
# SISTEMA DE NANO CACHES RESIDENCIAIS PARA DISTRIBUIÇÃO DE VÍDEO

Gabriel Guimarães Braga dos Santos Mendonça

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Rosa Maria Meri Leão

Rio de Janeiro
Março de 2015

SISTEMA DE NANO CACHES RESIDENCIAIS PARA DISTRIBUIÇÃO DE VÍDEO

Gabriel Guimarães Braga dos Santos Mendonça

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Rosa Maria Meri Leão, Dr.

Prof. Ana Paula Couto da Silva, D.Sc.

Prof. Daniel Sadoc Menasche, Ph.D.

Prof. Edmundo Albuquerque de Souza e Silva, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2015

Mendonça, Gabriel Guimarães Braga dos Santos

Sistema de nano caches residenciais para distribuição de vídeo/Gabriel Guimarães Braga dos Santos Mendonça. – Rio de Janeiro: UFRJ/COPPE, 2015.

XI, 61 p.: il.; 29, 7cm.

Orientador: Rosa Maria Meri Leão

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2015.

Referências Bibliográficas: p. 57 – 61.

1. P2P. 2. VoD. 3. STB. I. Leão, Rosa Maria Meri. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

*"E tudo quanto fizerdes, fazei-o de todo o coração, como ao Senhor, e não aos homens."*
*Paulo (Colossenses, 3:23)*

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

# SISTEMA DE NANO CACHES RESIDENCIAIS PARA DISTRIBUIÇÃO DE VÍDEO

Gabriel Guimarães Braga dos Santos Mendonça

Março/2015

Orientador: Rosa Maria Meri Leão

Programa: Engenharia de Sistemas e Computação

Nos últimos anos, a popularização das aplicações multimídia motivou inúmeras propostas para distribuição de vídeo usando arquiteturas *peer-to-peer* (P2P). Neste trabalho, apresentamos um sistema P2P para transmissão de vídeo sob demanda (VoD) usando *set-top boxes* (STBs) e roteadores domésticos. Propomos um algoritmo distribuído que considera a limitação de recursos nesses dispositivos e tem como objetivo alocar *seeds* em um *swarm* proporcionalmente ao número de *leechers*. Elaboramos um modelo aproximado do sistema que proporcionou uma redução significativa no espaço de estados: de exponencial para polinomial no número de *peers*. Validamos o modelo aproximado e obtivemos um erro médio inferior a 3%. Avaliamos o desempenho da plataforma através de simulações com dados de um sistema real de VoD, em que a popularidade dos conteúdos é bem modelada por uma distribuição Zipf com decaimento exponencial. Os resultados mostram que, para os *swarms* de baixa e média popularidade, o algoritmo proposto reduziu significativamente o tempo médio de *download*. Além disso, nosso algoritmo permitiu um tempo médio de *download* inferior à duração do vídeo para todos os *swarms*.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

RESIDENTIAL NANO CACHE SYSTEM FOR VIDEO STREAMING

Gabriel Guimarães Braga dos Santos Mendonça

March/2015

Advisor: Rosa Maria Meri Leão

Department: Systems Engineering and Computer Science

In recent years, the increasing popularity of multimedia applications inspired a large number of peer-to-peer (P2P) video streaming solutions. In this work, we present a P2P video-on-demand (VoD) system that utilizes set-top boxes (STBs) and home gateways for content distribution. We propose a distributed algorithm that considers the resource constraints in these devices and aims at allocating seeds in a swarm proportionally to the number of leechers. We developed an approximate model for the system that allowed a significant reduction of the state space: from exponential to polynomial on the number of peers. We validated the model and the mean error is below 3%. We evaluated our platform's performance through simulations using data from a real VoD system in which content popularity seems to follow a Zipf distribution with exponential cutoff. The results show that the proposed algorithm significantly reduces the video download time in the swarms with low and medium popularity. Furthermore, the mean video download time is below video duration for all swarms.

# Sumário

**Lista de Figuras** **ix**

**Lista de Tabelas** **xi**

**1 Introdução** **1**

**2 Trabalhos Relacionados** **4**

**3 Arquitetura do Sistema** **7**

3.1 Troca de *swarms* . . . . . 8

3.1.1 Outras políticas . . . . . 10

3.2 Protótipo . . . . . 11

**4 Modelagem** **14**

4.1 Limite de capacidade do sistema . . . . . 14

4.2 Troca de *swarms* . . . . . 15

4.2.1 Espaço de estados . . . . . 15

4.2.2 Probabilidade de saída de um *swarm* . . . . . 17

4.2.3 Probabilidade de entrada em um *swarm* . . . . . 19

4.2.4 Aproximação geométrica truncada . . . . . 21

4.2.5 Validação . . . . . 22

4.3 Tangram-II . . . . . 27

**5 Resultados** **31**

5.1 Análise estatística de um sistema real . . . . . 31

5.1.1 Processo de chegadas . . . . . 33

5.1.2 Distribuição de popularidade dos conteúdos . . . . . 35

5.2 Simulações . . . . . 46

5.2.1 Taxa de chegada . . . . . 47

5.2.2 Troca de *swarms* . . . . . 48

5.2.3 Servidor de conteúdo . . . . . 50

5.2.4 Distribuição de popularidade . . . . . 50

5.2.5 Carga de um sistema real de VoD . . . . . 51
5.3 Experimentos . . . . . 52

**6 Conclusão** **55**

**Referências Bibliográficas** **57**

# Lista de Figuras


3.1 Arquitetura do sistema. . . . . 8

3.2 Ilustração da política de troca de *swarms*. Os conteúdos estão ordenados em ordem crescente de carga. Um *peer* deve trocar o *swarm* de menor carga dentre aqueles que serve pelo de maior carga dentre os que armazena (*Peer* 1 troca *swarm* 1 pelo 7 e *Peer* 2 troca *swarm* 2 pelo 4). Caso a carga do primeiro seja maior ou igual à do segundo, a troca não ocorre (*Peer* 3). . . . . 10

3.3 Diagrama ilustrando a arquitetura do protótipo de nano cache. A biblioteca *libbtstream* é responsável pela recuperação dos arquivos e um módulo independente implementa nossa política de troca de *swarms*. 13

4.1 Gráfico da cardinalidade do espaço de estados em função do número de nano caches (*peers*) no sistema. Os demais parâmetros são mantidos constantes: $C = 10$, $B = 5$ e $m = 2$. . . . . 17

4.2 Exemplo de entrada gerada pelo Algoritmo 1 para $N = 3$ *peers*, $C = 8$ conteúdos disponíveis, $B = 5$ conteúdos em cache e $m = 2$ conteúdos servidos. Os *swarms* encontram-se em ordem crescente de carga. . . . 25

4.3 Comparação entre as distribuições para o pior caso. *Swarms* em ordem crescente de carga. . . . . 27

5.1 *Streaming* de vídeo sob demanda usando a plataforma Cisco TV CDS. 32

5.2 Número de acessos por dia de 01/12/2013 a 28/02/2014. As linhas verticais pontilhadas indicam os domingos. . . . . 33

5.3 Total de acessos por hora durante os 3 meses. Cada ponto indica o número de acessos entre Xh00 e Xh59. . . . . 34

5.4 Autocorrelação do número de acessos por minuto ao longo do período de 90 dias. Os valores destacados no eixo $x$ indicam intervalos de 12 horas. . . . . 35

5.5 Histograma normalizado do número de acessos por minuto. . . . . 36

5.6 Histograma normalizado do número de acessos por minuto no período de alto tráfego (15h a 2h). . . . . 37

5.7 Gráfico de probabilidade comparando amostras de acessos por minuto no período de alto tráfego (15h a 2h) com distribuição de Poisson. . . 38
5.8 Gráfico da função distribuição acumulada complementar (CCDF) empírica do número de visualizações de cada conteúdo. . . . . 39
5.9 Gráfico mostrando cauda da CCDF empírica do número de acessos a cada conteúdo e os diversos modelos testados. . . . . 41
5.10 Gráfico de probabilidade comparando a distribuição do número de acessos a cada conteúdo com a distribuição Zeta com decaimento exponencial. . . . . 42
5.11 Gráfico de popularidade dos conteúdos. . . . . 43
5.12 Gráfico com diferentes modelos para a distribuição de popularidade dos conteúdos. . . . . 45
5.13 Gráfico de probabilidade comparando amostras com distribuição Zipf com decaimento exponencial. . . . . 46
5.14 Comparação do tempo médio de *download* por *swarm* para diferentes valores de $\tau$. *Swarms* em ordem crescente de popularidade. . . . . 49
5.15 Comparação do tempo médio de *download* por *swarm* com e sem a participação do servidor de conteúdo. *Swarms* em ordem crescente de popularidade. . . . . 50
5.16 Comparação do tempo médio de *download* por *swarm* para as duas distribuições de popularidade. *Swarms* em ordem crescente de popularidade. . . . . 52
5.17 Resultados obtidos após 27 dias de experimentos com 9 nano caches. 54

# Lista de Tabelas


3.1 Descrição dos principais parâmetros do sistema . . . . . 8

4.1 Descrição dos parâmetros da validação com o método Monte Carlo . . 23
4.2 Valores de parâmetros adotados . . . . . 26
4.3 Resultados da avaliação com método Monte Carlo (V=50) . . . . . 27

5.1 Amostra dos *logs* de acesso do sistema. . . . . 33
5.2 Sumário das estatísticas dos *logs* de acesso do sistema. . . . . 33
5.3 Resultado da comparação do *fitting* da distribuição Zeta com outras distribuições. . . . . 40
5.4 Resultado do *fitting* da popularidade dos conteúdos. . . . . 45
5.5 Parâmetros padrão adotados nas simulações . . . . . 48
5.6 Avaliação do valor máximo de $\lambda$. Limite superior: 5467 *leechers* / h . 48
5.7 Avaliação do intervalo entre trocas de *swarms*. . . . . 49
5.8 Avaliação da distribuição de popularidade dos conteúdos. . . . . 51

# Capítulo 1

# Introdução

As aplicações multimídia têm se tornado cada vez mais populares nos últimos anos. Essa realidade motivou diversas soluções para transmissão de vídeo usando arquiteturas *peer-to-peer* (P2P), em que os *peers* participam do processo de distribuição fornecendo recursos para o sistema como banda e capacidade de armazenamento. Aplicações de sucesso como PPLive, QQLive e PPS.tv (PPStream) [1] utilizam essa metodologia para distribuir grandes volumes de tráfego de vídeo na Internet.

Provedores de serviço de Internet (ISPs) também podem se beneficiar de redes P2P. Encontramos na literatura propostas que sugerem o uso dos recursos disponíveis em *set-top boxes* (STBs) e/ou roteadores domésticos para a descentralização da distribuição de conteúdo em redes de banda larga [1–3]. Esse é um cenário favorável, já que esses dispositivos encontram-se sob controle do provedor de serviço e, em geral, estão ligados durante a maior parte do tempo.

Três importantes problemas têm sido estudados em redes P2P para transmissão de vídeo com STBs. Há trabalhos que, considerando o cenário no qual um único vídeo é assistido, abordam os problemas de (1) escolha dos *peers* que cada dispositivo deve servir (vizinhos) e (2) determinação da ordem em que os pedaços do arquivo devem ser baixados. No contexto de distribuição de vários vídeos, muitos trabalhos têm como foco o problema de (3) alocação de conteúdos na rede: dada uma distribuição de popularidade, quantas cópias de cada conteúdo deve haver no sistema e como uma alocação ótima pode ser alcançada. Diversos estudos anteriores [4–9] se restringem à analise de um desses problemas sem modelar com o devido detalhe as limitações impostas por um dispositivo embarcado. Comumente [1, 10, 11], considera-se apenas um limite no número de conexões de *upload* e *download* que um *peer* é capaz de estabelecer.

Focamos este trabalho em um problema pouco estudado: o de escolha dos conteúdos que cada dispositivo deve servir dentre aqueles que se encontram armazenados

[1] http://www.pplive.com/en/index.html, http://live.qq.com/ e http://www.pps.tv/

localmente. Em geral, *softwares* P2P para compartilhamento de arquivos implementam políticas de serviço baseadas em justiça, segundo as quais um *peer* deve retribuir a um *swarm* os recursos que recebeu. Políticas desse tipo podem não favorecer o desempenho do sistema, já que não se preocupam com o fato de que, em um dado momento, pode haver um desbalanceamento entre oferta e demanda por determinados conteúdos, que necessitarão de mais recursos do que outros.

Tradicionalmente, assume-se que a popularidade segue uma distribuição Zipf pura [4, 5, 7–9, 11–14]. Neste trabalho, adotamos uma distribuição de popularidade Zipf com decaimento exponencial. A distribuição que utilizamos se aproxima mais da encontrada em sistemas reais como YouTube, Dailymotion, Veoh e Netflix [15, 16].

O objetivo desta dissertação é propor e analisar uma plataforma P2P para transmissão de vídeo sob demanda (VoD) usando nano caches. Os nano caches são dispositivos presentes nas casas dos usuários, como STBs e roteadores domésticos, que estão sob controle de um provedor de serviço de Internet. Esses dispositivos formam uma rede P2P utilizando a banda dos clientes para distribuição de vídeos armazenados localmente. Para o projeto e avaliação dessa plataforma, levamos em consideração circunstâncias mais próximas da realidade como uma distribuição de popularidade com decaimento exponencial e uma limitação no número de conteúdos que um dispositivo é capaz de servir simultaneamente.

Nossas principais contribuições incluem:

- Proposta e avaliação de um algoritmo distribuído para balanceamento de carga através da escolha dos conteúdos servidos por cada nano cache.
- Protótipo embarcado de nano cache rodando em sistema operacional OpenWrt.
- Modelo probabilístico para o processo de escolha dos conteúdos servidos, que permite redução do espaço de estados de exponencial para polinomial no número de nano caches.
- Análise estatística de dados de um sistema brasileiro de vídeo sob demanda.
- Avaliação do desempenho do sistema proposto através de simulações usando a ferramenta Tangram-II.

Inicialmente, discutimos os trabalhos relacionados no capítulo 2. No capítulo 3, descrevemos a arquitetura do nosso sistema, nosso algoritmo para escolha dos conteúdos a serem servidos e nosso protótipo embarcado desenvolvido para prova de conceito. Em seguida, no capítulo 4, apresentamos nossos modelos para o limite de capacidade do sistema, para a política de troca de *swarms* e para a qualidade de serviço dos usuários. Apresentamos no capítulo 5 os resultados de nossas simulações

e experimentos, juntamente com uma análise de dados de um sistema real de VoD. Por fim, concluímos o trabalho no capítulo 6.

# Capítulo 2

# Trabalhos Relacionados

Neste capítulo, apresentamos outros trabalhos relacionados à distribuição de vídeo, arquiteturas P2P e redes formadas por roteadores domésticos e STBs.

Em um dos primeiros trabalhos a mencionar o uso de STBs para distribuição de vídeo em uma rede de banda larga, Suh *et al.* [1] propuseram um sistema *peer-to-peer* de VoD baseado em operações *push*. Nesse sistema, o provedor de serviço envia os conteúdos para os *peers* em uma etapa anterior à transmissão dos vídeos, com o objetivo de aumentar a disponibilidade dos conteúdos e otimizar o uso da banda de *upload* dos clientes. A proposta se baseia, portanto, numa política centralizada para gerência dos conteúdos armazenados em cada STB. Em nosso trabalho, assumimos que os *peers* armazenam conteúdos assistidos anteriormente e que o número de réplicas de cada conteúdo é controlado de maneira distribuída através de uma política simples de substituição de cache. Além disso, Suh *et al.* assumem que todos os conteúdos disponíveis no sistema são igualmente populares.

Diversos outros trabalhos abordam o problema de replicação de conteúdos em sistemas P2P de VoD. Alguns, como [4, 6, 17], sugerem a adoção de políticas simples de substituição de cache. Tewari e Kleinrock [4] mostraram que a replicação proporcional (número de réplicas de cada arquivo proporcional à sua popularidade) minimiza a banda "consumida" por cada *download* e que as políticas LFU, LRU, FIFO e substituição aleatória levam a uma alocação próxima à proporcional. Neste trabalho, assim como em muitos outros [5, 7–9, 11–14], os autores assumem que a popularidade dos conteúdos segue uma distribuição Zipf. Já em [6], Wu e Li mostraram através de simulações que heurísticas simples como LRU e LFU fornecem desempenho próximo ao ótimo em sistemas de VoD nos quais a popularidade dos conteúdos segue uma distribuição exponencial alongada (Weibull). Neste trabalho, assumimos uma distribuição de popularidade Zipf com decaimento exponencial, conforme observado em alguns sistemas reais [15, 16].

Outros trabalhos propõem estratégias de replicação mais complexas com a participação direta de um servidor, como em [1], ou com base em informação global. Tan

e Massoulié [9], por exemplo, fornecem um modelo de rede de perda (*loss network*) que permite determinar uma alocação ótima quando o número de usuários vai para infinito e sugerem estratégias do tipo *push* (em que conteúdos são enviados proativamente para os caches, sem que tenham sido requisitados pelos usuários) que levam a essa alocação. Já em [18], Liu *et al.* apresentam um modelo de filas para o cálculo da alocação ótima que não está atrelado a uma distribuição de popularidade específica. Seus resultados indicam que a heurística apresentada em [17], que usa a métrica ATD (*availability to demand ratio* - razão disponibilidade / demanda) adotada pelo cliente de VoD PPLive, leva a um desempenho próximo ao ótimo em cenários com distribuição de popularidade Zipf ou exponencial alongada. Por fim, Wu e Lui [7] sugerem que a replicação proporcional não é uma estratégia ótima. Em vez disso, propõem a replicação com base em uma métrica de déficit de banda, que, assim como a ATD, é calculada com informação global obtida de um servidor.

Assim como o nosso, o trabalho de Muñoz-Gea *et al.* [11] analisa outros aspectos do projeto de sistemas de distribuição de vídeo com STBs que não a alocação de conteúdos nos caches. Nele, os autores propõem um modelo analítico em que cada STB é modelado como uma fila. O desempenho do sistema é avaliado a partir da probabilidade de um cliente que assiste a um vídeo não encontrar nenhum STB com capacidade disponível para serví-lo. Como em [1, 10], o número de conexões de *upload* que um STB estabelece é limitado, mas assume-se que um *peer* é capaz de servir qualquer número de conteúdos dentre os que estão armazenados localmente. Em nosso trabalho, assumimos que, devido às limitações de memória e processamento de um dispositivo embarcado, este não é capaz de servir, ao mesmo tempo, todos os arquivos que armazena. Focamos nossa análise, portanto, na escolha dos vídeos que um *peer* deve servir. Outros trabalhos deixam de considerar um limite no número de conteúdos simultaneamente servidos por assumir que os caches armazenam um número muito pequeno de arquivos (menor do que 3) [7, 17, 19, 20]. Neste trabalho consideramos caches maiores, capazes de armazenar cerca de 20 vídeos.

O problema de colaboração entre *swarms* na distribuição de arquivos com BitTorrent foi estudado em [21, 22]. Já em [12, 20], o mesmo problema foi estudado no contexto de aplicações de VoD P2P. Como em nossa proposta, esses autores propõem que a escolha dos conteúdos que um *peer* serve seja feita com base em uma métrica de carga de um *swarm*. A métrica que definimos neste trabalho possui como vantagem o fato de ser facilmente calculada usando informações de um *tracker* ou DHT. As estratégias propostas em [12, 20], por outro lado, dependem do conhecimento da banda de *upload* dedicada pelos servidores de conteúdo a cada *swarm*, o que implica em um número maior de mensagens trocadas e, portanto, maior *overhead*.

Há ainda trabalhos que sugerem o uso de STBs e *gateways* domésticos num contexto mais geral de aplicações. Laoutaris *et al.* [2] introduzem o conceito de

nano *data centers* e fazem uma análise das vantagens e desvantagens de uma infraestrutura distribuída de armazenamento na borda da rede usando dispositivos domésticos. Whiteaker *et al.* [23] discutem os desafios envolvidos na criação de *gateways* domésticos avançados e apresentam um protótipo que usa virtualização para hospedar diversos serviços como VPNs, servidores *web*, distribuição e decodificação de vídeos e compartilhamento de arquivos com BitTorrent.

# Capítulo 3

# Arquitetura do Sistema

Neste capítulo, descrevemos a arquitetura e o funcionamento de nosso sistema, assim como nossa política de troca de *swarms* para escolha dos conteúdos servidos. Apresentamos também detalhes de nosso protótipo embarcado desenvolvido para prova de conceito.

Propomos um sistema formado por um conjunto de nano caches, um servidor de controle e um servidor de conteúdo, como ilustrado pela figura 3.1. Os nano caches são dispositivos embarcados (roteadores domésticos ou STBs - *set-top boxes*) com capacidade de armazenamento local suficiente para um conjunto pequeno de vídeos. Assumimos que esses dispositivos encontram-se em uma mesma rede de banda larga sob controle de um provedor de serviço de Internet (ISP) e que estão sempre conectados. O servidor de controle é responsável pela indexação dos conteúdos disponíveis, pela autenticação dos clientes e pela monitoração dos nano caches conectados (*tracker*). Já o servidor de conteúdo tem como função garantir a disponibilidade dos vídeos distribuídos no sistema, armazenando cópias de todos os conteúdos e, se necessário, participando da transmissão de vídeo. Mais servidores de conteúdo podem ser adicionados ao sistema para balanceamento de carga e redundância.

Quando um usuário assiste a um vídeo, seu nano cache faz o *download* a partir de outros nano caches (*peers*) e, se preciso, do servidor de contéudo[1]. Em vez de fazer o *download* a uma taxa igual à taxa de codificação do vídeo, o *peer* que recebe o conteúdo (*leecher*) tenta sempre saturar sua banda, minimizando o tempo de *download*. O conteúdo assistido é inteiramente armazenado para posterior *upload*. Se não há espaço disponível, uma política de substituição de *cache* como LRU (*Least Recently Used*), LFU (*Least Frequently Used*), substituição aleatória ou FIFO (*First-in, first-out*) é adotada. A escolha de uma estratégia ótima para replicação de conteúdos já foi bastante estudada na literatura [1, 4–9, 17, 19]. Focamos portanto nosso trabalho na escolha de quais conteúdos um nano cache deve servir dentre os

[1] A partir deste ponto, adotaremos a terminologia usada em sistemas P2P. Os termos nano cache e *peer* serão utilizados indistintamente.

Tabela 3.1: Descrição dos principais parâmetros do sistema

| Parâmetro | Definição |
|---|---|
| $N$ | Número de nano caches (*peers*) no sistema |
| $C$ | Número de conteúdos disponíveis no sistema |
| $B$ | Número de conteúdos que cabem em um cache |
| $m$ | Número de conteúdos que um nano cache serve |
| $\lambda$ | Taxa de chegada de *leechers* |
| $\tau$ | Intervalo de troca de *swarms* |
| $D_{nc}$ | Banda disponível para *download* nos nano caches |
| $U_{nc}$ | Banda disponível para *upload* nos nano caches |
| $U_s$ | Banda disponível para *upload* no servidor de conteúdo |
| $L_i$ | Número de *leechers* no *swarm* $i$ |
| $S_i$ | Número de *seeds* no *swarm* $i$ |
| $p_i$ | Popularidade relativa do conteúdo $i$ ($\sum_{i=1}^{C} p_i = 1$) |
| $d_i$ | Duração do vídeo $i$ |
| $b_i$ | Taxa de codificação (*bitrate*) do vídeo $i$ |

que possui.

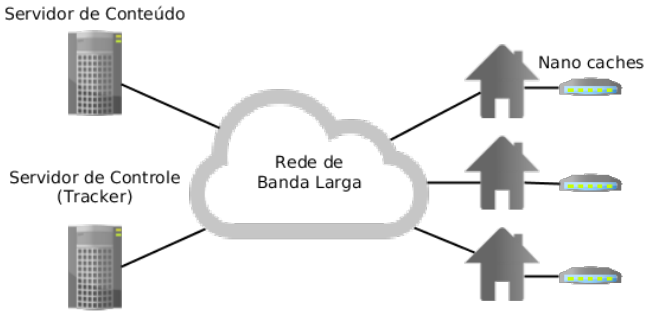



Figura 3.1: Arquitetura do sistema.

Os nano caches utilizam a banda de *upload* dos clientes para servir conteúdos que já foram baixados. Como o canal é compartilhado com outros dispositivos da rede doméstica, o nano cache deve minimizar sua interferência com o tráfego local. Se ele atua como roteador doméstico, sua taxa de *upload* pode ser adaptada de acordo com o tráfego medido em suas interfaces de entrada. Caso não seja possível para o nano cache medir diretamente o tráfego local (por exemplo, se for um STB), ele pode utilizar um algoritmo de controle de congestionamento baseado em latência como o LEDBAT [24]. Ainda assim, esperamos que haja bastante capacidade disponível para o sistema, já que usuários de banda larga raramente saturam sua banda de *upload* e *download* [25].

A notação a ser utilizada para descrição do sistema é apresentada na tabela 3.1.

## 3.1 Troca de *swarms*

Em geral, clientes P2P para distribuição de arquivos usando BitTorrent implementam uma política para escolha dos conteúdos que um cliente deve servir com foco

em justiça: se um *peer* obteve uma quantidade $X$ de recursos de um *swarm* (por exemplo, *bytes*), ele deve retribuir oferecendo ao mesmo *swarm* uma quantidade $Y$ proporcional a $X$. Esse tipo de estratégia não favorece o balanceamento de carga no sistema, pois não considera que certos *swarms* podem necessitar, em um dado momento, de mais recursos do que outros.

Nesta seção, apresentamos nossa política para troca de *swarms* baseada em carga. Com ela, podemos melhorar o desempenho do sistema (e consequentemente a qualidade percebida pelos usuários) transferindo recursos de *swarms* menos necessitados para aqueles mais necessitados.

Dos $C$ conteúdos disponíveis no sistema, um nano cache pode possuir até $B$ conteúdos, sendo $B \ll C$. Entretanto, devido às suas limitações de processamento e memória, um nano cache serve apenas $m$ dentre os $B$ vídeos armazenados. Precisamos portanto definir uma política para escolha desses $m$ conteúdos que devem ser servidos de modo a fazer uma boa alocação dos recursos disponíveis.

A cada vídeo distribuído no sistema corresponde um *swarm*, do qual participam os *peers* que assistem ao vídeo (*leechers*) e os que o servem (*seeds*). A escolha dos $m$ *swarms* que um *peer* deve servir leva em consideração uma métrica de carga, definida para o *swarm* $i$ como

$$\ell_i \triangleq \frac{L_i}{S_i + 1}, \tag{3.1}$$

onde $L_i$ e $S_i$ são, respectivamente, o número de *leechers* e *seeds* no *swarm* $i$. Com essa definição simples, evitamos uma divisão por zero quando não há *seeds* em um *swarm*. Um valor alto de carga pode indicar um número grande de clientes assistindo ao vídeo e/ou um número pequeno de nano caches servindo-o.

A carga de um *swarm* pode ser facilmente calculada por um nano cache a partir dos dados recebidos do servidor de controle, que atua como *tracker*. Portanto, a taxa com que a informação de carga é atualizada depende da frequência com que mensagens de *scrape* – que retornam o número de *seeds* e *leechers* em um ou mais *swarms* – são enviadas ao *tracker*. Mesmo sem a presença de um servidor *tracker*, essa métrica poderia ser estimada de maneira descentralizada utilizando, por exemplo, uma tabela hash distribuída (DHT).

A qualidade de experiência do usuário que assiste a um vídeo está indiretamente ligada à carga do *swarm* no qual ele é distribuído. Por isso, propomos uma política simples que reduz as diferenças de carga entre os *swarms* introduzindo um *overhead* muito pequeno. Diremos que um *peer* **serve** o conteúdo $k$ se ele é *seed* no *swarm* $k$ (e, portanto, possui uma cópia do vídeo em seu cache) e que ele **armazena** o conteúdo $k$ se ele possui uma cópia em seu cache mas não o serve. Nossa política consiste na troca de dois *swarms* em intervalos de tempo $\tau$. No momento da troca, o *peer* busca dentre os $m$ conteúdos que serve aquele que possui o *swarm* de menor

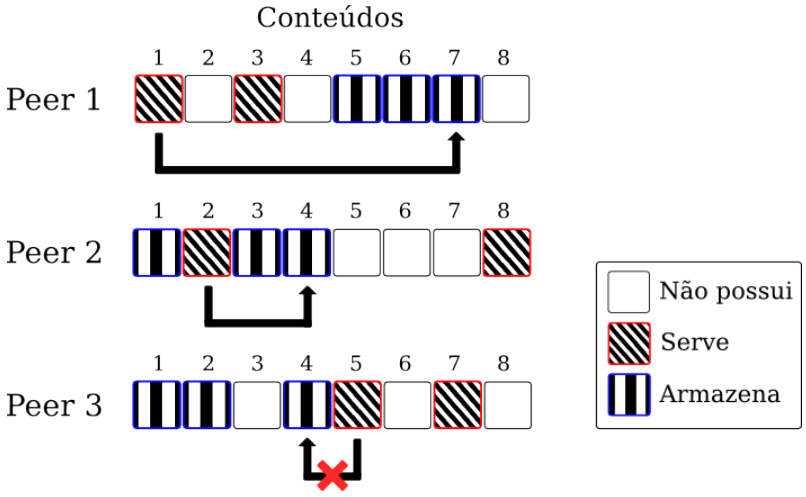



Figura 3.2: Ilustração da política de troca de *swarms*. Os conteúdos estão ordenados em ordem crescente de carga. Um *peer* deve trocar o *swarm* de menor carga dentre aqueles que serve pelo de maior carga dentre os que armazena (*Peer* 1 troca *swarm* 1 pelo 7 e *Peer* 2 troca *swarm* 2 pelo 4). Caso a carga do primeiro seja maior ou igual à do segundo, a troca não ocorre (*Peer* 3).

carga (*swarm* $i$). Depois, dentre os $(B-m)$ conteúdos que armazena, escolhe aquele com *swarm* de maior carga (*swarm* $j$). Se $\ell_j > \ell_i$, o *peer* deixa de ser *seed* em $i$ e passa a ser *seed* em $j$. Caso contrário, ele continua a servir os mesmos $m$ conteúdos. O processo é ilustrado pela figura 3.2.

### 3.1.1 Outras políticas

Grande parte dos *softwares* P2P para *download* de arquivos usando BitTorrent [26] possui um limite configurável para o número de *torrents* concluídos que o cliente pode servir. Aplicativos como o cliente BitTorrent $\mu$Torrent[2] implementam uma política de retribuição de recursos obtidos em um *swarm*, contribuindo com um determinado número de *bytes* ou tempo de serviço mínimos. Esse *software*, considerado atualmente o cliente BitTorrent oficial, define para cada *torrent* uma meta de *seeding* que é atingida quando o *torrent* é servido por um certo tempo, quando possui um número mínimo de *seeds* ou quando uma razão de compartilhamento $\frac{\text{nº de bytes baixados}}{\text{nº de bytes servidos}}$ é atingida. O cliente prioriza os *torrents* cuja meta não foi atingida em detrimento dos demais. E todos os *torrents* em mesma situação (meta atingida ou não) possuem mesma prioridade.

Outro aplicativo popular, o Vuze[3] (conhecido anteriormente como Azureus), também adota a política de meta de *seeding*, mas acrescenta um critério de *seed rank* para o desempate. Por padrão, o *seed rank* de um *swarm* $i$ é dado pela razão $\frac{L_i+S_i}{S_i}$.

[2] http://www.utorrent.com/
[3] http://www.vuze.com/

Além dessa métrica, semelhante à que definimos na equação 3.1, o Vuze oferece outras opções como número de *seeds* no *swarm* e rotação temporizada (*Round-robin*). Entretanto, esse cliente adota um critério extra que tem prioridade sobre os demais: o tempo de serviço. Um *torrent* que não tenha atingido um tempo mínimo de serviço desde que foi iniciado não é substituído. Deste modo, o *software* evita a troca constante dos *torrents* servidos.

A biblioteca *libtorrent*[4], que implementa o protocolo BitTorrent e é utilizada em diversas aplicações (como Deluge[5], Miro[6] e BTStream [27]) adota uma política semelhante à utilizada pelo cliente Vuze, com quatro critérios para seleção (em ordem de prioridade):

1. meta de *seeding*, atingida com certa razão de compartilhamento ($\frac{\text{nº de bytes baixados}}{\text{nº de bytes servidos}}$ e/ou $\frac{\text{tempo de } \textit{download}}{\text{tempo de } \textit{upload}}$) e tempo total de serviço;
2. ausência de *seeds* no *swarm*;
3. tempo de serviço desde que foi iniciado pela última vez;
4. *seed rank*, dado pela equação $\frac{L_i+1}{S_i}$.

Nossa proposta não determina metas de *seeding*, já que, em nosso sistema, o balanceamento da carga dos *swarms* é mais importante do que a justiça. Ademais, a política que definimos possui como vantagem o fato de ser bastante simples, com uma métrica de carga que pode ser facilmente calculada com informações recebidas diretamente do servidor de controle (*tracker*). Outros trabalhos sugerem que a alocação de recursos de *upload* do *peer* seja feita com base na taxa de *upload* do servidor para cada *swarm* [12, 20]. A adoção de tal estratégia implicaria em um *overhead* maior, pois dependeria da participação dos servidores de conteúdo e envolveria um número maior de mensagens de controle.

## 3.2 Protótipo

Como prova de conceito, desenvolvemos um protótipo de nano cache que implementa nossa política de troca de *swarms*. O protótipo consiste em um cliente P2P de distribuição de vídeo que faz uso da biblioteca *libbtstream*. Essa biblioteca faz parte do ambiente BTStream [27], um conjunto de ferramentas para criação de aplicativos multimídia P2P e avaliação experimental de algoritmos baseados no protocolo BitTorrent. Com a *libbtstream*, podemos facilmente desenvolver *softwares* para *streaming* de áudio e vídeo, já que ela encapsula um cliente BitTorrent e provê uma

[4]http://www.libtorrent.org/
[5]http://www.deluge-torrent.org/
[6]http://www.getmiro.com/

API simples para acesso sequencial dos pedaços, independentemente da ordem na qual os pedaços são recuperados dos demais *peers*. A biblioteca, implementada em C++, é uma extensão da *libtorrent*.

Como plataforma, usamos o sistema operacional embarcado OpenWrt[7], uma distribuição Linux para roteadores domésticos que fornece um sistema de arquivos com possibilidade de escrita e um gerenciador de pacotes, *opkg*, com acesso a um repositório contendo mais de 3500 pacotes. O sistema dá acesso a inúmeras funcionalidades de rede, como: roteamento, *firewall*, NAT, VPN, pontos de acesso sem fio e tunelamento IP.

As bibliotecas utilizadas foram adaptadas e compiladas para o OpenWrt. O cliente BitTorrent foi configurado para baixar os pedaços em ordem (sem utilizar a política *Rarest-first* padrão) e fazer *upload* de no máximo $m$ conteúdos como *seed*, sem servir um *swarm* no qual é *leecher*. Desenvolvemos um módulo independente de *software* responsável pela escolha dos conteúdos que o nano cache serve. Três políticas de seleção estão disponíveis:

1. *Round-robin* – todos os conteúdos são servidos com igual prioridade. Os *torrents* são armazenados em uma fila circular e, a cada $\tau$ segundos, o cliente substitui o vídeo que serve há mais tempo pelo próximo da fila.

2. Troca de *swarms* baseada em carga – os conteúdos são escolhidos de acordo com a política apresentada na seção 3.1. A cada $\tau$ segundos, o cliente troca o *swarm* de menor carga dentre aqueles serve pelo de maior carga dentre os demais que ele possui, caso a carga do primeiro seja menor do que a do segundo.

3. *libtorrent* – o cliente adota a política padrão da biblioteca *libtorrent*, apresentada na seção 3.1.1. A cada $\tau$ segundos, o conjunto de conteúdos servidos é atualizado.

Na figura 3.3, vemos um diagrama ilustrando a arquitetura adotada no protótipo.

Nosso *software* foi embarcado em um roteador doméstico TL-WDR3600 da TP-Link. Esse roteador possui um *System-on-Chip* (SoC) AR9344 da Qualcomm Atheros com processador MIPS operando a 560 MHz, 128 MB de memória RAM, 8 MB de memória flash para armazenamento interno e 2 portas USB 2.0. Para armazenamento dos vídeos, usamos um HD externo com capacidade de 500 GB. Como a *libtorrent* permite o uso do protocolo uTP [28] de controle de congestionamento no nível da aplicação – que implementa o LEDBAT [24] e minimiza a interferência do tráfego BitTorrent no tráfego doméstico – nosso protótipo de nano cache pode ser utilizado como roteador *wireless* sem impactar negativamente a experiência dos usuários.

[7] https://openwrt.org/

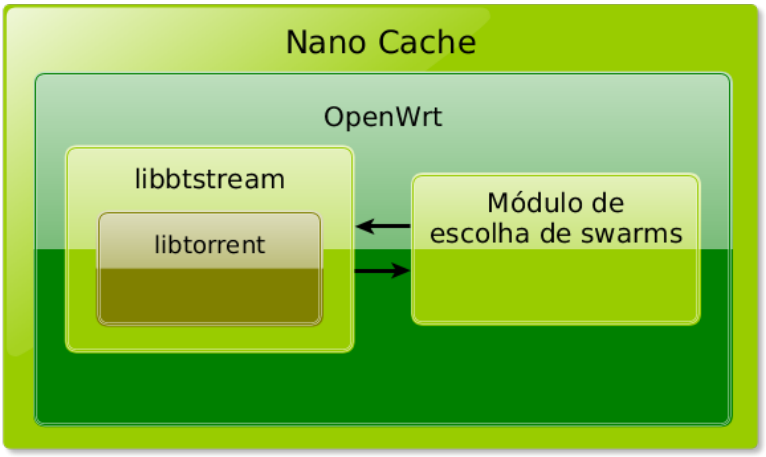



Figura 3.3: Diagrama ilustrando a arquitetura do protótipo de nano cache. A biblioteca *libbtstream* é responsável pela recuperação dos arquivos e um módulo independente implementa nossa política de troca de *swarms*.

Atualmente, possuímos uma rede com nano caches distribuídos entre 15 voluntários de um mesmo provedor de serviço de banda larga.

# Capítulo 4

# Modelagem

Neste capítulo, descrevemos três modelos que desenvolvemos para representar diferentes características do sistema. O primeiro é um modelo analítico bastante simples para o limite de capacidade do sistema. O segundo modelo proposto representa o processo de troca de *swarms* e permite uma drástica redução no espaço de estados do sistema. Já o terceiro é um modelo de simulação desenvolvido com a ferramenta Tangram-II [29] que nos permite avaliar o desempenho de nosso sistema de VoD em cenários diversos.

## 4.1 Limite de capacidade do sistema

Inicialmente, propomos um modelo simples para estimar a taxa máxima de chegada de *leechers* suportada. Com esse modelo, podemos dimensionar os recursos do sistema, como a banda de *upload* dos nano caches e do servidor de conteúdo. Dada uma taxa de chegada, o modelo também nos possibilita o cálculo da taxa de codificação limite para os vídeos, parâmetro de grande impacto na qualidade percebida pelos usuários.

Primeiro calculamos o número máximo de *leechers* que o sistema pode atender de modo eficiente. Assumiremos que todos os vídeos possuem mesma duração $d$ e taxa de codificação (*bitrate*) $b$. Para que os clientes sejam capazes de baixar o vídeo a uma taxa média igual a $b$, o valor esperado do número de *leechers* no sistema deve ser

$$E[L] = \frac{N \cdot U_{nc} + U_s}{b}, \tag{4.1}$$

onde $N \cdot U_{nc} + U_s$ é a soma da capacidade de *upload* dos nano caches e do servidor de conteúdo. Simultaneamente, queremos que o valor esperado do tempo de *download*, $T_D$, seja no pior caso igual a $d$. E, segundo a Lei de Little,

$$E[L] = \lambda \cdot E[T_D] \iff \lambda = \frac{E[L]}{E[T_D]}. \tag{4.2}$$

Aplicando (4.1) em (4.2) e substituindo $E[T_D]$ pela duração do vídeo, temos nosso limite para a taxa de chegada:

$$\lambda_{max} = \frac{N \cdot U_{nc} + U_s}{b \cdot d}. \tag{4.3}$$

Para uma taxa de chegada de *leechers* conhecida, podemos calcular a taxa de codificação máxima:

$$b_{max} = \frac{N \cdot U_{nc} + U_s}{\lambda \cdot d}. \tag{4.4}$$

Conhecendo o limite dado pela equação 4.4, podemos eventualmente aumentar a *bitrate* dos vídeos transmitidos e melhorar sua qualidade sem necessariamente causar um aumento no número de pausas na reprodução, o que causaria um impacto negativo na experiência do usuário.

Na seção 5.2, apresentamos uma avaliação desse modelo feita através de simulações. Mesmo sendo simples, esse modelo nos permite estimar com boa precisão a taxa máxima de chegada de *leechers* que o sistema é capaz de atender de maneira satisfatória.

## 4.2 Troca de *swarms*

Nossa política de troca de *swarms* (ver seção 3.1) determina que, em intervalos regulares, um *peer* tente sair de um dos *swarms* em que é *seed* para entrar em outro *swarm* de carga mais alta. A princípio, para modelar esse processo seria necessário conhecer todos os vídeos que cada nano cache armazena e serve. Entretanto, propomos uma aproximação que nos permite reduzir drasticamente o espaço de estados do modelo utilizando apenas dois vetores de variáveis de estado de tamanho $C$ que indicam o número de *seeds* ($S_i$) e *leechers* ($L_i$) em cada *swarm*, onde $C$ é o número de conteúdos disponíveis.

### 4.2.1 Espaço de estados

A princípio, se temos $N$ nano caches no sistema, podemos definir seu estado como uma matriz $N \times C$ em que cada posição $(i, j)$ indica uma dentre quatro possibilidades:

1. o *peer* $i$ assiste ao conteúdo $j$ (i.e., $i$ é *leecher* em $j$);

2. o *peer* $i$ serve o conteúdo $j$ (i.e., $i$ é *seed* em $j$);

3. o *peer* $i$ armazena o conteúdo $j$ (i.e., $i$ possui o conteúdo $j$, mas não é *seed* no *swarm* $j$);

4. o *peer* $i$ não possui e não assiste ao conteúdo $j$;

Deste modo, a cardinalidade de nosso espaço amostral $\Omega_m$ é dada por:

$$\mid \Omega_m \mid = \left[ \binom{C}{B} \cdot \binom{B}{m} \cdot (C - B + 1) \right]^N,$$

onde $B$ é o número de conteúdos em um nano cache e $m$ é o número de conteúdos servidos. Portanto, $\binom{C}{B}$ é o número de possíveis subconjuntos de conteúdos que um *peer* pode armazenar, $\binom{B}{m}$ é o número de possíveis subconjuntos de conteúdos que um *peer* pode servir (dentre os que possui) e $(C - B + 1)$ é o número de conteúdos que um *peer* pode estar assistindo mais um (caso não esteja vendo nenhum vídeo).

Fica evidente que esta é uma forma ingênua de modelar o sistema, já que o espaço de estados cresce exponencialmente com o número de nano caches. A solução de um modelo com esta complexidade seria inviável mesmo para valores pequenos de $N$, $C$, $B$ e $m$.

Propomos uma forma mais eficiente de modelar o sistema que leva a um espaço de estados menor e, ainda assim, permite a análise da dinâmica de troca de *swarms*. Em vez de modelar o conjunto de conteúdos que cada um dos $N$ *peers* armazena, serve e assiste, definimos como variáveis de estado apenas o número de *peers* que serve e assiste a cada conteúdo, de maneira análoga a [30], onde os autores modelam o número de *leechers* em cada *swarm*. Deste modo, em vez de uma matriz $N \times C$ de variáveis de estado, temos dois vetores de tamanho $C$ que indicam o número de *seeds* e *leechers* em cada *swarm*. Mostraremos mais tarde que, assumindo uma certa política de replicação de conteúdos, podemos fornecer uma boa aproximação mesmo sem guardar o conjunto de conteúdos que cada *peer* possui ou mesmo o número de cópias de cada conteúdo. Com esta representação, podemos fornecer o seguinte limite superior para a cardinalidade do espaço amostral $\Omega_v$:

$$\mid \Omega_v \mid < \binom{N + (C+1) - 1}{(C+1) - 1} \cdot \binom{m \cdot N + C - 1}{C - 1} = \binom{N + C}{C} \cdot \binom{m \cdot N + C - 1}{C - 1},$$

onde $\binom{N+(C+1)-1}{(C+1)-1}$ é o número de formas de alocar $N$ *leechers* em $C$ *swarms* mais um (caso não estejam assistindo a um conteúdo) e $\binom{m \cdot N + C - 1}{C-1}$ é igual ao número de modos de se alocar $m \cdot N$ *seeds* (já que cada *peer* serve exatamente $m$ conteúdos) nos $C$ *swarms*. Esse é um limite superior para a cardinalidade do espaço amostral porque não consideramos que um usuário é *seed* em $m$ *swarms* diferentes e que ele não pode ser *leecher* em um *swarm* em que já é *seed*. Apresentamos na figura

4.1 uma comparação visual da cardinalidade dos dois espaços amostrais em função do número de nano caches. Com a nova representação, o crescimento passa a ser polinomial no número de *peers*.

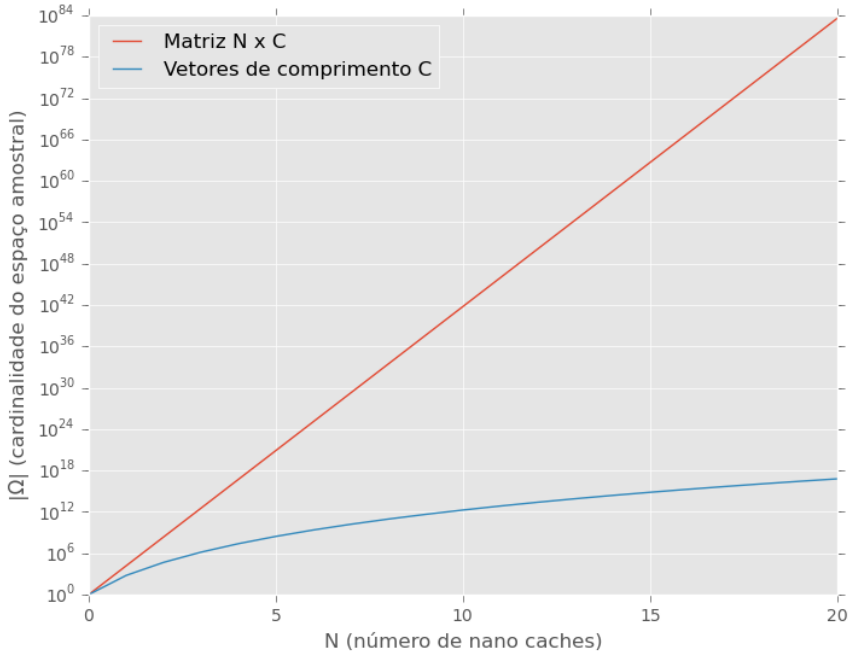



Figura 4.1: Gráfico da cardinalidade do espaço de estados em função do número de nano caches (*peers*) no sistema. Os demais parâmetros são mantidos constantes: $C = 10$, $B = 5$ e $m = 2$.

Apesar de a representação do estado do sistema através de vetores de *leechers* e *seeds* em cada *swarm* ser vantajosa por levar a um espaço de estados menor, ela não nos permite representar com exatidão a dinâmica de troca de *swarms*. Dado um estado do sistema, para modelar precisamente a decisão que um *peer* tomaria, seria preciso armazenar nas variáveis de estado quais os $B$ conteúdos que ele possui e quais os $m$ *swarms* que serve, além da carga de cada *swarm*. Com os dois vetores de variáveis de estado, não temos essa informação. Entretanto, podemos fornecer uma boa aproximação.

### 4.2.2 Probabilidade de saída de um *swarm*

Primeiro, iremos modelar o processo de saída de um *peer* de um *swarm* de carga mais baixa. Assumimos sem perda de generalidade que os $C$ *swarms* estão ordenados de acordo com sua carga, de modo que $\ell_i \leq \ell_j$ (ver equação (3.1)) para todo $i < j$, $1 \leq i, j \leq C$. Definimos então a variável aleatória $R$, que indica o *swarm* de menor carga que um nano cache escolhido ao acaso está servindo. $\Pr(R = k)$ é, portanto,

a probabilidade de um *peer* aleatório optar por sair do *swarm* $k$ [1]. Para que $k$ seja escolhido, é preciso que um *peer* seja *seed* em $k$ e não seja *seed* nos *swarms* anteriores a $k$ (com carga menor ou igual a $\ell_k$).

Seja $\omega_i$ o evento que indica que um *peer* está servindo o conteúdo $i$. Logo

$$\Pr(R = 1) = \Pr(\omega_1).$$

Como o *swarm* 1 possui a menor carga, a probabilidade de um *peer* escolhido ao acaso optar por deixar esse *swarm* é igual à fração de *peers* que servem o conteúdo 1. Por outro lado, para escolher o *swarm* 2 é preciso que o *peer* seja *seed* em 2 e não seja *seed* em 1 (caso contrário teria optado pelo *swarm* 1). Portanto, se $\Pr(\bar{\omega}_i) = 1 - \Pr(\omega_i)$,

$$\Pr(R = 2) = \Pr(\omega_2 \cap \bar{\omega}_1) = \Pr(\omega_2) \cdot \Pr(\bar{\omega}_1 \mid \omega_2).$$

De modo geral,

$$\begin{aligned}
\Pr(R = k) &= \Pr(\omega_k \cap \bar{\omega}_{k-1} \cap \bar{\omega}_{k-2} \cap \cdots \cap \bar{\omega}_1) \\
&= \Pr(\omega_k) \cdot Pr(\bar{\omega}_{k-1} \mid \omega_k) \\
&\quad \cdot \Pr(\bar{\omega}_{k-2} \mid \bar{\omega}_{k-1} \cap \omega_k) \\
&\quad \cdot \Pr(\bar{\omega}_{k-3} \mid \bar{\omega}_{k-2} \cap \bar{\omega}_{k-1} \cap \omega_k) \\
&\quad \cdots \\
&\quad \cdot \Pr(\bar{\omega}_1 \mid \bar{\omega}_2 \cap \cdots \cap \bar{\omega}_{k-1} \cap \omega_k).
\end{aligned} \tag{4.5}$$

Sabemos que há dependência entre os eventos $\omega_i$ e $\bar{\omega}_j$ $(i \neq j)$. Consideremos, por exemplo, o caso em que todos os conteúdos são igualmente populares e a distribuição dos conteúdos servidos por um *peer* é uniforme. Nesse caso, se um *peer* é *seed* em $i$, é menos provável que ele seja *seed* em $j$ (já que todo *peer* serve $m$ conteúdos diferentes, selecionados de maneira aleatória e uniforme). Portanto, $\Pr(\bar{\omega}_j \mid \omega_i) > \Pr(\bar{\omega}_j)$. Ainda assim, podemos propor uma aproximação para a equação 4.5 assumindo que os eventos são mutuamente independentes:

$$\Pr(\hat{R} = k) = c_1 \cdot \Pr(\omega_k) \cdot \prod_{i=1}^{k-1} \Pr(\bar{\omega}_i), \tag{4.6}$$

onde

$$c_1 = \left\{ \sum_{j=1}^{C} \Pr(\omega_j) \cdot \prod_{i=1}^{j-1} \Pr(\bar{\omega}_i) \right\}^{-1}$$

[1] Um *peer* pode decidir não sair do *swarm* escolhido se não houver outro *swarm* com carga maior que ele esteja apto a servir.

é uma constante de normalização.

A probabilidade de um nano cache ser *seed* em $k$ é dada por

$$\Pr(\omega_k) = \frac{S_k}{N}, \tag{4.7}$$

já que $S_k$ dentre os $N$ *peers* do sistema estão servindo o vídeo $k$.

Por fim, aplicando (4.7) em (4.6), temos

$$\Pr(\hat{R} = k) = c_1 \cdot \frac{S_k}{N} \cdot \prod_{i=1}^{k-1} \left[1 - \frac{S_i}{N}\right]. \tag{4.8}$$

### 4.2.3 Probabilidade de entrada em um *swarm*

Agora iremos modelar o processo de entrada de um *peer* em um *swarm*. Definimos a variável aleatória $A$, que indica o *swarm* de menor carga dentre aqueles cujo conteúdo um *peer* aleatório possui. Deste modo, $\Pr(A = k)$ é igual à probabilidade de um nano cache escolhido ao acaso escolher o conteúdo $k$ como candidato para servir. Vamos assumir que um *peer* escolhe $k$ se e somente se possui $k$ armazenado e não possui os conteúdos posteriores a $k$ (de carga maior ou igual a $\ell_k$). Não consideramos aqui os conteúdos que o *peer* está servindo no momento. Se, por exemplo, o nano cache já estivesse servindo $k$, este não poderia ser escolhido.

Como assumimos que os *swarms* estão ordenados de acordo com sua carga, sabemos que o *swarm* $C$ possui carga maior ou igual à de todos os outros *swarms*. Se um *peer* possui o conteúdo distribuído em $C$, então ele deve optar por entrar nesse *swarm*.

Seja $\sigma_i$ o evento que indica que um *nano cache* possui o vídeo $i$. Logo

$$\Pr(A = C) = \Pr(\sigma_C).$$

Para escolher o *swarm* $C-1$, é preciso que o *peer* possua o conteúdo $C-1$ e não possua o conteúdo $C$. Portanto, se $\Pr(\bar{\sigma}_C)$ é a probabilidade de não ter o conteúdo $C$,

$$\Pr(A = C-1) = \Pr(\sigma_{C-1} \cap \bar{\sigma}_C) = \Pr(\sigma_{C-1}) \cdot \Pr(\bar{\sigma}_C \mid \sigma_{C-1}).$$

De modo geral,

$$\begin{aligned}\Pr(A = k) &= \Pr(\sigma_k \cap \bar{\sigma}_{k+1} \cap \bar{\sigma}_{k+2} \cap \cdots \cap \bar{\sigma}_C) \\ &= \Pr(\sigma_k) \cdot Pr(\bar{\sigma}_{k+1} \mid \sigma_k) \\ &\quad \cdot \Pr(\bar{\sigma}_{k+2} \mid \bar{\sigma}_{k+1} \cap \sigma_k) \\ &\quad \cdot \Pr(\bar{\sigma}_{k+3} \mid \bar{\sigma}_{k+2} \cap \bar{\sigma}_{k+1} \cap \sigma_k) \\ &\quad \cdots \\ &\quad \cdot \Pr(\bar{\sigma}_C \mid \bar{\sigma}_{C-1} \cap \cdots \cap \bar{\sigma}_{k+1} \cap \sigma_k).\end{aligned}$$

Assumimos novamente que os eventos são mutuamente independentes. Deste modo, temos a aproximação

$$\Pr(\hat{A} = k) = c_2 \cdot \Pr(\sigma_k) \cdot \prod_{i=k+1}^{C} \Pr(\bar{\sigma}_i), \tag{4.9}$$

onde

$$c_2 = \left\{ \sum_{j=1}^{C} \Pr(\sigma_j) \cdot \prod_{i=j+1}^{C} \Pr(\bar{\sigma}_i) \right\}^{-1}$$

é uma constante de normalização.

A probabilidade de um *peer* possuir o conteúdo $k$ é aproximada por uma distribuição binomial. Para isso, assumimos que os $B$ conteúdos em um nano cache são amostrados de maneira independente (com reposição) e que o conteúdo $k$ é escolhido com probabilidade $p_k$. Fornecemos duas aproximações para $\Pr(\sigma_k)$. A primeira é a probabilidade de amostrar uma ou mais vezes o conteúdo $k$:

$$\begin{aligned}\Pr(\hat{\sigma}_k^{(1)}) &= 1 - \left[ \binom{B}{0} \cdot p_k^0 \cdot (1 - p_k)^{B-0} \right] \\ &= 1 - (1 - p_k)^B.\end{aligned} \tag{4.10}$$

E a segunda aproximação corresponde à amostragem do conteúdo $k$ exatamente 1 vez:

$$\begin{aligned}\Pr(\hat{\sigma}_k^{(2)}) &= \binom{B}{1} \cdot p_k^1 \cdot (1 - p_k)^{B-1} \\ &= B \cdot p_k \cdot (1 - p_k)^{B-1}.\end{aligned} \tag{4.11}$$

Aplicando 4.10 em 4.9, temos:

$$\begin{aligned}\Pr(\hat{A}_1 = k) &= c_2 \cdot \left[1 - (1 - p_k)^B\right] \cdot \prod_{i=k+1}^{C} \left\{1 - \left[1 - (1 - p_i)^B\right]\right\} \\ &= c_2 \cdot \left[1 - (1 - p_k)^B\right] \cdot \prod_{i=k+1}^{C} (1 - p_i)^B.\end{aligned} \tag{4.12}$$

E, aplicando 4.11 em 4.9, temos:

$$\Pr(\hat{A}_2 = k) = c_2 \cdot B \cdot p_k \cdot (1 - p_k)^{B-1} \cdot \prod_{i=k+1}^{C} \left[1 - B \cdot p_i \cdot (1 - p_i)^{B-1}\right]. \tag{4.13}$$

Poderíamos calcular $\Pr(\sigma_k)$ do mesmo modo como calculamos $\Pr(\omega_k)$ (ver equação 4.7). Nesse caso, além dos vetores de *leechers* e *seeds*, precisaríamos armazenar um vetor com o número de cópias de cada conteúdo no sistema.

Os resultados de nossa avaliação através do método Monte Carlo (ver seção 4.2.5) indicam que a aproximação por uma distribuição binomial para $\sigma_k$, em média, insere um erro pequeno em relação à amostragem sem reposição. Usando essa aproximação, evitamos a adição de outro vetor de variáveis de estado.

### 4.2.4 Aproximação geométrica truncada

Propomos, por fim, uma aproximação trivial para a *PMF* das variáveis aleatórias $R$ e $A$ usando uma distribuição geométrica truncada:

$$\Pr(\hat{R}_g = k) = c_R \cdot q_R \cdot (1 - q_R)^{k-1}, \tag{4.14}$$

onde

$$c_R = \left[\sum_{i=1}^{C} q_R \cdot (1 - q_R)^{i-1}\right]^{-1} = \left[q_R \cdot \frac{1 - (1 - q_R)^C}{1 - (1 - q_R)}\right]^{-1} = \left[1 - (1 - q_R)^C\right]^{-1}$$

é uma constante de normalização, $m$ é o número de conteúdos servidos por cada nano cache e $q_R = \frac{m}{C}$ é a probabilidade de um *peer* estar servindo um *swarm* qualquer se todos os *swarms* possuem o mesmo número de *seeds*. Analogamente, para a probabilidade de entrada em cada *swarm*,

$$\Pr(\hat{A}_g = k) = c_A \cdot q_A \cdot (1 - q_A)^{C-k}, \tag{4.15}$$

onde

$$c_A = \left[\sum_{i=1}^{C} q_A \cdot (1-q_A)^{C-i}\right]^{-1} = \left[q_A \cdot \frac{1-(1-q_A)^C}{1-(1-q_A)}\right]^{-1} = \left[1-(1-q_A)^C\right]^{-1}$$

é uma constante de normalização, $B$ é o número de conteúdos que cada nano cache possui e $q_A = \frac{B}{C}$ é a probabilidade de um *peer* possuir um conteúdo qualquer se todos os conteúdos são igualmente populares.

Nesta seção, modelamos o processo de troca de *swarms* usando duas variáveis aleatórias que indicam a decisão tomada por um *peer* escolhido ao acaso:

- $\boldsymbol{R}$ (equações 4.8 e 4.14) – *swarm* de menor carga dentre aqueles que o *peer* serve. É o *swarm* que o *peer* deve deixar;
- $\boldsymbol{A}$ (ver equações 4.12, 4.13 e 4.15) – *swarm* de maior carga dentre aqueles cujo conteúdo o *peer* possui. É o *swarm* no qual o *peer* deve entrar.

Tendo escolhido os *swarms* $r$ e $a$ para sair e entrar, respectivamente, o *peer* efetua a troca de *swarms* se e somente se $\ell_r < \ell_a$. Caso contrário, ele permanece como *seed* nos mesmos *swarms* por pelo menos mais $\tau$ unidades de tempo.

Com essas duas variáveis aleatórias, podemos modelar a decisão tomada por um *peer* sem conhecer o seu estado, ou seja, os $m$ *swarms* que ele serve e os $B$ conteúdos que possui. Se representássemos o estado de cada *peer*, o espaço de estados cresceria exponencialmente com o número de nano caches. Portanto, essa aproximação é essencial para resolver modelos com números de *leechers* e *seeds* próximos aos de sistemas reais.

### 4.2.5 Validação

Validamos nossas aproximações estimando o erro entre a probabilidade de entrada e saída em cada *swarm* calculada com base no estado completo do sistema (matriz $N \times C$ definida na seção 4.2.1) e a probabilidade dada por $\hat{R}$, $\hat{R}_g$, $\hat{A}_1$, $\hat{A}_2$ e $\hat{A}_g$ (com o estado representado por dois vetores de tamanho $C$). Para isso, usamos o método Monte Carlo, que pode ser resumido nestes passos:

1. definir um domínio de valores de entrada possíveis;
2. gerar uma entrada aleatória segundo uma distribuição de probabilidade;
3. efetuar um cálculo determinístico a partir da entrada;
4. repetir os passos 2 e 3 diversas vezes;
5. calcular uma estatística a partir dos resultados das diversas rodadas.

Tabela 4.1: Descrição dos parâmetros da validação com o método Monte Carlo

| Parâmetro | Definição |
|---|---|
| $N$ | Número de nano caches (*peers*) no sistema |
| $C$ | Número de conteúdos disponíveis no sistema |
| $B$ | Número de conteúdos que cabem em um cache |
| $m$ | Número de conteúdos que um nano cache serve |
| $V$ | Número de conteúdos assistidos por cada *peer* |
| $q_L$ | Probabilidade de um *peer* estar assistindo a um vídeo |
| $\alpha$, $\mu$ | Parâmetros da distribuição de popularidade |

Na tabela 4.1, apresentamos uma descrição dos parâmetros do algoritmo. Cada entrada $X_i$ é uma matriz $N \times C$ que indica os conteúdos que os *peers* assistem, servem e armazenam (possuem, mas não servem), conforme definido na seção 4.2.1. O Algoritmo 1 mostra o processo de escolha de uma nova entrada $X_i$. Para cada um dos $N$ *peers*, primeiro simulamos $V$ visualizações de conteúdos amostrados de maneira independente (com reposição) segundo uma distribuição de popularidade Zipf com truncamento exponencial (ver seção 5.1.2). Nesse processo, assumimos que o *peer* adota uma política de substituição aleatória de arquivos no cache. Após a simulação, obtemos um conjunto $A$ de vídeos que ele possui. Com probabilidade $q_L$, o *peer* assiste a algum vídeo. Nesse caso, amostramos um conteúdo a mais durante a simulação. Por fim, escolhemos aleatoriamente (sem reposição) um subconjunto de $A$ contendo $m$ vídeos, que representa os conteúdos servidos pelo nano cache. E, caso o *peer* seja um *leecher*, escolhemos mais um elemento aleatório de $A$ (o vídeo ao qual ele assiste).

Ao final do algoritmo, cada posição $[j, k]$ da entrada $X_i$ contém um valor de 0 a 3 que indica:

0. o *peer* $j$ não possui o conteúdo $k$ e não está assistindo-o;
1. o *peer* $j$ armazena o conteúdo $k$;
2. o *peer* $j$ serve o conteúdo $k$;
3. o *peer* $j$ assiste ao conteúdo $k$.

Na figura 4.2 ilustramos uma possível entrada gerada pelo Algoritmo 1.

Após a escolha de uma entrada aleatória $X_i$, calculamos a carga em cada *swarm* (ver equação 4.1) e ordenamos as colunas da matriz de modo a deixar os *swarms* em ordem crescente de carga. Em seguida, verificamos para cada linha $X_i[j]$ de $X_i$:

1. o índice do primeiro valor (mais à esquerda) igual a 2 – essa posição de $X_i[j]$ corresponde ao *swarm* de menor carga dentre aqueles que o *peer* serve e, portanto, é candidato para deixar de ser servido;

**Algoritmo 1:** Escolha de uma entrada $X_i$ para o método Monte Carlo.

**Entrada**: $N$, $C$, $B$, $m$, $V$, $q_L$, $\mathbf{p}$

$X_i \leftarrow \underbrace{\left.\begin{array}{|c|c|c|}\hline 0 & \dots & 0 \\ \hline \vdots & \ddots & \vdots \\ \hline 0 & \dots & 0 \\ \hline \end{array}\right\}_N}_{C};$

`// definição do estado de cada nano cache`
**para** $j \leftarrow 1$ **até** $N$ **faça**
  $A \leftarrow \emptyset$ ; `// conjunto de conteúdos armazenados no cache`
  $c \leftarrow B$ ; `// número de conteúdos a serem amostrados`
  $u \leftarrow U(0,1)$ ; `// variável aleatória uniforme`
  **se** $u \leq q_L$ **então**
    $c \leftarrow c + 1$ ; `// o peer assiste a um conteúdo`
  **fim**
  `// escolha dos conteúdos que o peer já assistiu`
  `// e estão armazenados no cache`
  **para** $k \leftarrow 1$ **até** $V$ **faça**
    escolha conteúdo aleatório $v$ segundo distribuição de popularidade $\mathbf{p}$;
    **se** $v \notin A$ **então**
      **se** $|A| = c$ **então**
        remova um conteúdo aleatório de $A$;
      **fim**
      $A \leftarrow A \cup \{v\}$;
    **fim**
  **fim**
  `// conteúdos que o peer possui`
  **para todo** $k \in A$ **faça**
    $X_i[j,k] \leftarrow 1$;
  **fim**
  `// conteúdos servidos pelo peer`
  escolha subconjunto aleatório $S$ de $A$ com $m$ elementos;
  **para todo** $k \in S$ **faça**
    $X_i[j,k] \leftarrow 2$;
  **fim**
  `// conteúdo assistido pelo peer`
  **se** $u \leq q_L$ **então**
    escolha elemento aleatório $a$ de $A - S$;
    $X_i[j,a] \leftarrow 3$;
  **fim**
**fim**
**retorna** $X_i$;

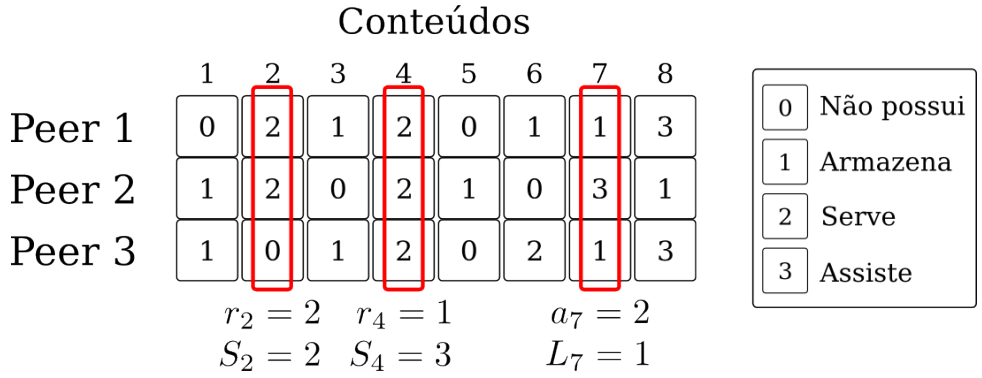



Figura 4.2: Exemplo de entrada gerada pelo Algoritmo 1 para $N = 3$ *peers*, $C = 8$ conteúdos disponíveis, $B = 5$ conteúdos em cache e $m = 2$ conteúdos servidos. Os *swarms* encontram-se em ordem crescente de carga.

2. o índice do último valor (mais à direita) igual a 1 – essa posição de $X_i[j]$ corresponde ao *swarm* de maior carga dentre aqueles que o *peer* armazena e, portanto, é candidato para passar a ser servido.

Seja $r_k$ o número de linhas em $X_i$ nas quais a posição do primeiro valor igual a 2 é $k$. Esse é o número de *peers* que escolhem deixar o *swarm* $k$. A probabilidade de um *peer* escolhido ao acaso optar por sair de $k$ é dada por:

$$\Pr(R = k) = \frac{r_k}{N}. \tag{4.16}$$

Analogamente, a probabilidade de um *peer* escolher passar a servir o conteúdo $k$ é dada por:

$$\Pr(A = k) = \frac{a_k}{N}, \tag{4.17}$$

onde $a_k$ é o número de linhas em $X_i$ nas quais o índice da última entrada igual a 1 é $k$. No exemplo da figura 4.2, podemos ver que apenas o *peer* 3 optaria por deixar o *swarm* 4 ($r_4 = 1$) e que dois *peers* (1 e 3) escolheriam passar a servir o conteúdo 7 ($a_7 = 2$).

Para a mesma entrada $X_i$, calculamos também as distribuições de probabilidade dadas por nossas aproximações (equações 4.8, 4.12, 4.13, 4.14 e 4.15). Nesse caso, o cálculo é feito apenas com base no número de *seeds* e *leechers* em cada *swarm*: para um *swarm* $k$, $S_k$ e $L_k$ são, respectivamente, o número de entradas iguais a 2 e 3 na coluna $k$ de $X_i$. Vemos que na figura 4.2 há 2 *seeds* no *swarm* 2 ($S_2 = 2$) e 1 *leecher* no *swarm* 7 ($L_7 = 1$).

Para avaliar a qualidade de nossas aproximações, usamos como métrica a raiz do erro médio quadrático (RMSE - *Root Mean Square Error*), dada por:

Tabela 4.2: Valores de parâmetros adotados

| Parâmetro | Valor |
|---|---|
| $N$ | 40000 |
| $C$ | 250 |
| $B$ | 20 |
| $m$ | 4 |
| $V$ | 50 |
| $q_L$ | 0,1 |
| $\alpha$ | 0,63 |
| $\mu$ | 0,02 |

$$REMQ(\hat{F}, X_i) \triangleq \sqrt{\frac{1}{C} \cdot \sum_{k=1}^{C} [\hat{F}_k(X_i) - F_k(X_i)]^2}, \tag{4.18}$$

onde $\hat{F}$ é um vetor de valores aproximados e $F$ é um vetor de valores reais. No nosso caso, comparamos para uma entrada $X_i$ os valores "reais" dados por $\Pr(R = k)$ e $\Pr(A = k)$ (equações 4.16 e 4.17) com os obtidos a partir das aproximações. E, a fim de fazer uma comparação independente de escala, usamos também a raiz do erro médio quadrático normalizada, definida usualmente como:

$$REMQN(\hat{F}, X_i) \triangleq \frac{REMQ(\hat{F}, X_i)}{\max_k F_k(X_i) - \min_k F_k(X_i)}. \tag{4.19}$$

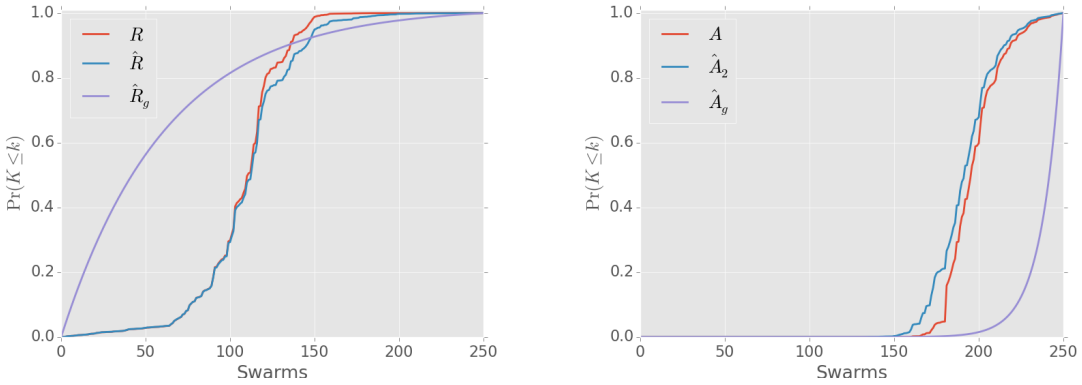
Acrescentamos que, no nosso caso, não podemos aplicar um teste de hipótese $\chi^2$ para comparar as duas distribuições, pois para uma entrada de tamanho $N = 40000$ por $C = 250$ obtemos classes (*swarms*) com poucas ou mesmo nenhuma amostra (poucos *peers* escolhem entrar ou sair desses *swarms*). Para tornar o teste válido, precisaríamos simular um número muito maior de *peers*.

Implementamos o método de validação descrito usando a linguagem Python e a biblioteca para computação científica SciPy [31], que permite o acesso a implementações eficientes de algoritmos para otimização, álgebra linear, estatística e processamento de sinais. Calculamos a média das métricas $REMQ$ e $REMQN$ para cada aproximação com 500 entradas. Os valores dos parâmetros, escolhidos a partir de dados de um sistema real (ver seção 5.1), são listados na tabela 4.2. O valor de $V$ foi escolhido de modo a permitir o preenchimento dos caches (com tamanho $B = 20$ vídeos) sem inviabilizar a execução do Algoritmo 1 para gerar muitas entradas $X_i$.

Na tabela 4.3, apresentamos os resultados com os respectivos intervalos de confiança de 95%. Nossas aproximações levaram, em média, a um erro pequeno, com $\overline{REMQN}$ menor do que 1,5% para a variável $R$ e menor do que 3% para $A$. Observamos também que uma variável aleatória geométrica truncada não modela bem o processo de escolha dos *swarms* nos quais um *peer* deve entrar ou sair, implicando

Tabela 4.3: Resultados da avaliação com método Monte Carlo (V=50)

| Aproximação | $\overline{REMQ}$ | $\overline{REMQN}$ |
|---|---|---|
| $\hat{R}$ | $1,10 \cdot 10^{-3} \pm 3,25 \cdot 10^{-6}$ | $1,38\% \pm 0,02$p.p. |
| $\hat{R}_g$ | $1,10 \cdot 10^{-2} \pm 5,95 \cdot 10^{-5}$ | $13,7\% \pm 0,19$p.p. |
| $\hat{A}_1$ | $2,93 \cdot 10^{-3} \pm 8,50 \cdot 10^{-5}$ | $2,77\% \pm 0,09$p.p. |
| $\hat{A}_2$ | $2,47 \cdot 10^{-3} \pm 3,23 \cdot 10^{-5}$ | $2,33\% \pm 0,05$p.p. |
| $\hat{A}_g$ | $1,71 \cdot 10^{-2} \pm 7,07 \cdot 10^{-5}$ | $16,1\% \pm 0,29$p.p. |




(a) Probabilidade de saída ($R$)

(b) Probabilidade de entrada ($A$)

Figura 4.3: Comparação entre as distribuições para o pior caso. *Swarms* em ordem crescente de carga.

em um $\overline{REMQN}$ de mais de 10%. Quanto às aproximações para $\Pr(A = k)$, vemos que as duas alternativas, $\hat{A}_1$ e $\hat{A}_2$, fornecem resultados próximos, tendo $\hat{A}_2$ um erro menor. Na figura 4.3, comparamos a função distribuição acumulada (CDF - *cumulative distribution function*) de $R$ e $A$ com suas aproximações para o pior caso (maior erro) dentre as 500 entradas aleatórias. Mesmo nesse cenário, as aproximações mostraram-se bastante satisfatórias.

Concluímos que o uso das variáveis $\hat{R}$ e $\hat{A}_2$ na modelagem do processo de troca de *swarms* permite um redução expressiva no espaço de estados sem introduzir um erro significativo. Além disso, nossas aproximações possuem a vantagem de serem simples e, portanto, fáceis de serem introduzidas em outros modelos.

## 4.3 Tangram-II

Nesta seção, descrevemos um modelo de simulação desenvolvido no ambiente de modelagem, simulação e experimentação Tangram-II [29]. Na ferramenta, um modelo é definido a partir de suas variáveis de estado, parâmetros (constantes), eventos, mensagens e recompensas.

Nosso modelo possui como variáveis de estado dois vetores de tamanho $C$ que

representam o número de *leechers* e *seeds* em cada *swarm* [2]. O vetor de *leechers*, **L**, é inicializado com zeros, indicando que, inicialmente, não há usuários assistindo a vídeos. Como há $N$ nano caches no sistema e cada um serve exatamente $m$ conteúdos, sabemos que há no total $N \cdot m$ *seeds* distribuídos entre os $C$ *swarms*. Cada posição do vetor de *seeds*, **S**, é inicializada com um número de *seeds* proporcional à popularidade de cada conteúdo, de modo que $\sum_{i=1}^{C} S_i = N \cdot m$.

Além destes, definimos outro vetor de variáveis de estado auxiliar que indica a taxa média de *download* em cada *swarm*. Esse vetor é função de **L** e **S** e é atualizado através de uma mensagem sempre que o valor das outras variáveis de estado é alterado. Para calcular a taxa média de *download* de um *leecher*, assumimos que:

1. todos os *leechers* em um *swarm* possuem a mesma taxa média de *download*;
2. todos os nano caches saturam sua banda de *upload*, que é igualmente dividida entre os $m$ *swarms* que servem;
3. os nano caches não servem conteúdos incompletos que estão sendo assistidos;
4. o servidor de conteúdo, com banda de *upload* $U_s$, não serve os *swarms* nos quais os *leechers* saturam sua banda de *download*, isto é, baixam o vídeo a uma taxa média igual a $D_{nc}$ (banda disponível).

A taxa média de *download* de todos *leechers* no *swarm* $k$, sem a participação do servidor de conteúdo, é dada por:

$$\bar{D}'_k = \frac{U_{nc} \cdot S_k}{m \cdot L_k},$$

onde $U_{nc}$ é a banda disponível nos nano caches para *upload*, $m$ é o número de conteúdos servidos por cada *peer* e $\frac{U_{nc}}{m}$ é a banda de *upload* oferecida pelos *seeds* aos *leechers* do *swarm* $k$. Acrescentando a banda do servidor, temos:

$$\bar{D}_k = \min\left(\bar{D}'_k + \frac{U_s}{Z}, D_{nc}\right), \tag{4.20}$$

onde $Z$ é o número de *leechers* que não saturam sua banda, ou seja, a soma do número de *leechers* nos *swarms* $i$ tal que $\bar{D}'_i < D_{nc}$, $1 \leq i \leq C$. Para calcular a taxa média de *download* de todos os *leechers*, independente do *swarm*, definimos mais uma variável auxiliar dada por:

$$\bar{D} = \frac{\sum_{i=1}^{C} \bar{D}_i \cdot L_i}{\sum_{i=1}^{C} L_i}.$$

Sempre que modificamos alguma posição do vetor **D**, recalculamos sua média $\bar{D}$.

[2] Utilizando o modelo para o processo de troca de *swarms* descrito na seção anterior, não precisamos de outro vetor de variáveis de estado para armazenar o número de réplicas de cada conteúdo.

Os parâmetros do modelo são os mesmos definidos na tabela 3.1. Assumimos que todos os vídeos possuem a mesma duração e taxa de codificação. O modelo pode ser facilmente estendido para o caso mais geral em que há diferença de duração e *bitrate* entre os conteúdos.

Nosso modelo possui três tipos de eventos:

1. **chegada de um *leecher* no *swarm*** $k$ - ocorre com taxa $\lambda \cdot p_k$, desde que $\sum_{i=1}^{C} L_i < N$ (número máximo de *leechers* no sistema) e $L_k + S_k < N$ (número máximo de *peers* em um *swarm*). Quando esse evento é processado, o valor de $L_k$ é incrementado em uma unidade;

2. **término de *download* no *swarm*** $k$ - ocorre com taxa $L_k \cdot \frac{D_k}{d \cdot b}$, onde $d \cdot b$ corresponde ao tamanho do vídeo em bits. Este evento é disparado apenas se há *leechers* em $k$ ($L_k > 0$). Quando processado, decrementa o valor de $L_k$ em uma unidade.

3. **troca de *swarms*** - ocorre com taxa $\frac{1}{\tau} \cdot N$. Quando processado, os *swarms* são ordenados em ordem crescente de carga e dois *swarms*, $r$ e $a$, são escolhidos aleatoriamente segundo as aproximações $\hat{R}$ e $\hat{A}_2$ (ver equações 4.8 e 4.13), respectivamente. Se $\ell_r < \ell_a$, a troca ocorre e $S_a$ é incrementado em uma unidade e $S_r$ decrementado. Caso contrário, as variáveis de estado se mantêm inalteradas.

A ferramenta Tangram-II permite, em seu módulo de simulação, que o tempo entre a ocorrência de dois eventos seja modelado por diversas distribuições diferentes. Ainda assim, a fim de simplificar nossa análise, assumimos que o intervalo entre eventos segue uma distribuição exponencial. Mostramos na seção 5.1.1 que essa é uma hipótese bastante razoável para o tempo entre chegadas de *leechers*.

Com o Tangram-II, obtemos medidas de desempenho através de recompensas, que podem estar associadas a determinados estados do sistema (recompensas de taxa) ou a transições entre estados (recompensas de impulso). Em nosso modelo, definimos sete tipos de recompensas:

1. **número de *leechers* no *swarm*** $k$ - recompensa de taxa associada à variável de estado $L_k$;

2. **número de *seeds* no *swarm*** $k$ - recompensa de taxa associada à variável de estado $S_k$;

3. **carga no *swarm*** $k$ - recompensa de taxa associada a $\ell_k$, função das variáveis $L_k$ e $S_k$;

4. **chegada de um *leecher* no *swarm*** $k$ - recompensa de impulso associada à ocorrência do evento correspondente;

5. **taxa média de *download* no *swarm*** $k$ - recompensa de taxa que nos permite calcular a média de $\bar{D}_k$ no tempo quando $L_k > 0$;

6. **reprodução satisfatória no *swarm*** $k$ - recompensa de taxa que nos permite calcular a fração de tempo em que $\bar{D}_k > b$ e $L_k > 0$;

7. **taxa média de *download*** - recompensa de taxa que nos permite calcular a média de $\bar{D}$ no tempo;

As variáveis de estado, parâmetros, eventos, mensagens e recompensas do modelo são especificadas na ferramenta usando uma linguagem semelhante à linguagem C. No total, precisamos definir $(2 \cdot C + 1)$ eventos e $(6 \cdot C + 1)$ recompensas diferentes em nosso modelo, o que poderia inviabilizar a construção de um modelo com um número grande de *swarms*. Todavia, desenvolvemos um *software* em Python capaz de gerar automaticamente a especificação do modelo para valores grandes de $C$ usando o mecanismo de *templates* Jinja [32]. O *software* também permite a geração em lote de modelos com diferentes especificações e parâmetros. Assim, podemos, por exemplo, criar diversos modelos variando o valor da taxa de chegada de *leechers* ou o intervalo entre trocas de *swarms*.

No próximo capítulo, apresentamos os resultados das simulações realizadas com nosso modelo no Tangram-II.

# Capítulo 5

# Resultados

No capítulo anterior, apresentamos detalhes do modelo desenvolvido com a ferramenta Tangram-II. Primeiramente, apresentamos neste capítulo uma análise estatística de dados de um sistema real de vídeo sob demanda. A distribuição de popularidade que consideramos no modelo foi baseada na análise desses dados. Em seguida, discutimos os cenários de simulação e parâmetros utilizados e mostramos os resultados obtidos. Por fim, descrevemos os resultados que obtivemos através de experimentos com nosso protótipo de nano cache.

## 5.1 Análise estatística de um sistema real

Encontramos na literatura trabalhos voltados para a análise de dados de sistemas reais de VoD como Youtube [15, 33], MSN Video [34], PowerInfo (do provedor de banda larga China Telecom) [33, 35], PPLive [17] e Yahoo! video [16]. Nesta seção, fornecemos uma análise estatística de dados de um sistema de vídeo sob demanda com mais de 20.000 conteúdos disponíveis. Temos como objetivo estimar parâmetros para nosso modelo que reflitam características de um sistema real.

O sistema de VoD que analisamos faz uso da plataforma Cisco TV CDS (*Content Delivery Systems*) [36], composta por uma rede de servidores de distribuição chamados CDEs (*Content Delivery Engines*). Esses servidores fazem o *streaming unicast* do conteúdo usando uma rede *Gigabit Ethernet* até dispositivos de radiofrequência, que modulam os dados usando QAM (*Quadrature Amplitude Modulation*) e o enviam ao STB na casa do usuário através de uma rede híbrida fibra-coaxial (HFC). Na figura 5.1 vemos um diagrama ilustrando o funcionamento do sistema.

Para a análise apresentada nesta seção, utilizamos *logs* do sistema que incluem registros de acesso de 19 municípios brasileiros no período de 1º de dezembro de 2013 a 28 de fevereiro de 2014. Cada registro do *log* corresponde ao início de uma sessão de *streaming*. Na tabela 5.1 podemos ver um exemplo dos dados. Dentre as informações disponíveis, temos:

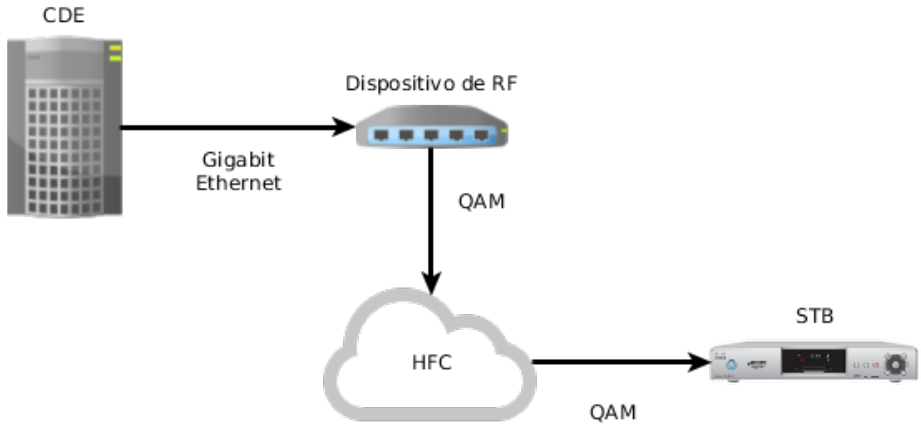



Figura 5.1: *Streaming* de vídeo sob demanda usando a plataforma Cisco TV CDS.

- **Identificador do cliente** - código do assinante.
- **Identificador do dispositivo** - código do STB que iniciou a sessão. Um cliente pode ter mais de um STB em casa.
- **Regional** - região em que se encontra o assinante. O país é subdividido em diversas regionais, que não seguem a distribuição usual do Brasil em 5 regiões. Analisamos dados de clientes das regionais Leste, São Paulo e Sul.
- ***Cluster*** - *cluster* do qual faz parte o usuário. Cada regional é dividida em um ou mais *clusters*.
- **Cidade** - município do cliente.
- ***Timestamp*** - data e hora de início do *streaming*, com precisão de minutos.
- **Título** - título do conteúdo assistido. No caso de alguns desenhos animados, séries e programas de TV, não é possível identificar o acesso a diferentes episódios. Esses episódios são identificados por um mesmo título nos *logs*.
- **Provedor** - provedor do conteúdo assistido.
- **Definição** - resolução do vídeo: HD ou SD.

Os dados não incluem o horário de término da sessão de *streaming*. Essa informação permitiria o cálculo da duração de cada sessão, estatística analisada em outros trabalhos [34, 35]. Além disso, não possuímos informações sobre ações interativas do usuário (como avançar, retroceder ou pausar a reprodução).

Na tabela 5.2, apresentamos um sumário das estatísticas dos *logs* analisados. O *log* contém mais de 1 milhão de sessões iniciadas por mais de 25 mil usuários.

Tabela 5.1: Amostra dos *logs* de acesso do sistema.

| Cliente | STB | Regional | Cluster | Cidade | *Timestamp* | Título | Provedor | Definição |
|---|---|---|---|---|---|---|---|---|
| 11111 | 1111111 | São Paulo | São Paulo | Embu | 2013-12-07 20:11:00 | Filme A | Telecine | HD |
| 22222 | 2222222 | Sul | Curitiba | Colombo | 2014-01-20 01:50:00 | Filme B | HBO | HD |
| 33333 | 3333333 | Leste | Nordeste | Olinda | 2014-02-13 12:01:00 | Filme C | Globosat | HD |
| ... | ... | ... | ... | ... | ... | ... | ... | |

No período de 90 dias, foram assistidos 5.267 conteúdos diferentes. Na figura 5.2, podemos ver um gráfico indicando o número de acessos por dia ao longo desse período. Vemos que essa variável possui um comportamento periódico, com picos aos domingos.

Tabela 5.2: Sumário das estatísticas dos *logs* de acesso do sistema.

| Sessões | Clientes | Dispositivos | Conteúdos | Cidades | Período |
|---|---|---|---|---|---|
| 1.004.829 | 25.933 | 38.299 | 5.267 | 19 | 90 dias |

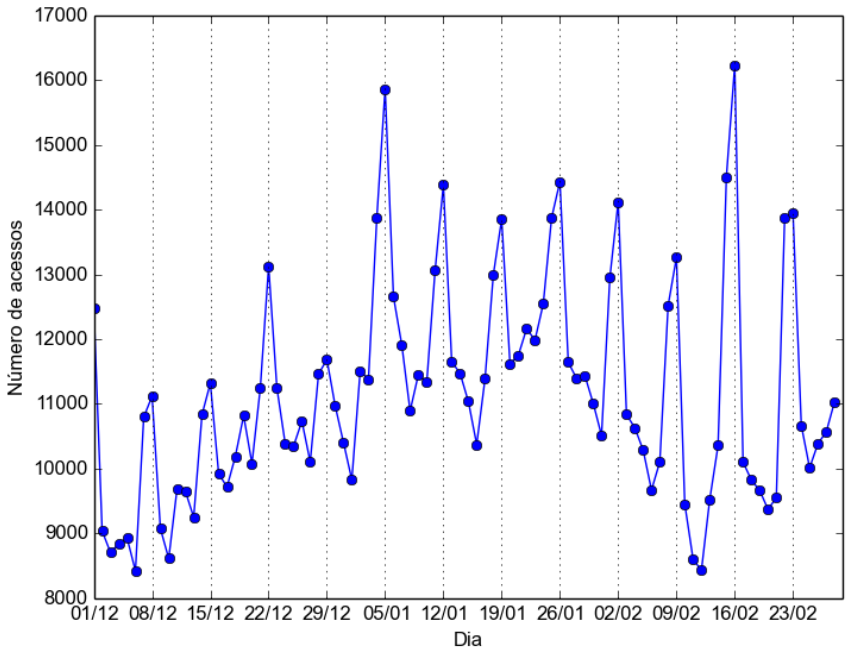



Figura 5.2: Número de acessos por dia de 01/12/2013 a 28/02/2014. As linhas verticais pontilhadas indicam os domingos.

### 5.1.1 Processo de chegadas

Analisamos agora a dinâmica de acessos ao sistema. O conhecimento quanto ao padrão de acesso dos usuários é fundamental para o dimensionamento de recursos em um sistema de VoD. Além disso, precisamos validar a hipótese usualmente assumida

de que o número de acessos em um determinado intervalo de tempo segue uma distribuição de Poisson.

Primeiramente, fazemos uma análise do padrão de acessos ao longo do dia. Vemos na figura 5.3 o número total de acessos durante cada hora do dia ao longo dos 90 dias de *logs*. Cada ponto do gráfico indica o número de acessos entre Xh00 e Xh59. Identificamos um período de 11 horas, entre 15h00 e 1h59, em que o número de visualizações é maior do que no restante do dia. Cerca de 67% dos acessos ocorreram nesse período. Vemos também que durante a hora de pico - entre 22h00 e 22h59 - a quantidade de acessos, 66149, não está distante da média do período de maior tráfego (61362 acessos). Na figura 5.4, plotamos a autocorrelação do número de acessos por minuto para diferentes valores de *lag*. Vemos um pico equivalente ao período de 24 horas, mostrando que o comportamento tende a se repetir diariamente.

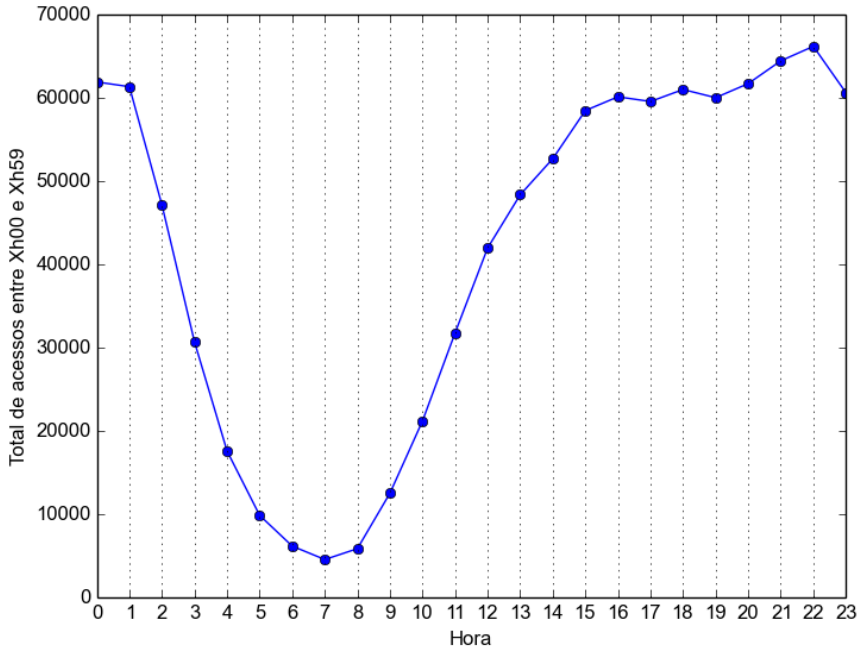



Figura 5.3: Total de acessos por hora durante os 3 meses. Cada ponto indica o número de acessos entre Xh00 e Xh59.

Consideramos agora a variável aleatória que representa o número de acessos por minuto, que é a granularidade de nossos dados. Mostramos na figura 5.5 um histograma indicando a fração de amostras com um determinado número de acessos. Cada amostra é o número $N$ de entradas no *log* em um intervalo de um minuto no período de 01/12/2013 a 28/02/2014. Notamos que essa distribuição apresenta um comportamento aparentemente bimodal, com picos em $N = 1$ e $N = 9$. É comum que isso ocorra quando há mistura de distribuições. Plotamos então na figura 5.6 o mesmo gráfico utilizando apenas as amostras em horários de pico (entre 15h e 2h).

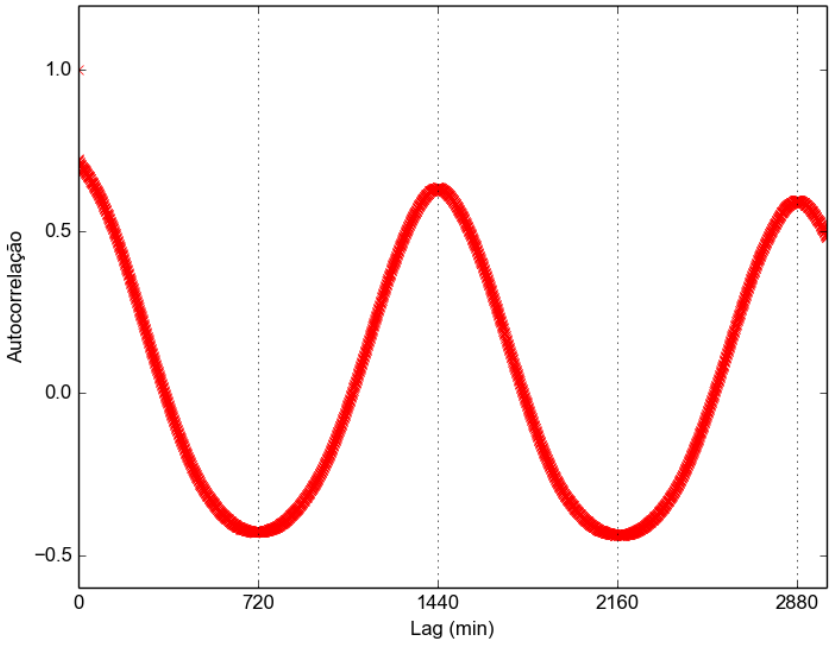



Figura 5.4: Autocorrelação do número de acessos por minuto ao longo do período de 90 dias. Os valores destacados no eixo $x$ indicam intervalos de 12 horas.

Juntamente com o histograma, vemos a função de probabilidade de massa de uma variável Poisson com parâmetro $\hat{\lambda} = 11,36$ estimado de acordo com o método dos momentos (igual à média amostral). Nesse caso, vemos que a variável pode ser bem modelada por uma distribuição de Poisson, como pode ser confirmado pelo gráfico de probabilidade (*probability plot*) mostrado na figura 5.7. A distância pequena dos pontos em relação à reta e o alto coeficiente de correlação ($r = 0.9925$) indicam que esse é um bom modelo.

### 5.1.2 Distribuição de popularidade dos conteúdos

Depois de analisar o processo de chegadas do sistema a partir dos dados obtidos, examinamos como as requisições dos usuários se distribuem entre os conteúdos oferecidos. O conhecimento da distribuição de popularidade dos conteúdos de um sistema de VoD é importante, por exemplo, para a definição de tamanhos de *cache* e políticas de replicação. Trabalhos anteriores mostram que é comum que o número de acessos a cada conteúdo siga uma distribuição de lei de potência em sistemas multimídia [16, 37]. Por isso, assume-se com frequência [4, 5, 7–9, 11, 12] a premissa de que a popularidade dos conteúdos segue uma distribuição Zipf. Entretanto, isso pode não ser válido para sistemas reais [15, 16]. Desejamos, portanto, verificar se essa hipótese - que faz com que um pequeno conjunto de vídeos seja responsável por grande parte dos acessos - se aplica ao sistema em análise.

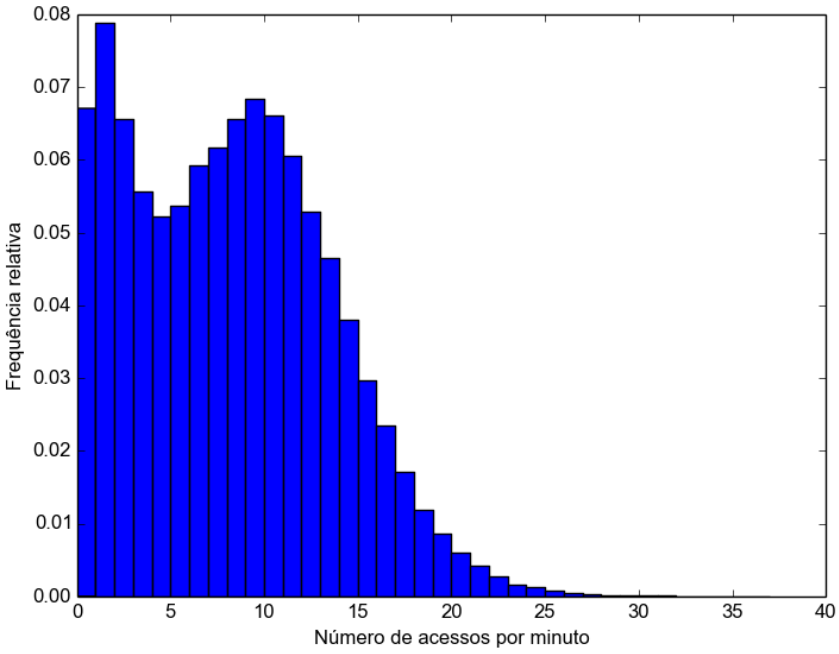



Figura 5.5: Histograma normalizado do número de acessos por minuto.

Para estudar o modo como as requisições dos usuários se dividem entre os vídeos mais e menos populares, adotamos duas representações complementares que são utilizadas com frequência na análise de leis de potência [38, 39]. A primeira, atribuída a Pareto [40], consiste em uma função distribuição acumulada complementar (CCDF - *complementary cumulative distribution function*). No nosso caso, plotamos a **fração de conteúdos com mais de $x$ visualizações**. A segunda representação é conhecida como gráfico *rank-frequency* e foi proposta por Zipf [41]. Nela, ordenamos os conteúdos em ordem decrescente de visualizações e indicamos a **fração de visualizações que o $r$-ésimo conteúdo recebeu**, onde $r$ é o seu *rank* (posição).

Nos dois casos, estamos mais interessados na cauda à direita da distribuição. No gráfico de Pareto, a cauda se refere aos conteúdos mais populares, enquanto que no gráfico *rank-frequency* a cauda se refere aos menos populares. Assim, podemos com as duas representações analisar os vídeos mais e menos acessados [15, 42].

### Distribuição do número de visualizações de um conteúdo

Em [40], Pareto apresenta uma análise da distribuição de receita anual de um grupo de habitantes da Europa. Esse trabalho motivou a criação da distribuição de Pareto, que é uma distribuição contínua de lei de potência. Desejamos verificar se o número de acessos aos conteúdos do sistema segue uma distribuição de Pareto discreta, conhecida como Zeta.

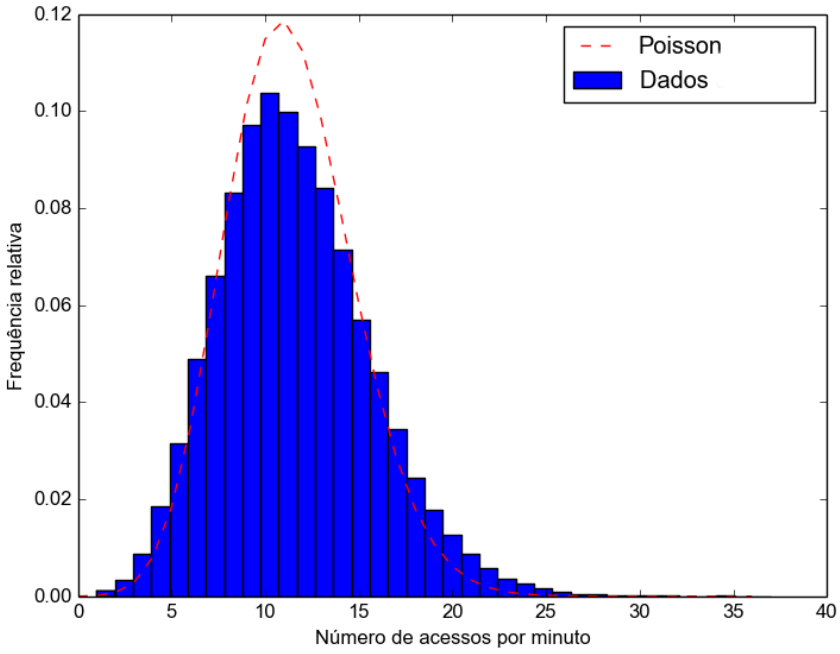



Figura 5.6: Histograma normalizado do número de acessos por minuto no período de alto tráfego (15h a 2h).

Na figura 5.8, vemos uma CCDF empírica em escala log-log que indica no eixo $y$ a fração de vídeos com mais do que $x$ visualizações. Notamos, por exemplo, que cerca de 90% dos conteúdos foram vistos menos de 400 vezes, enquanto os 0,1% vídeos mais populares foram assistidos, cada um, mais de 10000 vezes.

Dados que seguem uma lei de potência abrangem várias ordens de grandeza e se apresentam como uma reta no gráfico com os dois eixos em escala logarítmica [43]. Em nosso caso, não podemos fazer essa afirmação apenas através de uma avaliação gráfica. Seguimos, portanto, a metodologia sugerida por Clauset *et al.* [44], que permite identificar se um determinado conjunto de dados segue ou não uma distribuição de lei de potência (Pareto para variáveis contínuas ou Zeta para discretas). Essa metodologia consiste em três passos:

1. Estimativa dos parâmetros do modelo de lei de potência.
2. Cálculo do *goodness-of-fit* entre os dados e o modelo.
3. Comparação da distribuição de lei de potência com outros possíveis modelos.

Utilizamos então a biblioteca *Python powerlaw* [45], que fornece uma implementação do método definido em [44]. Primeiramente, estimamos os parâmetros de uma distribuição discreta Zeta com PMF:

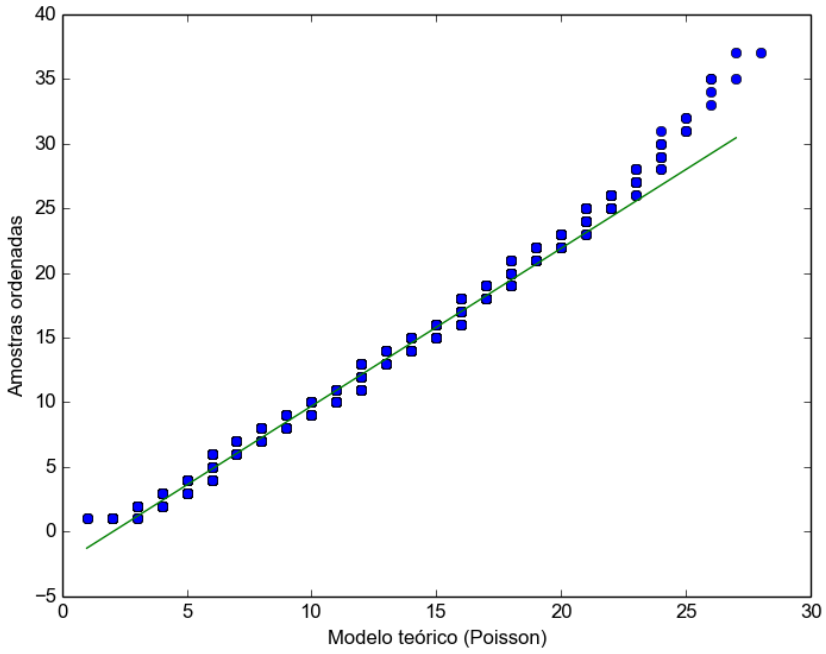



Figura 5.7: Gráfico de probabilidade comparando amostras de acessos por minuto no período de alto tráfego (15h a 2h) com distribuição de Poisson.

$$P[X = x] = \frac{x^{-\alpha}}{\zeta(\alpha, x_{min})},$$

onde

$$\zeta(\alpha, x_{min}) = \sum_{i=0}^{\infty} (i + x_{min})^{-\alpha}$$

é a função zeta de Hurwitz. O parâmetro $\alpha$ é conhecido como expoente ou parâmetro de escala e $x_{min}$ é o limite inferior a partir do qual a distribuição empírica passa a exibir um comportamento de lei de potência. O expoente $\alpha$ é estimado usando o método de máxima verossimilhança. Já o limite inferior $x_{min}$ pode ser identificado visualmente a partir do gráfico da CCDF ou estimado minimizando-se a distância entre a distribuição empírica e o modelo com parâmetro $\alpha$ estimado para valores maiores ou iguais a $x_{min}$. A métrica de distância entre distribuições usada por Clauset *et al.* é a estatística KS (Kolmogorov-Smirnov), definida como:

$$D = \max_{x \geq x_{min}} |P(x) - \hat{P}(x)|,$$

onde $P(x)$ é a CDF empírica e $\hat{P}(x)$ é a CDF do modelo que melhor se ajusta aos dados para $x \geq x_{min}$. O valor do estimador $\hat{x}_{min}$ é o que minimiza $D$. Com o *fitting* de nossos dados, obtivemos os valores $\hat{x}_{min} = 931$ e $\hat{\alpha} = 2,3761$.

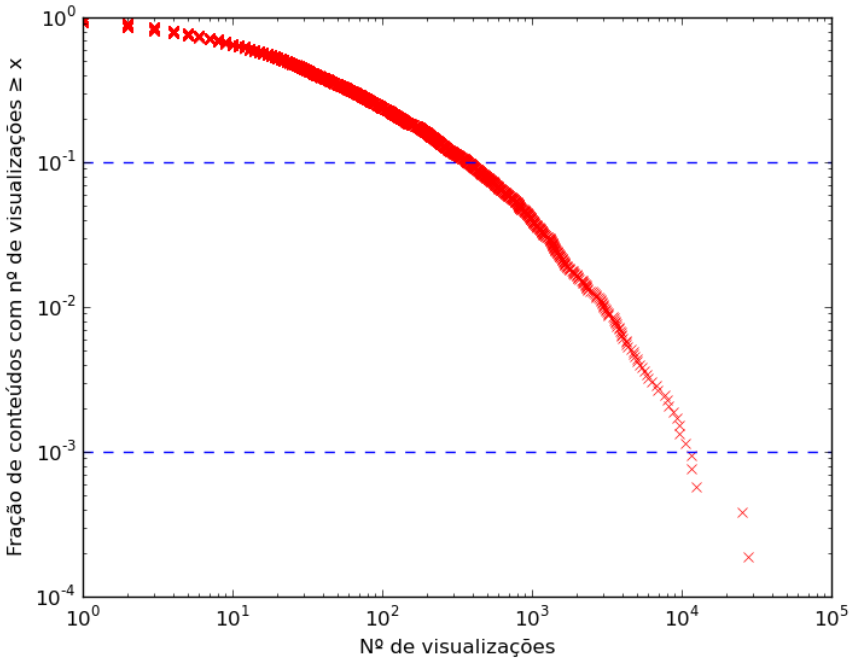



Figura 5.8: Gráfico da função distribuição acumulada complementar (CCDF) empírica do número de visualizações de cada conteúdo.

Após a estimativa dos parâmetros, testamos a hipótese de que os dados seguem uma distribuição de lei de potência para valores maiores do que $x_{min}$ (cauda da distribuição). Para isso, adotamos o teste de *goodness-of-fit* definido por Clauset *et al.*, que permite verificar se a diferença entre os dados e o modelo pode ser ou não atribuída a uma flutuação estatística. O teste consiste em:

1. estimar os parâmetros $\hat{\alpha}$ e $\hat{x}_{min}$ a partir dos dados e calcular a estatística KS;
2. gerar um grande número de conjuntos de dados sintéticos a partir do modelo com parâmetros $\hat{\alpha}$ e $\hat{x}_{min}$;
3. para cada conjunto de dados sintético, encontrar o modelo de lei de potência mais adequado e calcular a estatística KS;
4. calcular o $p$-valor como a fração de vezes em que a estatística KS obtida com os conjuntos de dados sintéticos foi maior do que a do conjunto de dados original.

Se o $p$-valor for menor do que o nível de significância pré-estabelecido para o teste, devemos rejeitar a hipótese de que a cauda da distribuição empírica segue uma lei de potência. Caso contrário, podemos assumir que o modelo de lei de potência é plausível. Em nosso teste de *goodness-of-fit*, adotamos um nível de significância de 10%.

Usando a biblioteca *powerlaw*, obtivemos um $p$-valor igual a $0,1077$. Não podemos, portanto, rejeitar a hipótese de que os dados seguem uma lei de potência a partir de $x_{min}$. Mesmo assim, não podemos afirmar que o número de acessos aos conteúdos do sistema de VoD segue uma distribuição Zeta. O terceiro passo do método de Clauset *et al.* consiste em fazer a comparação com outros modelos. Essa comparação pode ser feita com o teste de razão de verossimilhança (*likelihood ratio test*). Basicamente, dizemos que a distribuição que melhor descreve os dados é aquela de maior verossimilhança, ou seja, com maior probabilidade de ter gerado as amostras. Para julgar se a diferença entre os valores de verossimilhança é estatisticamente relevante, Clauset *et al.* sugerem o método proposto por Vuong [46]. Esse método permite o cálculo de um $p$-valor que indica se devemos aceitar ou rejeitar a hipótese nula de que a diferença se deve puramente a uma flutuação estatística. Se o $p$-valor for menor do que o nível de significância pré-estabelecido de 10%, temos uma indicação confiável de que o modelo de maior verossimilhança é o mais adequado dentre os dois.

Comparamos então a distribuição Zeta com outras três candidatas a fim de identificar qual melhor descreve a cauda de nossa distribuição empírica: exponencial, log-normal e Zeta com decaimento exponencial. A distribuição Zeta com decaimento exponencial possui um termo $e^{-\mu x}$ que, para $x > \frac{1}{\mu}$, domina o comportamento de lei de potência e implica em um decaimento exponencial. Apresentamos o resultado da comparação na tabela 5.3. A primeira coluna indica a distribuição que comparamos com a Zeta. Quando comparamos a Zeta com uma exponencial, a primeira resulta em maior verossimilhança e $p$-valor baixo, indicando ser mais adequada. Já a log-normal apresenta maior verossimilhança do que a Zeta, mas o alto $p$-valor $(0,3704)$ mostra que o resultado é inconclusivo. Por fim, ao compararmos a Zeta com sua versão com decaimento exponencial, obtemos um $p$-valor baixo e maior verossimilhança para a Zeta com decaimento exponencial.

Tabela 5.3: Resultado da comparação do *fitting* da distribuição Zeta com outras distribuições.

| Distribuição | Parâmetros | Maior verossimilhança | $p$-valor |
|---|---|---|---|
| Exponencial | $\hat{\lambda} = 6,03 \cdot 10^{-4}$ | Zeta | **0,0058** |
| Log-normal | $\hat{\mu} = 2,17, \hat{\sigma} = 2,09$ | Log-normal | 0,3704 (inconclusivo) |
| Zeta + Exp. | $\hat{\alpha} = 2,14, \hat{\mu} = 4,69 \cdot 10^{-5}$ | Zeta + Exp. | **0,0741** |

Mostramos na figura 5.9 a cauda da CCDF empírica ($x \geq 931$) juntamente com os modelos Zeta, exponencial, log-normal e Zeta com decaimento exponencial. No gráfico, vemos que as curvas log-normal e Zeta se aproximam mais dos dados. Entretanto, notamos que é difícil modelar o número de visualizações dos conteúdos mais populares. Os 5 conteúdos com mais entradas nos *logs* são desenhos animados cujos episódios são identificados por um mesmo título e, por isso, agregam um grande

número de acessos. Na figura 5.10, apresentamos um gráfico de probabilidade que compara a cauda da distribuição empírica com a Zeta com decaimento exponencial. Esse gráfico confirma a dificuldade para modelar o número de acessos aos títulos mais populares que agregam visualizações de diversos episódios diferentes. Ainda assim, grande parte dos pontos encontra-se próxima à reta, fortalecendo a hipótese de que uma distribuição Zeta com decaimento exponencial de parâmetros $\hat{a} = 2,14$ e $\hat{\mu} = 4,69 \cdot 10^{-5}$ é um bom modelo para o número de acessos aos conteúdos mais populares do sistema real de VoD.

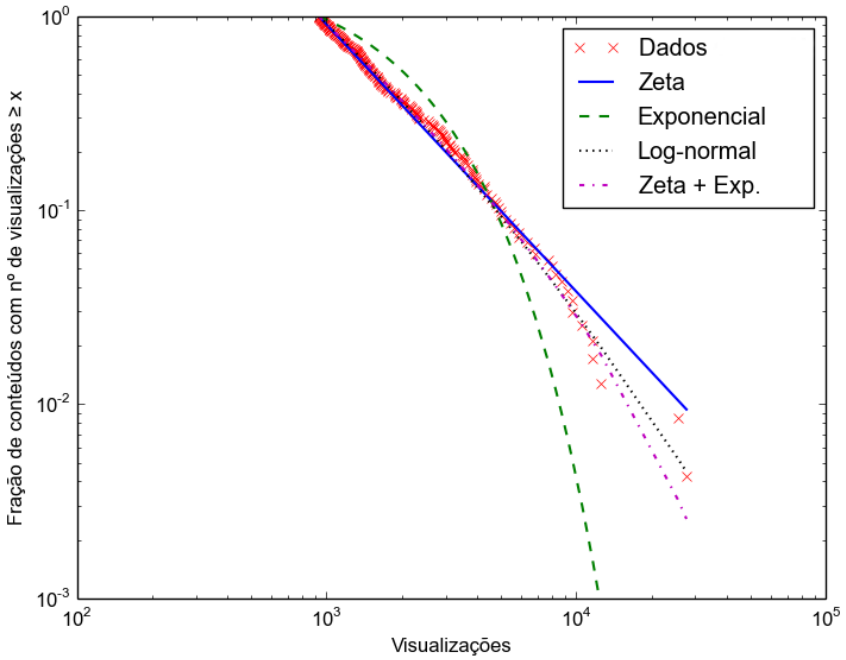



Figura 5.9: Gráfico mostrando cauda da CCDF empírica do número de acessos a cada conteúdo e os diversos modelos testados.

Com o decaimento na cauda da distribuição do número de acessos, temos indícios de que o número de visualizações de um conteúdo possui um limite superior. Esse mesmo comportamento foi observado, por exemplo, no *Youtube* [15]. O *Youtube* oferece conteúdo gerado pelos usuários (UGC - *User Generated Content*), diferindo do sistema aqui analisado. Todavia, a distribuição do número de visualizações dos vídeos do sistema também apresenta uma cauda bem modelada por uma Zeta com decaimento exponencial. No caso do *Youtube*, esse comportamento pode ser explicado pelo modelo *fetch-at-most-once* proposto por Gummadi *et al.* [47]. O modelo se baseia na premissa de que um usuário não assiste a um mesmo conteúdo mais de uma vez. Em nossos *logs*, os acessos a diferentes episódios de uma série ou programa de TV podem estar agregados. Vemos, por exemplo, um registro de 751 acessos de um dispositivo a um mesmo desenho animado ao longo dos 3 meses de registros.

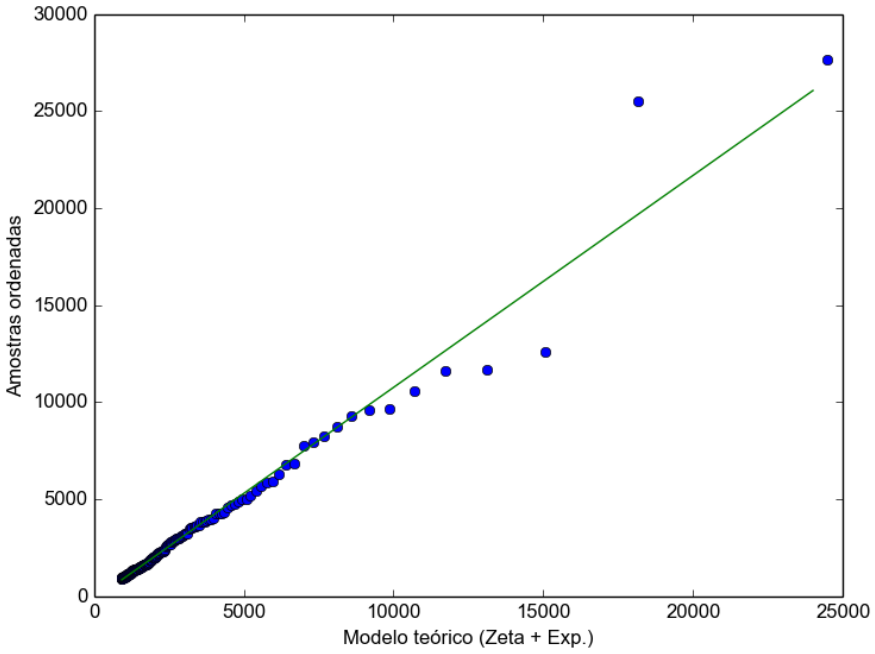



Figura 5.10: Gráfico de probabilidade comparando a distribuição do número de acessos a cada conteúdo com a distribuição Zeta com decaimento exponencial.

Nesse caso, todos os episódios foram identificados por um mesmo título. Ainda assim, se assumimos que os usuários assistem poucas vezes a cada episódio (em geral uma), o número de acessos de um usuário a determinado conteúdo passa a ser limitado superiormente. Desse modo, acreditamos que o modelo *fetch-at-most-once* também poderia ser aplicado ao sistema de VoD que analisamos.

### Popularidade relativa

Após analisar a distribuição empírica do número de visualizações de um conteúdo (representação de Pareto), desejamos verificar se uma distribuição Zipf é aplicável à popularidade relativa dos conteúdos. Definimos a popularidade relativa de um conteúdo como a razão entre o número de visualizações que ele teve e o número total de visualizações no período. Para essa análise, utilizamos o gráfico *rank-frequency* proposto por Zipf [41], no qual uma distribuição de lei de potência também se apresenta como uma reta quando os dois eixos estão em escala logarítmica. Nesse gráfico (ver figura 5.11), ordenamos os vídeos em ordem decrescente de visualizações e plotamos a fração de visualizações que o $r$-ésimo vídeo recebeu.

Diferentemente da análise anterior em que usamos como variável aleatória o número de visualizações de um conteúdo (total de 5.267 amostras), agora adotamos como v.a. o vídeo assistido em um registro dos *logs* (total de 1.004.829 amostras). Os vídeos são identificados por sua posição (*rank*), de modo que o vídeo mais assistido

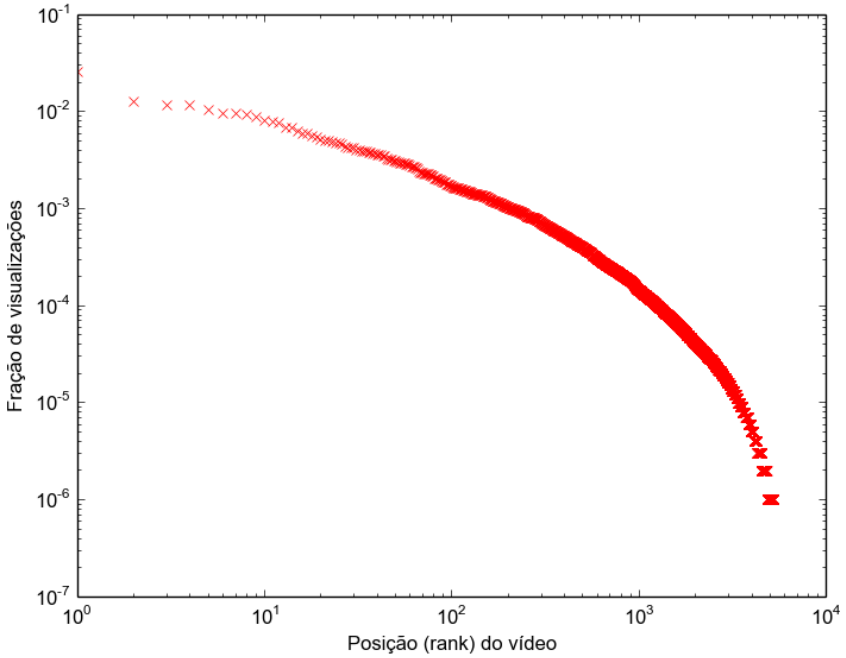



Figura 5.11: Gráfico de popularidade dos conteúdos.

possui identificador 1 e o menos assistido possui identificador 5.267.

Começamos a análise da distribuição de popularidade pela estimativa dos parâmetros das distribuições candidatas. A primeira alternativa a ser testada é uma distribuição Zipf, definida como

$$P[X_z = k] = \frac{k^{-\alpha}}{H_{m,\alpha}},$$

onde

$$H_{m,\alpha} = \sum_{j=1}^{m} j^{-\alpha}$$

e $m = 5.267$ é o número de conteúdos disponíveis.

No caso de algumas distribuições como a exponencial, é possível encontrar um estimador para seus parâmetros usando o método dos momentos. Para outras, como é o caso da distribuição de Pareto, podemos encontrar analiticamente os estimadores de máxima verossimilhança (MLE - *Maximum Likelihood Estimators*). Para uma distribuição Zipf, não é fácil encontrar uma fórmula fechada para um estimador do parâmetro $\alpha$. Optamos portanto por maximizar numericamente o logaritmo da função de verossimilhança. Definimos a função de verossimilhança como

$$L(x_1, ..., x_n \mid \alpha) = \prod_{i=1}^{n} P[X_z = x_i] = \prod_{i=1}^{n} \frac{x_i^{-\alpha}}{H_{m,\alpha}},$$

onde $x_1, ..., x_n$ são nossas amostras e $m = 5.267$ é o número de conteúdos disponíveis.

O seu logaritmo é dado por

$$l(x_1, ..., x_n \mid \alpha) = \log L(x_1, ..., x_n \mid \alpha)$$
$$= \sum_{i=1}^{n} \log \left( \frac{x_i^{-\alpha}}{H_{m,\alpha}} \right) = \sum_{i=1}^{n} \left( \log x_i^{-\alpha} - \log H_{m,\alpha} \right)$$
$$= -n \cdot \log H_{m,\alpha} - \alpha \cdot \sum_{i=1}^{n} \log x_i.$$

Para encontrar o parâmetro $\alpha$ que maximiza a função, utilizamos o método Nelder-Mead de otimização não-linear [48], também conhecido como método *Downhill Simplex*. Estimamos de maneira análoga os parâmetros $\alpha$ e $\mu$ da distribuição Zipf com decaimento exponencial definida como

$$P[X_{ze} = k] = \frac{k^{-\alpha} \cdot e^{-\mu \cdot k}}{H'_{m,\alpha,\mu}},$$

onde

$$H'_{m,\alpha,\mu} = \sum_{j=1}^{m} j^{-\alpha} \cdot e^{-\mu \cdot j}.$$

Sua função de verossimilhança é dada por

$$L(x_1, ..., x_n \mid \alpha, \mu) = \prod_{i=1}^{n} P[X_{ze} = x_i] = \prod_{i=1}^{n} \frac{x_i^{-\alpha} \cdot e^{-\mu \cdot x_i}}{H'_{m,\alpha,\mu}}$$

e seu logaritmo é dado por

$$l(x_1, ..., x_n \mid \alpha, \mu) = \log L(x_1, ..., x_n \mid \alpha, \mu) = \sum_{i=1}^{n} \log \left( \frac{x_i^{-\alpha} \cdot e^{-\mu \cdot x_i}}{H'_{m,\alpha,\mu}} \right)$$

$$= \sum_{i=1}^{n} \left( \log x_i^{-\alpha} + \log e^{-\mu \cdot x_i} - \log H'_{m,\alpha,\mu} \right)$$
$$= -n \cdot \log H'_{m,\alpha,\mu} - \alpha \cdot \sum_{i=1}^{n} \log x_i - \mu \sum_{i=1}^{n} x_i.$$

Apresentamos os resultados do *fitting* na tabela 5.4. Na figura 5.12, mostramos os três modelos juntamente com os dados empíricos. Os testes de razão de verossimilhança indicam, com $p$-valor próximo a zero, que o modelo Zipf com decaimento exponencial é o que mais se adequa aos dados. O gráfico de probabilidade, que compara esse modelo com a distribuição empírica, pode ser visto na figura 5.13. Os pontos aparecem bem próximos da reta, confirmando que uma distribuição Zipf com decaimento exponencial de parâmetros $\hat{\alpha} = 0,63$ e $\hat{\mu} = 9,69 \cdot 10^{-4}$ é um bom modelo.

O rápido decaimento na cauda, que indica uma queda brusca na popularidade dos

Tabela 5.4: Resultado do *fitting* da popularidade dos conteúdos.

| Distribuição | Parâmetros |
|---|---|
| Zipf | $\hat{\alpha} = 0,90$ |
| Zipf + Exp. | $\hat{\alpha} = 0,63, \hat{\mu} = 9,69 \cdot 10^{-4}$ |
| Log-normal | $\hat{\mu} = 4,68, \hat{\sigma} = 1,95$ |

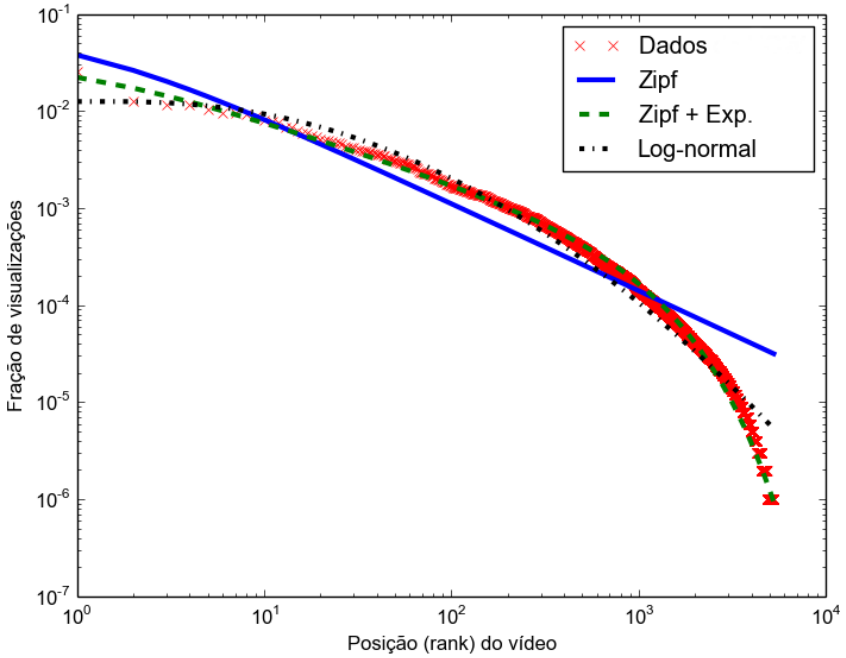



Figura 5.12: Gráfico com diferentes modelos para a distribuição de popularidade dos conteúdos.

vídeos menos populares, foi observado em outros sistemas de VoD como o Youtube [15, 49], o chinês PowerInfo [35], o Netflix [15] e o Dailymotion [16]. O parâmetro $\mu = 9,69 \cdot 10^{-4}$ indica no nosso caso um decaimento exponencial observado com mais intensidade a partir de $x \approx \frac{1}{\mu} \approx 1.032$ (1.032º conteúdo mais popular). Para valores menores de $x$, fica claro o domínio da lei de potência (uma reta no gráfico em escala log-log). Há duas explicações comuns para esse efeito. A primeira delas é o "envelhecimento" dos vídeos. À medida que o tempo passa, os vídeos mais antigos passam a receber cada vez menos acessos. Com os nossos dados, é difícil avaliar a presença desse fenômeno uma vez que não dispomos de um período muito longo de *logs* e não temos acesso à data de disponibilização (lançamento) de cada conteúdo. Uma segunda possível causa para o truncamento observado é a filtragem de informação [15]. Num sistema como o que analisamos, que não oferece mecanismos avançados de busca ou recomendação personalizada de conteúdo, esperamos que os usuários encontrem dificuldade para descobrir e eventualmente assistir aos vídeos pouco populares, dando preferência aos vídeos mais populares exibidos com maior destaque.

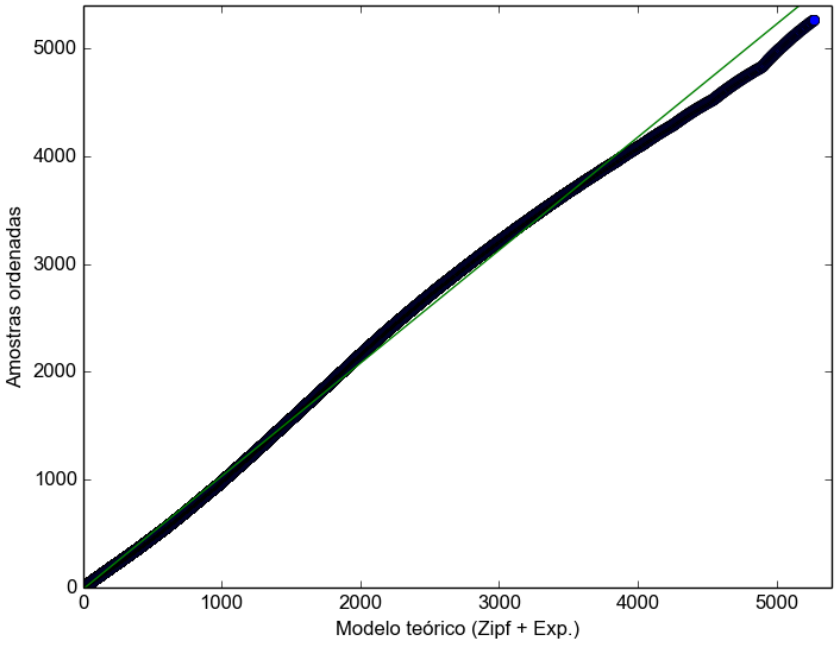



Figura 5.13: Gráfico de probabilidade comparando amostras com distribuição Zipf com decaimento exponencial.

## 5.2 Simulações

A fim de avaliar o desempenho de nossa proposta para transmissão de vídeo, executamos diversas simulações usando o modelo descrito na seção 4.3. Nesta seção, discutimos os cenários e parâmetros escolhidos para nossa análise e apresentamos os resultados obtidos. Nossos objetivos são:

1. validar o modelo para o limite de capacidade do sistema;
2. observar o impacto da política de troca de *swarms*;
3. avaliar o funcionamento com e sem a participação do servidor de conteúdo;
4. medir a influência de distribuições de popularidade com e sem decaimento exponencial;
5. avaliar o desempenho do sistema com carga equivalente à do sistema real de VoD analisado.

Em nossas simulações, assumimos uma população de 40000 *peers* com 10 Mbps de banda de *download* e 1 Mbps de *upload*. Cada nano cache armazena localmente 20 conteúdos e, dentre esses, serve 4. Devido a restrições da ferramenta Tangram-II,

limitamos o número de *swarms* a 250 [1]. Os conteúdos seguem uma distribuição de popularidade Zipf com decaimento exponencial com parâmetros $\hat{\alpha} = 0,63$ e $\hat{\mu} = 0,02$, retirados de nossa análise de um sistema real. O parâmetro $\hat{\mu}$, originalmente igual a $9,69 \cdot 10^{-4}$ com 5267 vídeos, foi escalado linearmente para um cenário com 250 conteúdos. Assumimos que todos os conteúdos possuem duração de 90 minutos e taxa de codificação (*bitrate*) de 5 Mbps (equivalente a um vídeo de alta definição). A menos que seja dito o contrário, assumimos uma taxa de chegada de 4800 *leechers* por hora ($\approx 1,34$ *leechers* por segundo), um intervalo para troca de *swarms* de 30 minutos e um servidor de conteúdo com banda de *upload* de 1 Gbps. Os valores adotados por padrão para os parâmetros do modelo podem ser vistos na tabela 5.5 (utilizamos a notação apresentada na seção 4.3).

Medimos o desempenho do sistema usando as seguintes métricas:

- $\bar{\boldsymbol{D}}$ – taxa média de *download*;
- $\bar{\boldsymbol{T}}_{\boldsymbol{D}}$ – tempo médio de *download*;
- $\boldsymbol{G}$ – fração de tempo com reprodução satisfatória (taxa média de *download* maior do que a taxa de codificação (5 Mbps));

As métricas $\bar{D}$ e $G$ são obtidas usando diretamente as recompensas definidas no modelo do Tangram-II. Já o tempo médio de *download* é calculado através da Lei de Little usando as recompensas de taxa média de chegada de *leechers* e número médio de *leechers* por *swarm*.

Todos os resultados apresentados nesta seção possuem intervalo de confiança de 95%.

### 5.2.1 Taxa de chegada

A fim de validar nosso modelo para o limite de capacidade e avaliar a escalabilidade do sistema, realizamos simulações mantendo a taxa de *upload* do servidor de conteúdo igual a 1 Gbps e variando o valor de $\lambda$. Nesse cenário, temos um limite para a taxa de chegada de aproximadamente 5467 *leechers* por hora. Os resultados são apresentados na tabela 5.6.

Vemos que o sistema apresenta um bom desempenho para valores abaixo do limite superior de escalabilidade. Mesmo para uma taxa de chegada de 5200 *leechers* por hora, observamos que em média 99% dos usuários experimentam tempo de *download* inferior a 5400 segundos. Esses resultados sugerem uma boa qualidade de experiência para os usuários, com pouco ou nenhum tempo de *buffering*.

[1] A ferramenta não é capaz de gerar modelos com mais de 250 *swarms* porque o número de eventos, o tamanho dos vetores de variáveis de estado e o número de recompensas crescem linearmente com o número de *swarms*.

Tabela 5.5: Parâmetros padrão adotados nas simulações

| Parâmetro | Valor |
|---|---|
| $N$ | 40000 |
| $C$ | 250 |
| $B$ | 20 ($\approx$ 66 GB de armazenamento) |
| $m$ | 4 |
| $\lambda$ | 4800 *leechers* / h |
| $\tau$ | 1800 s (30 min) |
| $D_{nc}$ | 10 Mbps |
| $U_{nc}$ | 1 Mbps |
| $U_s$ | 1 Gbps |
| $p_i$ | $\propto i^{-0,63} \cdot e^{-0,02 \cdot i}$ |
| $d_i$ | 5400 s (90 min) para todo $i$ |
| $b_i$ | 5 Mbps para todo $i$ |

Tabela 5.6: Avaliação do valor máximo de $\lambda$. Limite superior: 5467 *leechers* / h

| $\lambda$ (*leechers* / h) | $E[\bar{D}]$ (Mbps) | $\Pr(\bar{T}_D \leq 5400s)$ | $\Pr(G > 95\%)$ |
|---|---|---|---|
| 4800 | $9,90 \pm 0,03$ | $1,00 \pm 2,15 \cdot 10^{-4}$ | $0,97 \pm 5,04 \cdot 10^{-4}$ |
| 5200 | $8,79 \pm 0,25$ | $0,99 \pm 2,24 \cdot 10^{-3}$ | $0,93 \pm 9,57 \cdot 10^{-2}$ |
| 5400 | $5,51 \pm 0,25$ | $0,76 \pm 0,13$ | $0,02 \pm 3,58 \cdot 10^{-2}$ |
| 5600 | $3,98 \pm 0,09$ | $0,00 \pm 1,65 \cdot 10^{-4}$ | $0,00 \pm 0,00$ |

Para taxas de chegada muito próximas ou acima do limite, notamos que, como esperado, a qualidade se degrada, já que os nano caches e o servidor de conteúdo não possuem banda suficiente para atender à demanda. Isso se reflete na taxa média de *upload* menor, no alto tempo de *download* e na probabilidade próxima a zero de um *leecher* possuir uma taxa de *download* maior do que a taxa de codificação por mais de 95% do tempo.

### 5.2.2 Troca de *swarms*

Mostramos agora os resultados de nossa avaliação da política de troca de *swarms*. Comparamos nossas métricas de desempenho para diferentes valores de $\tau$ (tempo entre trocas). No cenário em que não há uma política especial para troca de *swarms* ($\tau = 0$), quando um *leecher* termina o *download* de um conteúdo ele passa a serví-lo imediatamente e deixa de servir um dos $m$ conteúdos que está servindo. Nesse caso, o *swarm* a ser deixado é escolhido aleatoriamente de acordo com a probabilidade do *peer* ser *seed* naquele *swarm* (ver equação 4.7).

Os resultados apresentados na tabela 5.7 sugerem uma piora de todas as métricas de desempenho quando não há troca de *swarms*. Observamos também uma pequena degradação da qualidade com o aumento do valor de $\tau$.

Na figura 5.14, vemos um gráfico com o tempo médio de *download* ($\bar{T}_D$) por

Tabela 5.7: Avaliação do intervalo entre trocas de *swarms*.

| Política | $E[\bar{D}](Mbps)$ | $\Pr(\bar{T}_D \leq 5400s)$ | $\Pr(G > 95\%)$ |
|---|---|---|---|
| Sem troca | $8,29 \pm 0,07$ | $0,94 \pm 6,27 \cdot 10^{-3}$ | $0,82 \pm 0,03$ |
| 30 min | $9,90 \pm 0,03$ | $1,00 \pm 2,15 \cdot 10^{-4}$ | $0,97 \pm 5,04 \cdot 10^{-4}$ |
| 120 min | $9,84 \pm 0,04$ | $1,00 \pm 3,31 \cdot 10^{-4}$ | $0,97 \pm 7,59 \cdot 10^{-4}$ |
| 480 min | $9,42 \pm 0,04$ | $0,99 \pm 6,07 \cdot 10^{-4}$ | $0,97 \pm 2,84 \cdot 10^{-3}$ |

*swarm*, com *swarms* em ordem crescente de popularidade. Notamos uma piora no desempenho quando não há troca de *swarms*. Sem a política de troca, os *swarms* de baixa e média popularidade apresentam tempo médio de *download* maior do que a duração do vídeo. Para alguns *swarms*, esse tempo chega a ser três vezes maior do que a duração do filme. Entretanto, para os 40 conteúdos mais populares, não vemos uma diferença significativa. Vemos também que o valor de $\bar{T}_D$ cresce juntamente com $\tau$ nos *swarms* de menor popularidade.

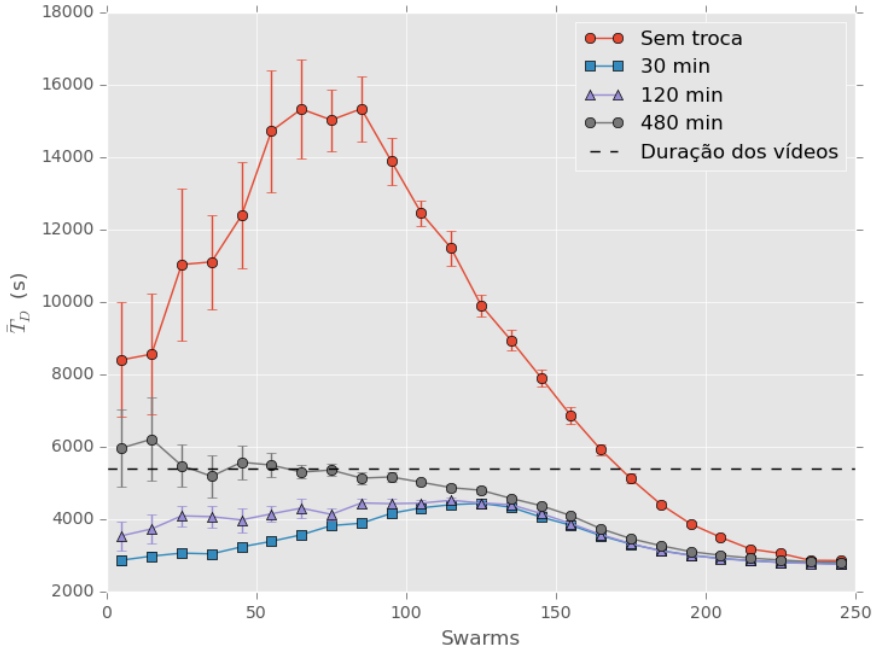



Figura 5.14: Comparação do tempo médio de *download* por *swarm* para diferentes valores de $\tau$. *Swarms* em ordem crescente de popularidade.

Concluímos que nossa política de troca de *swarms* causa grande impacto na qualidade percebida pelos usuários. E acreditamos que o valor do parâmetro $\tau$ pode ser ainda mais relevante em cenários com variação súbita de popularidade. Se o sistema entra em desequilíbrio, por exemplo quando um filme entra em promoção e a demanda por ele aumenta rapidamente, uma taxa maior de troca de *swarms* leva a uma migração mais rápida de *seeds* para os *swarms* mais necessitados.

### 5.2.3 Servidor de conteúdo

Nos voltamos agora para a análise do impacto da participação do servidor de conteúdo na transmissão dos arquivos de vídeo. Para isso, executamos simulações com $U_s$ igual a 0 Gbps. Na comparação com os resultados para $U_s$ igual a 1 Gbps, vemos uma diferença pouco significativa na taxa média de *download* dos *leechers*: $9,87 \pm 0,03$ Mbps sem servidor e $9,90 \pm 0,03$ Mbps com servidor.

A comparação do tempo médio de *download* por *swarm* indica que o impacto do servidor de conteúdo nesse cenário é significativo para os *swarms* menos populares. Como pode ser visto na figura 5.15, o cenário sem servidor apresentou valores de $\bar{T}_D$ maiores para os conteúdos menos populares. Além disso, o tamanho do intervalo de confiança indica uma maior variância amostral nos resultados das simulações sem servidor.

Os vídeos para os quais observamos um tempo médio de *download* maior representam cerca de 2% das visualizações do sistema. Ainda assim, o tempo médio de *download* manteve-se menor do que a duração dos vídeos em todos os *swarms*.

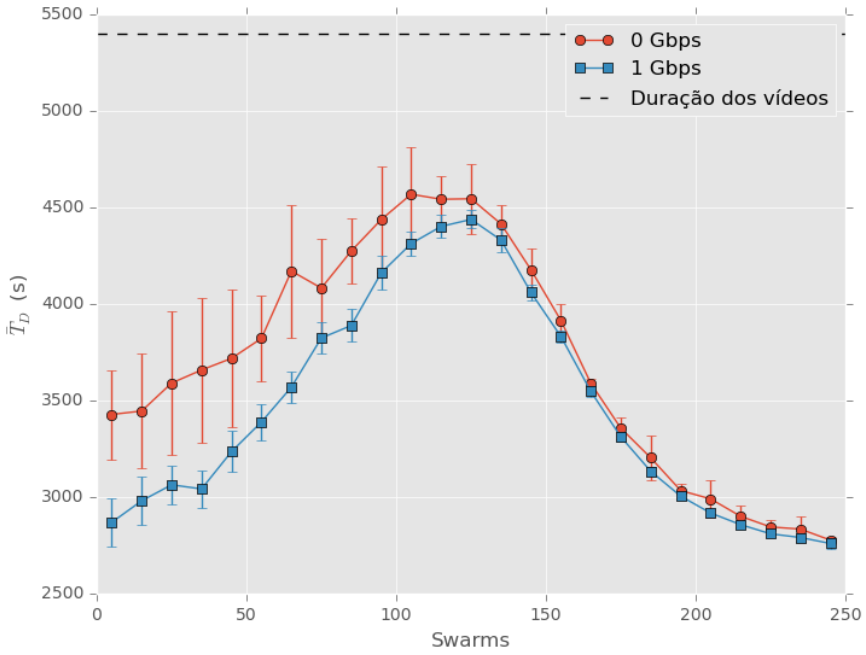



Figura 5.15: Comparação do tempo médio de *download* por *swarm* com e sem a participação do servidor de conteúdo. *Swarms* em ordem crescente de popularidade.

### 5.2.4 Distribuição de popularidade

Como mencionado anteriormente, assume-se com frequência que a distribuição de popularidade dos conteúdos em sistemas multimídia segue uma lei de potência. To-

Tabela 5.8: Avaliação da distribuição de popularidade dos conteúdos.

| Política | $E[\bar{D}](Mbps)$ | $\Pr(\bar{T}_D \leq 5400s)$ | $\Pr(G > 95\%)$ |
|---|---|---|---|
| Zipf | $9,83 \pm 0,04$ | $1,00 \pm 0$ | $1,00 \pm 0$ |
| Zipf + Exp. | $9,90 \pm 0,03$ | $1,00 \pm 2,15 \cdot 10^{-4}$ | $0,97 \pm 5,04 \cdot 10^{-4}$ |

davia, alguns estudos empíricos, como o que apresentamos na seção 5.1, identificaram a existência de distribuições de popularidade truncadas em sistemas de VoD. Para verificar o impacto do truncamento da cauda da distribuição no desempenho do sistema, comparamos a distribuição Zipf com decaimento exponencial com uma distribuição Zipf pura com parâmetro $\alpha = 0,8$ [2].

Apresentamos na tabela 5.8 a média por *peer* de nossas métricas de desempenho. Os resultados para as duas distribuições de popularidade são bastante semelhantes e não apresentam diferenças estatisticamente relevantes.

Na figura 5.16, vemos o tempo médio de *download* por *swarm* obtido nos dois cenários. Notamos uma grande diferença no formato das curvas. O sistema com uma distribuição Zipf pura apresenta pior desempenho para os *swarms* menos populares. Já no caso da distribuição com decaimento exponencial, os usuários tendem a experimentar um tempo de *download* maior ao assistirem aos conteúdos de popularidade média. Lembramos que, com uma distribuição Zipf com decaimento exponencial, os 100 vídeos menos populares somam cerca de 1% dos acessos ao sistema. Com a distribuição Zipf, os 100 conteúdos de menor popularidade recebem aproximadamente 14% das visualizações.

Ainda que o desempenho médio por *peer* seja bastante semelhante nos dois casos, concluímos que a aproximação da distribuição real de popularidade por uma distribuição Zipf pura pode não ser boa, já que o desempenho em cada *swarm* é consideravelmente diferente do observado quando uma distribuição com decaimento exponencial é utilizada.

### 5.2.5 Carga de um sistema real de VoD

Por fim, avaliamos o desempenho do sistema utilizando a carga extraída do sistema real de VoD. Rodamos simulações com uma taxa de chegada $\lambda$ igual a 682 visualizações por hora, igual à média do período de pico (entre 15h e 2h).

Ao compararmos esse valor de $\lambda$ com o limite superior dado por nosso modelo, vemos que um sistema de nano caches com essa carga operaria consideravelmente abaixo do limite de sua capacidade. Sem a participação do servidor de conteúdo, a taxa limite é de aproximadamente 5107 visualizações por hora para $N = 38299$ (número de clientes nos *logs*). Nossas simulações confirmam esse resultado: medimos

[2] Valor tipicamente usado na literatura [9, 12, 50].

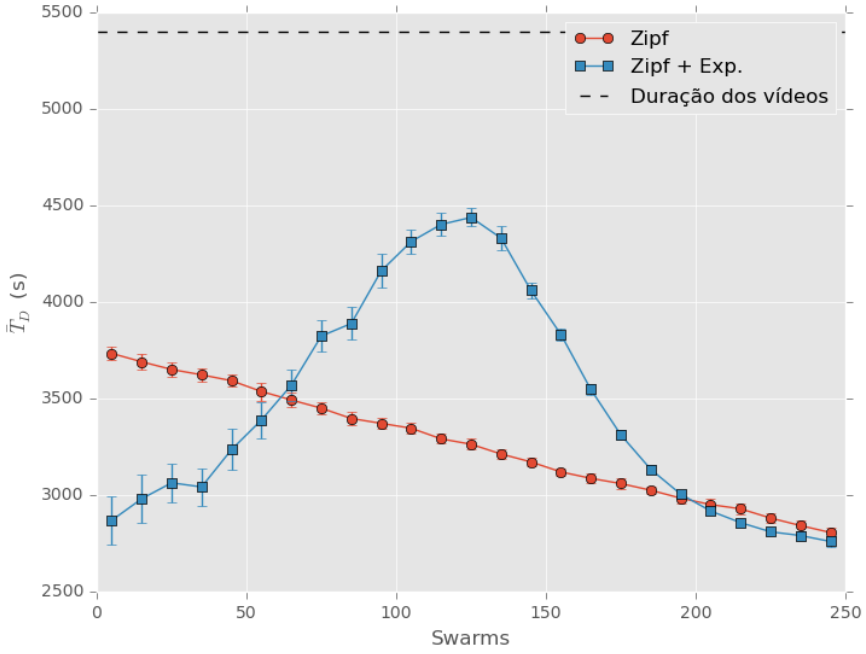



Figura 5.16: Comparação do tempo médio de *download* por *swarm* para as duas distribuições de popularidade. *Swarms* em ordem crescente de popularidade.

uma taxa média de *download* de $10 \pm 1,55 \cdot 10^{-4}$ Mbps e uma fração média de $99\% \pm 0,2$ p.p. dos usuários com tempo de *download* menor ou igual à duração dos vídeos.

## 5.3 Experimentos

Usando uma pequena rede de nano caches, conduzimos experimentos a fim de comparar o desempenho das três políticas de escolha de conteúdos servidos. Selecionamos para os experimentos 9 roteadores com mesma banda de *upload* disponível, de aproximadamente 1,2 Mbps. Inicialmente, selecionamos 25 *torrents* públicos populares, com grande número de *leechers*, e fizemos seu *download* em cada um dos nano caches. Depois, executamos diariamente uma aplicação que, durante um período de tempo determinado, serve $m = 5$ conteúdos dentre os 25 armazenados e, em intervalos de tempo $\tau = 30$ min, atualiza o conjunto de conteúdos sendo servidos de acordo com uma das três políticas definidas.

A aplicação foi executada diariamente durante um período de 27 dias, das 2h às 7h. Alocamos 3 nano caches para cada uma das políticas avaliadas. Durante a execução do experimento, coletamos a cada segundo as seguintes métricas:

1. lista de conteúdos sendo servidos;
2. taxa de *upload* total;

3. taxa de *upload* em cada *swarm*;

4. carga de cada *swarm*;

5. fração de carga servida pelo *peer*;

6. utilidade do *peer*.

Definimos as métricas 5 e 6 como:

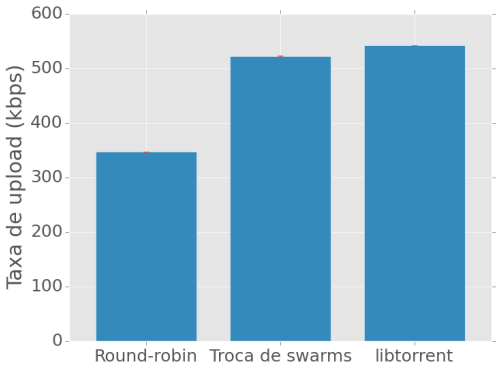
$$\text{Fração de carga servida} \triangleq \frac{\sum_{i \in \mathcal{I}} \ell_i}{\sum_{j=1}^{C} \ell_j} \tag{5.1}$$

$$\text{Utilidade} \triangleq \frac{\sum_{i \in \mathcal{I}} u_i \cdot \ell_i}{\sum_{j=1}^{C} \ell_j}, \tag{5.2}$$

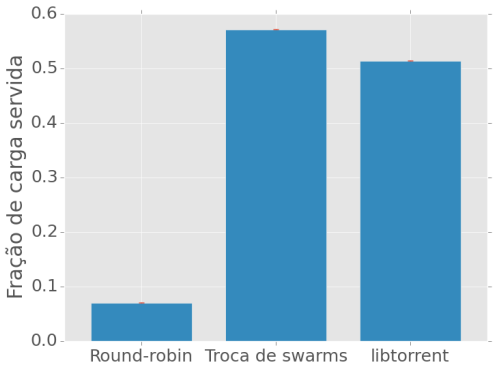
onde $\ell_i$ é a carga do *swarm* $i$ (ver equação 3.1), $u_i$ é a taxa de *upload* para o *swarm* $i$, $C$ é o número de *swarms* e $\mathcal{I}$ é o conjunto formado pelos índices dos conteúdos sendo servidos no momento. Portanto, o numerador da fração na equação 5.1 é a soma da carga dos *swarms* que o nano cache serve e, na equação 5.2, é a mesma soma ponderada pela taxa de *upload* nesses *swarms*. As duas métricas nos permitem avaliar o quão bem os recursos de um nano cache estão sendo utilizados. Queremos que, preferencialmente, os *swarms* de maior carga sejam priorizados.

Apresentamos os resultados de nossos experimentos na figura 5.17. Os gráficos contém a média das amostras coletadas a cada segundo juntamente com o intervalo de confiança de 95%. Para cada conjunto de amostras obtidas em um dia, descartamos aquelas relativas ao período transiente (primeiros 30 minutos do experimento). Notamos que, para todas as métricas, a política *Round-robin* apresentou um desempenho consideravelmente pior do que o das demais. Vemos também que a política adotada pela *libtorrent* levou, em média, a uma maior taxa de *upload*, ainda que a diferença seja pequena em relação à nossa proposta. Por outro lado, os nano caches que adotaram a política de troca de *swarms* baseada em carga obtiveram maior fração de carga servida e utilidade.

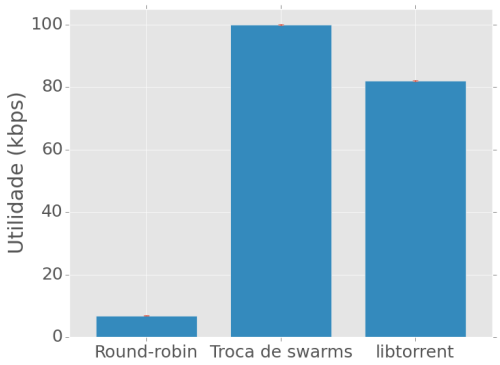
Concluímos que uma política puramente baseada na justiça como a política *Round-robin*, que dá igual importância a todos os conteúdos, leva à subutilização dos recursos do *peer* que faz *upload*, além de não favorecer a distribuição dos conteúdos com maior demanda. Já a nossa estratégia de troca de *swarms* permitiu uma taxa de *upload* média alta – bastante próxima da obtida com a política da biblioteca *libtorrent* – juntamente com uma maior utilidade e fração de carga servida. Com essa política simples que não prioriza a justiça entre *swarms*, podemos reduzir o desbalanceamento entre o número de clientes que assistem a determinado conteúdo (*leechers*) e o número de nano caches que o servem (*seeds*) e, consequentemente, melhorar a qualidade do serviço ofertado aos usuários do sistema.




(a) Taxa de *upload* média




(b) Média da fração de carga servida




(c) Utilidade média

Figura 5.17: Resultados obtidos após 27 dias de experimentos com 9 nano caches.

# Capítulo 6

# Conclusão

Neste trabalho, apresentamos uma plataforma P2P para transmissão de vídeo com dispositivos embarcados projetada e avaliada com base em circunstâncias reais. Diferentemente de outros trabalhos [4–9], não nos atemos aos problemas de alocação de conteúdos na rede e escolha dos vizinhos para os quais um *seed* faz *upload*, focando nossa análise no problema de escolha dos *swarms* que um *peer* serve.

Usando uma métrica de carga de um *swarm*, propomos uma política para definição dos conteúdos servidos. Nossa política de troca de *swarms* permite o balanceamento da carga do sistema de forma distribuída sem a adição de *overheads* e possui como principal vantagem o fato de ser mais simples do que as adotadas em outros sistemas [12, 20].

Nosso modelo probabilístico para a política de troca de *swarms* permite uma drástica redução no espaço de estados do sistema que, a princípio, deveria crescer exponencialmente com o número de nano caches. Desse modo, torna-se viável a análise do desempenho do sistema usando parâmetros próximos aos reais. Validamos o modelo através do método Monte Carlo e obtivemos um erro médio inferior a 3%. Em trabalhos futuros, pretendemos utilizar esse modelo para a dedução de novos resultados analíticos.

Através de simulações, avaliamos a qualidade do serviço ofertado aos usuários em diversos cenários e pudemos constatar que nosso algoritmo para troca de *swarms* reduziu significativamente o tempo médio de *download* para os conteúdos de média e baixa popularidade. Além disso, mostramos que nosso algoritmo permitiu, para todos os *swarms*, um tempo médio de *download* inferior à duração do vídeo. Temos a intenção de analisar no futuro o impacto de diferentes políticas para o servidor de conteúdo, como a prioridade aos *swarms* de maior carga. Ressaltamos que, embora tenhamos assumido nano caches com mesma banda de *upload* e *download* e vídeos de mesma duração e *bitrate*, nosso estudo pode ser facilmente estendido para cenários mais gerais.

Apresentamos também nesta dissertação um protótipo para distribuição de vídeo

em sistemas embarcados. Com ele, realizamos experimentos em uma pequena rede de nano caches para comparar nossa política com duas políticas implementadas por clientes BitTorrent. Identificamos que, além de permitir uma taxa média de *upload* alta, nossa política levou a uma melhor utilização dos recursos dos *peers*, beneficiando os *swarms* com escassez de recursos. Esperamos no futuro poder utilizá-lo em experimentos de maior escala, visando confirmar os resultados de nossas simulações e validar nossas premissas.

Em nossa análise de um sistema real de distribuição de vídeo sob demanda, identificamos que o processo de chegada de usuários nos horários de pico pode ser bem modelado por uma distribuição Poisson e que a distribuição de popularidade dos conteúdos não segue uma lei de potência, o que contradiz a hipótese assumida por diversos outros trabalhos [4, 5, 7–9, 11–14]. Em vez disso, confirmamos os resultados empíricos obtidos a partir de dados de outros sistemas [15, 16], observando uma distribuição de popularidade Zipf com decaimento exponencial.

Acreditamos ser este o primeiro trabalho a apresentar uma análise do desempenho de uma plataforma P2P de VoD com distribuição de popularidade Zipf com decaimento exponencial com parâmetros extraídos de um sistema real. Concluímos que ainda há muito a ser investigado em cenários como esse. Em especial, observamos que, devido à brusca queda na cauda da distribuição, novas medidas podem ser necessárias para atender adequadamente aos usuários interessados em conteúdos de nicho.

# Referências Bibliográficas


[1] SUH, K., DIOT, C., KUROSE, J., et al. "Push-to-peer video-on-demand system: Design and evaluation", *Selected Areas in Communications, IEEE Journal on*, v. 25, n. 9, pp. 1706–1716, 2007.

[2] LAOUTARIS, N., RODRIGUEZ, P., MASSOULIE, L. "ECHOS: edge capacity hosting overlays of nano data centers", *ACM SIGCOMM Computer Communication Review*, v. 38, n. 1, pp. 51–54, 2008.

[3] CHA, M., RODRIGUEZ, P., MOON, S. B., et al. "On next-generation telco-managed P2P TV architectures." In: *IPTPS*, p. 5, 2008.

[4] TEWARI, S., KLEINROCK, L. "Proportional Replication in Peer-to-Peer Networks." In: *INFOCOM*. Citeseer, 2006.

[5] NAFAA, A., MURPHY, S., MURPHY, L. "Analysis of a large-scale VOD architecture for broadband operators: a P2P-based solution", *Communications Magazine, IEEE*, v. 46, n. 12, pp. 47–55, 2008.

[6] WU, J., LI, B. "Keep cache replacement simple in peer-assisted vod systems". In: *INFOCOM 2009, IEEE*, pp. 2591–2595. IEEE, 2009.

[7] WU, W., LUI, J. "Exploring the optimal replication strategy in P2P-VoD systems: Characterization and evaluation", *Parallel and Distributed Systems, IEEE Transactions on*, v. 23, n. 8, pp. 1492–1503, 2012.

[8] ZHOU, Y., FU, T. Z., CHIU, D. M. "A unifying model and analysis of P2P VoD replication and scheduling". In: *INFOCOM, 2012 Proceedings IEEE*, pp. 1530–1538. IEEE, 2012.

[9] TAN, B., MASSOULIÉ, L. "Optimal content placement for peer-to-peer video-on-demand systems", *IEEE/ACM Transactions on Networking (TON)*, v. 21, n. 2, pp. 566–579, 2013.

[10] YANG, Y., CHOW, A., GOLUBCHIK, L., et al. "Improving QoS in bittorrent-like VoD systems". In: *INFOCOM, 2010 Proceedings IEEE*, pp. 1–9. IEEE, 2010.

[11] MUÑOZ-GEA, J. P., TRAVERSO, S., LEONARDI, E. "Modeling and evaluation of multisource streaming strategies in P2P VoD systems", *IEEE Transactions on Consumer Electronics*, v. 58, n. 4, pp. 1202–1210, 2012.

[12] HE, Y., XIONG, Z., ZHANG, Y., et al. "Modeling and analysis of multi-channel P2P VoD systems", *Journal of Network and Computer Applications*, v. 35, n. 5, pp. 1568–1578, 2012.

[13] CIULLO, D., MARTINA, V., GARETTO, M., et al. "How much can large-scale Video-on-Demand benefit from users' cooperation?" In: *INFOCOM, 2013 Proceedings IEEE*, pp. 2724–2732. IEEE, 2013.

[14] CIULLO, D., MARTINA, V., GARETTO, M., et al. "Peer-assisted VoD Systems: an Efficient Modeling Framework", *Parallel and Distributed Systems, IEEE Transactions on*, v. 25, n. 7, pp. 1852–1863, 2014.

[15] CHA, M., KWAK, H., RODRIGUEZ, P., et al. "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system". In: *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pp. 1–14. ACM, 2007.

[16] MITRA, S., AGRAWAL, M., YADAV, A., et al. "Characterizing web-based video sharing workloads", *ACM Transactions on the Web (TWEB)*, v. 5, n. 2, pp. 8, 2011.

[17] HUANG, Y., FU, T. Z., CHIU, D.-M., et al. "Challenges, Design and Analysis of a Large-scale P2P-vod System". In: *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pp. 375–388, New York, NY, USA, 2008. ACM. ISBN: 978-1-60558-175-0. doi: 10.1145/1402958.1403001. Disponível em: <`http://doi.acm.org/10.1145/1402958.1403001`>.

[18] LIU, F., LI, B., LI, B., et al. "Peer-assisted on-demand streaming: Characterizing demands and optimizing supplies", *Computers, IEEE Transactions on*, v. 62, n. 2, pp. 351–361, 2013.

[19] CHANG, L., PAN, J., XING, M. "Effective utilization of user resources in PA-VoD systems with channel heterogeneity", *Selected Areas in Communications, IEEE Journal on*, v. 31, n. 9, pp. 227–236, 2013.

[20] WANG, Z., WU, C., SUN, L., et al. "Strategies of collaboration in multi-channel P2P VoD streaming". In: *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pp. 1–5. IEEE, 2010.

[21] CHEN, Z., YAN, J. "Optimizing resource scheduling in BitTorrent file distribution network". In: *Management and Service Science, 2009. MASS'09. International Conference on*, pp. 1–4. IEEE, 2009.

[22] MENG, X., TSANG, P.-S., LUI, K.-S. "Analysis of distribution time of multiple files in a P2P network", *Computer Networks*, v. 57, n. 15, pp. 2900–2915, 2013.

[23] WHITEAKER, J., SCHNEIDER, F., TEIXEIRA, R., et al. "Expanding home services with advanced gateways", *ACM SIGCOMM Computer Communication Review*, v. 42, n. 5, pp. 37–43, 2012.

[24] SHALUNOV, S., HAZEL, G., IYENGAR, J., et al. "Low Extra Delay Background Transport (LEDBAT)". RFC 6817 (Experimental), dez. 2012. Disponível em: <`http://www.ietf.org/rfc/rfc6817.txt`>.

[25] GROVER, S., PARK, M. S., SUNDARESAN, S., et al. "Peeking behind the NAT: an empirical study of home networks." In: *Internet Measurement Conference*, pp. 377–390, 2013.

[26] COHEN, B. "The BitTorrent Protocol Specification (BEP 3)". `http://www.bittorrent.org/beps/bep_0003.html`, jan. 2008. [Online; acessado em 09/02/2015].

[27] MENDONÇA, G., LEÃO, R. "BTStream–Um ambiente para desenvolvimento e teste de aplicações de streaming P2P". In: *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, 2012.

[28] NORBERG, A. "uTorrent Transport Protocol (BEP 29)". `http://www.bittorrent.org/beps/bep_0029.html`, jun. 2009. [Online; acessado em 29/11/2014].

[29] DE SOUZA E SILVA, E., FIGUEIREDO, D. R., LEÃO, R. M. "The TANGRAM-II integrated modeling environment for computer systems and networks", *ACM SIGMETRICS Performance Evaluation Review*, v. 36, n. 4, pp. 64–69, 2009.

[30] WU, D., LIU, Y., ROSS, K. W. "Queuing network models for multi-channel P2P live streaming systems". In: *INFOCOM 2009, IEEE*, pp. 73–81. IEEE, 2009.

[31] JONES, E., OLIPHANT, T., PETERSON, P., et al. "SciPy: Open source scientific tools for Python". 2001–. Disponível em: <`http://www.scipy.org/`>. [Online; acessado em 09/01/2015].

[32] RONACHER, A., OTHERS. "Jinja: Template engine for Python". 2006– . Disponível em: <http://www.scipy.org/>. [Online; acessado em 11/01/2015].

[33] GUO, L., TAN, E., CHEN, S., et al. "The stretched exponential distribution of internet media access patterns". In: *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pp. 283–294. ACM, 2008.

[34] HUANG, C., LI, J., ROSS, K. W. "Can internet video-on-demand be profitable?" *ACM SIGCOMM Computer Communication Review*, v. 37, n. 4, pp. 133–144, 2007.

[35] YU, H., ZHENG, D., ZHAO, B. Y., et al. "Understanding User Behavior in Large-scale Video-on-demand Systems". In: *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, pp. 333–344, New York, NY, USA, 2006. ACM. ISBN: 1-59593-322-0. doi: 10.1145/1217935.1217968. Disponível em: <http://doi.acm.org/10.1145/1217935.1217968>.

[36] *Cisco TV CDS 3.2 RTSP Software Configuration Guide*. Cisco Systems, Inc., nov 2012. Disponível em: <http://www.cisco.com/c/en/us/td/docs/video/cds/cda/tv/3_2/configuration/rtsp_guide/tv_cds_3_2_rtsp_config.pdf>.

[37] ZINK, M., SUH, K., GU, Y., et al. "Watch global, cache local: YouTube network traffic at a campus network: measurements and implications". In: *Electronic Imaging 2008*, pp. 681805–681805. International Society for Optics and Photonics, 2008.

[38] NEWMAN, M. E. J. "Power laws, Pareto distributions and Zipf's law", *Contemporary Physics*, v. 46, n. 1, pp. 323–351, nov. 2004. doi: 10.1016/j.cities.2012.03.001. Disponível em: <http://arxiv.org/abs/cond-mat/0412004>.

[39] ADAMIC, L. A. "Zipf, Power-laws, and Pareto - a ranking tutorial". out. 2014. Disponível em: <http://www.hpl.hp.com/research/idl/papers/ranking>.

[40] PARETO, V. *Cours d'économie politique*. Librairie Droz, 1964.

[41] ZIPF, G. K. "Human Behaviour and the Principle of Least Effort", *Cambridge, Mass*, 1949.

[42] MITRA, S. *Characterising online video sharing and its dynamics*. Tese de Mestrado, Indian Institute of Technology, Delhi, 2009. Disponível em: <http://www.sidmitra.com/research/mitra_mtechThesis.pdf>.

[43] SOMEONE. "Placeholder", *Journal of Elasticity*, v. 42, n. 2, pp. 145–163, fev. 1996.

[44] CLAUSET, A., SHALIZI, C. R., NEWMAN, M. E. "Power-law distributions in empirical data", *SIAM review*, v. 51, n. 4, pp. 661–703, 2009.

[45] ALSTOTT, J., BULLMORE, E., PLENZ, D. "powerlaw: a Python package for analysis of heavy-tailed distributions", *PloS one*, v. 9, n. 1, pp. e85777, 2014.

[46] VUONG, Q. H. "Likelihood ratio tests for model selection and non-nested hypotheses", *Econometrica: Journal of the Econometric Society*, pp. 307–333, 1989.

[47] GUMMADI, K. P., DUNN, R. J., SAROIU, S., et al. "Measurement, Modeling, and Analysis of a Peer-to-peer File-sharing Workload". In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pp. 314–329, New York, NY, USA, 2003. ACM. ISBN: 1-58113-757-5. doi: 10.1145/945445.945475. Disponível em: <http://doi.acm.org/10.1145/945445.945475>.

[48] NELDER, J. A., MEAD, R. "A simplex method for function minimization", *The computer journal*, v. 7, n. 4, pp. 308–313, 1965.

[49] CHENG, X., DALE, C., LIU, J. "Understanding the Characteristics of Internet Short Video Sharing: YouTube as a Case Study", *CoRR*, v. abs/0707.3670, 2007. Disponível em: <http://arxiv.org/abs/0707.3670>.

[50] FRICKER, C., ROBERT, P., ROBERTS, J., et al. "Impact of traffic mix on caching performance in a content-centric network". In: *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pp. 310–315. IEEE, 2012.