

# Aula 6 - HTTP (Cookies, Web Caches), FTP, E-mail, P2P

Diego Passos

Universidade Federal Fluminense

Redes de Computadores

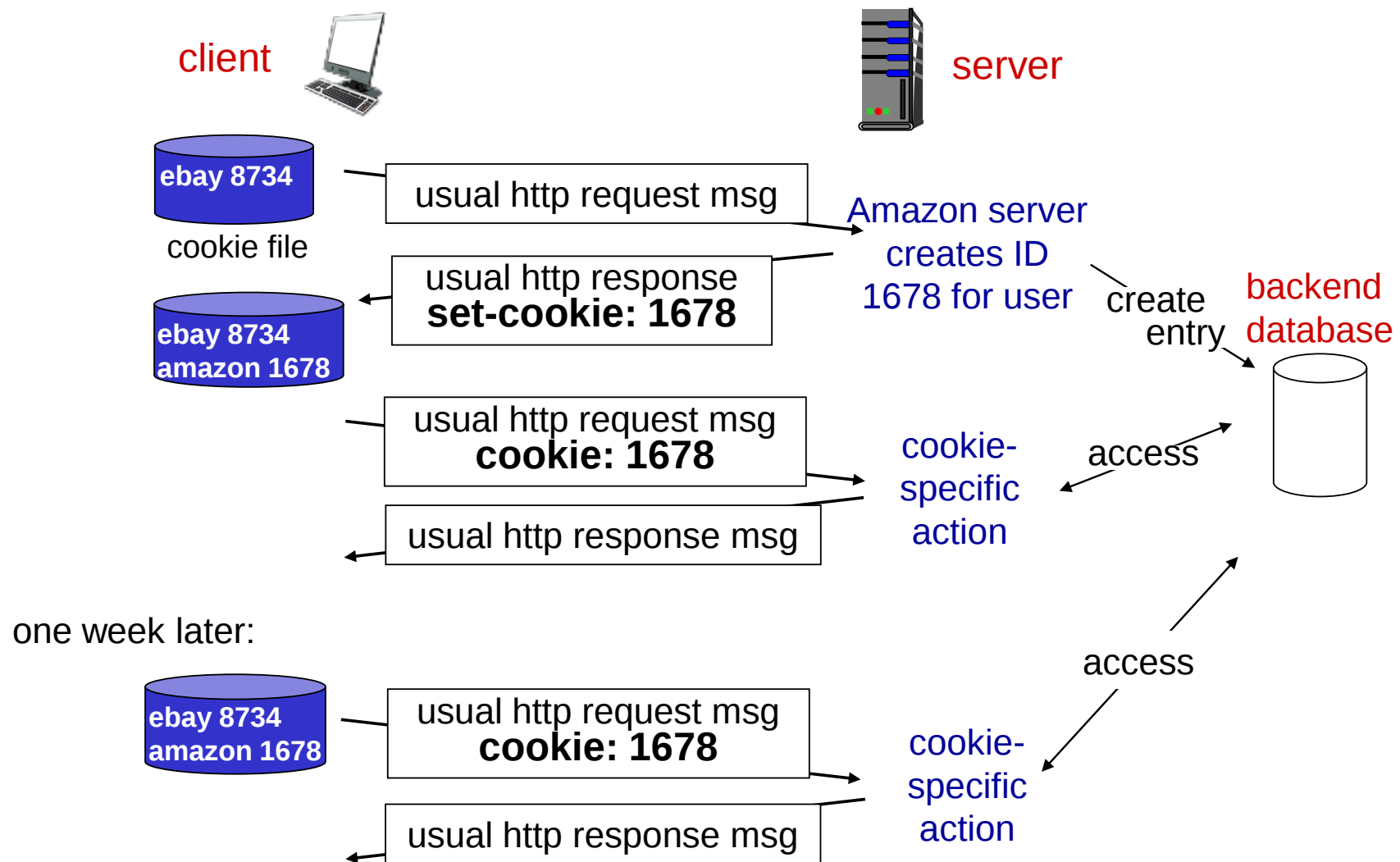
Material adaptado a partir dos slides  
originais de J.F Kurose and K.W. Ross.

# Cookies

# Estado do Usuário no Servidor: Cookies

- Muitos sites utilizam cookies.
- **Quatro componentes:**
  1. Entrada de cabeçalho relativa aos cookies na mensagem de resposta HTTP.
  2. Entrada de cabeçalho relativa aos cookies na mensagem de requisição HTTP.
  3. Arquivo de cookies mantido no *host* do usuário e gerenciado pelo *browser*.
  4. Base de dados no *backend* do site.
- **Exemplo:**
  - Susan sempre acessa a Internet de seu PC.
  - Visita site específico de comércio eletrônico pela primeira vez.
  - Quando requisição HTTP inicial chega ao servidor, *site* cria:
    - ID único.
    - Entrada na base de dados associada ao ID.

# Cookies: Mantendo Estado



# Cookies (Mais)

- **Para que cookies podem ser utilizados:**

- Autorização.
- “Carrinhos de compra”.
- Recomendações.
- Estado da sessão do usuário (e.g., webmail).

## Nota: cookies e privacidade

- Cookies permitem que sites aprendam muito sobre você.
- Você pode fornecer nome e e-mail para sites.

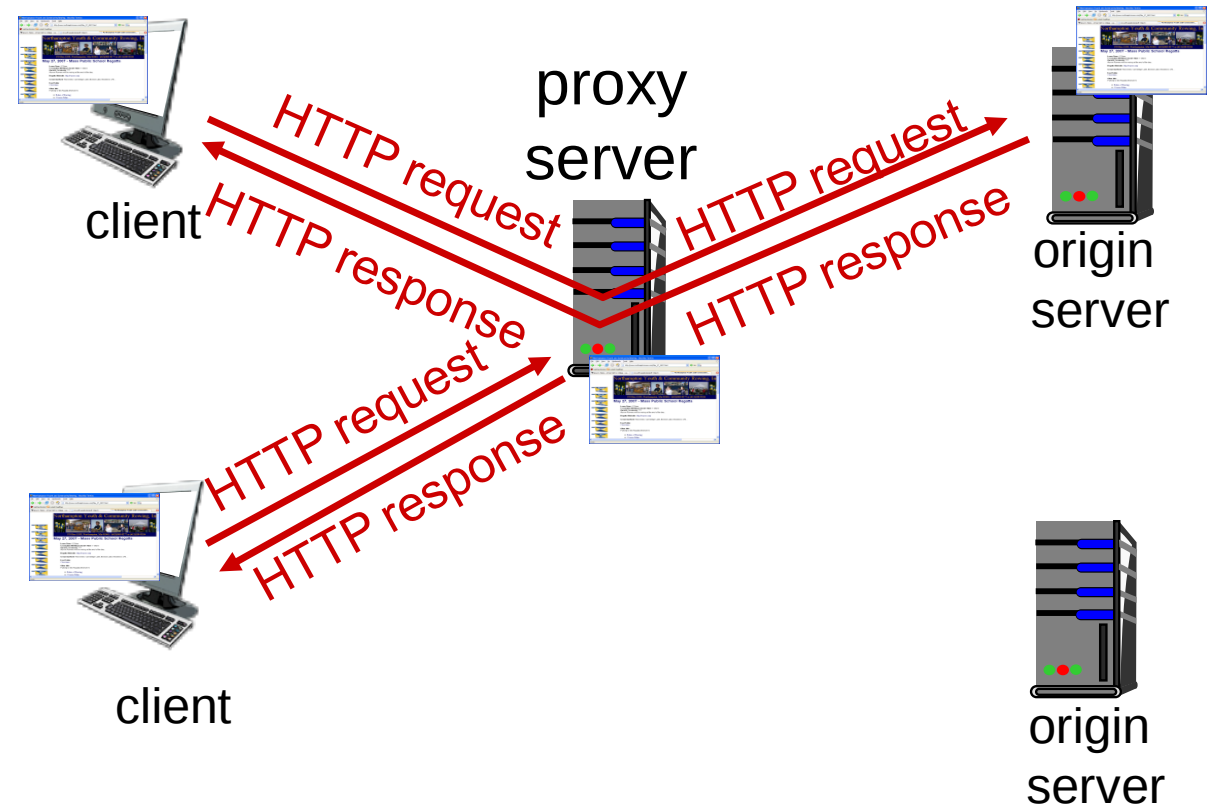
- **Como manter “estado”:**

- Sistemas finais mantêm estado no transmissor, receptor ao longo de várias transações.
- Cookies: mensagens HTTP carregam estado.

# Web Caches

# Web Caches (Servidores Proxy)

- **Objetivo:** satisfazer requisição do cliente sem envolver servidor original do conteúdo.
- Usuário configura *browser*: acessos web via cache.
- Browser envia todas as requisições HTTP ao cache.
  - Objetos em cache: retornados imediatamente.
  - Caso contrário, cache requisita objeto do servidor, retorna objeto ao cliente.



# Mais sobre Web Caches

- Cache age tanto como servidor, quanto como cliente.
  - Servidor para a requisição original.
  - Cliente para o servidor original do conteúdo.
- Tipicamente, cache é instalado na rede de acesso ou pelo ISP.
- **Por que usar um web cache?**
  - Reduzir o **tempo de resposta** para o cliente.
  - Reduzir o **tráfego no enlace de acesso** da instituição.
  - Internet densamente populada por cache: permite que provedores de conteúdo “pobres” entregue conteúdo de forma efetiva.
    - P2P também.



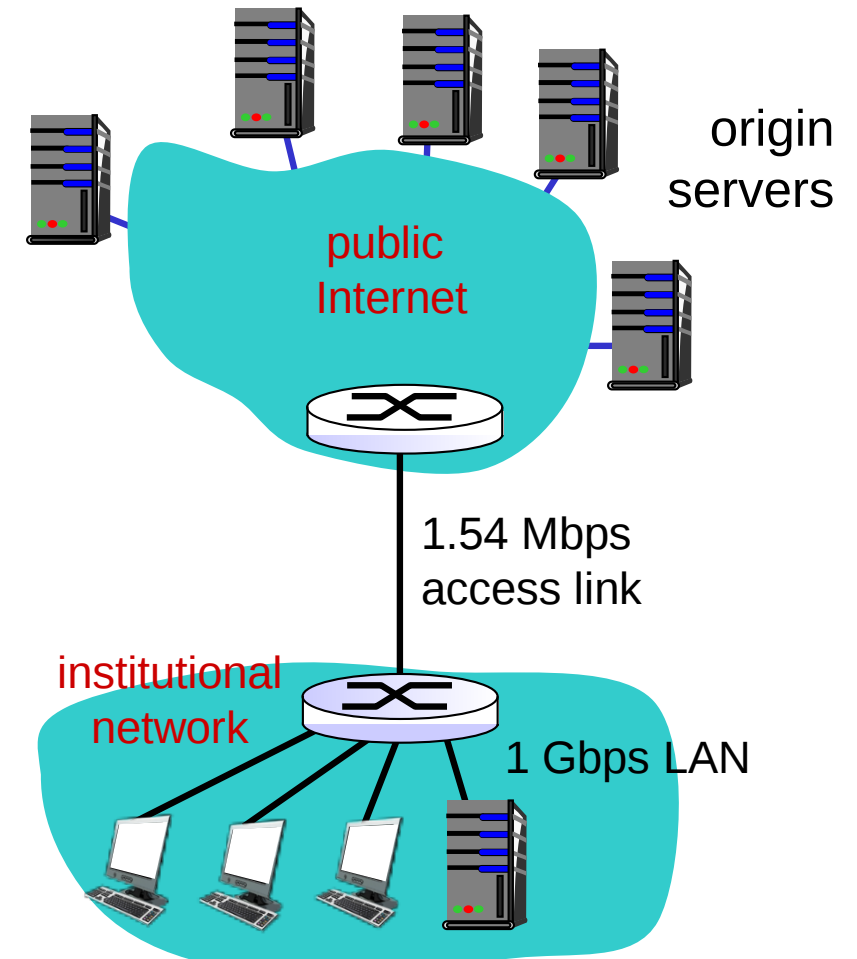
# Web Cache: Exemplo

- **Hipóteses:**

- Tamanho médio dos objetos: 100 kb.
- Taxa média de requisição do *browsers*: 15/s.
  - Taxa média para os *browsers*: 1,5 Mb/s.
- RTT do roteador de borda para qualquer servidor: 2s.
- Capacidade do enlace de acesso: 1,54 Mb/s.

- **Consequências:**

- Utilização da LAN: 0,15%.
- Utilização do enlace de acesso: **99%**!
- Atraso total = Atraso da Internet + atraso do roteador de borda + atraso da LAN.
  - = 2s + minutos +  $\mu s$



# Web Cache: Exemplo (Enlace de Maior Capacidade)

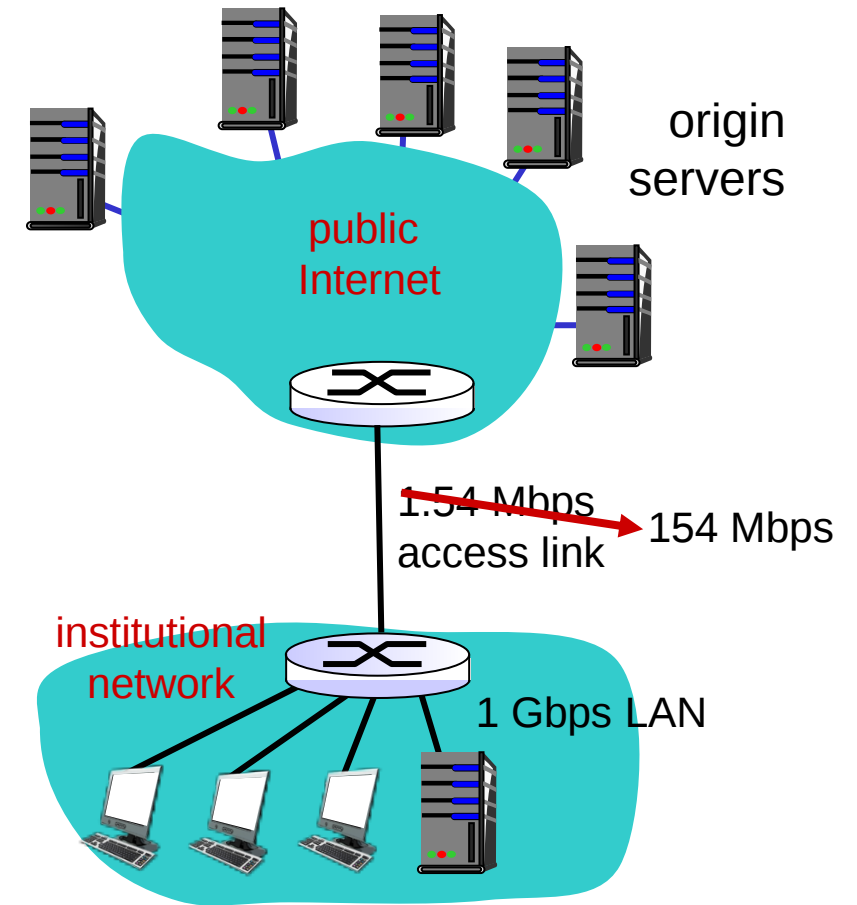
- **Hipóteses:**

- Tamanho médio dos objetos: 100 kb.
- Taxa média de requisição do *browsers*: 15/s.
  - Taxa média para os *browsers*: 1,5 Mb/s.
- RTT do roteador de borda para qualquer servidor: 2s.
- Capacidade do enlace de acesso: ~~1,54~~ 154 Mb/s.

- **Consequências:**

- Utilização da LAN: 0,15%.
- Utilização do enlace de acesso: ~~99%~~ 0,99%.
- Atraso total = Atraso da Internet + atraso do roteador de borda + atraso da LAN.
  - = 2s + ~~minutos~~ ms +  $\mu$ s

- **Custo:** aumento na capacidade do enlace de acesso (não é barato!).



# Web Cache: Exemplo (Solução com Web Cache, I)

- **Hipóteses:**

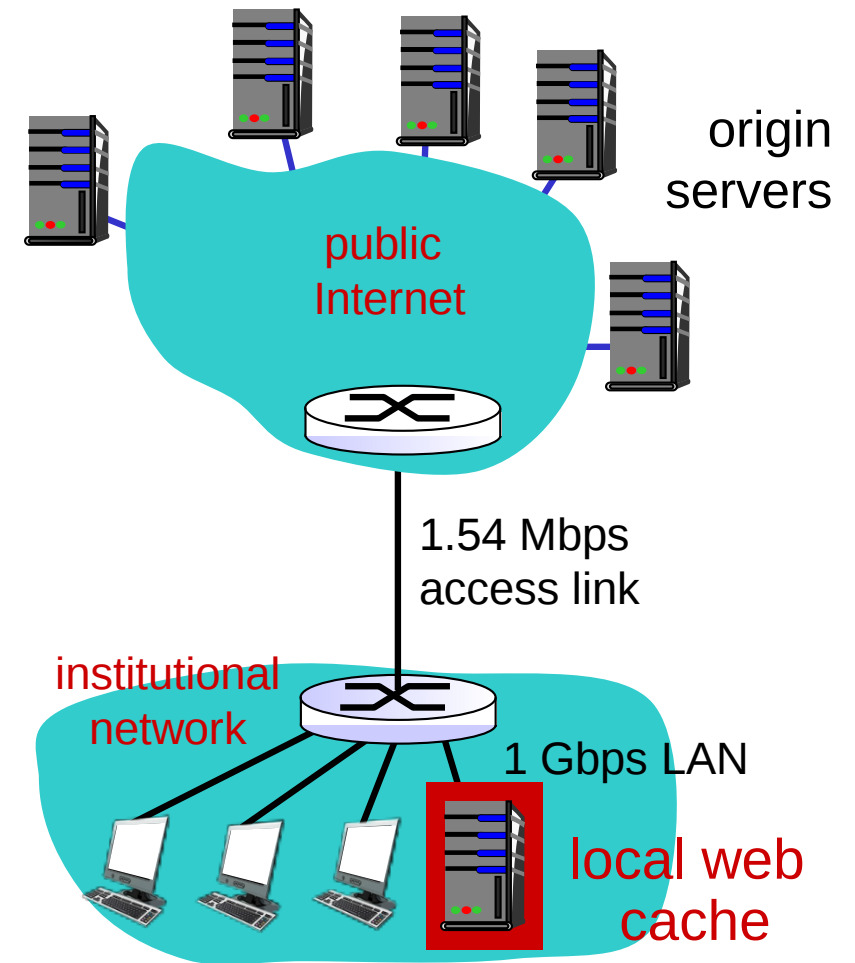
- Tamanho médio dos objetos: 100 kb.
- Taxa média de requisição do *browsers*: 15/s.
  - Taxa média para os *browsers*: 1,5 Mb/s.
- RTT do roteador de borda para qualquer servidor: 2s.
- Capacidade do enlace de acesso: 1,54 Mb/s.

- **Consequências:**

- Utilização da LAN: 0,15%.
- Utilização do enlace de acesso: ?
- Atraso total = ?

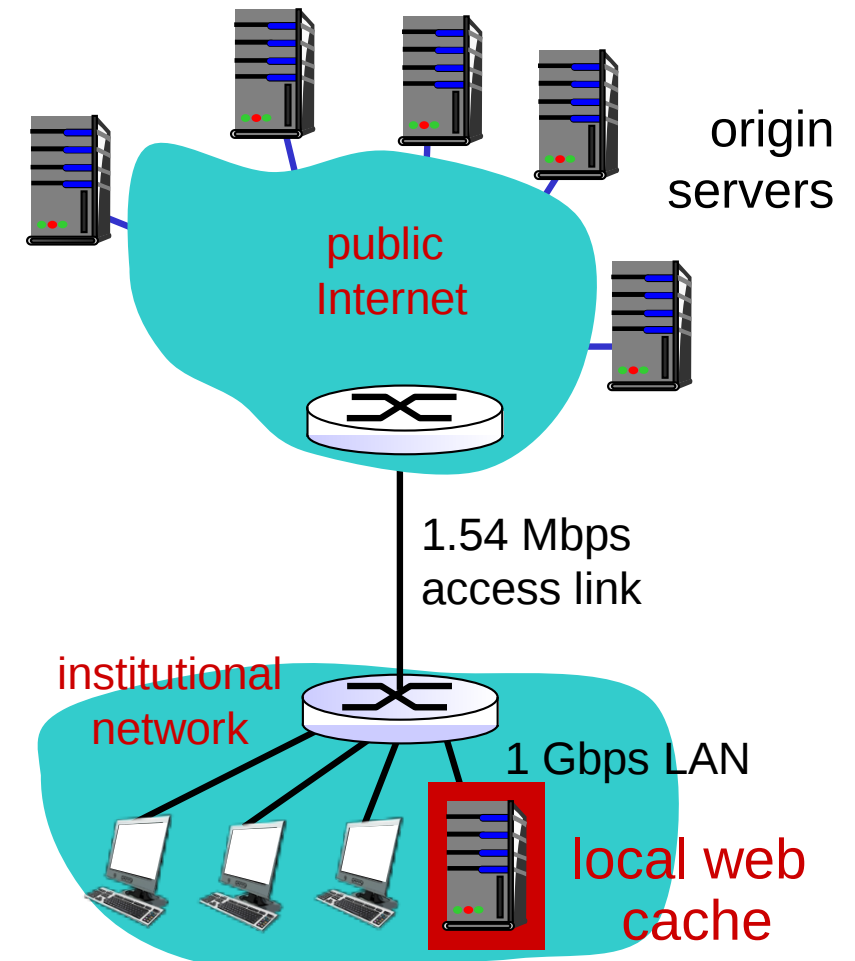
**Como calcular utilização do  
enlace de acesso, atraso?**

- **Custo:** web cache (barato!).



# Web Cache: Exemplo (Solução com Web Cache, II)

- **Calculando a utilização do enlace de acesso e atraso com cache.**
  - Assuma que a taxa de acerto (*hit rate*) é 0,4.
    - 40% das requisições satisfeitas no cache, 60% precisam ir à origem.
  - Utilização do enlace de acesso:
    - 60% das requisições usam o enlace.
    - $0,6 \times 1,5 = 0,9$  Mb/s de tráfego.
    - Utilização:  $\frac{0,9}{1,54} = 0,58$ .
  - Atraso total:
    - $0,6 \times (\text{atraso da origem}) + 0,4 \times (\text{atraso do cache})$ .
    - $0,6 \times 2,01 + 0,4 \times (ms) \approx 1,2$  s.
    - Menos que com enlace de 154 Mb/s (e mais barato!)



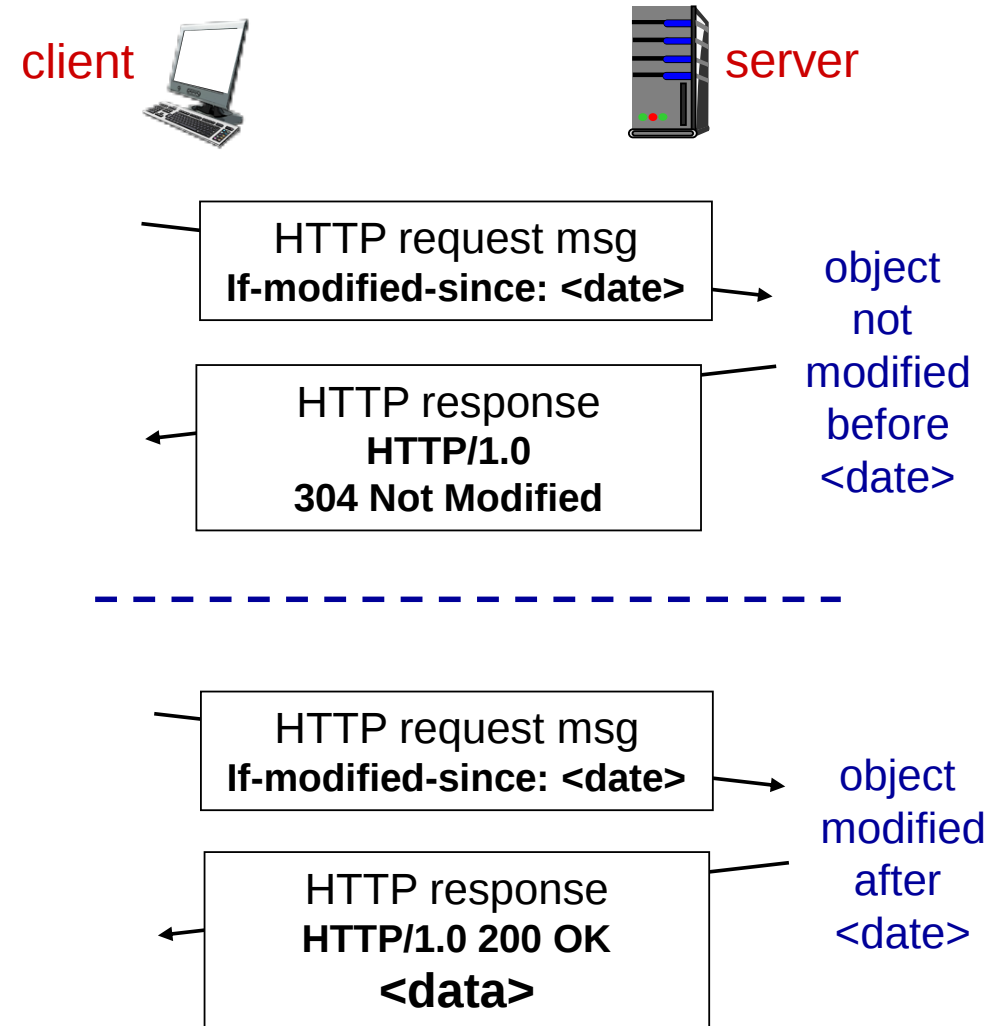
# GET Condicional

- **Objetivo:** não enviar objeto se cache tem versão atualizada.
  - Não há atraso de envio do objeto.
  - Reduz utilização do enlace de acesso.
- Cache: especifica data da cópia local na requisição HTTP.

If-modified-since: <data>

- Servidor: resposta não contém objeto se cópia está atualizada.

HTTP/1.0 304 Not Modified



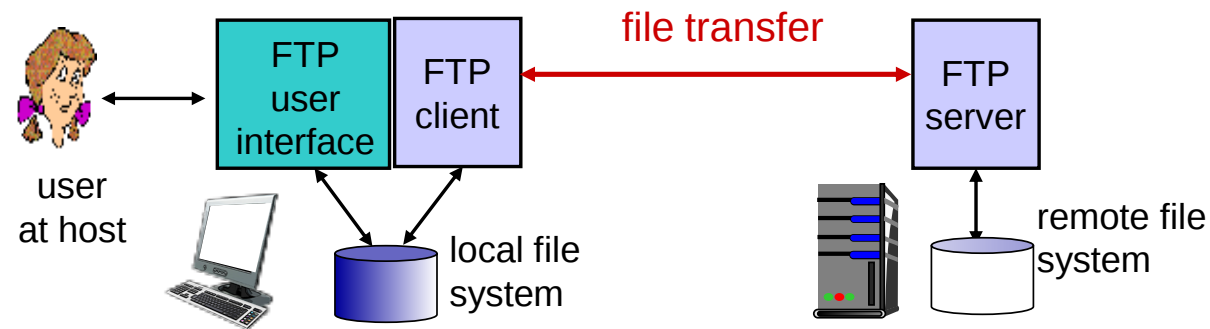
# GET Condicional: Exemplo



- Primeira requisição:
  - Data especificada posterior à última modificação do objeto.
  - Servidor envia cabeçalhos, mas não conteúdo.
  - Código 304 (Not Modified).
- Segunda requisição:
  - Data de um dia antes.
  - Servidor envia objeto, como em um GET “normal”.
- **Detalhe:** versão 1.1 do HTTP especificada nas requisições.
  - Resultado: conexão persistente utilizada por padrão.
  - Ambas as requisições feitas em uma única conexão.

# FTP

# FTP: File Transfer Protocol

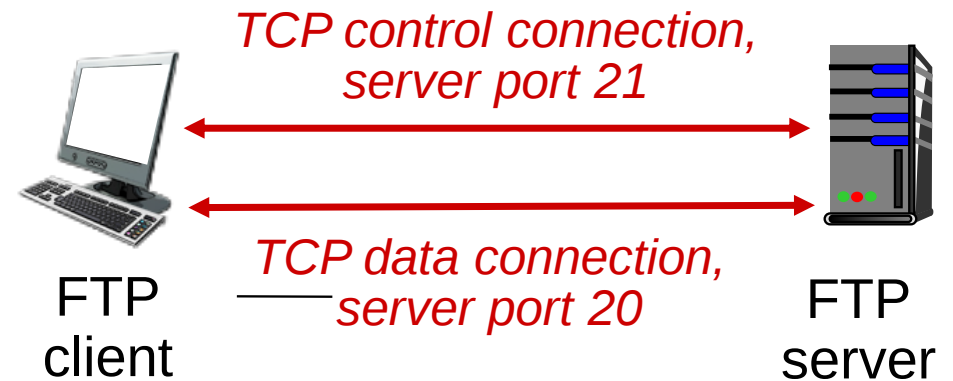


- Transfere arquivos de/para *host* remoto.
- Arquitetura Cliente-Servidor.
  - **Cliente:** lado que inicia transferência (seja de ou para *host* remoto).
  - **Servidor:** *host* remoto.
- FTP: RFC 959.
- Servidor FTP: escuta, por padrão, na porta 21.



# FTP: Conexões de Controle e Dados Separadas

- Cliente FTP contacta servidor na porta 21, usando TCP.
- Cliente autorizado através da conexão de controle.
- Cliente navega diretórios remotos, envia comandos pela conexão de controle.
- Quando servidor recebe comando de transferência de arquivo, **servidor** abre 2ª conexão TCP (para arquivo) para o cliente.
- Depois de transferir arquivo, servidor fecha conexão de dados.



- Servidor abre nova conexão TCP para cada arquivo enviado.
- Conexão dedicada para controle: comunicação **“fora-de-banda”**.
- Servidor FTP precisa manter “estado”.
  - Diretório corrente, autenticação.

# FTP: Comandos e Respostas

- **Exemplos de comandos:**

- Enviados como texto ASCII pela conexão de controle.
- USER username.
- PASS password.
- LIST.
  - Retorna lista dos arquivos no diretório corrente.
- RETR filename.
  - *Download* do arquivo.
- STOR filename.
  - *Upload* do arquivo.

- **Exemplos de códigos de retorno:**

- Código de *status* e descrição (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.

# E-mail

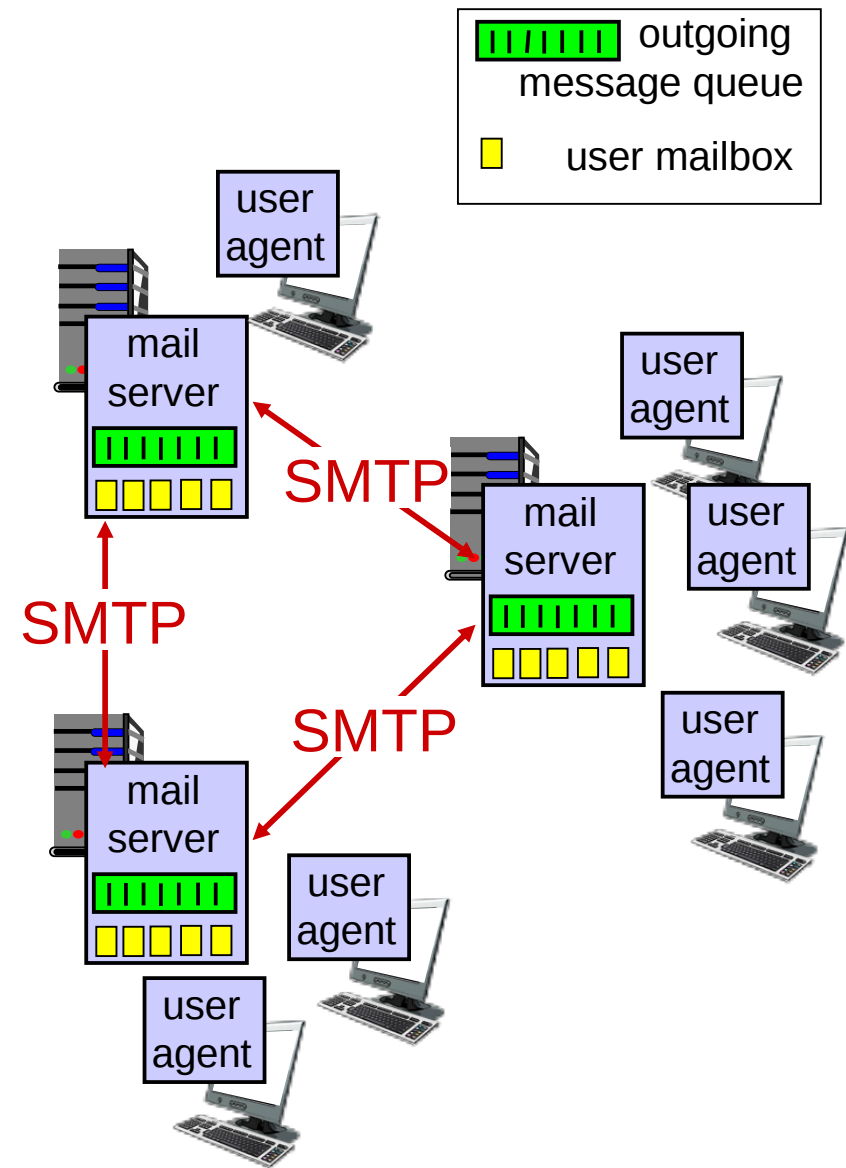
# E-mail

- **Três grandes componentes:**

- *User agents.*
- Servidores de e-mail.
- SMTP: *Simple Mail Transfer Protocol.*

- **User agent:**

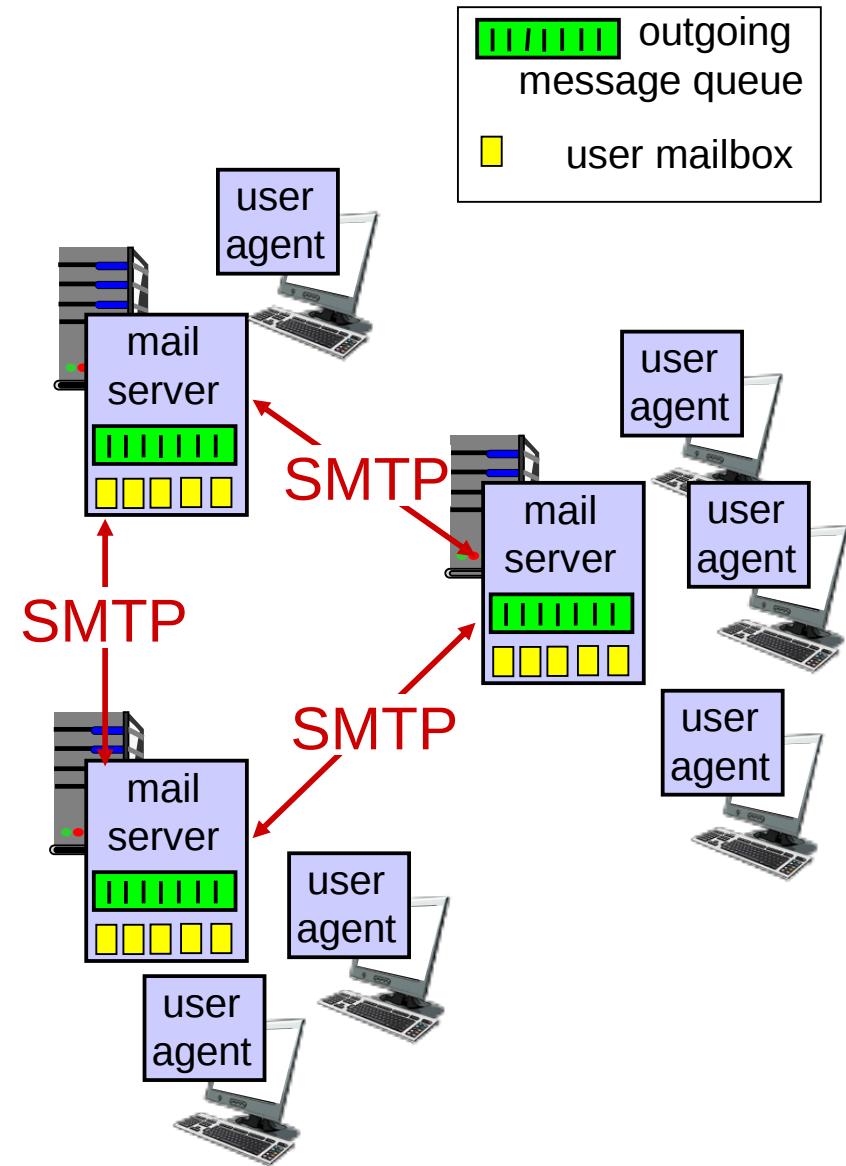
- Também conhecido como “leitor de e-mail”.
- Criação, edição, leitura de mensagens de e-mail.
- e.g., Outlook, Thunderbird, cliente de e-mail do iPhone.
- Mensagens enviadas, recebidas armazenadas no servidor.



# E-mail: Servidores de E-mail

- **Servidores de e-mail:**

- **Caixa de e-mail:** contém mensagens que chegam para o usuário.
- **Fila de mensagens:** contém mensagens a serem enviadas.
- **Protocolo SMTP:** comunicação entre servidores de e-mail para envio de mensagens.
  - Cliente: servidor que envia mensagem.
  - Servidor: servidor que recebe mensagem.

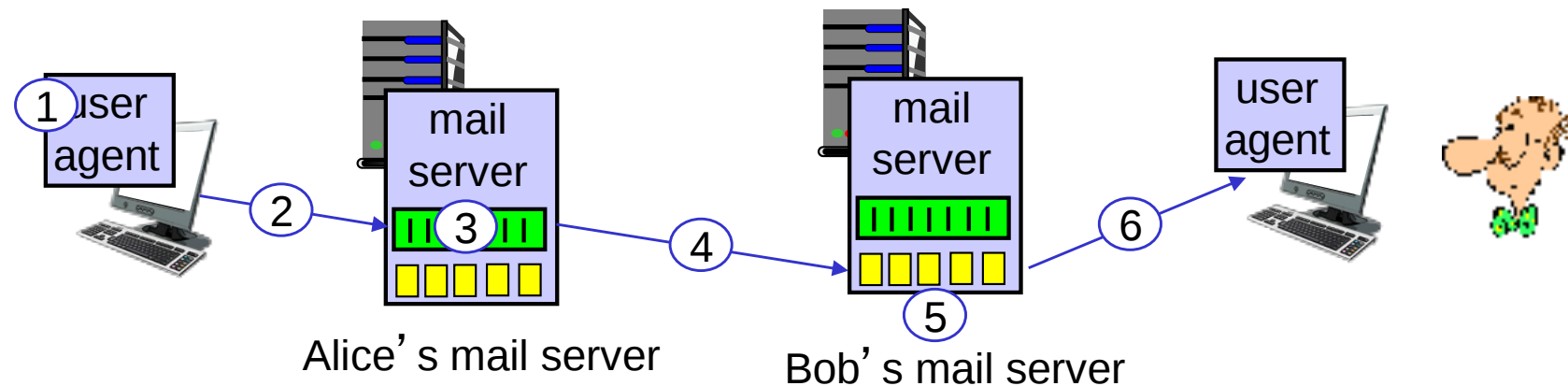


# E-mail: SMTP [RFC 2821]

- Utiliza TCP para transferir mensagens de forma confiável entre cliente e servidor, porta 25.
- Transferência direta: do servidor do destinatário diretamente para o servidor do remetente.
- Protocolo em três fases:
  - *Handshaking* (apresentação).
  - Transferência da mensagens.
  - Encerramento.
- Interação do tipo comando/resposta (similar ao FTP, HTTP).
  - **Comandos:** texto ASCII.
  - **Resposta:** código de *status* e descrição.
- Mensagens necessariamente em ASCII de 7 bits.

# Cenário: Alice Envia Mensagem para Bob

1. Alice usa *user agent* para criar mensagem para bob@somechool.edu.
2. O *user agent* de Alice envia mensagem ao seu servidor de e-mail (de Alice); mensagem é enfileirada.
3. No servidor de Alice, lado cliente do SMTP abre conexão TCP para o servidor de Bob.
4. Mensagem de Alice é enviada pela conexão TCP.
5. Servidor de e-mail de Bob coloca mensagem na caixa de entrada de Bob.
6. Bob usa seu *user agent* para ler a mensagem.



# Exemplo de Interação SMTP

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



# Experimente uma Interação SMTP Você Mesmo

- telnet servername 25
- Espere uma resposta 220 do servidor.
- Digite comandos HELO, MAIL FROM, RCPT TO, DATA, QUIT.

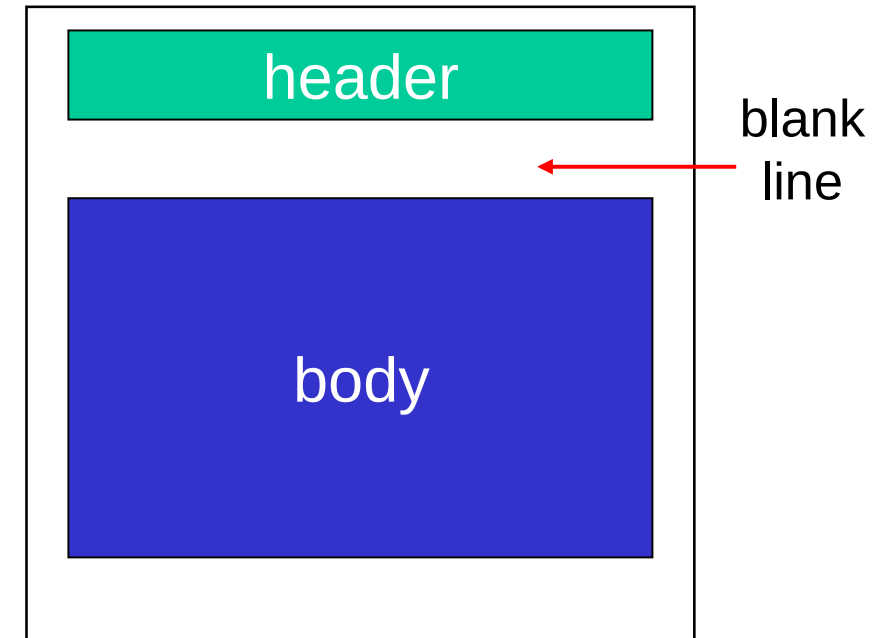
**Permite que você envie e-mail sem usar um cliente.**

# SMTP: Últimos Comentários

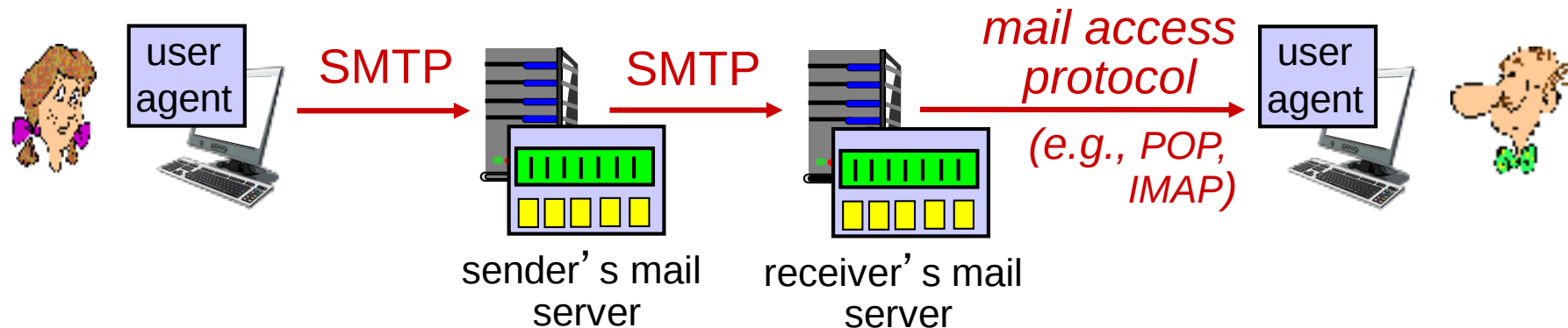
- SMTP utiliza conexões persistentes.
  - SMTP requer que mensagem seja composta apenas de caracteres ASCII de 7 bits.
  - Servidor utiliza CRLF . CRLF para determinar o fim da mensagem.
- **Comparação com o HTTP:**
    - HTTP: *pull*.
    - SMTP: *push*.
    - Ambos têm mensagens em ASCII, códigos de *status*.
    - HTTP: cada objeto encapsulado em sua própria mensagem de resposta.
    - SMTP: múltiplos objetos enviados em mensagem de múltiplas partes.

# Formato da Mensagem de E-mail

- SMTP: protocolo para troca de mensagens de e-mail.
- RFC 822: padrão para formato das mensagens.
  - Linhas de cabeçalho, *e.g.*:
    - To:
    - From:
    - Subject:
    - **Diferentes dos comandos MAIL FROM, RCPT TO do SMTP!**
  - Corpo: a “mensagem em si”.
    - Apenas caracteres ASCII.



# Protocolos de Acesso a E-mails



- **SMTP**: entrega mensagem ao servidor do destinatário.
- Protocolo de acesso de e-mail: destinatário obtém suas mensagens do seu servidor.
  - **POP**: Post Office Protocol [RFC 1939]: autorização, *download*.
  - **IMAP**: Internet Mail Access Protocol [RFC 1730]: mais funcionalidades, incluindo manipulação de mensagens armazenadas no servidor.
  - **HTTP**: gmail, hotmail, Yahoo! Mail, etc.

# O Protocolo POP3

- **Fase de autorização:**


- Comandos do cliente:
  - **user:** declara o nome do usuário.
  - **pass:** senha.

Servidor responde:


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

- **Fase de transações:**

- Cliente:
  - **list:** lista números das mensagens.
  - **retr:** obtém mensagem por número.
  - **dele:** apaga mensagem.
  - **quit:** encerra comunicação.



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



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

# POP3 (Mais) e IMAP

- **Mais sobre o POP3:**

- Exemplo anterior usa POP3 no modo “download and delete”.
- Mensagens são baixadas para *host* do Bob e apagadas do servidor.
- Há também o modo “download and keep”: cópias são deixadas no servidor.
- POP3 é *stateless* entre sessões.

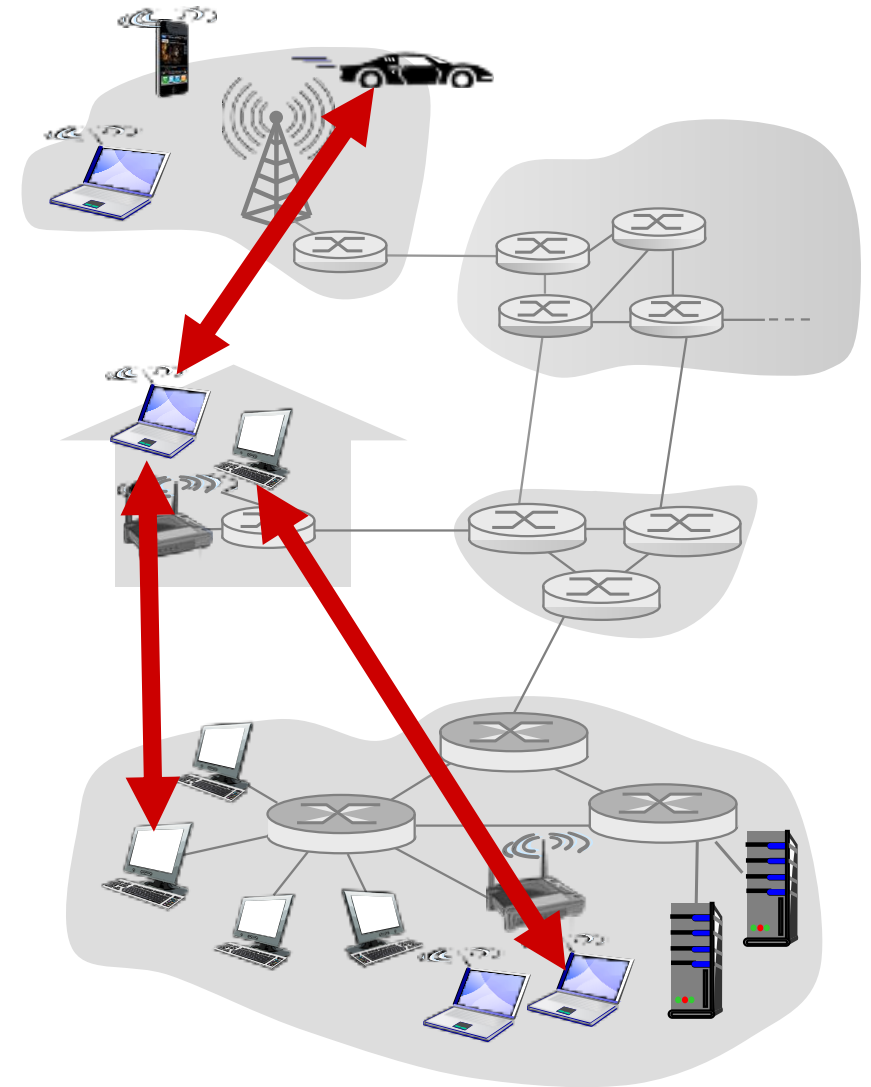
- **IMAP:**

- Mantém mensagens apenas no servidor.
- Permite ao usuário organizar mensagens em pastas.
- Mantém estado entre sessões do usuário.
  - Nomes de pastas e mapeamentos de mensagens para pastas.

# Aplicações P2P

# Arquitetura P2P Pura

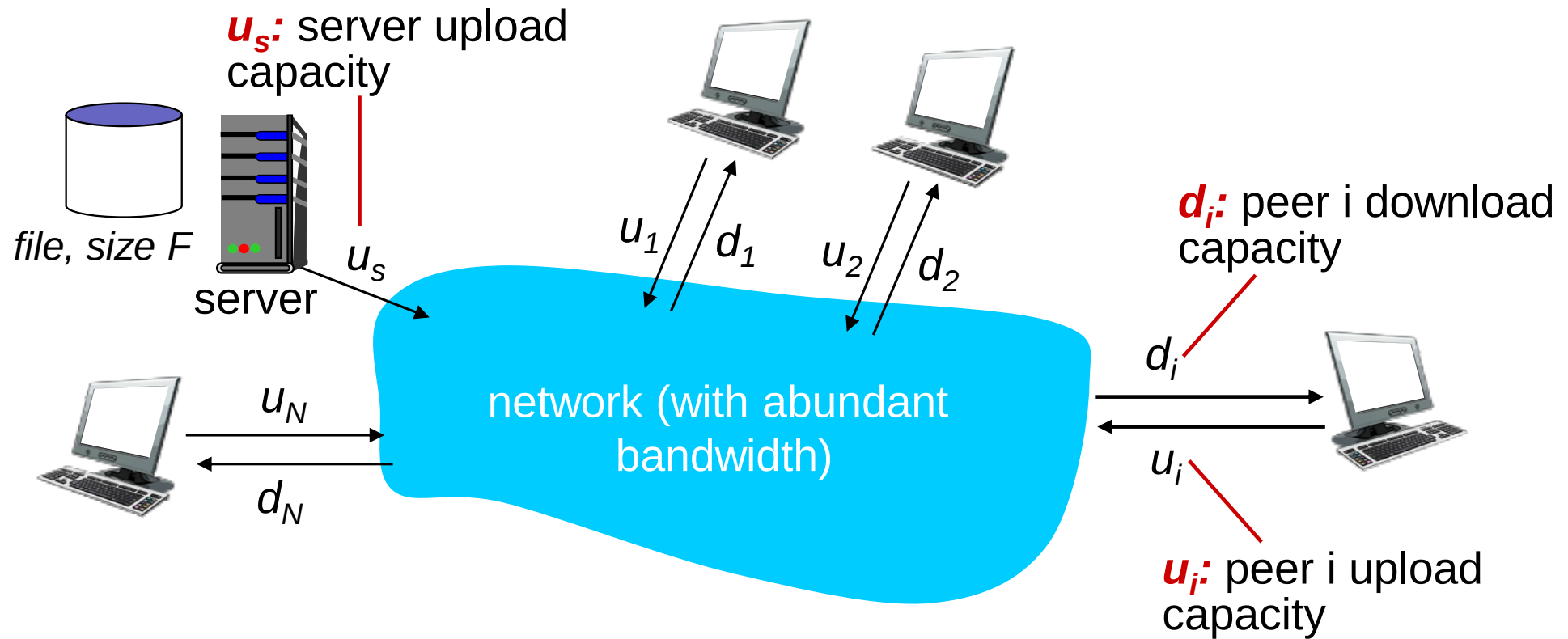
- Não há servidor sempre ligado.
- Sistemas finais arbitrários se comunicam diretamente.
- Pares se conectam à rede P2P de forma intermitente, podem trocar de endereço IP.
- **Exemplos:**
  - Distribuição de arquivos (e.g., BitTorrent).
  - Streaming (e.g., KanKan).
  - VoIP (e.g., Skype).





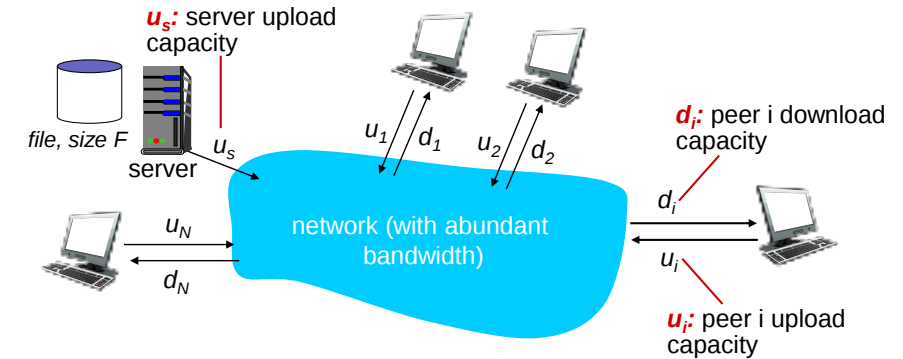
# Distribuição de Arquivos: Cliente–Servidor vs. P2P

- **Pergunta:** quanto tempo é necessário para distribuir um arquivo (tamanho  $F$ ) de um servidor para  $N$  clientes?
  - Capacidades de *download/upload* dos clientes é um recurso limitado.



# Distribuição de Arquivos: Cliente—Servidor

- **Transmissão pelo servidor:** precisa enviar (*upload*) **sequencialmente** N cópias do arquivo.
  - Tempo para enviar uma cópia:  $\frac{F}{u_s}$ .
  - Tempo para enviar N cópias:  $\frac{N \cdot F}{u_s}$ .
- **Cliente:** cada cliente precisa receber (*download*) sua cópia do arquivo.
  - $d_{min}$  = capacidade de *download* mínima entre todos os clientes.
  - Tempo máximo de *download* entre os clientes:  $\frac{F}{d_{min}}$ .



Tempo para distribuir arquivo de tamanho  $F$  para  $N$  clientes utilizando abordagem Cliente—Servidor

$$D_{C-S} = \max \left\{ \frac{N \cdot F}{u_s}, \frac{F}{d_{min}} \right\}$$

# Distribuição de Arquivos: P2P

- **Transmissão pelo servidor:** precisa enviar (*upload*) **ao menos uma cópia**.

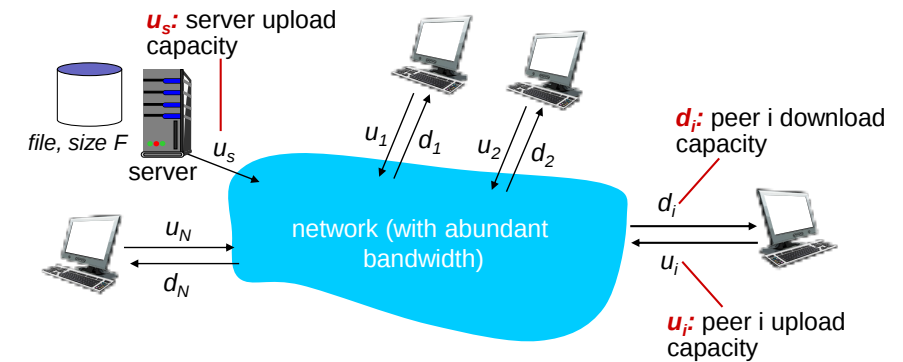
- Tempo para enviar uma cópia:  $\frac{F}{u_s}$ .

- **Cliente:** cada cliente precisa receber (*download*) sua cópia do arquivo.

- Tempo máximo de *download* entre os clientes:  
 $\frac{F}{d_{min}}$ .

- **Clientes:** em conjunto, clientes farão *download* de  $N \cdot F$  bits.

- Taxa máxima de *upload* (que limita taxa de *download*) é  $u_s + \sum u_i$ .



Tempo para distribuir arquivo de tamanho  $F$  para  $N$  clientes utilizando abordagem P2P

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

# Distribuição de Arquivos: Comparação (I)

- Caso Cliente—Servidor:

$$D_{C-S} = \max \left\{ \frac{N \cdot F}{u_s}, \frac{F}{d_{min}} \right\}$$

- Demanda cresce linearmente com N.
- Capacidade do servidor é fixa.

- Caso P2P:

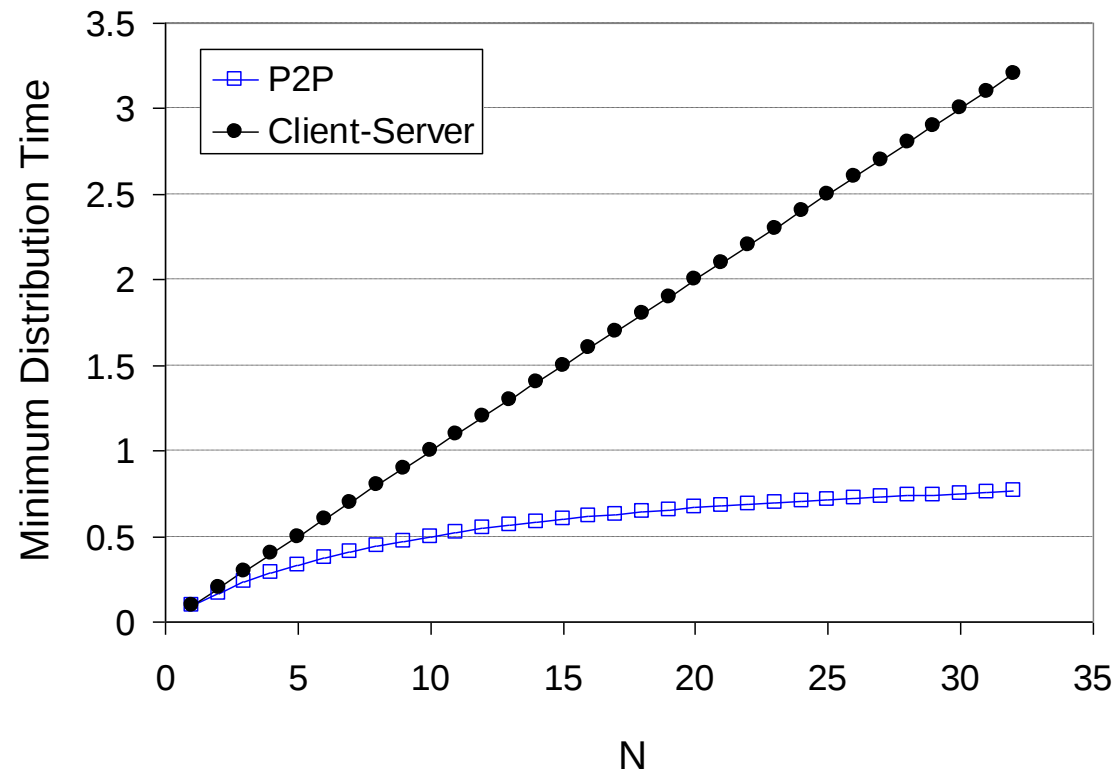
$$D_{P2P} = \max \left\{ \frac{F}{u_s}, \frac{F}{d_{min}}, \frac{N \cdot F}{u_s + \sum u_i} \right\}$$

- Demanda aumenta linearmente com N.
- Mas também a capacidade de *upload*.
- Resultado: **tempo de distribuição cresce, mas de forma mais escalável.**

# Distribuição de Arquivos: Comparação (II)

- Exemplo numérico:

- $\frac{F}{u} = 1$  hora.
- Capacidade de *upload* do servidor 10x maior que dos clientes.
- $d_{min} \geq us$ .

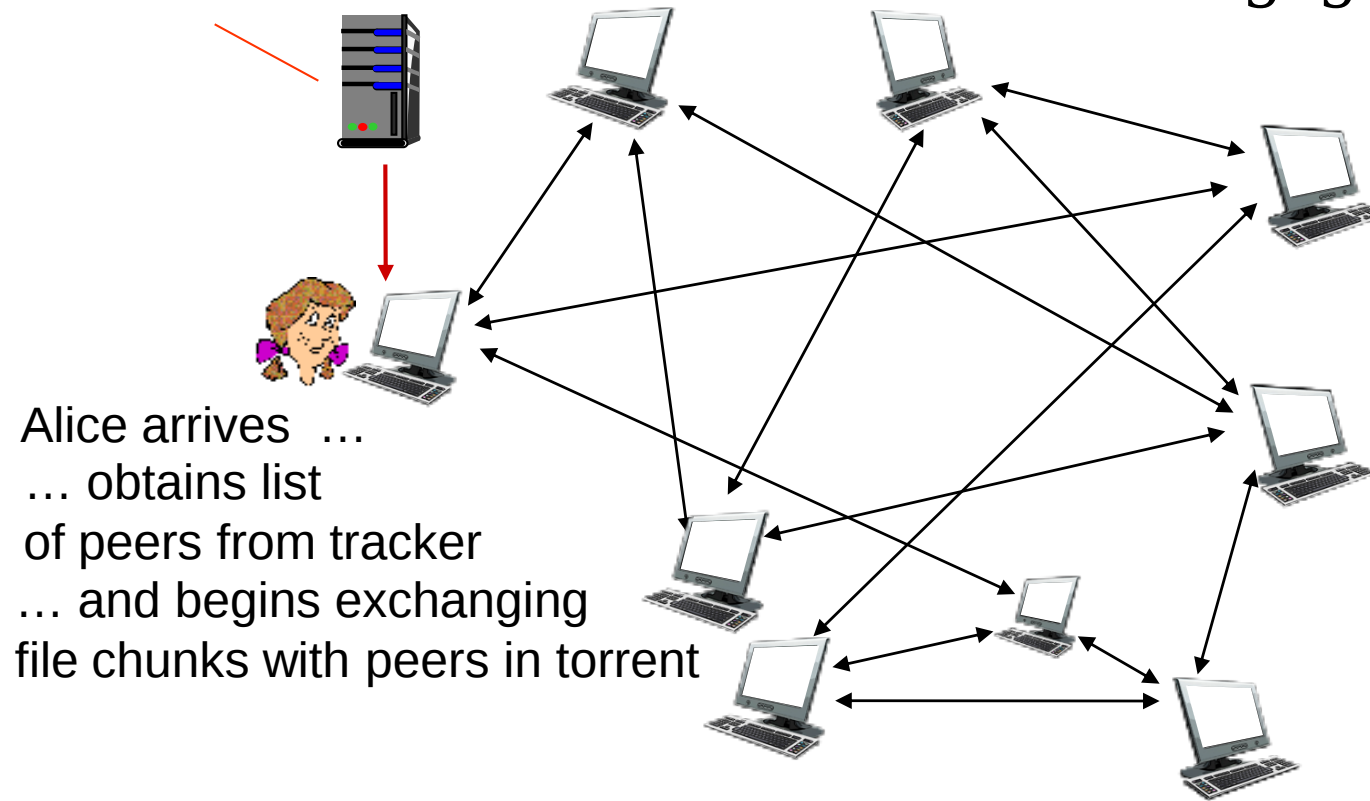


# Distribuição de Arquivos P2P: Exemplo do BitTorrent (I)

- Arquivo dividido em pedaços de (normalmente) 256KB.
- Pares no torrent enviam/recebem pedaços do arquivo.

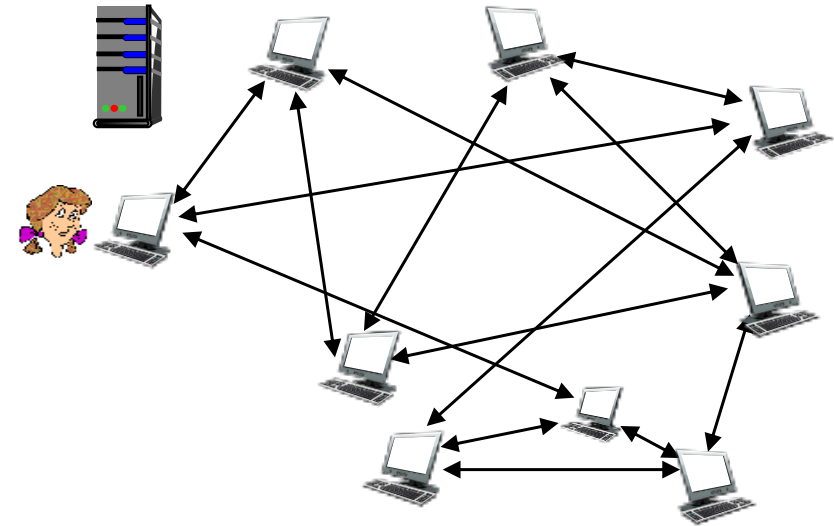
*tracker*: tracks peers  
participating in torrent

*torrent*: group of peers  
exchanging chunks of a file



# Distribuição de Arquivos P2P: Exemplo do BitTorrent (II)

- Par se junta ao torrent:
  - Não possui nenhum pedaço, mas os acumulará com o tempo de outros pares.
  - Se registra com o *tracker* para obter uma lista de pares, se conecta a um subconjunto dos pares (“vizinhos”).
- Enquanto baixa, pares fazem *upload* para outros pares.
- Par pode alterar os pares com que troca pedaços.
- **Churn:** pares vem e vão.
- Quando um par tem o arquivo inteiro (todos os pedaços), ele pode (de forma egoísta) sair ou (de forma altruísta) ficar no torrent.



# BitTorrent: Requisitando, Enviando Pedacos do Arquivo

- **Requisitando pedacos:**

- Em um dado momento, diferentes pares possuem diferentes subconjuntos dos pedacos do arquivo.
- Periodicamente, um par pede aos outros pares uma lista dos pedacos que possuem.
- Par então requisita pedacos que não possui aos pares, começando pelos mais raros.

- **Enviando pedacos: *tit-for-tat*.**

- Par envia pedacos aos 4 pares que atualmente o enviam pedacos na taxa mais alta.
  - Outros pares sofrem *choking* (i.e., não recebem pedacos).
  - Uma nova avaliação dos 4 melhores pares é feita a cada 10 segundos.
- A cada 30 segundos: par seleciona aleatoriamente outro par, começa a enviar pedacos.
  - *Optimistically unchoke*.
  - Dá oportunidade de pares demonstrarem que são bons *uploaders*.
  - Par escolhido de forma aleatória pode se tornar um dos 4 melhores.



# BitTorrent: Tit-For-Tat

- (1) Alice “optimistically unchokes” Bob
- (2) Alice becomes one of Bob’s top-four providers; Bob reciprocates
- (3) Bob becomes one of Alice’s top-four providers

