PTC 3450 - Aula 09

2.4 DNS – O serviço de diretório da Internet

2.6 Aplicativos P2P

2.7 Programação de socket: criando aplicações de rede

(Kurose, p. 106 - 120)

(Peterson, p. 239-242)

18/04/2017

Capítulo 2: conteúdo

2.1 Princípios de aplicativos de rede

2.2 *Web* e HTTP

2.3 Correio eletrônico

- SMTP, POP3, IMAP

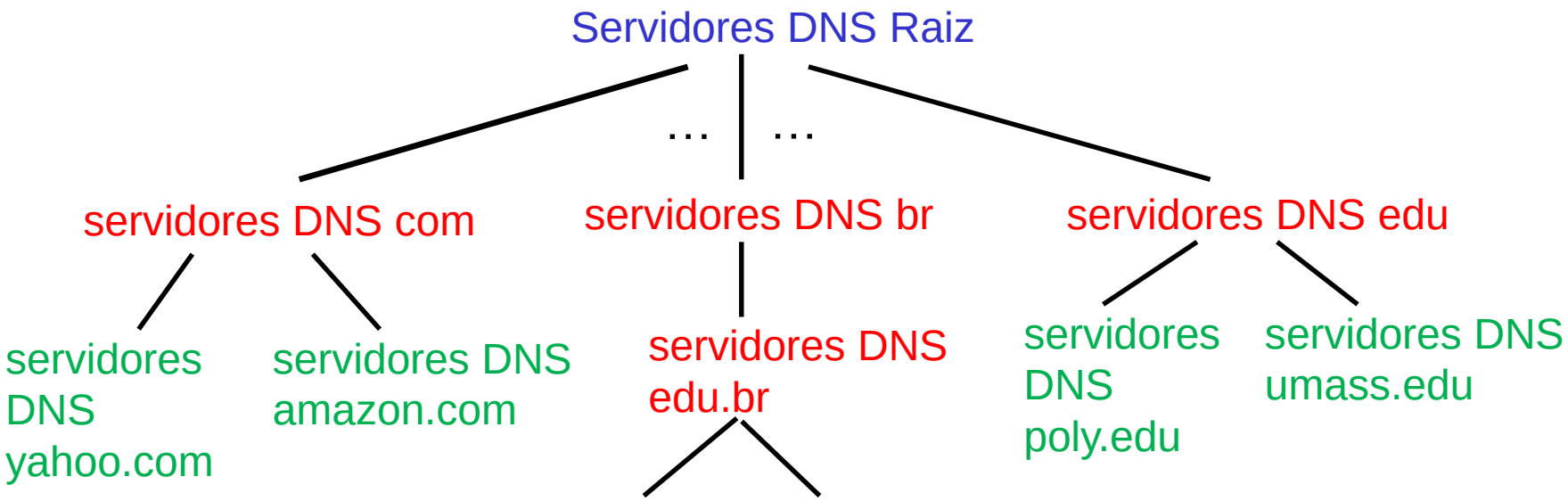
2.4 DNS

2.5 Aplicativos P2P

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

2.7 Programando *socket* com UDP e TCP

DNS: banco de dados hierárquico e distribuído



cliente quer IP para www.amazon.com; 1ª abordagem:

- cliente pede a servidor DNS raiz encontrar servidor DNS com
- cliente pede a servidor DNS *top level domain (TLD)* .com obter servidor amazon.com
- cliente pede a servidor DNS autoritário de amazon.com obter endereço IP para www.amazon.com

Atacando DNS

Se DNS não funcionar Internet (*web*, email, etc.) para!! ☹

ataques DDoS

- Bombardear servidores raízes com tráfego
  - Tentado em 21/10/2002
  - Filtragem do tráfego
  - Servidores DNS locais guardam IPs dos servidores TLD, permitindo pular servidores raízes
- Bombardear servidores TLD
  - Potencialmente mais perigoso

Ataques de redirecionamento

- *Man-in-the-middle*
  - Consultas interceptadas
- envenenar DNS
  - Enviar respostas falsas a servidores DNS, que as armazenam

Explorar DNS para DDoS

- Enviar consultas com endereço do emissor falso: IP alvo
- Requer amplificação

Até hoje robusto a ataques. Não conseguiu se impedir DNS de funcionar.

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

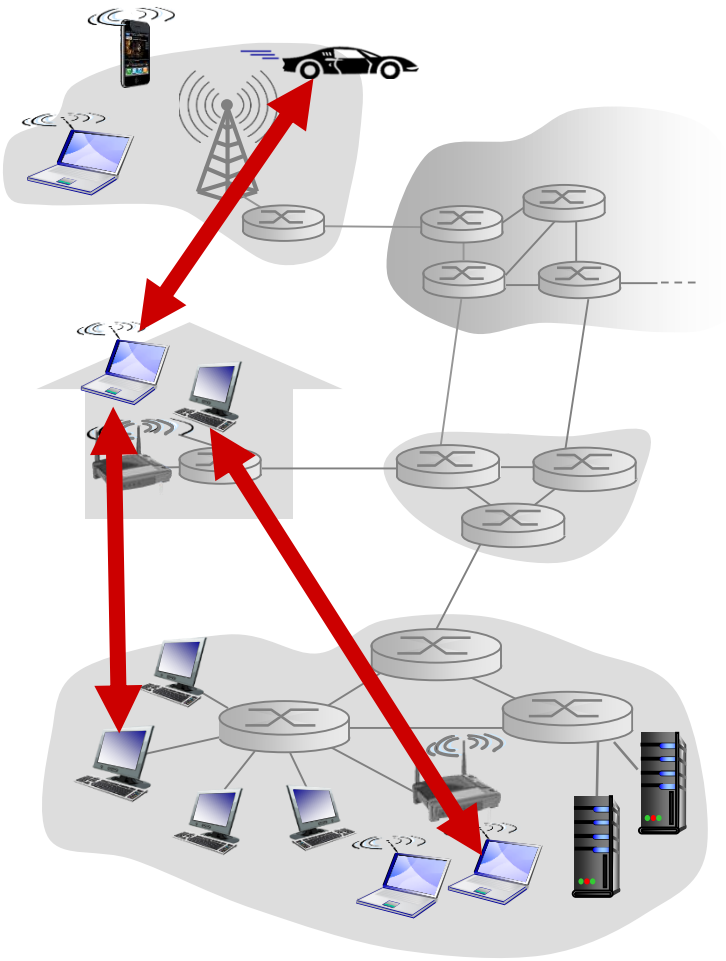
2.5 Aplicativos P2P

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

2.7 Programando *socket* com UDP e TCP

Arquitetura P2P pura

- *não há* servidor sempre disponível
- sistemas finais arbitrários (pares ou *peers*) comunicam-se diretamente
- pares conectam-se de forma intermitente e mudam endereços IP



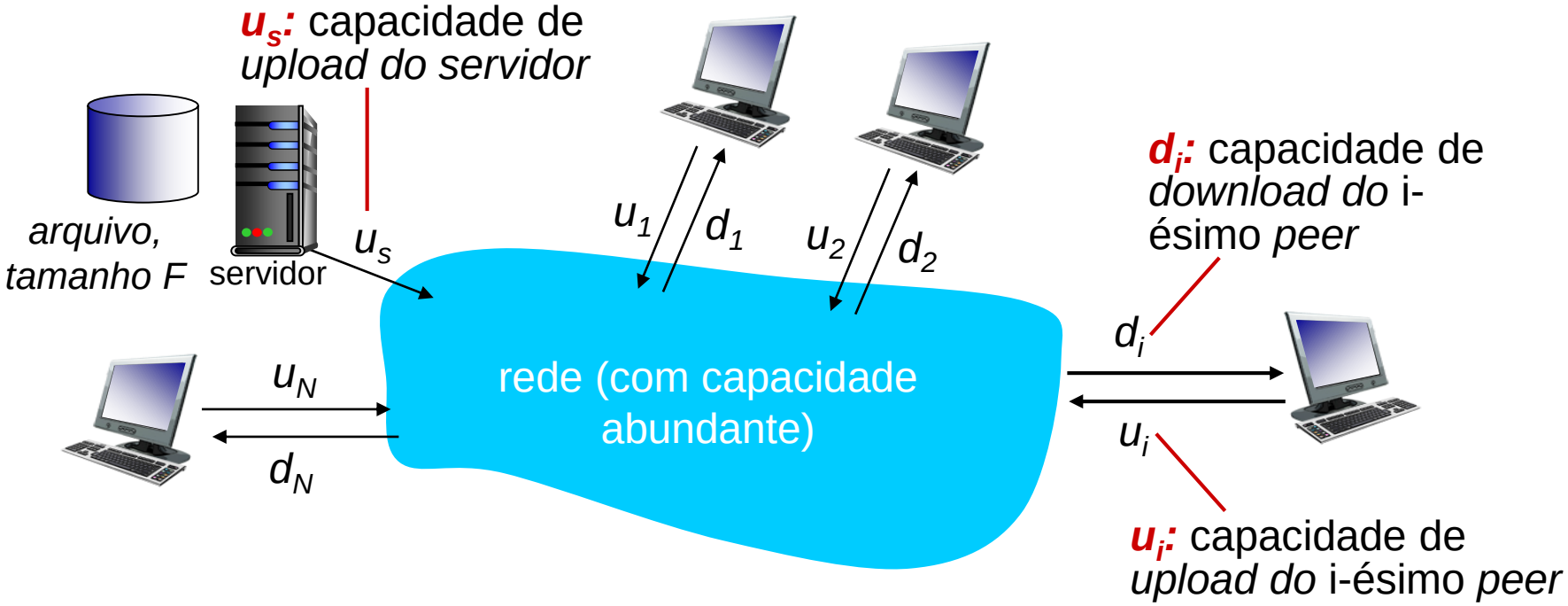
exemplos:

- distribuição de arquivos (BitTorrent)
- *Streaming* (Xunlei KanKan, Popcorn Time)
- VoIP (Skype)

Distribuição de arquivos: cliente-servidor vs P2P

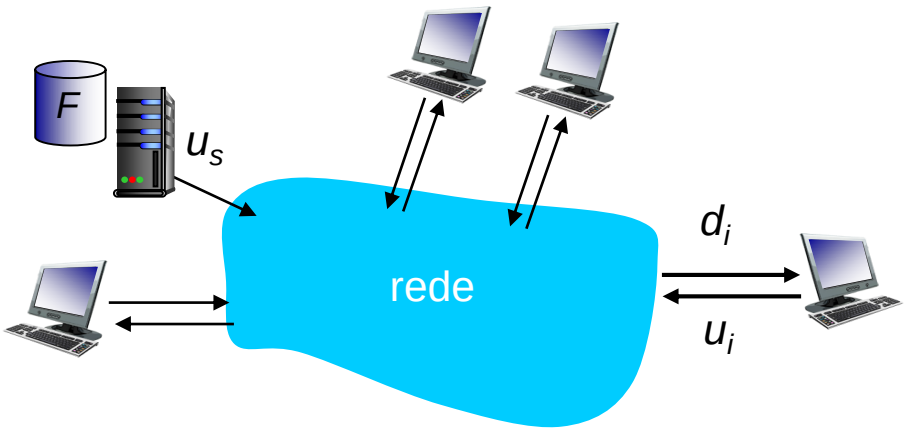
Questão: quanto tempo para distribuir arquivo (tamanho *F*) de um servidor para *N* *peers* (tempo de distribuição)?

- capacidade de *upload/download* dos *peers* é recurso limitado



Tempo de distribuição de arquivo: cliente-servidor

- *transmissão do servidor :* precisa enviar (*upload*) sequencialmente *N* cópias do arquivo:
  - tempo para enviar uma cópia: F/u_s
  - tempo para enviar N cópias: NF/u_s



- *cliente:* cada cliente precisa fazer *download* de cópia do arquivo
  - d_{min} = taxa de *download* mínima
  - mínimo tempo de *download nos* clientes : F/d_{min}

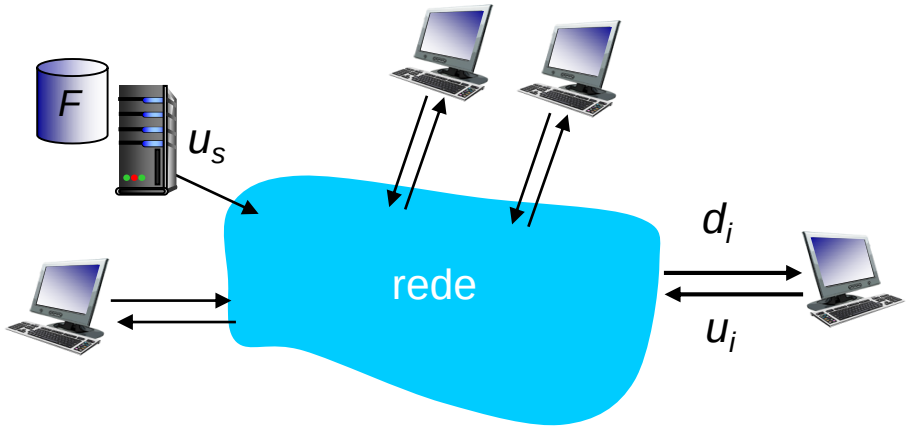
tempo de distribuição para N clientes usando abordagem cliente-servidor

$$D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$$

aumenta linearmente com N

Tempo para distribuição de arquivo: P2P

- *transmissão servidor :* precisa fazer *upload* de pelo menos uma cópia
  - tempo para enviar uma cópia: F/u_s
- *cliente:* cada cliente precisa fazer *download* de uma cópia do arquivo
  - mínimo tempo de *download nos clientes*: F/d_{min}



- *clientes:* de forma agregada precisam fazer *download* de *NF* bits
  - max taxa de *upload* (limitando max taxa de *download*) é $u_s + \Sigma u_i$

tempo de distribuição para N clientes usando abordagem P2P

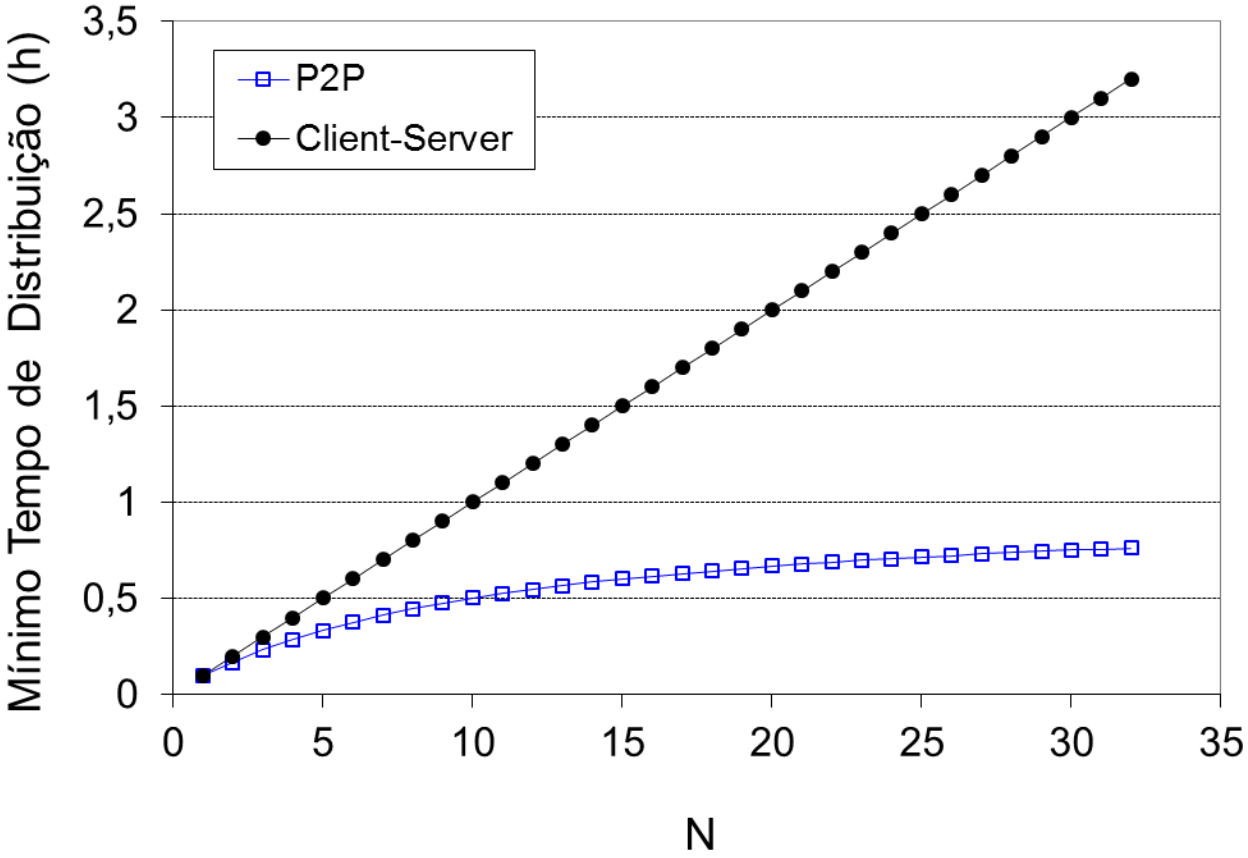
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

cresce linearmente com *N* …

… mas essa parcela também, já que cada *peer* traz sua capacidade de serviço

Exemplo: Cliente-servidor vs. P2P

taxa *upload* cliente = u, F/u = 1 hora, $u_s = 10u$, $d_{min} \geq u_s$

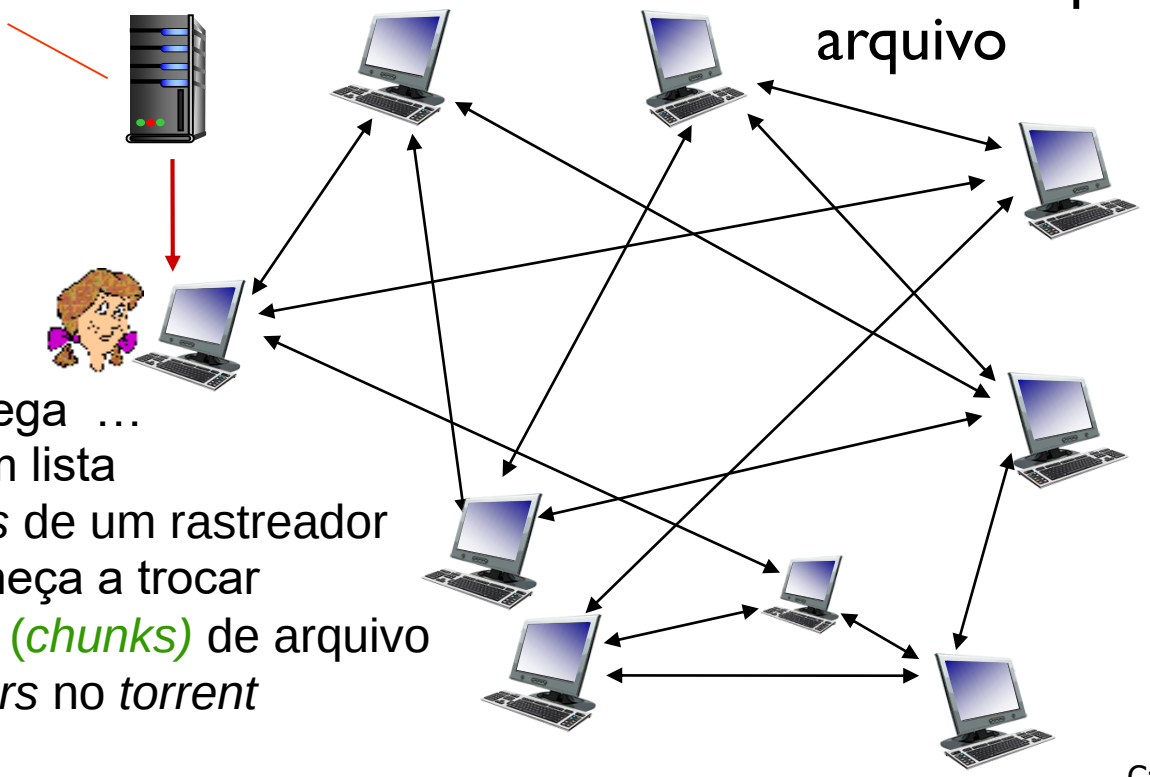


Distribuição de arquivos P2P: BitTorrent

- *torrent* - *peers* enviando/recebendo pedaços de um arquivo
- arquivo dividido em pedaços de 256 kbytes (típico)

rastreador (tracker): rastreia *peers* participando em *torrent*

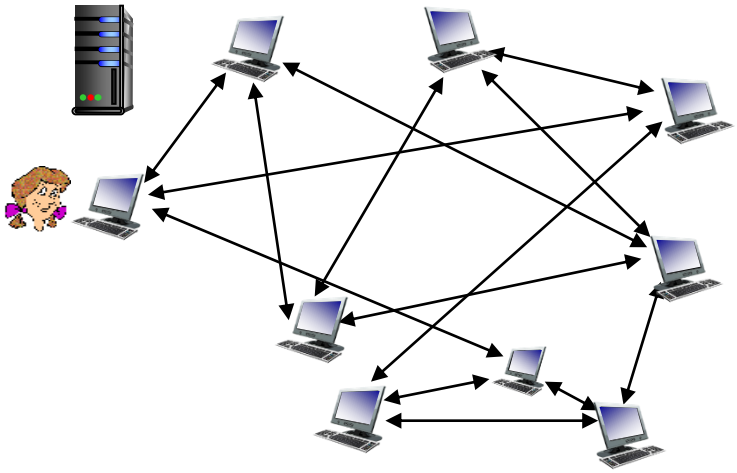
torrent: grupo de *peers* trocando pedaços de um arquivo



Alice chega …
… obtém lista de *peers* de um rastreador
… e começa a trocar pedaços (*chunks*) de arquivo com *peers* no *torrent*

Distribuição de arquivos P2P: BitTorrent

- *peer* juntando-se a *torrent*:
  - ainda não tem pedaços, mas os acumulará com o tempo obtendo de outros *peers*
  - registra-se com rastreador para obter lista de *peers* (IP), conecta-se a subconjunto deles (típico 50)



- enquanto faz *download*, *peer* faz *upload* de pedaços para outros *peers*
- *peer* pode mudar *peers* com quem ele troca pedaços
- *churn:* *peers* podem entrar e sair
- uma vez que *peer* tem arquivo inteiro, ele pode escolher (egoisticamente) sair ou (altruisticamente) permanecer no *torrent*

µTorrent 3.3.2

Arquivo Opções Ajuda

Busca na Web

Featured Torrent
Jogo Grátis:Pirate Storm 2.63 MB Ready for Download Instant Download

- Conteúdo em Destaque
- Torrents (1)
  - Baixando (1)
  - Enviando (0)
  - Concluído (0)
  - Ativo (1)
  - Inativo (0)
  - Categorias
- Feeds (0)
- Dispositivos (4)
  - Samsung Galaxy Tab
  - Samsung Galaxy Tab
  - Samsung Galaxy Tab
  - Samsung Galaxy Tab

n.	Nome	Taman...	Status	Disp...	Veloc. Dow...	Veloc. Up	TED	Avaliação	Reprodução	Seeds/...	Categoria	Adicionado	Concluído em
1	Back.to.the.Future.Trilogy.Complete.Bluray.1080p.DTS-HD.x264-Grym.(@BTNET)	51.6 GB	Baixando 71.2%		31.9 kB/s	106.0 kB/s	6d 8h	★★★★★	Stream	0.256		3 dias atrás	

Arquivos Info Peers Avaliações Rastreadores Velocidade

IP	Cliente	Flags	%	Veloc. Down.	Veloc. Up	Requisições	Enviado	Baixado	DI. Peer
dh207-110-228.xnet.hr	Vuze 5.3.0.0	UD Xe	37.0	17.6 kB/s	67.7 kB/s	6 | 0	13.3 MB	1.71 MB	796.4 k...
bb424df3.virtua.com.br [uTP]	BitTorrent 7.8.2	UD H...	2.7	0.9 kB/s	2.0 kB/s	3 | 13	5.50 MB	448 kB	159.2 k...
mal35-1-82-67-110-189.fbx.proxad.net	BitComet 1.36	uD XE	90.3	9.2 kB/s		5 | 0		368 kB	68.2 kB/s
123.114.83.135	Tixati 1.96	UD HX	83.4	0.7 kB/s	1.3 kB/s	2 | 0	608 kB	96.0 kB	83.4 kB/s
cpc32-bolt14-2-0-cust250.10-3.cable.virginm....	Transmission 2.82	uD XEP	49.9	2.2 kB/s		3 | 0	16.0 kB	80.0 kB	68.2 kB/s
client-201.230.79.64.speedy.net.pe	BitComet 1.35	UD HX	1.2		31.8 kB/s	2 | 0	2.14 MB	64.0 kB	30.3 kB/s
host217-42-12-236.range217-42.btcentralplus....	µTorrent 3.3.1	udS ...	59.8				1.87 MB	32.0 kB	1.0 MB/s
110.78.169.198 [uTP]	Transmission 2.31	D XEP	100.0	0.7 kB/s		2 | 0		16.0 kB	
nas1149321.lnk.telstra.net [uTP]	µTorrent 3.3.2	udS ...	28.6				112 kB	16.0 kB	22.7 kB/s
0542e0dc.skybroadband.com [uTP]	µTorrent Mac 1.8.4	ud H...	60.1						
S010628c68ea24914.wp.shawcable.net [uTP]	µTorrent Mac 1.8.4	Ud H...	1.7		1.0 kB/s		32.0 kB		
c-2ec207e7-74736162.cust.telenor.se [uTP]	BitTorrent 7.8	ud H...	18.3						
cpe-68-203-84-202.tx.res.rr.com [uTP]	BitTorrent 7.4	ud H...	23.8						
c-76-101-135-83.hsd1.fl.comcast.net [uTP]	µTorrent Mac 1.8.4	ud H...	47.1						
cpe-76-173-120-140.socal.res.rr.com [uTP]	BitTorrent 7.8.2	d HXP	100.0						
cpc20-walt13-2-0-cust98.13-2.cable.virginm.n...	µTorrent 3.3.1	HXEP	0.0						
a88-114-60-242.elisa-laajakaista.fi [uTP]	Vuze 5.2.0.0	d IXEP	100.0						
94-159-244-128.orange.net.il [uTP]	µTorrent 3.3.2	d HX...	100.0						
95.0.196.154.dynamic.ttnet.com.tr [uTP]	BitTorrent 7.8.2	ud XeP	96.2						
pool-108-30-204-22.nycmny.fios.verizon.net [...	BitTorrent 7.8.2	Ud H...	43.0		0.7 kB/s	0 | 2	80.0 kB		
113.232.67.6 [uTP]	µTorrent 3.3.2	d HXP	100.0						
115.223.24.5 [uTP]	µTorrent 3.3.2	ud XP	4.9				16.0 kB		
fm-dyn-139-194-85-133.fast.net.id [uTP]	µTorrent Mac 1.9.1	ud H...	78.8						
142-197-222-104.res.bhn.net [uTP]	µTorrent 3.3.2	ud H...	8.0		1.0 kB/s		96.0 kB		204.7 k...
pool-173-70-138-66.nwrknj.fios.verizon.net [u...	µTorrent 3.3.2	d HXP	97.9						

DHT: 284 nós D: 33.4 kB/s T: 3.1 MB U: 115.6 kB/s T: 24.1 MB

BitTorrent: pedindo, enviando pedaços de arquivo

pedindo pedaços:

- em qualquer momentos, *peers* diferentes têm subconjuntos diferentes de pedaços do arquivo
- periodicamente, Alice pede a cada *peer* uma lista de pedaços que eles têm
- Alice pede pedaços faltantes aos *peers*, pedaços mais raros primeiro (*rarest first*) – assim pedaços ficam distribuídos uniformemente

enviando pedaços: tit-for-tat

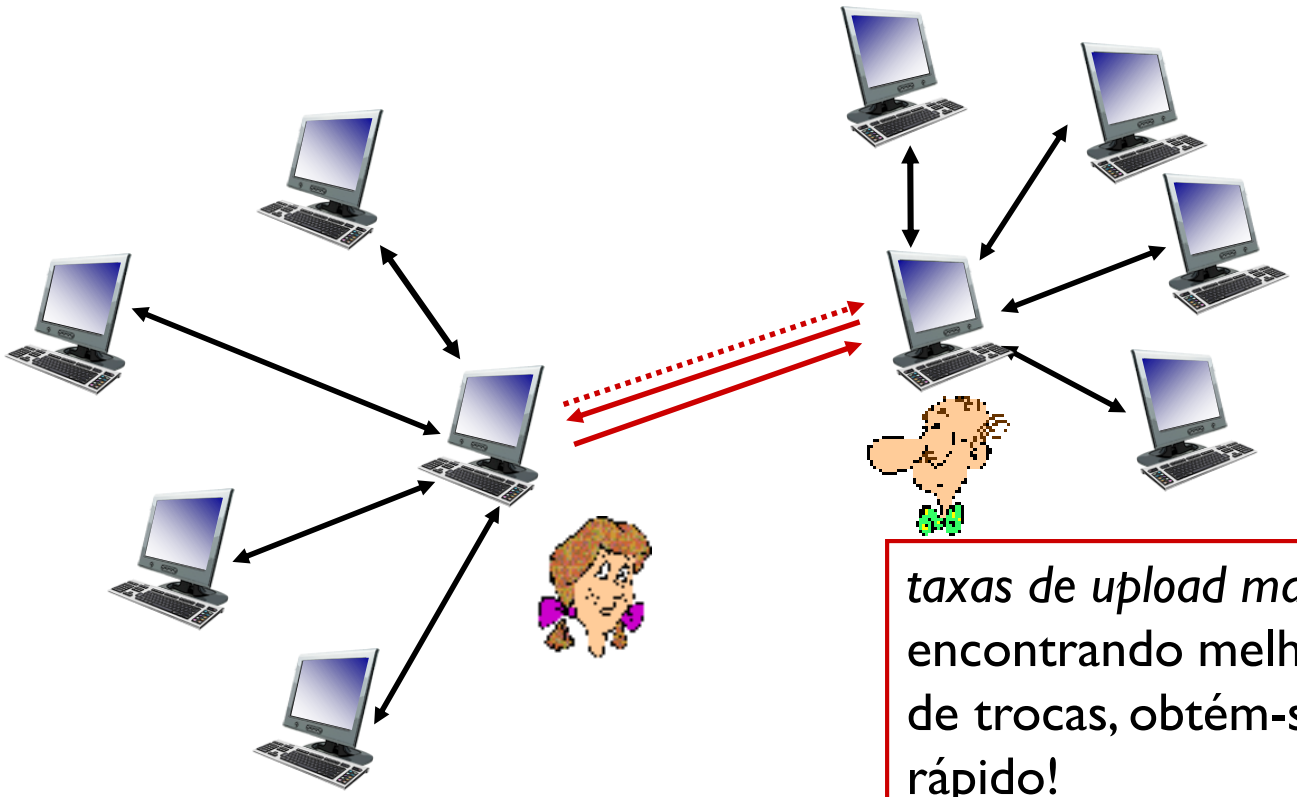
- Alice envia pedaços para os 4 *peers* atualmente enviando pedaços para ela à taxa mais alta
  - outros *peers* são congelados (*choked*) por Alice (não recebem pedaços dela)
  - top 4 reavaliados a cada 10 s
- a cada 30 s: seleciona aleatoriamente outro *peer*, inicia envio de pedaços
  - "otimistamente descongela" esse *peer*
  - novo *peer* escolhido pode-se juntar aos top 4

BitTorrent: tit-for-tat

(1) Alice "otimistamente descongela" Bob

(2) Alice torna-se um dos top-4 provedores de Bob; Bob retribui

(3) Bob torna-se um dos top-4 provedores de Alice



taxas de upload mais altas: encontrando melhores parceiros de trocas, obtém-se arquivo mais rápido!

Muitos outros mecanismos interessantes a estudar! Veja referências no livro!

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

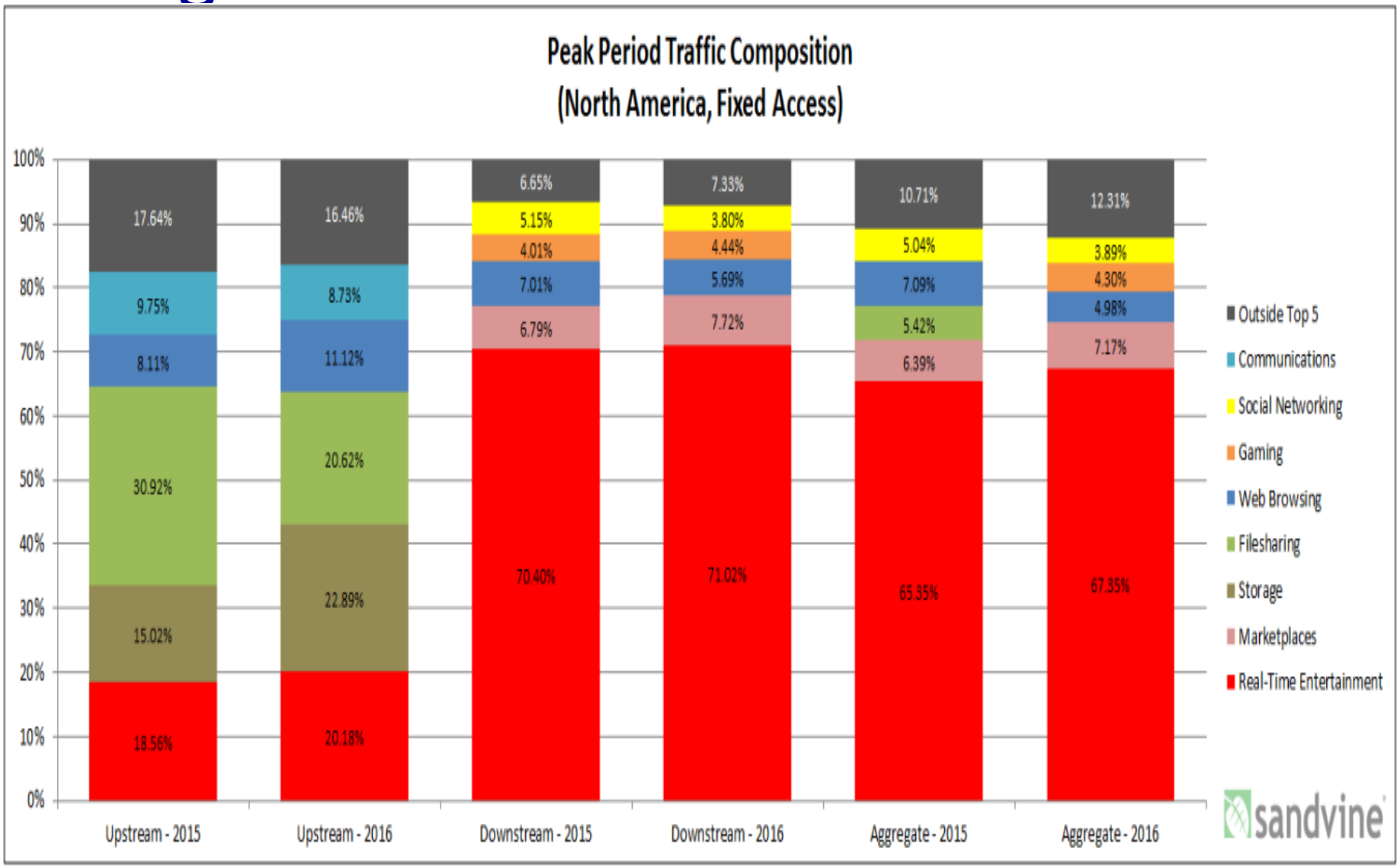
2.6 *Streaming* de vídeo e redes de distribuição de conteúdo

2.7 Programando *socket* com UDP e TCP

Video Streaming and CDNs: contexto

- Tráfego de vídeo : maior consumidor de banda da Internet
  - Netflix 35,15% e YouTube 17,53% do tráfego *downstream* em ISPs residenciais nos EUA (Sandvine report)
  - Netflix ~75milhões de usuários; YouTube: 1 bilhão de usuários
- Desafio: escala - como alcançar 1 bilhão de usuários???
  - Único super-servidor de vídeo não funcionará (por que?)
- Desafio: heterogeneidade
  - Usuários diferentes têm diferentes capacidades (e.g., móvel versus cabeado; banda larga versus acesso de baixa qualidade)
- *Solução:* infraestrutura distribuída em nível de aplicação

Tráfego Rede Fixa – EUA - 2016

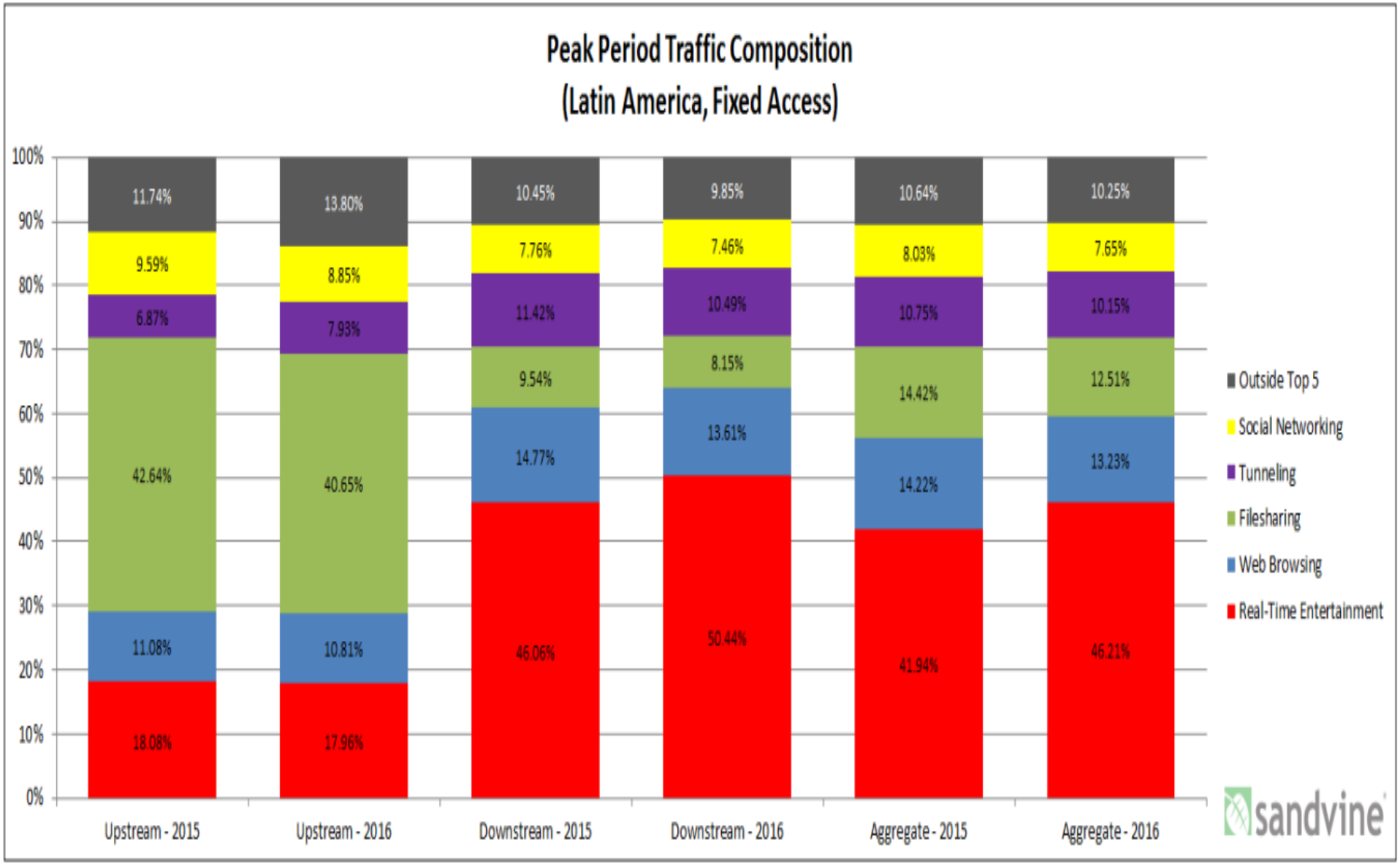


Tráfego Rede Fixa – EUA – 2016

Upstream		Downstream		Aggregate	
BitTorrent	18.37%	Netflix	35.15%	Netflix	32.72%
YouTube	13.13%	YouTube	17.53%	YouTube	17.31%
Netflix	10.33%	Amazon Video	4.26%	HTTP - OTHER	4.14%
SSL - OTHER	8.55%	HTTP - OTHER	4.19%	Amazon Video	3.96%
Google Cloud	6.98%	iTunes	2.91%	SSL - OTHER	3.12%
iCloud	5.98%	Hulu	2.68%	BitTorrent	2.85%
HTTP - OTHER	3.70%	SSL - OTHER	2.53%	iTunes	2.67%
Facebook	3.04%	Xbox One Games Download	2.18%	Hulu	2.47%
FaceTime	2.50%	Facebook	1.89%	Xbox One Games Download	2.15%
Skype	1.75%	BitTorrent	1.73%	Facebook	2.01%
	69.32%		74.33%		72.72%

sandvine

Tráfego Rede Fixa – América Latina - 2016

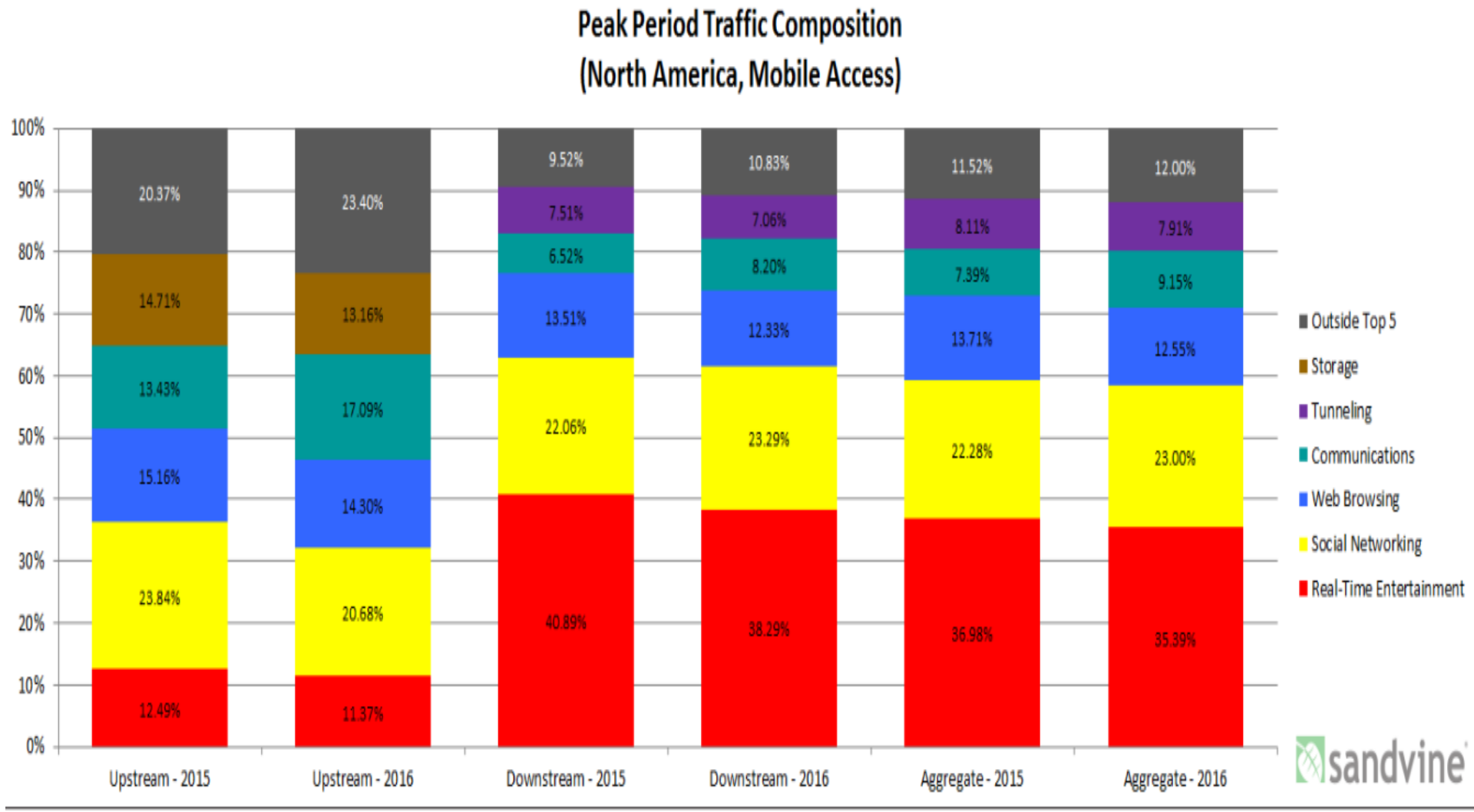


Tráfego Rede Fixa – América Latina - 2016

Upstream		Downstream		Aggregate	
BitTorrent	30.03%	YouTube	28.48%	YouTube	25.91%
YouTube	9.30%	HTTP - OTHER	11.66%	HTTP - OTHER	11.12%
HTTP - OTHER	7.59%	SSL - OTHER	9.76%	BitTorrent	10.06%
Facebook	6.72%	Netflix	8.31%	SSL - OTHER	9.28%
SSL - OTHER	6.19%	BitTorrent	6.96%	Netflix	7.45%
Ares	5.27%	Facebook	5.10%	Facebook	5.32%
Skype	2.53%	MPEG - OTHER	2.28%	MPEG - OTHER	2.10%
Netflix	1.97%	RTMP	1.79%	RTMP	1.66%
Dropbox	1.16%	Google Market	1.69%	Google Market	1.52%
MPEG - OTHER	0.92%	Flash Video	1.60%	Flash Video	1.46%
	71.69%		77.63%		75.87%

sandvine

Tráfego Móvel – EUA – 2016

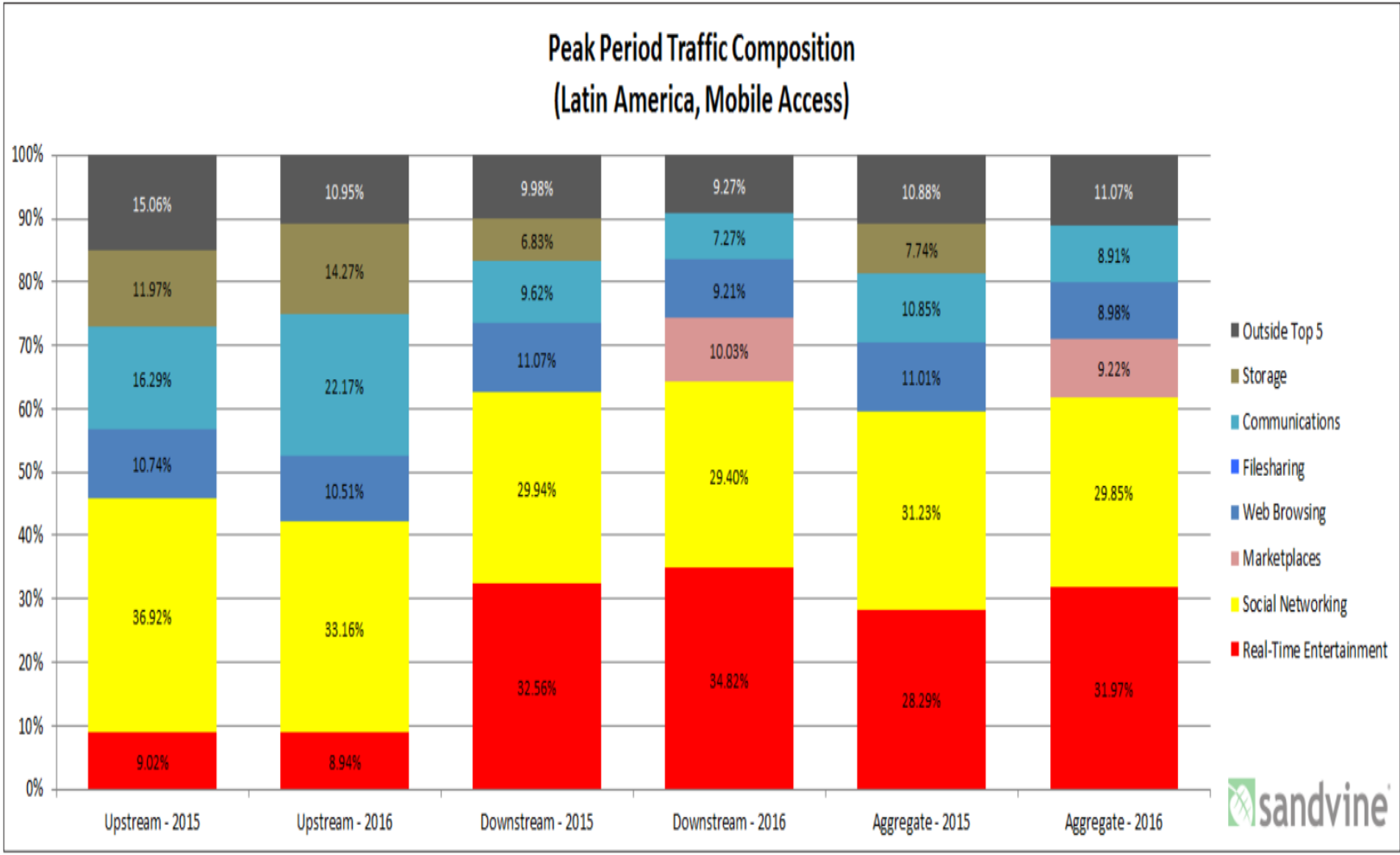


Tráfego Móvel – EUA -2016

Rank	Upstream	2016	Downstream		Aggregate	Share
1	Facebook	14.85%	YouTube	20.87%	YouTube	19.16%
2	SSL - OTHER	14.02%	Facebook	13.97%	Facebook	14.07%
3	Google Cloud	9.28%	HTTP - OTHER	9.36%	HTTP - OTHER	9.32%
4	HTTP - OTHER	8.92%	SSL - OTHER	6.85%	SSL - OTHER	7.62%
5	YouTube	5.01%	Instagram	6.66%	Instagram	6.31%
6	Snapchat	4.36%	Snapchat	5.17%	Snapchat	5.09%
7	Instagram	3.35%	Netflix	3.72%	Google Cloud	3.56%
8	BitTorrent	2.16%	iTunes	3.02%	Netflix	3.41%
9	FaceTime	1.97%	Google Cloud	2.87%	iTunes	2.86%
10	iCloud	1.82%	MPEG - OTHER	2.37%	MPEG - OTHER	2.17%
		65.76%		74.87%		73.57%

sandvine

Tráfego Móvel – América Latina - 2016



Tráfego Móvel – América Latina - 2016

Upstream		Downstream		Aggregate	
Facebook	30.49%	YouTube	26.09%	YouTube	23.91%
WhatsApp	15.76%	Facebook	22.92%	Facebook	23.55%
Google Cloud	11.96%	HTTP - OTHER	8.00%	HTTP - OTHER	7.70%
YouTube	6.18%	WhatsApp	7.98%	WhatsApp	7.43%
SSL - OTHER	5.94%	Instagram	4.91%	Google Market	5.85%
HTTP - OTHER	5.26%	Google Market	4.64%	Instagram	4.65%
Instagram	2.55%	MPEG - OTHER	4.46%	Google Cloud	4.41%
Google Market	1.57%	Google	3.50%	MPEG - OTHER	4.05%
MPEG - OTHER	0.94%	SSL - OTHER	2.95%	SSL - OTHER	3.27%
Snapchat	0.79%	Snapchat	1.02%	Snapchat	0.98%
	81.44%		86.28%		85.51%

sandvine

Multimídia: vídeo

- Principal característica: alta taxa de bit (até 10 Mbps para 4K!)
- *Streaming* de vídeo armazenado será aproximadamente 70% do consumo global de tráfego de Internet em 2015
- *Usuário menos sensível a falhas de vídeo do que a falhas de áudio*
- vídeo: sequência de imagens mostradas a taxa constante
  - e.g. 24 imagens/s
- imagem digital : matriz de pixels
  - cada pixel representado por bits
- Pode ser comprimido: compromisso qualidade de vídeo x taxa de bit
- codificação: usa redundância *dentro* e *entre* imagens para diminuir # bits usados para codificar imagem
  - espacial (dentro da imagem)
  - temporal (de uma imagem para a seguinte)

exemplo de codificação especial: ao invés de enviar *N* valores para mesma cor (tudo roxo), envia apenas dois valores: valor da cor (*roxo)* e *número de valores repetidos (*N)

quadro *i*

exemplo de codificação temporal: ao invés de enviar quadro complete em i+1, envia apenas diferenlças em relação ao quadro i

quadro *i+1*

Multimídia: vídeo

- *CBR: (constant bit rate):* vídeo codificado a taxa fixa
- *VBR: (variable bit rate):* taxa de codificação muda com quantidade de codificação temporal, espacial
- *exemplos:*
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG2 (DVD) 3-6 Mbps
  - MPEG4 (frequentemente usado na Internet, < 1 Mbps)

exemplo de codificação especial: ao invés de enviar *N* valores para mesma cor (tudo roxo), envia apenas dois valores: valor da cor (*roxo) e número de valores repetidos (*N)

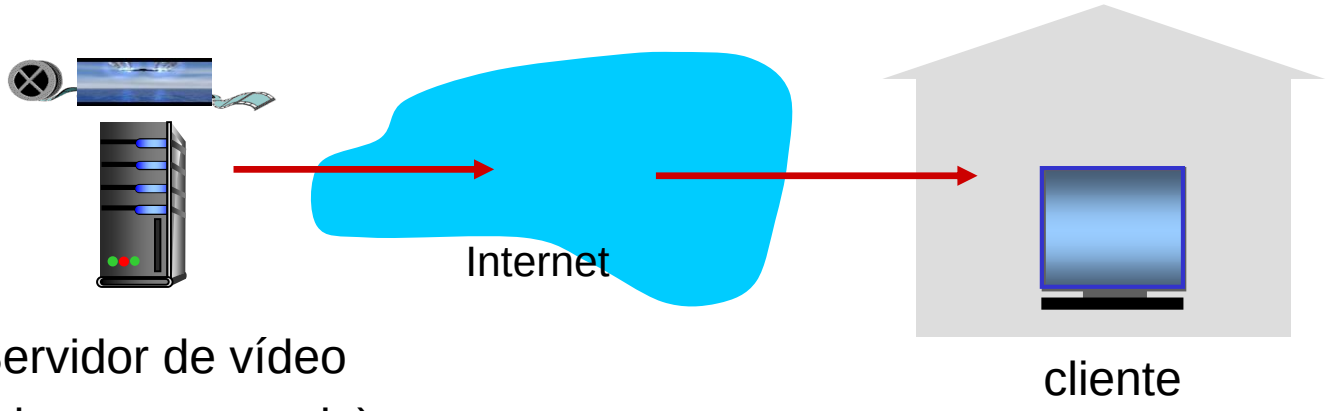
quadro *i*

exemplo de codificação temporal: ao invés de enviar quadro complete em i+1, envia apenas diferenlças em relação ao quadro i

quadro *i+1*

Streaming de vídeo armazenado

Cenário simplifcado:



Streaming multimídia: DASH

- *DASH: Dynamic, Adaptive Streaming over HTTP*
- *servidor:*
  - divide arquivo de vídeo em múltiplos pedaços
  - cada pedaço armazenado, codificado em diferentes taxas
  - *arquivo manifesto:* provê URLs para diferentes pedaços

- *cliente:*
  - periodicamente mede capacidade servidor-para-cliente
  - consultando manifesto, requer um pedaço de cada vez
    - escolhe máxima taxa de codificação sustentável dada a atual capacidade
    - pode escolher diferentes taxas de código em diferentes instantes de tempo (dependendo da banda disponível no momento)

Streaming multimídia: DASH

- *DASH: Dynamic, Adaptive Streaming over HTTP*
- *"inteligência"* no cliente: cliente determina
  - *quando* solicitar pedaço (de modo que esvaziamento do *buffer*, ou estouro não ocorra)
  - *qual taxa de codificação* solicitar (qualidade mais alta quando mais banda disponível)
  - *onde* requisitar pedaço (pode solicitar de servidor URL que está "próximo" ao cliente ou que tem maior largura de banda disponível)
- Usado amplamente nos aplicativos Netflix
  - pedaços de aproximadamente 4s
  - enquanto pedaço está sendo baixado, cliente mede vazão e roda algoritmo para determinar qualidade do próximo pedaço a ser pedido

Redes de Distribuição de Conteúdo (CDNs)

- *desafio:* como distribuir conteúdo (selecionado de milhões de vídeos) para centenas de milhares de usuário simultâneos?
- Exemplo: YouTube
  - Mais de 1 bilhão de usuários visitam todos os meses
  - Mais de 6 bilhões de horas de vídeo são assistidas por mês, quase uma hora para cada pessoa do planeta
  - 100 horas de vídeo são enviadas ao YouTube a cada minuto
  - Países que mais acessam: EUA (19%), Índia (8%), Japão (5%), Rússia (4%), Brasil (4%)
- *opção 1:* "mega-servidor", único, gigantesco
  - ponto único de falha
  - ponto único de congestionamento de rede
  - longo caminho para clientes distantes (maior chance de gargalo)
  - múltiplas copias de vídeo enviadas sobre enlace de saída

….mais simplesmente: essa solução *não escala*

Redes de Distribuição de Conteúdo (CDNs)

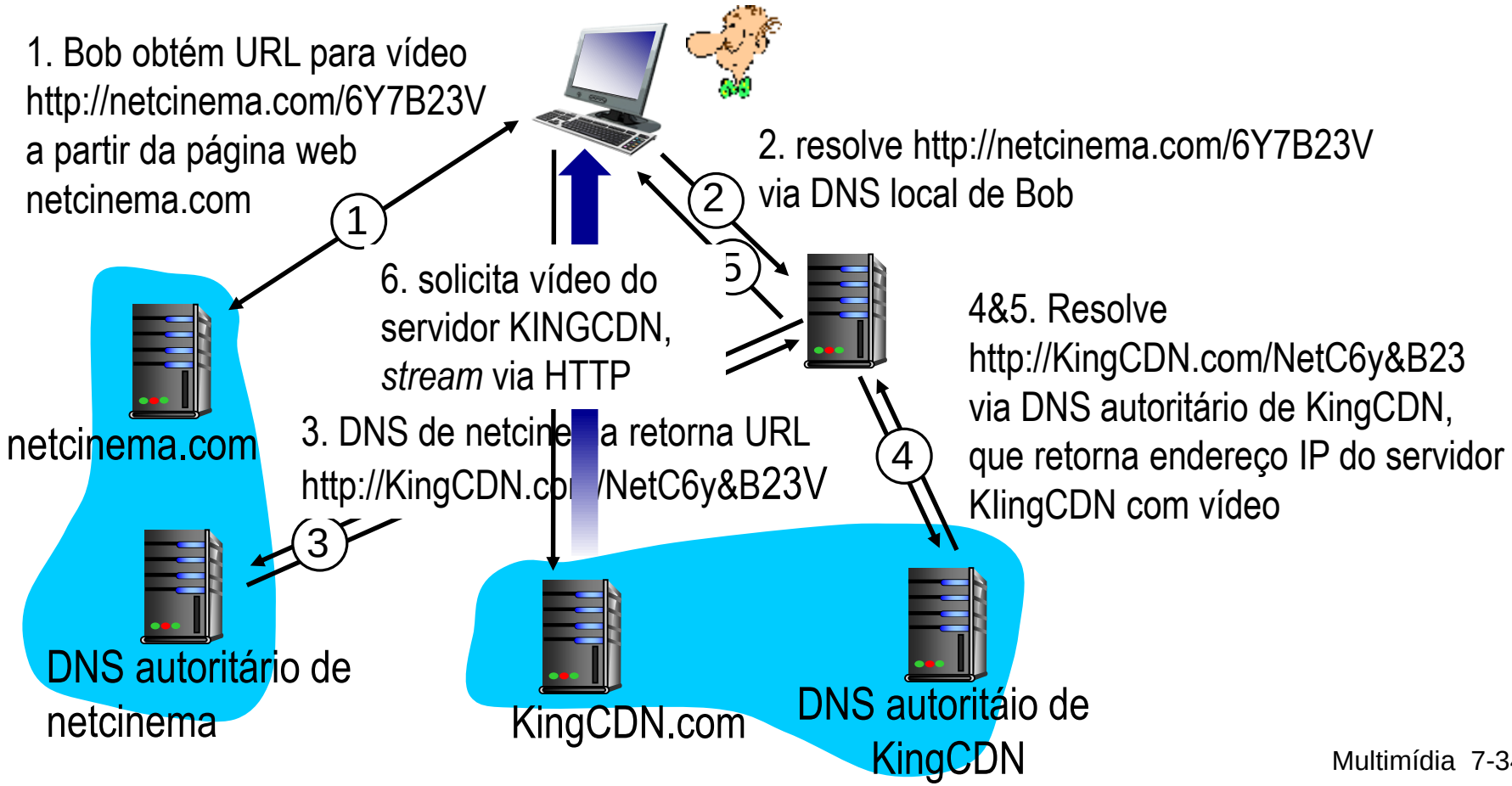
- *desafio:* como distribuir conteúdo (selecionado de milhões de vídeos) para centenas de milhares de usuário simultâneos?
- *opção 2:* armazenar/servir múltiplas cópias de vídeos em múltiplos locais geograficamente distribuídos *(CDN)*
  - *CDN privada*
    - Google, Netflix (Open Connect)
  - *CDN terceirizada*
    - Akamai, Limelight, Level-3 (para Netflix, Hulu, Microsoft)
  - *enter deep:* colocar servidores CDN *dentro* de muitas redes de acesso
    - mais próximo a usuários; difícil administração
    - usado pela Akamai, 1700 locais; Open Connect (Netflix); Google (TCP *splitting*; conteúdo estático)
  - *bring home:* menor número (dezenas) de *clusters* maiores em POPs próximos (mas não dentro) de redes de acesso, usando IXPs
    - usado por Limelight, Google (YouTube)
    - conectados por redes privadas de alta velocidade
    - mais fácil administração

Exemplo: CDN Limelight – *Bring Home*

CDN: cenário "simples"de acesso a conteúdo

Bob (cliente) solicita vídeo http://netcinema.com/6Y7B23V

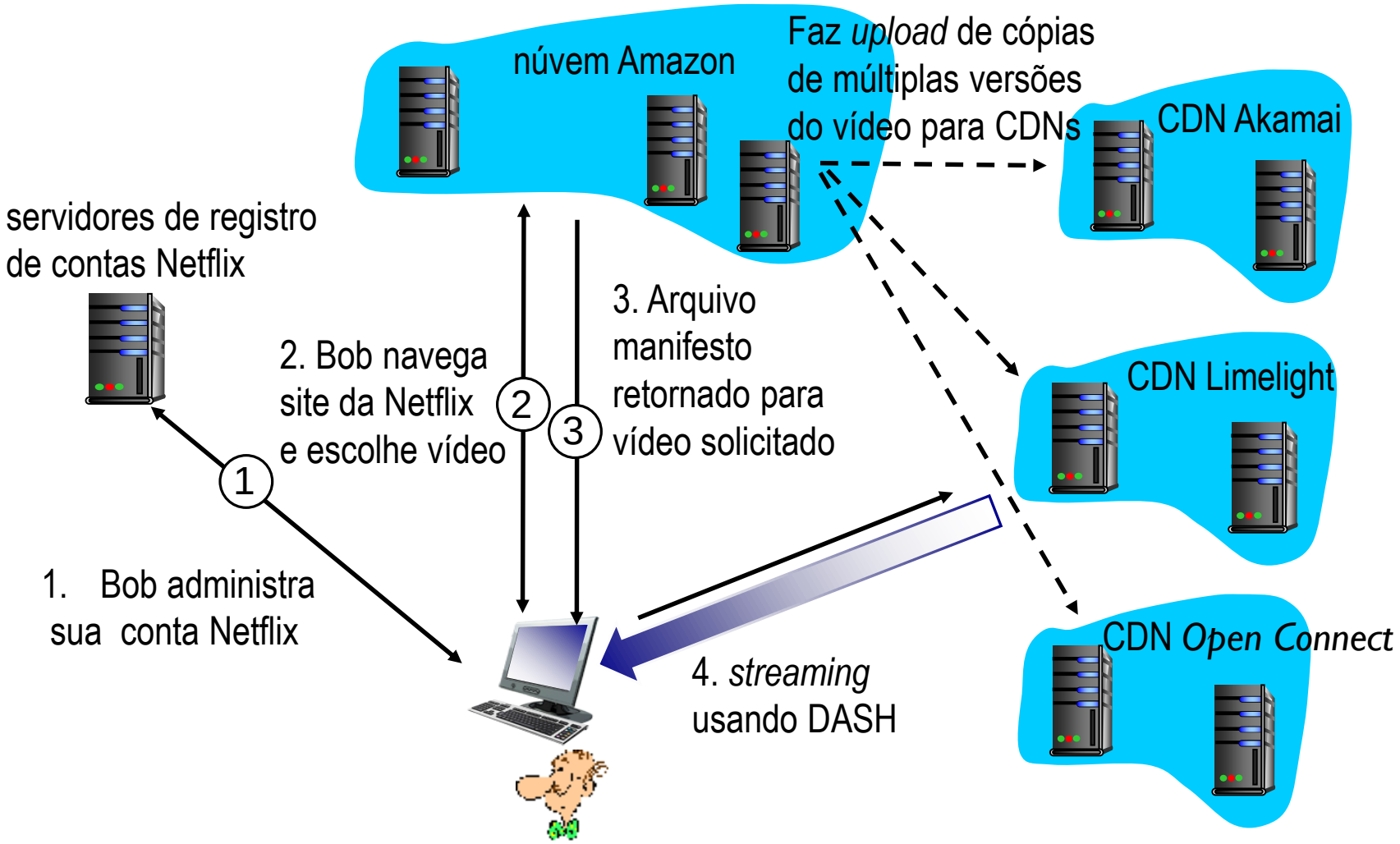
- vídeo armazenado em CDN em http://KingCDN.com/NetC6y&B23V



Estudo de caso : Netflix

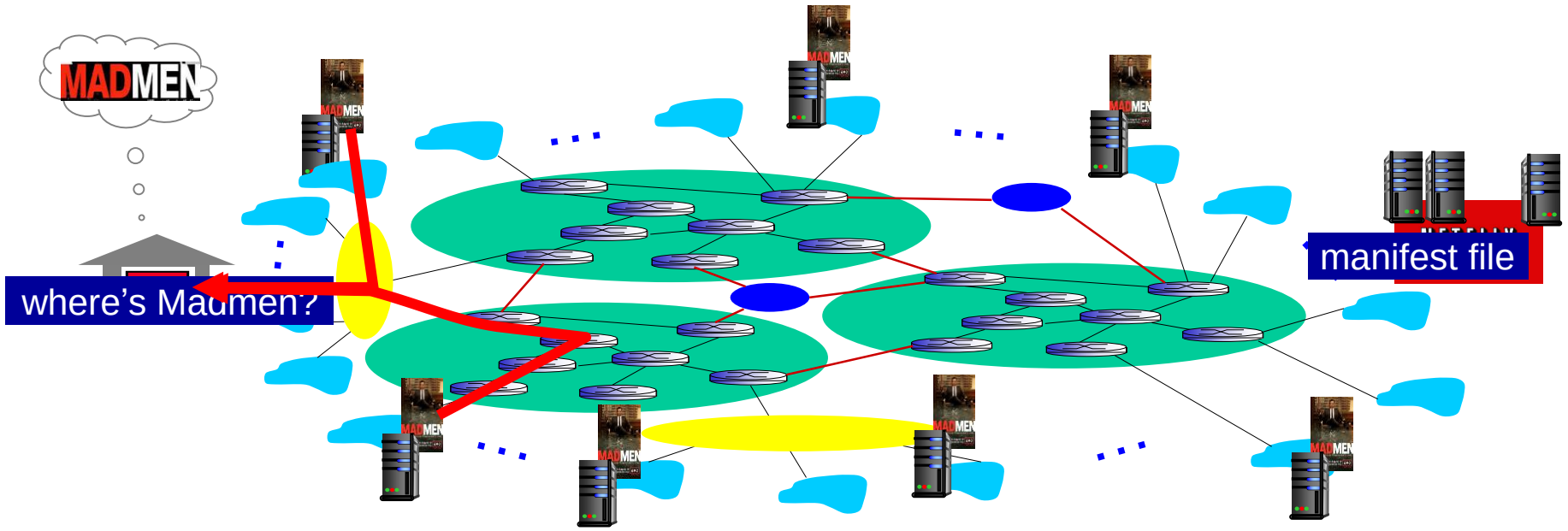
- 8,31% do tráfego *downstream na América Latina em 2016* (35,15% nos EUA)
- tem muito pouca infraestrutura (!), usa serviços terceirizados:
  - serviço de nuvem da Amazon:
    - Banco de dados de registro e pagamentos
    - Netflix faz *upload* do original do estúdio para a núvem Amazon
    - cria múltiplas versões do filme (diferentes codificações) na núvem
    - faz *upload* das versões da nuvem para CDNs
    - Nuvem hospeda páginas *web* do Netflix para navegação do usuário
  - *Inicialmente 3* CDNs terceirizados hospedavam e faziam *streaming* do conteúdo Netflix: Akamai (em SP), Limelight, Level-3 (em SP)
  - Desde 2012 criou CDN própria *Open Connect (instala rack gratuitamente em ISPs!!!) – Mais detalhes aqui e aqui*

Estudo de caso: Netflix



Redes de Distribuição de Conteúdo (CDNs)

- CDN: guarda cópias do conteúdo em nós CDN
  - Por exemplo, Netflix guarda cópias de *MadMen*
- Assinante requer conteúdo de CDN
  - Direcionado à cópia mais próxima, obtém
  - Pode escolher cópia diferente se caminho na rede estiver congestionado



Redes de Distribuição de Conteúdo (CDNs)

Desafios do OTT: lidar com uma internet congestionada

- De qual nó CDN obter o conteúdo?
- Comportamento do espectador perante congestionamento?
- Qual conteúdo colocar em cada nó CDN?

Mais em outros cursos…

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

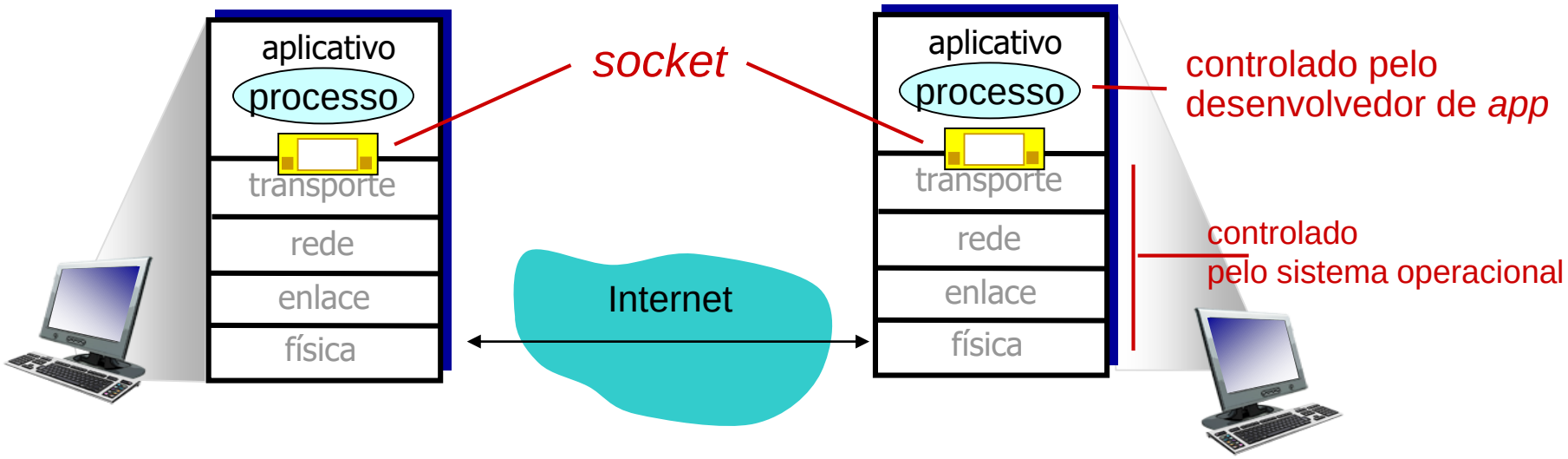
2.6 *Streaming* de vídeo e redes de distribuição de conteúdo

2.7 Programando *socket* com UDP e TCP

Programando *Sockets*

objetivo: aprender como construir aplicativos cliente/servidor que se comunicam usando *sockets*

socket: porta entre processo aplicativo e protocolo de transporte fim-a-fim



Programando *Sockets*

2 tipos de socket para 2 serviços de transporte:

- *UDP:* datagrama não confiável
- *TCP:* confiável, orientada a fluxo de bytes

Exemplo de Aplicativo:

1. Cliente lê uma linha de caracteres (dados) do seu teclado e envia os dados ao servidor.
2. Servidor recebe os dados e converte os caracteres em maiúsculas.
3. Servidor envia os dados modificados ao cliente.
4. Cliente recebe os dados modificados e mostra a linha na tela.

Programando *Sockets com UDP*

UDP: sem "conexão" entre cliente & servidor

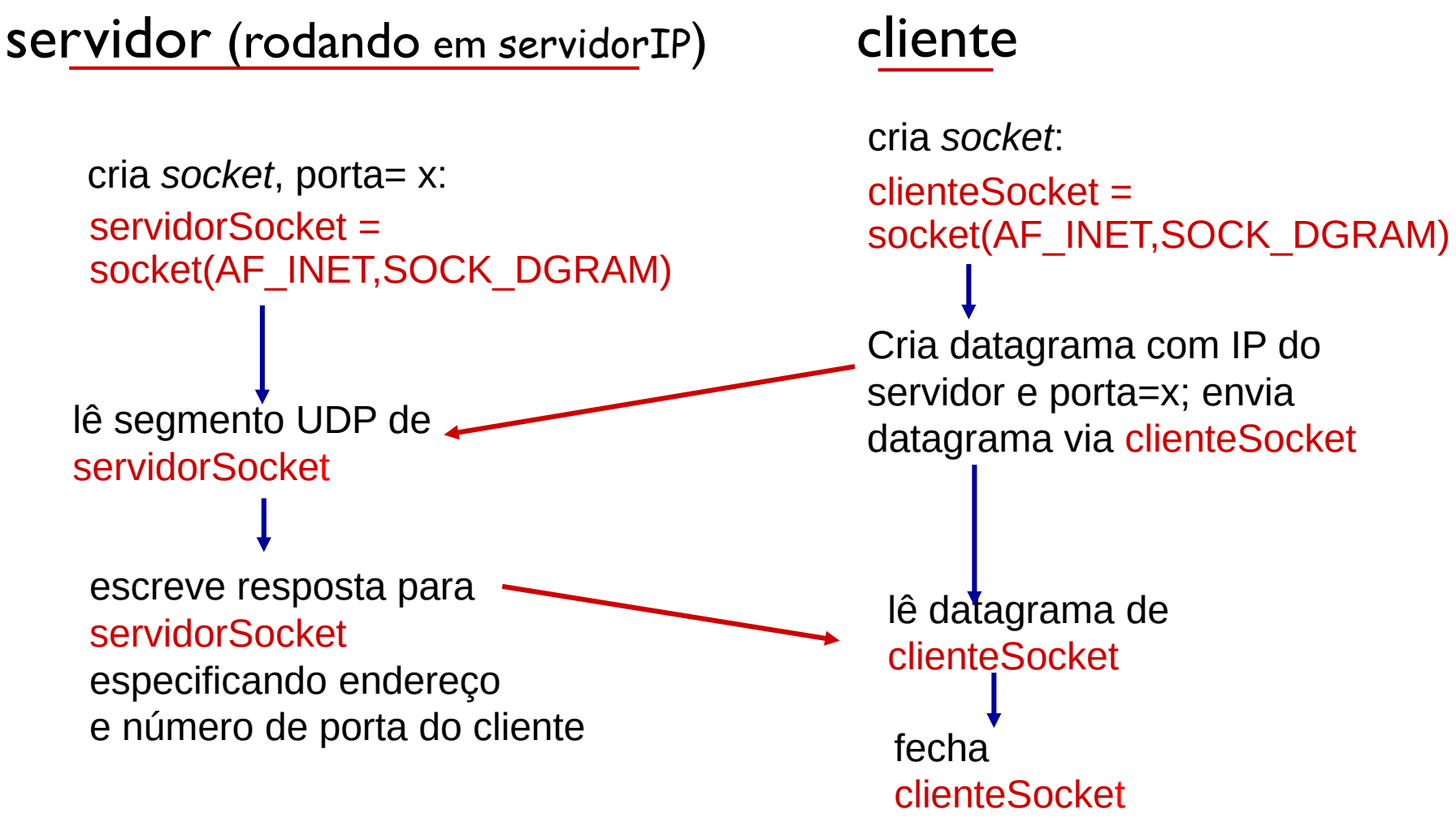
- sem "*handshaking*" antes de enviar dados
- remetente explicitamente anexa endereço IP e # porta do destino em cada pacote
- destinatário extrai endereço IP e # porta do emissor de cada pacote recebido

UDP: dados transmitidos podem ser perdidos ou recebidos fora de ordem

Ponto de vista do aplicativo:

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

Interação dos *socket* cliente/servidor: UDP



Exemplo de *app*: cliente UDP

UDPCliente.py em Python

inclui biblioteca de *socket* do Python

cria socket UDP no cliente

obtém entrada do teclado do usúário

Anexa nome e número da porta do servidor à mensagem; enviar pelo *socket*

lê caracteres respondidos do *socket* para uma *string*

imprime mensagem recebido e fecha *socket*

```
from socket import *

Nomeservidor = "hostname" # endereço IP ou nome do servidor

Portaservidor = 12000

clienteSocket = socket(socket.AF_INET,
                                        socket.SOCK_DGRAM)

mensagem = raw_input('Entre uma frase em minusc.:')

clienteSocket.sendto(mensagem,(Nomeservidor, Portaservidor))

Mensagemmodificada, Enderecoservidor= clienteSocket.recvfrom(2048)

print Mensagemmodificada

clienteSocket.close()
```

endereço IPv4

UDP

Exemplo de *app*: servidor UDP

ServidorUDP em Python

```
from socket import *

Portaservidor = 12000

servidorSocket = socket(AF_INET, SOCK_DGRAM)

servidorSocket.bind(('', Portaservidor))

print "O servidor está pronto para receber"

while 1:

    mensagem, enderecoCliente = servidorSocket.recvfrom(2048)

    mensagemModificada = mensagem.upper()

    servidorSocket.sendto(mensagemModificada, enderecoCliente)
```

cria *socket* UDP

amarra *socket* à porta local número 12000

laço eterno

Lê de *socket* UDP para mensagem, obtendo endereço do cliente (IP e porta)

envia *string* em maiúsculas de volta para o cliente