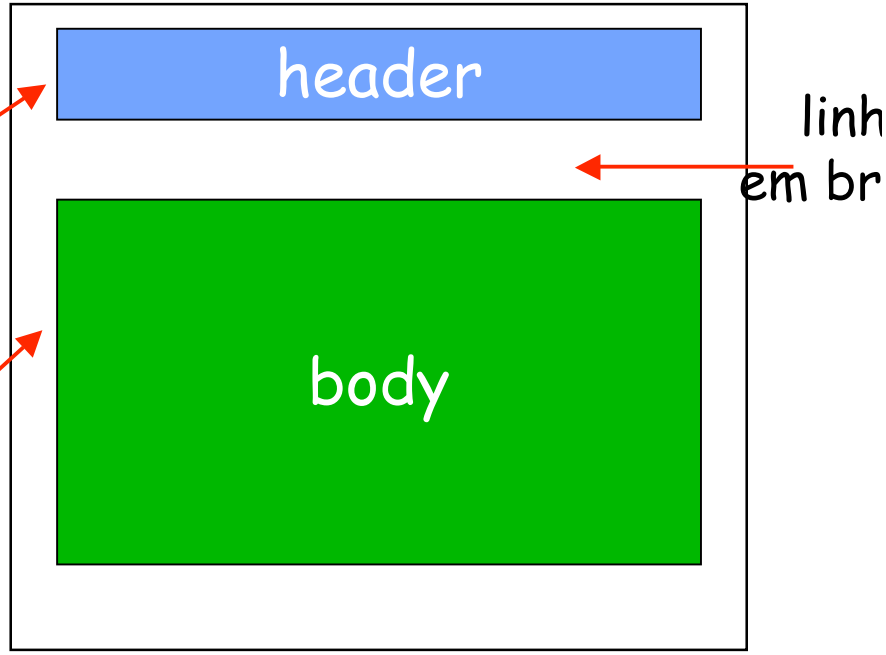
2 Formato da mensagem de correio

SMTP: protocolo para trocar mensagens de e-mail

RFC 822: padrão para mensagens do tipo texto:

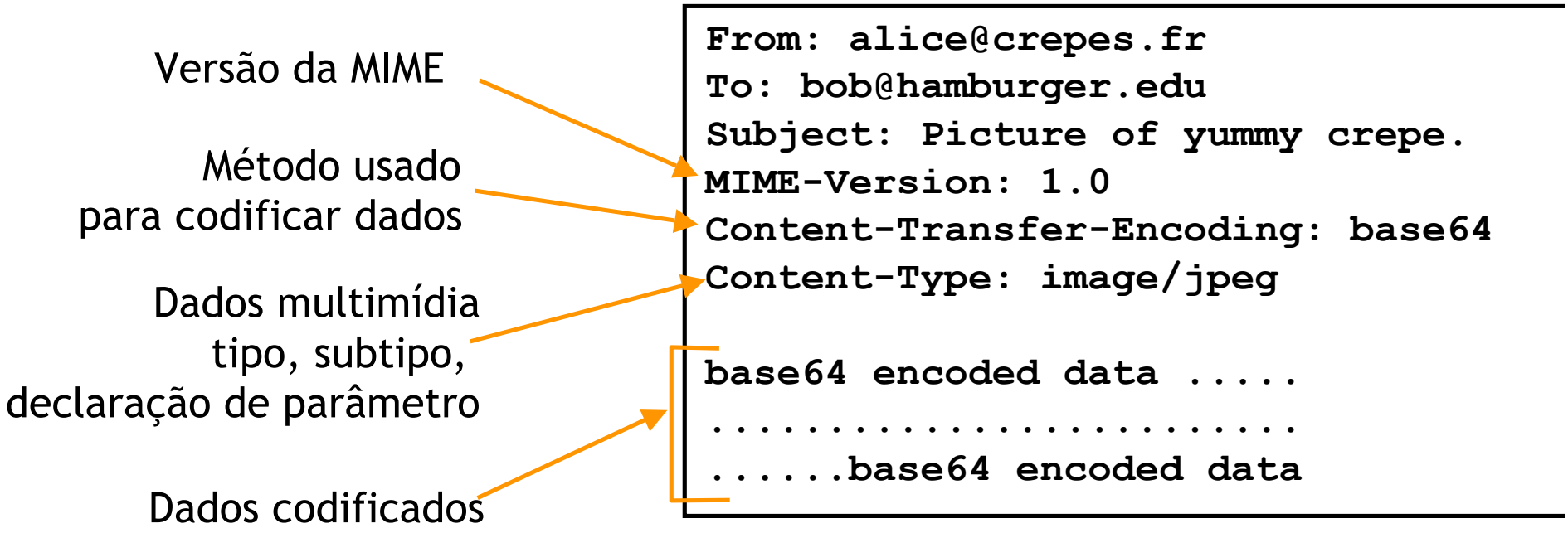
- linhas de cabeçalho, ex.:
  - To:
  - From:
  - Subject:

  diferente *dos comandos HTTP*
- corpo
  - a “mensagem”, ASCII somente com caracteres

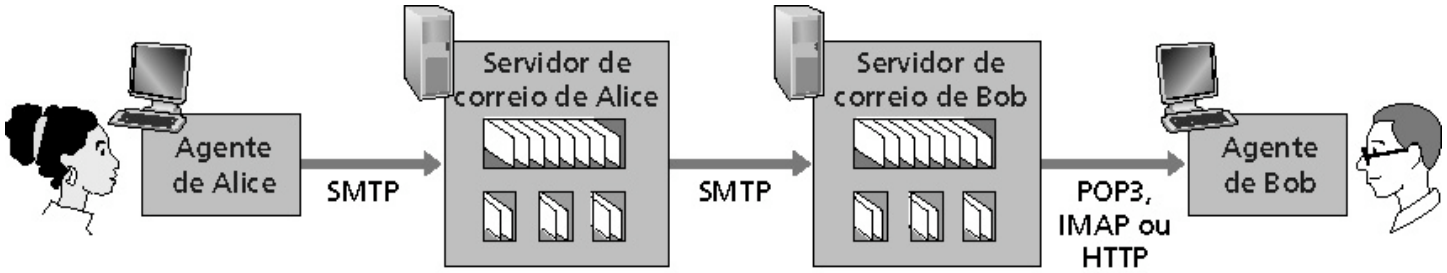


2 Formato das mensagens: extensões multimídia

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



2 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 Mail Access Protocol [RFC 1730]
    - Mais recursos (mais complexo)
    - Manipulação de mensagens armazenadas no servidor
  - HTTP: Hotmail, Yahoo! Mail etc.

2 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 o
```

2 POP3 (mais) e IMAP

Mais sobre POP3

- O exemplo anterior usa o modo “download-and-delete”
- Bob não pode reler o e-mail se ele trocar o cliente
- “download-and-keep”: cópias das mensagens em clientes diferentes
- POP3 é stateless através das sessões

IMAP

- Mantém todas as mensagens em um lugar: o servidor
- Permite que o usuário organize as mensagens em pastas
- IMAP mantém o estado do usuário através das sessões:
  - Nomes das pastas e mapeamentos entre os IDs da mensagem e o nome da p

2 Camada de aplicação

- 2.1 Princípios de aplicações de rede
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Correio eletrônico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Compartilhamento de arquivos P2P
- 2.7 Programação de socket com TCP
- 2.8 Programação de socket com UDP
- 2.9 Construindo um servidor Web

2 DNS: Dominain 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

2 DNS

DNS services

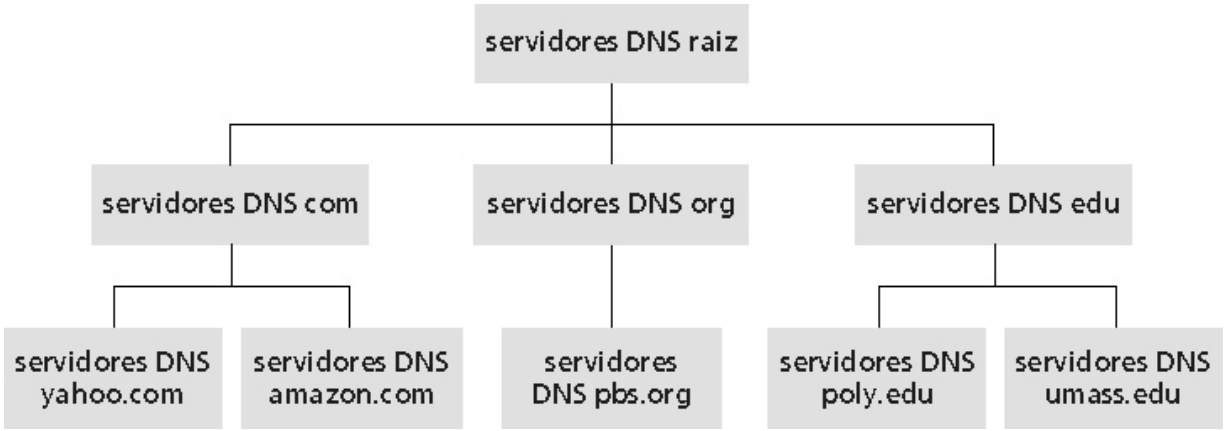
- Nome do hospedeiro para tradução de endereço IP
- Hospedeiro aliasing
  - Nomes canônicos e alias
    mail server aliasing
    distribuição de carga
  - Servidores Web replicados: estabelece o endereço IP para um nome canôni

Por que não centralizar o DNS?

- Ponto único de falha
- Volume de tráfego
- Base centralizada de dados distante
- Manutenção

Não é escalável!

2 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

2 DNS: servidores de nomes raiz

- São contatados pelos servidores de nomes locais que não podem resolver um nome
- Servidores de nomes raiz:
  - Buscam servidores de nomes autorizados se o mapeamento do nome não for conhecido
  - Conseguem o mapeamento
  - Retornam o mapeamento para o servidor de nomes local

Existem 13 servidores de nomes raiz no mundo

2 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, provêem nome de hospedeiro autorizado para mapeamentos IP para servidores de organizações (ex.: Web e mail).

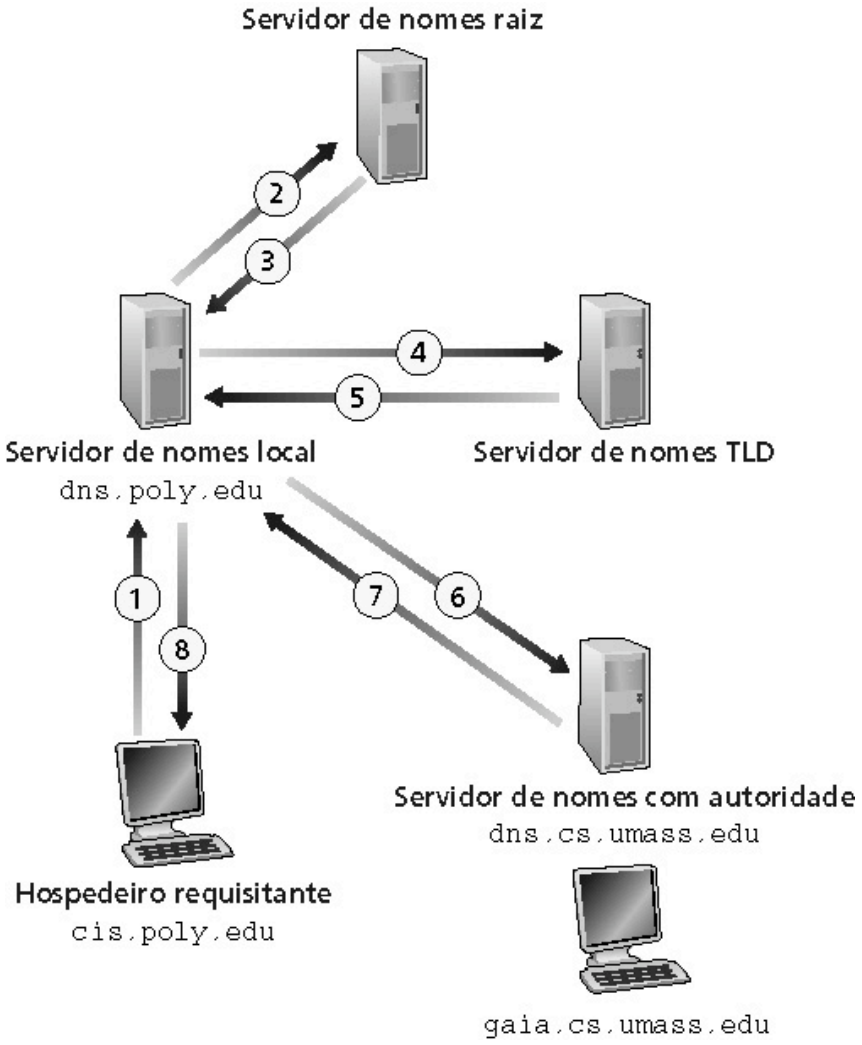
- Podem ser mantidos por uma organização ou provedor de serviços

2 Servidor de nomes local

- Não pertence estritamente a uma hierarquia
- Cada ISP (ISP residencial, companhia, universidade) possui um
- Também chamado de “servidor de nomes default”
- Quando um hospedeiro faz uma pergunta a um DNS, a pergunta é enviada para seu servidor DNS local
- Age como um proxy, encaminhando as perguntas para dentro da hierarquia

2 Exemplo

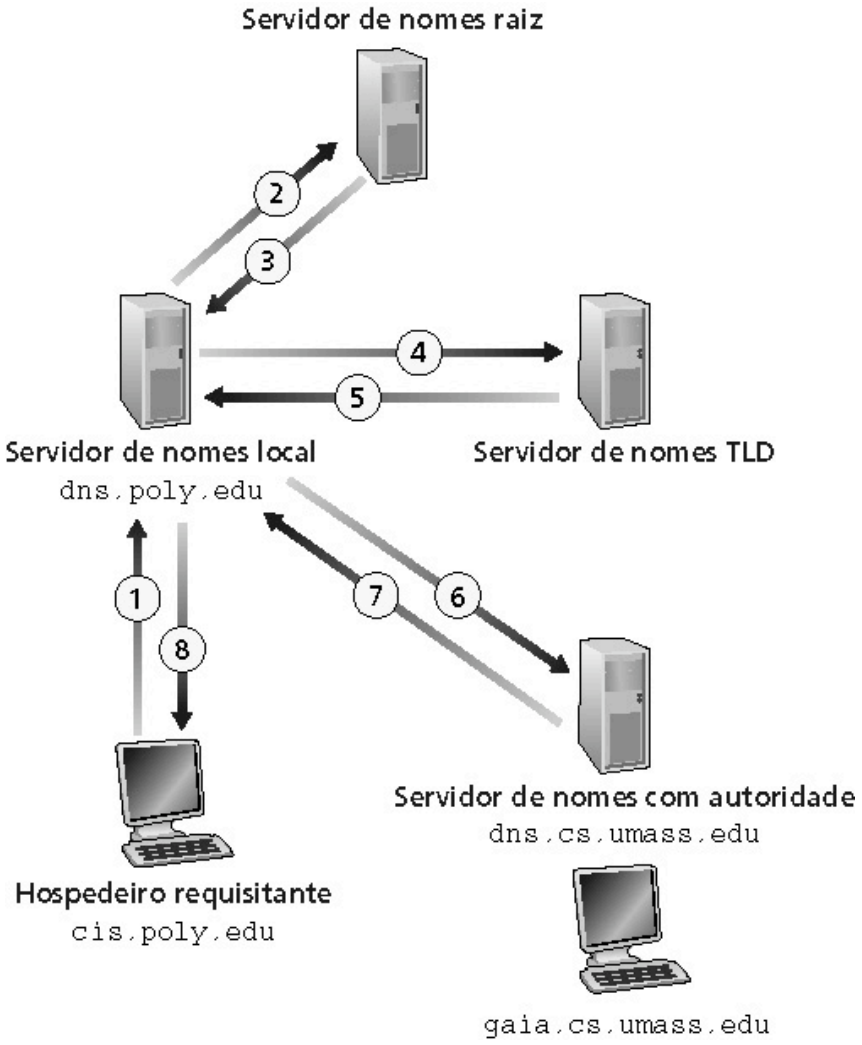
- O hospedeiro em cis.poly.edu quer o endereço IP para gaia.cs.umass.edu



2 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"



2 DNS: armazenando e atualizando registros

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

- Registros 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

2 Registros do DNS

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

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

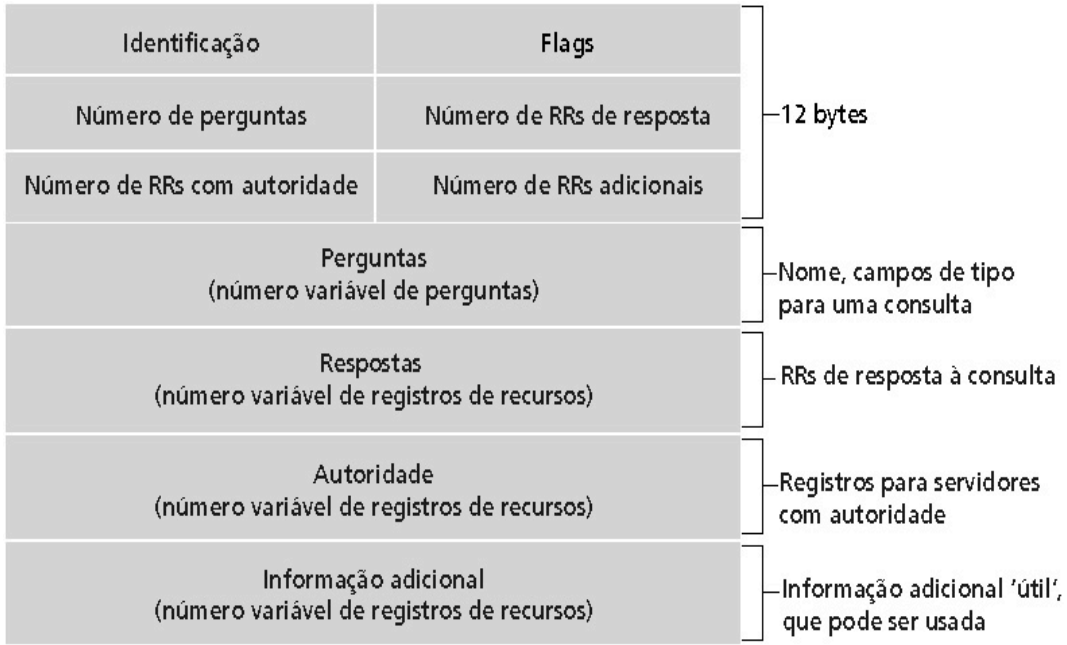
- Type = A
  - **name** é o nome do computador
  - **value** é o endereço IP
- Type = NS
  - **name** é um domínio (ex.: foo.com)
  - **value** é o endereço IP do servidor de nomes autorizados para este domínio
- Type = CNAME
  - **name** é um "apelido" para algum nome "canônico" (o nome real) www.ibm.com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

2 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



DNS: protocolo e mensagens

Camada de aplicação

- Exemplo: empresa recém-criada “Network Utopia”
- Registrar o nome networkutopia.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.networkutopia.com e um registro Tipo MX para networkutopia.com
- Como as pessoas obtêm o endereço IP do seu site Web?

Inserindo registros no DNS

2 Camada de aplicação

- 2.1 Princípios de aplicações de rede
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Correio eletrônico
    - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Compartilhamento de arquivos P2P
- 2.7 Programação de socket com TCP
- 2.8 Programação de socket com UDP
- 2.9 Construindo um servidor Web

2 Compartilhamento de arquivos P2P

Exemplo

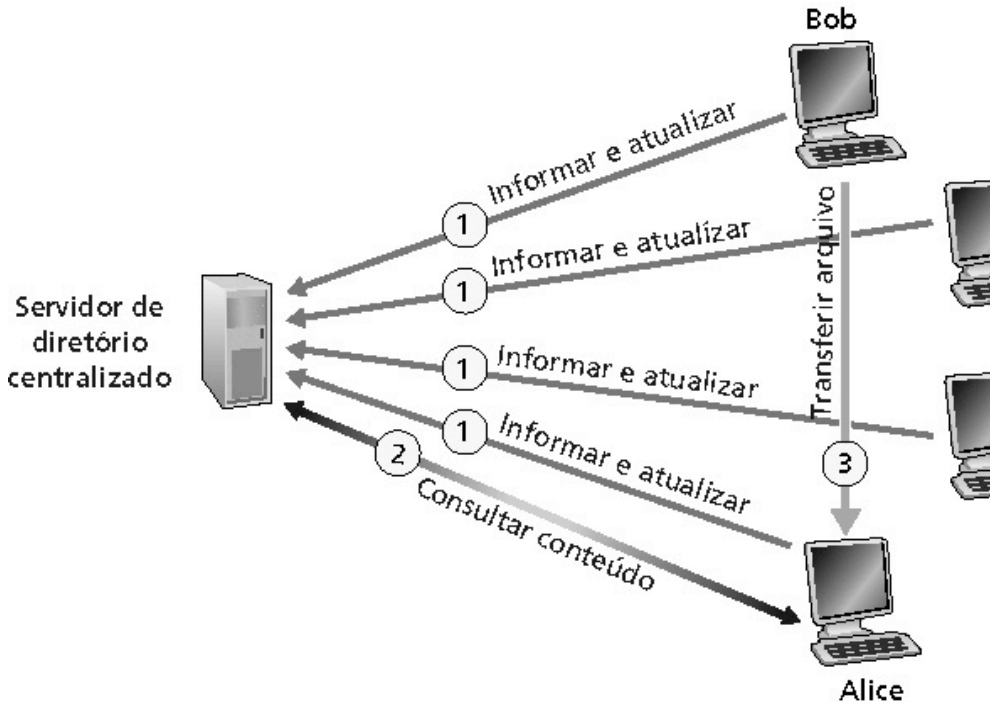
- Alice executa a aplicação cliente P2P em seu notebook
- intermitentemente, conecta-se à Internet; obtém novos endereços IP para cada conexão
- pede por “Hey Jude”
- a aplicação exibe outros pares que possuem uma cópia de Hey Jude
- Alice escolhe um dos pares, Bob
- o arquivo é copiado do PC de Bob para o notebook de Alice: HTTP
- enquanto Alice faz o download, outros usuários fazem upload de Alice
- o par de Alice é tanto um cliente Web como um servidor Web transiente

Todos os pares são servidores = altamente escaláveis!

2 P2P: diretório centralizado

Projeto original “Napster”

1) Quando um par se conecta, ele informa ao servidor central:
   - Endereço IP
   - Conteúdo
2) Alice procura por “Hey Jude”
3) Alice requisita o arquivo de Bob



2 P2P: problemas com diretório centralizado

- Ponto único de falhas
- Gargalo de desempenho
- Infração de copyright

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

2 Query flooding: Gnutella

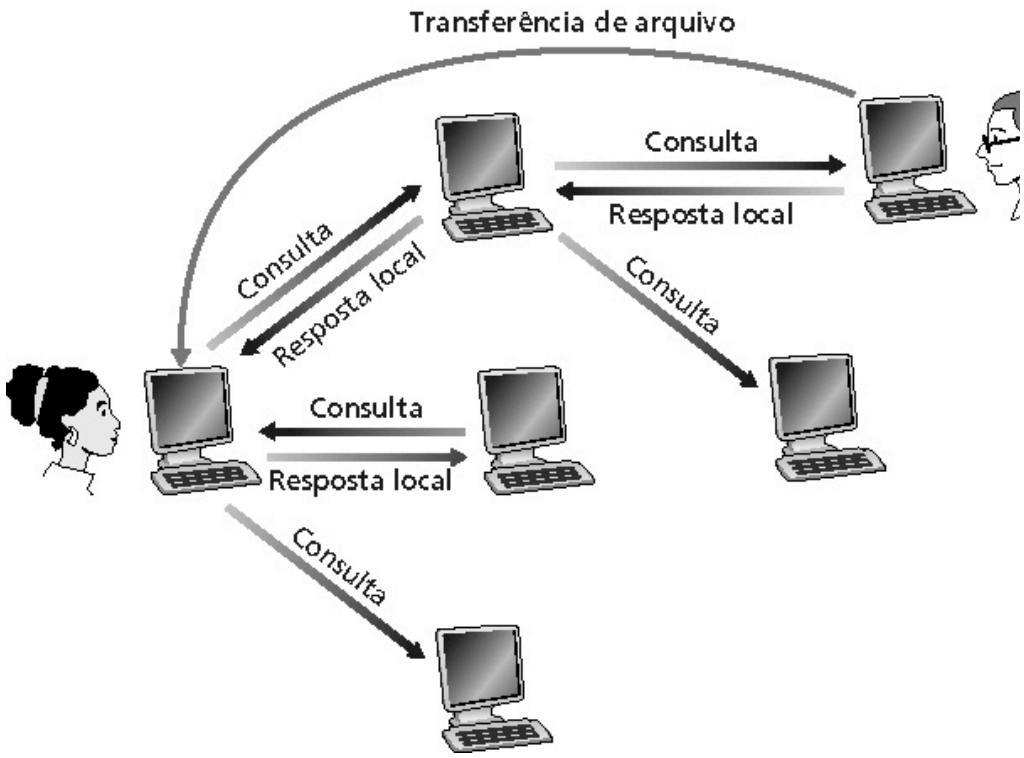
- Totalmente distribuído
  - Sem servidor central
- Protocolo de domínio público
- Muitos clientes Gnutella implementando o protocolo

Rede de cobertura: gráfico

- Aresta entre o par X e o Y se não há uma conexão TCP
- Todos os pares ativos e arestas estão na rede de sobreposição
- aresta não é um enlace físico
- Um determinado par será tipicamente conectado a <10 vizinhos na rede de sobreposição

2 Gnutella: protocolo

- Mensagem de consulta (query) é enviada pelas conexões TCP existentes
- Os pares encaminham a mensagem de consulta
- QueryHit (encontro) é enviado pelo caminho reverso



Escalabilidade: flooding de alcance limitado

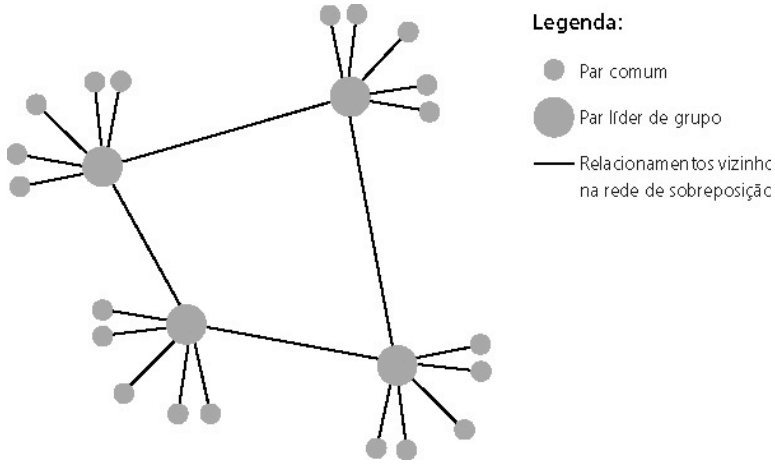
2 Gnutella: conectando pares

1. Para conectar o par X, ele precisa encontrar algum outro par na rede Gnutella: utiliza a lista de pares candidatos
2. X, seqüencialmente, tenta fazer conexão TCP com os pares da lista até estabelecer conexão com Y
3. X envia mensagem de Ping para Y; Y encaminha a mensagem de Ping
4. Todos os pares que recebem a mensagem de Ping respondem com mensagens de Pong
5. X recebe várias mensagens de Pong. Ele pode então estabelecer conexões TCP adicionais

Desconectando pares: veja o problema para trabalho de casa!

2 Explorando heterogeneidade: KaZaA

- Cada par é ou um líder de grupo ou está atribuído a um líder de grupo
  - Conexão TCP entre o par e seu líder de grupo
  - Conexões TCP entre alguns pares de líderes de grupo
- O líder de grupo acompanha o conteúdo em todos os seus "discípulos"



2 KaZaA

- Cada arquivo possui um hash e um descritor
- O cliente envia a consulta de palavra-chave para o seu líder de grupo
- O líder de grupo responde com os encontros:
  - Para cada encontro: metadata, hash, endereço IP
- Se o líder de grupo encaminha a consulta para outros líderes de grupo, eles respondem com os encontros
- O cliente então seleciona os arquivos para download
  - Requisições HTTP usando hash como identificador são enviadas aos pares que contêm o arquivo desejado

2 Artifícios do KaZaA

- Limitações em uploads simultâneos
- Requisita enfileiramento
- Incentiva prioridades
- Realiza downloads em paralelo

2 Camada de aplicação

- 2.1 Princípios de aplicações de rede
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Correio eletrônico

  SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Compartilhamento de arquivos P2P
- 2.7 Programação de socket com TCP
- 2.8 Programação de socket com UDP
- 2.9 Construindo um servidor Web

2 Programação de sockets

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

Socket API

- Introduzida no BSD4.1 UNIX, 1981
- Explicitamente criados, usados e liberados pelas aplicações
- Paradigma cliente-servidor
- Dois tipos de serviço de transporte via socket API:
  - Datagrama não confiável
  - Confiável, orientado a cadeias de bytes

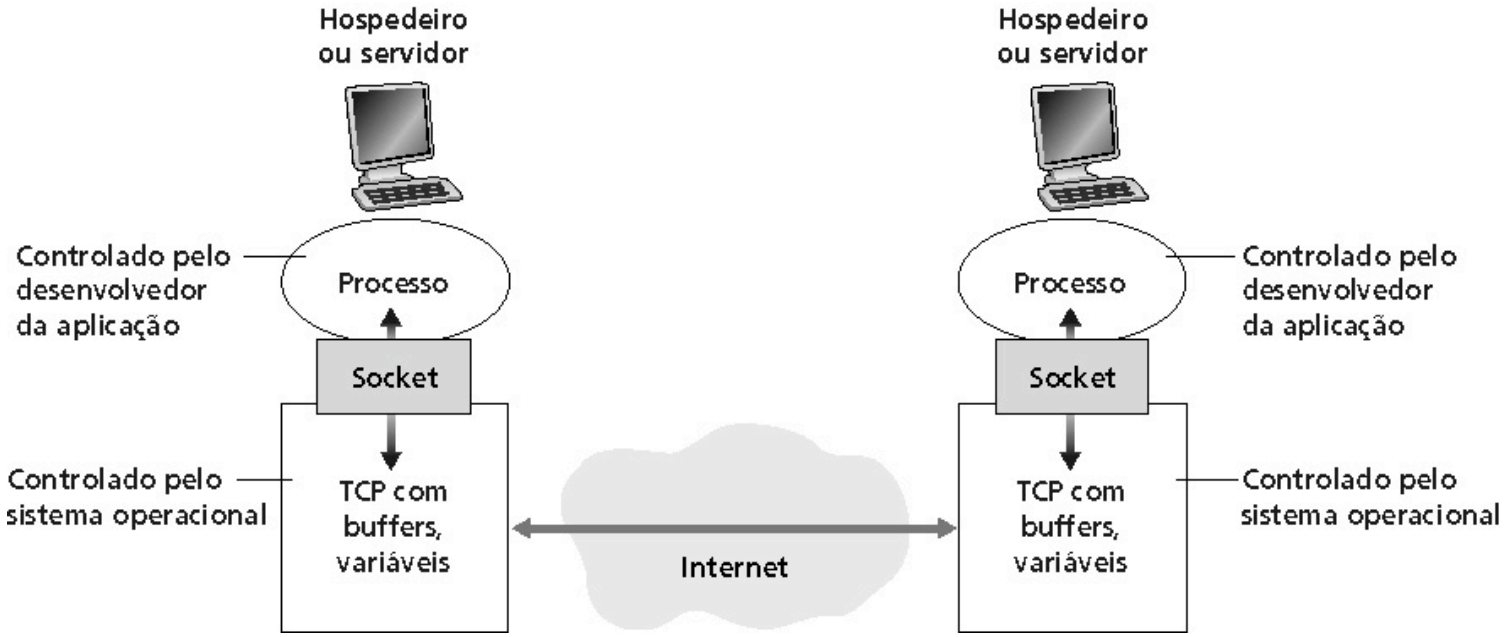
SOCKET

Uma 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)

2 Programação de sockets com TCP

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

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



2 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

2 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

2 Programação de sockets com TCP

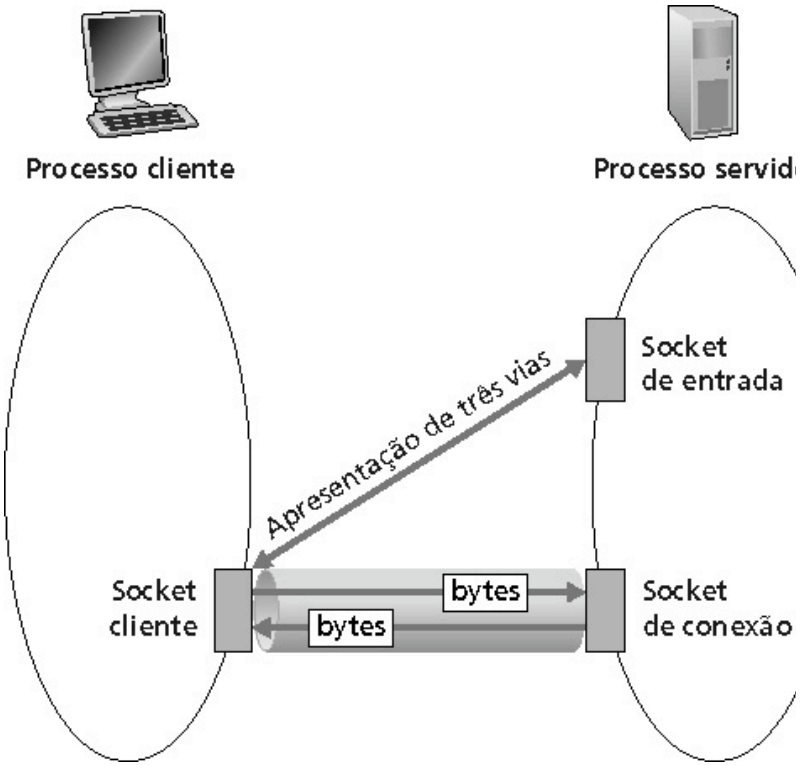
Exemplo de aplicação cliente-servidor:

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

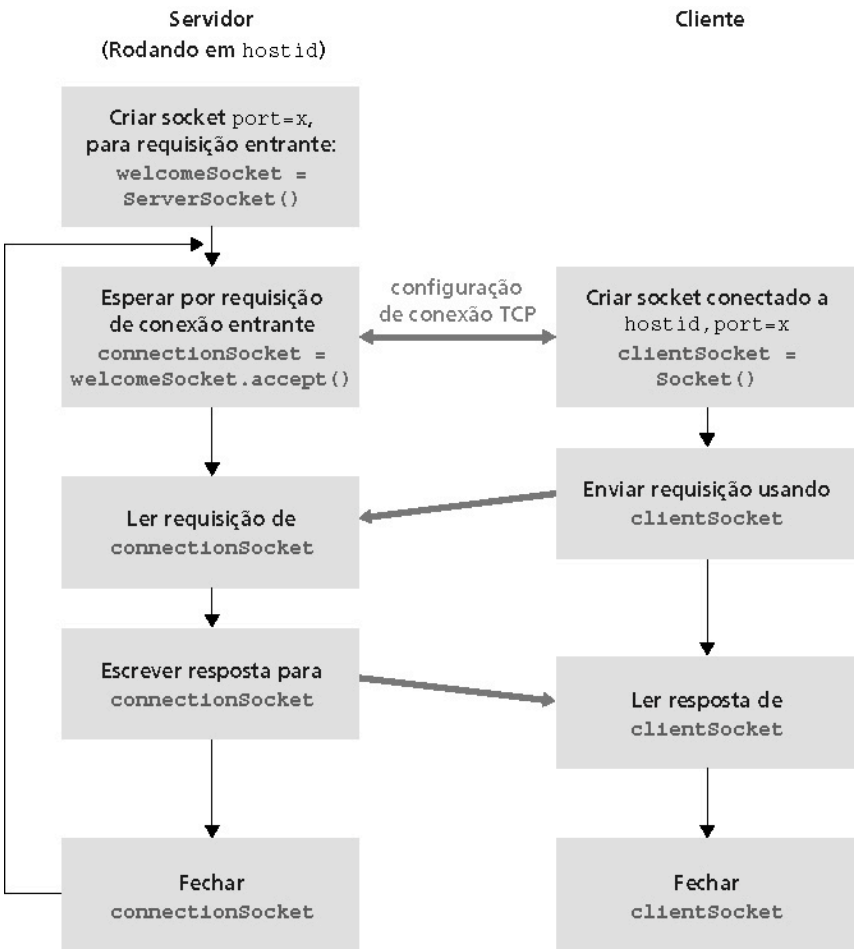
2) Servidor lê linha do socket

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

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



2 Interação cliente-servidor TCP



2 Exemplo: cliente Java (TCP)

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

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

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

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

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

Cria stream de entrada

Cria socket cliente, conecta ao servidor

Cria stream de saída ligado ao socket

2 Exemplo: cliente Java (TCP)

Cria stream de entrada ligado ao socket

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

sentence = inFromUser.readLine();

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

modifiedSentence = inFromServer.readLine();

System.out.println("FROM SERVER: " + modifiedSenten

clientSocket.close();

    }
}
```

Envia linha para o servidor

Lê linha do servidor

2 Exemplo: servidor Java (TCP)

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

class TCPServer {

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

      ServerSocket welcomeSocket = new ServerSocket(6789

      while(true) {

        Socket connectionSocket = welcomeSocket.accept()

       BufferedReader inFromClient =
          new BufferedReader(new
          InputStreamReader(connectionSocket.getInputStrea
```

Cria socket de aceitação na porta 6789

Espera, no socket de aceitação, por contato do cliente

Cria stream de entrada ligado ao socket

2 Exemplo: servidor Java (TCP)

Cria stream de saída ligado ao socket

Lê linha do socket

Escreve linha para o socket

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

clientSentence = inFromClient.readLine();

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

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

Fim do while loop, retorne e espere por outra conexão do cliente

2 Camadas de aplicação

- 2.1 Princípios de aplicações de rede
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Correio eletrônico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Compartilhamento de arquivos P2P
- 2.7 Programação de socket com TCP
- 2.8 Programação de socket com UDP
- 2.9 Construindo um servidor Web

2 Programação de sockets com UDP

UDP: não há conexão entre o cliente e o servidor

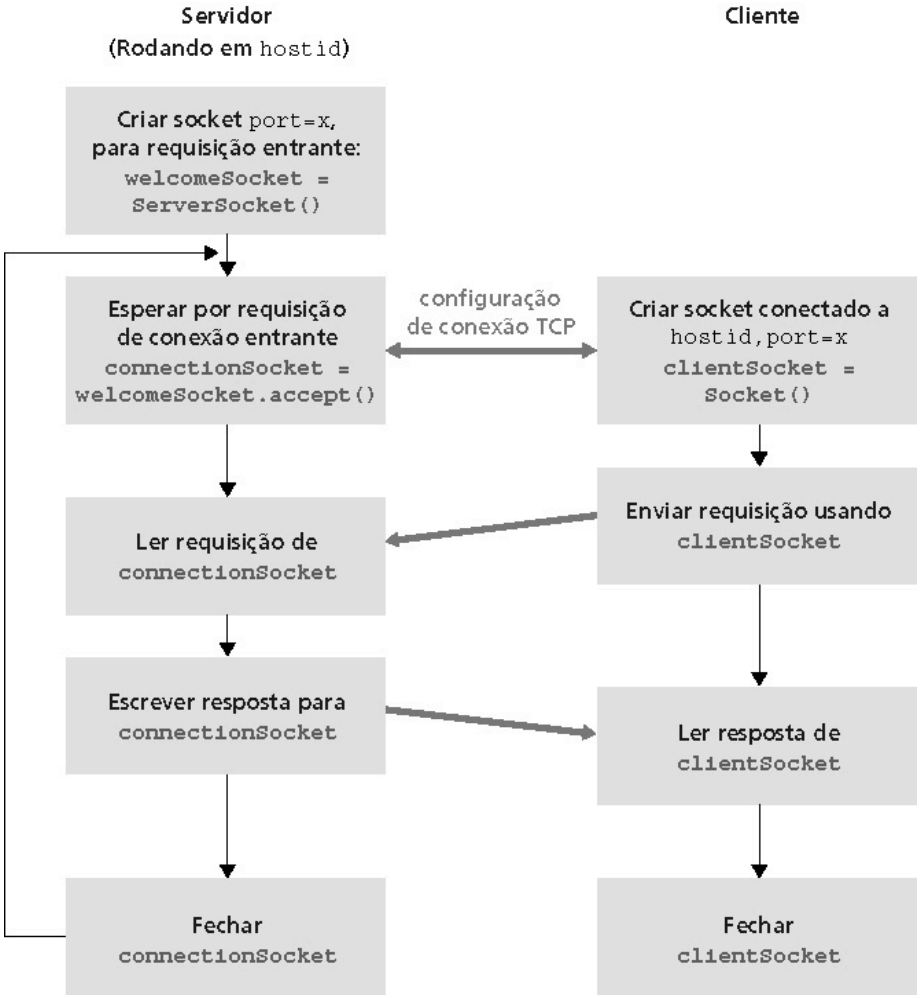
- Não existe apresentação
- Transmissor envia explicitamente endereço IP e porta de destino em cada mensagem
- Servidor deve extrair o endereço IP e porta do transmissor de cada datagrama recebido

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

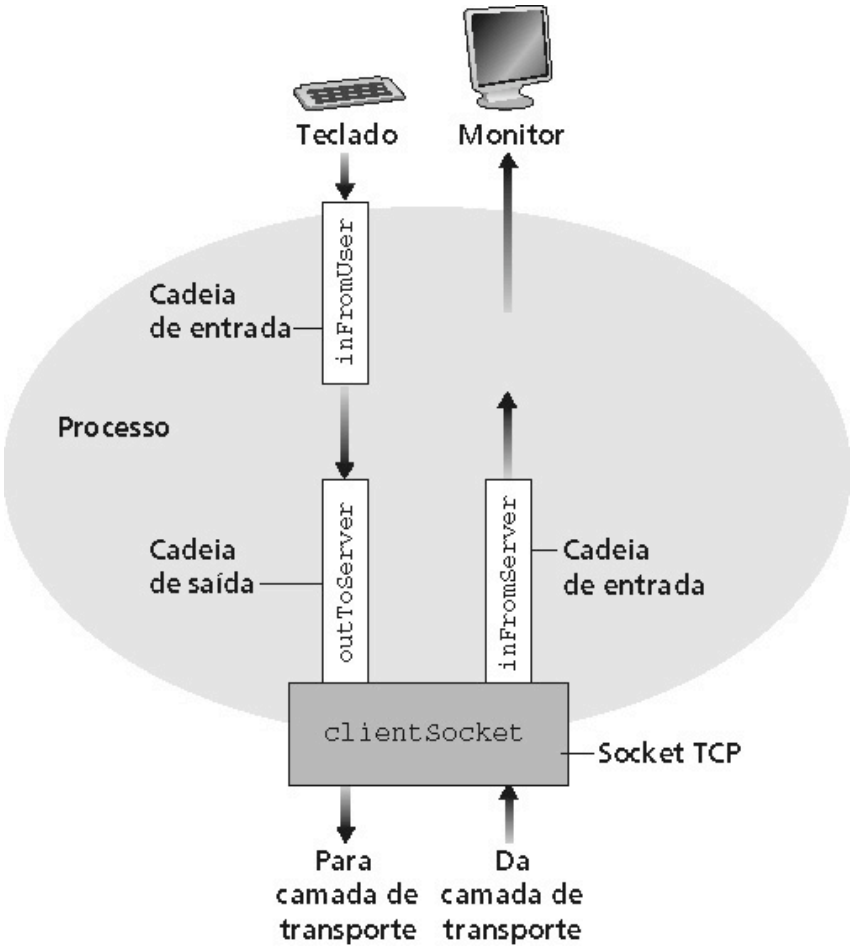
Ponto de vista da aplicação

UDP fornece a transferência não confiável de grupos de bytes (datagramas) entre o cliente e o servidor

2 Interação cliente-servidor: UDP



2 Exemplo: cliente Java (UDP)



2 Exemplo: cliente Java (UDP)

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

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

      DatagramSocket clientSocket = new DatagramSocket();

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

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

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

Cria stream de entrada

Cria socket cliente

Translada nome do hospedeiro para endereço IP usando DNS

2 Exemplo: cliente Java (UDP)

Cria datagrama com dados a enviar, tamanho, endereço IP porta

Envia datagrama para servidor

Lê datagrama do servidor

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

                clientSocket.send(sendPacket);

                DatagramPacket receivePacket =
                   new DatagramPacket(receiveData, receiveData.length

                clientSocket.receive(receivePacket);

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

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

2 Exemplo: servidor Java (UDP)

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

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

      DatagramSocket serverSocket = new DatagramSocket(9

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

      while(true)
        {

        DatagramPacket receivePacket =
          new DatagramPacket(receiveData, receiveData.leng
        serverSocket.receive(receivePacket);
```

Cria socket datagrama na porta 9876

Cria espaço para datagramas recebidos

Recebe datagrama

2 Exemplo: servidor Java (UDP)

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

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

                String capitalizedSentence = sentence.toUpperCas

sendData = capitalizedSentence.getBytes();

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

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

Obtém endereço IP e número da porta do transmissor

Cria datagrama para enviar ao cliente

Escreve o datagrama para dentro do socket

Termina o while loop, retorna e espera por outro datagrama

2 Camada de aplicação

- 2.1 Princípios de aplicações de rede
- 2.2 Web e HTTP
- 2.3 FTP
- 2.4 Correio eletrônico
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 Compartilhamento de arquivos P2P
- 2.7 Programação de socket com TCP
- 2.8 Programação de socket com UDP
- 2.9 Construindo um servidor Web

2 Construindo um servidor Web simples

- Manipule uma requisição HTTP
- Aceite a requisição
- Analise o cabeçalho
- Obtenha um arquivo requisitado do sistema de arquivo do servidor
- Crie uma mensagem de resposta HTTP:
  - Linhas de cabeçalho + arquivo
- Envie a resposta para o cliente
- Após criar o servidor, você pode requisitar um arquivo usando um browser (ex.: IE explorer)
- Veja o texto para mais detalhes

2 Resumo

Nosso estudo de aplicações está completo agora!

- Arquiteturas de aplicação
  - Cliente-servidor
  - P2P
  - Híbrida
- Exigências dos serviços de aplicação:
  - Confiabilidade, banda passante, atraso
- Modelo do serviço de transporte da Internet l
  - Orientado à conexão, confiável: TCP
  - Não confiável, datagramas: UDP
- Protocolos específicos:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
- Programação de sockets

2 Resumo

Mais importante: características dos **protocolos**

- Típica troca de mensagens comando/resposta:
  - Cliente solicita informação ou serviço
  - Servidor responde com dados e código de *status*
- Formatos das mensagens:
  - Cabeçalhos: campos que dão informações sobre os dados
  - Dados: informação sendo comunicada
- Controle *vs.* dados
  - In-band, out-of-band
- Centralizado *vs.* descentralizado
- Stateless *vs.* stateful
- Transferência de mensagens confiável *vs.* não confiável
- "Complexidade na borda da rede"