PTC 3450 - Aula 05

2.1 Princípios das aplicações de rede
2.2 A *Web* e o HTTP

(Kurose, p. 62 - 73)
(Peterson, p. 425 - 444)
21/03/2017

Capítulo 2: conteúdo

2.1 Princípios de aplicativos de rede

2.2 *Web* e HTTP

2.3 Correio eletrônico

- SMTP, POP3, IMAP

2.4 DNS

2.5 Aplicativos P2P

2.6 *Streaming* de vídeo e redes de distribuição de conteúdo

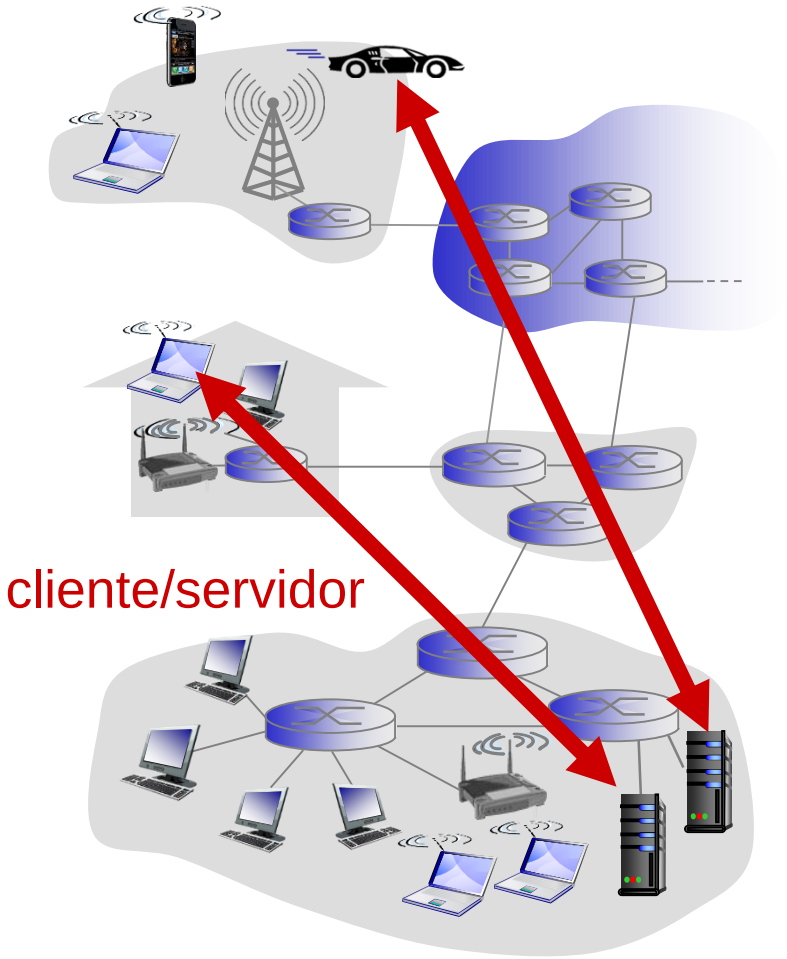
2.7 Programando *socket* com UDP e TCP

Arquiteturas de aplicativos

possíveis estruturas de aplicativos:

- cliente-servidor
- *peer-to-peer* (P2P)

Arquitetura cliente-servidor



servidor:

- em *host* sempre ligado
- endereço IP permanente
- *data centers* pensando em escala (servidor virtual)

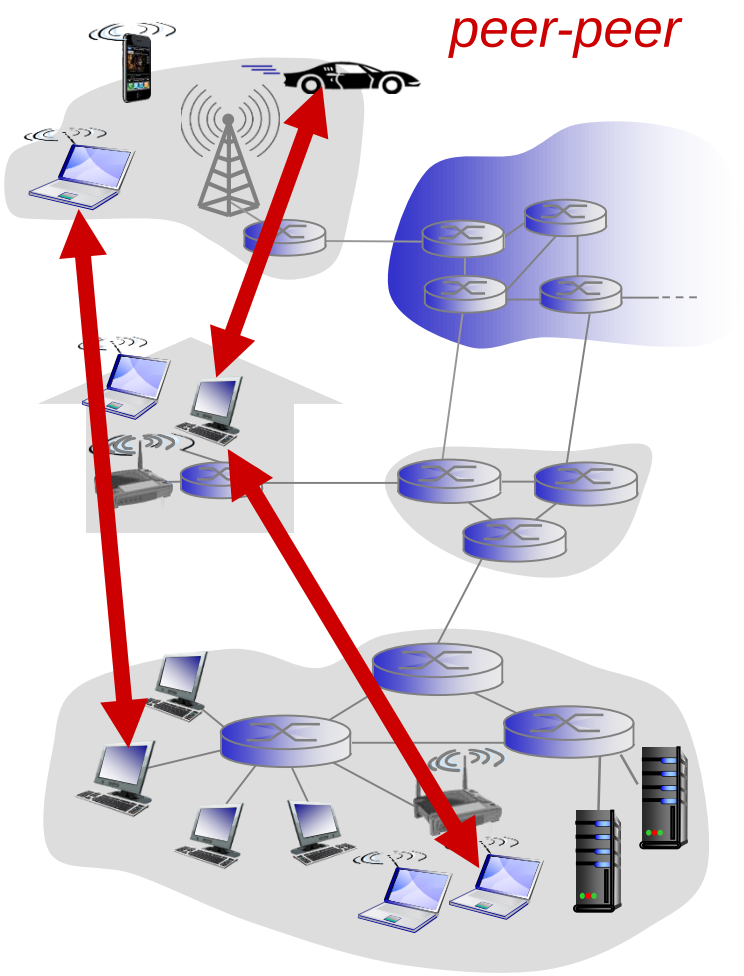
clientes:

- comunicam-se com servidor
- podem se conectar de forma intermitente
- podem ter endereços IP dinâmicos
- não se comunicam diretamente entre si

Web, FTP, Telnet, e-mail,...

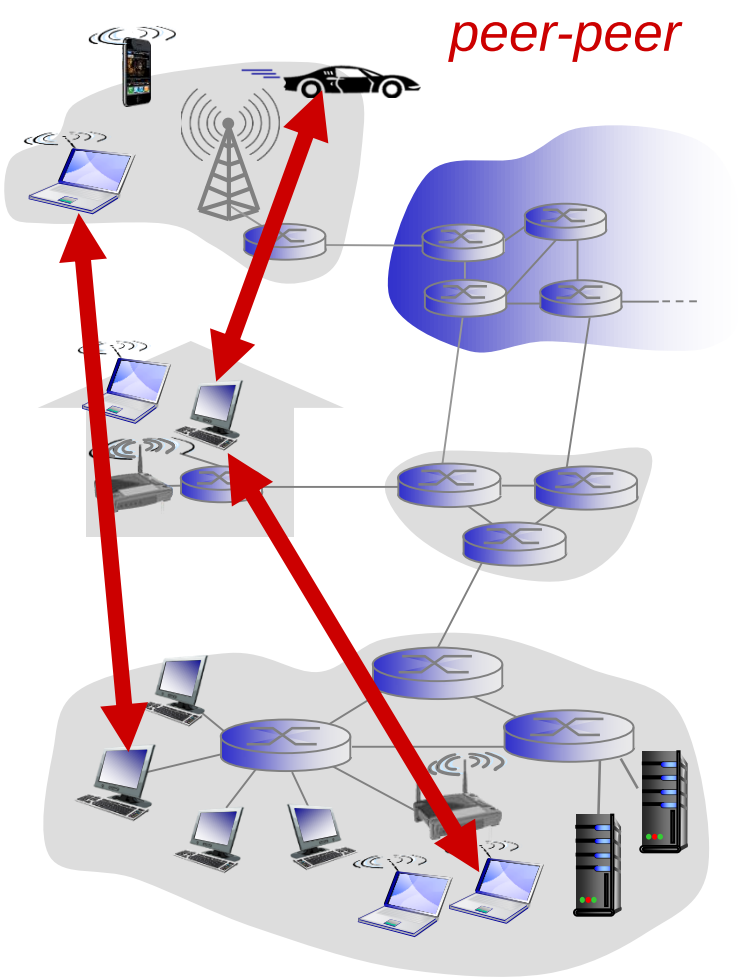
Arquitetura P2P

- *não há* servidor sempre ligado
- sistemas finais arbitrários (*peers*) comunicam-se diretamente
- *peers* requerem serviço de outros *peers* – provêm serviço para outros *peers* em retorno
  - *autoescalável* – novos *peers* trazem novos serviços, assim como novas demandas de serviço
- *peers* conectam-se de forma intermitente e mudam endereço IP
  - gerenciamento complexo
- BitTorrent, Skype, ...



Arquitetura P2P

- *Arquitetura crescente*
- *Problemas:*
  - *ISPs residenciais são assimétricos*
  - *Segurança*
  - *Motivação: por que eu vou ceder largura de banda, armazenamento, etc.?*
- Estruturas híbridas.
- WhatsApp é P2P ou cliente servidor???



Comunicação entre processos

processo: programa rodando em um *host*

- dentro do mesmo *host*, 2 processos se comunicam usando comunicação inter processos (definida pelo OS)
- processos em *hosts* diferentes comunicam-se trocando mensagens

clientes, servidores

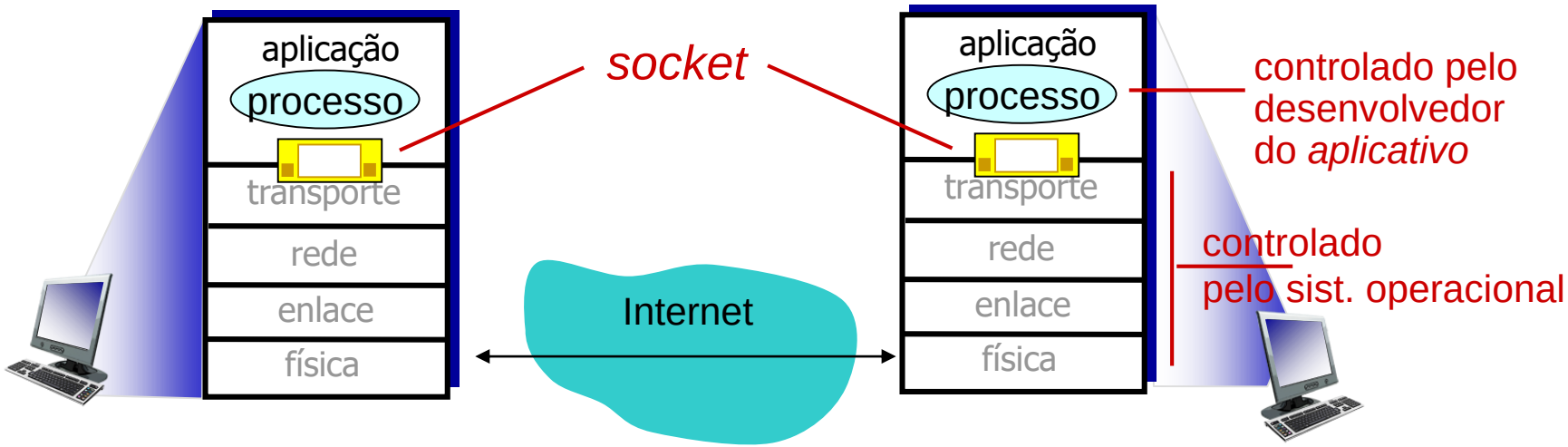
processo cliente: processo que inicia comunicação

processo servidor: processo que espera ser contatado

- nota: aplicativos com arquitetura P2P também tem processo cliente e processo servidor

Sockets

- processo envia/recebe mensagens de/para uma interface de *software* (API - *Application Programming Interface*) chamada *socket*
- *socket* análogo a porta
  - processo transmissor empurra mensagem pela porta
  - processo transmissor confia na infraestrutura de transporte do outro lado da porta para entregar mensagem ao *socket* no processo receptor



Endereçamento de processos

- para receber mensagens, processo precisa ter *identificador*
- dispositivo *host tem endereço* IP *de* 32 bits único
- *Q:* Endereço IP do *host* em que o processo roda é suficiente para identificar o processo?
  - *A:* Não! *Muitos* processos podem estar rodando no mesmo *host*!

- *identificador* inclui tanto endereço IP (32 bits) quanto número da porta associado com processo no *host*.
- exemplos de números de porta:
  - servidor HTTP: 80
  - servidor de *e-mail*: 25
  - Internet Assigned Numbers Authority (IANA)
  - http://www.iana.org
- para enviar mensagem HTTP para servidor *web* www.usp.br:
  - endereço IP: 200.144.183.244
  - número de porta: 80
- mais em breve…

Quais serviços de transporte um *app* precisa?

integridade dos dados

- alguns *apps* (e.g., transferência de arquivos, comércio na *web*) requerem transferência de dados 100% confiável *(reliable data transfer)*
- outros *apps* (e.g., áudio) podem tolerar alguma perda

timing

- alguns *apps* (e.g., telefonia Internet, jogos iterativos) requerem baixa latência para serem "efetivos"

vazão

- alguns *apps* (e.g., multimídia) requerem mínima vazão (bits/segundo) para serem "efetivos"
- outros *apps* ("*apps* elásticos") fazem uso de qualquer vazão disponível

segurança

- encriptação, integridade de dados, …

Requisitos do serviço de transporte: *apps* comuns

aplicativo	perda de dados	vazão	*timing*
transferência de arquivos	sem perdas	elástico	não
e-mail	sem perdas	elástico	não
páginas Web	sem perdas	elástico	não
áudio/vídeo em tempo real	tolerante	áudio: 5kbps-1Mbps vídeo:10kbps-5Mbps	sim, 100’s ms
áudio/vídeo armazenado	tolerante	mesmo acima	não
jogos interativos	tolerante	alguns kbps ou mais	sim, alguns s
mensagem de texto	sem perdas	elástico	sim, 100’s ms
			sim e não

Serviços dos protocolos de transporte Internet

serviços TCP:

- *transporte confiável* entre processos transmissor e receptor – dados recebidos sem erro e na ordem enviada
- *controle de fluxo:* transmissor não "soterra" receptor
- *controle de congestionamento:* regula transmissor quando rede está sobrecarregada
- *orientado a conexão:* requer *setup* entre processos cliente e servidor
- *não provê: timing*, garantia de mínima vazão, segurança

serviços UDP :

- *transferência de dados não confiável* entre processos transmissor e receptor
- *não provê:* integridade, controle de fluxo, controle de congestionamento, *timing*, garantia de vazão, segurança, ou *setup* de conexão

Q: Para que serve o UDP??

Apps internet : protocolos de aplicativos e transport

aplicativo	protocolo da camada de aplicação	protocolo da camada de transporte
e-mail	SMTP [RFC 2821]	TCP
aceso a terminal remoto	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
transferência de arquivos	FTP [RFC 959]	TCP
streaming de multimídia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP ou UDP
telefonia Internet	SIP, RTP, proprietários (e.g., Skype)	TCP ou UDP

Tornando o TCP seguro

TCP e UDP

- sem encriptação
- senhas não criptografadas enviados pelo *socket* atravessam a Internet não criptografadas

SSL (*Secure Sockets Layer*)

- provê conexão TCP criptografada
- integridade dos dados
- autenticação *end-point*

SSL é um *aplicativo* da camada de aplicação

- *Aplicativos* usam bibliotecas SSL, que "falam" com o TCP

socket API SSL

- Aplicativos passam dados para *socket* SSL que por sua vez passa para *socket* TCP
- Usado para *web*, e-mail, etc...

Protocolo da camada *app* define

- tipos de mensagens trocadas,
  - e.g., requisição, resposta
- sintaxe da mensagem:
  - campos da mensagem e como eles são delineados
- semântica das mensagens
  - significado das informações dos campos
- regras para quando e como processos enviam e respondem mensagens

protocolos abertos:

- definidos em RFCs
- permitem interoperabilidade
- Exemplos:
  - HTTP [RFC 2616]
  - SMTP [RFC 5321]

protocolos proprietários:

- e.g., Skype

Capítulo 2: conteúdo

2.1 Princípios de aplicativos de rede

2.2 *Web* e HTTP

2.3 Correio eletrônico

- SMTP, POP3, IMAP

2.4 DNS

2.5 Aplicativos P2P

2.6 *Streaming* de vídeo e redes de distribuição de conteúdo

2.7 Programando *socket* com UDP e TCP

Web e HTTP

Primeiro, uma revisão…

- *página web* consiste de *objetos (arquivos)*
- objeto pode ser arquivo HTML, imagem JPEG, *applet* Java, arquivo de áudio,…
- página *web* consiste de *arquivo HTML base* que inclui *diversos objetos referenciados*
- cada objeto é endereçável por uma *URL (Uniform Resource Locator),* e.g.,

`www.lcs.poli.usp.br/~marcio/index_arquivos/image002.jpg`

nome do *host* | local do objeto