# Curso de Redes de Computadores

Vinicius F. Caridá

UEMG
FRUTAL

# Capítulo 2: Camada de Aplicação

## Metas do capítulo:

- O que?
  - Aprender aspectos conceituais e de implementação de protocolos de **aplicação** em redes.
  - Modelo cliente servidor.
  - Modelos de serviço.
- Como?
  - Estudo de **protocolos populares** da camada da aplicação.

## Conhecer...

- Protocolos específicos:
  - *http*
  - *FTP*
  - *SMTP*
  - *POP3*
  - *DNS*
  - *P2P*
- Como é feita a programação de aplicações de rede.
  - Programação usando *sockets*.

UEMG FRUTAL

# Conceitos

Clientes, servidores, processos, *sockets* e outros bichos…

UEMG FRUTAL

# Princípios de aplicação de rede

- Exemplos de aplicações de rede: correio eletrônico, a Web, mensagem instantânea, login em computador remoto como Telnet e SSH, compartilhamento de arquivos P2P, transferência de arquivos, etc.

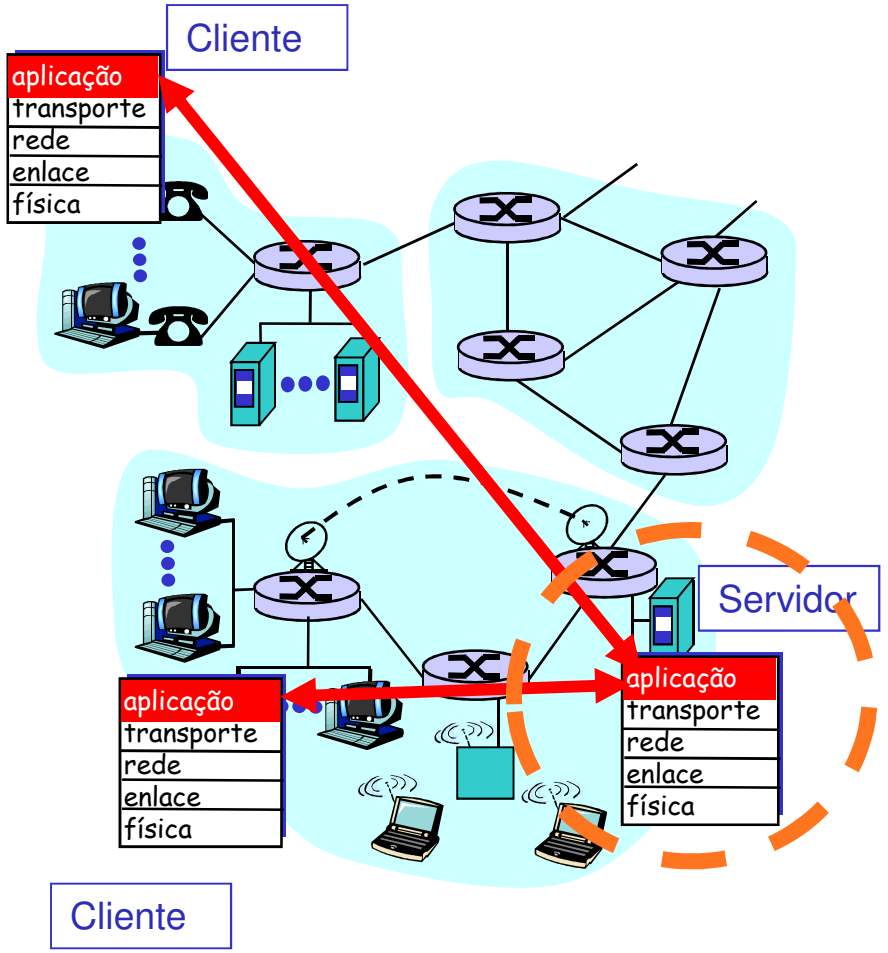
# Princípios de aplicação de rede

- O cerne do desenvolvimento de aplicação de rede é escrever programas que rodem em sistemas finais diferentes e se comuniquem entre si pela rede. Por exemplo, a na Web há dois programas distintos que se comunicam, o programa do browser (roda na máquina do ususário) e o programa servidor Web (roda na máquina do servidor Web).

# Aplicações e protocolos da camada de aplicação

Aplicação: processos distribuídos em comunicação

- Rodam em hosts.
  - Sistemas finais.
- **Trocam mensagens** para implementar aplicação.
- Exemplo:
  - Correio eletrônico, transferência de arquivos, WWW, login remoto, VoIP, etc...



# Protocolos de aplicação

- **Protocolo** da camada de aplicação:
  - **Não é a aplicação.**
  - É APENAS **uma parte da aplicação**.
    - Define **mensagens trocadas** por aplicações, e **ações** as tomadas em sua resposta.
  - **Usam serviços** providos por protocolos de camadas inferiores.

Os processos em dois sistemas de extremidade diferentes comunicam-se logicamente entre si, trocando **mensagens** através da rede de computadores.

- Um processo de **emissão** cria e emite mensagens na rede;
- um processo de **recepção** recebe estas mensagens
  - e responde possivelmente emitindo mensagens de volta.

UEMG FRUTAL


# Arquitetura

- A arquitetura da aplicação determina como a aplicação é organizada nos vários sistemas finais.

# Arquitetura

- Em uma arquitetura cliente-servidor há um hospedeiro sempre em funcionamento, denominado servidor, que atende a requisições de muitos outros hospedeiros, denominados clientes, estes podem estar em funcionamento às vezes ou sempre. Os clientes não se comunicam diretamente uns com os outros. O servidor tem um endereço fixo, denominado endereço de IP, o cliente sempre pode contatá-lo, enviando um pacote ao endereço do servidor.

# Arquitetura

- Em aplicações cliente-servidor, muitas vezes acontece de um único hospedeiro servidor ser incapaz de atender a todas as requisições de seus clientes, por essa razão, muitas vezes são utilizados conjuntos de hospedeiros (server farm) para criar servidor virtual poderoso em arquiteturas cliente-servidor.

# Arquitetura

- Em uma arquitetura P2P pura, não há um servidor sempre funcionando no centro da aplicação, em vez disso pares arbitrários de hospedeiros comunicam-se diretamente entre si. Como os pares se comunicam sem passar por nenhum servidor especial, a arquitetura é denominada peer-to-peer, onde nela nenhuma das máquinas participantes precisa estar sempre em funcionamento. Um de suas características mais fortes é a escalabilidade, onde cada par adicional não apenas aumenta a demanda, mas também a capacidade de serviço.

# Arquitetura

- Por outro lado, devido à sua natureza altamente distribuída e descentralizada, pode ser difícil de gerenciar aplicações P2P. Muitas aplicações são organizadas segundo arquiteturas hibridas cliente/servidor/P2P, a Napster era um exemplo disso, no sentido de que era P2P porque arquivos MP3 eram trocados diretamente entre pares, sem passar por servidores dedicados, sempre em funcionamento, mas também era cliente-servidor, já que um par consultava um servidor central para determinar quais pares que estavam em funcionamento tinham um arquivo MP3 desejado.

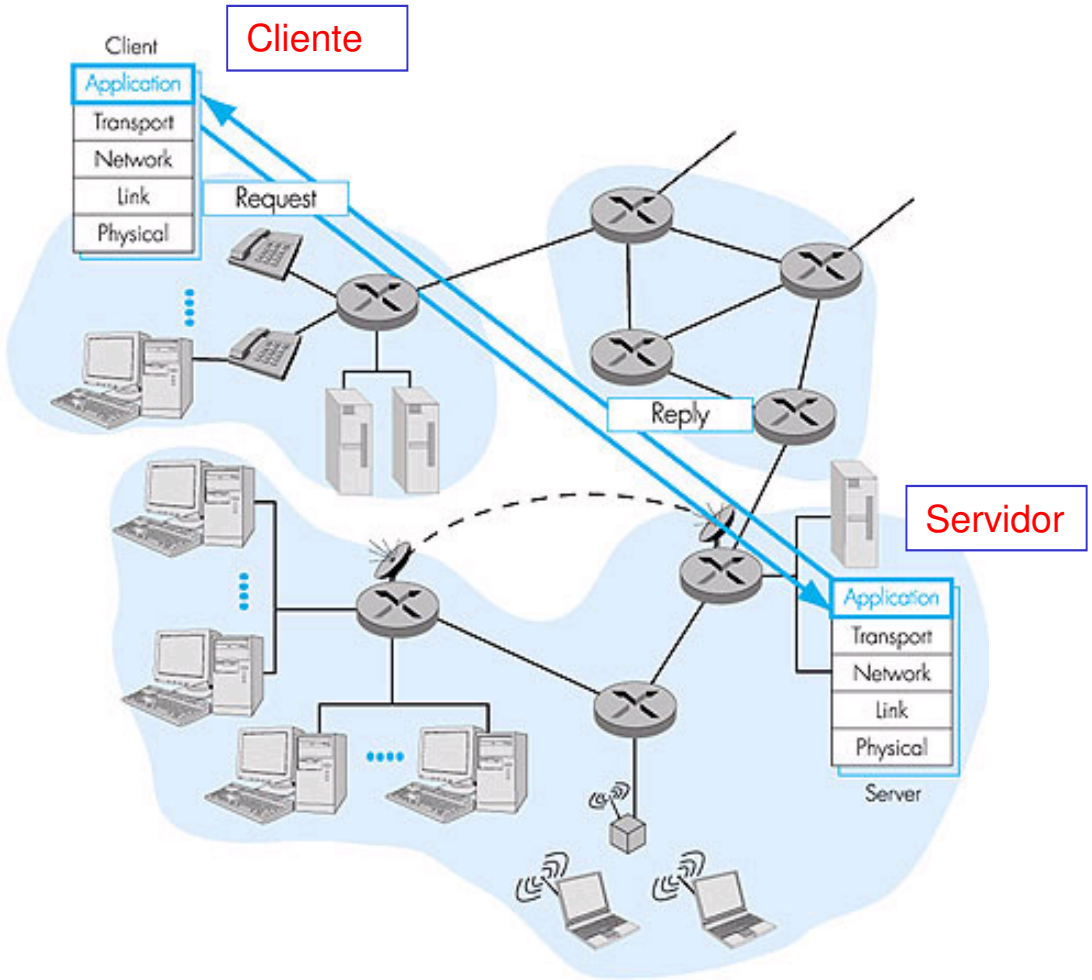
# Como os processos trocam mensagens:

- Um protocolo da camada de aplicação **define como os processos** de aplicação **trocam mensagens**.
  - Funcionando em sistemas de extremidade diferentes,
- Um protocolo da **camada de aplicação** define:

- Os **tipos de mensagens trocadas.**
  - Por exemplo, mensagens do pedido e mensagens de resposta.
- A **sintaxe** dos vários tipos de mensagem.
  - Os **campos** na mensagem, e como os campos são delineados.
- A **semântica dos campos.**
  - Isto é, o **significado da informação** nos campos.
- As **regras.**
  - Determinar **quando** e como um processo emite e responde mensagens.

UEMG FRUTAL

# Como os processos trocam mensagens



# Aplicações de rede: **DEFINIÇÕES**

- Um **processo** é um programa que roda num *host*.
- Dois processos **no mesmo *host*** se comunicam usando **comunicação entre processos (*interprocess comunnication*),** definida pelo sistema operacional (SO).
- Dois **processos em *hosts* distintos** se comunicam usando um **protocolo da camada de transporte**.

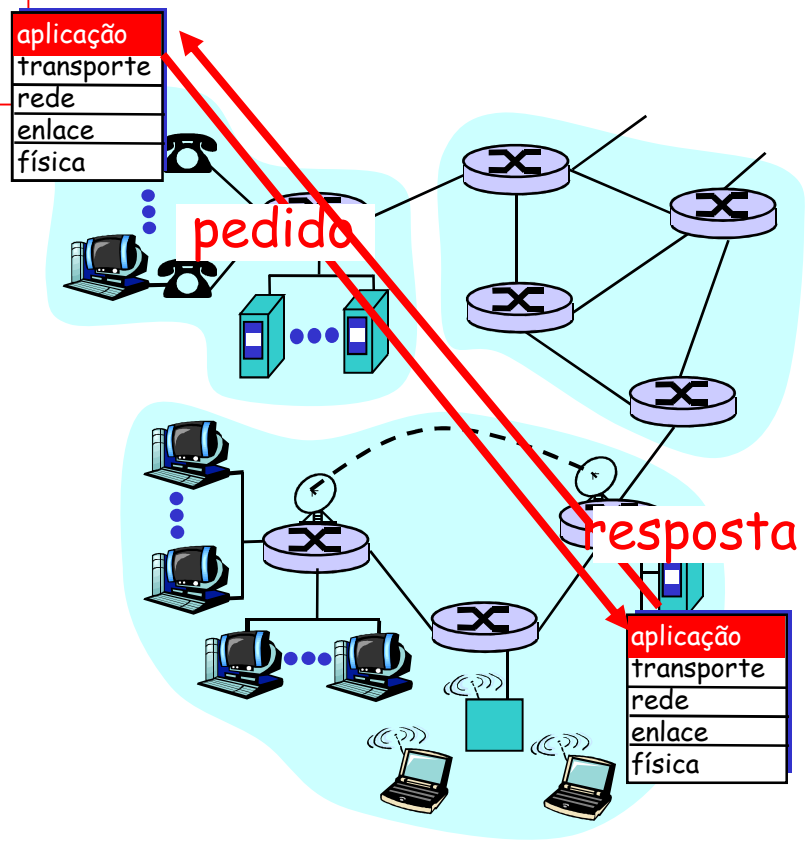
- Um agente de usuário (UA) é uma **interface entre o usuário e a aplicação de rede**.
  - WWW: *browser*.
  - Correio: leitor/compositor de mensagens.
  - *Streaming* A/V: *player* de mídia.

# Processos clientes e processos servidores

- Para cada par de processos comunicantes normalmente rotulamos um dos dois processos de cliente(que inicia a comunicação) e o outro de servidor(processo que é contatado para iniciar a sessão).
- Na Web, um processo browser inicia o contato com um processo do servidor Web(cliente) e o processo do servidor Web é o servidor. No compartilhamento de arquivos P2P, quando o ParA solicita ao ParB o envio de um arquivo especifico, o ParA é o cliente enquanto o ParB é o servidor.

# Paradigma cliente-servidor

Aplicação de rede típica tem duas partes: ***cliente*** e ***servidor***




**Cliente:**

- Inicia contato com o servidor
- Define-se como aquele que "chama".
- Solicita serviço do servidor.

**Servidor:**

- Provê o serviço requisitado ao cliente.

# Protocolos da camada de aplicação

*API - Aplication Program Interface*:

- Interface de programação de aplicações.

- Define a **interface** entre a **aplicação** e **camada de transporte**.
- *APIs* → definidas pelos RFCs.
- *Socket* (= tomada).
  - Dois processos se comunicam enviando dados para um ***socket*** , ou lendo dados de um ***socket.***

... voltaremos mais tarde a este assunto.

# Comunicação pelos *sockets* (1)

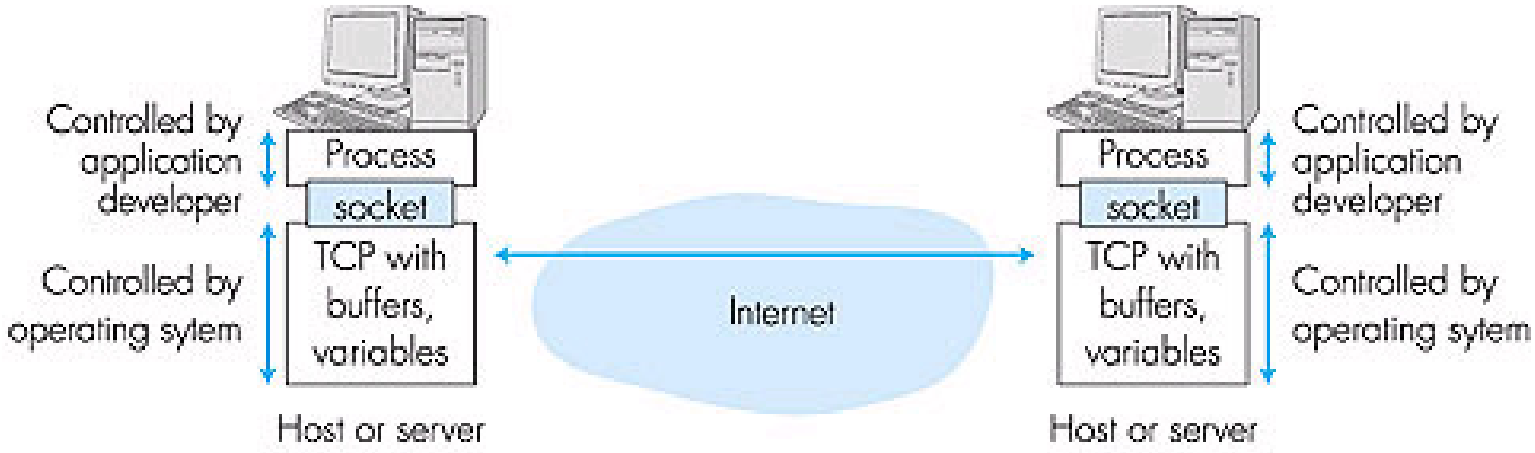
- O *socket* pode ser entendido como a **porta** do processo:
  - Um processo **emite e recebe mensagens** da rede, **através de seus *sockets*.**
  - Quando um processo quer **emitir uma mensagem a um outro processo** em um outro host, **empurra a mensagem para o *socket*.**
  - O processo supõe que há um **infra-estrutura de transporte** no outro lado, a qual transportará a mensagem até a porta do processo do destino.

UEMG FRUTAL

# Sockets:

- Um socket é a interface entre a camada de aplicação e a de transporte dentro de uma máquina.

# Comunicação pelos *sockets* (2)




– Como um processo identifica outro?

# Endereçamento de processos:

- Para que um processo em um hospedeiro envie uma mensagem a um processo em outro, o processo de origem tem de identificar o processo destinatário. Para isso ele precisa de duas informações, o nome ou o endereço da máquina hospedeira e um identificador que especifique o processo destinatário.

# Como um processo identifica outro ?

Pergunta: Como um processo pode "identificar"o outro processo com o qual quer se comunicar?

- **Endereço IP** do *host* do outro processo.
  - Por enquanto: o ***IP ADDRESS* é um valor de 32-bits que identifica unicamente o sistema de extremidade**. Mais precisamente: identifica unicamente a **interface** (placa) que conecta esse host à Internet. Ex. `200.145.9.9`
- "**Número de porta**" : permite que o hospedeiro receptor determine para qual processo deve ser entregue a mensagem. Exemplo: Porta 80/TCP.

  Ex. `200.145.9.9:80` (RFC 1700 - Portas *well-known*)

# Endereçamento de processos:

- O processo destinatário é identificado por seu **endereço de IP** que é uma quantidade de 32 bits que identifica exclusivamente o sistema final. Além de saber o endereço, o processo de origem tem de identificar o processo que está rodando no outro hospedeiro, um **número de porta** de destino atende a essa finalidade.

- Um protocolo de camada de aplicação define como processos de uma aplicação, que funcionam em sistemas finais diferentes, passam mensagens entre si, em particular ele define os tipos de mensagens trocadas, a sintaxe dos vários tipos de mensagem, a semântica dos campos, regras para determinar quando e como um processo envia e responde mensagens. Muitos protocolos de camada de aplicação são proprietários e não estão disponíveis ao publico. É importante distinguir aplicações de rede de protocolos de camada de aplicação, um protocolo de camada de aplicação é apenas um pedaço de aplicação de rede.


*De que serviços uma aplicação necessita?*

# De que serviços de transporte uma aplicação precisa? (1)

- Quando se desenvolve uma aplicação, deve-se **escolher um dos protocolos disponíveis do transporte**.
- Como se faz esta escolha?
  - Estuda-se os serviços fornecidos pelos protocolos disponíveis do transporte, e escolhe-se o protocolo com os serviços que melhor se adaptem às necessidades de sua aplicação.
  - "Com Conexão" ou "Sem Conexão".

# De que serviços de transporte uma aplicação precisa? (2)

**Perda de dados**:

- Algumas aplicações (por exemplo áudio) podem tolerar algumas perdas.
- Outras (por exemplo, transferência de arquivos, telnet) exigem transferência 100% confiável.

**Temporização:**

- Algumas aplicações (por exemplo, telefonia em Internet, jogos interativos) requerem baixo retardo para serem "viáveis".

**Largura de banda:**

• Algumas aplicações (por exemplo , multimídia) requerem quantia mínima de banda para serem "viáveis".
• Outras aplicações ("elásticas") conseguem usar qualquer quantia de banda disponível.

UEMG FRUTAL

# Transferência confiável de dados

- Algumas aplicações exigem transferência de dados totalmente confiável, isto é, não pode haver perda de dados (que pode ter consequências devastadoras). Outras aplicações podem tolerar uma certa perda de dados, mais notavelmente aplicações de multimídia.

# Largura de banda

- Algumas aplicações têm de transmitir dados a uma certa velocidade para serem efetivas. Se essa largura de banda não estiver disponível, a aplicação precisará codificar a uma taxa diferente ou então desistir, já que receber metade da largura de banda que precisa de nada adianta para tal aplicação sensível à largura de banda. Embora aplicações sensíveis à largura de banda exijam uma dada quantidade de largura de banda, aplicações elásticas(correio eletrônico, transferência de arquivos, etc) podem fazer uso de qualquer quantidade mínima ou máxima que por acaso esteja disponível.

# Temporização

- O requisito final de serviço é a temporização. Aplicações interativas em tempo real, exigem limitações estritas de temporização na entrega de dados para serem efetivas.

# Serviços providos por protocolos de transporte Internet

## Serviço TCP:

- Orientado a conexão: estabelecimento exigido entre cliente e servidor.
- Transporte confiável entre processos emissor e receptor.
- Controle de fluxo: emissor não vai "afogar" receptor.
- Controle de congestionamento: estrangular emissor quando a rede carregada.
- Não oferece: garantias temporais ou de banda mínima (*QoS*).

## Serviço UDP:

- Transferência de dados **não confiável** entre processos remetente e receptor.
- **Não provê**: estabelecimento da conexão, confiabilidade, controle de fluxo, controle de congestionamento, garantias temporais ou de banda mínima.

**Exercício:** Descreva onde e quando é interessante usar UDP, e quando não é.

UEMG FRUTAL

# Exercício: Veja quais as principais aplicações Internet e que tipo de protocolos de transporte elas utilizam, e em quais portas operam.

| **Aplicação** | **Protocolo da camada de apl** | **Protocolo de transporte usado** |
|---|---|---|
| Correio eletrônico | smtp [RFC 821] | ? |
| Acesso terminal remoto | telnet [RFC 854] | ? |
| WWW | http [RFC 2068] | ? |
| Transf. de arquivos | ftp [RFC 959] | ? |
| streaming multimídia | proprietário (Ex: *RealNetworks*) | ? |
| Servidor de arquivo | NFS | ? |
| VoIP | proprietário (Ex: Skype) | ? |

UEMG FRUTAL


# Os principais protocolos de aplicação

- O HTTP (Protocolo de Transferência de Hipertexto) é um protocolo da camada de aplicação, ele é implementado em dois programas, um cliente e outro servidor. Os dois programas, executados em sistemas finais diferentes, conversam um com o outro por meio da troca de mensagens HTTP. O protocolo define a estrutura dessas mensagens e o modo como o cliente e o servidor as trocam.

- Uma **página Web** é constituída de objetos que são simplesmente arquivos que se podem acessar com um único URL. A maioria das páginas Web é constituída de um arquivo-base HTML e diversos objetos referenciados. Cada URL tem dois componentes, o nome do hospedeiro do servidor que abriga o objeto e o nome do caminho do objeto.

- Um **browser** é um agente de usuário para a Web, apresenta a página requisitada ao usuário e fornece numerosas características de navegação e de configuração. Um **servidor Web** abriga objetos Web, cada um endereçado por um URL.

- Quando um usuário requisita uma pagina Web, o browser envia ao servidor mensagens de requisição HTTP para os objetos da página, o servidor recebe as requisições e responde com mensagens de resposta HTTP que contêm os objetos.

- Até 1997, essencialmente todos os browser e servidores Web implementavam a versão HTTP/1.0, a partir de 1998 eles começaram a implementar a versão HTTP/1.1. O HTTP /1.1 é compatível com o HTTP /1.0, um servidor Web que executa a versão 1.1 pode se comunicar com um servidor que executa a 1.0.

- O HTTP usa o TCP como seu protocolo de transporte subjacente.
- O servidor HTTP não mantém nenhuma informação sobre clientes, por isso é denominado protocolo sem estado.

# W W W

- Página WWW:
  - Consiste de “objetos”
  - Endereçada por um URL: *Universal Resource Locator*.
  - Quase todas as páginas WWW consistem de:
  - Página base HTML, e
  - Vários objetos referenciados.
- URL tem duas partes: nome do host, e nome de caminho.

- Agente de usuário para WWW = *browser*:
  - MS Internet Explorer.
  - Netscape Communicator.
- Servidor para WWW se chama “servidor WWW”:
  - Apache (*Open Software*).
  - *MS Internet Information Server* (IIS).

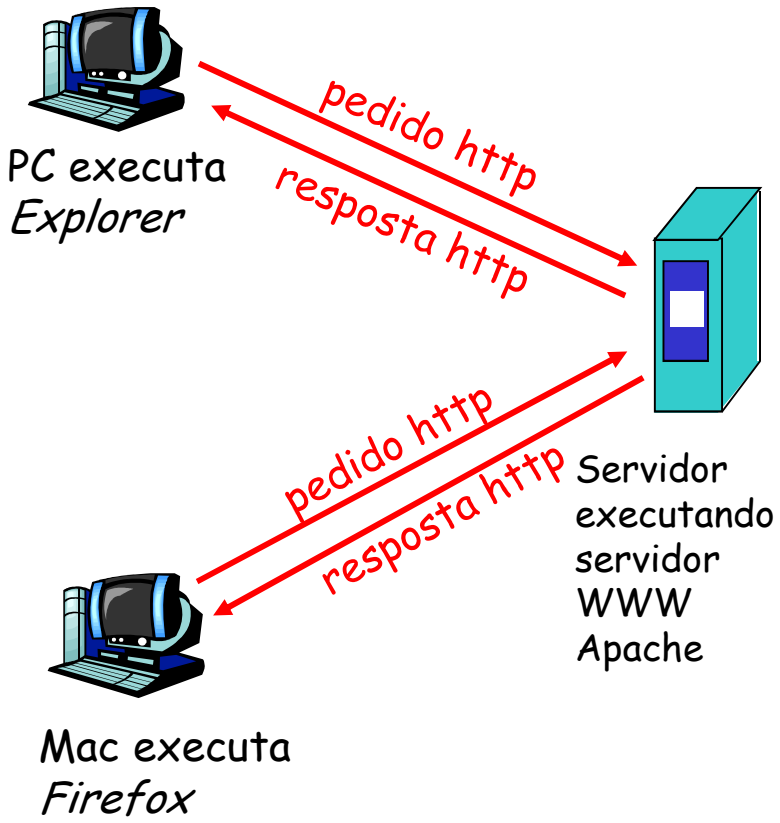
**adriano.acmesecurity.org/themes/noprob/logo_cnpq.png**

UEMG FRUTAL

# WWW: o protocolo http

*http: Hypertext Transfer Protocol*

- Protocolo da **camada de aplicação** para WWW.
- Modelo cliente-servidor
  - *cliente:* *browser* que solicita, recebe ("visualiza") **objetos** WWW.
  - *servidor:* servidor WWW **envia objetos** em resposta a pedidos.
- http1.0: RFC 1945
- http1.1: RFC 2068



# Mais sobre o protocolo http:

Usa serviço de transporte TCP:

1.) Cliente inicia conexão TCP (cria *socket*) ao servidor, na porta 80.

2.) Servidor aceita conexão TCP do cliente.

3.) Troca mensagens http (mensagens do protocolo da camada de aplicação) entre *browser* (cliente http) e servidor WWW (servidor http).

4.) **Encerra** conexão TCP.

http é "**sem estado**" ("***stateless***")

- Servidor não mantém informação sobre pedidos anteriores do cliente.

Protocolos que mantêm "**estado**" são **complexos**!

• O histórico passado (estado) tem que ser mantido.

• Demanda recursos maiores.

• São chamados de Protocolos "***Statefull***".

# Exemplo de http (1)

Exemplo: Um usuário digita a URL www.unesp.br/index.html

(Ex: contém texto, referências a 10 imagens jpeg)

1a. Cliente http inicia conexão TCP ao servidor http (processo) em www.unesp.br.

Porta 80 é padrão para servidor http.

1b. servidor http no hospedeiro www.unesp.br espera por conexão TCP na porta 80. "Aceita" conexão, avisando ao cliente

2. cliente http envia *mensagem de pedido* de http (contendo URL) através do *socket* da conexão TCP

3. servidor http recebe mensagem de pedido, elabora *mensagem de resposta* contendo objeto solicitado (index.html), e envia mensagem via *socket*.

tempo

Continua...

# Exemplo de http (2)

4. servidor http encerra conexão TCP .

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

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

tempo

E então, continua...

UEMG FRUTAL

# Conexões não-persistente e persistente

**Não persistente:**

- HTTP/1.0
- Servidor analisa pedido, responde, e encerra conexão TCP.
- 2 RTTs para trazer cada objeto (RTT = *round trip time*)
- Transferência de cada objeto sofre de **partida lenta**.

**Persistente:**

- Default no HTTP/1.1
- **Na mesma conexão** TCP: servidor analisa pedido, responde, analisa novo pedido, etc... → não fecha a conexão.
- Cliente envia pedidos para todos objetos referenciados, assim que recebe o HTML base .
- **Menos RTTs, e menos partida lenta.**

A maioria de browsers usa conexões TCP paralelas.

UEMG FRUTAL

# Conexão não-persistente (1)

- O cliente HTTP **inicia uma conexão TCP** com o servidor.
- O cliente **emite mensagem de requisição HTTP** usuário através do socket associado com a conexão do TCP que foi estabelecida.
- Servidor HTTP **recebe o *request* através do *socket*** associado com a conexão estabelecida, recupera o objeto `(index.html)` de seu armazenamento, encapsula o objeto em uma mensagem HTTP de resposta, e emite a mensagem de resposta ao cliente através do *socket*.
- O **servidor** diz ao cliente para **fechar a conexão TCP**.
- O cliente recebe a mensagem de resposta. **A conexão TCP termina**. A mensagem indica que o objeto encapsulado é um arquivo HTML. O cliente extrai o arquivo da mensagem de resposta, analisa, e encontra referências a 10 objetos do JPEG.
- **As primeiras quatro etapas são repetidas** então **para cada um dos objetos** JPEG referenciados.

UEMG FRUTAL

# Conexão não persistente

- Cada conexão TCP é encerrada após o servidor enviar o objeto, ela não persiste para os outros objetos. Usuários podem configurar browsers modernos para controlar o grau de paralelismo( conexões paralelas podem ser abertas).
- O tempo de requisição que transcorre entre a requisição e o recebimento de um arquivo-base HTTP por um cliente é denominado tempo de viagem de ida e volta (RTT), já inclui atrasos.

# Conexão não persistente

- Possui algumas desvantagens: uma nova conexão deve ser estabelecida e mantida para cada objeto solicitado. Para cada uma delas, devem ser alocados buffers TCP e conservadas variáveis TCP tanto no cliente quanto no servidor.

# Conexão não-persistente (2)

- Quando o browser recebe uma página, mostra a página ao usuário. Dois browsers diferentes podem interpretar (isto é, mostrar ao usuário) um webpage de maneiras um diferentes.
  - **O HTTP não define como uma webpage é interpretada por um cliente.**
  - As especificações do HTTP (RFC1945 e RFC2616) definem somente o protocolo de comunicação entre o programa HTTP do cliente e o programa HTTP do servidor.
- Nas etapas das conexões não persistentes, **cada conexão do TCP é fechada depois que o servidor envia o objeto**: a conexão não persiste para outros objetos.
- **Cada conexão TCP transporta exatamente uma mensagem de pedido e uma mensagem de resposta**. No nosso exemplo, quando um usuário requisita uma página, 11 conexões TCP são geradas.

(discutir conexões paralelas)

UEMG FRUTAL

# Round Trip Time (1)

- Quantidade de tempo gasta entre cliente **solicitar um arquivo HTML até que o arquivo esteja recebido**?
- Tempo ***round-trip-time*** (***RTT***), é o tempo gasto para um pacote viajar do cliente ao servidor e então voltar ao cliente. **É o tempo de ida-e-volta.**
- ***RTT*** inclui os atrasos de propagação, de enfileiramento e de processamento em routers e comutadores intermediários.
- Usuário clica num *hyperlink*. Isto faz com que o browser **inicie uma conexão do TCP entre o browser e o web server**.
- Envolve um "*three-way handshake*": o cliente emite uma mensagem TCP ao servidor, o servidor reconhece e responde com uma mensagem. Finalmente, o cliente confirma de volta ao servidor.

# Round Trip Time (2)

- Mais um RTT decorre após as duas primeiras partes do *three-way handshake*.
  - Após ter terminado as primeiras duas partes do *handshake*, **o cliente emite a mensagem de requisição HTTP na conexão TCP**, e o TCP "agrega" a última confirmação (a terceira parte do *three-way handshake*) na mensagem do pedido. Uma vez a mensagem do pedido chega no usuário, o usuário emite o arquivo HTML na conexão do TCP.
- Esta interação HTTP "*request-response*" **gasta outro RTT**.
- Assim, **o tempo de resposta total é dois RTTs** mais o tempo da transmissão do arquivo HTML pelo servidor.

# Conexão persistente

- Em conexões persistentes, o servidor deixa a conexão TCP aberta após enviar a resposta, requisições e repostas subsequentes entre os mesmos cliente e servidor podem ser enviadas por meio da mesma conexão. Normalmente, o servidor HTTP fecha uma conexão quando ela não usada durante um certo tempo.

# Conexão persistente (1)

- Cada objeto sofre dois RTTs:
  - um RTT para estabelecer a conexão TCP, e
  - um RTT para solicitar e receber um objeto.
- Cada objeto sofre do “partida lenta”(*slow start*) do TCP.
- **Com as conexões persistentes, o servidor deixa a conexão TCP aberta após ter emitido uma resposta.**
- Os pedidos e as respostas subseqüentes entre o mesmos cliente e servidor podem ser emitidos na mesma conexão.
- Uma *webpage* inteira (no exemplo, o arquivo HTML base e as 10 imagens) pode ser emitido sobre uma única conexão persistente do TCP.

# Conexão persistente (2)

- Duas versões de conexões persistentes:
- **sem *pipelining*** (paralelismo) e **com *pipelining***.
  - **Sem *pipelining***: o cliente emite um pedido novo somente quando a resposta precedente foi recebida.
  - **Com *pipelining***: o cliente emite um pedido assim que encontrar uma referência.
    - **É o default para HTTP/1.1.**
    - **Somente um RTT para todos os objetos solicitados.**

UEMG FRUTAL

# Mensagem http de **requisição** (1)

- Dois tipos de mensagem http: *pedido*, *resposta*
- Mensagem de pedido http:
  - ASCII (formato legível por humanos).

linha do pedido (comandos GET, POST, HEAD)

linhas do cabeçalho

```
GET /dir/page.html HTTP/1.1
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr
```

(*carriage return* (CR), *linefeed* (LF) adicionais)

Carriage return, line feed indica fim de mensagem

- Escrita em ASCII, é constituída de cinco linhas, cada uma seguida de um ‘carriage return’ e ‘line feed’. Embora esta mensagem tenha 5 linhas, uma mensagem pode ter muito mais ou menos que isso. A primeira linha de uma mensagem é denominada linha de requisição, as subsequentes são denominadas linhas de cabeçalho.

- O método GET é usado quando o browser requisita um objeto e este é identificado no campo do URL.
- Especifica o hospedeiro no qual o objeto se encontra.
- O browser está dizendo que não quer usar conexões persistentes.
- Especifica o agente de usuário.
- Mostra que o usuário prefere receber uma versão ,na língua especificada, do objeto se esse existir no servidor.

- HTTP/1.0 permite somente três tipos de métodos GET, POST e HEAD. Além desses três métodos, a especificação HTTP/1.1 permite vários métodos adicionais,

- **GET**: Método que solicita algum recurso ou objeto ao servidor
- **HEAD**: Solicita informações de um determinado objeto sem que esse seja enviado ao cliente apenas para testar a validade do último acesso.
- **POST**: Método usado para envio de arquivo/dados ou formulário HTML ao servidor.
- **OPTIONS**: Por meio desse método o cliente obtém as propriedades do servidor.
- **DELETE**: Informa por meio do URL o objeto a ser deletado.
- **TRACE**: Para enviar mensagem do tipo loopback para teste.
- **PUT**: Aceita criar ou modificar algum objeto do servidor.
- **CONNECT**: Comunicar com servidores Proxy.

# Mensagem http de **resposta** (1)

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

linhas de cabeçalho

dados, p.ex., arquivo html solicitado

```
HTTP/1.1 200 OK
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 ...
```

# Códigos de *status* da resposta http

Aparecem na primeira linha da mensagem de resposta cliente-servidor. Alguns códigos típicos:

**`200 OK`**

- Sucesso. Objeto pedido segue mais adiante nesta mensagem.

**`301 Moved Permanently`**

- Objeto pedido mudou de lugar, nova localização especificado mais adiante nesta mensagem (*Location*:)

**`400 Bad Request`**

- Mensagem de pedido não entendida pelo servidor.

**`404 Not Found`**

- Documento pedido não se encontra neste servidor.

**`505 HTTP Version Not Supported`**

- Versão de http do pedido não aceita por este servidor.

UEMG FRUTAL

# *Interação usuário-servidor: cookies*

- Cookies, permitem que sites monitorem seus usuários. A tecnologia dos cookies tem quatro componentes, uma linha de cabeçalho de cookie na mensagem de resposta HTTP ; uma linha de cabeçalho de cookie na mensagem de requisição HTTP; um arquivo de cookie mantido no sistema final do usuário e gerenciado pelo browser do usuário; um banco de dados de apoio no site Web.

# Conteúdo HTTP

- Além da Web, é usado para aplicações de comércio eletrônico para transferência de arquivos XML de uma máquina a outra; para transferir Voice XML, WML e frequentemente é usado como o protocolo de transferência de arquivos no compartilhamento de arquivos P2P.

# HTML (HyperText Markup Language)

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

UEMG FRUTAL

- Um cachê Web servidor proxy) é uma entidade da rede que atende requisições HTTP em nome de um servidor Web de origem. Tem seu próprio disco de armazenamento, e mantém ,dentro dele, cópias de objetos recentemente requisitados.

- O browser de um usuário pode ser configurado de modo que todas as suas requisições HTTP sejam dirigidas primeiramente ao cache Web. Quando um browser requisita um objeto, ele estabelece uma conexão TCP com o cachê Web e envia a ele uma requisição HTTP para um objeto, o cachê verifica se tem uma cópia do objeto armazenada, se tiver, a envia para o browser do cliente, se não tiver, o cache abre uma conexão TCP com o servidor de origem, envia uma requisição do objeto para a conexão TCP, após receber essa requisição, o servidor de origem envia o objeto ao cache, quando este recebe o objeto, ele guarda uma cópia em seu armazenamento local e envia outra, ao browser do cliente.

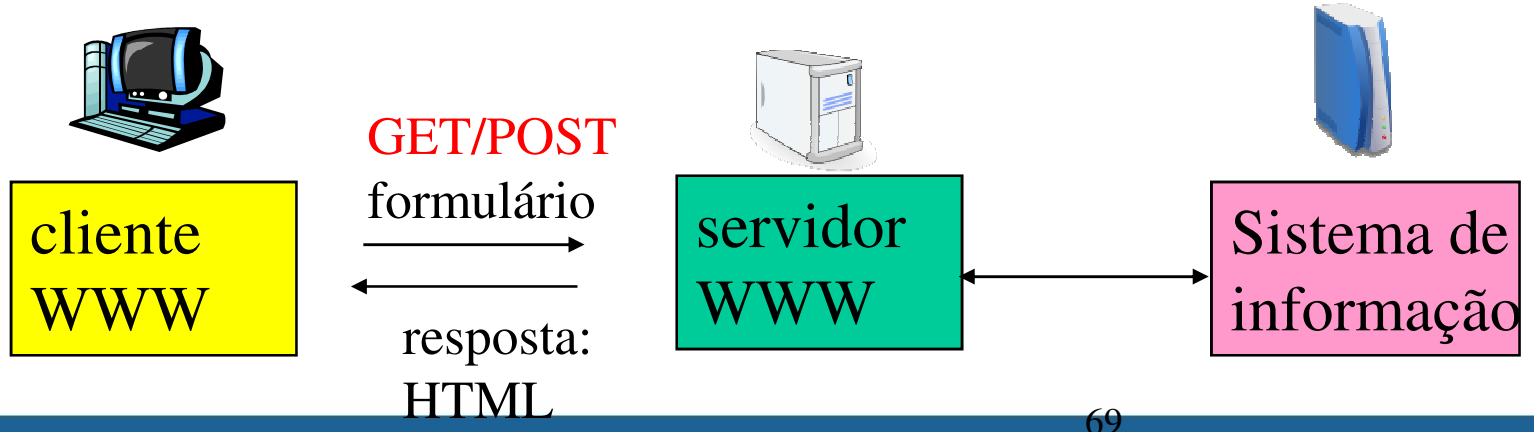

- Um cache é, ao mesmo tempo, um servidor(quando recebe requisições de um browser) e um cliente(quando envia requisições para um servidor de origem).
- O cache na Web tem sido utilizado amplamente na Internet por duas razões: um cache Web pode reduzir substancialmente o tempo de resposta para a requisição de um cliente; caches Web podem reduzir substancialmente o tráfego no enlace de acesso de uma instituição qualquer à Internet e também o da Internet como um todo, melhorando o desempenho para todas as aplicações.

# Web: Formulários e páginas dinâmicas

(interação bidirecional e integração com banco de dados)

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

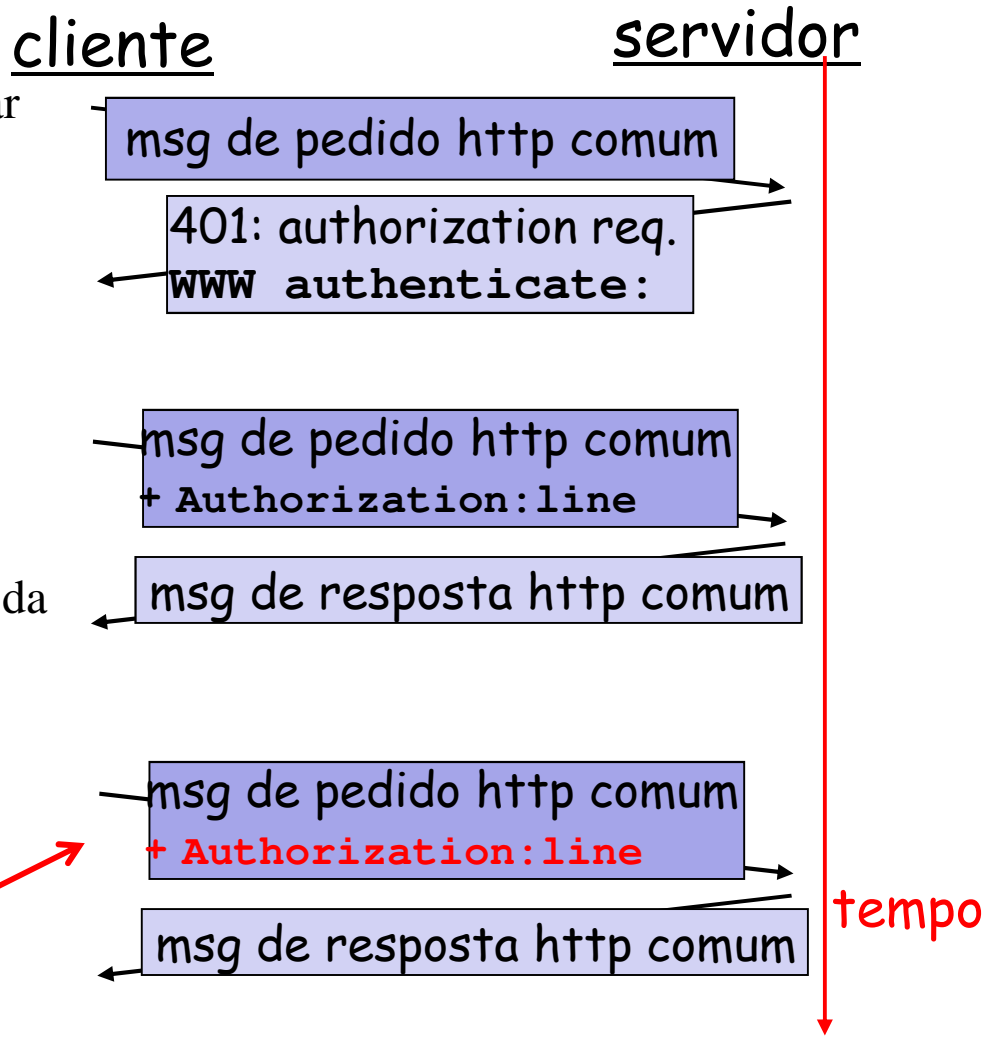
- *Forms*: processados usando *scripts* (programas que rodam no servidor WWW)
  - *scripts* permitem acesso a diferentes serviços.
  - servidor WWW atua como *gateway* universal.



# Web: autenticação

- Autenticação: controle de acesso ao servidor.
- Sem estado: cliente deve apresentar autorização em cada pedido.
- Autorização: tipicamente nome e senha.
  - `authorization:` linha de cabeçalho no pedido.
  - Se não for apresentada autorização, servidor nega acesso, e coloca no cabeçalho da resposta

    `WWW authenticate:`




*Browser:* cache nome e senha para evitar que sejam pedidos ao usuário a cada acesso.

# Web: *cookies*

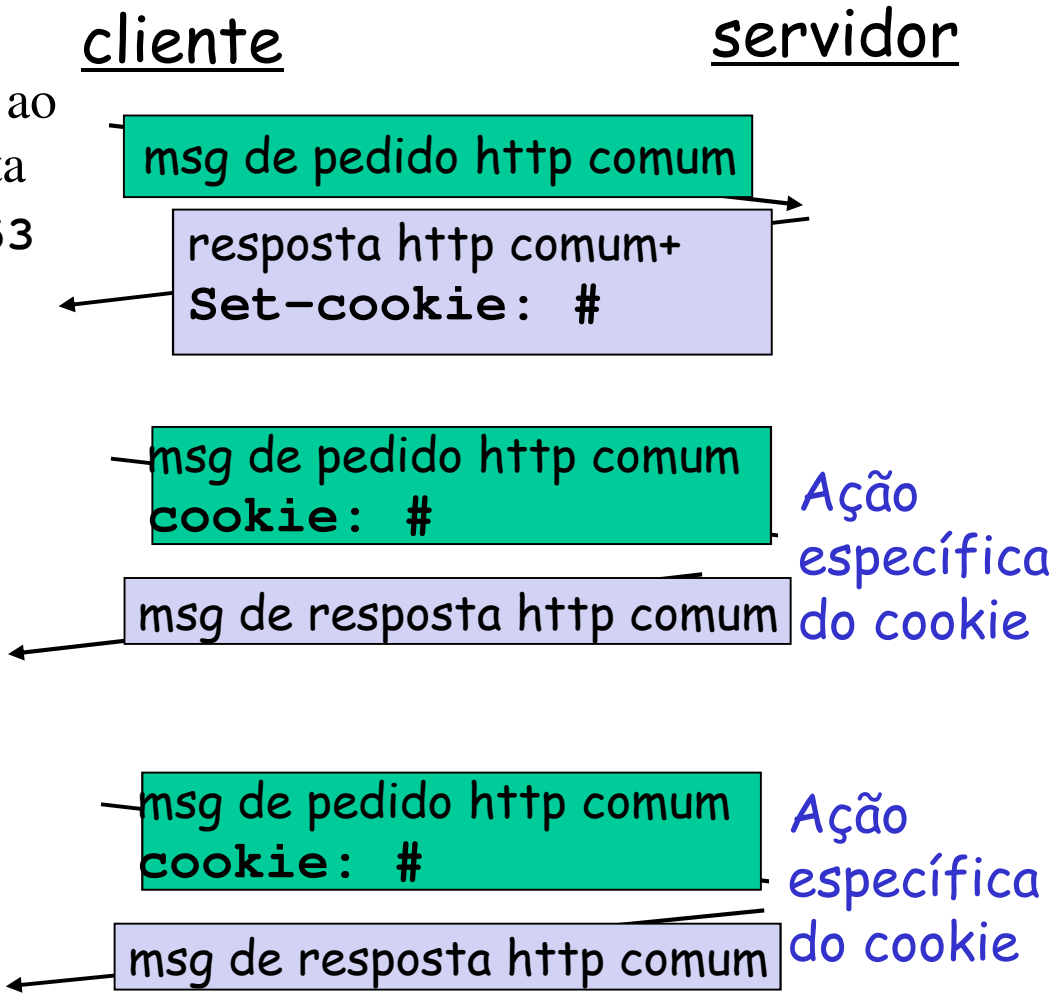
- **Servidor** envia "*cookie*" ao cliente na msg de resposta

  `Set-cookie: 1678453`

- **Cliente** apresenta *cookie* nos pedidos posteriores

  `cookie: 1678453`

- **Servidor** casa o *cookie* apresentado com a info guardada no servidor.
  - Autenticação.
  - Lembrando preferências do usuário, opções anteriores, etc…



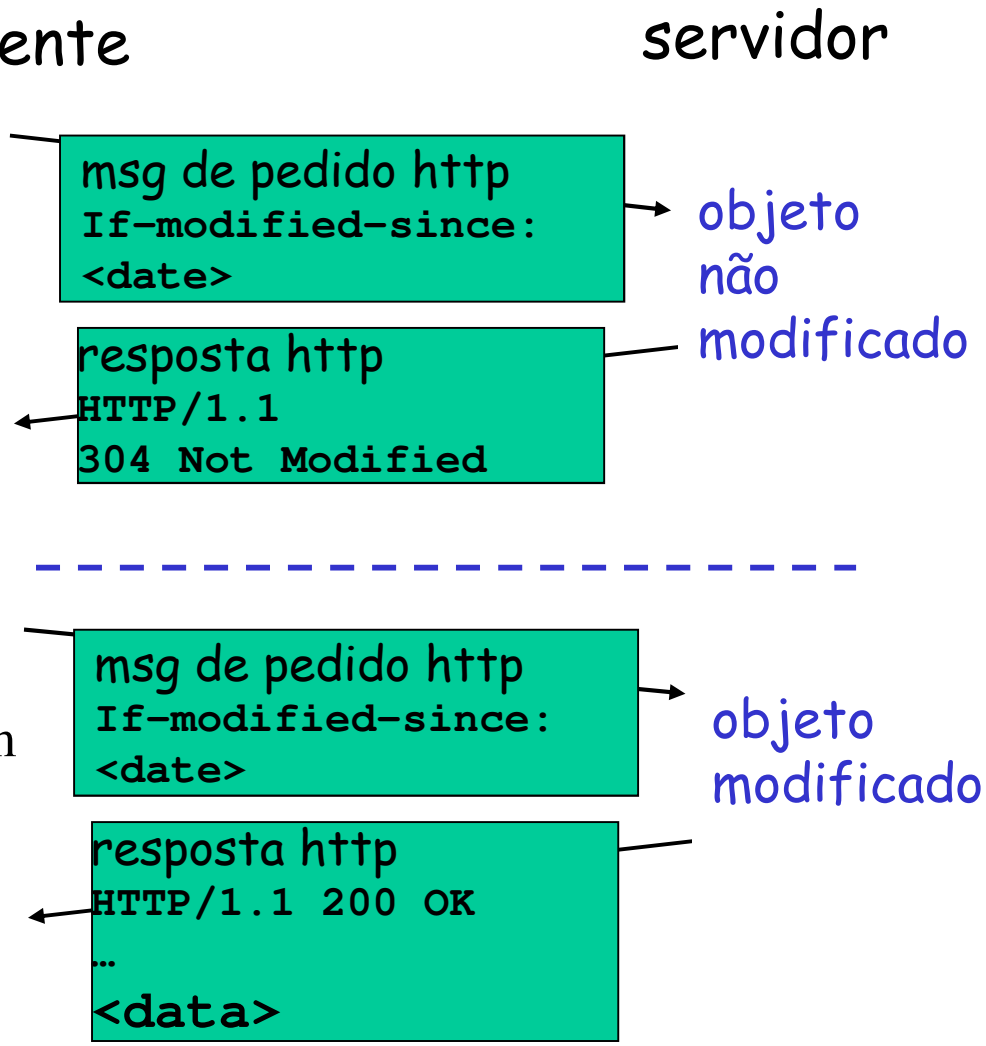
# Web: *GET* condicional

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

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

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

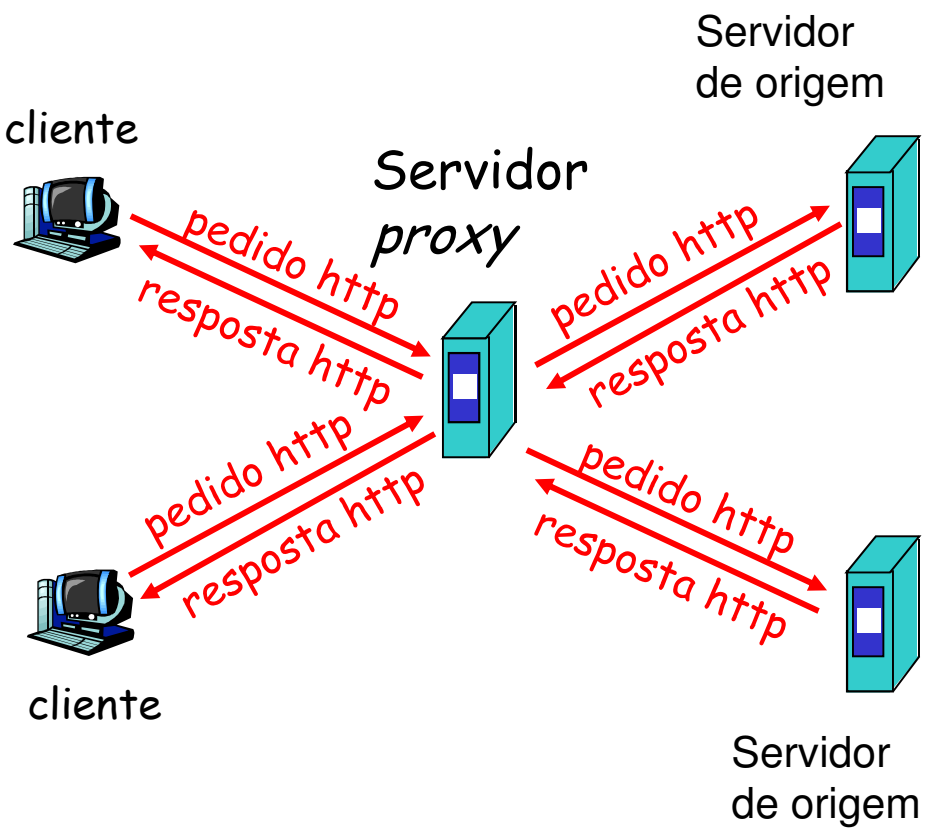
```
HTTP/1.1 304 Not
   Modified
```



# Web: *cache proxy*

Meta: atender pedido do cliente sem envolver servidor de origem.

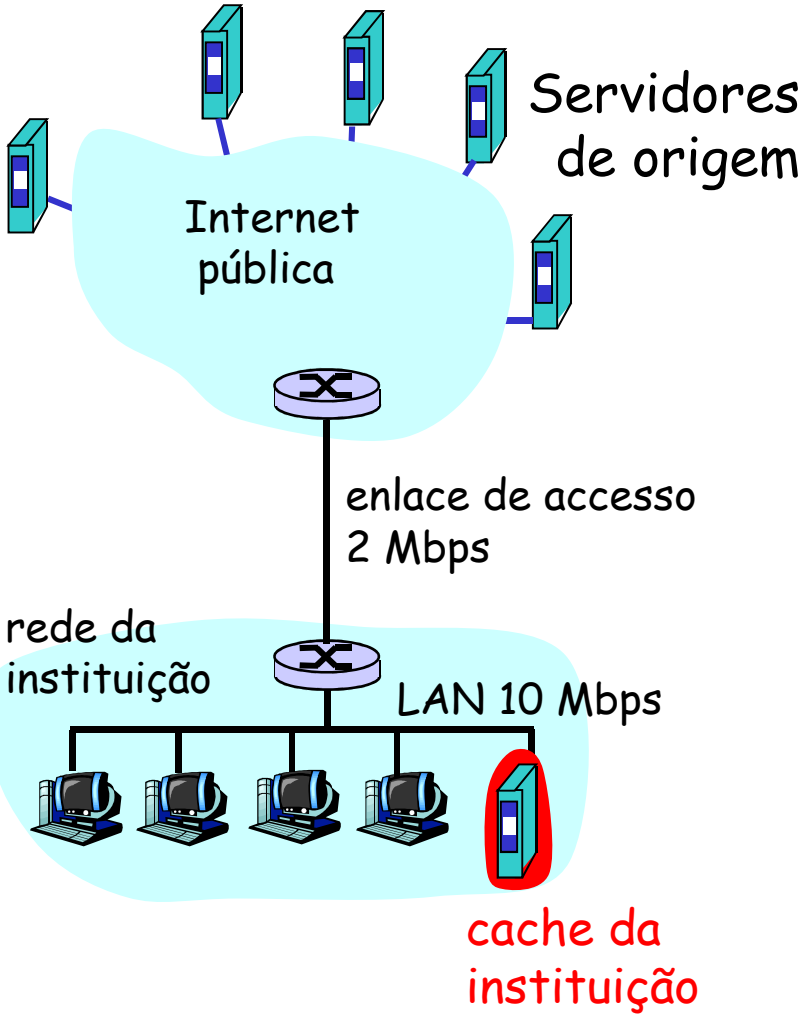
- Usuário configura browser: acessos WWW via procurador (*proxy*).
- Cliente envia todos pedidos http ao procurador.
  - Se objeto está no cache do procurador, este o devolve imediatamente na resposta http.
  - Senão, solicita objeto do servidor de origem, armazena e depois devolve resposta http ao cliente.



# Por que usar cache WWW ?

Suposição: cache está "próximo" do cliente (por exemplo na mesma rede).

- Tempo de resposta menor.
- Diminui tráfego aos servidores distantes,
  - Muitas vezes é o gargalo é o enlace (*link*) que liga a rede da instituição até a Internet.



# FTP

- O usuário quer transferir arquivos de ou para um hospedeiro remoto, para acessar a conta remota, ele deve fornecer uma identificação e uma senha, após fazer isso, ele pode transferir arquivos do sistema local de arquivos para o sistema remoto e vice-versa. O usuário interage com o FTP por meio de um agente de usuário FTP. Primeiro, ele fornece o nome do hospedeiro remoto(estabelece conexão TCP com o processo servidor), depois ele fornece sua identificação e senha sendo enviadas pela conexão TCP.

# FTP

- O HTTP e o FTP são protocolos de transferência de arquivos e têm muitas características em comum: por exemplo, ambos utilizam o TCP.Mas também possuem diferenças importantes, o FTP usa duas conexões TCP paralelas para transferir um arquivo, uma de controle e uma de dados. A primeira é usada para enviar informações de controle entre os dois hospedeiros, a segunda é usada para efetivamente enviar um arquivo.

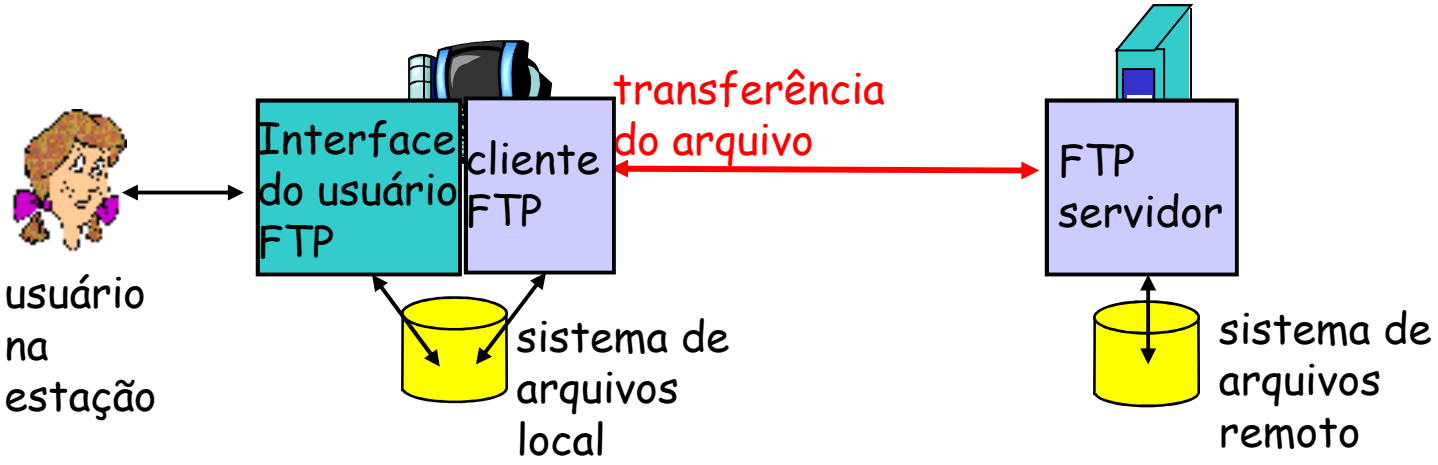
# FTP

- Como o FTP usa uma conexão de controle separada, dizemos que ele envia suas informações de controle fora da banda. O HTTP envia linhas de cabeçalho de requisição e de resposta pela mesma conexão TCP que carrega o próprio arquivo transferido, por essa razão diz-se que ele envia suas informações de controle na banda.

# FTP

- O FTP envia exatamente um arquivo pela conexão de dados e em seguida a fecha, se o usuário quiser enviar um outro arquivo, o FTP abrirá outra conexão de dados. Durante uma sessão, o servidor FTP deve manter informações de estado sobre o usuário, ele deve associar a conexão de controle com uma conta de usuário específica e também deve monitorar o diretório corrente do usuário enquanto este passeia pela arvore do diretório remoto.

# FTP: o protocolo de transferência de arquivos

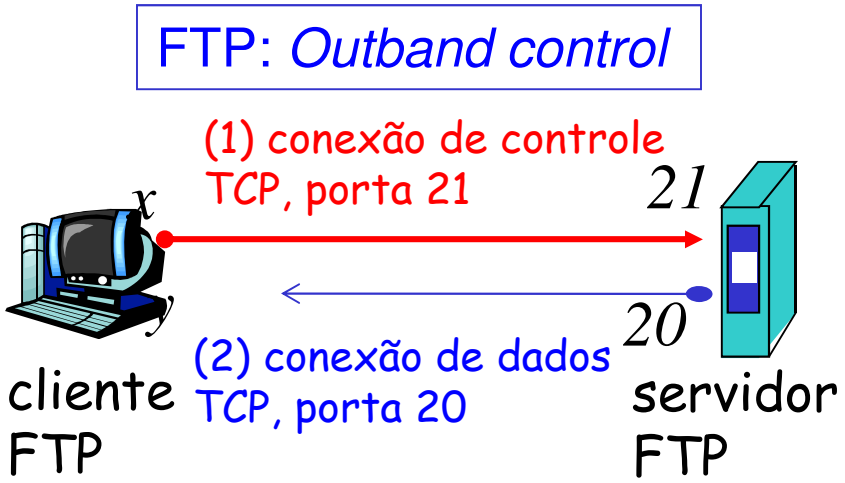



- Transferir arquivo de/para hospedeiro remoto
- Modelo cliente/servidor
  - *cliente:* lado que inicia transferência.
  - *servidor:* host remoto.
- FTP - *File Transfer Protocol*: definido pelo [RFC 959]
- Servidor FTP: atende na porta 21

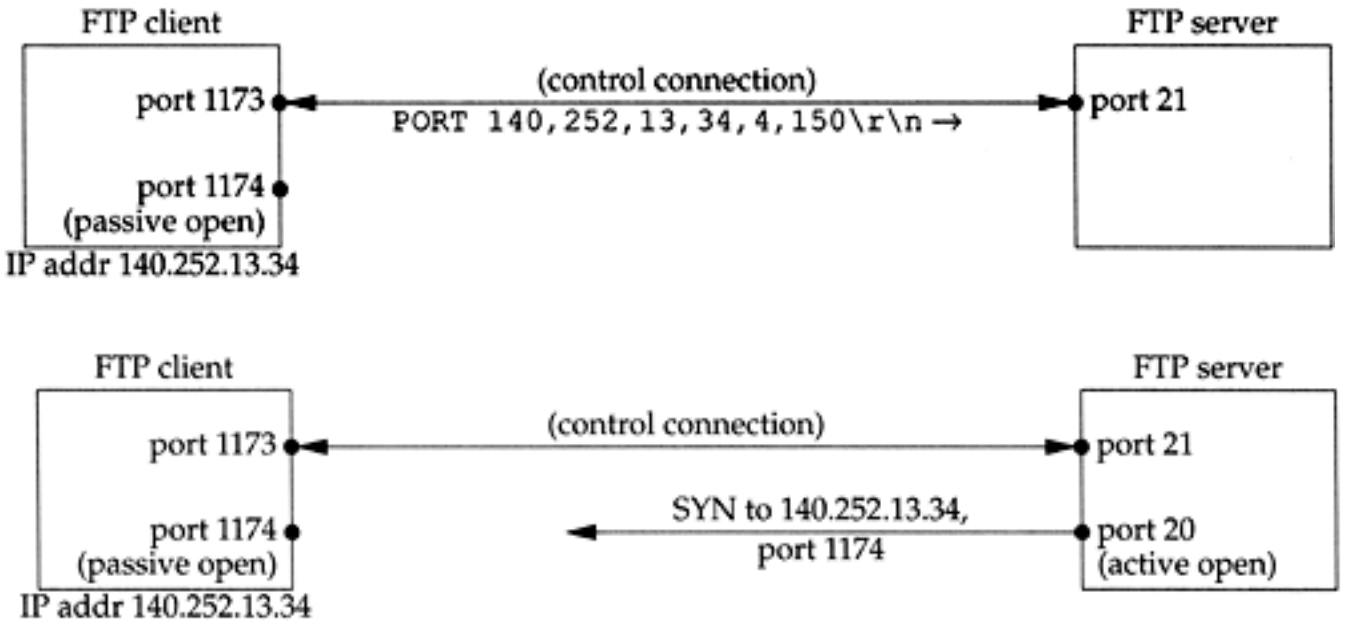
# FTP: conexões separadas para controle e dados

- Cliente FTP: TCP como protocolo de transporte.
- **São abertas duas conexões TCP paralelas:**
  - Controle: troca comandos, respostas entre cliente, servidor (porta 21).

    "controle fora da banda"
  - Dados: dados de arquivo de/para servidor (porta 20).
- Servidor ftp mantém "estado": diretório corrente, e autenticação realizada.

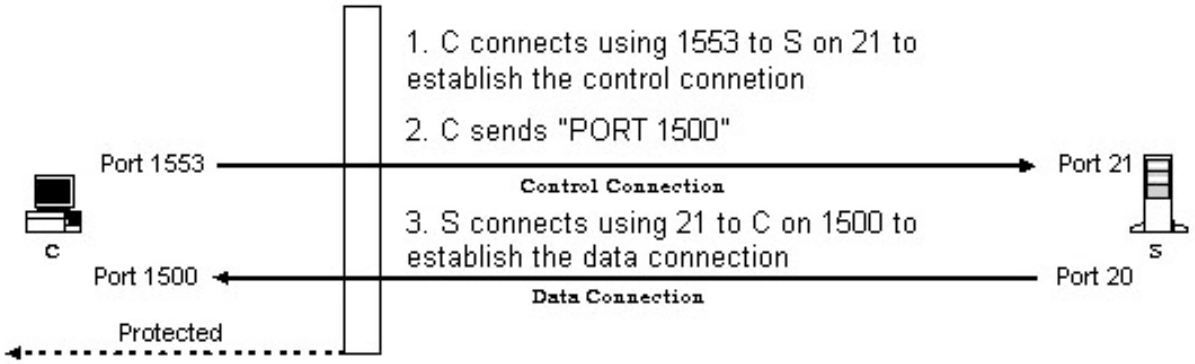


# FTP: conexões separadas para controle e dados

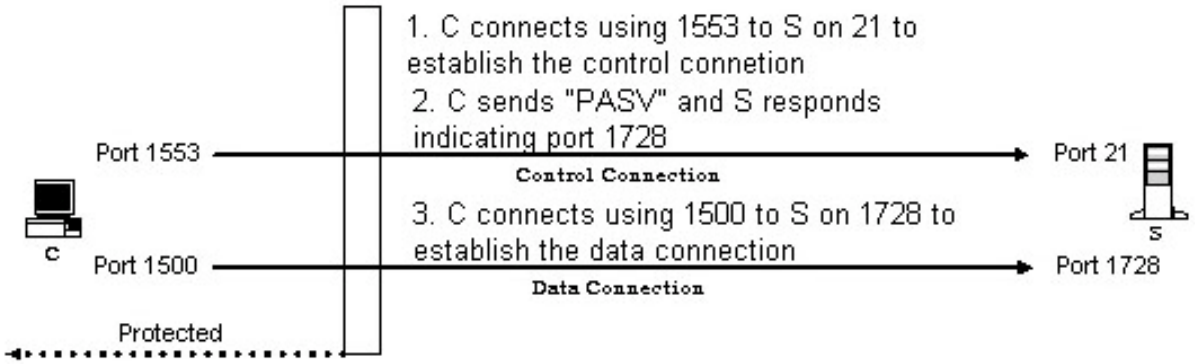



Richard Stevens, TCP/IP Illustrated Vol. 1 Fig. 27.5

# FTP *Active Open & Passive Open*




**Active Data Transmission**




**Passive Data Transmision**

http://pintday.org/whitepapers/ftp.jpg 8.03.2010

# FTP: comandos e respostas

## Comandos típicos:

- Enviados em texto ASCII pelo canal de **controle.**
- **`USER`** ***`nome`***
- **`PASS`** ***`senha`***
- **`LIST`** devolve lista de arquivos no diretório corrente
- **`RETR arquivo`** recupera (lê) arquivo remoto
- **`STOR arquivo`** armazena (escreve) arquivo no *host* remoto.

## Códigos de retorno típicos

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

UEMG FRUTAL

# Correio eletrônico

- Tal como o correio normal, o e-mail é um meio de comunicação assíncrono, rápido, fácil de distribuir e barato. No sistema de correio da Internet há três componentes, os agentes de usuários, servidores de correio e o SMTP. Servidores de correio formam o núcleo da infraestrutura do e-mail, cada destino. Cada destinatário tem uma caixa postal localizada em um dos servidores de correio.

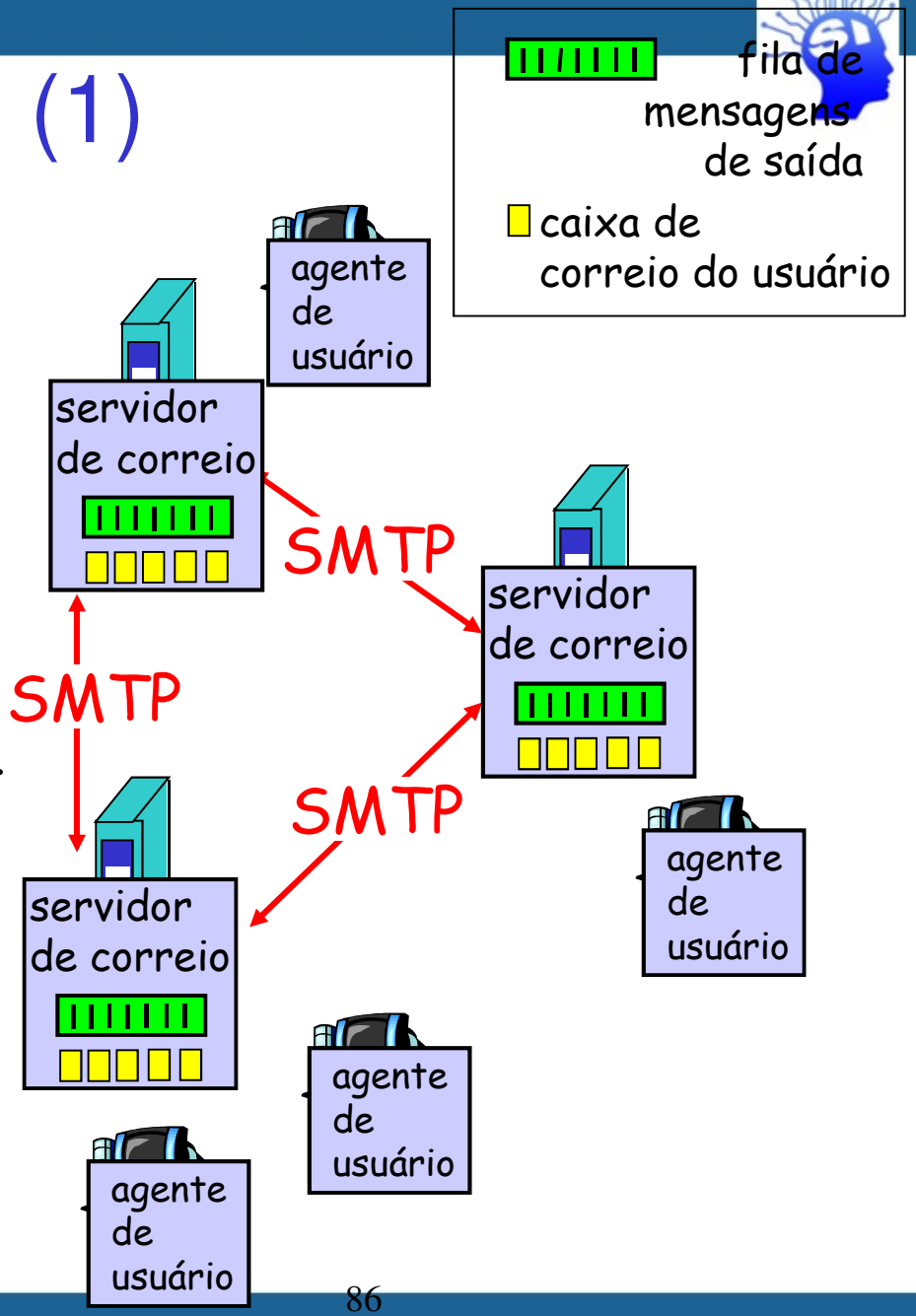
# Correio eletrônico

- Uma mensagem típica inicia sua jornada no agente de usuário do remetente, vai até o servidor de correio dele e viaja até o servidor de correio do destinatário, onde é depositada na caixa postal. Quando o destinatário que acessar as mensagens de sua caixa postal, o servidor de correio que contem sua caixa postal o autentica. Se o servidor de correio do remetente não puder entregar a correspondência ao servidor dele, manterá a mensagem em uma fila de mensagens e tentará transferi-la depois.

# Correio Eletrônico (1)

Três grandes componentes:

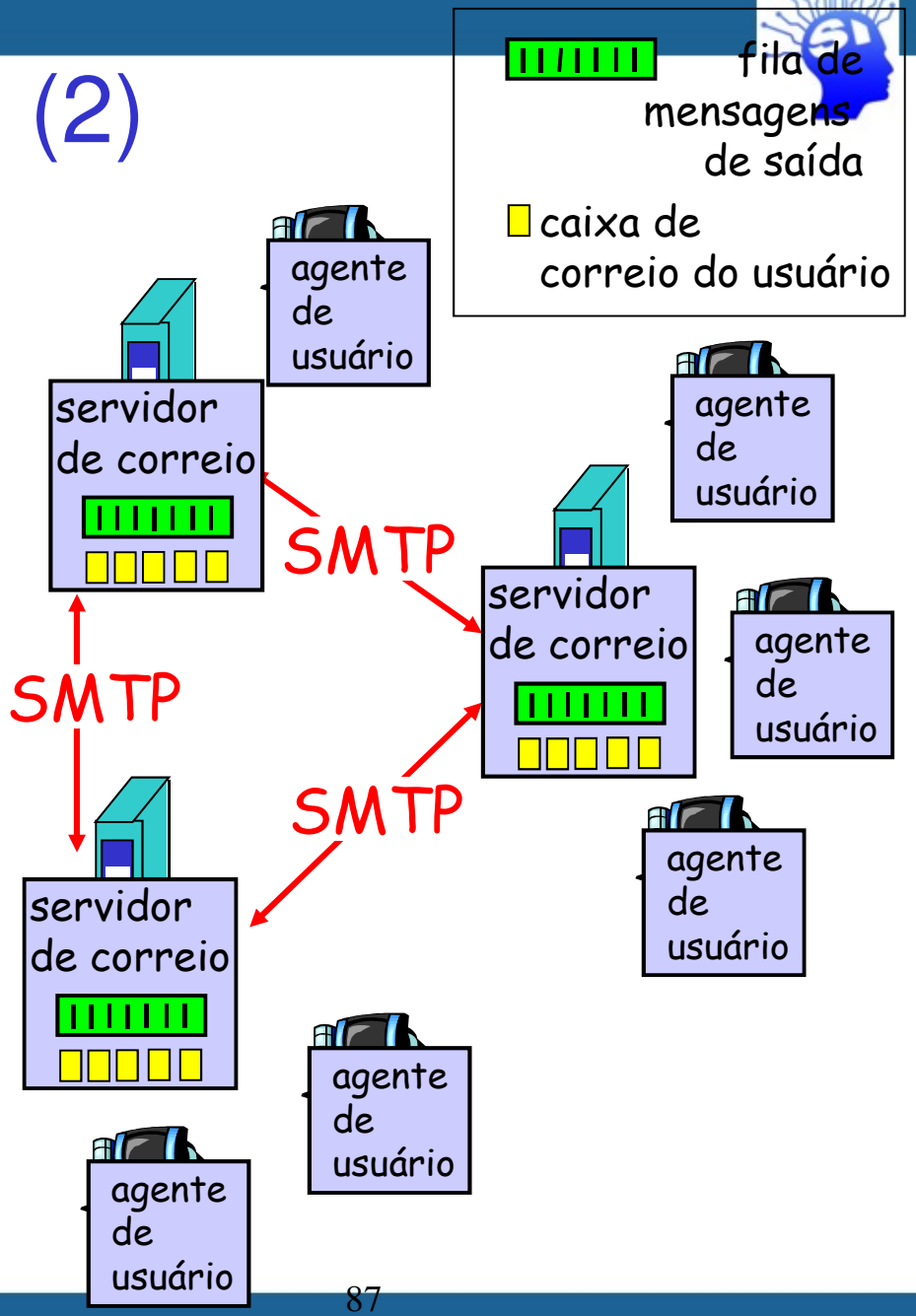
- Agentes de usuário:
  - *Mail User Agent* (MUA).
- Agente de transporte: Servidores de correio
  - *Mail Transport Agent* (MTA).
- Protocolo de correio:
  - *Simple Mail Transfer Protocol* (SMTP).



# Correio Eletrônico (2)

MUA - Agente de Usuário

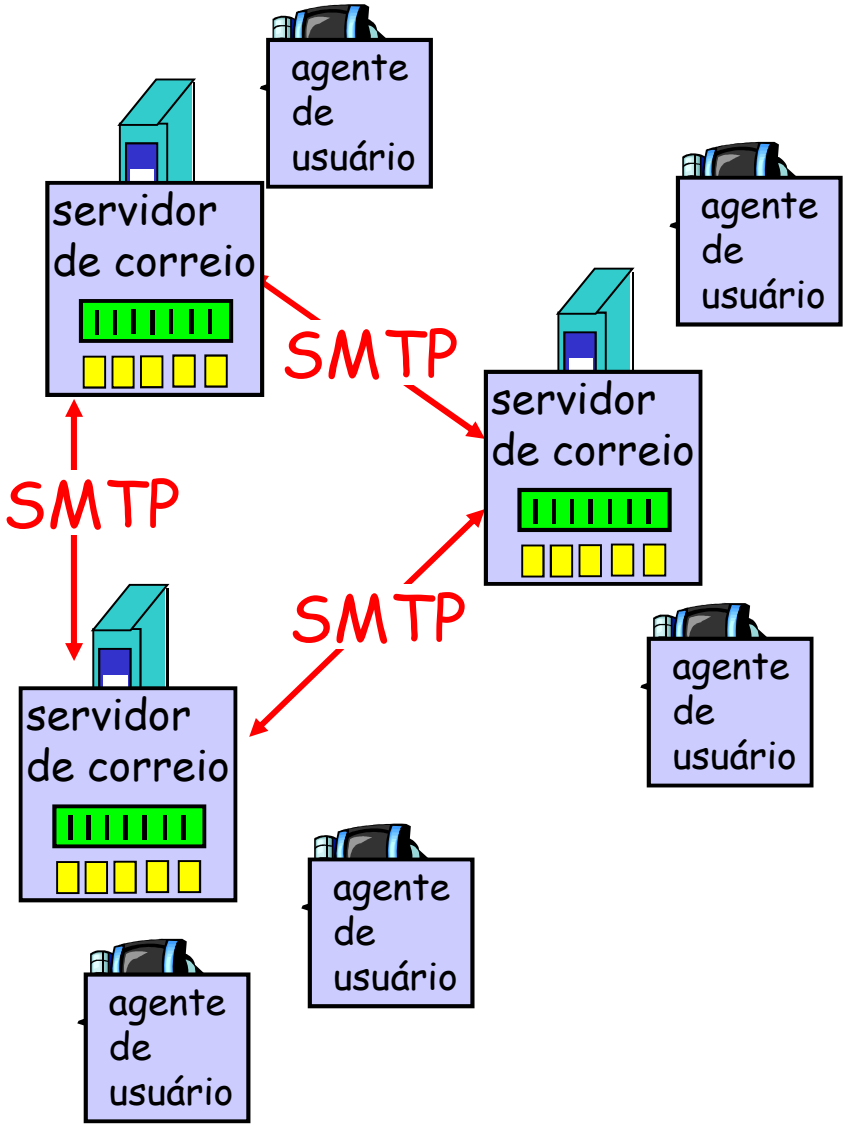
- Conhecido como “leitor de e-mail”.
- É o lado “cliente”.
- Compor, editar, ler mensagens de correio
- Exemplo: Eudora, Outlook, elm, Pegasus, Netscape Messenger, etc...
- Mensagens de saída e chegada são armazenadas no servidor.



# Correio Eletrônico: servidores

## Servidores de correio

- Caixa de correio contém mensagens de chegada (ainda não lidas) p/ usuário.
- Fila de mensagens contém mensagens de saída (a serem enviadas).
- Protocolo SMTP entre servidores para transferir mensagens de correio.
  - Cliente: servidor de correio que envia
  - "Servidor": servidor de correio que recebe.



# SMTP

- O SMTP transfere mensagens de servidores de correio remetentes para servidores de correio destinatários. O SMTP é uma tecnologia antiga que possui certas características arcaicas, por exemplo, restringe o corpo de todas as mensagens de correio ao simples formato ASCII de 7 bits.

# SMTP

- Enviando uma mensagem:
  - O remetente chama seu agente de usuário para email, fornece o endereço do destinatário, compõe uma mensagem e instrui o agente a enviar a mensagem.
  - O agente de usuário do remetente envia a mensagem para o seu servidor de correio, onde ela é colocada em uma fila de mensagens.
  - O lado cliente do SMTP, que funciona no servidor de correio do remetente, vê a mensagem na fila e abre uma conexão TCP para um servidor SMTP, que funciona no servidor de correio do destinatário.

# SMTP

- Enviando uma mensagem (cont.):
  - Após alguns procedimentos iniciais de apresentação, o cliente SMTP envia a mensagem do remetente para dentro da conexão TCP.
  - No servidor de correio do destinatário, o lado servidor SMTP recebe a mensagem e a coloca na caixa postal dele.
  - O destinatário chama seu agente de usuário para ler a mensagem quando for mais conveniente para ele.

# SMTP

- O SMTP não usa servidores de correio intermediários para enviar correspondência, mesmo quando os dois servidores estão localizados em lados opostos do mundo.
- Como o SMTP transfere uma mensagem de um servidor de correio remetente para um servidor de correio destinatário:

# SMTP

- O Cliente SMTP faz com que o TCP estabeleça uma conexão na porta 25 com o servidor SMTP, se o servidor não estiver em funcionamento, o cliente tenta novamente depois, senão, o servidor e o cliente trocam alguns procedimentos de apresentação, assim que acabam de se apresentar, o cliente envia a mensagem.

# Correio Eletrônico: SMTP [RFC 821]

- Usa TCP para a transferência confiável de mensagens de correio do cliente ao servidor. Porta 25/TCP
- Transferência direta: servidor remetente ao servidor receptor.
- Três fases da transferência:
  - *Handshaking* (cumprimento).
  - Transferência das mensagens
  - Encerramento
- Interação comando/resposta
  - Comandos: texto ASCII
  - Resposta: código e frase de status
- Mensagens precisam ser em ASCII de 7-bits.

UEMG FRUTAL

# Interação SMTP típica

```
S: 220 doces.br
C: HELO consumidor.br
S: 250  Hello consumidor.br, pleased to meet you
C: MAIL FROM: <ana@consumidor.br>
S: 250 ana@consumidor.br... Sender ok
C: RCPT TO: <bernardo@doces.br>
S: 250 bernardo@doces.br ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Eu gostaria de comprar alguns doces.
C: Você tem uma lista de preços? Obrigada. -Ana.
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 doces.br closing connection
```

## Exercício: experimente uma interação SMTP com seu servidor de e-mail.

- **`telnet nomedoservidor.algumlugar.br 25`**
- Observe a resposta 220 do servidor
- Entre comandos HELO, MAIL FROM, RCPT TO, DATA, QUIT, HELP.

Estes comandos permitem que você envie correio sem usar um cliente (leitor de correio). Basta conhecer o formato das mensagens do protocolo.

Pesquisar a respeito das mensagens de protocolo mais usadas no SMTP.

UEMG FRUTAL

# SMTP: detalhes

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

Exercício: faça uma comparação entre o SMTP e o http.

- HTTP: pull (puxar)?
- E-mail: push (empurrar)?
- Ambos tem interação comando/resposta, e códigos de status em ASCII?
- HTTP: cada objeto é encapsulado em sua própria mensagem de resposta?
- SMTP: múltiplos objetos de mensagem enviados numa mensagem de múltiplas partes?
- Discuta outras questões que achar relevantes.

UEMG FRUTAL

# Mensagem de e-mail

- Formato: Possui um cabeçalho contendo informações (From, To, Subject) e o corpo da mensagem.

- A extensão MIME para dados que não seguem o padrão ASCII: Para enviar conteúdo que não seja texto ASCII, o agente de usuário remetente deve incluir cabeçalhos adicionais na mensagem, esses cabeçalhos são Content-Type (permite que o agente de usuário destinatário realiza uma ação adequada sobre a mensagem) e Content-Transfer-Enconding (converter o corpo da mensagem à sua forma original).

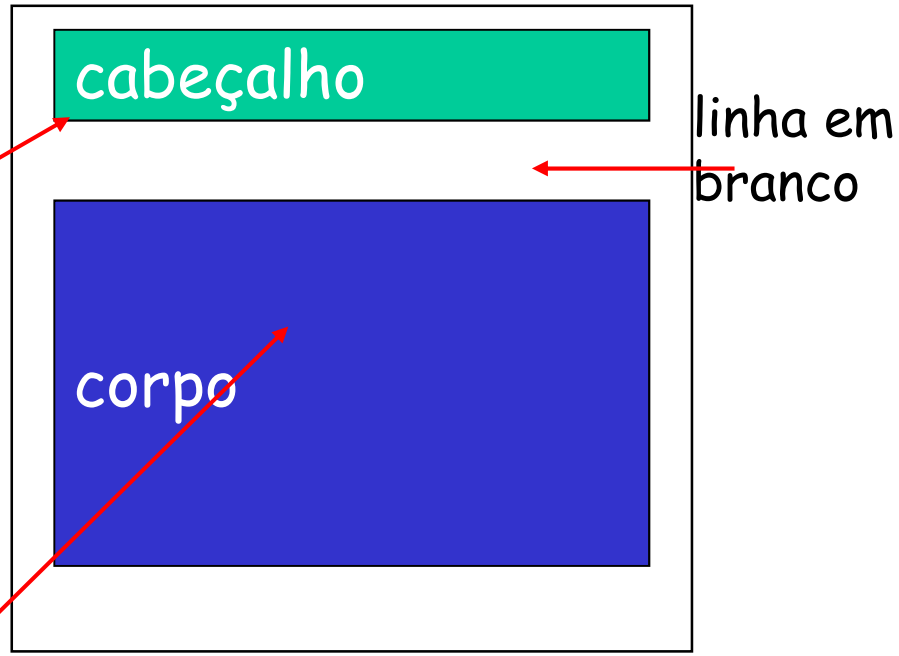
# Mensagem de e-mail

- O cabeçalho Received: Especifica o nome do servidor SMTP que enviou a mensagem, o do que recebeu e o horário em que o servidor destinatário recebeu a mensagem.

# Formato de uma mensagem de e-mail

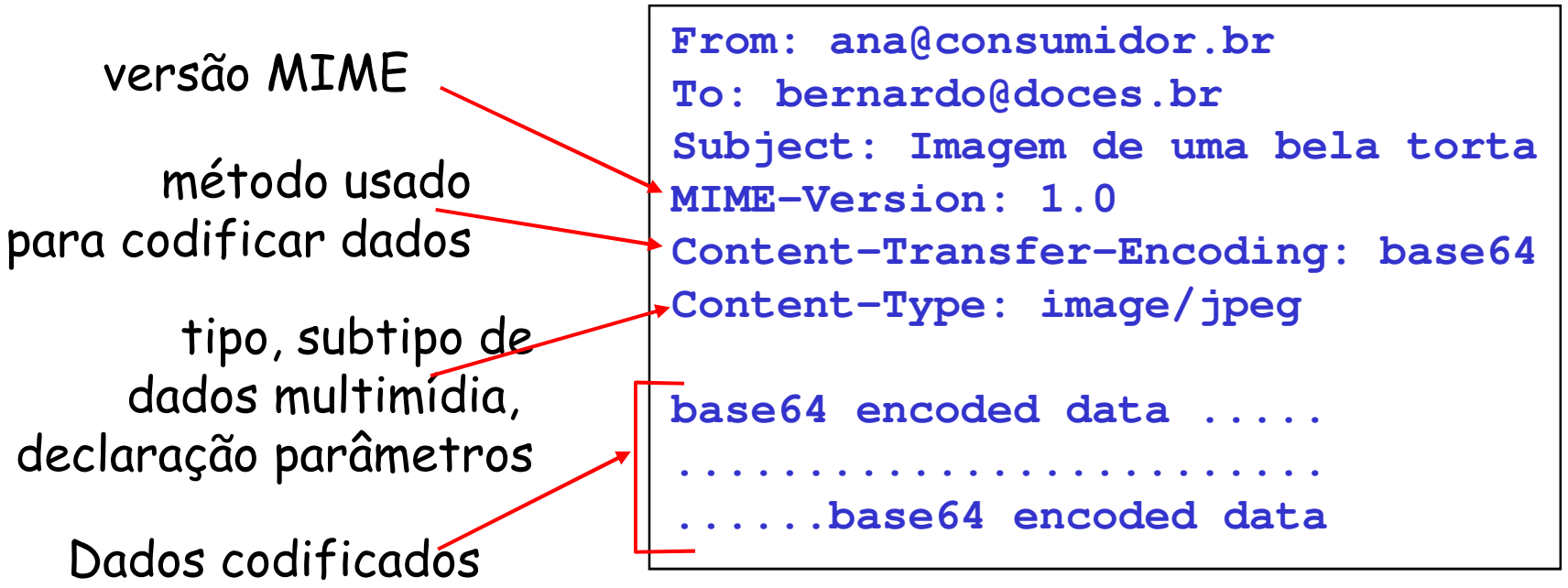
- RFC 822: **padrão para formato de mensagem**:
- Linhas de cabeçalho (*header*):
  - *To:*
  - *From:*
  - *Subject:*

  (NÃO são os comandos de smtp)
- Corpo
  - É a "mensagem".
  - Somente de caracteres ASCII .
  - Termina com um "." ponto



## Formato de uma mensagem: extensões para multimídia

- MIME: *multimedia mail extension*, RFC 2045, 2056
- Linhas adicionais no cabeçalho da mensagem declaram tipo do conteúdo MIME.



# Tipos MIME

**`Content-Type: tipo/subtipo; parâmetros`**

## Text

- sub-tipos exemplos: **`plain`**, **`html`**
- **`charset="iso-8859-1"`**, **`ascii`**

## Audio

- Sub-tipos exemplos : **`basic`** (8-bit codificado mu-law), **`32kadpcm`** (codificação 32 kbps).

## Image

- sub-tipos exemplos : **`jpeg`**, **`gif`**

## Application

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

## Video

- sub-tipos exemplos : **`mpeg`**, **`quicktime`**

UEMG FRUTAL

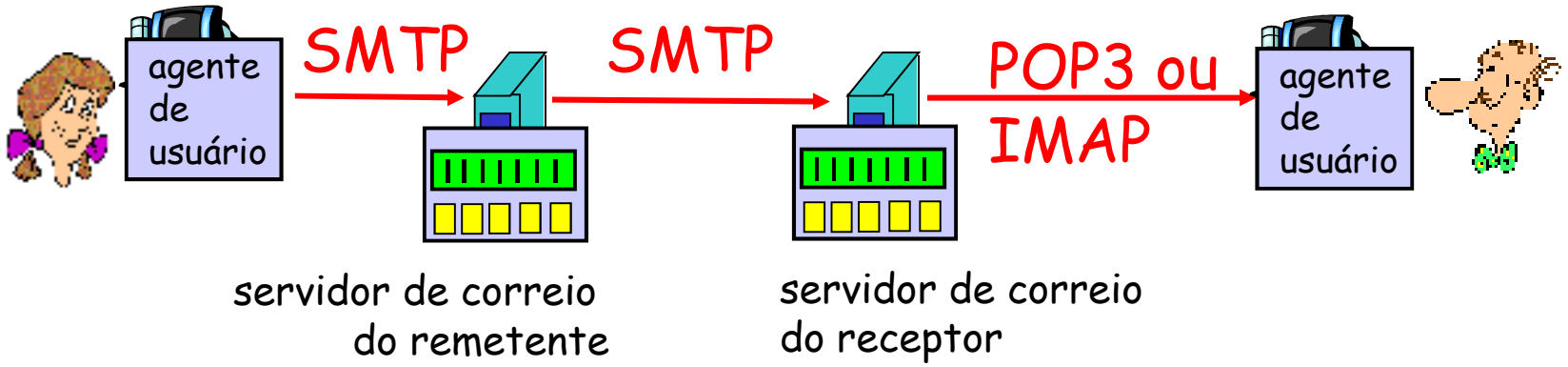
# Tipo Multiparte

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

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

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

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

# Protocolos de acesso ao correio

- Como o agente de usuário do destinatário não pode usar SMTP para obter as mensagens por que essa é uma operação de recuperação e o SMTP é um protocolo de envio, existem protocolos especiais que acessam o correio, entre eles o POP3, IMAP e HTTTP.

# Protocolos de acesso ao correio




- SMTP: entrega/armazenamento no servidor do receptor.
- Protocolo de acesso ao correio: recupera do servidor.
  - **POP**: *Post Office Protocol* [RFC 1939]
    - Autorização (agente <-->servidor) e transferência
  - **IMAP**: *Internet Mail Access Protocol* [RFC 1730]
    - Mais comandos e mais opções (mais complexo).
    - Manuseio de msgs armazenadas no servidor

Através de HTTP: Hotmail , Yahoo! Mail, Webmail, etc. (não é exatamente um "protocolo" de e-mail e sim um mecanismo)

# POP3

- Protocolo de acesso de correio extremamente simples, possui funcionalidade limitada. O POP3 começa quando o agente de usuário(o cliente) abre uma conexão TCP com o servidor de correio(o servidor) na porta 110. Com a conexão ativada, o protocolo passa por três fases: autorização(o agente de usuário envia um nome de usuário e uma senha para autenticar o usuário), transação(recupera mensagens, o agente de usuário pode marcar mensagens que devem ser apagadas,), atualização(apaga as mensagens que foram marcadas).

# POP3

- Em uma transação POP3, o agente de usuário emite comandos e o servidor, uma resposta para cada um deles, há duas respostas possíveis +OK (quando ocorreu tudo bem) e –ERR:(informa que houve algo de errado). A fase de autorização tem dois comandos principais user e pass. O servidor POP3 mantém como informação de estado apenas quais mensagens devem ser apagadas. O modo ler-e-apagar não permite que o usuário acesse uma mensagem já lida através de um outro computador. O modo ler-e-guardar permite que isso aconteça.

# Protocolo POP3

## Fase de autorização

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

## Fase de transação, cliente:

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

```
S: +OK POP3 server ready
C: user ana
S: +OK
C: pass faminta
S: +OK user successfully logged on
```

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

UEMG FRUTAL

# IMAP

- Protocolo com mais recursos e mais complexo que o POP3. Um servidor IMAP associa cada mensagem a uma pasta. Este protocolo provê comandos que permitem que os usuários criem pastas e transfiram mensagens de uma para outra e comandos que os usuários podem usar para pesquisar pastas remotas em busca de mensagens que obedeçam a critérios específicos.Mantém informação de estado de usuário. Tem comandos que permitem que um agente de usuário obtenha componentes de mensagens.

# E-mail pela Web

- O agente de usuário é um browser Web comum e o usuário se comunica com sua caixa postal via HTTP. E quando se quer enviar uma mensagem de e-mail, esta é enviada do browser do remetente para seu servidor de correio e não por SMTP.

# DNS: o serviço de diretório da Internet

- Hospedeiros da Internet podem ser identificados de muitas maneiras. Um identificador é seu nome de hospedeiro (hostname), fáceis de lembrar, porém eles fornecem pouca, se é que alguma, informação sobre a localização de um hospedeiro na Internet.

# DNS: o serviço de diretório da Internet

- Além disso, como nomes de hospedeiros podem consistir em caracteres alfanuméricos de comprimento variável, seriam difíceis de serem processados por roteadores. Hospedeiros também são identificados pelos endereços de IP(constituído de 4 bytes), que possui uma estrutura hierárquica(ao examiná-lo da esquerda para a direita, obtém-se informações específicas sobre onde o hospedeiro se encontra).

# DNS: Domain Name System (1)

Pessoas:

Possuem muitos identificadores: CPF, nome, no. de Passaporte, RG, etc...

Dispositivos na Internet:

- Dispositivos Internet (hosts, roteadores, etc...) usam **números.**
- Endereço IP (**32 bits**): usado para endereçar datagramas.
- "Nome" : usado por humanos.
- www.unesp.br = 200.145.9.9

Como mapear entre nome e endereço IP?

UEMG FRUTAL

# DNS: Domain Name System (2)

- (1) Uma base de dados distribuída implementada em uma hierarquia de muitos servidores de nomes (***nameservers***).
- (2) Um protocolo da camada de aplicação que permite que os hosts e os servidores de nomes se comuniquem, de modo a fornecer o **serviço de tradução** → "resolver".

**Resolver nome = traduzir nome em endereço IP.**

Os servidores de nomes (*nameservers*) são freqüentemente máquinas de Unix que rodam o software *Berkeley Internet Name Domain* (***Bind***).
O protocolo do DNS funciona sobre UDP e usa a porta 53.

- Para que a máquina do usuário possa enviar uma mensagem de requisição HTTP ao servidor WEB, ela precisa primeiramente obter o endereço de IP e isso é feito da seguinte maneira:
  - A própria máquina do usuário executa o lado cliente da aplicação DNS
  - O browser extra o nome de hospedeiro do URL e passa o nome para o lado cliente da aplicação DNS.
  - O cliente DNS envia uma consulta contendo o nome do hospedeiro para um servidor DNS.
  - O cliente DNS finalmente recebe uma resposta, que inclui o endereço IP para o nome do hospedeiro.
  - Tão logo o browser receba o endereço do DNS, pode abrir uma conexão TCP com o processo servidor http localizado naquele endereço IP.

# Servidores de nomes DNS

Por quê não centralizar o DNS?

- Ponto único de falha.
- Volume de tráfego.
- Base de dados centralizada seria distante.
- Dificuldade de Manutenção da BD.

Não é escalável!

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

Servidor de nomes local:

- Cada provedor, empresa ou instituição tem ***servidor de nomes local (default)***
- Pedido DNS de um host vai primeiro ao servidor de nomes local.

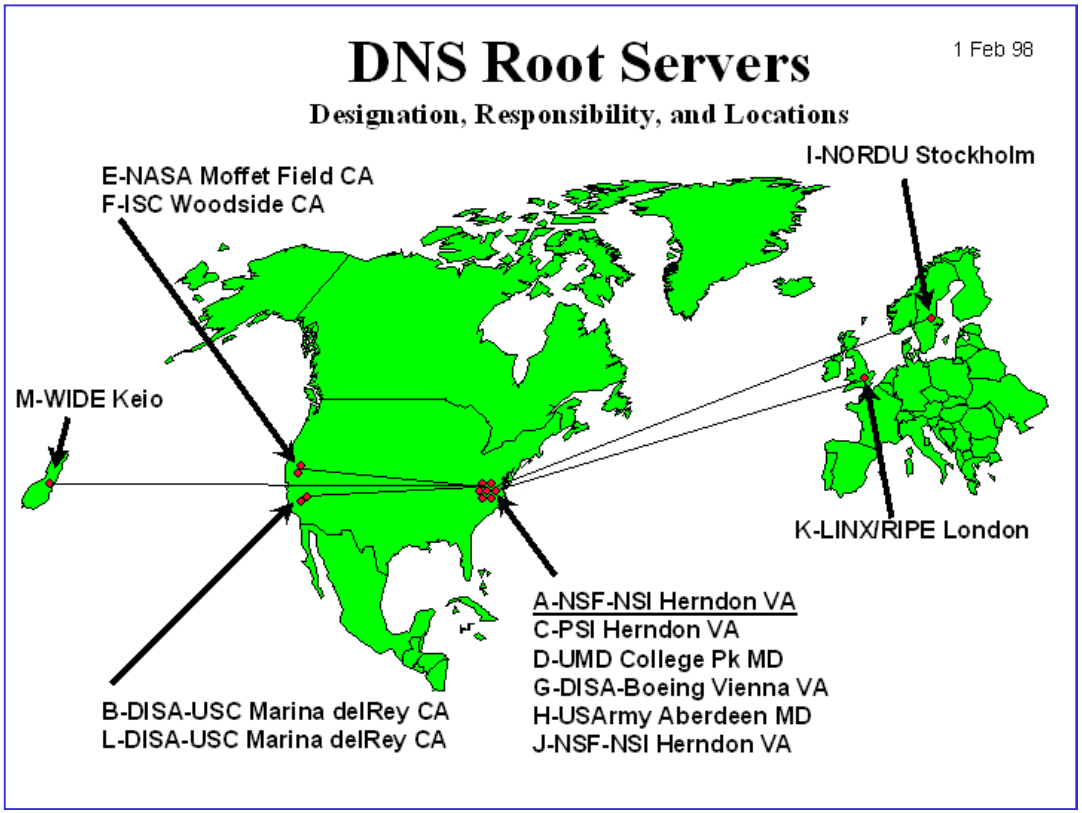
Servidor de nomes autoritativo:

- Para o host: guarda nome, endereço IP dele.
- Pode realizar tradução nome/endereço para este nome.

UEMG FRUTAL

# DNS: Servidores raiz (root servers)

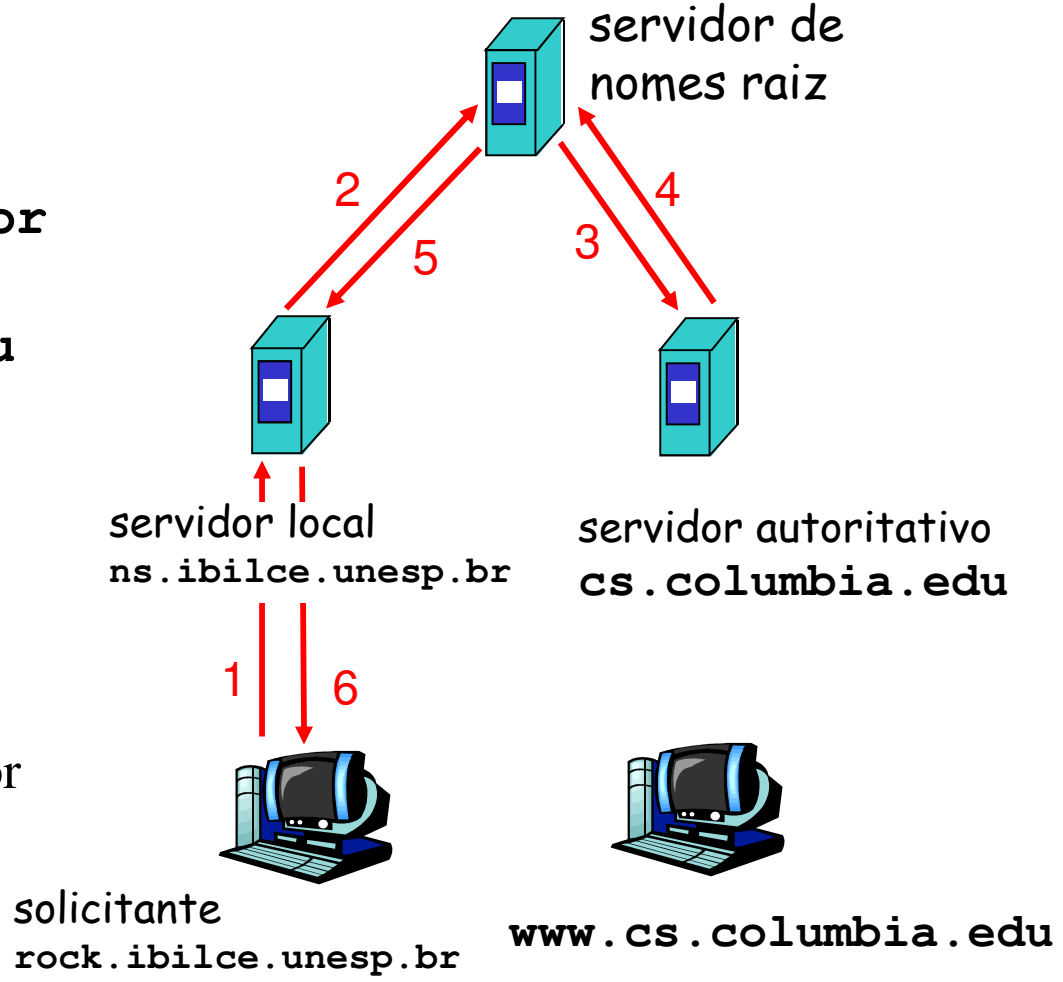
- Procurado por servidor local que não consegue resolver o nome.
- Servidor raíz:
  - Procura servidor autoritativo se mapeamento for desconhecido.
  - Obtém tradução.
  - Devolve mapeamento ao servidor local.
- Existem cerca de uma dúzia de servidores raíz no mundo.



# Exemplo simples do DNS

Host
**`rock.ibilce.unesp.br`**
requer endereço IP de
**`www.cs.columbia.edu`**

1. Contata servidor DNS local,
**`ns.ibilce.unesp.br`**
2. **`ns.ibilce.unesp.br`**
contata servidor raiz, se
necessário.
3. Servidor raiz contata servidor
autoritativo.
**`cs.columbia.edu`**, se
necessário



# Serviços importantes do DNS:

- Pedidos de hospedeiro: Um hospedeiro com nome complicado pode ter um ou mais apelidos. Um nome como ralayl.west-coast.enterprise.com (nome canônico) pode ter, por exemplo, enterprise.com como apelido. O DNS pode ser chamado por uma aplicação para obter o nome canônico correspondente a um apelido fornecido.

# Serviços importantes do DNS:

- Apelidos de servidor de correio: Endereços de e-mail são fáceis de lembrar, por exemplo bob@hotmail.com, porém o nome de hospedeiro do hospedeiro do servidor do Hotmail é mais complicado do que hotmail.com, então o DNS pode ser chamado por uma aplicação de correio para obter o nome canônico a partir do apelido fornecido.

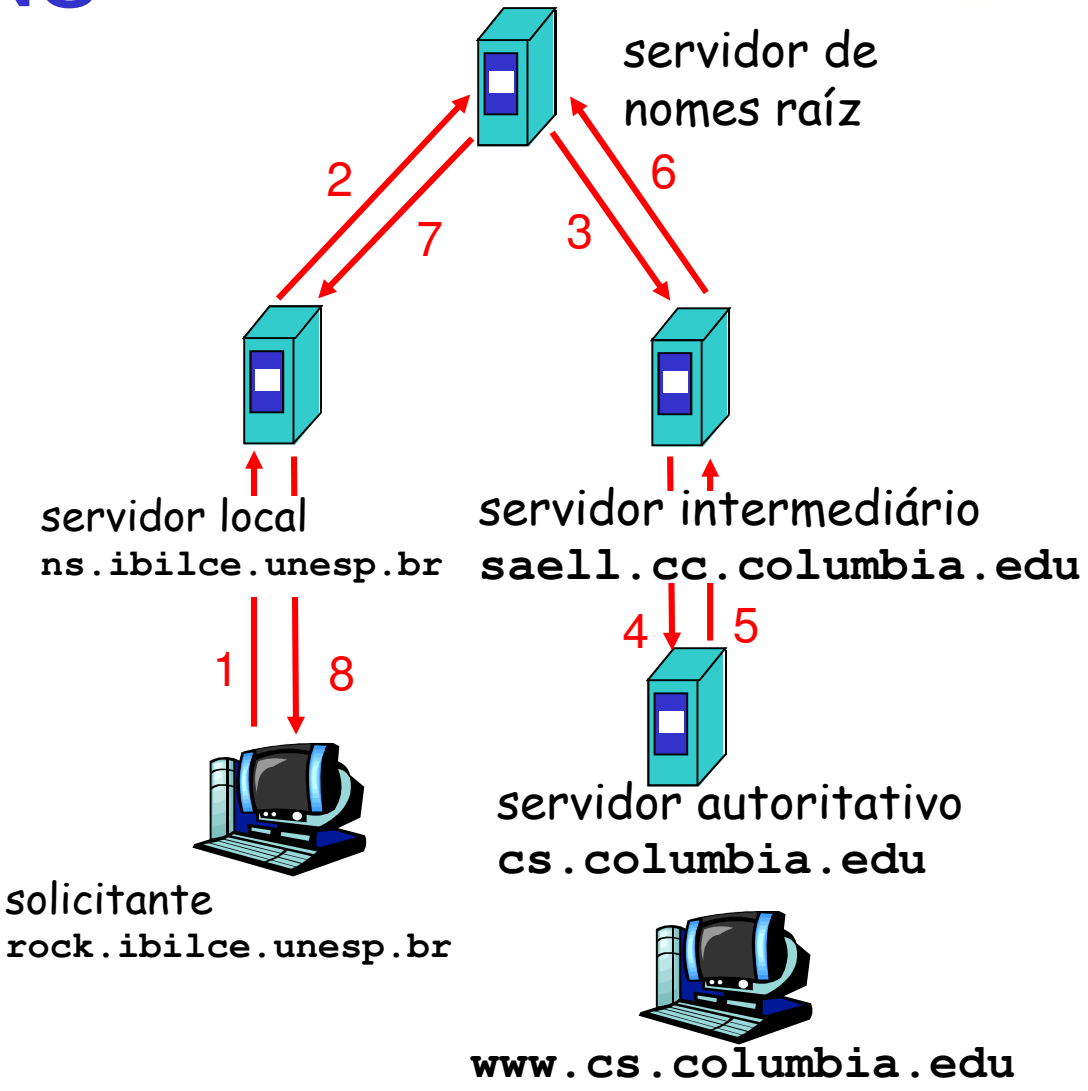
# Serviços importantes do DNS:

- Distribuição de carga: O DNS também é usado para realizar distribuição de carga entre servidores replicados, tais como os servidores Web replicados. Sites movimentados são replicados em vários servidores, sendo que cada servidor roda em um sistema final diferente e tem um endereço IP diferente. Assim, um conjunto de endereços IP fica associado a um único nome canônico e contido no banco de dados do DNS.

# Exemplo de DNS

Servidor raíz:

- Pode não conhecer o servidor de nomes autoritativo.
- Pode conhecer *servidor de nomes intermediário:* a quem contactar para descobrir o servidor de nomes autoritativo.



## Consulta Iterativa

- O hospedeiro envia uma mensagem de consulta DNS a seu servidor de nomes local(dns.poly.edu). Essa mensagem contém o nome de hospedeiro a ser traduzido. O servidor de nomes local transmite a mensagem de consulta a um servidor de nomes raiz, que percebe o sufixo edu e retorna ao servidor de nomes local uma lista de endereços IP contendo servidores TLD responsáveis por edu.

## Consulta Iterativa

- Então, o servidor de nomes local retransmite a mensagem de consulta a um desses servidores TLD, que recebe o sufixo umass.edu e responde com o endereço IP do servidor de nomes com autoridade para a University of Massachusetts (dns.umass.edu). Finalmente, o servidor de nomes local reenvia a mensagem de consulta diretamente a dns.umass.edu que responde com o endereço IP de gaia.cs.umass.edu

## Consulta Recursiva:

- O hospedeiro requisita o endereço IP, este envia uma mensagem de consulta para o servidor de nomes loca, que envia ao servidor de nomes raiz, que envia ao de nomes TLD, que envia ao de nomes com autoridade, que retorna o endereço IP para o TLD, que retorna ao de nomes raiz, que retorna ao de nomes local.

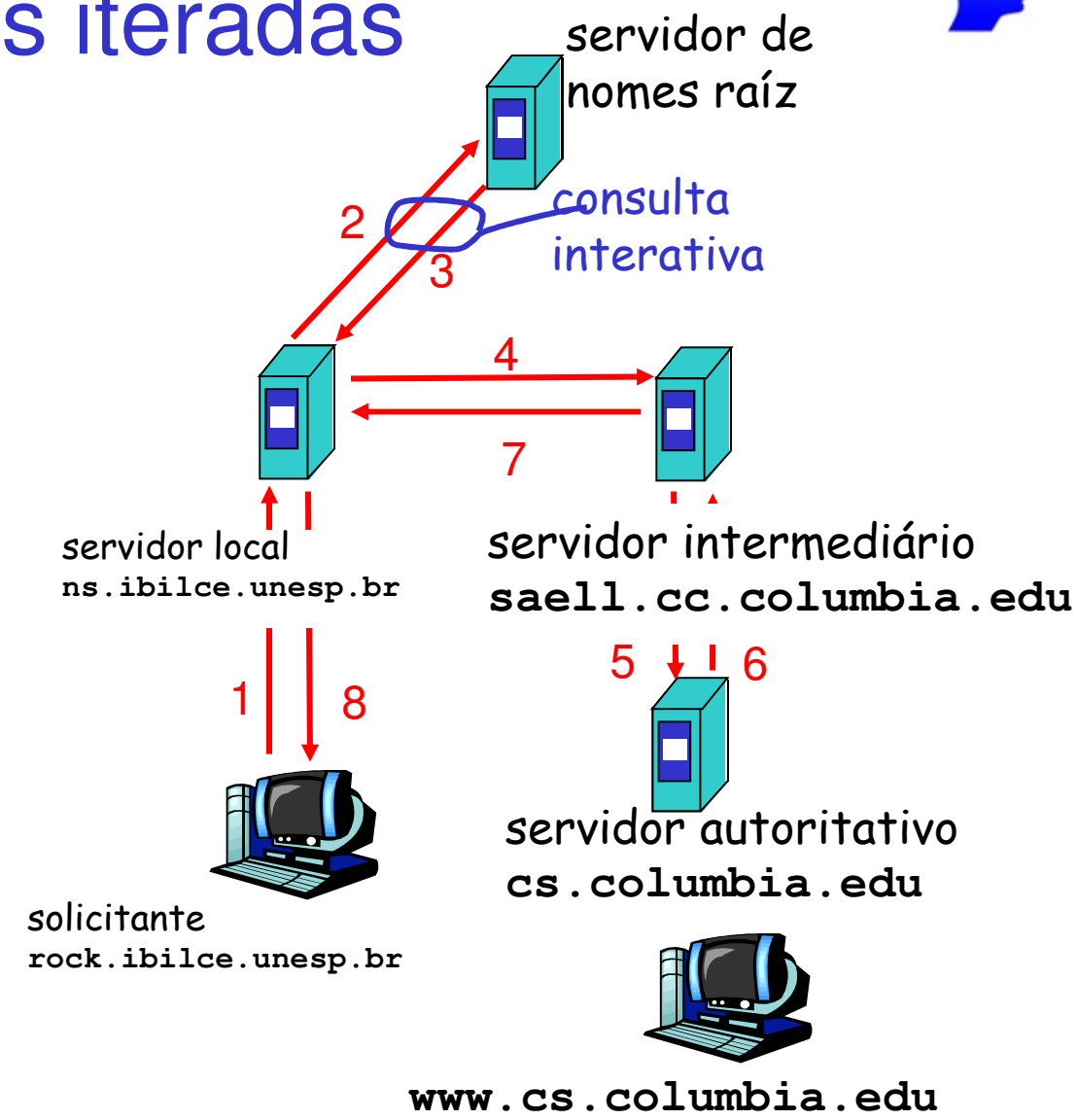
# DNS: consultas iteradas

Consulta recursiva:

- Transfere a responsabilidade de resolução do nome para o servidor de nomes contatado.
- Carga pesada?

Consulta iterada:

- Servidor consultado responde com o nome de um servidor de contato.
- **"Não conheço este nome, mas pergunte para esse servidor"**



## Cache DNS:

- O DNS explora extensivamente o cache para melhorar o desempenho quanto ao atraso e reduzir o numero de mensagem DNS que ricocheteia pela Internet. A ideia por trás do cache é muito simples. Em uma cadeia de consultas, quando um servidor de nomes recebe uma resposta DNS, ele pode fazer cache das informações da resposta em sua memória local.

# DNS: uso de cache, atualização de dados

- Uma vez um servidor qualquer aprende um mapeamento, ele o coloca numa *cache* local.
  - Futuras consultas são resolvidas usando dados da *cache*.
  - Entradas no cache são sujeitas a temporização (desaparecem depois de certo tempo)
    ttl = ***time to live*** (sobrevida).

## Mensagens DNS:

- As duas únicas espécies de mensagens DNS são mensagens de consulta e de resposta DNS. A semântica de vários campos de uma mensagem é a seguinte:

## Mensagens DNS:

- 1. Os primeiros 12 bytes formam a seção de cabeçalho que contém vários campos: O primeiro campo é o identificador da consulta. Campos de flag (flag para dizer se a mensagem é de consulta ou resposta, flag de autoridade que é marcado em uma mensagem de resposta quando o servidor de nomes é um servidor com autoridade para um nome consultado, flag de recursão, quando um cliente quer que um servidor de nomes proceda recursivamente sempre que não tem registro). Há também quatro campos de ‘numero de’.

## Mensagens DNS:

- 2. A seção pergunta contém informações sobre a consulta que está sendo feita. Inclui um campo de nome que contém o nome que está sendo consultado e um campo de tipo que indica o tipo de pergunta que está sendo feito sobre o nome.

## Mensagens DNS:

- 3. Em uma resposta de um servidor de nomes, a seção resposta contém os registros de recursos par ao nome que foi consultado originalmente.

- 4. A seção de autoridade contém registros de outros servidores com autoridade.

- 5. A seção adicional contém outros registros úteis.

# Registros DNS

DNS: BD distribuída contendo *resource records* (RR)

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

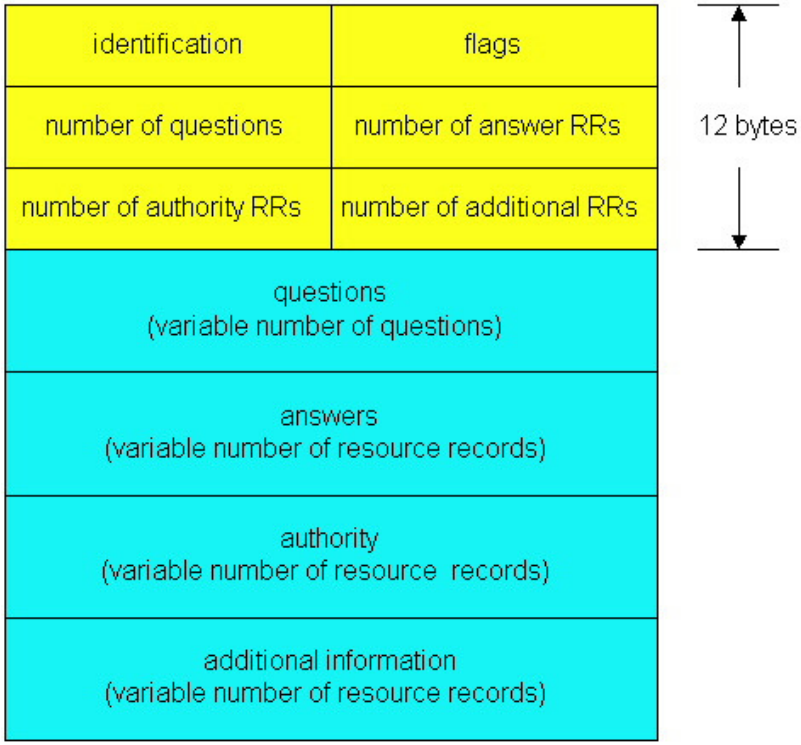
- Tipo=A
  - **nome** é nome de hospedeiro
  - **valor** é endereço IP
- Tipo=NS
  - **nome** é domínio (p.ex. foo.com.br)
  - **valor** é endereço IP de servidor de nomes autoritativo para este domínio
- Tipo=CNAME
  - **nome** é nome alternativo (alias) para algum nome "canônico" (verdadeiro)
  - **valor**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

# DNS: protocolo, mensagens

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

Cabeçalho de mensagem

- *identification*: ID de 16 bit para pedido, resposta ao pedido usa mesmo ID
- *flags*:
  - pedido ou resposta
  - recursão desejada
  - recursão permitida
  - resposta é autoritativa



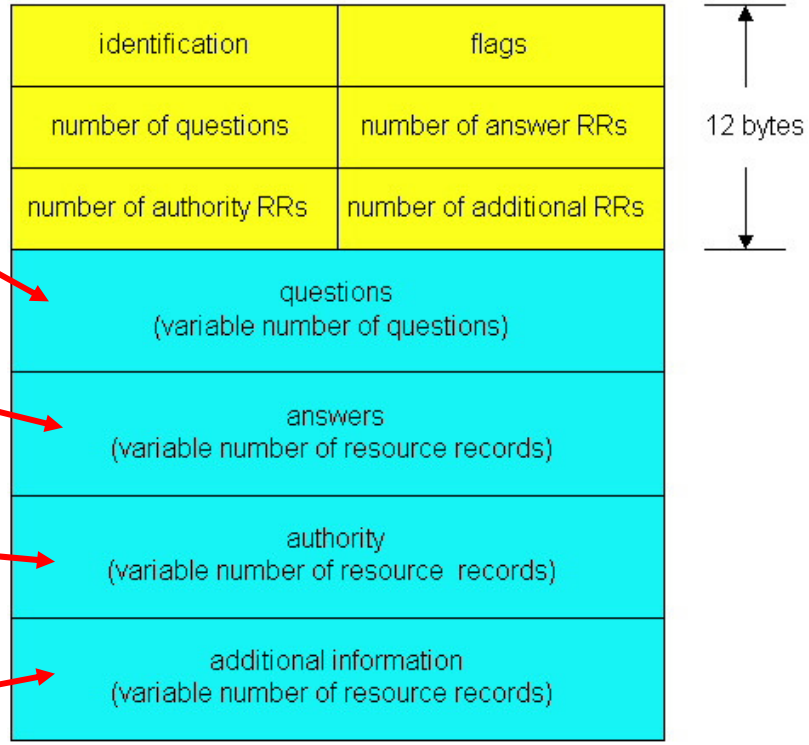
# DNS: protocolo, mensagens

campos nome, tipo num pedido

RRs em resposta ao pedido

registros para servidores autoritativos

info adicional "relevante" que pode ser usada



# Compartilhamento de arquivos

Aplicações P2P

UEMG
FRUTAL

# Compartilhamento de arquivos P2P

Exemplo:

- Alice executa a aplicação cliente P2P em seu notebook.
- Intermitentemente → conecta-se à Internet;
  - Obtém novo endereço IP para cada conexão.
- Procura por um determinado arquivo de uma música.
- A aplicação exibe outros ***peers*** que possuem uma cópia da música.
- Alice escolhe um dos pares: Bob.
- O arquivo é copiado do PC de Bob para o notebook de Alice:
  - por exemplo via HTTP.
- Enquanto Alice faz o download: outros usuários fazem upload da máquina de Alice.
  - O ***peer*** "Alice" é tanto um cliente Web como um servidor Web transiente.

**Todos os pares são servidores = altamente escaláveis!**

# Modelo Cliente / Servidor

- Modelo mais usado na Internet.
  - Dependente de servidores bem configurados em com informação acessível.
- Não explora o potencial de **computação distribuída** proveniente da Rede.
  - A existência de um ou milhares de computadores é indiferente na interação de um usuário típico com a rede.
- PCs clientes com capacidade considerável ficam "escondidos", formando um exército com alto potencial.

# Definição de sistema P2P (1)

**Compartilhamento** de recursos e serviços computacionais **diretamente** entre sistemas.

# Definição de sistema P2P (1)

- Classe de aplicações que leva vantagem de recursos disponíveis nas **bordas** da rede.
- Quais recursos?
  - Armazenamento.
  - Tempo de CPU.
  - *Bandwidth*.
  - Conteúdo.
  - Presença humana.

UEMG FRUTAL

# Questões Fundamentais:

**Colaboração**

**Cooperação**

**Compartilhamento**

# Características P2P (1)

- Sistemas **distribuídos** sem controle centralizado ou organização hierárquica.
- Software executado em cada *peer* é equivalente em funcionalidade.
- Quantidade explosiva e exponencial de adeptos.

# Características P2P (2)

- Cada participante **age como cliente e servidor ao mesmo tempo** (***servent***).
- Cada cliente "paga" a sua participação fornecendo acesso a (alguns de) seus recursos.
  - *Peering*.

# Características P2P (3)

- Sem coordenação central.
- Sem banco de dados central.
- Sem local único de falha ou gargalo.
- Nenhum ponto (*peer*) tem visão global do sistema.
- Comportamento global definido por interações locais.
- Todos os dados e serviços são acessíveis de qualquer ponto.
- Pontos são autônomos.
- Pontos e conexões não são confiáveis.

# Principais vantagens

- Escalabilidade
  - Não há gargalo para crescimento.
- Robustez
  - Não há ponto de falha único.
- Flexibilidade
  - Auto-configuração / configuração dinâmica.

# Comparação entre Roteadores e sistemas P2P

- **Descobrem** e mantêm topologia.
- Não são clientes nem servidores.
- Continuamente **falam uns com os outros**.
- São inerentemente tolerantes a falhas.
- São autônomos.

# Teste P2P

- O sistema aceita conectividade variável e endereços IP temporários?
- O sistema dá uma autonomia significativa aos computadores na borda da rede?
- Os nós podem trocar informações livremente entre si, sem arbitragem central?

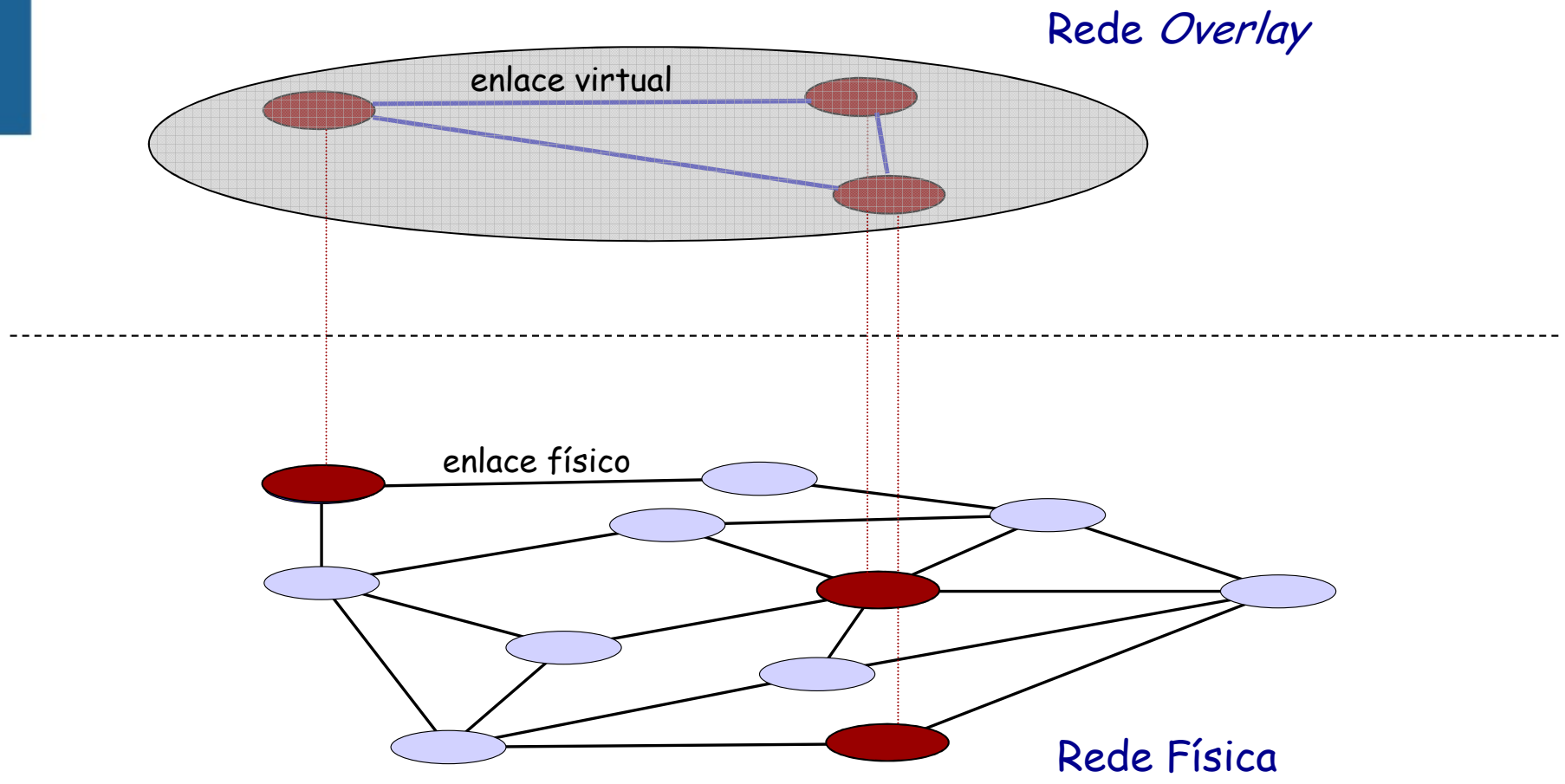
# Sistemas P2P: Requisitos

- Descoberta de recursos/serviços.
  - Baseada em: nome, endereço, rota, métrica, etc...
- Roteamento
  - Roteamento de aplicação: conteúdo, interesse, etc...
  - Roteamento entre super-nós: Kazaa, Morpheus,...
  - Roteamento baseado em capacidade (bandwidth)
- Robustez e tolerância a falhas (nó e enlace).
- Armazenamento distribuído e atualizações.
- Escalabilidade
- Confiança nos pares (autenticação, autorização)
- Monitoramento de vizinhos.

# P2P X Redes de Cobertura (*Overlay*)

- ***Overlay***
  - Rede virtual: rede sobre outra rede (IP).
  - Os enlaces são conexões entre nós da rede.
- P2P freqüentemente utilizada para criar *overlays*.
  - Oferecendo serviços que poderiam ser implementados na camada IP.
  - Estratégia muito útil para implantação.
- Em certos casos, pode contornar barreiras econômicas.
  - Exemplo: IP era um *overlay* (sobre a rede de telefonia)
- Nem todos os overlays são P2P (AKAMAI).

# Redes *Overlay*



# Modelos de Sistemas P2P (Classificação 1)

- **Modelo Centralizado**
  - Índice global mantido por um autoridade central.
  - Contato direto entre clientes e provedores.
    - Exemplo: Napster.
- **Modelo Descentralizado**
  - Sem índice global (sem coordenação global)
    - Exemplos: Gnutella, Freenet.
- **Modelo Hierárquico**
  - Introdução dos super-nós (super-nodes ou super-peers).
  - Mistura dos modelos centralizado e descentralizado
    - Exemplos: KaZaA, Morpheus.

# Modelos de Sistemas P2P (Classificação 2)

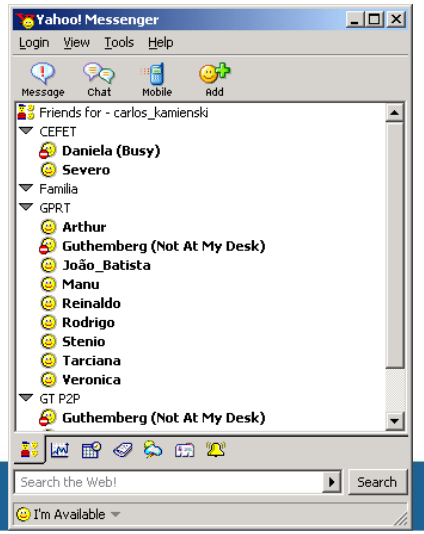
- ***Centralized Service Location*** **(CSL)**
  - Busca centralizada
  - Exemplo: Napster
- ***Flooding-based Service Location*** **(FSL)**
  - Busca baseada em inundação
  - Exemplo: Gnutella
- *Distributed Hash Table-based Service Location* (DHT)
  - Busca baseada em tabela de *hash* distribuída
  - Exemplos: Azureus, CAN, Pastry, Tapestry, Chord.

# Modelos de Sistemas P2P (Classificação 3)

- *Modelo Centralizado*
  - Napster, intant mesengers (ICQ, MSN, etc...)
- Modelo Descentralizado e Estruturado
  - DHT.
  - Bittorrent.
- Modelo Descentralizado e Não Estruturado.
  - Super-Nós: KaZaA
  - Inundação: Gnutella

# Aplicação: Troca de Mensagens

- IM (*Instant Messaging*)
- Aplicação popular na Internet, pela facilidade de enviar mensagens on-line.
  - Exemplos:
    - MSN Messenger (http://messenger.msn.com)
    - Yahoo! Messenger (http://messenger.yahoo.com)
    - ICQ – (http://web.icq.com)
    - Jabber – (http://ww.jabber.org)



# Aplicação: Compartilhamento de Arquivos

- Aplicação de maior sucesso na Internet.
  - Permite usuários compartilharem diretamente seus arquivos, músicas, vídeos, etc...
  - Pode apresentar problemas de **violação de direitos**.
- Características
  - Área de armazenamento.
  - Disponibilidade de informações.
  - Anonimato.
  - Gerenciamento.

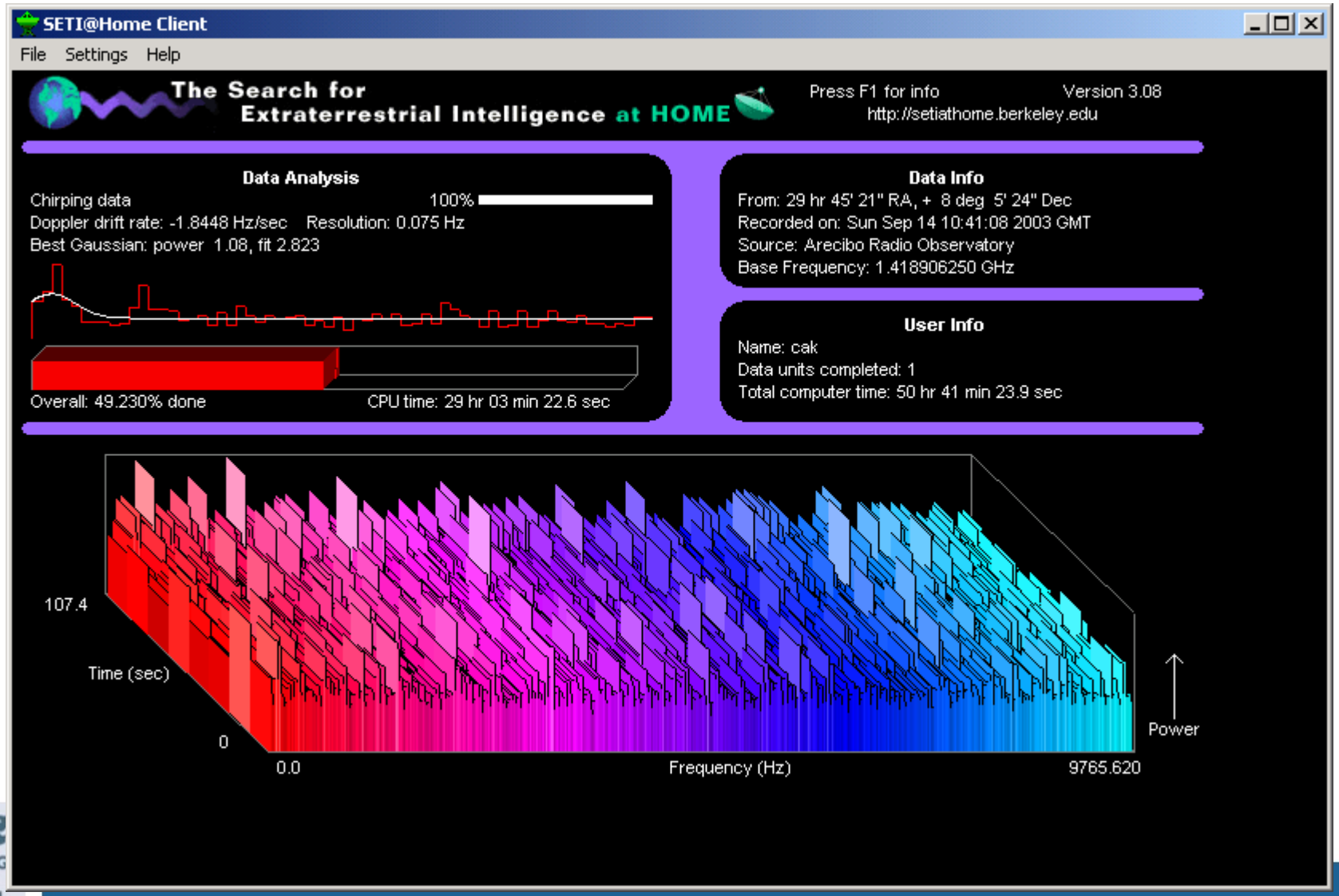
# Compartilhamento: Aplicações

- Napster (http://www.napster.com)
- KaZaA (http://www.kazaa.com)
- Gnutella (http://www.gnutella.com)
  - BearShare (http://www.bearshare.com)
  - LimeWire (http://www.limewire.com)
- Freenet (http://www.freenetproject.org)
- Imesh (http://www.imesh.com)
- Morpheus (http://www.morpheus.com)
- Grokster (http:// www.grokster.com)
- Bittorent (http://www.bittorent.com)

# Aplicações: Computação Distribuída

- Idéia de aproveitar recursos computacionais ociosos não é nova.
- Grade Computacional (Grid)
  - Solução de computação distribuída para engenharia e ciências, baseada em compartilhamento de recursos em larga escala.
  - Semelhanças e diferenças com P2P.
- SETI@Home (http://setiathome.ssl.berkeley.edu)
  - *The Search for Extraterrestrial Intelligence*
  - Usuários executam partes da "busca".

# SETI@Home


SETI@Home Client
File Settings Help
The Search for Extraterrestrial Intelligence at HOME
Press F1 for info
Version 3.08
http://setiathome.berkeley.edu
Data Analysis
Chirping data
100%
Doppler drift rate: -1.8448 Hz/sec Resolution: 0.075 Hz
Best Gaussian: power 1.08, fit 2.823
Overall: 49.230% done
CPU time: 29 hr 03 min 22.6 sec
Data Info
From: 29 hr 45' 21" RA, + 8 deg 5' 24" Dec
Recorded on: Sun Sep 14 10:41:08 2003 GMT
Source: Arecibo Radio Observatory
Base Frequency: 1.418906250 GHz
User Info
Name: cak
Data units completed: 1
Total computer time: 50 hr 41 min 23.9 sec
107.4
Time (sec)
0
0.0
Frequency (Hz)
9765.620
Power


# Questões de Projeto P2P

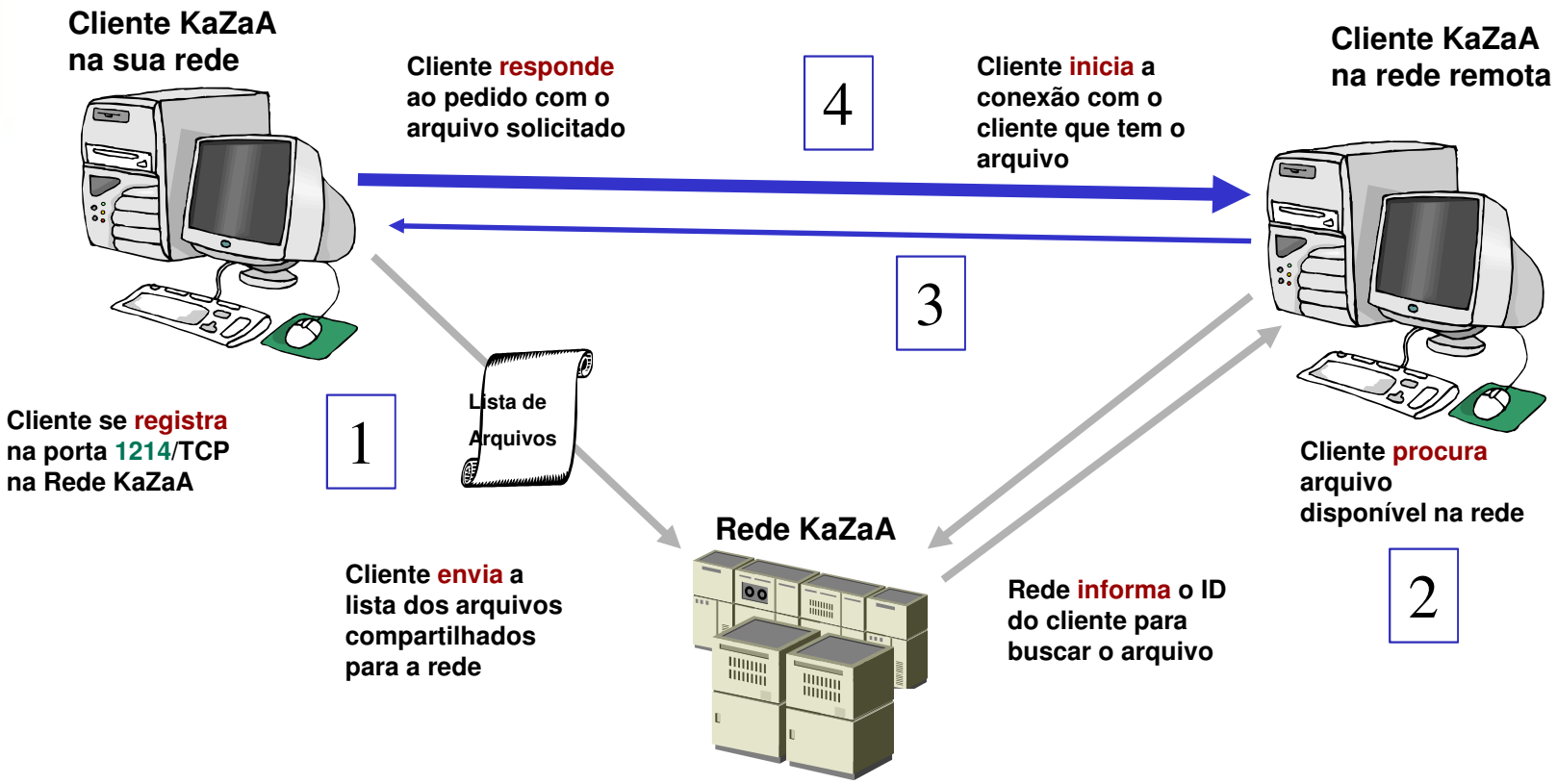
# Questão: endereçamento

- Comunicação P2P pura necessita de conexões diretas entre os *peers*.
- Barreiras de endereçamento/proteção impedem essa comunicação direta.
  - **DNS**: só traduz os endereços das máquinas que o administrador da rede quer revelar.
  - **Firewall**: bloqueia a comunicação de entrada/saída da rede, de acordo com critérios de segurança.
  - **NAT** (*Network Address Translation*): traduz endereços de rede interna (ex.: 10.0.1.1, 172.16.4.22, 192.168.0.4) em endereços públicos (ex.: 200.249.188.1, 150.161.2.1)
  - **Proxy**: interpõem-se na comunicação fim a fim (http) para filtrar páginas indesejáveis.

# Contornando o DNS

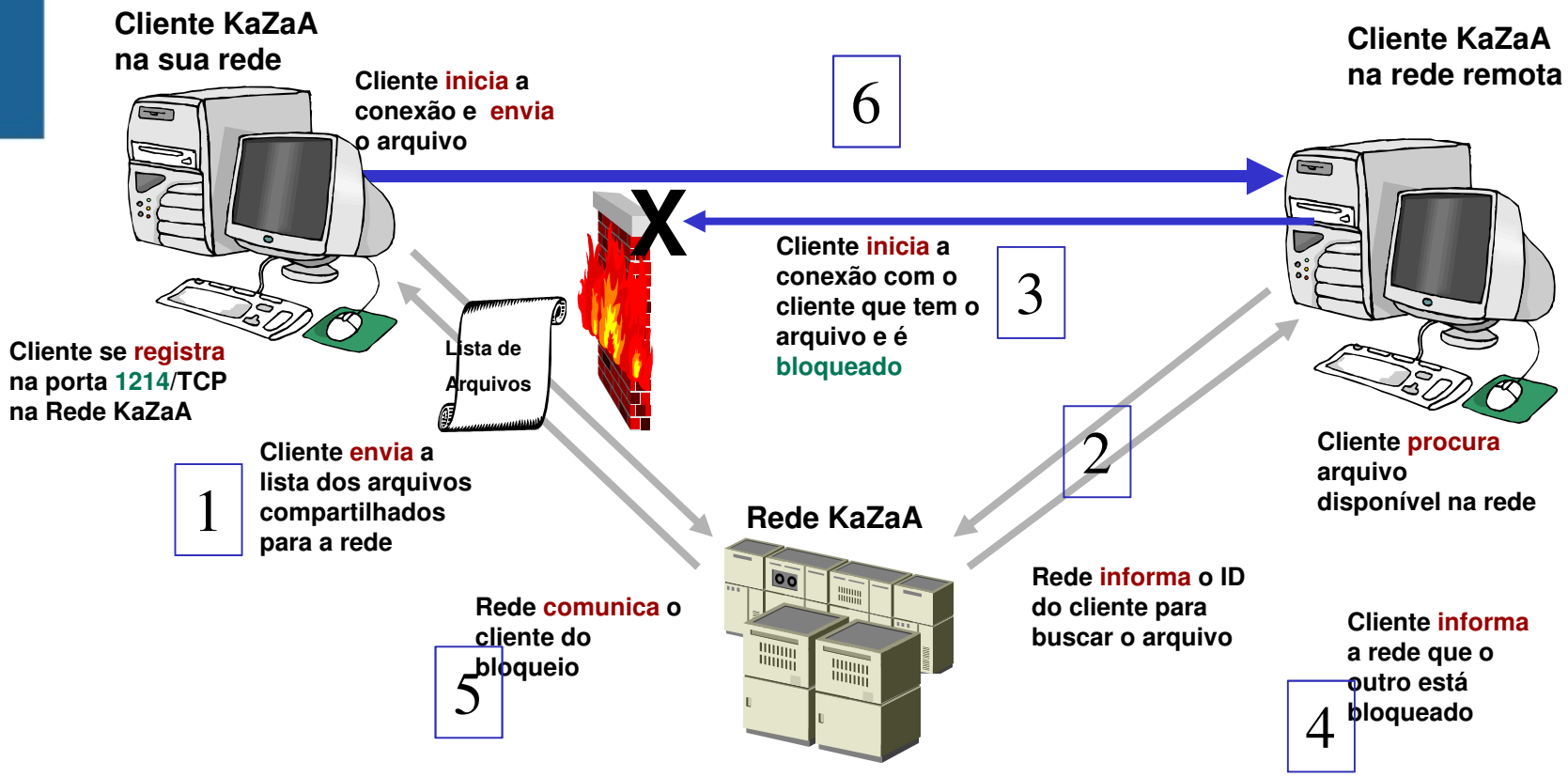
- **Cadastro próprio**
  - Napster, ICQ, Groove
- **Endereços IP de membros da rede.**
  - Gnutella, KaZaA, Overnet
- **Endereços IP de servidores fixos.**
  - SETI@Home.

# Contornando o Firewall: KaZaA (normal)




Operação normal do KaZaA versão 2

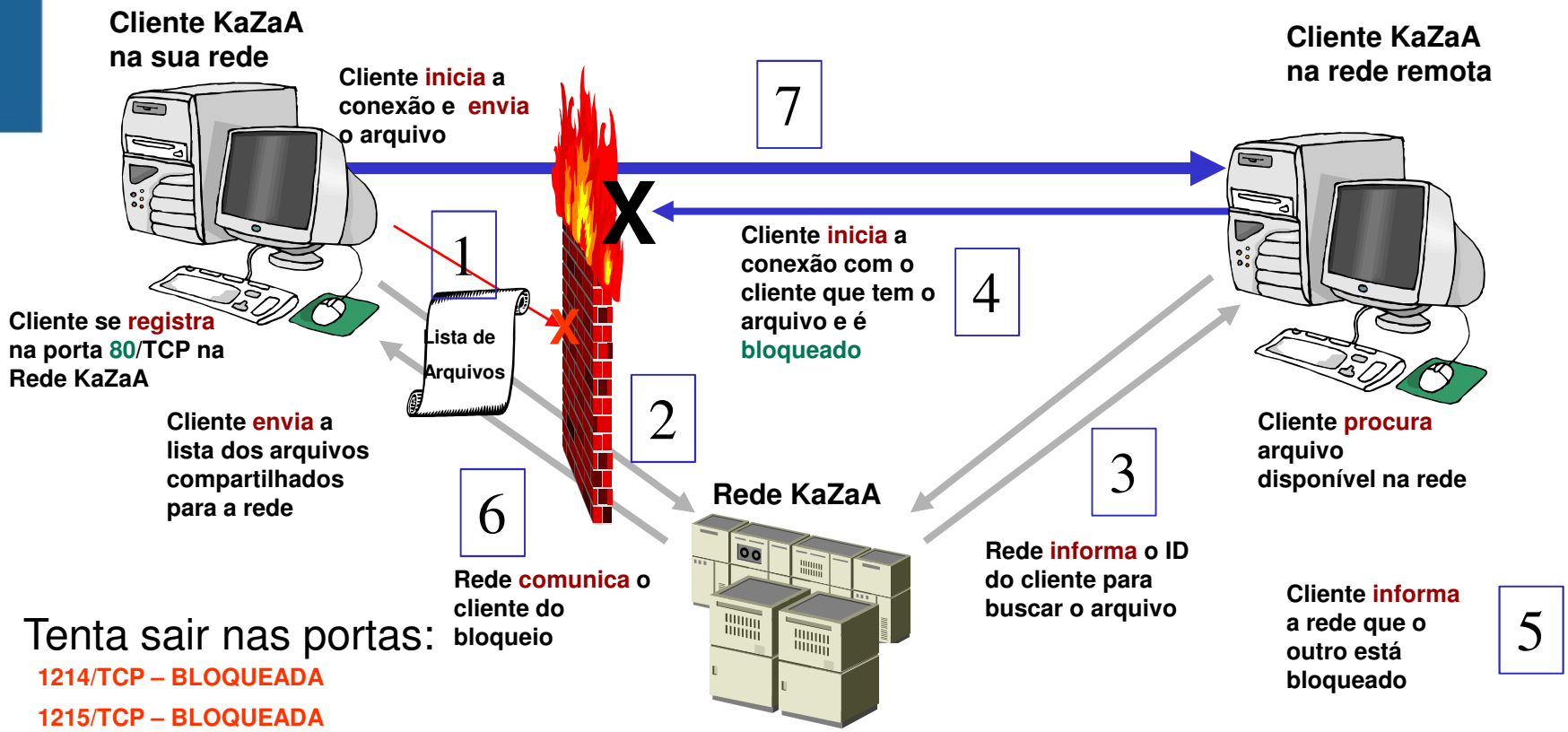
# Contornando o Firewall: KaZaA
## (bloqueio entrada)




Operação quando o KaZaA é bloqueado (porta 1214 na entrada)

# Contornando o Firewall: KaZaA

## (bloqueio entrada e saída)




Tenta sair nas portas:

- 1214/TCP – BLOQUEADA
- 1215/TCP – BLOQUEADA
- MUITAS OUTRAS/TCP – BLOQUEADA
- Tenta centenas de portas, incluindo: 80, 53, 1024, etc

Operação quando o KaZaA é bloqueado (porta 1214 entrada/saída)

UEMG FRUTAL

# Contornando o NAT (situação)

- NATs são projetados para o modelo cliente/servidor.
  - Clientes iniciam (*active open*) a conexão com servidores bem conectados com endereços estáveis e nomes DNS.
- Assimetria:
  - Máquinas **internas** podem iniciar conexão com máquinas externas.
  - Máquinas externas **NÃO podem iniciar conexão com máquinas internas** (mesmo sabendo endereço IP e porta).

# Contornando NAT (técnicas)

http://midcom-p2p.sourceforge.net/draft-ford-midcom-p2p-01.txt

- Intermediação (*relaying*).
- Conexão reversa
  - Uma máquina tem endereço IP válido.
- Perfuração de buracos UDP (*hole punching*)
  - Variação: predição no número de porta
- Abertura simultânea de conexão TCP
  - Requer sincronização precisa de pacotes SYN e predição do próximo número de porta
- Uso do protocolo Midcom

http://midcom-p2p.sourceforge.net/draft-ford-midcom-p2p-01.txt

# Questão: Conectividade

- Heterogeneidade de conexões dos *peers*
  - Tecnologia / capacidade (banda) / assimetria.
- **Muitos** *peers* em conexões de **baixa capacidade** e alta instabilidade.
- **Poucos** *peers* em conexões de alta capacidade e baixa instabilidade.
- Indícios de que as redes P2P apresentem características Small World
  - Distribuições de lei de potência (Power Law)

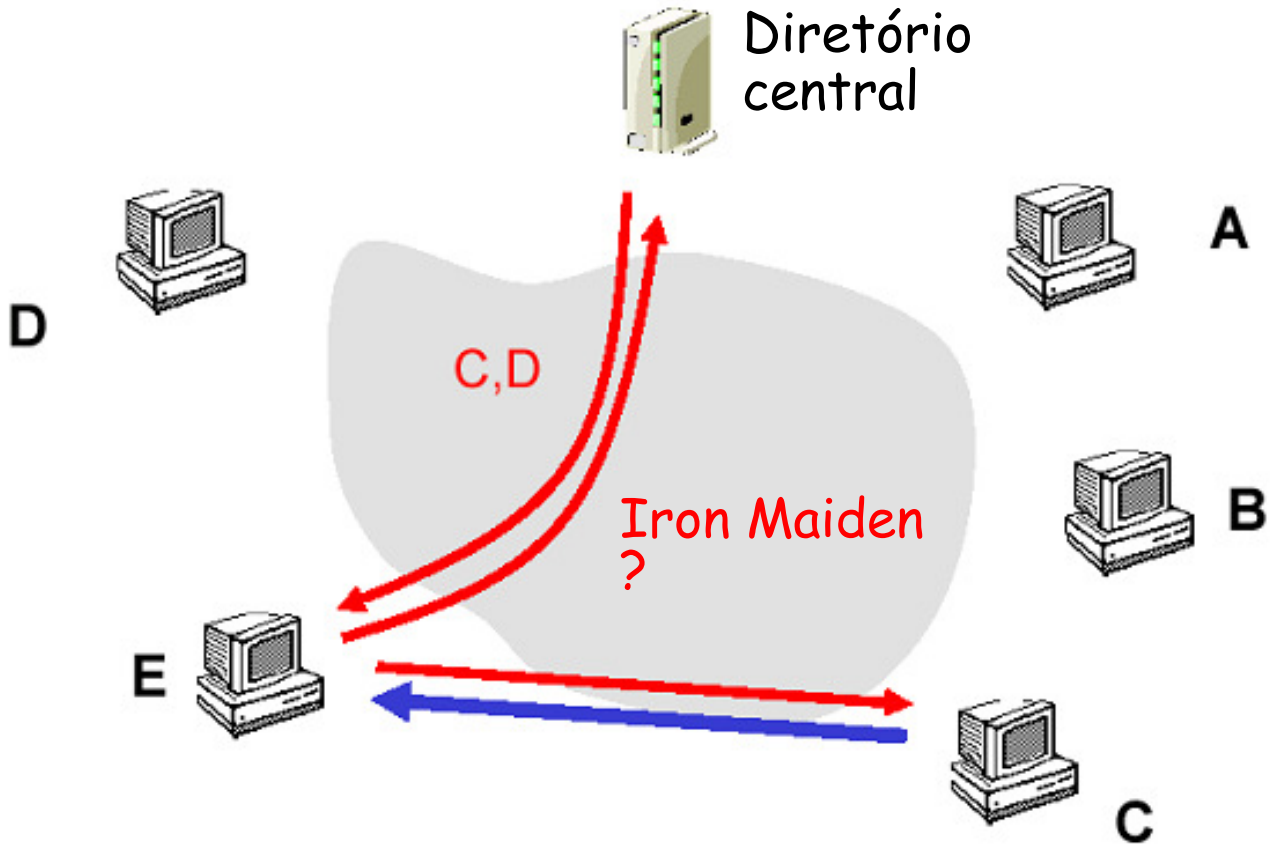
# Questão: Escalabilidade

- É **benefício imediato da descentralização**.
- Limitações da escalabilidade:
  - Quantidade de operações centralizadas.
  - Manutenção de estado (usuários/aplicações), etc...
- P2P é mais escalável que cliente/servidor.
  - Em um sistema P2P, o **número de servidores aumenta com o número de clientes**.
- Problemas de escalabilidade em P2P
  - Napster, Gnutella, Freenet
- Sistemas P2P estruturados (DHT) são escaláveis.

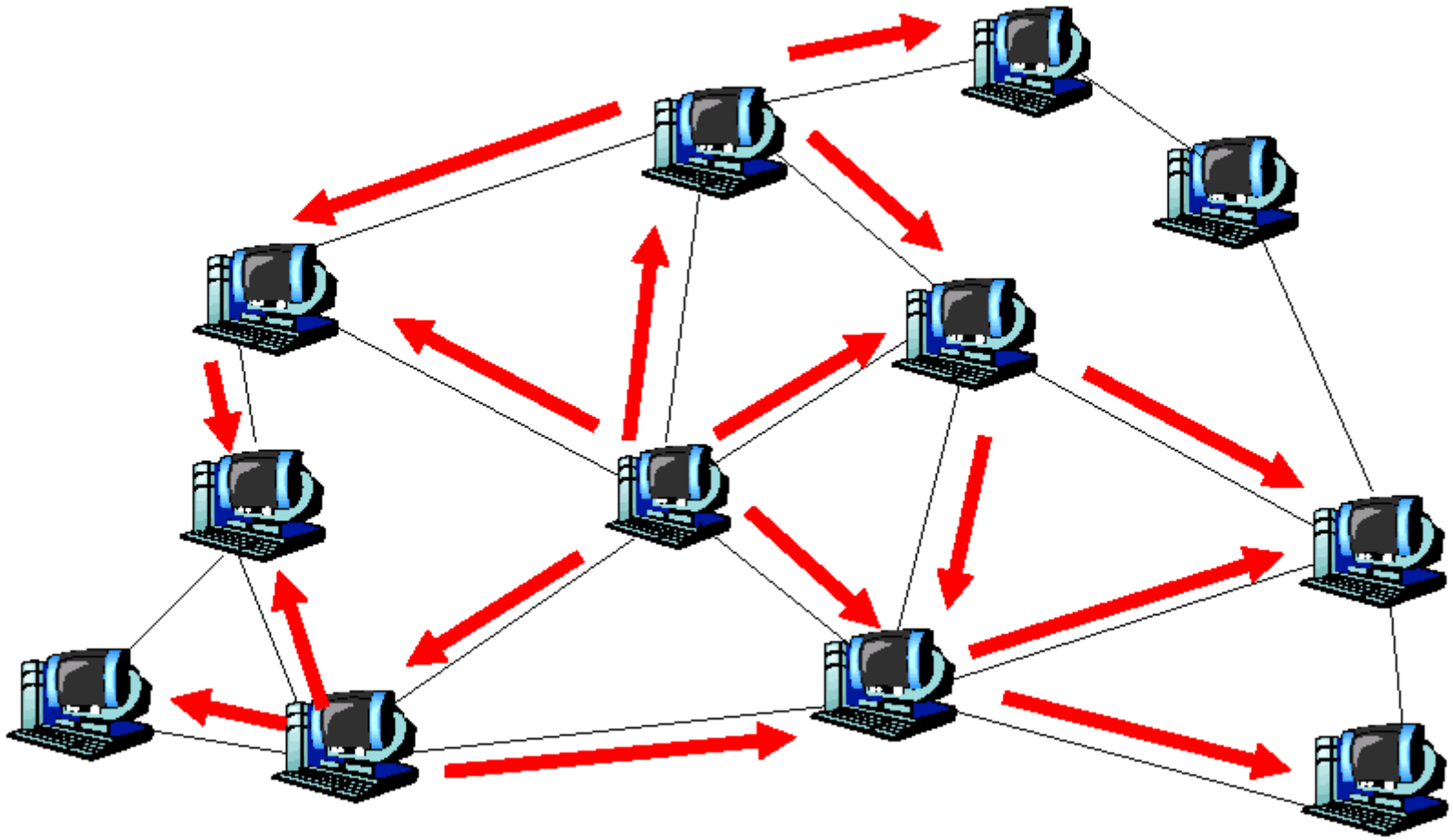
# Questão: Roteamento

- Localizar informação em rede grande, volátil e distribuída **não é simples.**
- **Localização** e busca de informação dependem do roteamento utilizado.
- Abordagens comuns:
  - **Modelo Centralizado**
  - **Modelo por Inundação**
  - **Modelo de Super-Nós**
  - **Modelo DHT**

# Modelo Centralizado

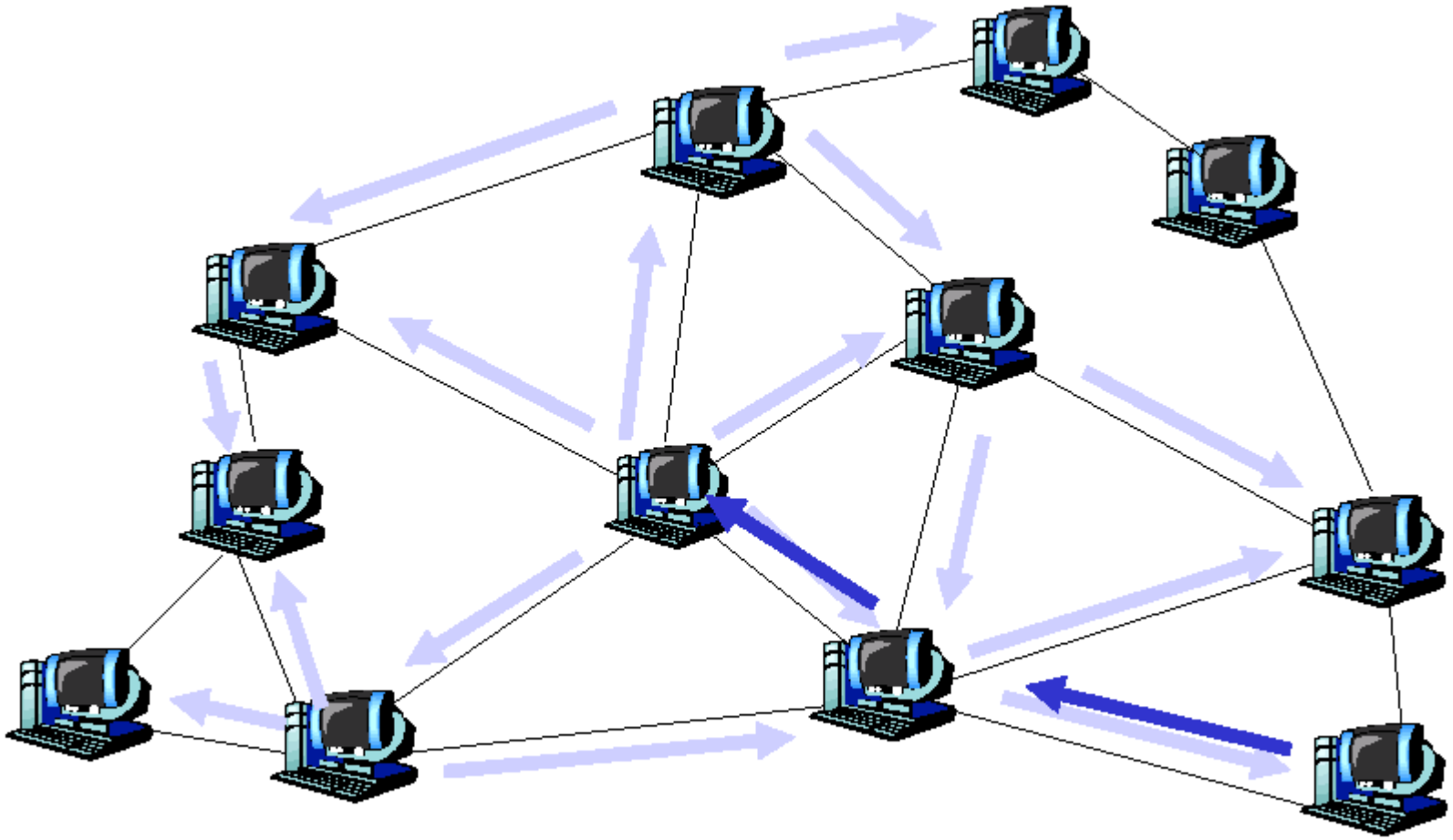


# Modelo por Inundação

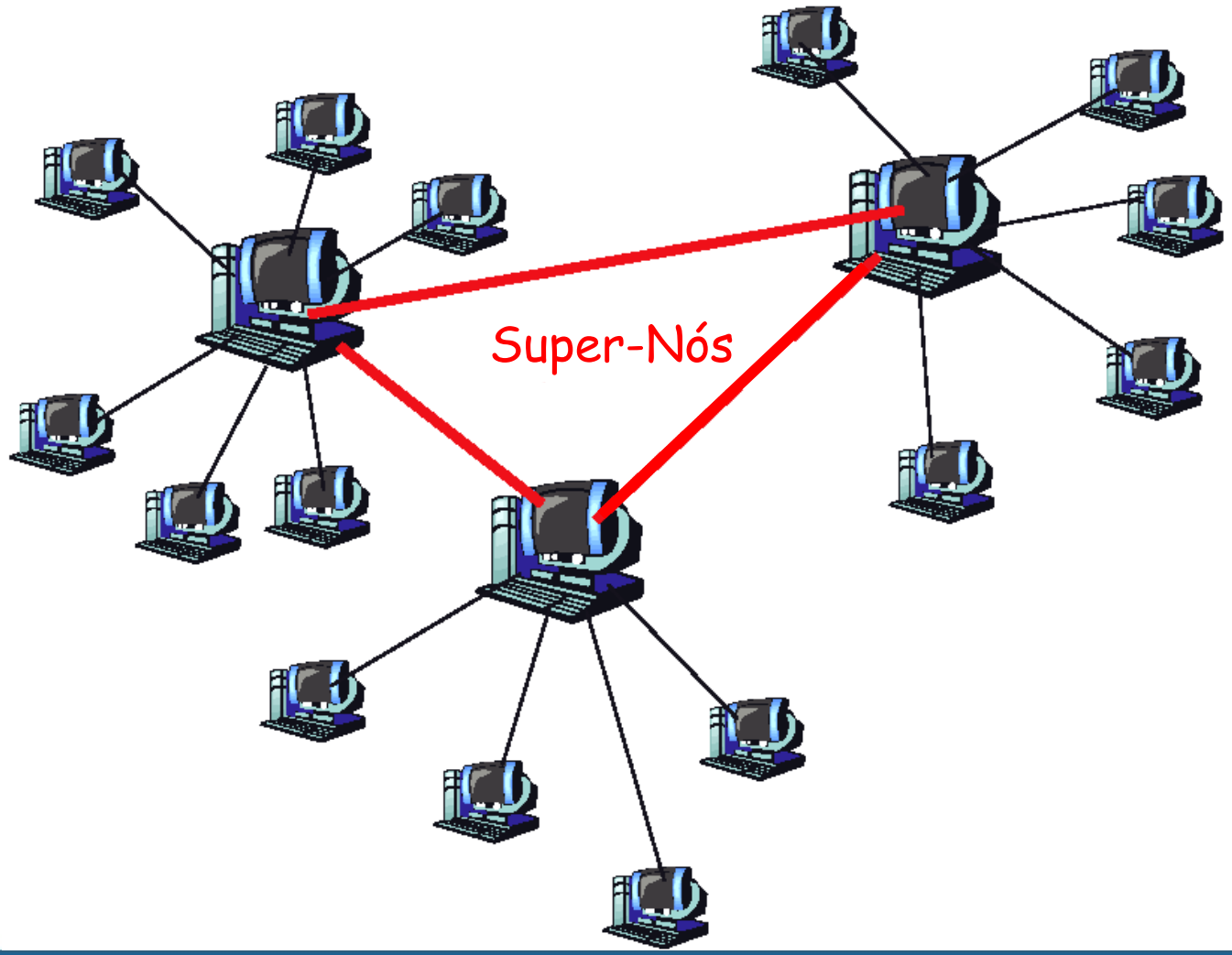


UEMG
FRUTAL

# Modelo por Inundação



# Modelo de Super-Nós

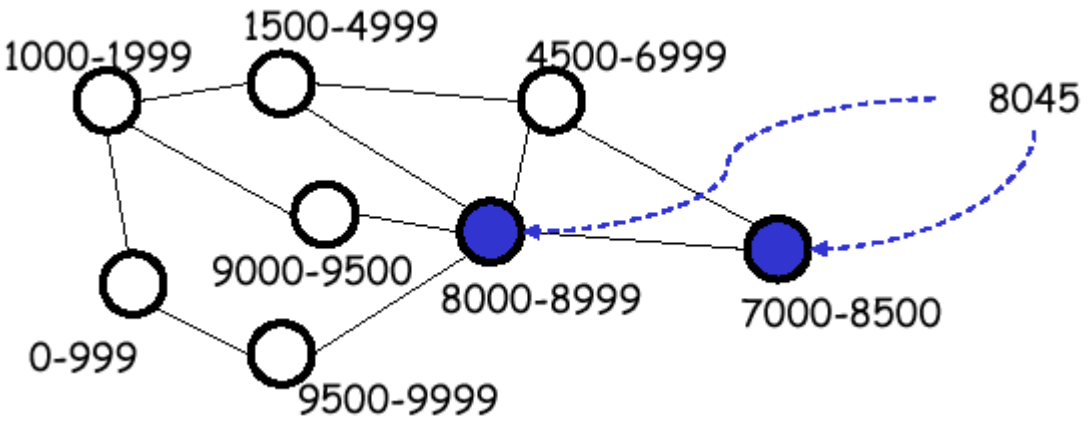


# Modelo DHT

- Hash
  - Estrutura de dados importantes para desenvolvimento.
- Hash distribuído na escala da Internet
- Importante para sistemas distribuídos grandes.
- Sistemas P2P.
- Espelhamento de servidores Web.

# DHT: Funcionamento

- Função de *hash* mapeia objeto para um **identificador único**.
  - Ex: hash("Aquarela do Brasil") → **8045**
- Faixa de resultados da função de hash é distribuída pela rede.



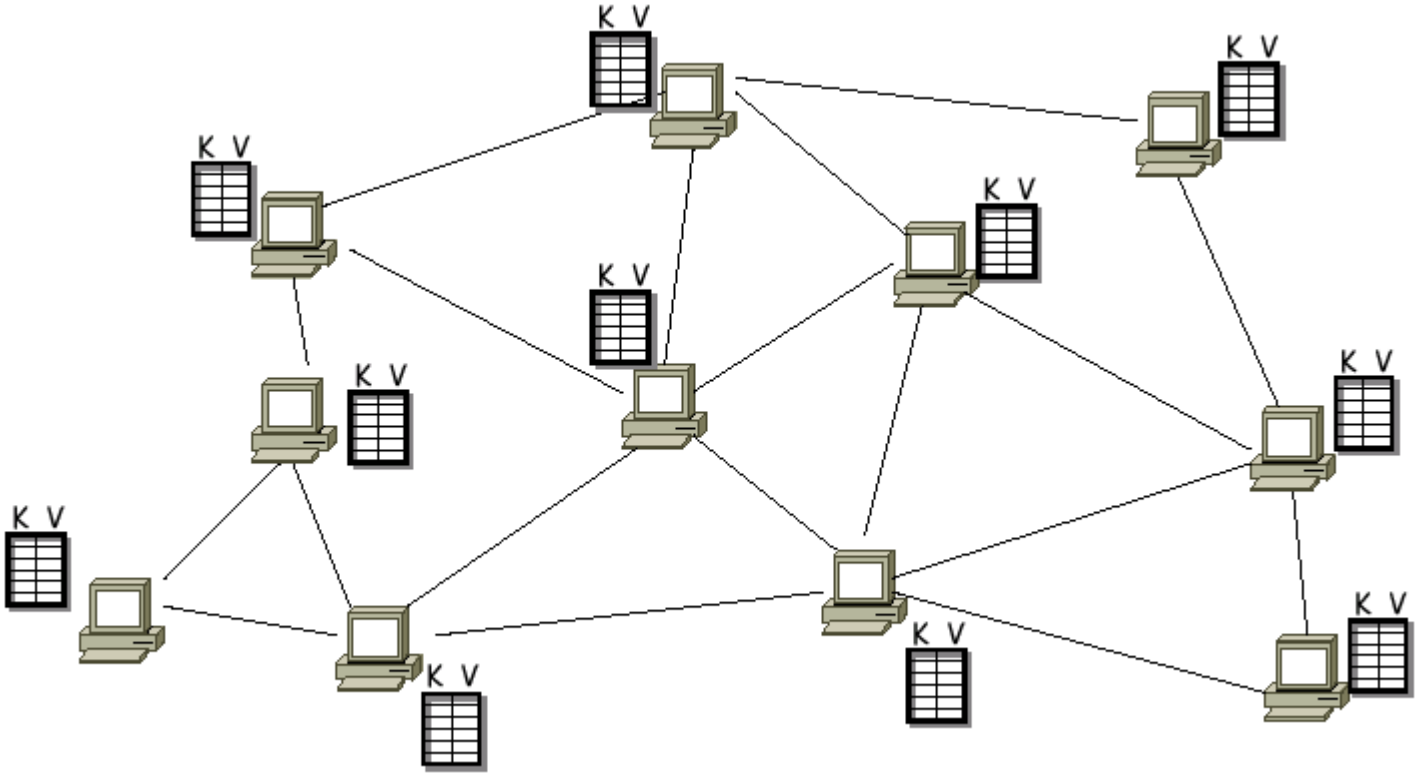
# DHT: Funcionamento

- Cada nó deve "conhecer" pelo menos **uma cópia do objeto que foi colocado na sua faixa de *hash***.
- Localização dos objetos
  - Nós armazenam os objetos que são mapeados para a sua faixa de *hash*.
  - Nós armazenam apontadores para os objetos na sua faixa.

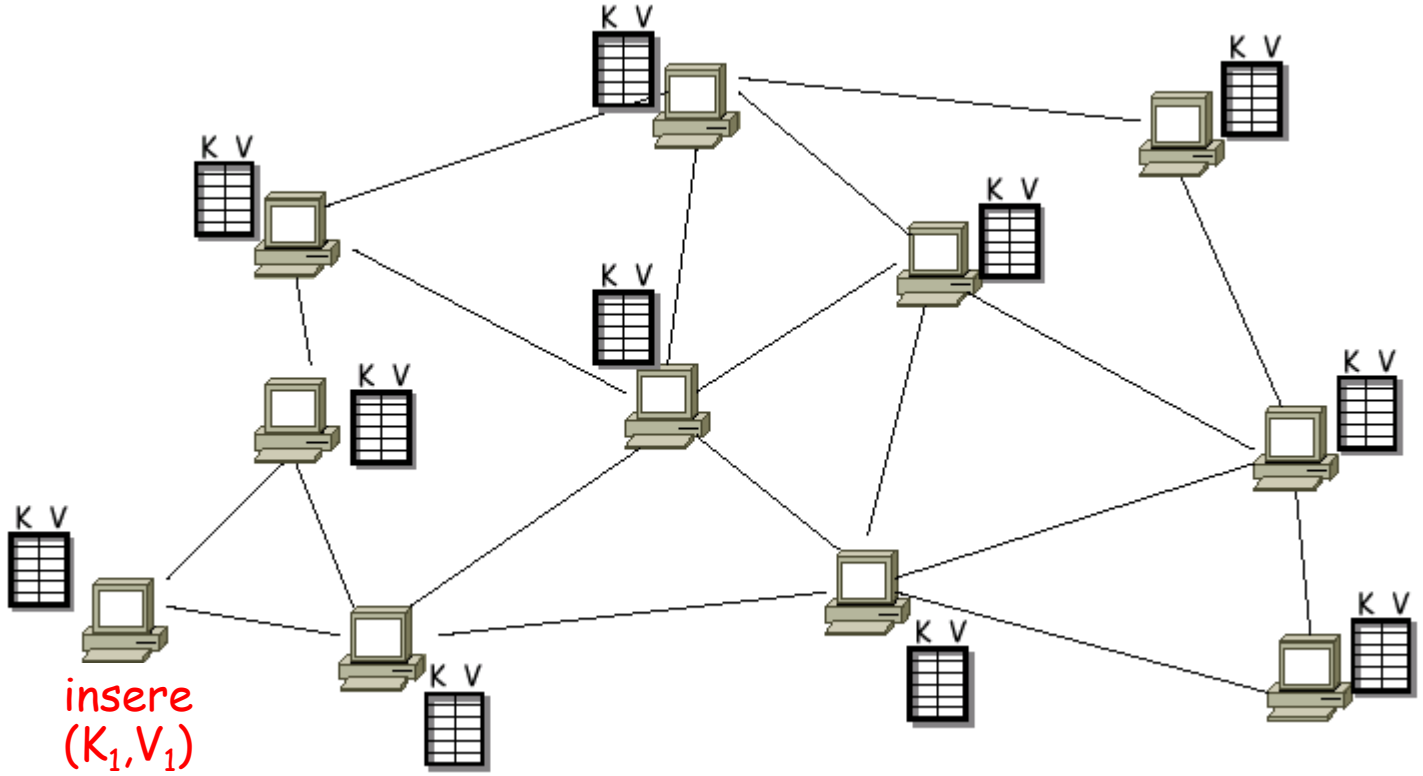
# DHT: Roteamento

- **Para cada objeto**, o nó (os nós) cuja faixa cobre o objeto deve ser **alcançável por um caminho "curto"**.
  - De qualquer outro nó.
- Em geral, qualquer função aleatória de *hash* "boa" é suficiente.
  - Padrão SHA-1 (colisão praticamente impossível)

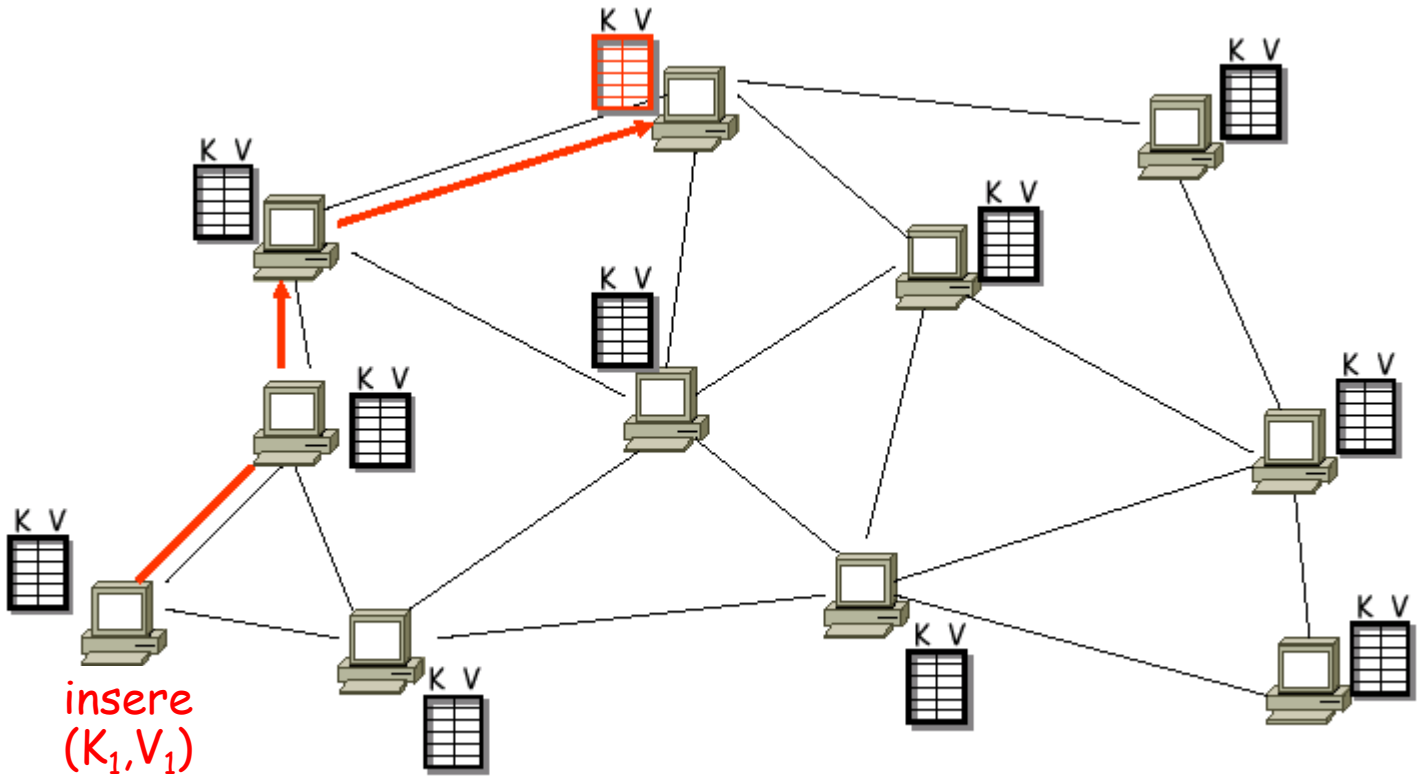
# DHT: Idéia Básica



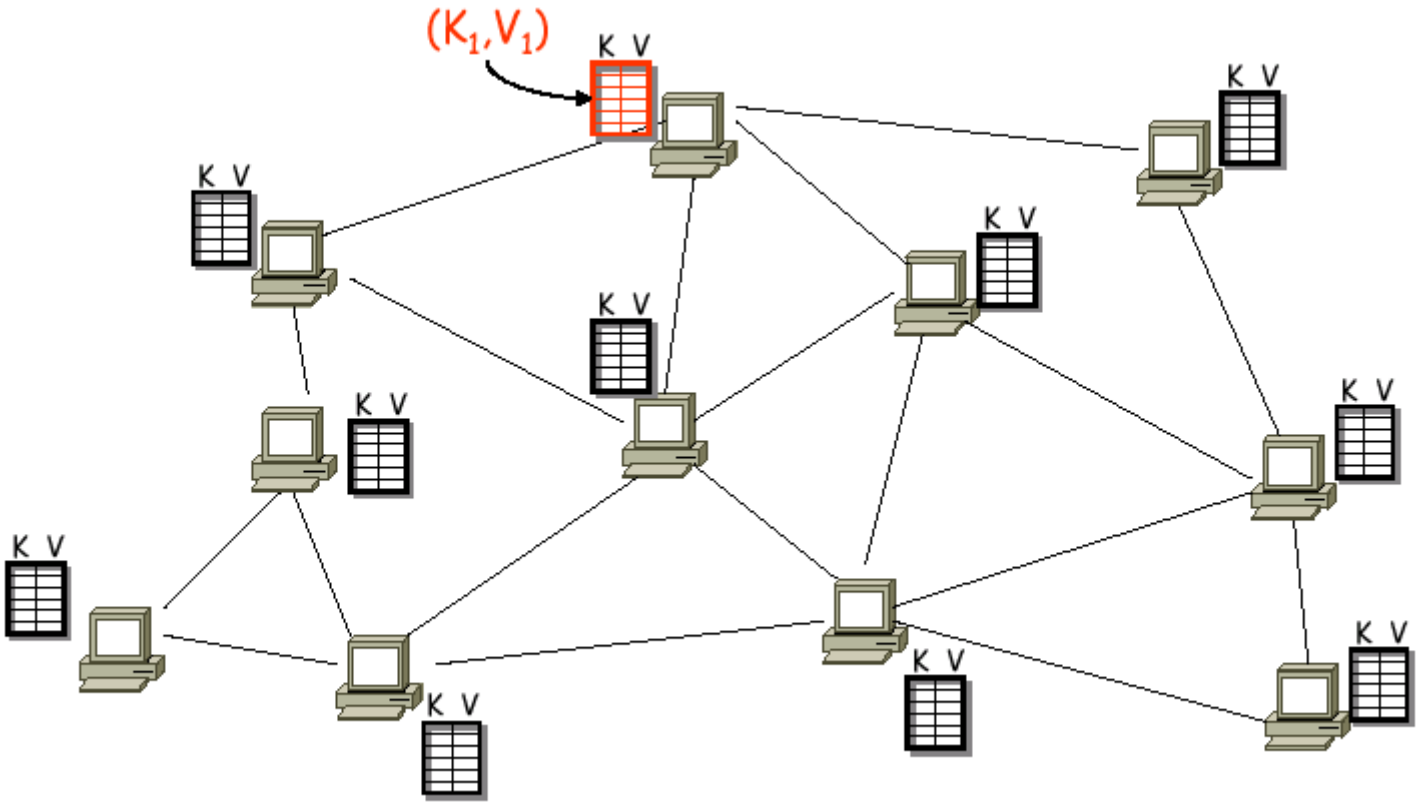
# DHT: Idéia Básica



# DHT: Idéia Básica



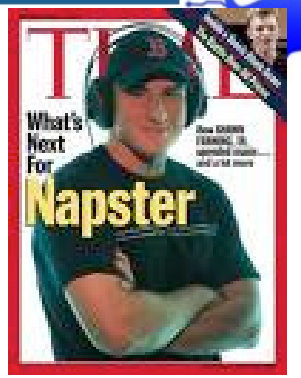
# DHT: Idéia Básica



# Aplicações mais populares

UEMG
FRUTAL

# Naspter




- Primeiro programa de compartilhamento massivo de arquivos através de P2P.
- Shawn Fanning.
  - Primeira versão: 1999
  - Popularidade: início de 2000.
  - Pico: 2001 → 8 M users/dia ≈ 20 M músicas / dia.
  - No início de 2001 não resistiu a uma série de ações legais e o serviço foi fechado em março.
    - Batalha judicial com a RIAA*
  - Novembro de 2002 → direitos para a Roxio.

* *Recording Industry Association of America.*

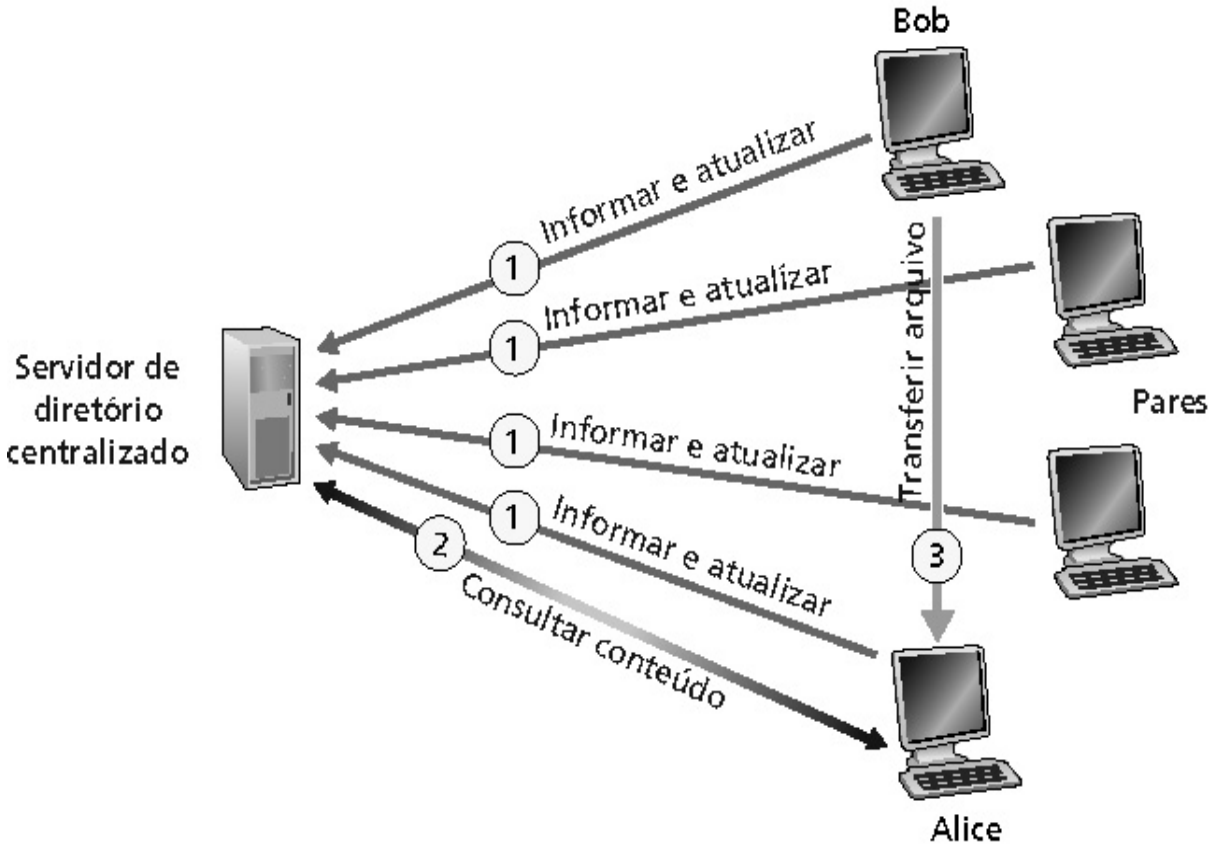
# Naspter simplificado

- Quando um par se conecta, ele informa ao servidor central:
  - **Endereço IP.**
  - **Conteúdo.**
- Alice procura por "*Hey Jude*".
- Servidor central informa onde existe este arquivo.
- Alice requisita o arquivo de Bob.

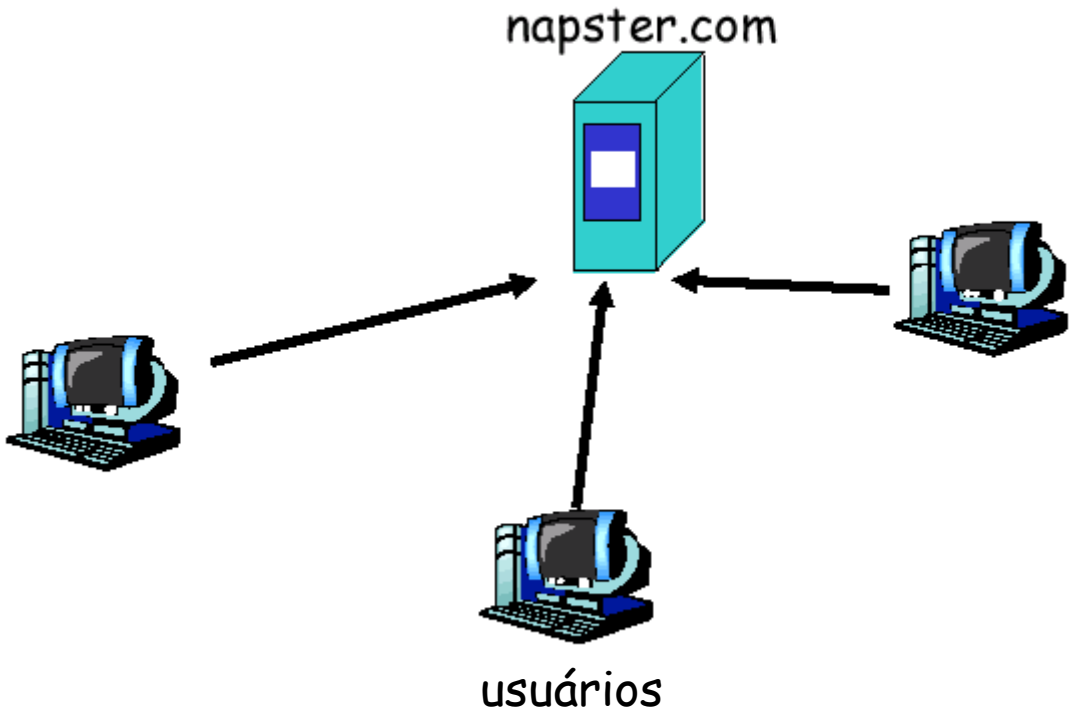
# Napster: funcionamento

1. Cliente se conecta com servidor e envia a sua lista de arquivos compartilhados.
2. **Cliente envia palavras-chave para fazer busca na lista completa.**
3. Cliente testa taxa de transmissão dos pares que têm o arquivo solicitado (*ping*).
4. ***Peers* respondem (*pong*).**
5. O arquivo é transferido **diretamente** entre os pares.

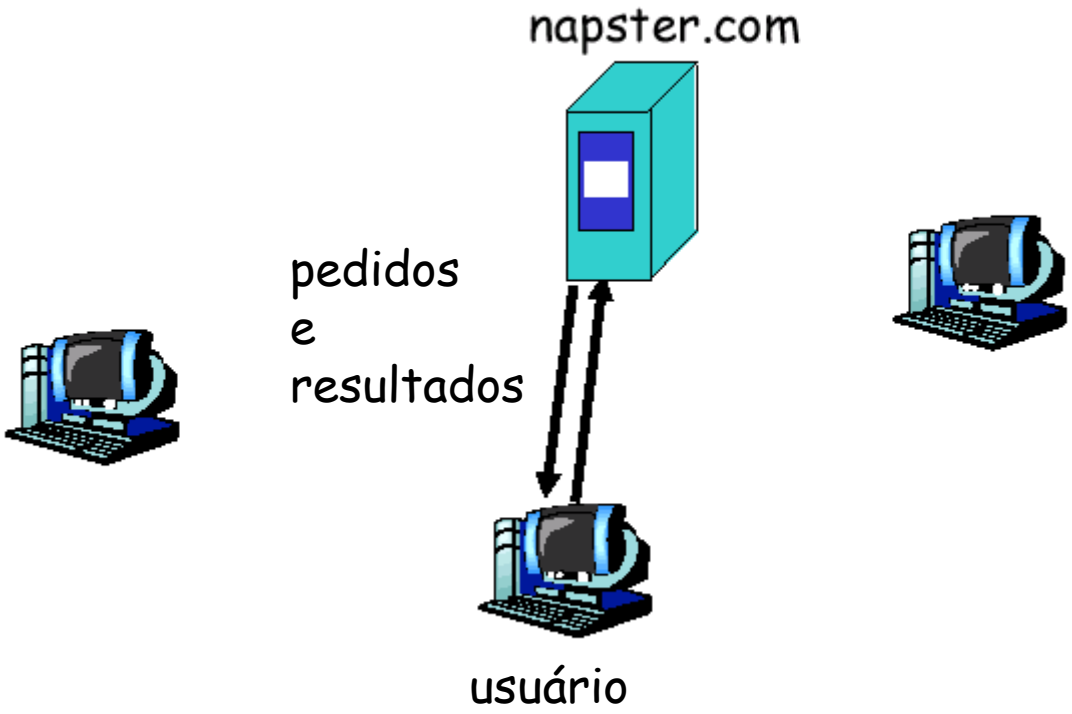
# P2P: diretório centralizado



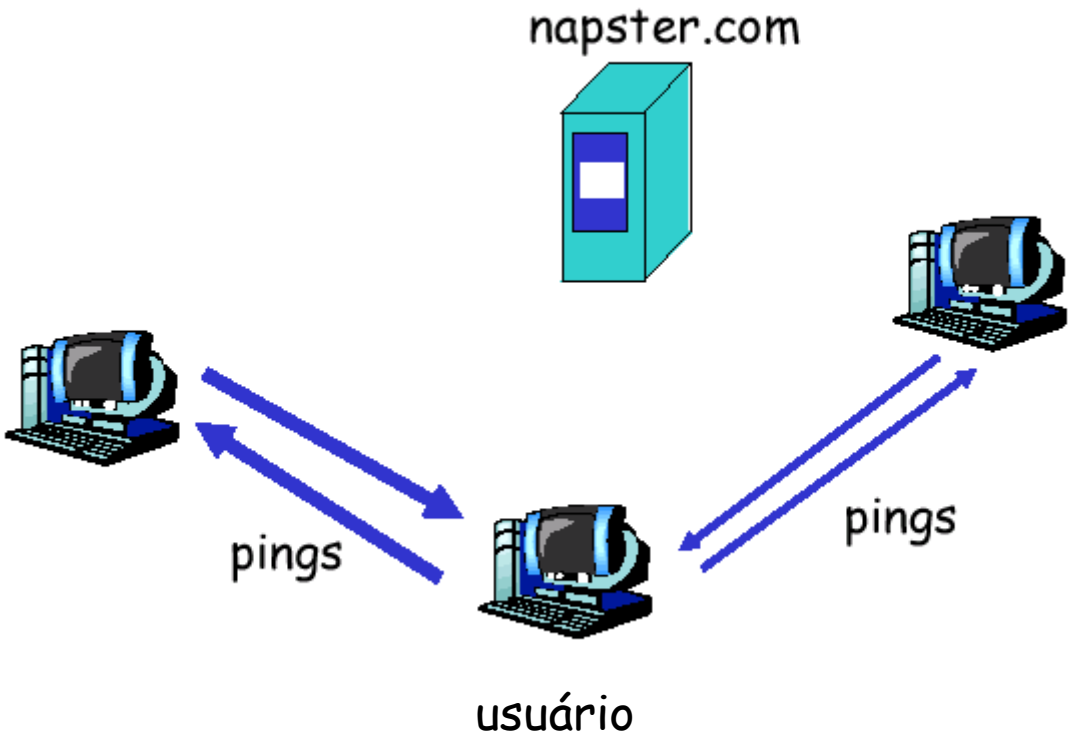
# Napster: funcionamento (1)



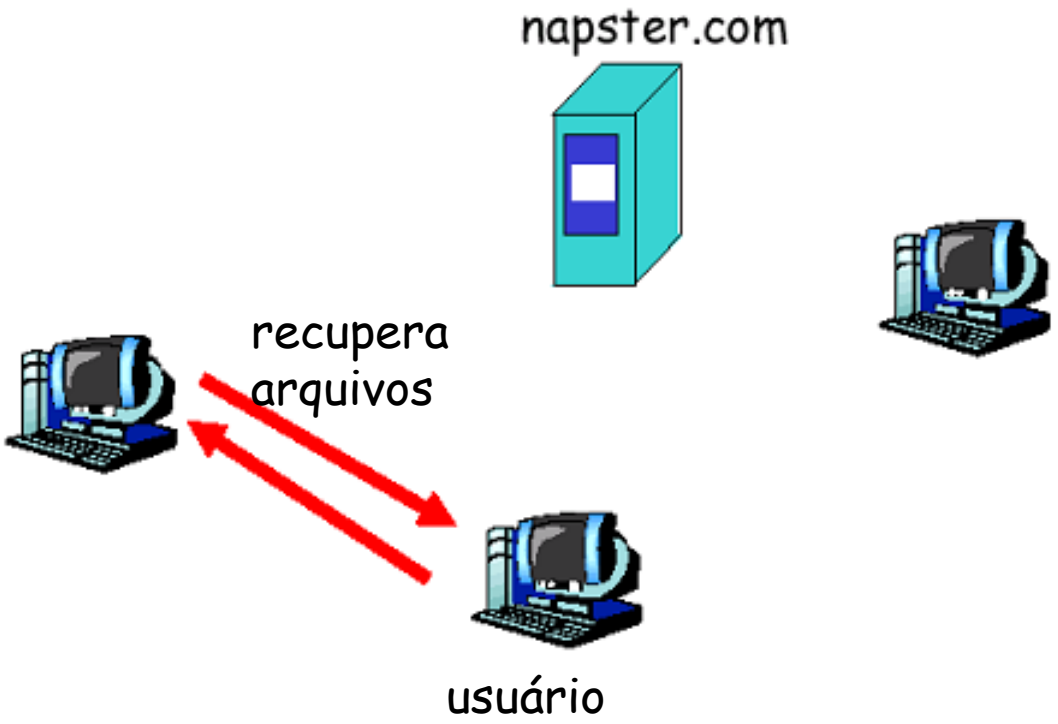
# Napster: funcionamento (2)



# Napster: funcionamento (3)



# Napster: funcionamento (4)



# Napster - Problemas

- Servidor centralizado
  - **Ponto único de falhas.**
  - Pode usar o DNS para balancear carga entre servidores.
  - Sujeito a congestionamentos.
  - Sob controle total do Napster.
    - Apenas ilusão de liberdade.
- Nenhuma segurança:
  - Sem criptografia.
  - Sem autenticação.
  - Sem privacidade (identidade é revelada).

# Gnutella (1)

- Sistema de busca **totalmente distribuído**.
  - Protocolo aberto.
- Busca baseada em inundação (*flooding*).
- História:
  - 14/03/2000: Disponibilizado sob licença pública GNU no servidor web da Nullsoft (pertencente à AOL).
  - Retirado apenas algumas horas depois.
  - Tarde demais: muitos usuários fizeram download.
  - O protocolo Gnutella foi "descoberto" através de engenharia reversa.
  - Outros clientes foram disponibilizados e Gnutella começou a se popularizar.

# Gnutella (2)

- A sua principal característica é a **busca distribuída**.
- Problema: Tráfego gerado = **Escalabilidade**.
- Despertou grande interesse na comunidade acadêmica.
  - Não depende de servidor central.
  - Problema inicial: descobrir algum nó que está na rede. Depois, já está na rede.

# Protocolo Gnutella

- Compartilhamento sobre uma rede de **overlay**
  - Nós mantêm conexões TCP abertas.
  - Mensagens são difundidas (inundadas) ou então propagadas de volta.
- Protocolo:

| | Inundação | Propagação de volta | Nó a nó |
|---|---|---|---|
| Participação | PING | PONG | |
| Consulta | QUERY | QUERY HIT | |
| Transferência de arquivos | | | GET, PUSH |

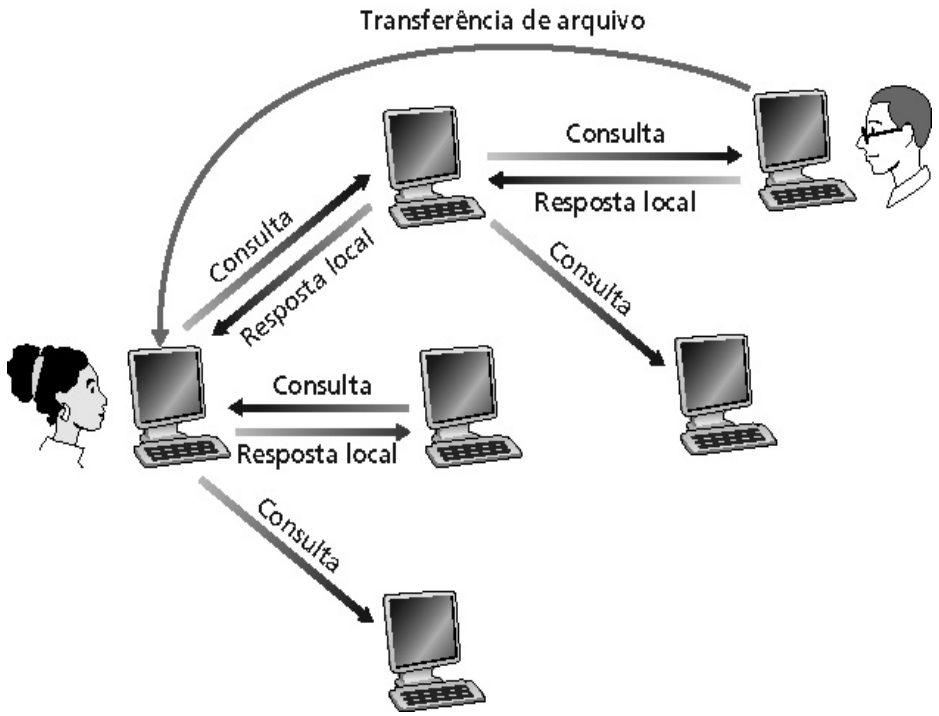
UEMG FRUTAL

# *Query flooding*: Gnutella

- Rede de cobertura: grafo.
- 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.

# 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.

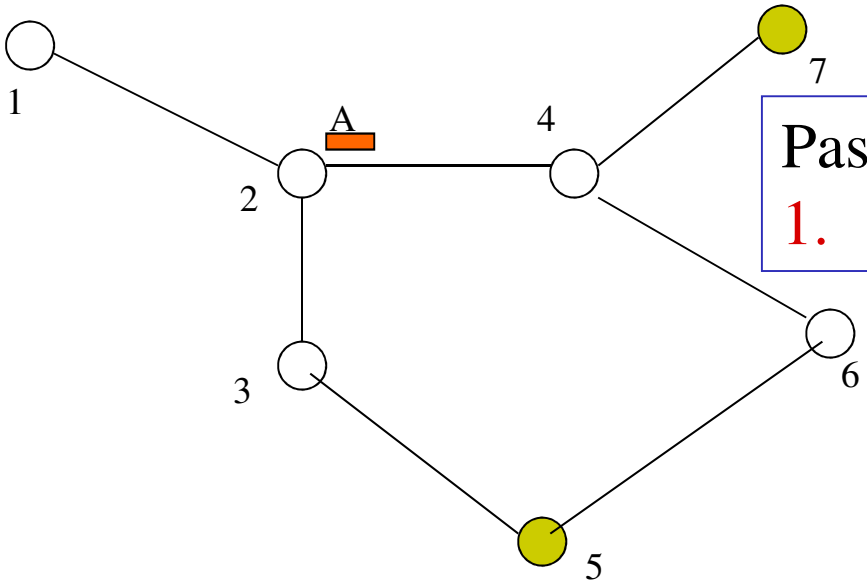



Para resolver problemas de escalabilidade: *flooding* de alcance limitado.

# Gnutella: conectando pares

- Para conectar o par X, ele precisa encontrar algum outro par na rede Gnutella: utiliza a **lista de pares candidatos.**
- X seqüencialmente, tenta fazer conexão TCP com os pares da lista, até estabelecer conexão com Y.
- X envia mensagem de *ping* para Y;
  - Y encaminha a mensagem de *ping* aos vizinhos.
- Todos os pares que recebem a mensagem de *ping* respondem com mensagens de *pong*.
- X recebe várias mensagens de *pong*.
  - Ele pode então estabelecer conexões TCP adicionais.

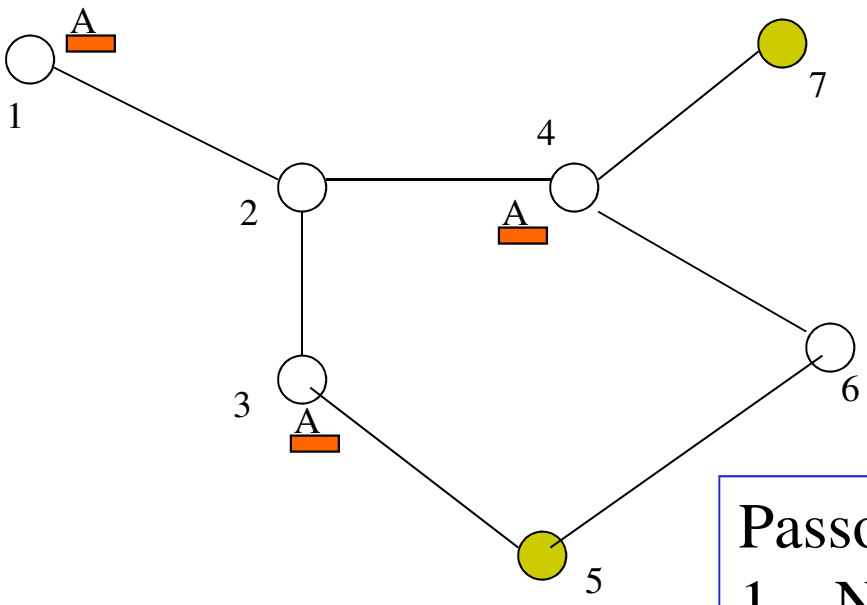
# Gnutella: Mecanismo de busca




Passos:

1. Nó 2 inicia busca do arquivo A

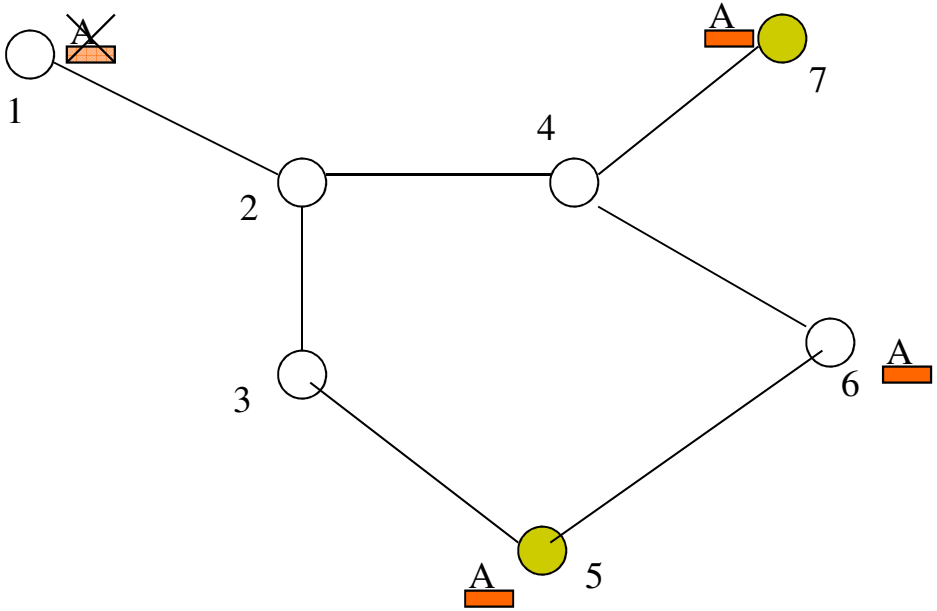
# Gnutella: Mecanismo de busca




Passos:

1. Nó 2 inicia busca do arquivo A.
2. Envia mensagens a vizinhos.

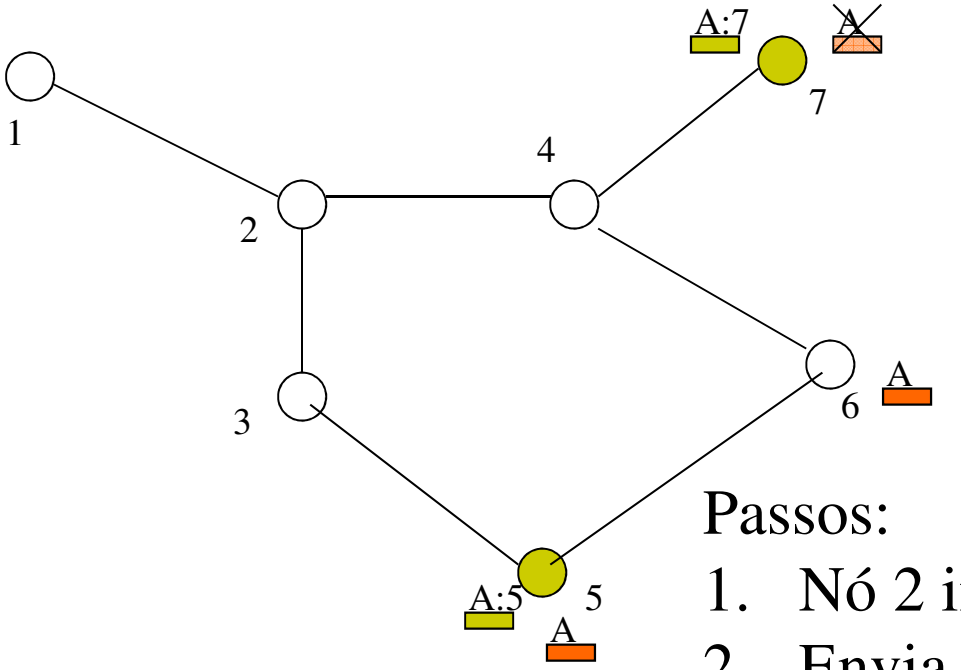
# Gnutella: Mecanismo de busca




Passos:

1. Nó 2 inicia busca do arquivo A.
2. Envia mensagens a vizinhos.
3. Vizinhos encaminham mensagem.

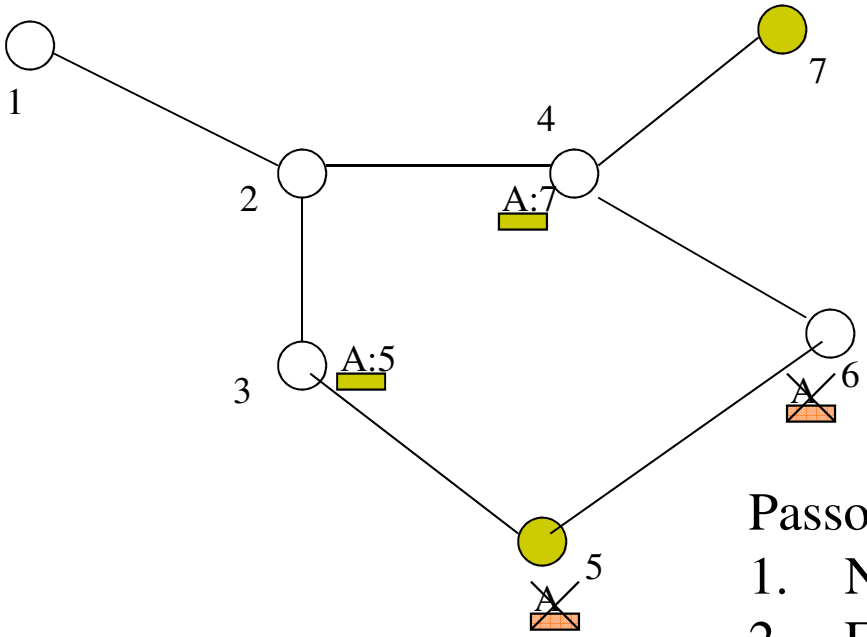
# Gnutella: Mecanismo de busca




Passos:

1. Nó 2 inicia busca arquivo *A*.
2. Envia mensagens a vizinhos.
3. Vizinhos encaminham mensagem.
4. Nós com arquivo *A* enviam mensagem de resposta.

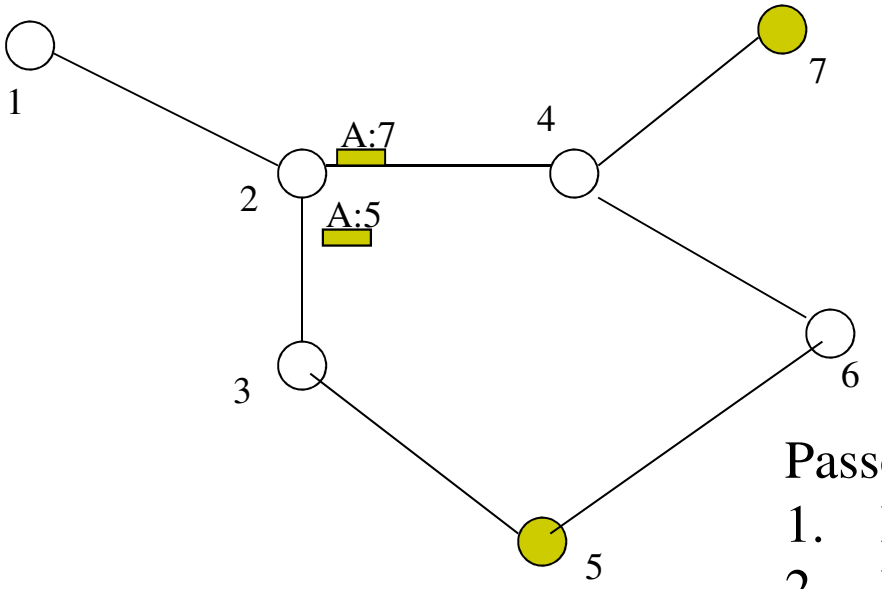
# Gnutella: Mecanismo de busca




Passos:

1. Nó 2 inicia busca arquivo A.
2. Envia mensagens a vizinhos.
3. Vizinhos encaminham mensagem.
4. Nós com arquivo A enviam. mensagem de resposta.
5. Mensagem de resposta propagada de volta.

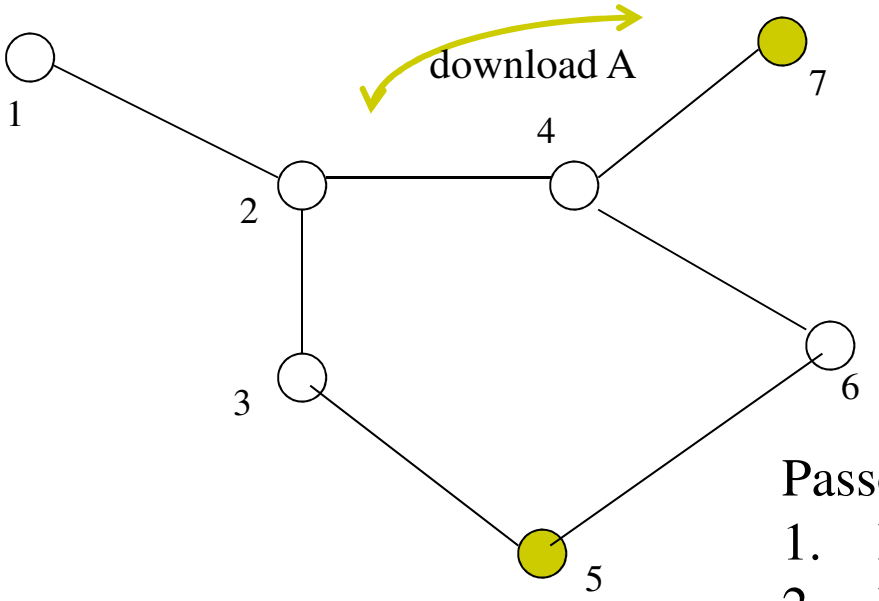
# Gnutella: Mecanismo de busca




Passos:

1. Nó 2 inicia busca arquivo A.
2. Envia mensagens a vizinhos.
3. Vizinhos encaminham mensagem.
4. Nós com arquivo A enviam mensagem de resposta.
5. Mensagem de resposta propagada de volta.

# Gnutella: Mecanismo de busca



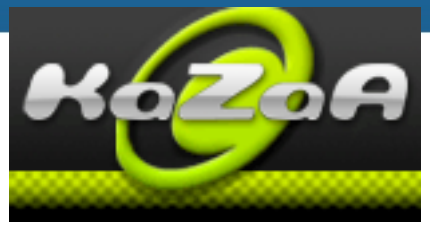

Passos:

1. Nó 2 inicia busca arquivo A.
2. Envia mensagens a vizinhos.
3. Vizinhos encaminham mensagem.
4. Nós com arquivo A enviam mensagem de resposta.
5. Mensagem de resposta propagada de volta.
6. Arquivo A é transferido.

# Gnutella - Comentários

- Versões mais novas do protocolo Gnutella utilizam conceito de super-nós para minimizar o tráfego de inundação.

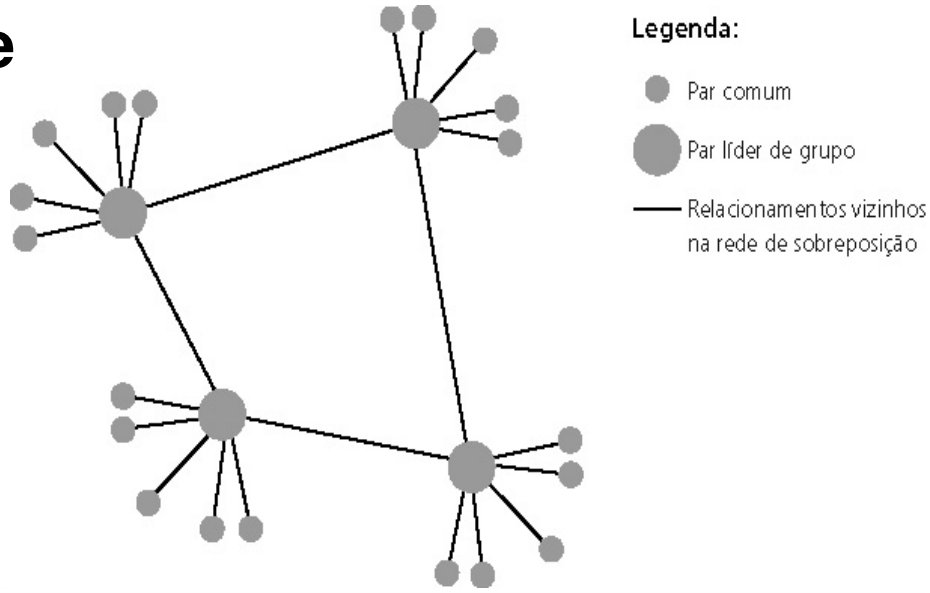
# KaZaA




- **Software proprietário.**
- Tentativa de implementar reputação.
- Protocolo *FastTrack*
- Outros clientes
  - Versão pirata: KaZaA Lite
  - Morpheus, Grokster
- Arquitetura:
  - Descentralizada e não estruturada.
  - **Hierárquica**: Baseada em super-nós (*supernodes*).

# Explorando heterogeneidade: KaZaA

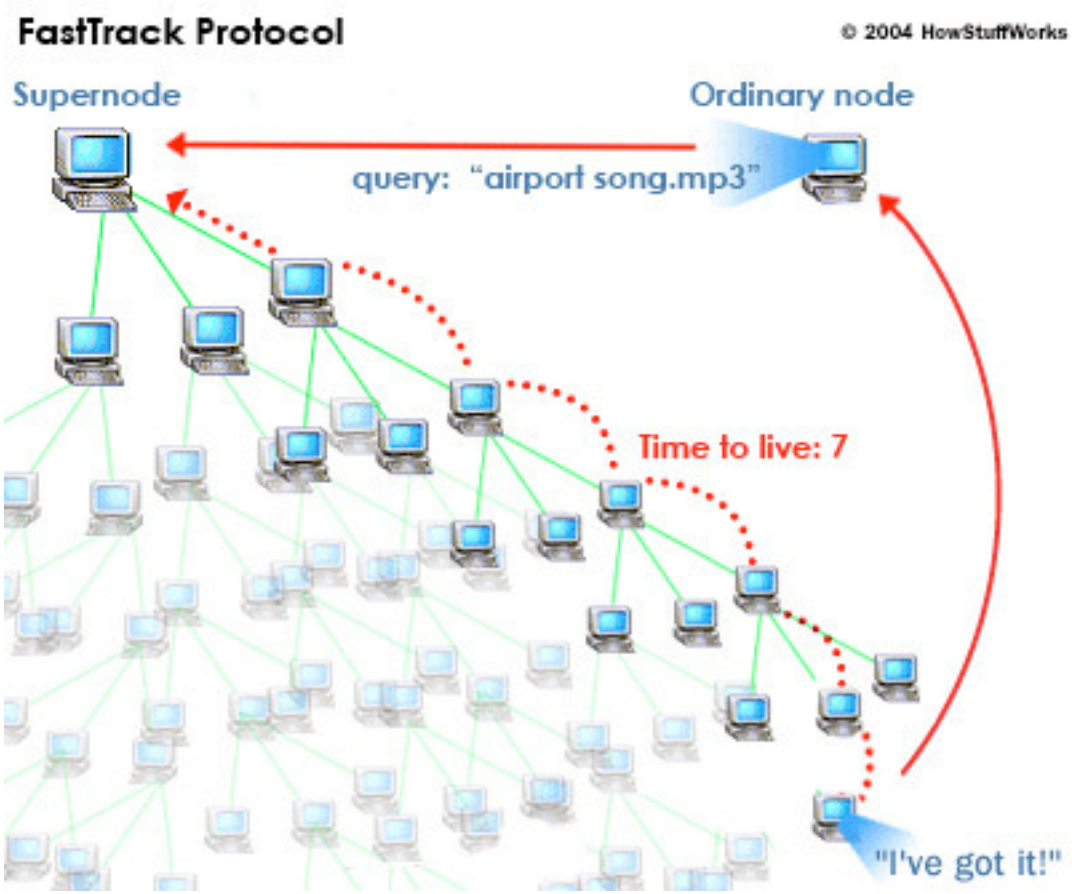
- Cada par é **ou um líder** de grupo **ou está atribuído a um líder de grupo**.
- Conexão TCP entre o par e seu líder de grupo.
- Conexões TCP **entre alguns pares de líderes de grupo**.
- O **líder de grupo** acompanha o conteúdo em todos os seus "discípulos".



# KaZaA – funcionamento (1)

- 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 *matches*:
- Para cada match: **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.

# KaZaA – funcionamento (2)



# Artifícios do KaZaA

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

# BitTorrent

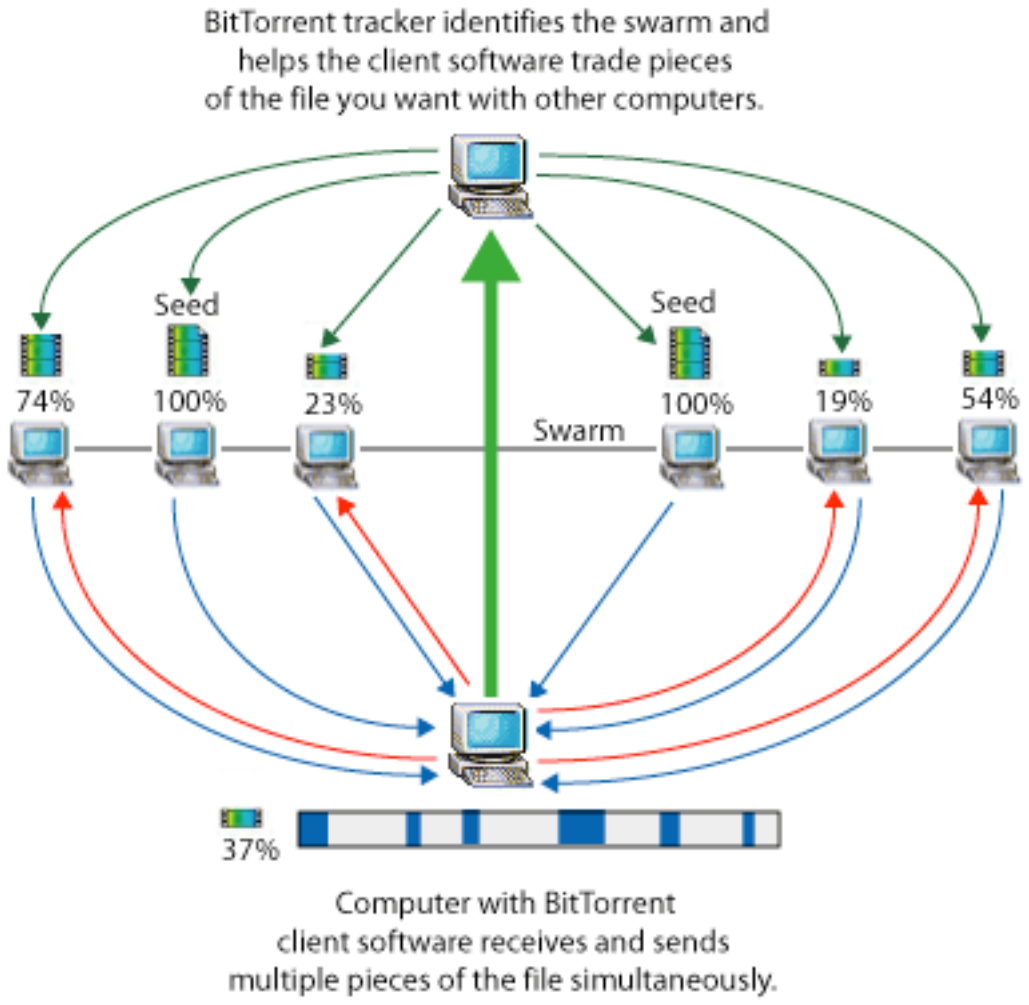

- Bram Cohen – Abril de 2001.
- Procolo aberto.
- Descentralizado.
- Modelo híbrido.
- DHT (Kademlia).
- Implementado pelo Azureus (atual Vuze)

# BitTorrent (2)

- ***Tracker***.
  - *Peer* cria um arquivo de metadata: ***.torrent*** contendo um *hash* do que vai ser compartilhado.
  - Envia para o *tracker* → servidor supernode.
  - Clientes buscam informações dos compartilhamento nos *trackers*.
    - Também possibilidade de operação *trackerless* e *multitracker*.
- Clientes:
  - BitTorrent, µTorrent, rTorrent, KTorrent, BitComet → DHT
  - Vuze (Ex-Azureus) → suporta *trackerless* (incompatível com DHT, apesar de que desenvolveu o DHT primeiro).

UEMG FRUTAL

# BitTorrent- funcionamento



# Programação com *sockets*

Como funcionam as aplicações cliente/servidor

UEMG
FRUTAL

# Programação com *sockets*

Meta: aprender construir aplicação cliente/servidor que se comunica usando *sockets*.

## *API Sockets*

- Surgiu em BSD4.1 UNIX, 1981
- Explicitamente criados, usados e liberados por aplicações.
- Paradigma cliente/servidor
- Dois tipos de serviço de transporte via ***API Sockets***
  - Datagrama não confiável
  - Fluxo de bytes, confiável

### *socket*

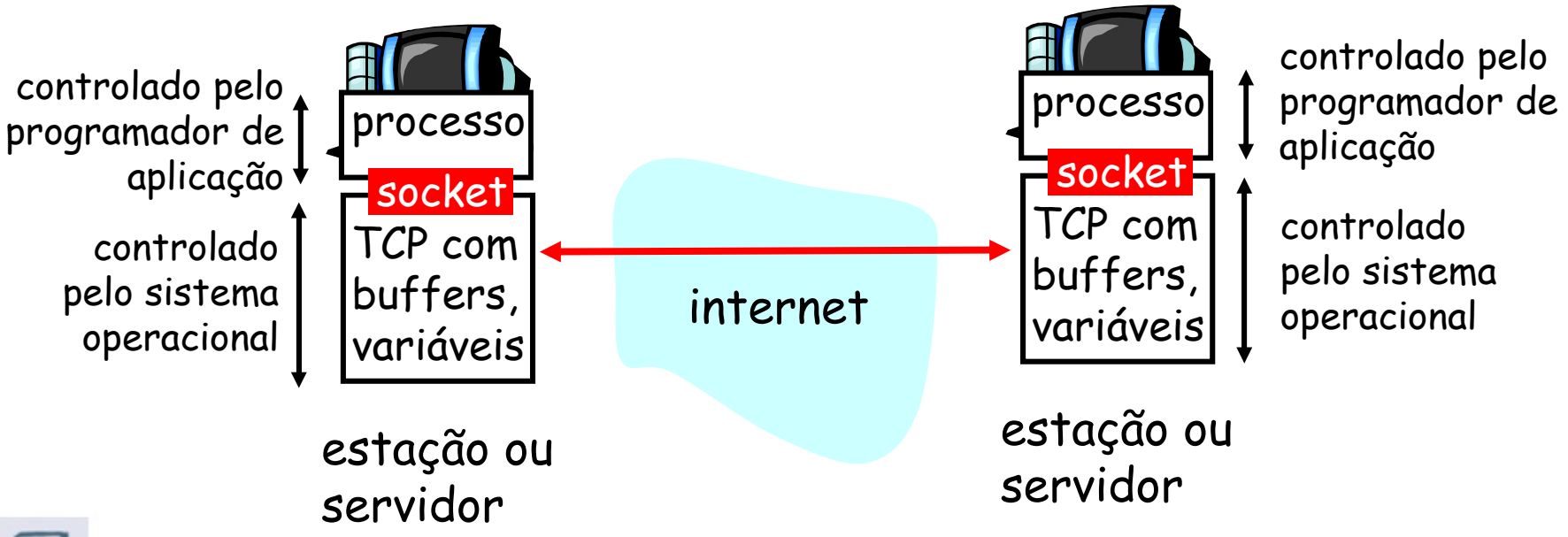
Uma interface (uma "porta"), local ao hospedeiro, criada por e pertencente à aplicação, e controlado pelo SO, através da qual um processo de aplicação pode tanto enviar como receber mensagens de/para outro processo de aplicação
(remoto ou local)

UEMG FRUTAL

# Programação com *sockets* usando TCP

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

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



# Programação com sockets usando TCP

Cliente deve contactar servidor:

- Processo servidor deve antes estar em execução
- **Servidor deve antes ter criado *socket* (porta) que aguarda contato do cliente**

Cliente contacta servidor por:

- Criar *socket* TCP local ao cliente
- Especificar endereço IP, número de porta do processo servidor

- Quando cliente cria *socket*: TCP do cliente estabelece conexão ao servidor TCP
- Quando contactado pelo cliente, servidor TCP cria *socket* novo processo servidor poder se comunicar com o cliente.
  - Permite que o servidor converse com múltiplos clientes.

ponto de vista da aplicação

*TCP provê transferência confiável, ordenada de bytes ("tubo") entre cliente e servidor*

UEMG FRUTAL

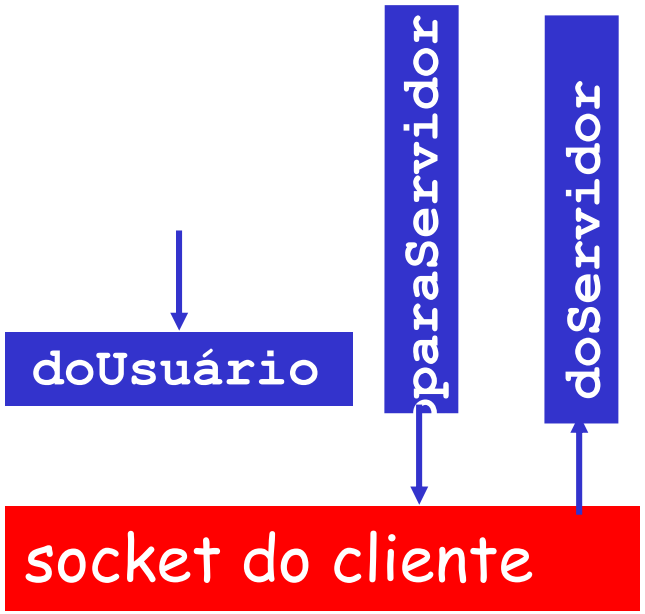
# Programação com sockets usando TCP

Exemplo de aplicaçao cliente-servidor:

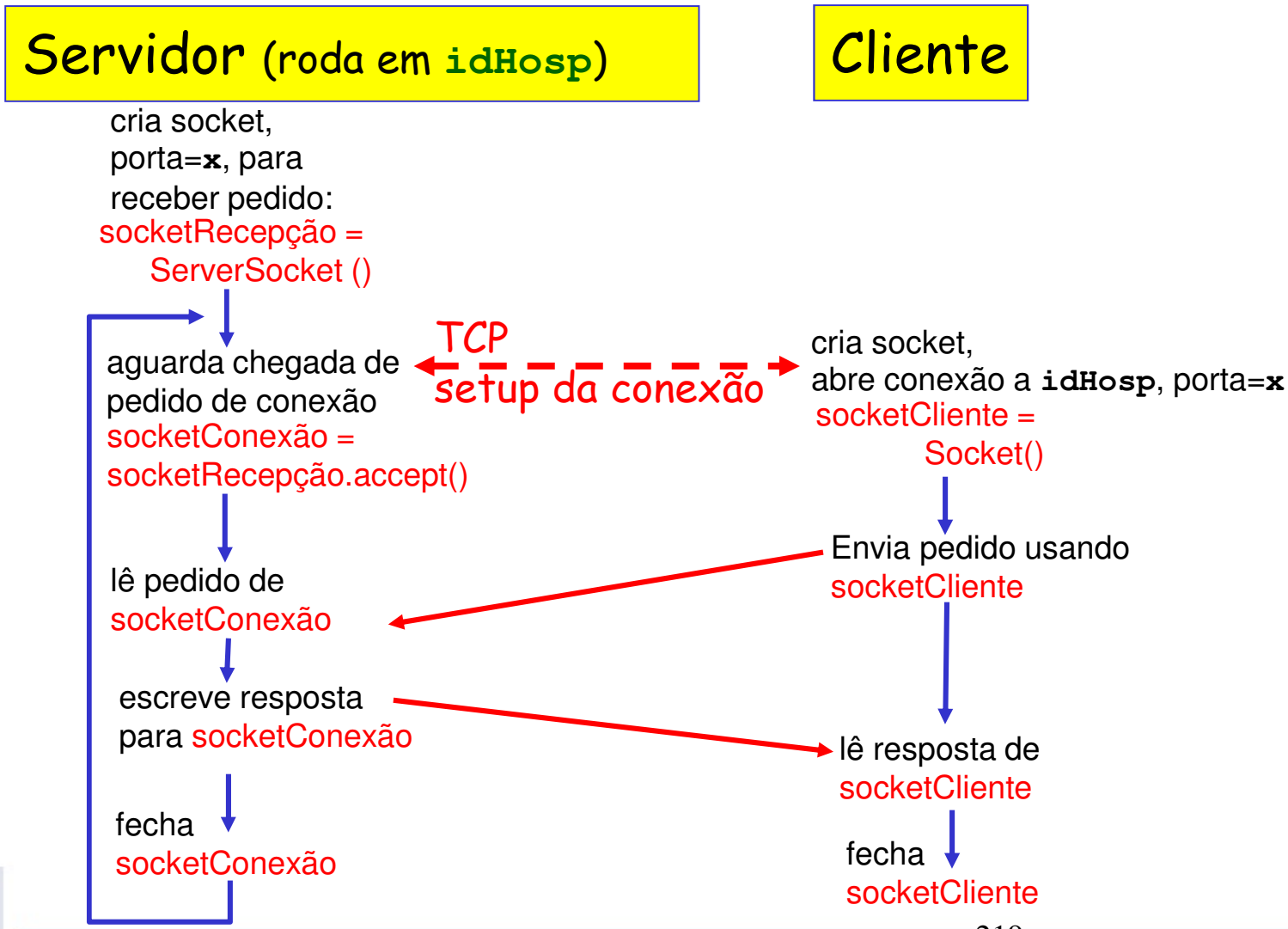
- Cliente lê linha da entrada padrão **(fluxo `doUsuário`),** envia para servidor via socket **(fluxo `paraServidor`).**
- Servidor lê linha do *socket*.
- Servidor converte linha para letra maiúscula, e devolve para o cliente.
- Cliente lê linha modificado do socket **(fluxo `doServidor`)**, imprime-a

***Input stream*:** seqüência de bytes para **dentro** do processo.

***Output stream*:** seqüência de bytes para fora do processo.



# Interações cliente/servidor com socket: TCP



# Exemplo: cliente Java TCP (1)

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

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

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

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

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

Contém classe para streams de I/O

Contém classes para suporte a rede

Cria fluxo de entrada

Cria socket de cliente, conexão ao servidor

Cria fluxo de saída anexado ao socket

# Exemplo: cliente Java TCP (2)

Cria fluxo de entrada ligado ao socket

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

frase = doUsuario.readLine();

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

fraseModificada = doServidor.readLine();

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

socketCliente.close();

    }
}
```

Envia linha ao servidor

Lê linha do servidor

UEMG FRUTAL

# Exemplo: servidor Java TCP (1)

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

class servidorTCP {

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

      ServerSocket socketRecepcao = new ServerSocket(6789);

      while(true) {

        Socket socketConexao = socketRecepcao.accept();

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

Cria socket para recepção na porta 6789

Aguarda, no socket para recepção, o contato do cliente

Cria fluxo de entrada, ligado ao socket

# Exemplo: servidor Java TCP (2)

Cria fluxo de saída, ligado ao socket

Lê linha do socket

Escreve linha para o socket

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

fraseCliente= doCliente.readLine();

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

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

Final do laço while. Volta ao início e aguarda conexão de outro cliente

# Programação com *sockets* usando UDP

UDP: não tem "conexão" entre cliente e servidor.

- Não tem "*handshaking*"
- Remetente coloca explicitamente endereço IP e porta do destino.
- Servidor deve extrair endereço IP, porta do remetente do datagrama recebido

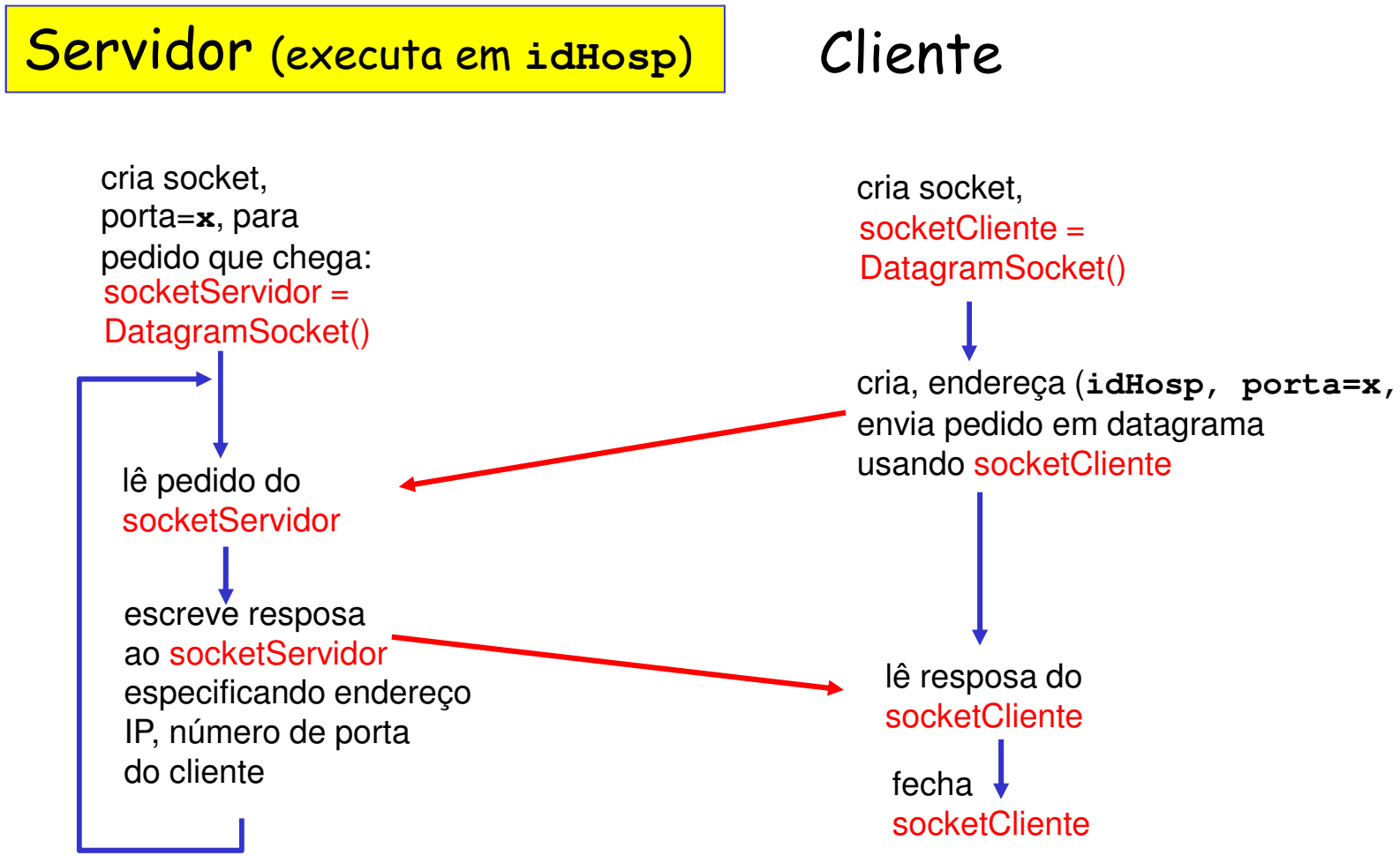
Dados transmitidos podem ser recebidos fora de ordem, ou perdidos.

ponto de vista da aplicação

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

UEMG FRUTAL

# Interações cliente/servidor com *socket*: UDP



# Exemplo: cliente Java UDP (1)

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

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

      BufferedReader do Usuario=
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket socketCliente = new DatagramSocket();

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

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

      String frase = doUsuario.readLine();

      dadosEnvio = frase.getBytes();
```

Cria fluxo de enrada

Cria socket de cliente

Traduz nome de hospedeiro para endereço IP usando DNS

# Exemplo: cliente Java UDP (2)

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

Envia datagrama ao servidor

Lê datagrama do servidor

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

socketCliente.send(pacoteEnviado);

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

socketCliente.receive(pacoteRecebido);

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

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

}
```

# Exemplo: servidor Java UDP (1)

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

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

      DatagramSocket socketServidor = new DatagramSocket(9876);

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

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

          socketServidor.receive(pacoteRecebido);
```

Cria socket para datagramas na porta 9876

Aloca memória para receber datagrama

Recebe datagrama

# Exemplo: servidor Java UDP (2)

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

InetAddress IPAddress = pacoteRecebido.getAddress();

int port = pacoteRecebido.getPort();

String fraseEmMaiusculas = frase.toUpperCase();

dadosEnviados = fraseEmMaiusculas.getBytes();

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

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

Obtém endereço IP e No. de porta do remetente

Cria datagrama p/ enviar ao cliente

Escreve datagrama para o socket

Fim do laço while.
Volta ao início e aguarda chegar outro datagrama.

# Capítulo 2: Sumário

Terminamos nosso estudo de aplicações de rede!

- Requisitos do serviço de aplicação:
  - Confiabilidade, banda, retardo.
- Paradigma cliente-servidor.
- Modelo de serviço do transporte orientado a conexão, confiável da Internet: TCP.
- Modelo não confiável: datagramas UDP.
- Protocolos específicos:
  - HTTP
  - FTP.
  - SMTP, IMAP e POP3.
  - DNS.
  - P2P.
- Sockets
  - Implementação cliente/servidor.
  - Usando sockets tcp, udp.

UEMG FRUTAL

# Capítulo 2: Sumário

## Mais importante - aprendemos de **protocolos**:

- Troca típica de mensagens em pedido/resposta:
  - Cliente solicita info ou serviço.
  - Servidor responde com dados, e código de *status*.
- Formatos de mensagens:
  - Cabeçalhos: campos com informações sobre dados (metadados).
  - Dados: informação sendo comunicada.

- Msgs de controle X dados.
  - na banda, fora da banda
- Centralizado X descentralizado
- Sem estado X com estado
- Transferência de msgs confiável X não confiável
- "Complexidade na borda da rede".

UEMG FRUTAL


**Vinicius Fernandes Caridá**

vfcarida@gmail.com