

Aula 5 - Introdução à Camada de Aplicação, HTTP (I)

Diego Passos

Universidade Federal Fluminense

Redes de Computadores

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

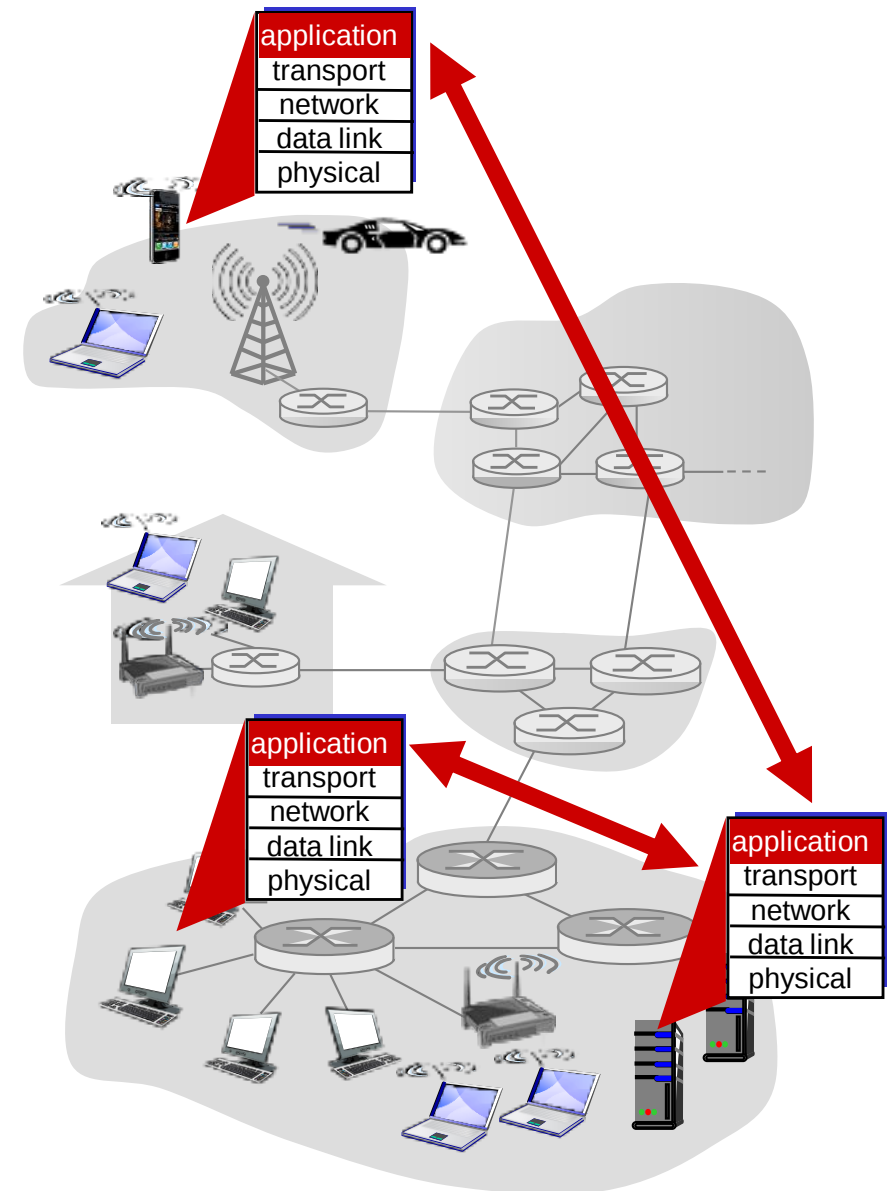
Princípios de Aplicações de Redes

Algumas Aplicações de Redes

- E-mail.
- Web.
- Mensagem de texto.
- Login remoto.
- Troca de arquivos via P2P.
- Jogos multi-usuários.
- *Streaming de vídeo armazenado.*
 - Youtube, Hulu, Netflix, ...
- Voz sobre IP (e.g., Skype).
- Vídeo conferência em tempo real.
- Redes sociais.
- Busca.
- ...
- ...

Criação de uma Aplicação de Rede

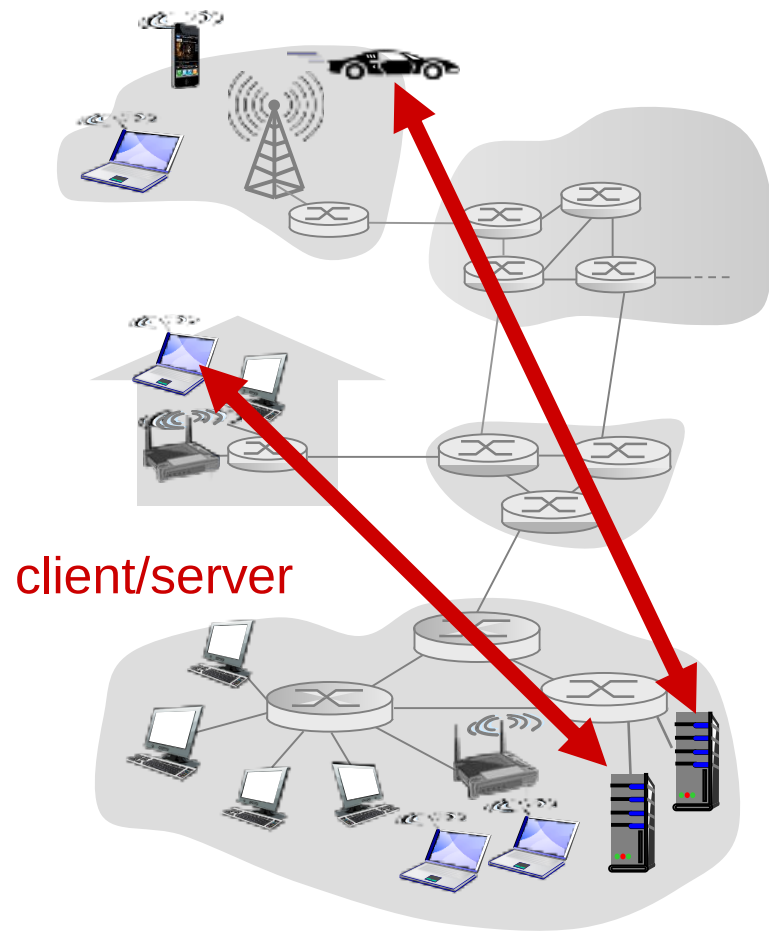
- **Escrever programas que:**
 - Rodem em (diferentes) sistemas finais.
 - Se comuniquem via rede.
 - e.g., *software* de um servidor web que se comunica com o *software* de um *browser*.
- **Não é necessário se preocupar com a escrita de *software* para dispositivos do núcleo.**
 - Dispositivos do núcleo não rodam aplicações.
 - **Idealmente.**
 - Implantação apenas nos *hosts* permite **rápido desenvolvimento, popularização das aplicações.**



Arquiteturas de Aplicações de Rede

- Aplicações podem ser estruturadas em duas arquiteturas básicas:
 - Cliente-servidor.
 - Par-a-par (P2P, do inglês *peer-to-peer*).

Arquitetura Cliente-Servidor



- **Servidor:**

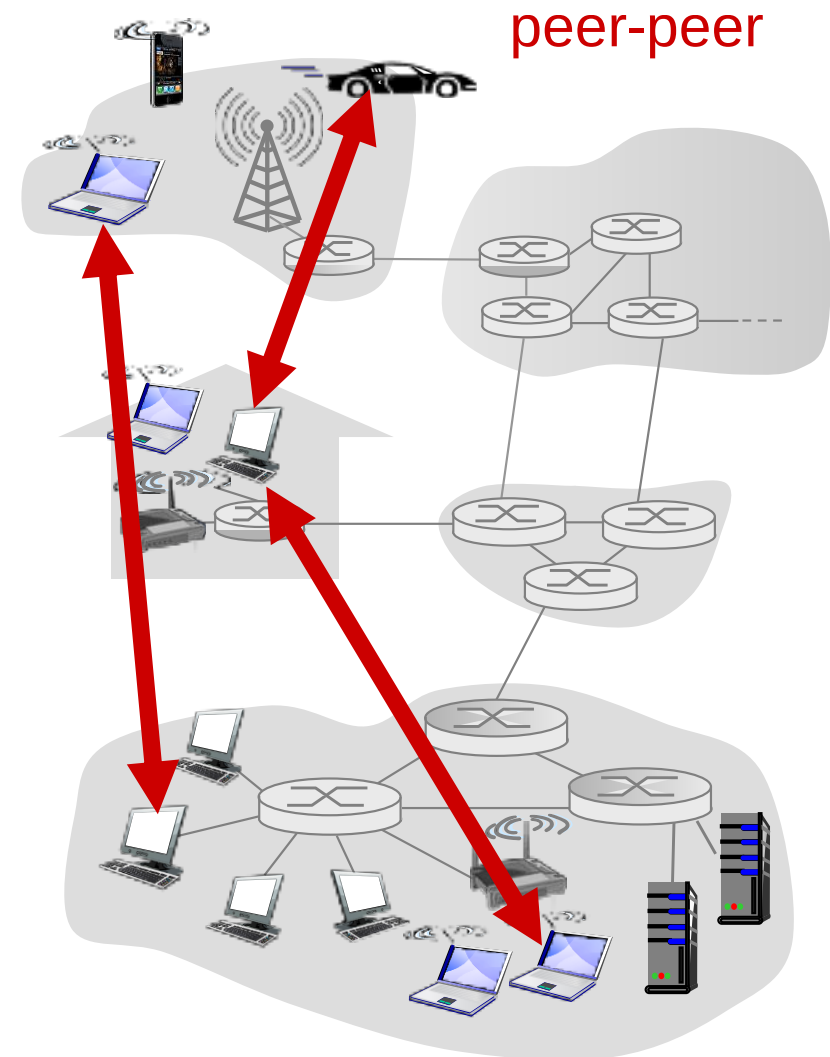
- Host sempre ligado.
- Endereço IP permanente.
- *Data Centers* podem ser usados para escalabilidade.

- **Cliente:**

- Se comunica com o servidor.
 - Geralmente, inicia comunicação.
- Pode estar conectado de forma intermitente.
- Pode utilizar endereços dinâmicos.
- **Dois ou mais clientes não se comunicam diretamente.**
 - Servidor atua como **refletor**.

Arquitetura P2P

- Não há servidor sempre ligado.
- Comunicação direta entre sistemas finais quaisquer.
- **Pares** requisitam serviços de outros pares, proveem serviços em retorno.
- Pares se conectam de forma intermitente, podem mudar seu endereço IP.
 - Gerenciamento **mais complexo**.
- Por outro lado...
 - **Auto-escalabilidade**: mais pares trazem mais demanda, mas também agregam capacidade ao serviço.



Comunicação entre Processos

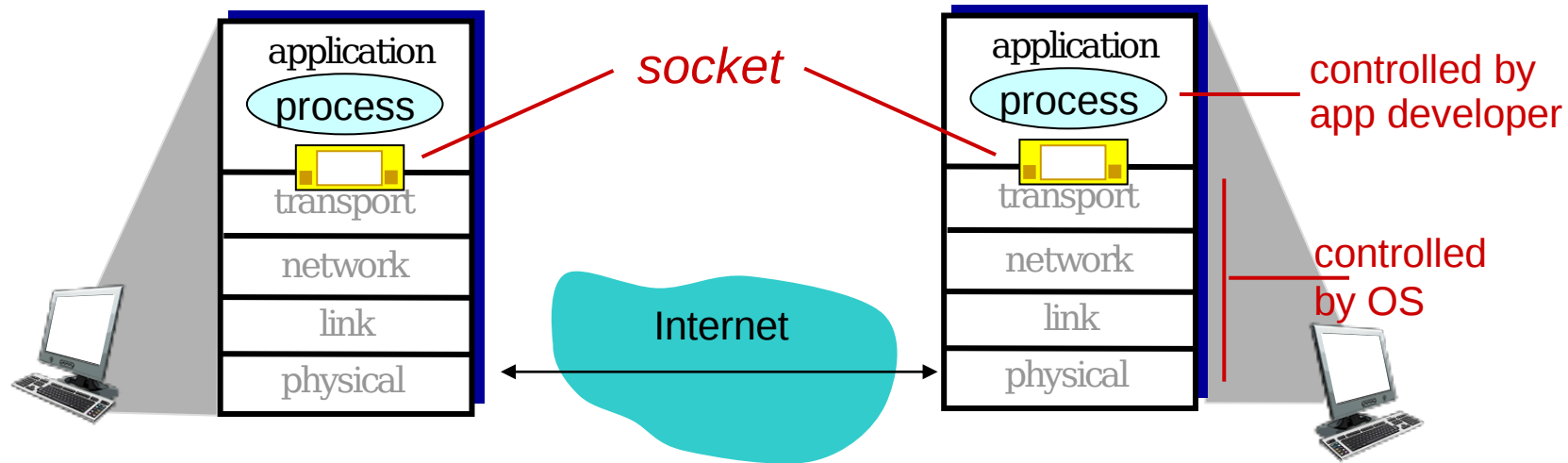
- Aplicações são compostas por **processos**.
 - Executados dentro de um *host*.
- Dentre de um mesmo *host*, processos de comunicam utilizando mecanismos de **comunicação inter-processos**.
 - Definidos pelo SO.
 - e.g., leitura/escrita de arquivos.
 - e.g., memória compartilhada.
- Em *hosts* diferentes, processos se comunicam através de **troca de mensagens**.

Clientes, Servidores

- **Processo cliente:** inicia a comunicação.
 - **Processo servidor:** aguarda pedidos de comunicação.
-
- Nota: aplicações P2P possuem tanto processos servidores, quanto processos clientes.

Sockets

- Processos enviam/recebem mensagens utilizando um **socket**.
- Socket é análogo a uma janela.
 - Processo transmissor passa dados pela janela.
 - Do outro lado, implementação da camada de transporte se encarrega em entregar mensagem.
 - Na recepção, procedimento é análogo: processo espera que transporte entregue dados pelo socket.



Endereçando Processos

- Para receber mensagens, processos devem ser **endereçados**.
 - i.e., devem possuir algum **identificador**.
- Hosts possuem endereços únicos.
 - Endereço IP (mais detalhes no Cap. 4).
- **Pergunta:** endereço IP do *host* é suficiente para identificar o processo?
 - **Resposta:** não, vários processos de rede podem ser executados simultaneamente no mesmo *host*.
- Solução: identificação de um processo usa tanto IP do *host*, quanto **números de porta**.
- Exemplos de números de porta:
 - Servidor HTTP: 80.
 - Servidor de e-mail (SMTP): 25.
- Para enviar mensagem HTTP para servidor web do IC:
 - Endereço IP: 200.20.15.48.
 - Porta: 80.
- Mais sobre isso em breve...

O que um Protocolo de Camada de Aplicação Define?

- **Tipos de mensagens trocadas.**
 - e.g., requisição, resposta.
- **Sintaxe das mensagens.**
 - Quais os **campos** da mensagem, como dados são representados nestes campos.
- **Semântica das mensagens.**
 - Significado dos valores nos campos.
- **Regras** para como e quando gerar e responder a mensagens.
- **Protocolos abertos:**
 - Definidos em RFCs.
 - Permitem interoperabilidade.
 - e.g., HTTP, SMTP.
- **Protocolos proprietários:**
 - e.g., Skype.

De que Serviço de Transporte a Aplicação Precisa?

- **Integridade de dados:**

- Algumas aplicações precisam de 100% de confiança na transmissão de dados.
 - e.g., transferência de arquivos.
 - Dados precisam chegar **completos e corretos**.
- Outras toleram algumas perdas, corrupções.
 - e.g., áudio.

- **Temporização:**

- Algumas aplicações precisam de baixo atraso para serem “efetivas”.
 - e.g., telefonia, jogos interativos.

- **Vazão:**

- Algumas aplicações requerem uma vazão mínima para funcionar.
 - e.g., multimídia.
- Outras se adaptam a qualquer vazão disponível.
 - São **“elásticas”**.
 - e.g., transmissão de arquivos.

- **Segurança:**

- Criptografia, autenticidade, ...

Requisitos de Serviço de Transporte: Algumas Aplicações Comuns

Aplicação	Perda de Pacotes	Vazão	Requisitos Temporais
Transferência de Arquivos	Não Tolera	Elástica	Não
E-mail	Não Tolera	Elástica	Não
Navegação Web	Não Tolera	Elástica	Não
Vídeo/Áudio Ao-Vivo	Tolera	Áudio: 5 kb/s – 1 Mb/s Vídeo: 10 kb/s – 5 Mb/s	Centenas de ms
Vídeo/Áudio Armazenado	Tolera	Áudio: 5 kb/s – 1 Mb/s Vídeo: 10 kb/s – 5 Mb/s	Alguns Segundos
Jogos Interativos	Tolera	Alguns kb/s	Centenas de ms
Mensagens de Texto	Não Tolera	Elástica	Alguns Segundos

Serviços dos Protocolos de Transporte da Internet

● Serviço do TCP:

- Transporte confiável de dados.
- Controle de fluxo: transmissor não sobrecarregará receptor.
- Controle de congestionamento: reduzir taxa de transmissão quando a rede está congestionada.
- **Mas não provê:** garantias temporais, vazão mínima garantida, segurança.
- Serviço orientado a conexão: *setup* inicial é necessário entre cliente e servidor.

● Serviço do UDP:

- Transmissão **não confiável** de dados.
- Não provê: confiabilidade, controle de fluxo, controle de congestionamento, garantias temporais, vazão mínima, segurança, *setup* de conexão.
- **Pergunta:** para que existe um UDP?

Aplicações na Internet: Protocolos de Transporte Utilizados

Aplicação	Protocolo de Aplicação	Protocolo de Transporte
E-mail	SMTP [RFC 2821]	TCP
Login Remoto	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
Transferência de Arquivos	FTP [RFC 959]	TCP
Streaming Multimídia	HTTP (e.g., Youtube), RTP [RFC 1889]	TCP ou UDP
Telefonia via Internet	SIP, RTP, proprietários (e.g., Skype)	TCP ou UDP

Adicionando Segurança ao TCP

- **TCP e UDP.**

- Não inclem criptografia.
- Senhas em texto plano enviadas pelo socket atravessam a Internet em texto plano.

- **SSL.**

- Provê conexão TCP criptografada.
- Integridade de dados.
- Autenticação das pontas.

- **SSL está na camada de aplicação.**

- Aplicações usam bibliotecas SSL para “conversar” com o TCP.

- **API de sockets com SSL.**

- Senhas em texto plano enviadas pelo socket atravessam a Internet criptografadas.
- Mais detalhes em outras disciplinas (Cap. 7).

Web e HTTP

Web e HTTP

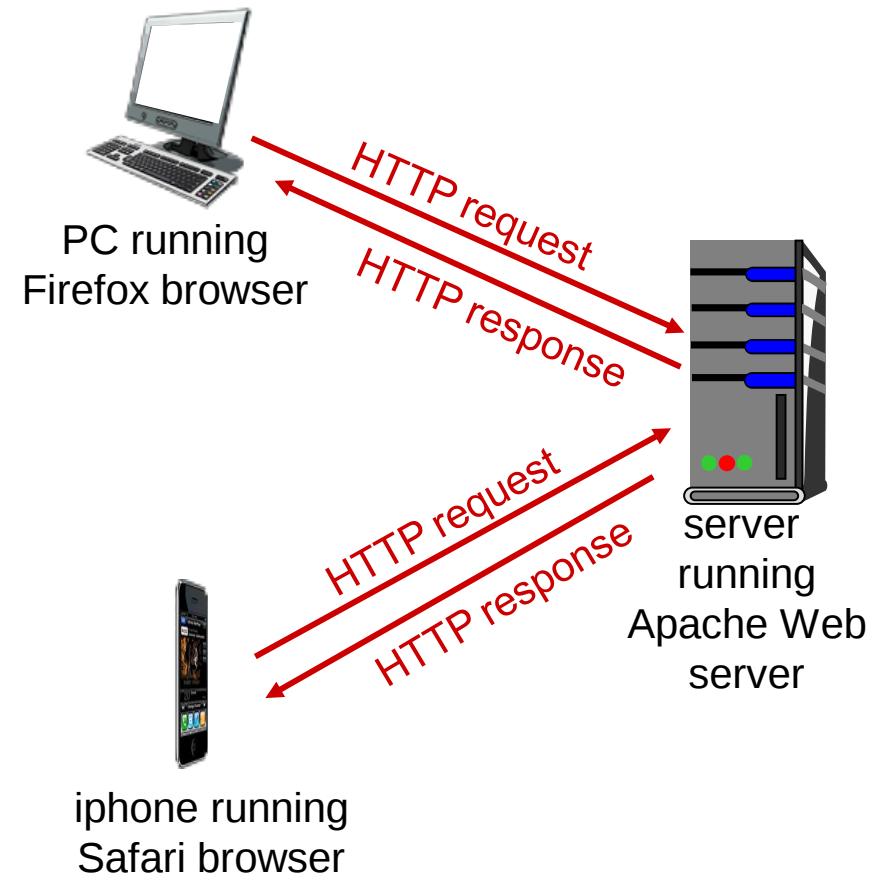
- Uma **página web** é formada por **objetos**.
 - Arquivos HTML, imagens *jpg*, *png*, vídeos, áudios, ...
- Arquivo HTML base da página faz **referência** a diversos objetos.
- Cada objeto é **endereçável através de uma URL**. Exemplo:

`http://www.someschool.edu/someDept/pic.gif`

Esquema Nome do Host Caminho

Visão Geral do HTTP (I)

- **HTTP: HyperText Transfer Protocol.**
 - Protocolo de camada de aplicação da Web.
 - Arquitetura cliente — servidor.
 - **Cliente:** browser que requisita, recebe (usando protocolo HTTP) e “exibe” objetos Web.
 - **Servidor:** servidor Web envia (usando protocolo HTTP) objetos em resposta a requisições.



Visão Geral do HTTP (II)

- **Usa TCP.**

- Cliente inicia conexão TCP (cria socket) com o servidor, porta 80.
- Servidor aceita conexão TCP do cliente.
- Mensagens HTTP (mensagens do protocolo de aplicação) trocadas entre *browser* (cliente HTTP) e servidor Web (servidor HTTP).
- Conexão TCP é fechada.

- **HTTP é “stateless”.**

- Servidor não mantém informação sobre requisições passadas do cliente.

Nota

- **Protocolos que mantêm estado são complexos!**
 - Histórico do passado (estado) precisa ser armazenado.
 - Se servidor/cliente cai, suas visões do estado podem ser inconsistentes, devem ser sincronizadas.

- **HTTP não-persistente.**

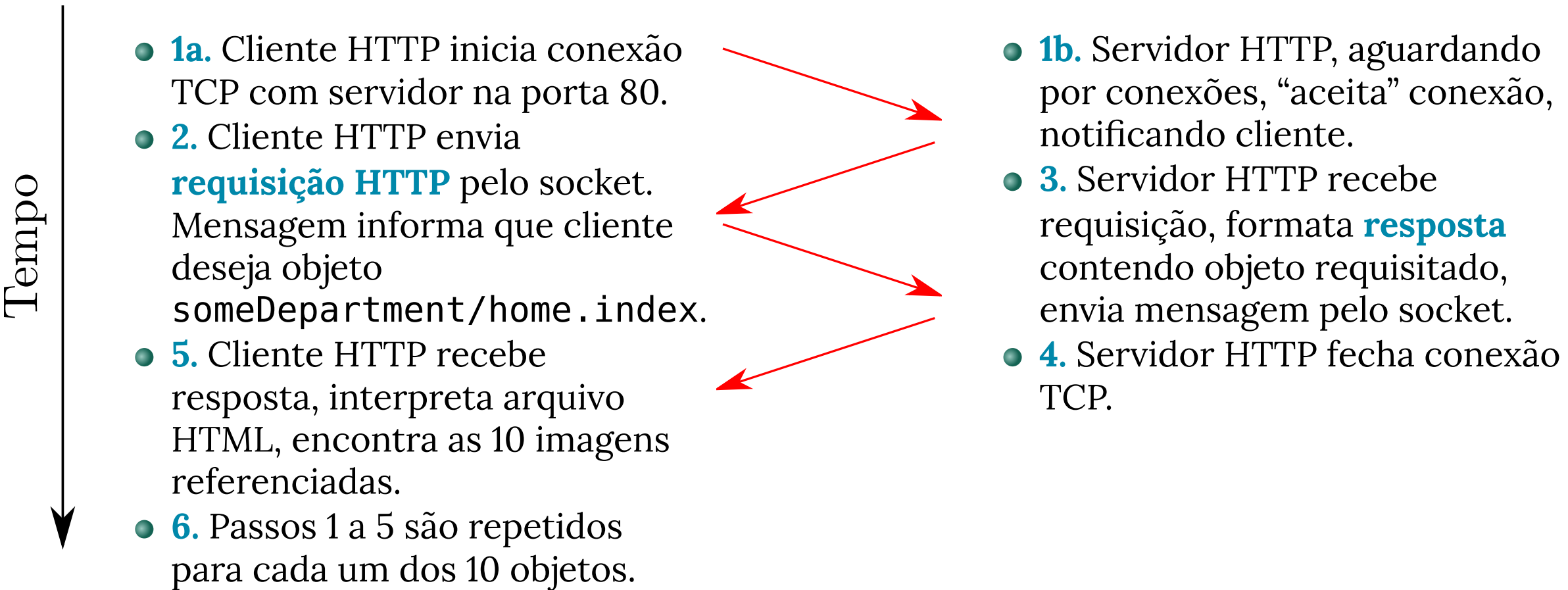
- Um único objeto enviado pela conexão TCP.
 - Conexão é fechada em seguida.
- Baixar múltiplos objetos requer múltiplas conexões.

- **HTTP persistente.**

- Múltiplos objetos podem ser enviados por uma única conexão TCP entre cliente e servidor.

HTTP Não-persistente: Exemplo

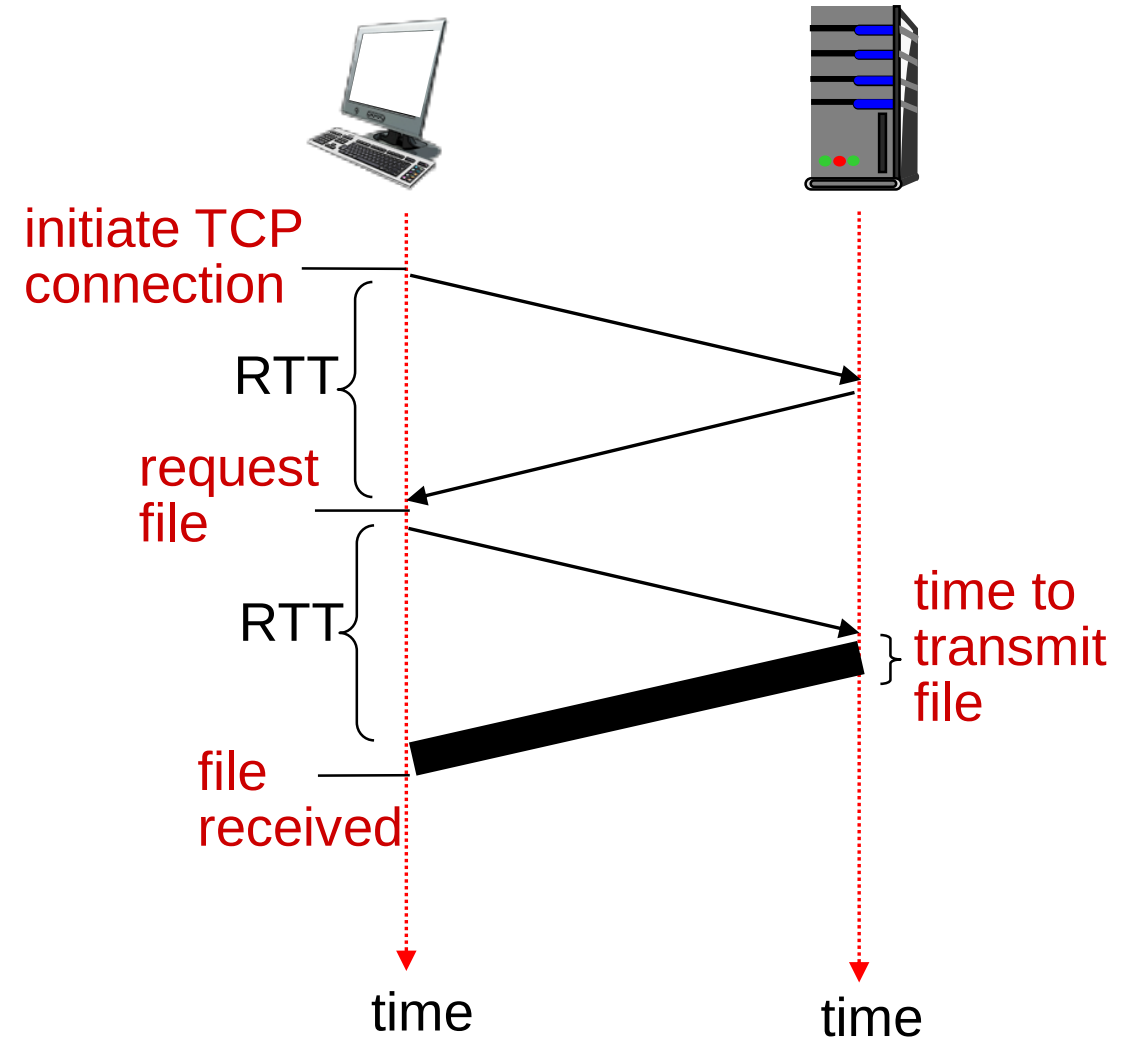
- Suponha que usuário digita a URL: `www.someschool.edu/someDepartment/home.index`.
 - Contém referências para 10 imagens `jpg`.



HTTP Não-persistente: Tempo de Resposta

RTT

- Round-Trip-Time, ou tempo de ida e volta.
 - Tempo para que um pequeno pacote chegue até o servidor e volte para o cliente.
-
- **Tempo de resposta do HTTP.**
 - Um RTT para iniciar a conexão TCP.
 - Um RTT para requisição HTTP e volta dos primeiros bytes da resposta.
 - Tempo de transmissão do arquivo.
 - Em suma: $2\text{RTT} + \text{transmissão do arquivo}$.



HTTP Persistente

- **Problemas do HTTP não persistente:**

- Requer 2 RTTs por objeto.
- **Overhead** do SO para cada conexão TCP.
- *Browser* geralmente abre múltiplas conexões paralelas para obter objetos.

- **HTTP Persistente:**

- Servidor deixa conexão aberta após enviar resposta.
- Mensagens HTTP subsequentes entre mesmos cliente e servidor utilizam conexão aberta.
- Cliente requisita objeto tão logo encontre referência a ele.
- Um único RTT pode ser necessário para todos os objetos referenciados.

HTTP Persistente vs. Pipelining

- **HTTP Persistente:** servidor não fecha conexão imediatamente após término do objeto.
 - Dá oportunidade ao cliente de enviar novas requisições, reaproveitar conexão aberta.
 - Evita atraso inicial de abertura da conexão TCP.
- **Pipelining:** cliente pode enviar várias requisições em sequência, sem esperar respostas chegarem.
 - Enquanto primeira resposta começa a voltar, novas requisições chegam ao servidor.
 - Cria paralelismo entre requisições e respostas.
 - Tende a melhorar o desempenho.
- Em suma: dois mecanismos distintos, embora **comumente utilizados em conjunto**.

Mensagem de Requisição HTTP

- HTTP tem dois tipos de mensagem: **requisição e resposta**.
- Requisição HTTP: codificada em ASCII (i.e., formato legível por humanos).
 - **human-readable**.

The diagram illustrates the structure of an HTTP request message. It shows a request line followed by several header lines, each ending with a carriage return and line feed character. Annotations with arrows point to these elements: 'request line (GET, POST, HEAD commands)' points to the first line; 'header lines' points to the subsequent lines; 'carriage return, line feed at start of line indicates end of header lines' points to the final blank line; and 'carriage return character' and 'line-feed character' point to the backslash-r and backslash-n characters at the end of the first header line.

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

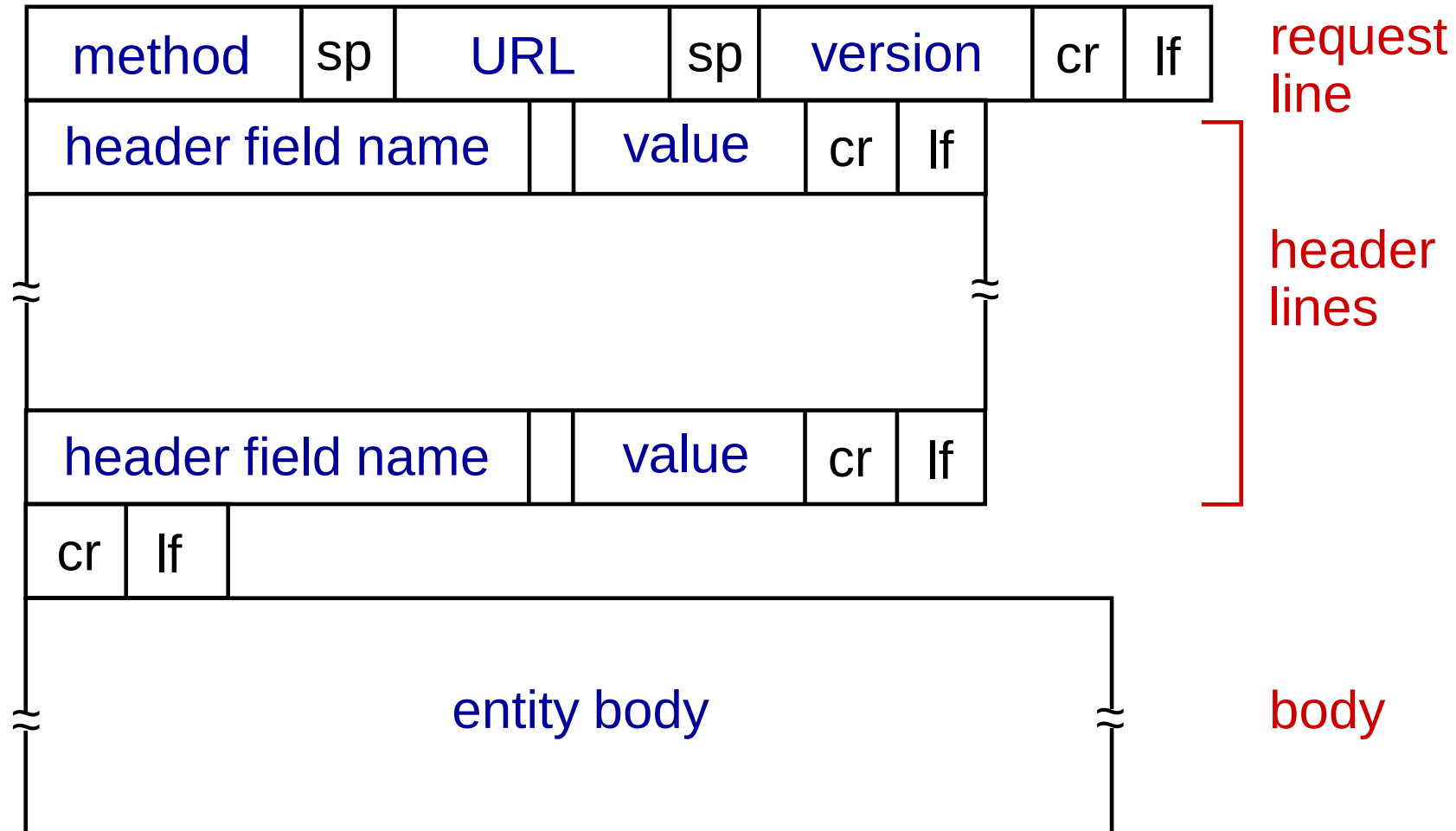
request line
(GET, POST,
HEAD commands)

header
lines

carriage return,
line feed at start
of line indicates
end of header lines

carriage return character
line-feed character

Mensagem de Requisição HTTP: Formato Geral



Upload de Informações de Formulário

- **Método POST.**

- Páginas web comumente incluem formulários.
- Valores dos campos são enviados no corpo da requisição HTTP.

- **Upload através da URL.**

- Utiliza o método HTTP GET.
- Valores dos campos são preenchidos na própria URL requisitada:

`www.somesite.com/animalsearch?monkeys&banana`

Tipos de Métodos HTTP

- **HTTP/1.0:**

- GET.
- POST.
- HEAD.
 - Pede que servidor não envie objeto na resposta.

- **HTTP/1.1:**

- GET, POST, HEAD.
- PUT.
 - Faz upload de objeto para a URL especificada.
- DELETE.
 - Apaga objeto da URL especificada.

Mensagem de Resposta do HTTP

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-
1\r\n
\r\n
data data data data data ...
```

Mensagem de Resposta do HTTP: Códigos de Status

- Aparece na primeira linha da resposta enviada pelo servidor.
- Alguns exemplos de códigos:
 - **200 OK**
 - Requisição bem sucedida, objeto se encontra no corpo da mensagem.
 - **301 Moved Permanently**
 - Objeto requisitado foi movido, nova localização se encontra no corpo da mensagem.
 - **400 Bad Request**
 - Servidor não entendeu a requisição.
 - **404 Not Found**
 - Objeto requisitado não foi encontrado no servidor.
 - **505 HTTP Version Not Supported**

Experimente o HTTP Você Mesmo

1. Faça um Telnet para o seu servidor web favorito:

```
telnet www.midiacom.uff.br 80
```

Abre uma conexão TCP para a porta 80 (porta padrão para servidores HTTP) no servidor `www.midiacom.uff.br`. Tudo que é digitado no terminal é enviado pela conexão.

2. Digite uma requisição HTTP do tipo GET:

```
GET /~diego/ HTTP/1.1  
Host: www.midiacom.uff.br
```

Ao digitar isso (e duas vezes `enter` ao final), você envia esta requisição GET mínima (mas completa) para o servidor.

3. Veja a resposta enviada pelo servidor.

- Ou use o Wireshark para olhar requisições/respostas HTTP capturadas.

Alguns Experimentos com HTTP (I)

- HTTP 1.0: conexão não-persistente.
- Conexão fechada assim que objeto é enviado.



Alguns Experimentos com HTTP (II)

- HTTP 1.1: por padrão, conexão persistente.
- Servidor mantém conexão aberta .
 - Durante algum tempo.



Alguns Experimentos com HTTP (III)

- HTTP 1.1: **pode** operar em modo não-persistente.
- Exige configuração explícita no cabeçalho:

Connection: close

