

## Parte 2: Camada de Aplicação

### Objetivos:

- ❑ conceitual, aspectos de implementação de protocolos de aplicação para redes
  - o modelos de serviço da camada de transporte
  - o paradigma cliente-servidor
  - o paradigma P2P
- ❑ aprender sobre protocolos examinando alguns protocolos populares da camada de aplicação
  - o HTTP
  - o FTP
  - o SMTP / POP3 / IMAP
  - o DNS

## Algumas aplicações de rede

- ❑ Correio eletrônico
- ❑ Web
- ❑ Troca instantânea de mensagens
- ❑ *Remote login*
- ❑ Compartilhamento de arquivos P2P
- ❑ Jogos multiusuário em rede
- ❑ *Streaming* de vídeo armazenado
- ❑ Telefonia pela Internet
- ❑ Conferência de vídeo em tempo-real
- ❑ Computação com paralelismo massivo

## Paradigma Cliente-Servidor

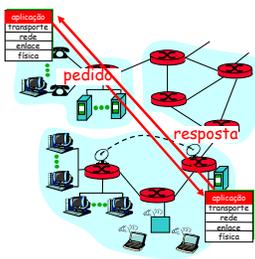
Aplicações de rede típicas têm duas partes: *cliente* e *servidor*

### Cliente:

- ❑ inicia a comunicação com o servidor ("fala primeiro")
- ❑ tipicamente solicita serviços do servidor,
- ❑ Web: cliente implementado no navegador; correio-e: leitor de correio

### Servidor:

- ❑ fornece os serviços solicitados ao cliente
- ❑ e.x., servidor Web envia a página Web solicitada, servidor de correio-e envia as mensagens, etc.

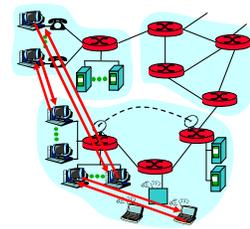


## Arquitetura P2P pura

- ❑ Sistemas finais arbitrários comunicam-se diretamente
- ❑ Pares estão conectados intermitentemente e trocam endereços IP
- ❑ exemplo: Gnutella

Muito escalável

Mas difícil de gerenciar



## Híbrida de cliente-servidor e P2P

### Napster

- o Transferência de arquivos é P2P
- o Busca de arquivos centralizada:
  - Pares registram o conteúdo em servidor central
  - Pares consultam o mesmo servidor central para localizar conteúdo

### Mensagens Instantâneas (*Instant messaging*)

- o Conversa (*chatting*) entre dois usuários é P2P
- o Detecção de presença e localização é centralizada:
  - Usuário registra seu endereço IP em um servidor central quando fica em-linha
  - Usuário contata o servidor central para encontrar o endereço IP dos parceiros

## Comunicação entre processos

**Processo:** programa executando num hospedeiro.

- ❑ dentro do mesmo hospedeiro: **comunicação interprocesso** (definida pelo OS).
- ❑ processos executando em diferentes hospedeiros se comunicam através de **trocas de mensagens**
  - o definem um **protocolo de aplicação**

❑ **Processo cliente:** processo que inicia a comunicação

❑ **Processo servidor:** processo que espera para ser contatado

- ❑ Nota: aplicações P2P têm processos clientes & processos servidores

## Interfaces de Programação

### API: *application programming interface*

- ❑ define a interface entre a camada de aplicação e de transporte
- ❑ soquete (*socket*): API da Internet
  - o dois processos se comunicam enviando dados para o soquete e lendo dados do soquete

**Q:** Como um processo "identifica" o outro processo com o qual ele quer se comunicar?

- o Endereço IP do computador no qual roda o processo remoto
- o "número de porta" - permite ao computador receptor determinar o processo local para o qual a mensagem deve ser entregue.

## Aplicações e Protocolo de Aplicação

### Aplicação: processos distribuídos comunicantes

- o rodam nos hospedeiros da rede no "espaço do usuário"
- o trocam mensagens para realização da aplicação
- o e.x., email, ftp, Web

### Protocolos de aplicação

- o fazem parte das aplicações
- o definem as mensagens trocadas e as ações tomadas
- o usam serviços de comunicação das camadas inferiores (TCP, UDP)

### ❑ Protocolos de domínio público:

- o defined in RFCs
- o allows for interoperability
- o Ex., HTTP, SMTP

### ❑ Protocolos proprietários:

- o Ex., KaZaA

## Serviços de Transporte

### Perda de dados

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

### Banda Passante

- ❑ algumas aplicações (e.x., multimídia) exigem uma banda mínima
- ❑ outras aplicações ("aplicações elásticas") utilizam toda a largura de banda oferecida

### Temporização

- ❑ algumas aplicações (e.x., telefonia sobre a Internet, jogos interativos) exigem baixos atrasos

## Requisitos de Transporte de Aplicações Comuns

Aplicação	Perdas	Banda	Sensível ao Atraso
transf. de arquivos	sem perdas	elástica	não
correio-e	sem perdas	elástica	não
documentos Web	tolerante	elástica	não
áudio/vídeo tempo-real	tolerante	áudio: 5Kb-1Mb vídeo: 10Kb-5Mb	sim, 100's ms
áudio/vídeo armazenado	tolerante	igual à anterior	sim, segundos
jogos interativos	tolerante	Kbps	sim, 100's ms
mensagens instantâneas	sem perda	elástica	sim e não

## Serviços de Transporte da Internet

### serviço TCP:

- ❑ *orientado a conexão*: requerido o estabelecimento de conexão entre cliente e servidor
- ❑ *transporte confiável* de dados entre os processos emissor e receptor
- ❑ *controle de fluxo*: compatibilização de velocidade entre o transmissor e o receptor
- ❑ *controle de congestionamento*: protege a rede do excesso de tráfego
- ❑ *não oferece*: garantias de temporização e de banda mínima

### serviço UDP:

- ❑ transferência de dados não confiável entre os processos transmissor e receptor
- ❑ não oferece: estabelecimento de conexão, confiabilidade, controle de fluxo e de congestionamento, garantia de temporização e de banda mínima.

## Aplicações e Protocolos de Transporte da Internet

Aplicação	Protocolo de Aplicação	Protocolo de Transporte
e-mail	smtp [RFC 821]	TCP
acesso a terminal remotos	telnet [RFC 854]	TCP
Web	http [RFC 2068]	TCP
transferência de arquivos	ftp [RFC 959]	TCP
streaming multimídia	RTP ou proprietário (e.g. RealNetworks)	TCP ou UDP
servidor de arquivos remoto	NFS	TCP ou UDP
telefonia Internet	RTP ou proprietário (e.g., Vocaltec)	tipicamente UDP

## A Web e o HTTP

- ❑ Página consiste de **objetos**
- ❑ Objetos: arquivo HTML, imagem JPEG, applet Java, arquivo de áudio,...
- ❑ Página Web consiste de um **arquivo-base-HTML** que inclui vários objetos referenciados
- ❑ Cada objeto é endereçável por uma **URL**
- ❑ Exemplo de URL:

www.someschool.edu/someDept/pic.gif

nome do  
hospedeiro

caminho

## A Web: o protocolo HTTP

http: *hypertext transfer protocol*

- ❑ protocolo da camada de aplicação da Web
- ❑ modelo cliente/servidor
  - o *cliente*: navegador que solicita, recebe e apresenta objetos da Web
  - o *servidor*: envia objetos em resposta a pedidos
- ❑ http1.0: RFC 1945
- ❑ http1.1: RFC 2068



## O protocolo HTTP

http: *serviço de transporte TCP*:

- ❑ cliente inicia uma conexão TCP (cria soquete) com o servidor na porta 80
- ❑ o servidor aceita a conexão TCP do cliente
- ❑ mensagens http (mensagens do protocolo de camada de aplicação) são trocadas entre o navegador (cliente http) e o servidor Web (servidor http)
- ❑ A conexão TCP é fechada

O http é um protocolo "sem estado" (*stateless*)

- ❑ o servidor não mantém informação sobre requisições passadas do cliente

Protocolos que mantêm informações de estado são complexos!

- ❑ a história passada (estado) deve ser mantida
- ❑ 'quedas' do cliente/servidor podem tornar as visões do 'estado' inconsistentes

## Conexões HTTP

**Não-persistente**

- ❑ http/1.0: servidor analisa pedido, envia resposta e fecha a conexão TCP
- ❑ 2 RTTs para obter um objeto
  - o conexão TCP
  - o solicitação e transferência do objeto
- ❑ cada transferência sofre com a taxa de envio inicialmente lenta do TCP (*slow-start*)
- ❑ muitos navegadores abrem várias conexões paralelas

**Persistente**

- ❑ modo *default* para http/1.1
- ❑ podem ser enviados vários objetos na mesma conexão TCP
- ❑ o cliente envia pedido para todos os objetos referenciados tão logo ele recebe a página HTML básica.
- ❑ poucos RTTs, menos partidas lentas (*slow start*).

## HTTP não persistente

Usuário entra com a URL: www.someSchool.edu/someDepartment/home.index (contém referência a 10 imagens jpeg)

- 1a. o cliente http inicia conexão TCP com o servidor http (processo) em www.someSchool.edu, porta.
- 1b. o servidor http no hospedeiro www.someSchool.edu esperando pela conexão TCP na porta 80, "aceita" a conexão, notificando o cliente
2. o cliente http envia uma *mensagem de requisição* http (contendo a URL) através de soquete da conexão TCP. A mensagem indica que o cliente quer o objeto someDepartment/home.index
3. o servidor http recebe a mensagem de requisição, forma uma *mensagem de resposta* contendo o objeto solicitado (someDepartment/home.index) e envia a mensagem através do soquete

tempo

## HTTP não persistente (cont.)

4. servidor http fecha conexão TCP.
5. cliente http recebe a mensagem de resposta contendo o arquivo html, apresenta o conteúdo html. Analisando o arquivo html encontra 10 objetos jpeg referenciados
6. Passos 1-5 são repetidos para cada um dos 10 objetos jpeg.

tempo

## HTTP Persistente

### HTTP não persistente:

- ❑ Requer 2 RTTs por objeto
- ❑ SO deve trabalhar e alocar recursos para cada conexão TCP
- ❑ Mas, o navegador geralmente abre conexões TCP paralelas para buscar os objetos referenciados

### HTTP Persistente

- ❑ O servidor deixa a conexão aberta após enviar a resposta
- ❑ Mensagens HTTP subsequentes entre o cliente e o servidor são enviadas pela conexão

### Persistente sem pipelining:

- ❑ Cliente faz uma nova requisição apenas quando a resposta anterior for recebida
- ❑ Um RTT para cada objeto referenciado

### Persistente com pipelining:

- ❑ default no HTTP/1.1
- ❑ Cliente envia requisições assim que encontra um objeto referenciado
- ❑ pode chegar a um RTT para todos os objetos referenciados

## Formato das Mensagens

- ❑ dois tipos de mensagens HTTP: *requisição*, *resposta*

### mensagem de requisição http:

- o ASCII (formato legível para humanos)

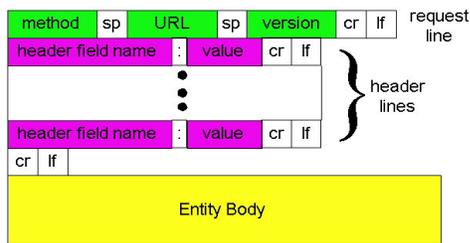
```
GET /somedir/page.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: fr
```

linhas de pedido (comandos GET, POST, HEAD)

linhas de cabeçalho

Carriage return line feed (extra carriage return, line feed) indica fim da mensagem

## Requisição HTTP: formato geral



## Carregando entrada de formulários

### Método Post :

- ❑ Páginas Web geralmente incluem entradas para formulários
- ❑ Entrada é carregada para o servidor no corpo da entidade

### URL :

- ❑ Usa o método GET
- ❑ Entrada é carregada em campo da URL da linha de requisição:

[www.somesite.com/animalsearch?monkeys&banana](http://www.somesite.com/animalsearch?monkeys&banana)

## Métodos

### HTTP/1.0

- ❑ GET
- ❑ POST
- ❑ HEAD
  - o Pede ao servidor para não incluir o objeto solicitado na resposta

### HTTP/1.1

- ❑ GET, POST, HEAD
- ❑ PUT
  - o Carrega arquivo no corpo da entidade para o caminho especificado na URL
- ❑ DELETE
  - o Apaga o arquivo especificado na URL

## Resposta HTTP

```
HTTP/1.0 200 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
```

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

linhas de cabeçalho

dados, e.x., arquivo html

```
data data data data data ...
```

## Códigos de status das respostas

### 200 OK

- o request succeeded, requested object later in this message

### 301 Moved Permanently

- o requested object moved, new location specified later in this message (Location:)

### 400 Bad Request

- o request message not understood by server

### 404 Not Found

- o requested document not found on this server

### 505 HTTP Version Not Supported

## HTTP Cliente: faça você mesmo!

### 1. Telnet para um servidor Web:

```
telnet www.eurecom.fr 80
```

Abre conexão TCP para a porta 80 (porta default do servidor http) em www.eurecom.fr. Qualquer coisa digitada é enviada para a porta 80 em www.eurecom.fr

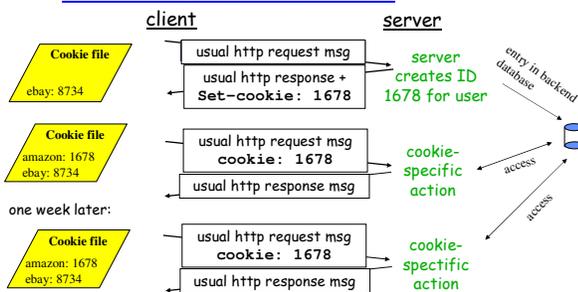
### 2. Digite um pedido GET http:

```
GET /~ross/index.html HTTP/1.0
```

Digitando isto (tecle carriage return duas vezes), você envia este pedido HTTP GET mínimo (mas completo) ao servidor http

### 3. Examine a mensagem de resposta enviada pelo servidor http!

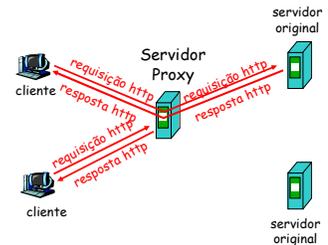
## Cookies: Mantendo "estado"



## Web Caches (servidores proxy)

**Objetivo:** satisfazer a uma requisição do cliente sem envolver o servidor Web originador da informação

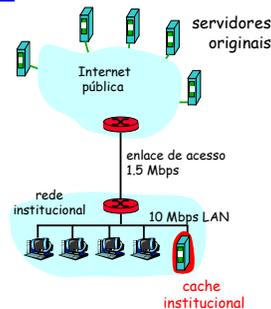
- o usuário configura o navegador: acesso à Web é feito através de um proxy
- o cliente envia todos os pedidos http para o web cache
  - o se o objeto existe no web cache: web cache retorna o objeto
  - o senão web cache solicita o objeto do servidor original, então envia o objeto ao cliente.



## Porque Web Caching?

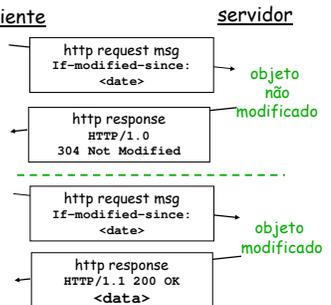
Assuma: cache "perto" do cliente (ex., na mesma rede)

- o menor tempo de resposta
- o reduz o tráfego para servidores distantes
  - o links externos podem ser caros e facilmente congestionáveis



## GET Condicional: cache do lado cliente

- o **Razão:** não enviar objeto se o cliente já possui armazenado uma versão atualizada.
- o cliente: especifica a data da versão armazenada no pedido HTTP
- o servidor: resposta não contém objeto se a cópia é atualizada:



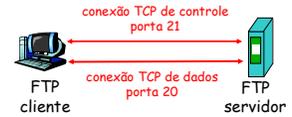
## ftp: o protocolo de transferência de arquivos



- ❑ transferência de arquivos de e para o computador remoto
- ❑ modelo cliente servidor
  - o *cliente*: lado que inicia a transferência (seja de ou para o lado remoto)
  - o *servidor*: hospedeiro remoto
- ❑ ftp: RFC 959
- ❑ servidor ftp: porta 21

## ftp: conexões de controle e dados separadas

- ❑ cliente ftp contata o servidor ftp na porta 21, especificando o TCP como o protocolo de transporte
- ❑ duas conexões TCP paralelas são abertas:
  - o *controle*: troca de comandos e respostas entre cliente e servidor. "controle fora da banda"
  - o *dados*: dados do arquivo do/para o servidor
- ❑ servidor ftp mantém o "estado": diretório corrente, autenticação anterior



## comandos e respostas ftp

### Exemplos de comandos:

- ❑ envie um texto ASCII sobre canal de controle
- ❑ USER *username*
- ❑ PASS *password*
- ❑ LIST retorna listagem de arquivos no diretório atual
- ❑ RETR *filename* recupera (obtem) o arquivo
- ❑ STOR *filename* armazena o arquivo no hospedeiro remoto

### Exemplos de códigos de retorno

- ❑ código de status e frase (como no http)
- ❑ 331 Username OK, password required
- ❑ 125 data connection already open; transfer starting
- ❑ 425 Can't open data connection
- ❑ 452 Error writing file

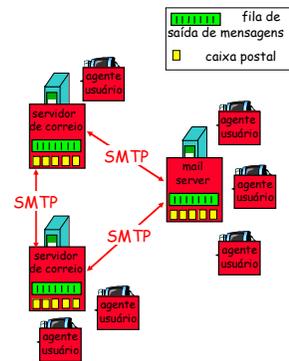
## Correio Eletrônico

### Três componentes principais:

- ❑ agentes de usuário
- ❑ servidores de correio
- ❑ simple mail transfer protocol: smtp

### Agente de usuário

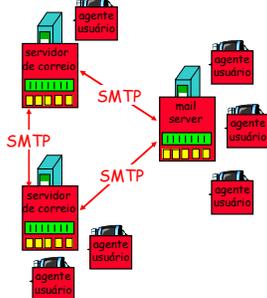
- ❑ "leitor de correio"
- ❑ composição, edição, leitura de mensagens de correio
- ❑ ex., Eudora, Outlook, elm, Netscape Messenger
- ❑ mensagens de entrada e de saída são armazenadas no servidor



## Correio eletrônico: servidores de correio

### Servidores de Correio

- ❑ *caixa postal* contém mensagens que chegaram (ainda não lidas) para o usuário
- ❑ *fila de mensagens* contém as mensagens de correio a serem enviadas
- ❑ *protocolo smtp* permite aos servidores de correio trocarem mensagens entre eles
  - o cliente: servidor de correio que envia
  - o "servidor": servidor de correio que recebe



## Correio Eletrônico: smtp [RFC 821]

- ❑ usa TCP para transferência confiável de mensagens de correio do cliente ao servidor, porta 25
- ❑ transferência direta: servidor que envia para o servidor que recebe
- ❑ três fases de transferência
  - o *handshaking* (apresentação)
  - o transferência de mensagens
  - o fechamento
- ❑ interação comando/resposta
  - o *comandos*: texto ASCII
  - o *resposta*: código de status e frase
- ❑ mensagens devem ser formatadas em código ASCII de 7 bits

## Exemplo de interação SMTP

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

## Tente o SMTP você mesmo:

- ❑ telnet <nome do servidor> 25
  - ❑ veja resposta 220 do servidor
  - ❑ envie comandos HELO, MAIL FROM, RCPT TO, DATA, QUIT
- a seqüência acima permite enviar um comando sem usar o agente de usuário do remetente

## SMTP: palavras finais

- ❑ SMTP usa conexões persistentes
  - ❑ SMTP exige que as mensagens (cabeçalho e corpo) estejam em ASCII de 7 bits
  - ❑ algumas seqüências de caracteres não são permitidas nas mensagens (ex., CR LF . CR LF). Assim mensagens genéricas têm que ser codificadas (usualmente em "base-64" ou "quoted printable")
  - ❑ Servidor SMTP usa CR LF . CR LF para indicar o final da mensagem
- Comparação com http:**
- ❑ http: pull
  - ❑ email: push
  - ❑ ambos usam comandos e respostas em ASCII, interação comando / resposta e códigos de status
  - ❑ http: cada objeto encapsulado na sua própria mensagem de resposta
  - ❑ smtp: múltiplos objetos são enviados numa mensagem multiparte

## Formato das Mensagens

smtp: protocolo para trocar mensagens de e-mail

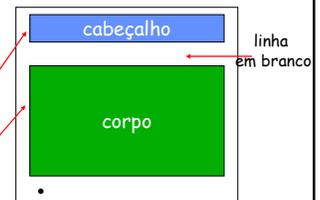
RFC 822: padrão para mensagens do tipo texto:

- ❑ linhas de cabeçalho, e.g.,

- o To:
- o From:
- o Subject:

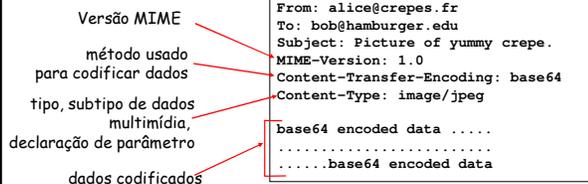
*diferente dos comandos SMTP:*

- ❑ corpo
  - o a "mensagem", ASCII somente com caracteres



## Formato das Mensagens: extensões multimídia

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



## Tipos MIME

Content-Type: tipo/subtipo; parâmetros

### Text

- ❑ exemplo de subtipos: plain, html

### Image

- ❑ exemplo de subtipos: jpeg, gif

### Audio

- ❑ exemplo de subtipos: basic (codificado 8-bit  $\mu$ -law), 32kadtcm (codificação 32 kbps)

### Video

- ❑ exemplo de subtipos: mpeg, quicktime

### Application

- ❑ outros dados que devem ser processados pelo leitor antes de serem apresentados "visualmente"
- ❑ exemplo de subtipos: msword, octet-stream

## Tipo Multiparte

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=98766789

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

Dear Bob,
Please find a picture of a crepe.
--98766789
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

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

## Protocolos de acesso ao correio



- ❑ SMTP: entrega e armazenamento no servidor do destino
- ❑ Protocolo de acesso: recuperação de mensagens do servidor
  - o POP: Post Office Protocol [RFC 1939]
    - autorização (agente <->servidor) e descarga (download)
  - o IMAP: Internet Mail Access Protocol [RFC 1730]
    - mais recursos (mais complexo)
    - manipulação de mensagens armazenadas no servidor
  - o HTTP: Hotmail, Yahoo! Mail, etc.

## Protocolo POP3

### fase de autorização

- ❑ comandos do cliente:
  - o **user**: declara nome do usuário
  - o **pass**: password
- ❑ respostas do servidor
  - o +OK
  - o -ERR

### fase de transação, cliente:

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

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

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

## DNS: Domain Name System

### Pessoas: muitos identificadores:

- o RG, nome, passaporte

### hospedeiros, roteadores Internet:

- o endereços IP (32 bit) - usados para endereçar datagramas
- o "nome", ex., gaia.cs.umass.edu - usados por humanos

### Q: relacionar endereços IP com nomes?

### Domain Name System:

- ❑ **base de dados distribuída** implementada numa hierarquia de muitos **servidores de nomes**
- ❑ **protocolo de camada de aplicação** hospedeiro e roteadores se comunicam com servidores de nomes para **resolver** nomes (tradução nome/endereço)
  - o nota: função interna da Internet, implementada como protocolo da camada de aplicação
  - o complexidade na "borda" da rede

## DNS: Domain Name System

### Serviços do DNS

- ❑ Tradução de nome de hospedeiro para endereço IP
- ❑ Apelidos (*aliasing*) de hospedeiros
  - o Nomes canônicos e apelidos
- ❑ Apelido de servidor de correio eletrônico
- ❑ Distribuição de carga
  - o Servidores Web replicados: conjunto de endereços IP para um nome canônico

## Servidores de Nomes DNS

### Porque não centralizar o DNS?

- ❑ nenhum servidor tem todos os mapeamentos de nomes para endereços IP
- ❑ ponto único de falha
- ❑ volume de tráfego
- ❑ base de dados distante
- ❑ manutenção

### servidores de nomes locais:

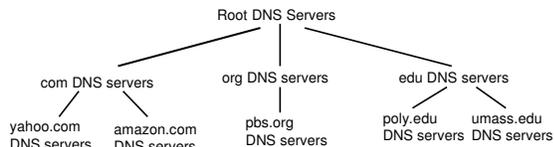
- o cada ISP ou empresa tem um **servidor de nomes local (default)**
- o Consultas dos computadores locais ao DNS vão primeiro para o servidor de nomes local

### Não cresce junto com a rede! (não é "escalável")

### servidor de nomes com autoridade:

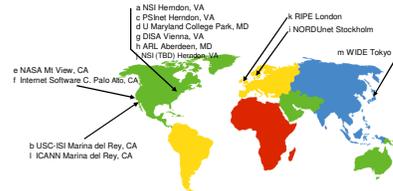
- o para um computador: armazena o nome e o endereço IP daquele computador
- o pode mapear nomes para endereços para aquele nome de computador

## Base de dados distribuída e hierárquica



## DNS: Servidores de Nomes Raiz

- são contactados pelos servidores de nomes locais que não podem resolver um nome
- servidores de nomes raiz::
  - o buscam servidores de nomes com autoridade se o mapeamento do nome não for conhecido
  - o conseguem o mapeamento
  - o retornam o mapeamento para o servidor de nomes local



existem 13 servidores de nomes raiz no mundo

## Servidores TLD e servidores com autoridade

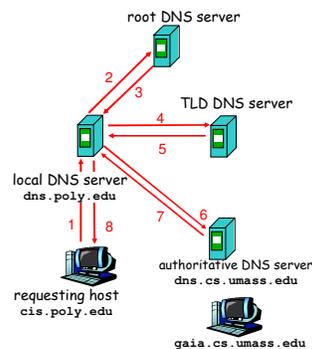
- **Top-level domain (TLD) servers:** responsáveis pelos domínios com, org, net, edu, etc, e todos os domínios de alto nível de domínios de países (br.uk, fr, ca, jp).
- **Servidores DNS com autoridade (Authoritative DNS servers):** servidores DNS das organizações. Fazem o mapeamento de nomes de hospedeiros para para endereços IP para servidores das organizações (ex., Web and mail).
  - o Podem ser mantidos pela organização ou por um provedor de serviços

## Servidor de Nomes Local

- A rigor não pertence à hierarquia
- Cada ISP (ISP residencial, companhia, universidade) tem um.
  - o Também chamado de "servidor de nomes default"
- Quando um hospedeiro faz uma consulta ao DNS, ela é enviada para o seu servidor local de DNS
  - o Funciona como um proxy, re-encaminha a consulta na hierarquia

## Exemplo

- Hospedeiro em cis.poly.edu quer o IP de gaia.cs.umass.edu



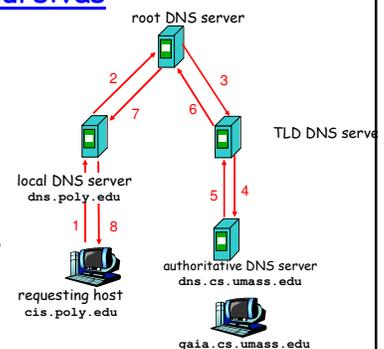
## Consultas Recursivas

### consulta recursiva :

- deixa o servidor de nomes contactado com a responsabilidade de resolver o nome
- carga elevada?

### consulta iterativa:

- Servidor contactado responde com o nome do servidor a contactar
- "Eu não conheço este nome, mas pergunte a este servidor"



## DNS: armazenando e atualizando registros

- ❑ quando um servidor de nomes aprende um mapeamento, ele o armazena em um registro do tipo *cache*
  - o registros do cache tornam-se obsoletos (desaparecem) depois de um certo tempo
- ❑ mecanismos de atualização e notificação estão sendo projetados pelo IETF
  - o RFC 2136
  - o <http://www.ietf.org/html.charters/dnsind-charter.html>

## registros do DNS

DNS: BD distribuída que armazena registros de recursos (RR)

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

- ❑ Type=A
  - o name é o nome do hospedeiro
  - o value é o endereço IP
- ❑ Type=CNAME
  - o name é um "apelido" para algum nome "canônico" (o nome real)
  - o value é o nome canônico
- ❑ Type=NS
  - o name é um domínio (ex. foo.com)
  - o value é o nome do servidor de nomes com autoridade para este domínio
- ❑ Type=MX
  - o name é o nome canônico do servidor de correio associado com name

## DNS: protocolo e mensagens

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

cabeçalho da msg

- ❑ **identificação**: número de 16 bit para consulta, resposta usa esse mesmo número
- ❑ **flags**:
  - o consulta ou resposta
  - o recursão desejada
  - o recursão disponível
  - o resposta com autoridade

identification	flags
number of questions	number of answer RRs
number of authority RRs	number of additional RRs
questions (variable number of questions)	
answers (variable number of resource records)	
authority (variable number of resource records)	
additional information (variable number of resource records)	

12 bytes

## DNS: protocolo e mensagens



## Inserção de registros no DNS

- ❑ Exemplo: criada a empresa "Network Utopia"
- ❑ Registro do nome `networkutopia.com` em um **registro** (e.g., Network Solutions)
  - o Deve-se fornecer ao registro os nomes e endereços IP dos servidores de nomes com autoridade (primários e secundários)
  - o O registro insere dois RRs no servidor TLD com:
 

```
(networkutopia.com, dns1.networkutopia.com, NS)
(dns1.networkutopia.com, 212.212.212.1, A)
```
- ❑ Colocar no servidor com autoridade um registro Tipo A para `www.networkutopia.com` e um Tipo MX para `networkutopia.com`

## Compartilhamento de arquivos P2P

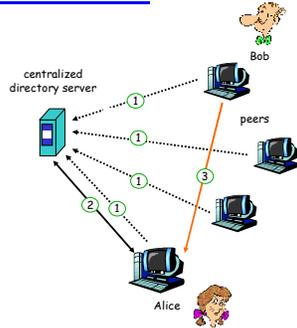
### Exemplo

- ❑ Alice roda uma aplicação cliente P2P no seu *notebook*
  - ❑ Intermittentemente conecta-se à Internet; recebe um novo endereço IP para cada conexão
  - ❑ Procura por "Hey Jude"
  - ❑ A aplicação mostra outros pares que têm uma cópia de "Hey Jude".
- ❑ Alice escolhe um dos pares, Bob.
  - ❑ Arquivo é copiado do PC de Bob para o *notebook* de Alice: HTTP
  - ❑ Enquanto Alice descarrega (*download*), outros usuários carregam (*upload*) a partir do micro de Alice.
  - ❑ O par de Alice é tanto um cliente Web como um servidor Web transiente.
- Todos os pares são servidores = muito escalável!

## P2P: diretório centralizado

projeto original do Napster

- 1) quando um par se conecta, ele informa ao servidor central:
  - o IP
  - o conteúdo
- 2) Alice consulta por "Hey Jude"
- 3) Alice requisita o arquivo de Bob



## P2P: problemas com diretórios centralizados

- ❑ Ponto único de falha
- ❑ Gargalo de desempenho
- ❑ Violação dos direitos de cópia (Copyright)

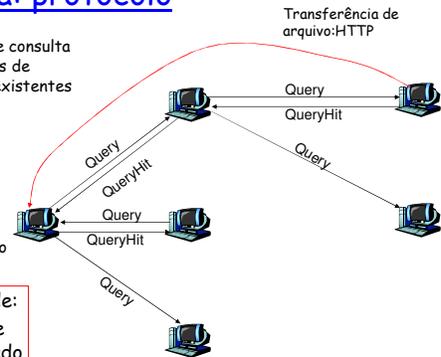
transferência de arquivos é descentralizada, mas a localização de conteúdo é muito centralizada

## Inundação de consultas: Gnutella

- ❑ totalmente distribuída
    - o não há servidor central
  - ❑ protocolo de domínio público
  - ❑ muitos clientes Gnutella implementam o protocolo
- rede de sobreposição (overlay network): grafo**
- ❑ vértice entre os pares X e Y se houver uma conexão TCP
  - ❑ todos os pares ativos e vértices formam a rede de sobreposição (overlay net)
  - ❑ vértice não é um enlace físico
  - ❑ um dado par estará geralmente conectado a < 10 vizinhos de overlay

## Gnutella: protocolo

- ❑ mensagem de consulta enviada através de conexões TCP existentes
- ❑ pares re-encaminham mensagens de consulta
- ❑ resposta à consulta enviada pelo caminho reverso



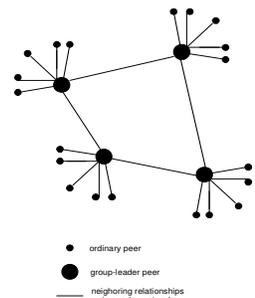
Escalabilidade: Inundação de escopo limitado

## Gnutella: Par juntando-se à rede

1. Um par X juntando-se à rede X deve encontrar outro par na rede in Gnutella : use lista de pares candidatos
2. X seqüencialmente tenta estabelecer uma conexão TCP com pares na lista até conseguir com algum Y
3. X envia uma mensagem Ping a Y; Y re-encaminha essa mensagem.
4. Todos os pares que receberem a mensagem Ping respondem com uma mensagem Pong
5. X recebe várias mensagens Pong. Ele podem então estabelecer conexões TCP adicionais

## Explorando a heterogeneidade: KaZaA

- ❑ Cada par ou é um líder de grupo ou é associado a um grupo de líder.
  - o conexão TCP entre o par e o líder do grupo.
  - o conexões TCP entre alguns líderes de grupos.
- ❑ O líder de um grupo acompanha o conteúdo de todos os seus filhos.



## KaZaA: Consulta

- ❑ Cada arquivo tem um código de *hash* e um descritor
- ❑ Cliente envia consulta ao seu líder de grupo
- ❑ Líder de grupo responde com "casamentos" (*matches*):
  - o Para cada casamento: metadados, *hash*, IP
- ❑ Se o líder do grupo reencaminhar consultas para outros líderes, eles respondam com os casamentos
- ❑ O cliente seleciona então os arquivos para descarregar
  - o requisições HTTP usando o identificador de *hash* enviado pelos pares que têm o arquivo desejado

## Kazaa: truques

- ❑ Limitações no número de cargas (*uploads*) simultâneas.
- ❑ Fila de requisições
- ❑ Prioridades de incentivo
- ❑ Descarga paralela

## Capítulo 2: Sumário

- ❑ Arquiteturas de aplicações
  - o Cliente-servidor
  - o P2P
  - o híbrida
- ❑ Requisitos de serviços de aplicação:
  - o confiabilidade, largura de banda, retardo
- ❑ Modelo de serviços de transporte da Internet
  - o Orientado a conexões, confiável: TCP
  - o Não confiável, datagramas: UDP
- ❑ Protocolos específicos:
  - o HTTP
  - o FTP
  - o SMTP, POP, IMAP
  - o DNS

## Capítulo 2: Sumário

*Mais importante: aprendizado sobre protocolos*

- ❑ Troca típica de mensagens do tipo requisição /resposta:
  - o cliente requisita informação ou serviço
  - o Servidor responde com dados ou código de status
- ❑ Formatos de mensagens:
  - o cabeçalhos: campos contendo informação sobre os dados
  - o dados: informação sendo comunicada
- ❑ Mensagens de controle vs. mensagens de dados
  - o dentro / fora da banda (*in-band, out-of-band*)
- ❑ centralizado vs. descentralizado
- ❑ com estado vs. sem estado (*stateless vs. stateful*)
- ❑ Transferência de mensagens confiável vs. não confiável
- ❑ "complexidade nas bordas da rede"