
# BITLIVERY: UM ALGORITMO PARA TRANSMISSÃO DE CONTEÚDO ENTRE MÚLTIPLOS SERVIDORES E MÚLTIPLOS CLIENTES

Alejandra Klachquin

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: Daniel Ratton Figueiredo

Rio de Janeiro
Setembro de 2013

BITLIVERY: UM ALGORITMO PARA TRANSMISSÃO DE CONTEÚDO ENTRE MÚLTIPLOS SERVIDORES E MÚLTIPLOS CLIENTES

Alejandra Klachquin

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. Daniel Ratton Figueiredo, Ph.D.

Prof. Ricardo Guerra Marroquim, D.Sc.

Prof. Antonio Augusto de Aragao Rocha, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2013

Klachquin, Alejandra

BitLivery: Um algoritmo para transmissão de conteúdo entre múltiplos servidores e múltiplos clientes/Alejandra Klachquin. – Rio de Janeiro: UFRJ/COPPE, 2013.

XII, 59 p.: il.; 29, 7cm.

Orientador: Daniel Ratton Figueiredo

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2013.

Referências Bibliográficas: p. 57 – 59.

1. distribuição de conteúdo. 2. múltiplos servidores. 3. algortimo centralizado. 4. distribuição de conteúdo adaptativa. I. Figueiredo, Daniel Ratton. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

*Nam Mioho Rengue Kyo*

# Agradecimentos

Gostaria de agradecer a todos que me apoiaram e incentivaram, em especial ao meu orientador Professor Daniel R. Figueiredo pela paciência e motivação constante. Agradeço também às pessoas que me ajudaram com carinho durante todos estes anos: Carol, colegas de laboratório e trabalho, e amigos, em especial João Paixão e Leopoldo por estarem sempre presentes. Gostaria de agradecer especialmene também aos meus pais, Graciela e Carlos por todas as conversas e carinho, e ao Anderson, companheiro, amigo e namorado, por todo o apoio, lanchinhos e puxões de orelha quando necessários. Por fim, gostaria de agradecer à COPPE, CNPq e CAPES pelo apoio financeiro, e à Globo.com pelo apoio à conclusão deste trabalho.

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

# BITLIVERY: UM ALGORITMO PARA TRANSMISSÃO DE CONTEÚDO ENTRE MÚLTIPLOS SERVIDORES E MÚLTIPLOS CLIENTES

Alejandra Klachquin

Setembro/2013

Orientador: Daniel Ratton Figueiredo

Programa: Engenharia de Sistemas e Computação

Sistemas de distribuição de conteúdo de larga escala têm tornado-se cada vez mais importantes devido à crescente oferta e demanda de conteúdo digital na Internet. Uma característica das propostas recentes nesta área é a replicação de conteúdo em diversos pontos da rede tais como nas abordagens CDN (*Content Delivery Network*) e ICN (*Information Centric Network*). Neste trabalho propomos o BitLivery, um algoritmo adaptativo baseado na arquitetura cliente-servidor com múltiplos servidores com conteúdo replicado. Sua principal característica é o dinamismo das conexões entre clientes e servidores, que não são pré-estabelecidas e que podem ser encerradas e iniciadas em função da vazão sendo obtida e da capacidade de download do cliente. Desta forma, os clientes podem se adaptar rapidamente às mudanças nas condições da rede e dos servidores. Um simulador específico foi desenvolvido para avaliar o algoritmo proposto em diferentes cenários e medir seu desempenho. Os resultados obtidos com simulação indicam o potencial de adaptação e eficiência do algoritmo.

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

BITLIVERY: A MULTI-SERVER CONTENT DELIVERY ALGORITHM

Alejandra Klachquin

September/2013

Advisor: Daniel Ratton Figueiredo

Department: Systems Engineering and Computer Science

Large content distribution systems are getting bigger attention, matter of a lot of studies and implemantetions due to increasing demand and offer of digital content. A common feature of recent proposes is content replication, as in CDNs (Content Delivery Network) and ICN (Information Centric Network). In this work we propose BitLivery, an adaptative algorithm for multiple servers with replicated content based on client-server arquitecture. Its main feature is the dynamism of connections between clients and servers, which are not predetermined and can be started and ended as a function of flow being obtained and client's download capacity. Therefore, clients can quickly adapt to changing network conditions and servers. A particular simulator has been developed for evaluating the proposed algorithm at several cenarios and measure its performance. Results through simulation indicate its adaptative and perfomance potential.

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# Introdução

Vivemos hoje na chamada Era da Informação ou Era Digital. Com a rápida evolução e barateamento das tecnologias de informação e comunicação, houve a popularização dos computadores pessoais e smartphones. Acompanhando esse processo, o crescimento do acesso à Internet, em especial da banda larga, impulsionou uma tendência chamada de convergência digital, onde serviços tradicionais passaram a estar cada vez mais disponíveis na Internet. Como veremos a seguir, a revolução digital modificou o estilo de vida da sociedade.

## 1.1 Motivação

Hoje, textos, imagens, áudio e vídeo podem ser facilmente transformados em conteúdo digital e disponibilizados na Web através de seus inúmeros sistemas tais como Youtube, Facebook, Flickr e Soundcloud. Meios tradicionais de informação como rádio, televisão e jornal estão cada vez mais se adaptando para fornecer seus serviços inteiramente via Web e instituições de ensino já são capazes de ministrar cursos superiores quase exclusivamente através de video-aulas e tutorias online. Videolocadoras são cada vez menos utilizadas, sendo substituidas por serviços como Netflix[2], assim como DVD's, CD's e livros são cada vez menos adquiridos em seus formatos convencionais, em contra partida ao conteúdo puramente digital, disponíveis em lojas como iTunes e Amazon[3]. O problema de prover conteúdo digital pela Internet de forma eficiente é chamada distribuição de conteúdo.

Com o crescimento da Internet e da convergência digital, a distribuição de conteúdo tornou-se ainda mais importante, pois ela pode afetar diretamente a qualidade do serviço oferecido. Entretanto, a Internet é um sistema distribuído e heterogêneo. Distribuído, pois não existe uma entidade centralizadora que possua informações sobre as características de todos os usuários e suas respectivas demandas por serviços. Heterogêneo, pois os dispositivos utilizados hoje para acessar conteúdo digital possuem recursos muito distintos tais como capacidade de conexão, proces-

samento, tamanho de tela, etc, e diferentes tipos de serviços possuem diferentes restrições em relação à qualidade de entrega dos dados [4]. Por não existir um agente centralizador que possa planejar a distribuição de conteúdo de forma otimizada, o sistema deve ser capaz de se adequar ao ambiente, que é relativamente imprevisível, onde podem ocorrer variações significativas sobre a demanda de serviços, falhas de equipamentos de armazenamento de dados e rede e até mesmo ataques maliciosos a provedores de conteúdo.

## 1.2 O controle da distribuição de conteúdo

Os sistemas que se encarregam da entrega dos dados aos clientes são projetados e administrados pelos provedores de conteúdo, que podem criar suas próprias soluções ou utilizar sistemas de terceiros. O fato dos provedores não conhecerem as características da infraestrutura da rede de seus clientes torna a distribuição de conteúdo um problema difícil, já que os clientes querem receber os dados requisitados de maneira rápida e íntegra, além de terem o conteúdo sempre disponível. Para solucionar esse problema, foram criadas técnicas baseadas em diferentes paradigmas e abordagens. Uma das primeiras técnicas desenvolvidas e mais amplamente utilizada é o modelo cliente-servidor.

Esse modelo consegue atender a altas demandas devido à replicação de conteúdo, mantendo cópias em múltiplos servidores (ou caches), que estão sempre conectados à Internet aguardando requisições dos clientes. No modelo cliente-servidor, um dispositivo cliente se conecta a um servidor que possui o conteúdo desejado e que possa enviar os dados ao cliente. Em geral, o controle sobre a entrega dos dados está localizada nos servidores, não no cliente. Assim, ao utilizar o modelo cliente-servidor para distribuir conteúdo, o problema torna-se decidir como e em qual servidor o cliente irá buscar o conteúdo. Alguns serviços bancários online e sistemas de e-mail via Web utilizam o modelo cliente-servidor para entregar seus dados.

Com o crescimento do número de usuários da Internet e da dificuldade de prover um serviço de entrega de dados de qualidade, surgiram as CDNs (*Content Delivery Network*, ou Rede de Fornecimento de Conteúdo), redes de servidores interligados, distribuídos em diversas partes do mundo, que são contratados pelos provedores com o objetivo de hospedar e prover conteúdo de forma transparente para os clientes com alto desempenho e disponibilidade. As CDNs foram criadas com o objetivo de complementar o modelo cliente-servidor. A Akamai [5] é hoje uma das maiores CDNs, hospedando serviços da NBC, Toyota, FedEx, e Apple, por exemplo.

Outra técnica desenvolvida posteriormente ao modelo cliente-servidor é a arquitetura P2P (*Peer-to-Peer* ou Par-a-Par), que, ao contrário das CDNs, tem como objetivo substituir a arquitetura cliente-servidor. O modelo P2P é caracterizado

como uma rede cooperativa entre clientes, que pode ou não ter uma entidade central. Quando essa existe, a função dessa é apenas coordenar clientes que desejam obter o mesmo conteúdo, e não necessariamente distribuir o conteúdo propriamente dito. O controle nesse modelo está praticamente todo nos clientes, sendo a distribuição do conteúdo feita apenas por eles. Esse paradigma revolucionou a distribuição de conteúdo por ser uma arquitetura escalável, eficiente, de alta disponibilidade e baixo custo.

## 1.3 Objetivo

O objetivo deste trabalho é um algoritmo de distibuição de conteúdo que segue o modelo cliente-servidor com múltiplos servidores mas que possui uma das características do modelo P2P. Em particular, o controle sobre quais servidores conectar-se é dinâmico e realizado integralmente pelos clientes. Movendo a solução do problema para o cliente, diferente da maioria das técnicas que utilizam o modelo cliente-servidor, esperamos atender as especificidades de cada usuário de acordo com a sua própria demanda. Além disso, como o algoritmo é dinânimo, se adapta a mudanças no ambiente, como congestionamento na rede, que poderiam causar perda de dados e assim diminuir a vazão para o cliente, e sobrecarrega ou falha de servidores. A ideia central é substituir dinamicamente servidores que estão oferecendo baixa qualidade de serviço. Um exemplo onde nosso algoritmo poderia ser implementado é o iTunes, de onde é possível fazer download de CDs, DVDs e livros.

Abordagens com algoritmos adaptativos que dividem o conteúdo em pequenos blocos de dados para transmissão e podem ou não mover a inteligência do sistema para o cliente vêm sendo estudadas por diversos grupos de pesquisa. Em especial diversas versões vêm sendo desenvolvidas para a proposta conhecida como *Multipath TCP*, um algoritmo adaptativo que utiliza múltiplos caminhos para entregar o conteúdo através de uma única conexão TCP, sendo assim transparente para a aplicação. No Capítulo 2 veremos melhor esses trabalhos e como eles se relacionam com a nossa proposta.

## 1.4 Contribuição da Tese

Esta dissertação tem três principais contribuições listadas a seguir.

- **Algoritmo adaptativo:** É proposto um algoritmo adaptativo para distribuição de conteúdo baseado em múltiplos servidores (ou múltiplos caminhos). A ideia deste algoritmo é buscar o menor número de servidores que saturam a banda do cliente, considerando que a qualquer momento um servidor pode sair

do sistema ou ficar sobrecarregado. A única informação prévia que os clientes utilizam para fazer a seleção de servidores são suas identidades (endereço IP). Iniciamente, o cliente conecta-se a apenas um servidor, com o objetivo de saturar sua banda de download. Caso seja necessário, novas conexões são adicionadas, uma por vez. Por haver um custo associado às conexões, o cliente avalia a vazão recebida, descartando servidores com baixo desempenho. Ao saturar a sua banda, o cliente passa então a diminuir o número de conexões buscando mantê-la saturada com um menor número de conexões. Esse algoritmo será melhor detalhado no Capítulo 3. Dentre suas vantagens podemos citar: clientes não necessitam comunicar-se entre si, resiliência a falhas e não requer conhecimento sobre a capacidade dos servidores ou sobre suas demandas.

- **Simulador:** Implementamos um simulador de eventos discretos, que modela precisamente o algoritmo sendo proposto de forma a avaliá-lo em diferentes cenários. O simulador possui eventos de chegada de clientes e servidores em diversos tempos da simulação, com a possibilidade de configurar tanto clientes como servidores com capacidades heterogêneas e também limitar o conhecimento dos clientes, informando a identidade de apenas um subconjunto dos servidores disponíveis. A partir disso, outros eventos são gerados no simulador, como conexão e desconexão entre clientes e servidores, falha de servidores e saída de clientes. Esse simulador está disponível publicamente [6], podendo ser utilizado para a avaliação de outros algoritmos de escolha de servidores e de alocação de fluxo, assim como implementação de novos eventos, como cancelamento de download e interação entre clientes. Os eventos processados no simulador, seus parâmetros, váriáveis e medidas de interesse serão detalhados no Capítulo 5.

- **Avaliação do algoritmo:** Avaliação da eficiência do algoritmo, tanto com relação à vazão obtida pelos clientes assim como sua adaptatibilidade considerando diferentes cenários, que possuem eventos de falha de servidores, entrada de clientes em rajadas e conhecimento limitado a cerca dos servidores diponíveis. Os resultados dessa avaliação indicam que o algoritmo é eficiente nos diferentes cenários, se adaptando bem às condições encontradas e oferencendo uma boa taxa de download aos clientes.

## 1.5 Estrutura da Tese

O restante desta dissertação está organizada da seguinte maneira. No Capítulo 2 descrevemos estudos e propostas relacionadas ao tema desta dissertação, e em seguida,

no Capítulo 3, detalhamos o algoritmo proposto para o problema da distribuição de conteúdo. O algoritmo utilizado pelos servidores para alocar suas capacidades aos clientes está detalhado no Capítulo 4. O simulador utilizado na avaliação da proposta e seus parâmetros estão descritos no Capítulo 5. Os resultados obtidos são apresentados no Capítulo 6. Finalmente, no Capítulo 7, apresentamos a conclusão deste estudo e trabalhos futuros.

# Capítulo 2

# Trabalhos Relacionados

A seguir, serão apresentados os trabalhos que abordam o mesmo tema desta dissertação.

## 2.1 Algoritmos para distribuição de contéudo

Multipath TCP (MPTCP) é uma nova proposta para o protocolo TCP que possui diversas implementações e estudos [7–10]. Essa proposta visa a utilização de múltiplos caminhos entre um mesmo cliente e servidor, para uma única conexão TCP, a fim de aumentar a eficiência e flexibilidade nos algoritmos de controle de congestionamento, utilizando os múltiplos caminhos para trasmissão dos pacotes de dados. É interessante notar que o problema de múltiplos caminhos entre um mesmo cliente e servidor está diretamente relacionado com o problema de múltiplos servidores com conteúdo replicado. Em particular, o segundo pode ser transformado no primeiro, adicionando-se um servidor virtual conectado a todos os servidores, como ilustrado na Figura 2.1 retirada de [1].

### 2.1.1 Seleção de caminhos e controle de congetionamento com múltiplos caminhos

Em [1] é feito um estudo sobre os benefícios da utilização de múltiplos caminhos quando existe um controle de fluxo, como é o caso do protocolo TCP. No estudo, o controle de fluxo é dividido em duas classes, coordenado e não coordenado. No primeiro, o fluxo a ser controlado é obtido como função (soma) do fluxo de todos os caminhos, enquanto no segundo o controle é feito em função do fluxo individual de cada caminho. No modelo matemático utilizado para a análise, o algoritmo de controle de fluxo é mapeado em uma função objetiva onde os clientes buscam os melhores caminhos maximizando sua utilização. O controle coordenado é modelado usando uma única função de utilização por cliente, onde o argumento da função é o

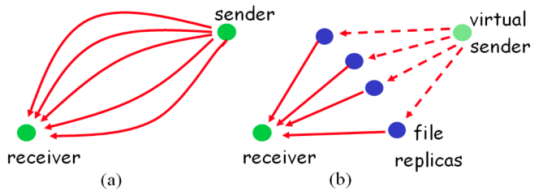



**Figure 1: (a) A canonical multipath example. (b) A BiTorrent example where a receiving peer receives data from four peers. A virtual sender has been included to show the relationship to canonical multipath.**

Figura 2.1: Retirado de [1] **(a)** Um exemplo do problema de múltiplos caminhos entre clientes e servidores **(b)** Um exemplo em redes P2P, onde um usuário recebe conteúdo de outros 4 usuários. Um servidor virtual foi incluido para mostrar sua relação com múltiplos caminhos

fluxo agregado dos caminhos. Já o controle não coordenado é modelado usando uma função por caminho, isto é, o argumento da função é o fluxo individual dos caminhos, e a soma é feita sobre a utilidade obtida em cada caminho. O estudo conclui que, quando o conjunto de caminhos é fixo, o caso coordenado possui melhor desempenho, porém, ao permitir que os clientes aleatoriamente troquem os caminhos buscando aumentar sua utilização, nos dois casos é possível chegar à utilização máxima, com a ressalva no caso não coordenado de que a função objetiva deve ser a mesma para todos os caminhos. Em especial, quando os caminhos podem ser substituidos, é possível alcançar a utilização máxima limitando os caminhos a um pequeno número, como por exemplo, dois.

### 2.1.2 Melhor performance e robustez

Resultados alcançados em [11] indicam que o uso do MPTCP leva de fato a uma melhor utilização da banda, melhor alocação de fluxo e maior vazão, além de maior resiliência a falhas. Em testes realizados em diversas topologias de datacenters e com diversos padrões de tráfego, MPTCP obteve sempre melhor ou igual desempenho sobre o TCP tradicional.

### 2.1.3 Controle de congestionamento

Apesar de resultados promissores, testes realizados em [10] indicam que o uso do MPTCP pode penalisar usuários do TCP tradicional, degradando sua vazão caso um bom algoritmo de congestionamento não seja utilizado pelo MPTCP. Nesse trabalho, um novo algoritmo é proposto e avaliado para mostrar que o mesmo atinge o ponto de equilíbrio ótimo de Pareto, satisfazendo os seguintes objetivos desejáveis em um

algoritmo de controle congestionamento:

- **Aumentar a vazão:** O fluxo total com múltiplos caminhos deve ser igual ou superior ao fluxo alcançado utilizando o único e melhor caminho disponível
- **Não atrapalhar:** Um fluxo de uma conexão com múltiplos caminhos não deve utilizar mais capacidade em nenhum dos caminhos do que utilizaria caso fosse um fluxo único
- **Balancear tráfego:** O fluxo com múltiplos caminhos deve balancear seu tráfego, de forma a respeitar os dois objetivos acima

## 2.2 Redes orientadas a conteúdo (ROC)

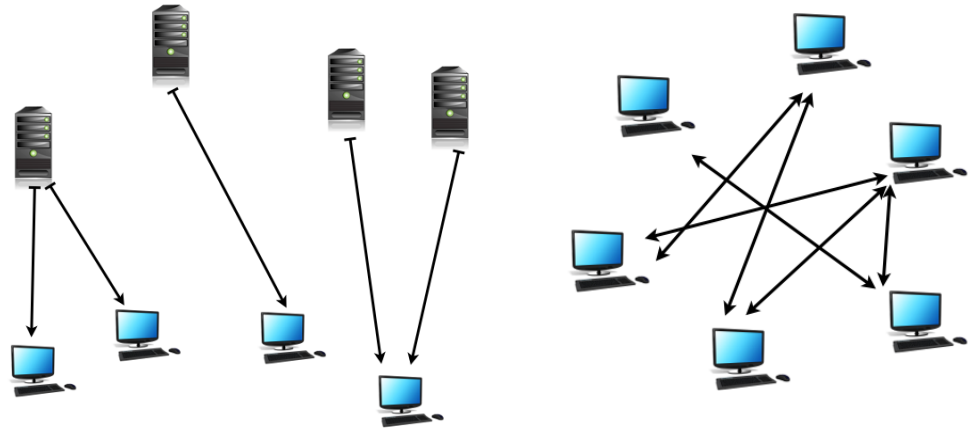
Outras propostas para resolver o problema da distribuição de conteúdo visam modificar de forma mais profunda os protocolos de rede, tendo como foco a entrega dos dados, e não mais a comunicação direta (fim-a-fim) entre os sistemas finais. Neste cenário, a localização do conteúdo não se torna importante, pois o protocolo é responsável também por localizar o conteúdo. As Redes Orientadas a Conteúdo visam aumentar a eficiência da entrega e a disponibilidade de conteúdos utilizando novos conceitos como conteúdo nomeado, roteamento baseado em nomes, segurança aplicada diretamente a conteúdo e armazenamento de dados nos elementos do núcleo da rede [12–14]. Essa solução implica modificações nos roteadores da rede, que devem ser capazes de armazenar e encaminhar dados baseando-se em identificadores de conteúdo, e não endereços de origem e destino, assim como processar novos tipos de pacotes de controle [15].

## 2.3 Paradigma P2P

Redes P2P constituem uma arquitetura de rede, construída na camada de aplicação, diferentemente das ROCs, que têm como alvo uma implementação na camada de rede. Isso é uma vantagem, pois soluções que exigem modificações no núcleo da rede, isto é, nos protocolos na camada de rede e equipamentos, são mais difíceis de serem adotadas ficando dependentes da padronização e colaboração ou acordo entre terceiros.

As redes P2P têm como foco prover e compartilhar recursos computacionais, visando uma maior escalabilidade do sistema. Ao participar dessa rede, além de aumentarem a demanda por recursos, os usuários agregam seus próprios recursos computacionais e ajudam no compartilhamento do conteúdo, por exemplo. Dessa forma, ao invés de possuir um provedor de conteúdo com capacidade de upload fixa,

a rede P2P aumenta sua capacidade a cada novo cliente, possibilitando implementar sistemas de distribuição com milhares de usuários a baixo custo [16] .



(a) Uma rede cliente-servidor. Servidores aguardam requisições de clientes e entregam conteúdo. Não há interação entre os clientes

(b) Uma rede P2P, onde a troca do conteúdo ocorre somente entre clientes, não existindo um servidor que aguarda requisições e entrega dados

Figura 2.2: Arquiteturas de distribuição de conteúdo

Apesar da arquitetura P2P ser eficiente e ter tornado-se muito popular, vulnerabilidades que podem afetar a rede impossiblitando ou atrasando a disseminação de conteúdo [17] e problemas de autenticidade e integridade dos dados ainda persistem [18]. Em sistemas com a arquitetura cliente-servidor, o controle sobre a integridade dos dados é maior, pois apenas os servidores dedicados proveem o conteúdo e soluções de contingência de ataques à servidores podem ser mais facilmente implementadas [19].

## 2.4 Justiça (*Fairness*)

Sistemas computacionais que possuem recursos a serem compartilhados (como CPU, barramento, banda e memória) utilizam diferentes mecanismos para alocar tais recursos entre clientes (ou processos) de maneira que todos sejam eventualmente servidos. A decisão de qual mecanismo utilizar é conhecida como problema de justiça. O conceito do que é uma divisão justa varia de acordo com cada sistema e existem diversas métricas e definições matemáticas para solucionar problemas de justiça no compartilhamento de recursos.

Como exemplo no dia-a-dia, podemos considerar o problema de dividir uma pizza entre oito pessoas. A partir da demanda, podemos estabelecer duas perguntas: quais são as possíveis divisões e quais divisões são mais justas? Uma divisão simples é distribuir a pizza igualmente para cada pessoa, ou seja, cada um recebe um oitavo

de pizza. Porém, essa seria uma divisão justa? Considere agora que uma das pessoas pese o dobro das outras, ou para facilitar a análise, considere que ela possui o dobro da fome. Dividir a pizza igualmente neste caso continuaria sendo uma boa divisão, ou seja, uma divisão justa? Uma outra maneira de resolver o problema seria dividir a pizza proporcionalmente ao peso de cada pessoa, isto é, a pessoa com o dobro do peso recebe o dobro de pizza. Como demonstramos brevemente, dividir a pizza é um exemplo de problema de justiça no compartilhamento de recursos.

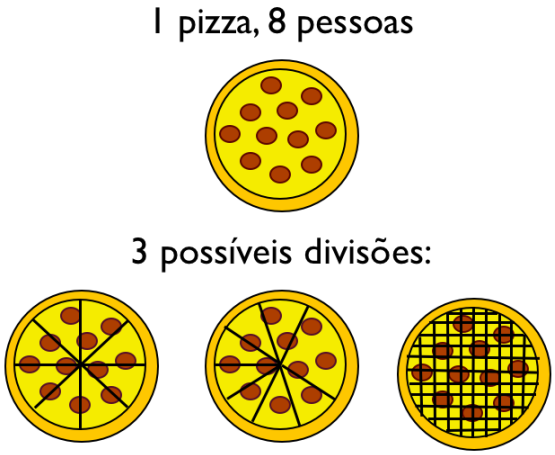



Figura 2.3: Exemplo de possíveis divisões de uma pizza para 8 pessoas, onde uma possui o dobro da fome das outras: à esquerda, 8 fatias iguais. Ao centro, 1 pedaço tem o dobro do tamanho dos outros 7. À direita, divisão à francesa.

O estudo da Justiça é um problema central na computação, pois a todo momento processos estão compartilhando recursos. Sistemas que não possuem bons mecanismos de justiça podem sofrer de um problema conhecido como indeterminismo ilimitado (*unbounded indeterminacy*), onde a quantidade de tempo necessária para servir um pedido pode tornar-se ilimitada, mesmo que o mecanismo garanta que em algum momento o pedido será servido [20]. Um mecanismo de alocação de recursos muito popular, que possui diversas aplicações nas literatura, é o método *Max-Min Fairness*, cujo objetivo é maximizar a alocação mínima que um cliente (ou processo) recebe do recurso compartilhado [21].

### 2.4.1 Max-Min Fairness e Preenchimento Progressivo

Intuitivamente, o mecanismo *Max-Min Fairness* aloca a menor demanda requerida para todos os clientes (ou processos) e em seguida distribui o restante do recurso igualmente para clientes que possuem maior demanda progressivamente, até esgotar o recurso. Formalmente, uma alocação *Max-Min Fairness* segue as seguintes condições:

1. Recursos são alocados em ordem crescente de demanda
2. Nenhum cliente recebe uma alocação maior que sua demanda
3. Nenhum cliente com menor demanda recebe maior alocação que um cliente de maior demanda
4. Clientes com demandas não-satisfeitas recebem a mesma alocação

Considerando o exemplo em que o recurso é a banda de um servidor, a divisão *Max-Min Fairness* entre um conjunto de clientes com capacidades fixas pode ser obtida através de um algoritmo de preechimento progressivo, que é feito da seguinte maneira. Iniciamos com todos os fluxos iguais a zero e simultaneamente incrementamos todos os fluxos igualmente até atingir o limite da capacidade dos clientes de menor demanda. Isto é, a vazão do fluxo de menor capacidade é maximizada. O fluxo para os clientes de menor demanda não será mais incrementado, já que as demandas estão satisfeitas e prosseguimos aumentado o fluxo para os clientes restantes da mesma maneira, tendo como limite a capacidade dos clientes de segunda menor demanda. Ao longo da alocação, todos os clientes que param de receber aumento de fluxo possuem necessariamente demandas saturadas. Dessa forma, esses clientes possuem a mesma ou menor vazão que os clientes de maiores demandas. Esse algoritmo prossegue até que não seja mais possível aumentar o fluxo para os clientes, ou porque o recurso foi esgotado (no caso, a capacidade do servidor foi esgotada), ou porque todas as demandas foram satisfeitas. Na Figura 2.4 temos um exemplo de como preencher recipentes com esse algoritmo, onde cada recipiente ilustra a demanda de cada cliente.

Para sistemas com múltiplos servidores, este problema não está bem definido, pois é necessário otimizar o fluxo alocado para clientes e o fluxo alocado dos servidores. Assim como os recursos recebidos pelos clientes devem ser justos, recursos alocados dos servidores também devem ser justos, de maneira a não sobrecarregar nenhum servidor, ou deixá-los ociosos ou subutilizados. Este problema está discutido com mais detalhes no Capítulo 4.

### 2.4.2 Definição Formal Max-Min Fairness

Considere uma rede com um conjunto $V$ de nós, que pode ser dividido nos dois subconjuntos a seguir: $v \in V_s$ se é fonte e $v \in V_c$ se é dreno, onde $v \in V$ possui capacidade $C_v$. Seja também $\mathcal{K}$ (de tamanho $K$) o conjunto de sessões ativas entre fontes e drenos, $M_k$ a vazão desejada para a sessão $k \in \mathcal{K}$, $\mathcal{K}_v$ o conjunto de sessões que o nó $v$ participa, e finalmente $\overrightarrow{z} = (z_1, z_2, ..., z_K)$ o vetor de de fluxos alocado para as sessões ativas. Podemos definir de maneira mais formal *Max-Min Fairness* da seguinte maneira. Para que um vetor $z$ de fluxos seja viável, para cada nó $v \in V$,

$$\sum_{k \in \mathcal{K}_v} z_k \leq C_v$$

e para cada sessão $k \in \mathcal{K}$ temos $z_k \leq M_k$. Um vetor de fluxos viável $\overrightarrow{z}$ é *Max-Min* se não é possível aumentar a vazão de uma sessão $z_p$, permanecendo uma alocação viável, sem diminuir a vazão de alguma outra sessão $z_q\,(p \neq q)$ com $z_p \leq z_q$.

Isto é, para qualquer outro vetor $\overrightarrow{y}$ viável, se esse possuir uma sessão com maior vazão que outra em $\overrightarrow{z}$, então existirá uma segunda em $\overrightarrow{z}$ ainda menor, e outra sessão em $\overrightarrow{y}$ inferior a todas as outras [22]:

$$(\exists p \in \mathcal{K}) y_p > z_p \Rightarrow (\exists q \in \mathcal{K}) y_q < z_q < z_p$$

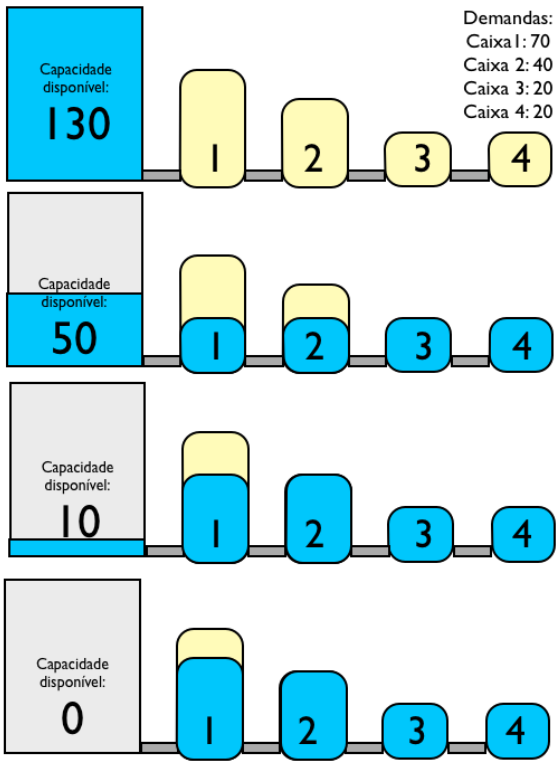



Figura 2.4: Exemplo de alocação Max-Min Progressive Filling

Voltando ao exemplo da pizza para oito pessoas, uma alocação *Max-Min Fairness* pode ser obtida cortando a pizza à francesa sem que os pequenos pedaços sejam alocados inicialmente. Ao invés, os pequenos pedaços de pizza, infinitesimais, são entregues para as pessoas igualmente até que as respectivas fomes sejam saciadas. Uma pessoa com menos fome irá comer menos que alguém tem mais fome. Ao final da pizza, pode haver pessoas que continuam com fome, entretanto elas terão recebido a mesma quantidade e o máximo possível sem privar pessoas com menos fome de qualquer pedaço de pizza.

# Capítulo 3

# Algoritmo Proposto

Neste capítulo, iremos apresentar o BitLivery, um algoritmo distribuído e adaptativo para download de conteúdo disponível em múltiplos servidores. O BitLivery é um algoritmo distribuído pois o mesmo é executado nos clientes. A decisão a quantos e quais servidores um cliente irá se conectar é tomada nos próprios clientes sem utilizar informação global do estado do sistema, como número de clientes ou demanda e capacidade de cada servidor. Em especial, o BitLivery é um algoritmo adaptativo, que permite que o cliente substitua, aumente ou diminua, suas conexões com os servidores, visando o aumento da vazão com o menor número de conexões. O objetivo é oferecer uma maior robustez com relação a mudanças na rede e nos servidores, garantindo um melhor serviço (taxa de download) para os clientes. A seguir, na Seção 3.1, descrevemos o cenário do problema que iremos solucionar. A notação utilizada nesta dissertação para discutir a proposta está resumida nas Tabelas 3.1 e 3.2.

## 3.1 Cenário

Em um sistema com $S$ servidores e $C$ clientes, temos que o servidor $s_i$, $1 \leq i \leq S$, possui a respectiva capacidade de upload $u_i$, $0 > u_i \leq U$, e o cliente $c_j$, $1 \leq j \leq C$, possui a respectiva capacidade de download $d_j$, $0 > d_j \leq D$. Consideramos que todos os servidores possuem o mesmo conteúdo de tamanho $L$, e aguardam requisições de clientes por esse conteúdo. Cada cliente irá conhecer a identidade de $W$ servidores, podendo estabelecer conexões com todos que estiverem disponíveis. Como estabelecer e manter múltiplas conexões TCP implicam impactos de desempenho conhecidos e estudados em [1, 23], como consumo de CPU, memória e largura de banda, utilizamos um mecanismo para desestimular a abertura de conexões arbitrárias. Para cada conexão que um cliente abra com um servidor, será feito um descréscimo na sua capacidade de download, reduzindo a utilidade da banda. Dessa forma, em conjunto com o algoritmo BitLivery, teremos um mecanismo para manter

| Parâmetro | Descrição |
|---|---|
| $L$ | Tamanho do conteúdo em bits |
| $S$ | Número máximo de servidores no sistema |
| $C$ | Número máximo de clientes no sistema |
| $U$ | Capacidade máxima de updload dos servidores |
| $D$ | Capacidade máxima de download dos clientes |
| $\rho$ | Porcentagem de banda gasta com custo por conexão |
| $W$ | Quantidade máxima de servidores que um cliente irá conhecer ao entrar no sistema |
| $u_i$ | Capacidade de upload do servidor $s_i$ |
| $d_j$ | Capacidade de download do cliente $c_j$ |

Tabela 3.1: Parâmetros do sistema

| Variável | Descrição |
|---|---|
| $n_i$ | Número de clientes conectados ao servidor $s_i$ |
| $m_j$ | Número de servidores conectados ao cliente $c_j$ |
| $k_{ij}$ | Indica se o servidor $s_i$ está conectado ao cliente $c_j$ |
| $f_{ij}$ | Fluxo alocado do servidor $s_i$ para o cliente $c_j$ |
| $a_{ij}$ | Número total de bits do conteúdo que o servidor $s_i$ enviou para o cliente $c_j$. |

Tabela 3.2: Descrição das variáveis do sistema

o número de conexões baixo.

Assim, podemos formalizar nosso problema da seguinte forma. Seja um grafo bipartido com $S$ fontes e $C$ drenos, onde cada fonte corresponde a um servidor e cada dreno corresponde a um cliente, onde cada fonte $s_i$ possui uma capacidade $u_i$ e cada dreno $c_j$ uma demanda $d_j$. As arestas irão representar os fluxos e por isso terão pesos. A soma dos pesos das arestas que saem de uma fonte está restrita pela capacidade da respectiva fonte, e a soma dos pesos das arestas que entram nos drenos está restrita à sua demanda menos um custo $p_j \times m_j$ sobre a capacidade, onde $p_j = \rho \times d_j$ e $\rho \in (0, 1)$, e $m_j$ é o número de arestas no dreno $c_j$. A partir desse cenário, como conectar as arestas de forma eficiente, sem sobrecarregar as fontes ou deixá-las ociosas, buscando atingir as demandas dos drenos utilizando o menor número de arestas? Ver Figuras 3.1 e 3.2.

Em um sistema centralizado, onde possuímos todas as informações necessárias, é relativamente fácil solucionar esse problema, montando e resolvendo o respectivo

$u_1$=10 $d_1$=3 $\rho$ =0,1
$u_2$=5 $d_2$=2 $p_1$=0,3
$u_3$=5 $d_3$=5 $p_2$=0,2
$u_4$=7 $d_4$=1 $p_3$=0,5
$d_5$=5 $p_4$=0,1
$d_6$=2 $p_5$=0,5
$d_7$=5 $p_6$=0,2
$p_7$=0,5

Figura 3.1: Capacidade das fonte, demanda dos drenos e custo por conexão

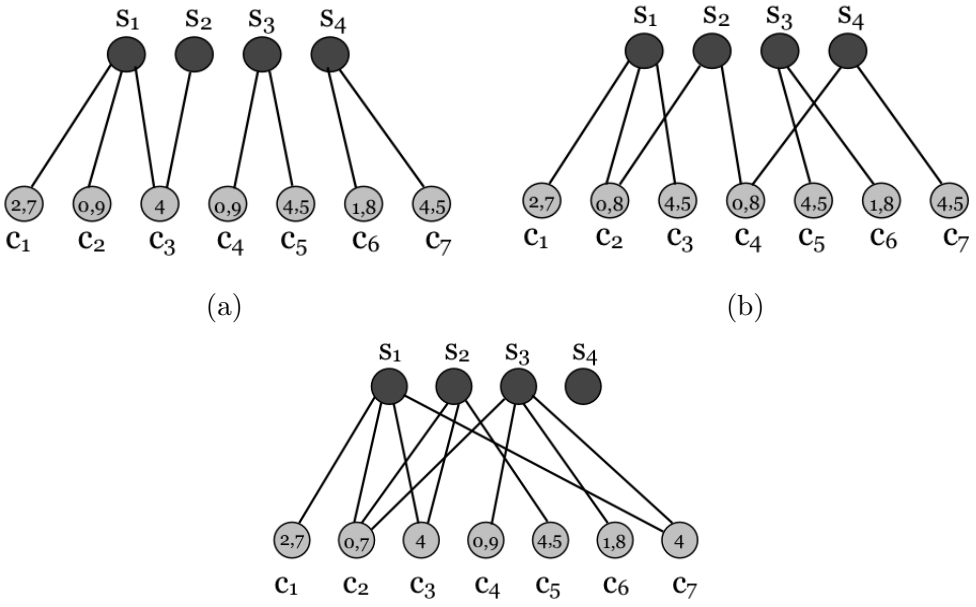



Figura 3.2: Três exemplos conexões possíveis. Dentro dos drenos temos a capacidade efetiva. Na Figura 4.1c, podemos ver um exemplo onde fontes estão sobrecarregadas se comparadas com a fonte $s_4$, que está ociosa. Em especial, os drenos $c_5$ e $c_7$ estão com suas demandas comprometidas conectando-se a fontes sobrecarredas, ao invés de conectar-se a apenas à fonte $s_4$, por exemplo.

problema de otimização desse modelo. Entretanto, em um sistema real é muito custoso manter o sistema atualizado e sincronizado de forma centralizada. Assim, devemos ser capazes de solucionar o problema de forma distribuída e com informação limitada. Neste trabalho, buscamos uma solução distribuída, localizada nos clientes. A única informação do sistema que os clientes possuem é a identidade de $W$ servidores e o fluxo recebido de cada conexão mantida com os servidores, que pode variar com o tempo. A capacidade dos servidores, o número de clientes no sistema e a demanda entre os outros clientes e servidores não são conhecidos.

## 3.2 BitLivery

O algoritmo de escolha de conexões que vamos propor tem como objetivo saturar a banda dos clientes, de forma a obter o menor tempo de download possível com o menor número de conexões. O BitLivery não possui memória, isto é, não utiliza conhecimento anteriormente adquirido sobre a capacidade dos servidores em seu processo de decisão. Dessa forma, um cliente sempre seleciona um servidor de forma aleatória e com probabilidade uniforme dentre os servidores conhecidos pelo cliente. Caso o cliente não encontre servidores disponíveis em sua lista ao selecionar novas conexões, o algoritmo será interrompido e novamente executado após um intervalo de tempo.

Ao entrar no sistema, o cliente irá conhecer a identidade de $W$ servidores escolhidos aleatoriamente dentre os $S$ servidores, e irá selecionar apenas um servidor para iniciar o download do conteúdo. Uma vez estabelecida esta conexão inicial, o cliente irá monitorar a vazão recebida e atualizar sua lista de conexões seguindo os critérios listados a seguir.

1. **Cliente possui uma conexão e banda saturada:** a conexão é mantida e nenhum novo servidor é adicionado. Esta é a única condição de estabilidade, no sentido de não gerar mudanças pelo cliente.

2. **Cliente possui mais de uma conexão e banda saturada:** os servidores que enviam um fluxo menor do que o custo por conexão são descartados, ou seja, tais conexões são encerradas. Caso nenhum servidor seja descartado por esse critério, a conexão de menor vazão é fechada.

3. **Cliente possui uma ou mais conexões e banda não saturada:** os servidores que enviam um fluxo menor do que o custo por aquela conexão são descartados. Caso o cliente esteja conectado a menos de $W$ servidores, um novo servidor é adicionado (diferente do(s) descartado(s)), ou seja, uma nova conexão é estabelecida com um dos servidores de conhecimento do cliente.

A motivação para fechar conexões que enviam um fluxo menor que o custo por conexão é diminuir a demanda sobre o servidor que demonstra estar sobrecarregado, além de buscar aproveitar melhor o custo da conexão com um novo servidor que possa oferecer um fluxo maior. É importante notar que, mesmo que mais de um servidor seja descartado por esse critério em uma análise do BitLivery, apenas uma nova conexão é adicionada por vez. Caso essa nova configuração de conexões não seja capaz de saturar a banda do cliente, um próximo servidor será adicionado em uma próxima análise. Já o descarte do servidor de menor fluxo quando a banda está saturada é motivado pela diminiução das conexões, consequentemente aumentando

a banda efetivamente disponível para dados, uma vez que o restante de conexões do cliente pode (ou não) ser capaz de suprir essa demanda.

### 3.2.1 Algoritmo

| Variável | Descrição |
|---|---|
| $Fluxo_s$ | Fluxo alocado pelo servidor $s$ a um cliente |
| $MenorFluxo$ | Valor do menor fluxo recebido por um cliente em uma conexão |
| $MenorFluxoId$ | Identificador do servidor que envia o menor fluxo para um cliente |
| $CustoConexao$ | Custo referente a uma conexão entre um cliente e um servidor |
| $BandaSaturada$ | Possui valor 1 se o cliente está com sua banda de download saturada, 0 caso contrário |
| $NaoHouveDescartes$ | Possui valor 1 se o cliente não descartou nenhum servidor em uma iteração do BitLivery, 0 caso contrário |
| $ServidoresConhecidos$ | Conjunto com a identidade dos servidores conhecidos por um cliente |
| $ServidoresConectados$ | Conjunto com a identidade dos servidores conectados a um cliente |
| $CapacidadeDownload$ | Capacidade de download de um cliente |
| $TentativasConexoes$ | Contabiliza tentativas de conexão com servidores |

Tabela 3.3: Variáveis utilizadas no Algoritmo 1

No Algoritmo 1 descrevemos em linguagem estruturada o procedimento de escolha de servidores proposto. As variáveis referenciadas ao longo do algoritmo estão descritas na Tabela 3.3 e as funções na Tabela 3.4. O parâmetro de entrada do BitLivery, executado em cada cliente de maneira independente, é o conjunto $ServidoresConhecidos$, recebido pelo cliente ao ingressar no sistema, que possui a identidade de um subconjunto de servidores. Além disso, o cliente utiliza a variável de sistema $Fluxo_s$, que indica a vazão recebida pelo cliente por cada servidor $s \in ServidoresConectados \subset ServidoresConhecidos$. A saída do algoritmo é a matriz de conexões atualizada. É importante notar que todas as informações utilizadas ao longo do algoritmo são locais no cliente. Mais que isso, o algoritmo é extramente simples e de custo linear no número de servidores que o cliente conhece.

As fases do algoritmo são

1. **Linha 2 a 4:** Inicialização de variáveis

| Função | Descrição |
|---|---|
| $exit(n)$ | Interrompe a execução do algoritmo e retorna o valor $n$ |
| $call(a, \delta_t)$ | Escalona execução de $a$ depois do intervalo de tempo $\delta_t$ |
| $is\_alive(s)$ | Verifica se o servidor $s$ está disponível no sistema |
| $open\_connection(s)$ | Estabelece uma conexão entre um cliente e o servidor $s$ |
| $close\_connection(s)$ | Encerra a conexão entre um cliente e o servidor $s$ |
| $select\_server(\Phi)$ | Seleciona com probabilidade uniforme um servidor do conjunto $\Phi$. Para facilitar a leitura do algoritmo, consideramos que, em uma execução do BitLivery, um mesmo servidor não é sorteado duas vezes. |

Tabela 3.4: Funções utilizadas no Algoritmo 1

2. **Linha 5 a 11:** Busca pelo servidor de menor vazão e desconexão dos servidores com vazão menor que o custo por conexão

3. **Linha 12 a 20:** Seleciona um servidor disponível caso o cliente não esteja conectado a nenhum, isto é, se o conjunto $ServidoresConhecidos$ é vazio. Caso não haja servidores disponíveis, o algoritmo é interrompido e reescalonado para ser executado após um intervalo $\delta_t$

4. **Linha 22 a 24:** Se o cliente está saturado, nenhum servidor foi descartado por ter vazão menor que o custo por conexão e o cliente está conectado a mais de um servidor, o servidor de menor vazão é selecionado para descarte

5. **Linha 26 a 35:** Se o cliente não está saturado e não está conectado a todos os servidores disponíveis a ele, um novo servidor é selecionado aleatoriamente. Caso não haja servidores disponíveis, o algoritmo é interrompido e reescalonado para ser executado após um intervalo $\delta_t$

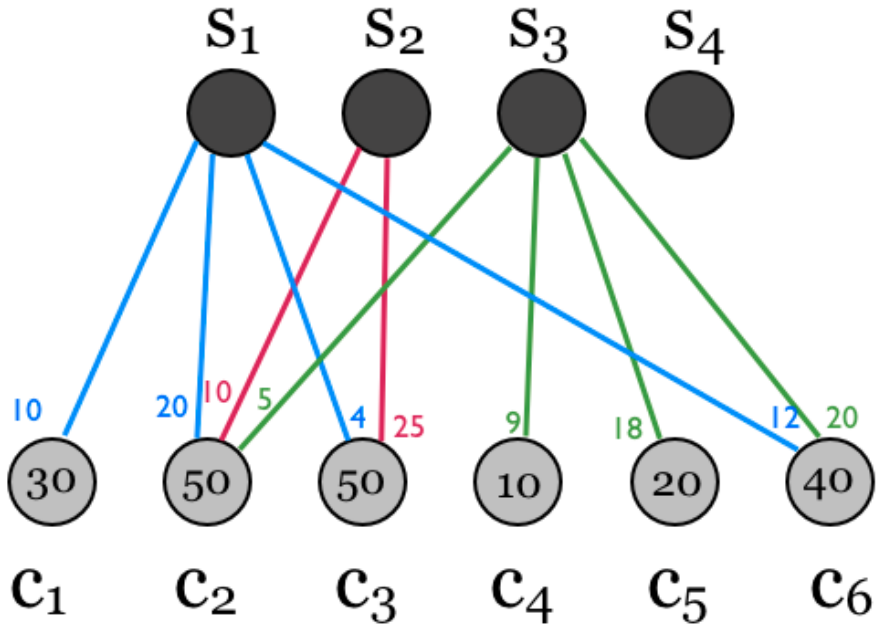
**Algoritmo 1**: BitLivery - Algoritmo de escolha de conexões

```
1  início
2      NaoHouveDescartes ← 1;
3      TentativasConexoes ← 0;
4      MenorFluxo ← CapacidadeDownload;
5      foreach s in ServidoresConectados do
6          se Fluxo_s < CustoConexao então
7              close_connection(s);
8              NaoHouveDescartes ← 0;
9          se MenorFluxo > Fluxo_s então
10             MenorFluxo ← Fluxo_s;
11             MenorFluxoId ← s;
12     se ServidoresConectados = ∅ então
13         repeat
14             se TentativasConexoes = |ServidoresConhecidos| então
15                 call(BitLivery, δ_t);
16                 exit(1);
17             s = select_server(ServidoresConhecidos);
18             TentativasConexoes ← TentativasConexoes + 1;
19         until is_alive(s) ;
20         open_connection(s);
21     senão
22         se BandaSaturada então
23             se NaoHouveDescartes & |ServidoresConectados| > 1 então
24                 close_connection(MenorFluxoId);
25         senão
26             se |ServidoresConectados| < |ServidoresConnhecidos| então
27                 ServidoresRestantes =
                   ServidoresConhecidos − ServidoresConectados;
28                 repeat
29                     se TentativasConexoes = |ServidoresRestantes| então
30                         call(BitLivery, δ_t);
31                         exit(1);
32                     s = select_server(ServidoresRestantes);
33                     TentativasConexoes ← TentativasConexoes + 1;
34                 until is_alive(s) ;
35                 open_connection(s);
36 fim
```

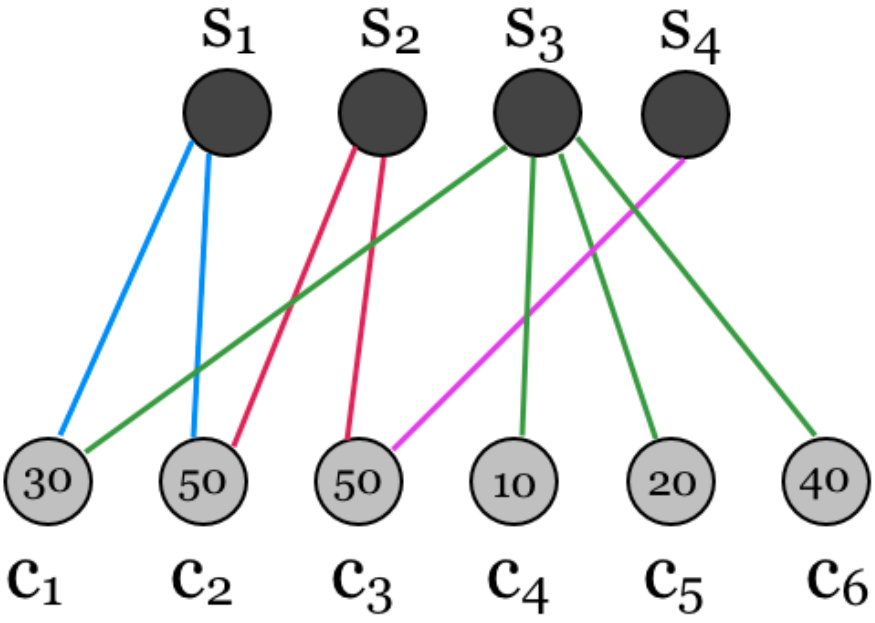
### 3.2.2 Exemplo

A Figura 3.3 ilustra um exemplo do funcionamento do algoritmo BitLivery. Dentro dos drenos $c_1 - c_6$ estão as respectivas capacidades, onde $\rho = 0,1$, assim o custo por cada conexão é equivalente a 10% do valor da capacidade. Ao lado de cada aresta da Figura 3.3a temos um valor de mesma cor referente ao seu peso (o fluxo alocado pela fonte). Ao aplicar o algoritmo Bitlivery nesse cenário, cada dreno irá tomar as seguintes ações:

- $c_1$ − Como não está com sua demanda saturada, estabelece uma nova aresta (conexão), sorteando a fonte $s_3$
- $c_2$ − Como está com sua demanda saturada, e possui mais de uma aresta, descarta a fonte com aresta de menor peso (fluxo) $s_3$
- $c_3$ − Descarta a fonte $s_1$, pois o peso de sua aresta é menor que o custo por conexão e estabelece uma nova aresta, sorteando a fonte $s_4$
- $c_4$ − Mantém a configuração atual pois está com sua demanda saturada e possui apenas uma aresta
- $c_5$ − Mantém a configuração atual pois está com sua demanda saturada e possui apenas uma aresta
- $c_6$ − Como está com sua demanda saturada, e possui mais de uma aresta, descarta a fonte com aresta de menor peso $s_1$




(a) Representação da matriz $K(t)$ que representa a configuração das conexões no tempo $t$




(b) Nova configuração da matriz $K(t')$ no tempo $t'$, após execução do algortimo BitLivery em todos os drenos

Figura 3.3

# Capítulo 4

# Alocação de banda entre clientes e servidores

O problema de como os clientes devem se conectar a servidores discutido no Capítulo 3 depende fundamentalmente de como as capacidades dos servidores serão alocadas aos fluxos dos clientes. Para resolver o problema de distribuição de conteúdo é então necessário solucionar dois problemas: como conectar os clientes aos servidores e como alocar os recursos (a capacidade dos servidores) para cada conexão. Considerando a configuração de conexões apresentada na Figura 3.2a, é possível alocar os fluxos de diversas formas, como apresentamos na Figura 4.1, onde o valor de $f_{ij}$ representa o fluxo alocado da fonte $s_i$ para o dreno $c_j$. Qual dessas alocações é mais adequada? Como estabelecer um mecanismo de alocação a partir de uma configuração de conexões entre servidores e clientes? Neste Capítulo será descrito o mecanismo utilizado pelos servidores para alocar a banda aos clientes que será utilizado nesta dissertação, considerando que os clientes têm capacidades distintas, possuímos múltiplos servidores e temos um padrão de conexão estabelecido.

Retomando o problema da pizza descrito na Seção 2.4.1, podemos considerar as diferentes capacidades (ou demandas) dos clientes como pessoas com diferentes fomes e múltiplos servidores como diversas pizzas. Além disso, imagine que cada pessoa se alimente de apenas um subconjunto de pizzas por vez, representando as conexões. Como distribuir os pedaços de forma que as pizzas sejam igualmente consumidas e as pessoas fiquem o mais satisfeitas possível ( minimize o "restante de fome")? Como as pessoas estão sendo servidas por diversas pizzas, a fome de cada pessoa do ponto de vista de cada pizza não é fixa, pois a todo momento ela pode ou não estar sendo suprida por uma outra pizza.

$f_{11} = 2{,}7$ $f_{22} = 0{,}0$ $f_{35} = 3{,}2$ $f_{44} = 0{,}0$
$f_{12} = 0{,}8$ $f_{24} = 0{,}8$ $f_{36} = 1{,}8$ $f_{47} = 4{,}5$
$f_{13} = 4{,}5$

(a)

$f_{11} = 2{,}7$ $f_{22} = 0{,}4$ $f_{35} = 3{,}6$ $f_{44} = 0{,}3$
$f_{12} = 0{,}4$ $f_{24} = 0{,}5$ $f_{36} = 1{,}4$ $f_{47} = 4{,}5$
$f_{13} = 4{,}5$

(b)

$f_{11} = 2{,}5$ $f_{22} = 0{,}0$ $f_{35} = 2{,}5$ $f_{44} = 0{,}0$
$f_{12} = 0{,}8$ $f_{24} = 0{,}8$ $f_{36} = 1{,}8$ $f_{47} = 2{,}5$
$f_{13} = 2{,}5$

(c)

Figura 4.1: Três alocações possíveis para a configuração ilustrada na Figura 3.2a.

## 4.1 Alocação

O algoritmo de alocação de banda que utilizamos no nosso trabalho é motivado pelo algoritmo *Max-Min Fairness Progressive Filling* descrito na Seção 2.4.1. Esse mecanismo foi escolhido por ser uma alocação natural, da qual diversos sistemas reais e estudos também se baseiam [24, 25]. É importante notar que a alocação dos recursos não é o foco deste trabalho, sendo necessária pois o algoritmo de escolha de servidores precisa realizar decisões dada uma alocação de fluxos para as conexões dos clientes.

A alocação ocorre a cada conexão e desconexão dos clientes, ou seja, toda vez que a matriz de conexão $K$ muda, e é feita de forma centralizada e assumindo conhecimento das capacidades dos clientes. Desta forma, a cada mudança de conexão, os fluxos aos clientes são recalculados respeitando suas demandas. A alocação é realizada da seguinte forma: cada servidor divide a sua capacidade igualmente entre os clientes a ele conectados, isto é, $u_i/n_i$, e um a um vão alocando essa fração a seus clientes. Caso um cliente necessite um fluxo menor do que $u_i/n_i$ para saturar sua banda, apenas a banda necessária é alocada e o cliente é saturado. Se após o último servidor realizar sua alocação, algum servidor ainda possuir banda disponível para alocar e possuir clientes conectados a ele que não saturaram sua banda, a banda restante é novamente dividida para esses clientes, e o processo de alocação prossegue repetidamente até saturar todos os clientes ou esgotar as capacidades dos servidores. Assim, podemos considerar que esse algoritmo é *work conserving*, já que toda banda que pode ser alocada é aproveitada.

Idealmente, cada servidor poderia alocar uma fração de banda infinitesimal a cada cliente conectado por iteração do algoritmo. O processo de alocação de cada

servidor poderia se repetir quantas vezes fosse necessário até distribuir toda a capacidade de cada um dos servidores progressivamente. Neste caso, a alocação centralizada seria o mais próximo possível do *Max-Min Fairness Progressive Filling* do ponto de vista de um único servidor, tornando a alocação mais justa dentro deste cenário. Porém, ao diminuir a fração alocada a cada cliente, mais iterações são necessárias para que a alocação convirja, aumentando o custo computacional do algoritmo de alocação. Dessa forma, temos um *tradeoff* entre uma melhor alocação dos recursos e o custo computacional do algoritmo. O desenvolvimento de um algoritmo de alocação mais eficiente está fora do escopo deste trabalho, tendo em vista que o foco da dissertação é o algoritmo de escolha de conexões e o algoritmo cumpre seu objetivo de forma satisfatória.

## 4.2 Formalização

Podemos formalizar o problema de alocação de banda entre servidores e clientes da seguinte forma. Determinar os valores de fluxo $f_{ij}$ (do servidor $s_i$ para o cliente $c_j$) da matriz de fluxos $F$, tal que

1. $\sum_{i=1}^{S} f_{ij} \leq d_j - O(m_j)$
2. $\sum_{j=1}^{C} f_{ij} \leq u_i$
3. $f_{ij} \geq 0$
4. $m_j = \sum_{i=1}^{S} k_{ij}$

Considerando como condição inicial

- $\sum_{i=1}^{S} a_{ij} = 0$

E condição de parada

- $\sum_{i=1}^{C} a_{ij} = L$

Lembrando que $O(m_j)$ é uma função de custo associado ao número de conexões que o cliente $c_j$ possui, que reduz a capacidade útil dos clientes. Exemplos de função de custo são $\alpha \times m_j$, onde $\alpha$ é uma constante positiva, e $\log m_j$. No nosso modelo, a função de custo é a primeira, onde $\alpha$ é uma função linear da capacidade do cliente, a saber $\rho \times d_j$, onde $\rho \in (0, 1)$. Como a capacidade do cliente é fixa, $\alpha$ é constante.

## 4.3 Algoritmo

O Algoritmo 2 apresenta em linguagem estruturada o procedimento de alocação de banda descrito informalmente na Seção 4.1. As variáveis referenciadas estão descritas na Tabela 4.1. Os parâmetros de entrada do algoritmo são os vetores de conexão de cada servidor com os clientes, as capacidades de upload dos servidores e as capacidades de download efetivas dos clientes. A saída do algoritmo são os fluxos alocados por cada servidor aos clientes.

| **Variável** | **Descrição** |
|---|---|
| $Servidores$ | Conjunto com a identidade de todos os servidores |
| $Clientes$ | Conjunto com a identidade de todos os clientes conectados ao sistema |
| $FluxoMedio$ | Fração de banda alocada aos clientes por um servidor em uma iteração do algoritmo de alocação de banda |
| $s.Fluxo_c$ | Fluxo alocado pelo servidor $s$ ao cliente $c$ |
| $s.Conectados_c$ | Possui valor 1 se o servidor $s$ está conectado ao cliente $c$, 0 caso contrário |
| $s.ClientesAservir$ | Indica quantos clientes não saturados estão conectados ao servidor $s$ |
| $c.Saturado$ | Possui valor 1 se o cliente $c$ está com a banda saturada, 0 caso contrário |
| $s.CapacidadeUpload$ | Capacidade de upload do servidor $s$ |
| $c.CapacidadeDownloadEfeitva$ | Capacidade efetiva de download do cliente $c$, isto é, sua capacidade total de download menos o custo pelas conexões estabelecidas |
| $s.CapacidadeRestante$ | Capacidade de upload não alocada do servidor $s$ |
| $c.CapacidadeRestante$ | Capacidade de download não alocada do cliente $c$ |

Tabela 4.1: Variáveis utilizadas no Algoritmo 2

As fases do algoritmo são

1. **Linha 2 a 9:** Inicialização das variáveis $CapacidadeRestante$, $ClientesAservir$ e $Fluxo$ para todos os servidores, e $CapacidadeRestante$ e $Saturado$ para todos os clientes

2. **Linha 14 a 15:** Caso o servidor possua clientes não saturados e possua fluxo suficiente para alocar, a variável $FluxoMedio$ é calculada

3. **Linha 17 a 21:** Caso o cliente e o servidor estejam conectados, o cliente não esteja saturado, e o cliente possua capacidade suficiente disponível, a fração $FluxoMedio$ é adicionada à vazão do servidor ao cliente, e as respectivas capacidades restantes são recalculadas

4. **Linha 22 a 25:** Caso o cliente e o servidor estejam conectados, o cliente não esteja saturado e o cliente possua uma capacidade disponível menor que $FluxoMedio$, só o necessário é alocado, o cliente passa para o estado saturado, e as capacidades restantes são recalculadas.

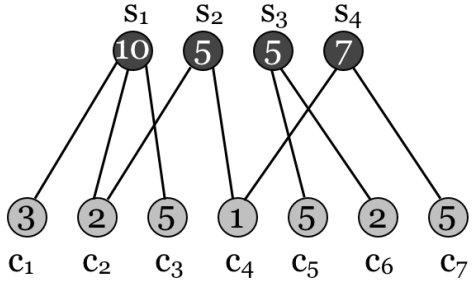
5. **Linha 22 a 25:** Servidores conectados a um cliente que teve sua banda saturada decrementam a quantidade de clientes a servir

**Algoritmo 2**: Alocação de banda

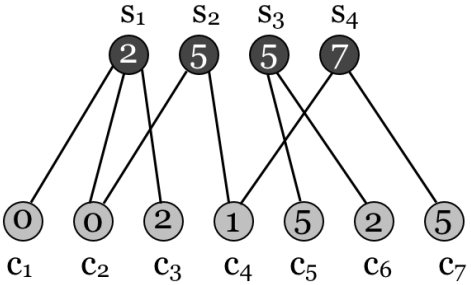
```
1  início
2      foreach s in Servidores do
3          s.CapacidadeRestante ← s.CapacidadeUpload;
4          foreach c in Clientes do
5              se s.Conectados_c então
6                  s.ClientesAservir ← s.ClientesAservir + 1;
7              s.Fluxo_c ← 0;
8              c.Saturado ← 0;
9              c.CapacidadeRestante ← c.CapacidadeDownloadEfetiva;
10     ContinuarAlocacao = 1;
11     while ContinuarAlocacao do
12         ContinuarAlocacao ← 0;
13         foreach s in Servidores do
14             se s.ClientesAservir ≠ 0 &
               s.CapacidadeRestante ≥ s.ClientesAservir então
15                 FluxoMedio ← ⌊s.CapacidadeRestante / s.ClientesAservir⌋;
16                 foreach c in Clientes do
17                     se s.Conectados_c & not(c.Saturado) então
18                         se FluxoMedio ≤ c.CapacidadeRestante então
19                             s.Fluxo_c ← s.Fluxo_c + FluxoMedio;
20                             s.CapacidadeRestante ←
                               s.CapacidadeRestante − FluxoMedio;
21                             c.CapacidadeRestante ←
                               c.CapacidadeRestante − FluxoMedio;
22                         senão
23                             s.Fluxo_c ← s.Fluxo_c + c.CapacidadeRestante;
24                             s.CapacidadeRestante ←
                               s.CapacidadeRestante − c.CapacidadeRestante;
25                             c.Saturado ← 1;
26                             foreach k in Servidores do
27                                 se k.Conectados_c então
28                                     k.ClientesAservir ← k.ClientesAservir − 1;
29                     ContinuarAlocacao ← 1;
30 fim
```

## 4.4 Exemplo

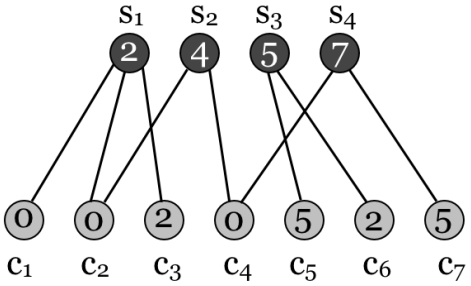
Na Figura 4.2 apresentamos um exemplo de alocação para determinar os fluxos sobre um padrão de conexões, representado por um grafo de conexão bipartido com capacidades heterogêneas. Na Figura 4.2a temos dentro de cada fonte e dreno suas respectivas capacidades e demandas. Para facilitar a apresentação, vamos considerar que as demandas dos drenos já tiveram o custo por conexão removidos, e por isso apresentamos já a demanda efetiva. Nas Figuras seguintes, as capacidades e demandas das fontes e drenos são reduzidas de acordo com a utilização dos recursos.



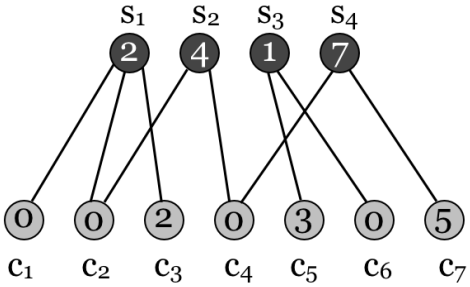

(a) Do nó $s_1$ ao $s_4$ temos as fontes, com suas respectivas capacidades. Do nó $c_1$ ao $c_7$ os drenos com suas respectivas demandas.



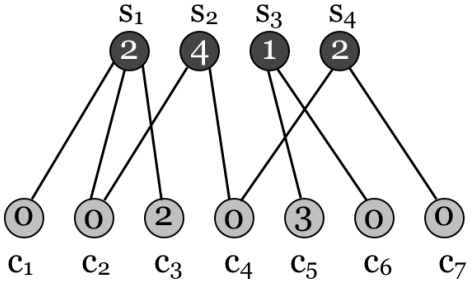

(b) Nó $s_1$ aloca 3 unidades para nó $c_1$, 2 para nó $c_2$, e 3 para o nó $c_3$, ficando com 2 unidades.



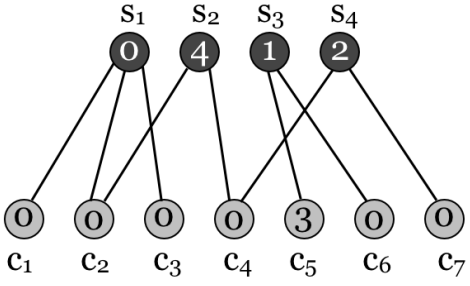

(c) Nó $s_2$ aloca 1 unidade para nó $c_4$, ficando com 4 unidades.



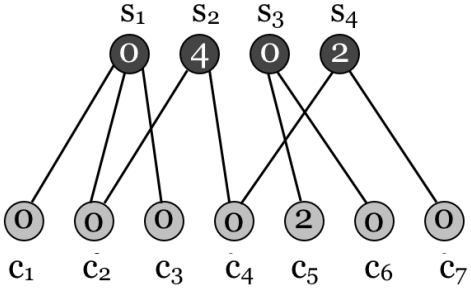

(d) Nó $s_3$ aloca 2 unidades para nó $c_5$ e 2 para o nó $c_6$, ficando com 1 unidade.




(e) Nó $s_4$ aloca 5 unidades para nó $c_7$, ficando com 2 unidades.




(f) Nó $s_1$ aloca 2 unidade para nó $c_3$, ficando esgotado.




(g) Nó $s_3$ aloca 1 unidade para nó $c_5$, ficando esgotado. A locação termina, pois não há nenhuma fonte com capacidade restante conectada a um dreno não saturado.

$f_{11} = 3$ $f_{22} = 0$ $f_{35} = 3$ $f_{44} = 0$
$f_{12} = 2$ $f_{24} = 1$ $f_{36} = 2$ $f_{47} = 5$
$f_{13} = 5$

(h) Valores finais dos elementos da matriz de Fluxo $F$ para os nós conectados após executar o algoritmo de alocação de banda. O restante dos valores não representados são iguais a zero.

Figura 4.2: Exemplo de alocação.

# Capítulo 5

# Simulador

Para avaliar o desempenho e funcionalidade do algoritmo proposto neste trabalho, foi projetado e implementado um simulador de eventos discretos. Simuladores de eventos discretos modelam o funcionamento de sistemas a partir do processamento de uma sequência de eventos dispostos em intervalos discretos de tempo. Esses eventos são organizados em uma fila em ordem crescente do tempo de execução previsto. O tempo atual do sistema é o tempo do primeiro evento da fila. Ao processar esse evento, o evento é descartado e novos eventos que ocorrerão no futuro podem ou não ser gerados. A ordem em que eventos são gerados não é necessariamente a ordem em que os eventos serão processados, pois o processamento ocorre de acordo com o tempo de execução do evento e não seu tempo de geração. Variáveis que definem o estado do sistema somente podem ser modificadas durante a execução de um evento, nunca entre eventos consecutivos. Dessa forma, a simulação salta no tempo de um evento para o próximo na fila sem perder consistência. Os estados e variáveis que definem o sistema do simulador utilizado neste trabalho serão descritos na Seção 5.1.

De maneira geral, podemos descrever o funcionamento do simulador desde o ponto de vista de um cliente da seguinte forma. O cliente se conecta ao sistema e recebe uma lista com a identidade de um subconjunto de servidores que possuem um determinado conteúdo. Este cliente irá estabelecer uma conexão com um servidor arbitrário dessa lista, e, após um tempo, irá perceber uma mundaça no fluxo total de dados recebido por intervalo de tempo. Caso sua banda de download (efetiva) não esteja saturada, um segundo servidor será conectado. Esse processo irá se repetir, adicionando servidores de maneira aleatória um por vez, até que a banda de download esteja saturada. Além da vazão total recebida, a vazão individual de cada servidor também é monitorada. Caso o servidor esteja enviando um fluxo muito pequeno, esse será substituido em busca de outro que possa oferecer melhor vazão. Ao saturar a banda, o cliente irá buscar manter sua banda saturada com o menor número de servidores possível, iniciando assim um processo reverso de descarte gradual de servidores. Vale lembrar que é vantajoso manter o número de

conexões baixo, já que para cada conexão, uma fração da capacidade efetiva de download do cliente é reduzida. Assim, o cliente irá descartar o servidor de menor vazão e verificar se os servidores restantes continuam santurando sua banda. Em caso positivo, o processo se repete, até chegar a somente um servidor que sature sua banda ou até perceber que sua banda não está mais saturada, interrompendo o processo de descarte e, no segundo caso, voltando a adicionar servidores. Como a cada conexão e desconexão entre clientes e servidores o fluxo alocado por todos os servidores é recalculado a todos os clientes do sistema, a vazão total recebida pelo cliente pode aumentar ou diminuir ao longo do donwload, levando-o a adicionar ou remover servidores. Essa dinâmica de adição e descarte de servidores prossegue até que o cliente eventualmente termine o download do conteúdo e deixe o sistema, desconectando-se de todos os servidores para os quais estava conectado.

Esse simulador foi implementado inteiramente na linguagem C sem o uso de bibliotecas de simulação adicionais e está publicamente disponível e documentado tanto para análise quanto alteração do código [6]. Esse simulador pode ser facilmente adaptado para avaliar diversos algoritmos, tanto de alocação de banda, quanto de seleção de servidores. Além disso, novas funcionalidades e eventos podem ser adicionados, além de novas varíaveis para coletar mais informações sobre o estado do sistema.

## 5.1 Implementação

Nesse simulador, o estado do sistema é mantido utilizando as variáveis descritas na Tabela 3.2 do Capítulo 3. Além das variáveis de estado, o simulador possui diversos parâmetros que devem ser pré configurados, listados a seguir.

### 5.1.1 Parâmetros

Os principais parâmetros do simulador estão listados a seguir.

- **Tamanho do arquivo:** Tamanho do conteúdo a ser obtido pelos clientes em bits.
- **Número de clientes:** Número máximo de clientes que pode haver no sistema ao mesmo tempo.
- **Número de servidores:** Número máximo de servidores que pode haver no sistema ao mesmo tempo.
- **Conhecimento:** Número de servidores que um cliente conhece ao entrar no sistema. Este parâmetro é utilizado para limitar o conhecimento do cliente sobre o conjunto de servidores.

- **Tempo máximo de simulação:** Tempo máximo que uma simulação pode durar.

- **Taxa de entrada/saída do sistema:** Parâmetro $\lambda_1$ utilizado para criar um valor $\delta_1$ com distribuição exponencial para escalonar o evento de entrada ou saída do sistema. Este atraso é introduzido para simular o tempo de processamento e transmissão que pacotes enfrentam de um ponto a outro na Internet.

- **Taxa de mudança de fluxo:** Parâmetro $\lambda_2$ utilizado para criar um valor $\delta_2$ com distribuição exponencial para escalonar o evento de mundaça de fluxo. Este atraso representa o tempo que o cliente leva para perceber e estimar a variação da vazão recebida.

- **Taxa de conexão/desconexão:** Parâmetro $\lambda_3$ utilizado para criar um valor $\delta_3$ com distribuição exponencial para escalonar o evento de conexão ou desconexão. Possui a mesma intuição de $\lambda_1$. Devemos assumir que $\delta_2 \gg \delta_3$, pois podemos dizer que o tempo para estimar a vazão recebida por um servidor é pelo menos uma ordem de grandeza acima do tempo para estabelecer uma conexão ou desconexão TCP. Além de ser uma consideração razoável, essa premissa é necessária para manter a consistência dos estados do simulador. Caso um evento de mudança de fluxo para um determinado cliente seja alocado antes que um evento de conexão ou desconexão previamente alocado para esse mesmo cliente, poderíamos gerar inconsistências, como o exemplo a seguir. Suponha que esse cliente tenha previamente pedido para desconectar-se de um determinado servidor. Para o cliente, esse servidor não estará mais no conjunto de servidores conectados, porém, a desconexão só será concretizada de fato quando o evento for executado, isto é, quando o pedido chegar ao servidor. Um novo evento de fluxo alocado sem a premissa poderia gerar um evento de pedido de conexão para esse mesmo servidor com tempo de execução prévio ao pedido desconexão.

- **Limiar para mudança de fluxo:** Caso o novo fluxo tenha uma diferença $\Delta_f$ maior que $\frac{limiar \times FluxoAntigo}{100}$ entre o fluxo antigo, para mais ou para menos, uma alteração no fluxo é percebida pelo cliente. Este limiar existe para desconsiderar pequenas variações no fluxo comuns em sistemas reais.

- **Custo por conexão:** Porcentagem da capacidade dos clientes, que a cada conexão é reduzida da capacidade de downlod do cliente. Isto é, o custo por conexão para um cliente que possui a capacidade de download $d$, é $\frac{custo \times d}{100}$, onde $0 < custo < 50$.

- **Capacidade dos clientes:** Pode ser homogênea ou heterogênea. No caso heterogêneo, é possível atribuir valores alteatórios entre um máximo e um mínimo ou dividir os clientes em categorias com duas ou três capacidades distintas, sendo a proporção de clientes em cada categoria em grupos de 50%-50%, 34%-66% e 33%-33%-34%.

- **Capacidade dos servidores:** Pode ser homogênea ou heterogênea. No caso heterogênio, é possível atribuir valores alteatórios entre um máximo e um mínimo ou dividir os servidores em categorias com duas ou três capacidades distintas, sendo a proporção de servidores em cada categoria em grupos de 50%-50%, 34%-66% e 33%-33%-34%.

### 5.1.2 Lista de eventos

Como visto anteriormente, variáveis de sistema somente podem ser modificadas durante a execução de eventos. Além disso, o simulador de eventos discreto projetado neste trabalho utiliza uma fila para oganizar seus eventos. Eventos, ao serem criados, recebem um carimbo de tempo que representa o momento na simulação em que o evento deve ser executado. Dessa forma, após serem criados, eventos são inseridos na fila em ordem crescente de execução no tempo. Os eventos do simulador estão listados a seguir e são descritos na Seção 5.1.3.

- **Entrada do cliente $c_j$:** representa a entrada do $j$-ésimo cliente no sistema.

- **Saída do cliente $c_j$:** representa a saída do $j$-ésimo cliente do sistema.

- **Entrada do servidor $s_i$:** representa a entrada do $i$-ésimo servidor no sistema.

- **Saída do servidor $s_i$:** representa a saída do $i$-ésimo servido no sistema.

- **Conexão do cliente $c_j$ com o servidor $s_i$:** representa o recebimento de um pedido de conexão do cliente $c_j$ pelo servidor $s_i$.

- **Desconexão do cliente $c_j$ com o servidor $s_i$:** representa o recebimento de um pedido de desconexão do cliente $c_j$ pelo servidor $s_i$ .

- **Timeout de conexão do servidor $s_i$ para o cliente $c_j$:** representa a falha de conexão recebida pelo cliente $c_j$ do servidor $s_i$.

- **Mundança de fluxo do cliente $c_j$:** representa uma mudança de fluxo em uma ou mais conexões do cliente $c_j$.

Todo cliente possui um evento de saída escalonado, que será inicializado com um valor muito grande e atualizado ao longo da simulação, de acordo com o progresso

do download e o fluxo instantâneo agregado alocado ao cliente. Associado a esse cliente também existirá apenas um ou nenhum evento de mudança de fluxo, que será reescalonado na lista de eventos de tempos em tempos, de acordo com a variação da alocação de fluxo pelos servidores.

### 5.1.3 Processamento dos eventos

O processamento de cada evento está descrito abaixo, detalhando como as variáveis de estado são modificadas e como eventos futuros podem ser gerados. Iremos assumir que nenhum servidor envia conteúdo duplicado a um cliente, isto é, todo bit enviado constitui uma parte única e exclusiva do conteúdo. Essa hipótese é baseada no fato de que os clientes podem solicitar blocos distintos de dados aos servidores, efetivamente evitando a transmissão de conteúdo duplicado. Dessa forma, a soma de cada coluna da matriz $A$ deve ser igual ao tamanho do arquivo $L$ ao final do download.

- Entrada do cliente $c_j$ no tempo $t$
  - Gera evento de conexão no tempo $t + \delta_3$ para o cliente com um servidor escolhido aleatoriamente
- Saída do cliente $c_j$ no tempo $t$
  - Gera eventos de desconexão para cada servidor conectado ao cliente $c_j$ no tempo $t + \delta_3$
  - Remove evento de mudança de fluxo ou conexão referentes a esse cliente que estejam previamente escalonados na lista de eventos
- Conexão/desconexão de cliente com servidor no tempo $t$
  - Atualiza a matriz $A$ para todos os clientes com todos os servidores usando a matriz de fluxo atual $F$ da seguinte forma: $a_{ij} \leftarrow a_{ij} + f_{ij} \times (t - t_{last})$, onde $t_{last}$ é o tempo em que ocorreu a alocação de fluxo atual $F$
  - Atualiza a matriz de conexões $K$ de acordo com o evento
  - Executa algoritmo de alocação de fluxo para determinar nova matriz $F$ e atualiza $t_{last} \leftarrow t$
  - Reescalona eventos de saída (término de download) para todos os clientes usando $A$ e $F$ atualizados
  - Reescalona eventos de mudança de fluxo no tempo $t + \delta_2$ para todos os clientes que não possuem um evento de mundaça de fluxo escalonado e que tiveram uma variação de fluxo de pelo menos $\Delta_f$ em ao menos uma conexão

- Timeout de conexão do servidor $s_i$ com cliente $c_j$ no tempo $t$
  - Atualiza a matriz $A$ para todos os clientes com todos os servidores usando a matriz de fluxo atual $F$ da seguite forma: $a_{ij} \leftarrow a_{ij} + f_{ij} \times (t - t_{last})$, onde $t_{last}$ é o tempo em que ocorreu a alocação de fluxo atual $F$
  - Atualiza a matriz de conexões $K$ colocando em 0 toda a linha do servidor
  - Executa algoritmo de alocação de fluxo para determinar nova matriz $F$ e atualiza $t_{last} \leftarrow t$
  - Reescalona eventos de saída para todos os clientes usando $A$ e $F$ atualizados
  - Reescalona eventos de mudança de fluxo no tempo $t + \delta_2$ para todos os clientes que não possuem um evento de mundaça de fluxo escalonado e que tiveram uma variação de fluxo de pelo menos $\Delta_f$ em ao menos uma conexão
- Mudança de fluxo de cliente $c_j$ no tempo $t$
  - Executa algoritmo de escolha de conexões para o cliente $c_j$
  - Caso a lista de conexões do cliente $c$ tenha sido alterada, gerar eventos de conexão/desconexão no tempo $t + \delta_3$ para as mudanças correspondentes

## 5.2 Resultados do simulador

### 5.2.1 Logs

O simulador gera arquivos de log tão detalhados quanto seja desejado, sendo possível analisar a criação e processamento de todos os eventos gerados, o fluxo alocado por cada servidor a cada instante de tempo, a escolha dos clientes ao realizar a seleção de servidores, a lista de eventos futuros, etc. A figura 5.1 ilustra dois exemplos de log de simulações com diferentes níveis de debug. Os eventos processados são listados seguindo o padrão

$$TEMPO\_DE\_EXECUCAO \quad TIPO\_DE\_EVENTO \quad ID \quad [ID]$$

onde o segundo identificador pode ou não existir. Em eventos processados entre clientes e servidores, como conexão e desconexão, sempre temos à esquerda o identificador do servidor e à direita o identificador do cliente. Assim uma linha de log "90.215141 $CONNECTION\_EVENT$ 14 63", significa um evento de conexão que ocorreu no tempo 90.215141 entre o servidor $s_{14}$ e o cliente $c_{63}$. Um evento referente somente a um cliente ou a um servidor, caso tenha somente um identificador, como "88.273369 $FLOW\_EVENT$ 95", representa um evento de mundança de fluxo no tempo 88.273369 do cliente $c_{95}$. Caso tenha dois identificadores, como "88.759384 $SERVER\_NUMBER\_CONNECTIONS$ 22 2", representa que o servidor $s_{22}$ estava conectado a 2 clientes no tempo 88.759384. Como podemos ver na Figura 5.1a, um evento de simulação pode dar origem a mais de uma messagem de log. No caso, um evento de desconexão gerou a mensagem de evento de desconexão e outras duas informando a quantidade atualizada de conexões que o cliente e servidor passaram a ter. A quantidade de mensagens de log que um evento pode gerar depende do nível de debug estabelecido. Entretanto, é importante perceber que apenas um evento pode ocorrer no mesmo instante de tempo de simulação. A Figura 5.1b contém um trecho da lista de eventos futuros de uma simulação, cada um com seu tempo único de execução.

```
88.273369 FLOW_EVENT 95
88.759384 CONNECTION_EVENT 22 13
88.759384 SERVER_NUMBER_CONNECTIONS 22 2
88.759384 CLIENT_NUMBER_CONNECTIONS 13 2
88.860527 FLOW_EVENT 49
89.148590 DISCONNECTION_EVENT 26 11
89.148590 SERVER_NUMBER_CONNECTIONS 26 0
89.148590 CLIENT_NUMBER_CONNECTIONS 11 1
90.215141 CONNECTION_EVENT 14 63
90.215141 SERVER_NUMBER_CONNECTIONS 14 2
```

(a) Mensagens de log: evento de desconexão gerou três mensagens de log no tempo 89.148590

```
QUEUE_CONNECTION_EVENT    | Time: 2431.985352 | Server id: 31 | Client id: 70
QUEUE_CONNECTION_EVENT    | Time: 2432.330078 | Server id: 82 | Client id: 62
QUEUE_FLOW_EVENT          | Time: 2432.599365 | Client id: 46
QUEUE_FLOW_EVENT          | Time: 2432.794922 | Client id: 74
QUEUE_CONNECTION_EVENT    | Time: 2433.645020 | Server id: 78 | Client id: 44
QUEUE_DISCONNECTION_EVENT | Time: 2434.898193 | Server id: 24 | Client id: 63
QUEUE_CONNECTION_EVENT    | Time: 2435.677979 | Server id: 78 | Client id: 7
QUEUE_FLOW_EVENT          | Time: 2435.695312 | Client id: 77
```

(b) Lista de eventos futuros em ordem crescente de execução

Figura 5.1: Dois exemplos de log

### 5.2.2 Medidas de Interesse

As medidas de interesse geradas pelo simulador são:

- $\kappa_j$ – Tempo de download do cliente $c_j$

  Seja $t_j^0$ o tempo de entrada do cliente $c_j$ no sistema e $t_j^T$ o tempo de saída, o valor de $\kappa_j$ é encontrado através da Equação (5.1).

  $$\kappa_j = t_j^T - t_j^0 \tag{5.1}$$

  A partir de $\kappa_j$ temos também $\kappa$, o tempo médio de download dos clientes dado por

  $$\kappa = \frac{\sum_{j=1}^{C} \kappa_j}{C} \tag{5.2}$$

- $\omega_j$ – Média temporal do número de conexões do cliente $c_j$

  Média ponderada no tempo dos valores assumidos por $m_j$ entre $t_j^0 \leq t \leq t_j^T$. Assim, seja $m_j(t)$ a função que descreve o valor de $m_j$ no tempo $t$, o

valor de $\omega_j$ é encontrado através da Equação (5.3).

$$\omega_j = \frac{\int_{t=t_j^0}^{t_j^T} m_j(t)\,\mathrm{d}t}{\kappa_j} \tag{5.3}$$

A partir de $\omega_j$ temos também $\omega_c$, a média temporal do número de conexões dos clientes dado por

$$\omega_c = \frac{\sum_{j=1}^{C} \omega_j}{C} \tag{5.4}$$

- $\omega_i$ − Média temporal do número de conexões do servidor $s_i$

  Média ponderada no tempo dos valores assumidos por $n_i$ entre $t_i^0 \leq t \leq t_i^T$, onde $t_i^0$ é o tempo de entrada do servidor $s_i$ no sistema e $t_i^T$ é o tempo de saída. Assim, seja $n_i(t)$ a função que descreve o valor de $n_i$ no tempo $t$, o valor de $\omega_i$ é encontrado através da Equação (5.5).

$$\omega_i = \frac{\int_{t=t_i^0}^{t_i^T} n_i(t)\,\mathrm{d}t}{t_i^T - t_i^0} \tag{5.5}$$

  A partir de $\omega_i$ temos também $\omega_s$, a média temporal do número de conexões dos servidores dado por

$$\omega_s = \frac{\sum_{i=1}^{S} \omega_i}{S} \tag{5.6}$$

- $\mu_j$ − Média temporal do fluxo do cliente $c_j$

  Seja $f_j(t)$ a soma do fluxo total alocado para o cliente $c_j$ no tempo $t$, isto é,

$$f_j(t) = \sum_{i=1}^{S} f_{ij}(t)$$

  onde $f_{ij}(t)$ é o fluxo alocado do servidor $s_i$ para o cliente $c_j$ no tempo $t$. O valor de $\mu_j$ é dado pela média ponderada no tempo dos valores assumidos por $f_j(t)$ entre $t_j^0 \leq t \leq t_j^T$, descrita na Equação 5.7.

$$\mu_j = \frac{\int_{t=t_j^0}^{t_j^T} f_j(t)\,\mathrm{d}t}{\kappa_j} \tag{5.7}$$

  A partir de $\mu_j$ temos também $\mu_c$, a média temporal do fluxo dos clientes dado por

$$\mu_c = \frac{\sum_{j=1}^{C} \mu_j}{C} \tag{5.8}$$

- $\mu_i$ − Média temporal do fluxo do servidor $s_i$

  Seja $f_i(t)$ a soma do fluxo total alocado pelo servidor $s_i$ no tempo $t$, isto é,

  $$f_i(t) = \sum_{j=1}^{C} f_{ij}(t)$$

  onde $f_{ij}(t)$ é o fluxo alocado do servidor $s_i$ para o cliente $c_j$ no tempo $t$. O valor de $\mu_i$ é dado pela média ponderada no tempo dos valores assumidos por $f_i(t)$ entre $t_i^0 \leq t \leq t_i^T$, descrita na Equação 5.9.

  $$\mu_i = \frac{\int_{t=t_i^0}^{t_i^T} f_i(t)\,\mathrm{d}t}{t_i^T - t_i^0} \tag{5.9}$$

  A partir de $\mu_i$ temos também $\mu_s$, a média temporal do fluxo dos servidores dado por

  $$\mu_s = \frac{\sum_{i=1}^{S} \mu_i}{S} \tag{5.10}$$

- $\mu'_j$ − Média temporal do fluxo médio por conexão do cliente $c_j$

  Seja $\phi_j(t)$ a média do fluxo alocado para o cliente $c_j$ no tempo $t$, isto é,

  $$\phi_j(t) = \frac{\sum_{i=1}^{S} f_{ij}(t)}{m_j}$$

  onde $f_{ij}(t)$ é o fluxo alocado do servidor $s_i$ para o cliente $c_j$ no tempo $t$. O valor de $\mu'_j$ é dado pela média ponderada no tempo de $\phi_j^t$ entre $t_j^0 \leq t \leq t_j^T$, descrita na Equação 5.11.

  $$\mu'_j = \frac{\int_{t=t_j^0}^{t_j^T} \phi_j(t)\,\mathrm{d}t}{\kappa_j} \tag{5.11}$$

  A partir de $\mu'_j$ temos também $\mu'_c$, a média temporal do fluxo dos clientes dado por

  $$\mu'_c = \frac{\sum_{j=1}^{C} \mu'_j}{C} \tag{5.12}$$

- $\mu'_i$ − Média temporal do fluxo médio por conexão do servidor $s_i$

  Seja $\phi_i(t)$ a média do fluxo alocado pelo servidor $s_i$ no tempo $t$, isto é,

  $$\phi_i(t) = \frac{\sum_{j=1}^{C} f_{ij}(t)}{n_i}$$

  onde $f_{ij}(t)$ é o fluxo alocado do servidor $s_i$ para o cliente $c_j$ no tempo $t$. O valor de $\mu'_i$ é dado pela média ponderada no tempo de $\phi_i^t$ entre

$t_i^0 \leq t \leq t_i^T$, descrita na Equação 5.13.

$$\mu_i' = \frac{\int_{t=t_i^0}^{t_i^T} \phi_i(t)\,\mathrm{d}t}{t_i^T - t_i^0} \tag{5.13}$$

A partir de $\mu_i'$ temos também $\mu_s'$, a média temporal do fluxo dos servidores dado por

$$\mu_s' = \frac{\sum_{i=1}^{S} \mu_i'}{S} \tag{5.14}$$

- $\nu_j'$ – Média temporal da variância do fluxo médio por conexão do cliente $c_j$

  Seja $\sigma_j^2(t)$ a variância da média do fluxo alocado para o cliente $c_j$ no tempo $t$, isto é,

  $$\sigma_j^2(t) = var(\phi_j(t))$$

  O valor de $\nu_j'$ é dado pela média ponderada no tempo de $\sigma_j^2(t)$ entre $t_j^0 \leq t \leq t_j^T$, descrita na Equação 5.15.

  $$\nu_j' = \frac{\int_{t=t_j^0}^{t_j^T} \sigma_j^2(t)\,\mathrm{d}t}{\kappa_j} \tag{5.15}$$

  A partir de $\nu_j'$ temos também $\nu_c'$, a média temporal da variância do fluxo médio por conexão dos clientes dado por

  $$\nu_c' = \frac{\sum_{j=1}^{C} \nu_j'}{C} \tag{5.16}$$

- $\nu_i'$ – Média temporal da variância do fluxo médio por conexão do servidor $s_i$

  Seja $\sigma_i^2(t)$ a variância da média do fluxo alocado pelo servidor $s_i$ no tempo $t$, isto é,

  $$\sigma_i^2(t) = var(\phi_i(t))$$

  O valor de $\nu_i'$ é dado pela média ponderada no tempo de $\sigma_i^2(t)$ entre $t_i^0 \leq t \leq t_i^T$, descrita na Equação 5.17.

  $$\nu_i' = \frac{\int_{t=t_i^0}^{t_i^T} \sigma_i^2(t)\,\mathrm{d}t}{t_i^T - t_i^0} \tag{5.17}$$

  A partir de $\nu_i'$ temos também $\nu_s'$, a média temporal da variância do fluxo médio por conexão dos servidores dado por

  $$\nu_s' = \frac{\sum_{i=1}^{S} \nu_i'}{S} \tag{5.18}$$

# Capítulo 6

# Avaliação de Desempenho

Neste Capítulo iremos avaliar o algoritmo BitLivery proposto no Capítulo 3. Na Seção 6.1 discutimos brevemente os valores esperados em um sistema centralizado no qual a vazão total do sistema é a maior possível, utilizando um modelo simplificado do sistema. Em seguida na Seção 6.2 apresentamos os cenários avaliados com o simulador descrito no Capítulo 5 e os resultados obtidos.

## 6.1 Avaliação Teórica

Considere um sistema com $S$ servidores com capacidade de upload $U$, $C$ clientes com capacidade de download $D$ e custo por conexão $P$, onde $F*$ é o fluxo ótimo que cada servidor deve alocar aos clientes conectados e $M*$ é o número ótimo de conexões que os clientes devem estabelecer para obter a maior taxa de download. Como buscamos o melhor caso, iremos considerar que o sistema é centralizado, e dessa forma podemos conectar os servidoes e os clientes de modo a distribuir toda a banda disponível. A análise está dividida em cinco casos listados a seguir.

### 6.1.1 Caso 1:

$$U < D \quad e \quad S = C$$

Neste caso, a banda conjunta que o sistema tem a oferecer é $U \times S$, que é menor do que a banda conjunta dos clientes $D \times C = D \times S$. Mais que isso, temos um servidor para cada cliente, dessa forma o maior fluxo que cada cliente pode receber está limitado pela a capacidade de upload dos servidores e pela sua capacidade efetiva de download, isto é $(D - P)$. Então temos:

$$\begin{aligned} M* &= 1 \\ F* &= \min\{U, (D - P)\} \end{aligned}$$

### 6.1.2 Caso 2:

$$U < D \quad e \quad S < C$$

Neste caso, a banda conjunta que o sistema tem a oferecer é menor do que a banda conjunta dos clientes. Para distribuir toda a capacidade do sistema entre os clientes, sempre haverá pelo menos um cliente conectado a dois servidores. O fluxo máximo que cada cliente pode receber está limitado pela banda total dos servidores dividido pelo número de clientes e pela capacidade efetiva de download do cliente, isto é $(D - 2 \times P)$.

$$\begin{aligned} M* &= 2 \\ F* &= min\left\{ \frac{S \times U}{C}, (D - 2 \times P) \right\} \end{aligned}$$

### 6.1.3 Caso 3:

$$U \geq D \quad e \quad S \geq C$$

Neste caso há pelo menos um servidor para cada cliente, e a capacidade de upload do servidor é maior que a capacidade de download do cliente, assim podemos afirmar que todos os clientes podem ser saturados com apenas uma conexão.

$$\begin{aligned} M* &= 1 \\ F* &= D - P \end{aligned}$$

### 6.1.4 Caso 4:

$$U < D \quad e \quad S > C$$

Neste caso, não sabemos se a banda total dos servidores $S \times U$ é menor ou maior que a demanda total dos clientes, $C \times D$. Entretanto sabemos que a função $F(M)$, que indica o fluxo total recebido por todos os clientes do sistema quando cada um está conectado a $M$ servidores, está limitada pelo mínimo entre

- A banda de upload total do sistema, $S \times U$
- A banda total efetiva dos clientes, $C \times (D - M \times P)$
- O total de banda máxima de $M$ servidores, isto é, o máximo que $M$ conexões podem dar a um cliente: $C \times M \times U$, ou seja, o número de clientes multiplicado pelo número de conexões que cada um estabelece, múltiplicado pelo máximo que um servidor poderia dedicar por conexão, sua capacidade $U$. Neste caso, consideramos que há servidores suficientes no sistema para que cada servidor possua apenas um cliente conectado a ele

Assim, temos que

$$F(M) = min\{S \times U \quad , \quad C \times (D - M \times P) \quad , \quad C \times M \times U\}$$

e nosso objetivo é maximizar essa função, encontrando o valor de $M$ que nos retorna $F*$. Então

$$F* = max_{1 \leq M \leq W \leq S}\{F(M)\}$$

.

Para solucionar esse problema, precisamos encontrar o ponto de interseção entre as equações. Seja $y$ valor do ponto de interseção entre as equações:

$$\begin{aligned} y &= C \times M \times U \\ y &= C \times (D - P \times M) \\ y &= S \times U \end{aligned}$$

Podemos igualar as equações

$$\begin{aligned} C \times M \times U &= C \times (D - P \times M) \\ C \times M \times U &= C \times D - C \times P \times M \\ C \times M \times U + C \times P \times M &= C \times D \\ M(C \times U + C \times P) &= C \times D \\ C \times M(U + P) &= C \times D \\ M(U + P) &= D \\ M &= \frac{D}{U + P} \end{aligned}$$

E

$$\begin{aligned} C \times M \times U &= S \times U \\ M &= \frac{S \times U}{C \times U} \\ M &= \frac{S}{C} \end{aligned}$$

Então,

$$\begin{aligned} M* &= min\left\{S, \frac{D}{U + P}, \frac{S}{C}\right\} \\ F* &= U \times M* \end{aligned}$$

### 6.1.5 Caso 5:

$$U \geq D \quad e \quad S < C$$

Neste caso, o número médio de clientes por servidor é dado por $\frac{C}{S}$. Assim, se $U \geq \lceil \frac{C}{S} \rceil \times (D - P)$, isto é, se a capacidade $U$ de um servidor é capaz de saturar todos os clientes conectados a ele:

$$\begin{aligned} M* &= 1 \\ F* &= D - P \end{aligned}$$

Senão, os clientes deverão se conectar a dois servidores, e a banda dos servidores será distribuida de acordo com o gargalo do sistema:

$$\begin{aligned} M* &= 2 \\ F* &= min\left\{(D - 2 \times P), \frac{S \times U}{M}\right\} \end{aligned}$$

Caso o gargalo seja a banda total dos servidores, cada cliente receberá $\frac{S \times U}{M}$, isto é, toda a banda do sistema será igualmente distribuída a todos, e nenhum cliente estará saturado. Caso contrário, é possível estabelecer uma configuração de forma que todos os clientes fiquem saturados, isto é, o fluxo total recebido por cada cliente será $D - 2 \times P$.

## 6.2 Avaliação Empírica

### 6.2.1 Cenário estático

No cenário estático iremos avaliar o sistema com entrada em rajada de clientes e servidores ao início da simulação, sem cancelamento de download ou queda de servidores. Para cada cenário, foram realizadas 200 simulações e o intervalo de confiança das médias encontradas é de 99%. Os parâmetros utilizados nos experimentos estão na Tabela 6.1.

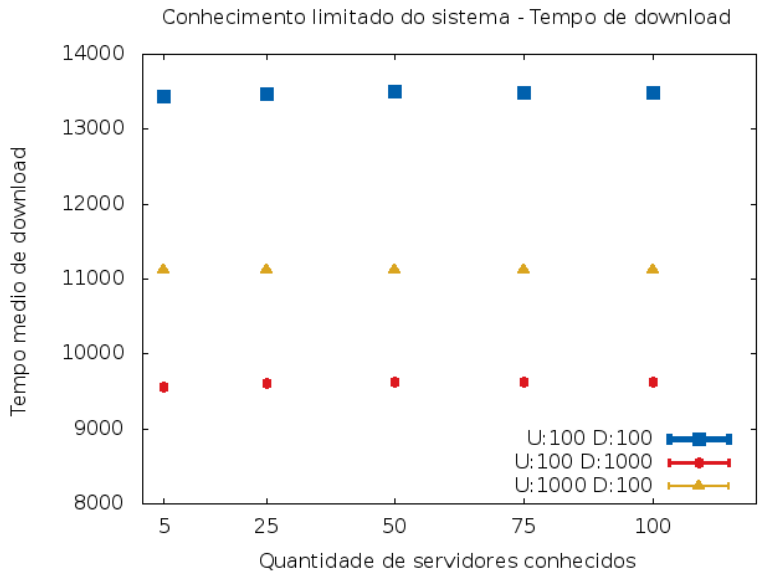
Tabela 6.1: Parâmetros das simulações

| Parâmetros | Valores |
|---|---|
| Tamanho do arquivo | 1000000 bits |
| Tempo máximo de simulação | $\infty$ |
| Taxa de entrada/saída do sistema | 1 |
| Taxa de mudança de fluxo | 200 |
| Taxa de conexão/desconexão | 10 |
| Limiar para mudança de fluxo | 10 |
| Custo por conexão | 12 |

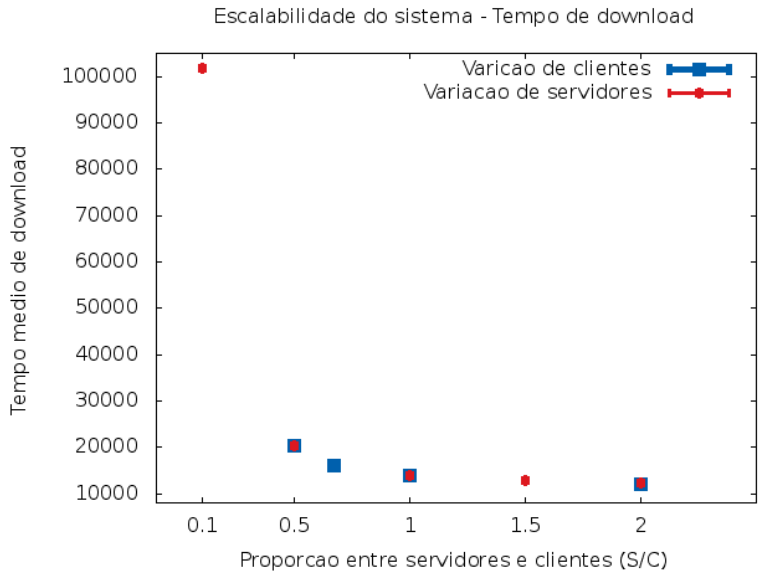
#### Tempo médio de download

A Figura 6.1a apresenta os valores obtidos para o tempo médio de download dos clientes no eixo vertical, e no eixo horizontal estão os valores utilizados nas simulações para o conhecimento do sistema, isto é, 5 servidores, 25, e assim por diante, num total de 100 servidores. O número total de clientes também é 100. As três curvas reprensentam três distintas configurações, sendo a azul um sistema onde clientes e servidores possuem a mesma capacidade (a demanda e a oferta são equivalentes) sendo $S = C = 100$, a curva amarela um sistema onde servidores possuem 10 vezes a capacidade dos clientes (a oferta é 10 vezes maior que a demanda) sendo $S = 1000$ e $C = 100$, e a curva vermelha um sistema onde clientes possuem 10 vezes a capacidade dos servidores (a demanda é 10 vezes maior que a oferta), assim $S = 100$ e $C = 1000$.

Os resultados mostram que o algoritmo mantém o desempenho mesmo limitando o conhecimento a um pequeno número de servidores, indo de acordo com os resultados esperados descritos em [1]. Em particular, nota-se inclusive uma pequena melhora de desempenho quando o conhecimento é menor (lembrando que a escolha do grupo de servidores conhecidos é feito de maneira aleatória), o que indica que os clientes se interferem menos ou encontram mais rápido a configuração ótima.




(a)




(b)

Figura 6.1: Variação do tempo médio de download

Em um cenário ideal, as curvas azul e amarela deveriam coincidir, pois a demanda dos clientes é a mesma. Pela curva amarela, podemos ver que o algoritmo alcança o resultado ótimo quando há bastante oferta no sistema, o que indica que os clientes são capazes de rapidamente se parear aos servidores. Já na curva azul, quando a demanda e a oferta são iguais, os clientes perdem desempenho tentando achar a configuração ideal, tendo uma piora de aproximadamente 20%. A curva vermelha indica a eficiência do algoritmo em distribuir os clientes e os servidores de forma que a oferta total do sistema seja aproveitada pelos clientes, já que o tempo de download foi o menor possível. Vale notar que a curva está abaixo de 10000, pois os clientes

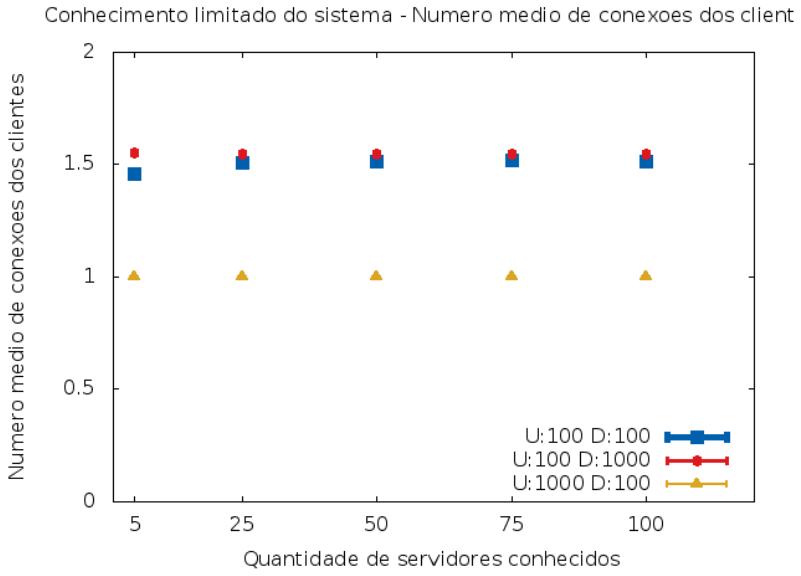
não entram exatamente ao mesmo tempo. Isso quer dizer que os clientes mais recentes são capazes de encontrar os servidores ociosos que atendiam a clientes que terminaram o download, ganhando maior vazão que clientes mais antigos, baixando assim a média do tempo de download.

Já a Figura 6.1b apresenta os valores obtidos para o tempo médio de download dos clientes no eixo vertical, e no eixo horizontal estão os valores utilizados nas simulações para a razão entre servidores e clientes ($S/C$), a fim de verificar a escalabilidade do algoritmo. Dessa forma, avaliamos os impactos para o aumento do número de servidores e o aumento do número de clientes separadamente. Os valores para $S$ e $C$, quando constantes, é igual a 100. A curva azul representa o aumento do número de clientes, onde cada ponto possui o valor de $C$ para 50, 100, 150 e 200, e a curva vermelha, cada ponto representa o resultado para o experimento com $S$ igual a 10, 50, 100, 150 e 200 servidores. Em todos os experimentos, clientes e servidores possuem a mesma capacidade, igual a 100.

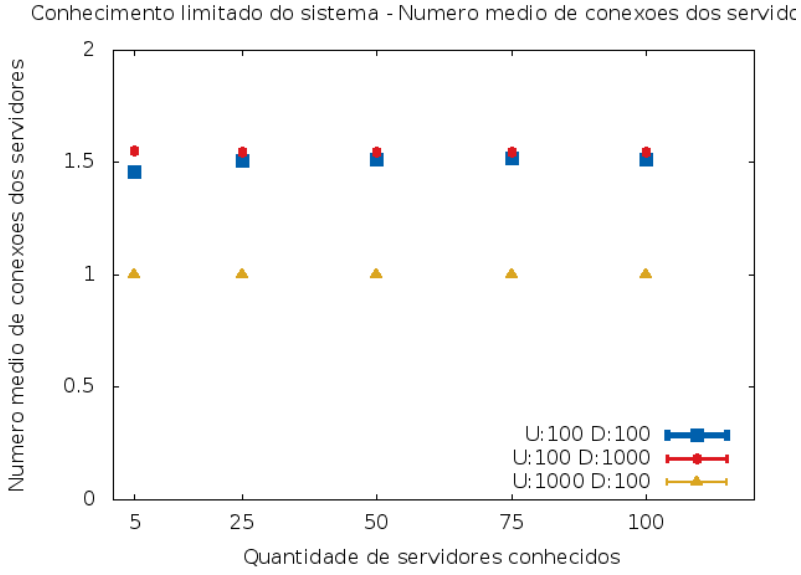
A partir dos dados obtidos, verificamos que o sistema se comporta de maneira equivalente tanto com o aumento do número de clientes como de servidores, pois os pontos azuis e vermelhos se sobrepõem. Da mesma forma como os resultados dos cenários anteriores indicam, podemos concluir que a partir do momento que temos a mesma oferta e demanda no sistema (isto é, a oferta do sistema deixa de ser o gargalo), o desempenho do tempo de download se aproxima bastante do tempo ótimo.

### Número médio de conexões

A Figura 6.2 apresenta os valores obtidos para o número médio conexões dos clientes e dos servidores no eixo vertical, e no eixo horizontal estão os valores utilizados nas simulações para o conhecimento do sistema. Os parâmetros $S$, $C$, $U$ e $D$, são os mesmos dos experimentos da Figura 6.1a. Mais uma vez, vemos que o algoritmo se comporta bem com conhecimento limitado. As curvas continuam representando os mesmos cenários em relação às capacidades dos clientes e servidores. A curva amarela confirma o indício de que os clientes e servidores rapidamente se pareiam, pois ambos possuem apenas uma conexão na média durante a simulação. Já as curvas azul e vermelha possuem valor 1,5. A curva azul indica o motivo pelo qual há perda de desempenho quando servidores e clientes possuem a mesma capacidade, que é a dificuldade em se parear, interferindo-se uns aos outros. O que provavelmente ocorre é que, quando dois clientes com apenas uma conexão se conectam a um mesmo servidor, acabam recebendo quase metade de sua demanda. Ao se conectarem a um segundo servidor, os clientes são capaz de saturar sua banda (pois os servidores são capazes de distribuir igualmente suas capacidades), desconectando-se de um servidor, de acordo com o segundo critério do BitLivery




(a)

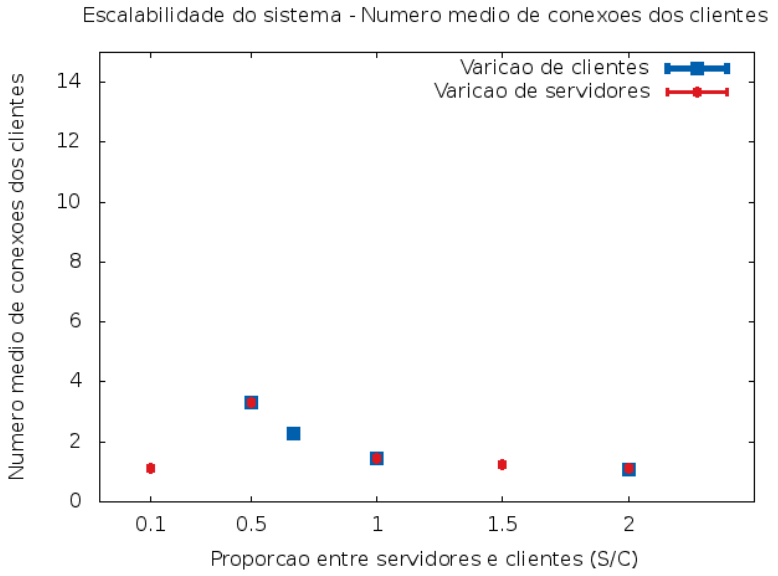



(b)

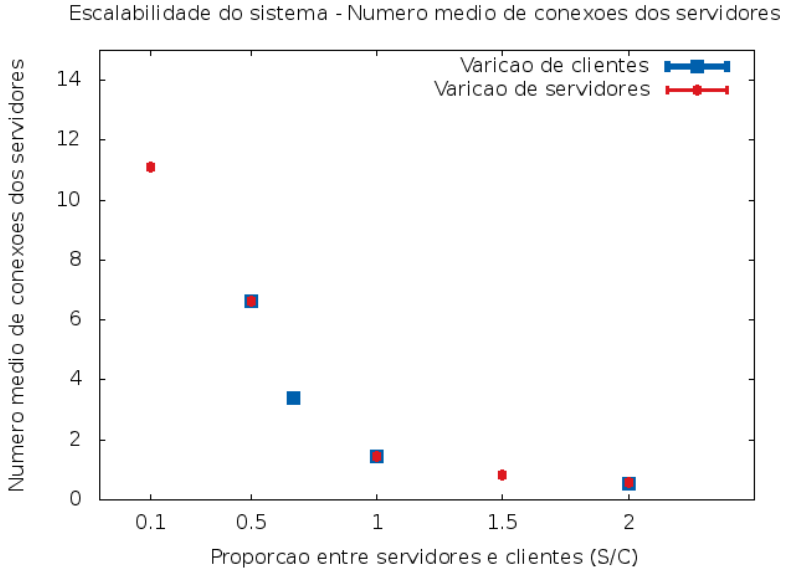
Figura 6.2: Variação do número médio de conexões ao limitar o conhecimento dos clientes sobre a rede

descrito na Seção 3.2. A perda de 20% de desempenho em relação ao caso ótimo é devido ao custo pela segunda conexão. Já a curva vermelha possui esse valor pois os clientes se conectam a um servidor inicialmente e, como sua banda não será saturada, um segundo servidor será adicionado. Entretanto, o valor recebido pelos servidores (no máximo 100) será menor que o custo por conexão (120), dessa forma o segundo servidor será descartado, voltando a apenas um servidor.

A Figura 6.3 apresenta os valores obtidos para o número médio de conexões de




(a)




(b)

Figura 6.3: Variação do número médio de conexões entre clientes e servidores ao aumentar ou diminuir a proporção entre eles

clientes e servidores no eixo vertical, e no eixo horizontal variamos a razão entre servidores e clientes, utilizando os mesmos parâmetros dos experimentos da Figura 6.1b. Pelos valores obtidos para as razões 1 e 1.5, podemos ver que os clientes e servidores mantém na média de 1 a 1.5 conexões tanto para o aumento do número de clientes como de servidores, seguindo a mesma dinâmica das curvas azul e vermelha da Figura 6.2. Para a razão igual a 2, clientes e servidores possuem a média de apenas uma conexão para os dois cenários, indicando que é possível rapidamente parear cada cliente com um servidor exclusivo. Para a razão 0.1, também temos apenas uma

conexão na média, pois durante a maior parte da simulação nenhum servidor conseguirá alocar uma vazão maior que o custo por conexão, desistimulando a abertura de mais de uma conexão. Por fim, para as razões 0.5 e 2/3, podemos ver um aumento do número de conexões dos clientes. Isso ocorre pois os clientes tentam saturar suas bandas abrindo mais conexões. Entretanto, a capacidade total do sistema $S \times U$ é menor que a maior demanda efetiva, $C \times (D - P)$, o que mantém a vazão média igual para os clientes a medida que novas conexões são abertas. O número médio de conexões dos clientes está limitado pelo valor de $m$ tal que $D - (P \times m) < \frac{S \times U}{C}$.

**Vazão média**

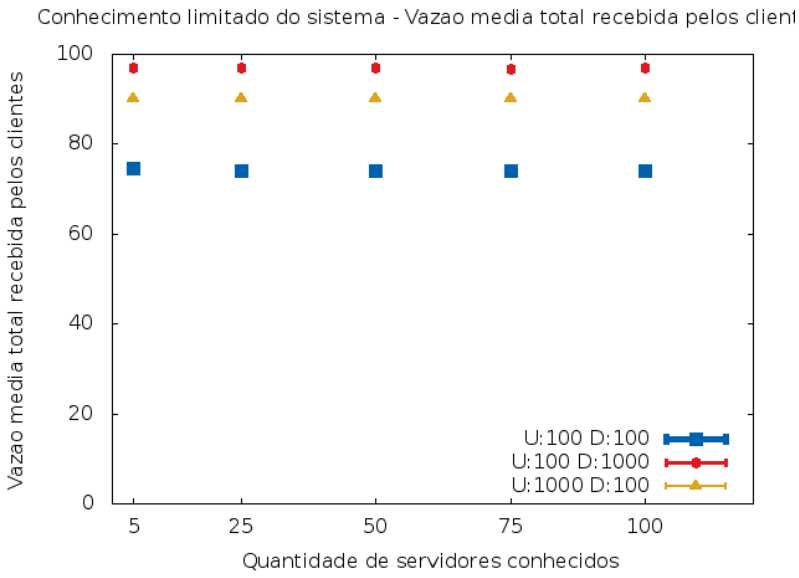
A Figura 6.4a apresenta os valores obtidos para a vazão média recebida pelos clientes no eixo vertical, no eixo horizontal estão os valores utilizados nas simulações para o conhecimento do sistema, os parâmetros $S$, $C$, $U$ e $D$ são os mesmos dos experimentos da Figura 6.1a assim como as cores das curvas continuam representando os mesmos cenários. Como o tempo de download indicava, a vazão média total recebida pelos clientes é praticamente a máxima possível, sendo aproximadamente $U$ para o caso em que o gargalo é a banda dos servidores (curva vermelha), $D - P$ no caso em que o gargalo é a banda dos clientes e o sistema tem mais oferta que demanda (curva amarela), e $D - 2 \times P$ no caso em que a oferta é igual à demanda (curva azul).

A Figura 6.4b, que representa a vazão média enviada pelos servidores, é equivalente à Figura 6.4a e podemos chegar às mesmas conclusões. Além disso, podemos ver que a variância é praticamente nula, mais um indicativo de que os recursos do sistema são bem distribuidos para os clientes.

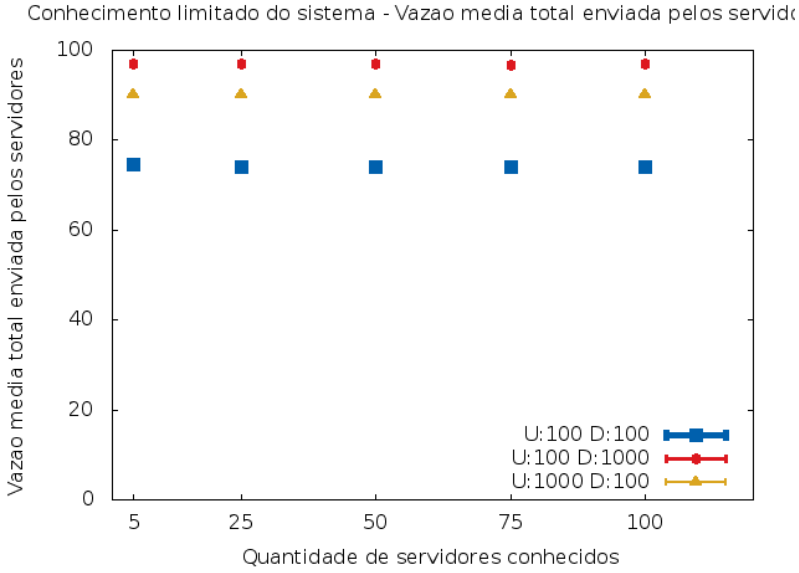
Por fim, os gráficos da Figura 6.5 apresentam no eixo vertical os valores obtidos para a vazão média total recebida pelos clientes na Figura 6.5a, e a a vazão média total enviada pelos servidores na Figura 6.5b. No eixo horizontal variamos a razão entre servidores e clientes, utilizando os mesmos parâmetros dos experimentos da Figura 6.1b. A partir dos resultados podemos concluir que o algoritmo distribui totalmente sua capacidade para os clientes até o momento em que o gargalo do sistema passa a ser a demanda dos clientes (que está restrita pela capacidade dos clientes e o custo por conexão) assim como nos casos anteriores.

### 6.2.2 Cenário Dinâmico

Na Figura 6.6, cada linha representa o fluxo agregado recebido por cada cliente a cada instante de tempo, em uma simulação com 100 servidores, 1000 clientes e conhecimento total da rede. A partir do instante 3000, metade dos servidores saem do sistema, retornando novamente a partir do instante 7000, em ambos os casos com



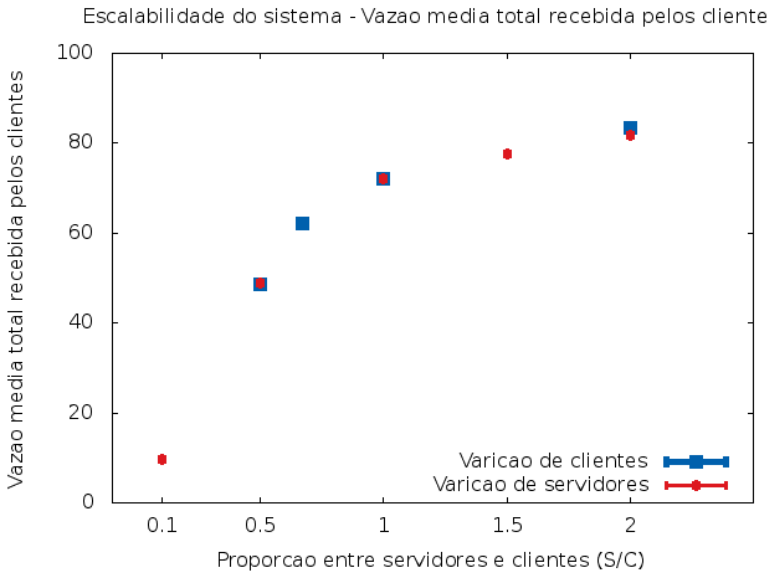

(a)



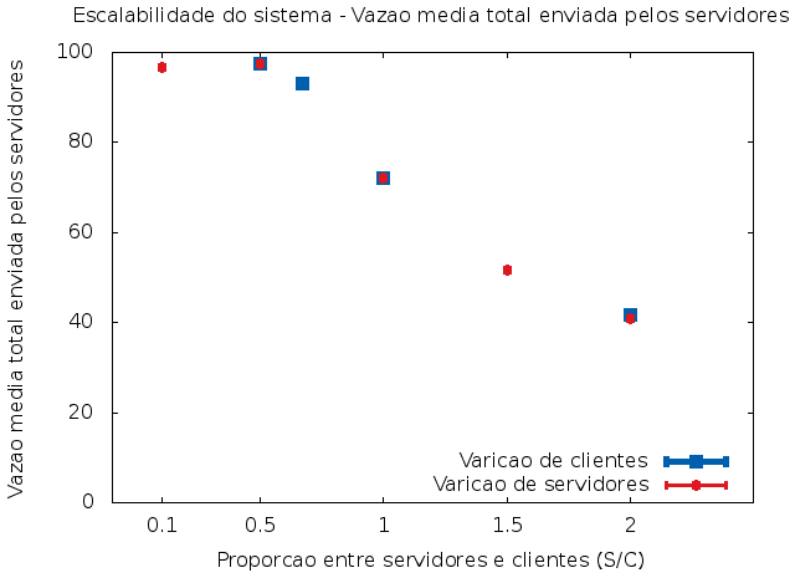

(b)

Figura 6.4: Variação da vazão média total ao limitar o conhecimento do sistema

distribuição exponencial dos eventos. Podemos concluir que os clientes rapidamente se adaptam às novas condições do sistema.




(a)




(b)

Figura 6.5: Variação da vazão média total ao aumentar ou diminuir a proporção entre servidores e clientes

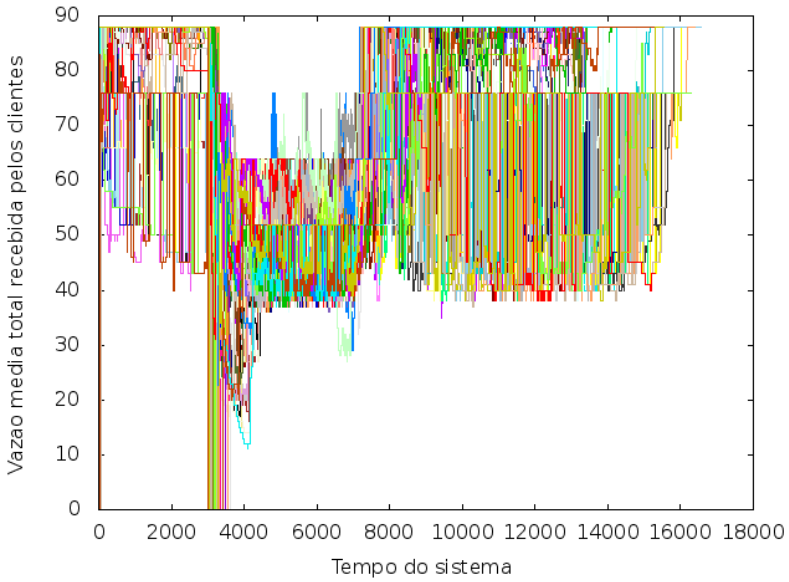



Figura 6.6: Fluxo agregado por cliente, onde cada linha representa um cliente

# Capítulo 7

# Conclusões e Perspectivas Futuras

Devido à crescente demanda por conteúdo digital na Internet, mecanismos de distribuição de conteúdo eficientes são cada vez mais necessários, motivando diversos estudos e implementações sob diferentes paradigmas abordagens. Nesta dissertação, propomos o algoritmo BitLivery baseado na arquitetura cliente-servidor, com múltiplos servidores que armazenam um mesmo conteúdo. Nossa proposta difere da maioria por implementar o algoritmo inteiramente nos clientes, deixando-os livres para buscar o melhor conjunto de servidores que atendam sua demanda, sem pré-estabelecer a quantos ou quais servidores esses irão requisitar o conteúdo.

O objetivo do algoritmo é maximizar a vazão recebida pelos servidores com o menor número de conexões, considerando que os servidores podem ficar sobrecarregados ou indisponíveis a qualquer momento. O algoritmo deve ser capaz de se adaptar rapidamente às mudanças da rede e à disponibilidade dos servidores, oferecendo robustez sem impactar a qualidade do download. Para avaliar o desempenho do algoritmo proposto, desenvolvemos um simulador de eventos discretos que modela precisamente seu funcionamento. A partir dos cenários avaliados, concluímos que o BitLivery é capaz de distribuir de forma eficiente os recursos do sistema.

## 7.1 Trabalhos futuros

### 7.1.1 Avaliação

Para melhor avaliar o algortimo proposto, gostaríamos de estudar os seguintes casos listados a seguir

- Avaliar o sistema no cenário onde clientes entram com uma distribuição de Poisson
- Avaliar cenários dinâmicos, com entrada e saída de clientes e servidores em diversos tempos da simulação, assim como desistência de download

- Avaliar o tempo de convergência do sistema em diversos cenários
- Avaliar o impacto global causado pela busca de pareamento ideal de um único cliente. Isto é, em média, quantos clientes e servidores são afetados após o estabelecimento ou encerramento de conexão entre um cliente e um servidor específicos?
- Comparar a solução BitLivery com sistemas P2P e CDNs
- Avaliação do algoritmo por meio de experimentos reais, em laboratório, implementando de fato o cliente e inserindo características típicas de redes, como atrasos, falhas e perda de pacotes, além dos protocolos de comunicação do pacote TCP/IP

### 7.1.2 Novas soluções

Os resutados mostram que o custo por conexão funciona de forma efetiva como um limitante para a abertura de conexões. Entretanto, um novo mecanismo poderia ser implementado no algoritmo criando um nível mais complexo de controle de utilização de banda, onde o cliente é capaz aumentar ou diminuir o custo por conexão dinamicamente ao longo de um download. Isso poderia ser realizado de maneira passiva, isto é, através de mensagens de controle entre servidores e clientes, por exemplo, indicando que os servidores estão sobrecarredados, estimulando assim o aumento do parâmetro. Entretanto, essa abordagem faria com que o BitLivery dependesse de variáveis fora do escopo do cliente. Dessa forma, como continuação deste trabalho propomos a implementação de uma solução que dependa somente do cliente, como por exemplo, a utilização de memória sobre a vazão total recebida.

# Referências Bibliográficas


[1] KEY, P., MASSOULIÉ, L., TOWSLEY, D. "Path selection and multipath congestion control", *Commun. ACM*, v. 54, n. 1, pp. 109–116, jan. 2011. ISSN: 0001-0782. doi: 10.1145/1866739.1866762.

[2] NETFLIX, I. . "Netflix". Disponível em: `https://www.netflix.com`, 2013. [Internet; acessado em 07-Junho].

[3] AMAZON.COM, I. . "Amazon.com". Disponível em: `http://www.amazon.com/`, 2013. [Internet; acessado em 07-Junho].

[4] MORAES, I. M., CAMPISTA, M. E. M., MOREIRA, M. D., et al. "Distribuiç ao de Vıdeo sobre Redes Par-a-Par: Arquiteturas, Mecanismos e Desafios", *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC*, v. 2008, pp. 115–171, 2008.

[5] 2013, A. T. "Akamai Technologies". Disponível em: `http://www.akamai.com/html/about/index.html`, 2013. [Internet; acessado em 07-Junho].

[6] KLACHQUIN, A. "BitLivey Repository". Disponível em: `https://github.com/alejandraklachquin/bitlivery`, 2013. [Internet; acessado em 15-Setembro].

[7] ZHANG-SHEN, R., MCKEOWN, N. "Designing a Predictable Internet Backbone with Valiant Load-Balancing". In: *in IWQoS, 2005*, pp. 178–192, 2005.

[8] WISCHIK, D., HANDLEY, M., BRAUN, M. B. "The Resource Pooling Principle", *ACM CCR*, 2008.

[9] RAICIU, C., WISCHIK, D., HANDLEY, M. "Practical congestion control for multipath transport protocols", *University College London, London/United Kingdom, Tech. Rep*, 2009.

[10] KHALILI, R., GAST, N., POPOVIC, M., et al. "MPTCP is not pareto-optimal: performance issues and a possible solution". In: *Proceedings of the 8th*

*international conference on Emerging networking experiments and technologies*, pp. 1–12. ACM, 2012.

[11] RAICIU, C., BARRÈ, S., PLUNTKE, C., et al. "Improving datacenter performance and robustness with multipath TCP". In: *SIGCOMM 2011, Toronto, Canada*, August 2011. See http://www.multipath-tcp.org for related work on MPTCP and the Linux kernel implementation used in this paper.

[12] JACOBSON, V., SMETTERS, D. K., THORNTON, J. D., et al. "Networking named content". In: *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pp. 1–12. ACM, 2009.

[13] KOPONEN, T., CHAWLA, M., CHUN, B.-G., et al. "A data-oriented (and beyond) network architecture". In: *ACM SIGCOMM Computer Communication Review*, v. 37, pp. 181–192. ACM, 2007.

[14] VISALA, K., LAGUTIN, D., TARKOMA, S. "LANES: an inter-domain data-oriented routing architecture". In: *Proceedings of the 2009 workshop on Re-architecting the internet*, pp. 55–60. ACM, 2009.

[15] DE BRITO, G. M., VELLOSO, P. B., MORAES, I. M. "Redes Orientadas a Conteúdo: Um Novo Paradigma para a Internet", *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC*, v. 2012, pp. 211–264, 2012.

[16] FERREIRA, F. M. *Sobre dois fenômenos em redes P2P do tipo BitTorrent*. Tese de Doutorado, Universidade Federal do Rio de Janeiro, 2011.

[17] KONRATH, M. A., BARCELLOS, M. P., MANSILHA, R. B. "Attacking a Swarm with a Band of Liars: evaluating the impact of attacks on BitTorrent". In: *The Seventh IEEE International Conference on Peer-to-Peer Computing (IEEE P2P 2007)*. IEEE, set. 2007.

[18] ZHANG, X., CHEN, S., SANDHU, R. "Enhancing data authenticity and integrity in P2P systems", *IEEE Internet Computing*, v. 9, pp. 18–25, 2005.

[19] SEKAR, V., DUFFIELD, N., VAN DER MERWE, K., et al. "LADS: Large-scale automated DDoS detection system". In: *Proc. of USENIX ATC*, v. 2006, 2006.

[20] WIKIPEDIA. "Unbounded nondeterminism — Wikipedia, the free encyclopedia". Disponível em: `http://en.wikipedia.org/wiki/Unbounded_nondeterminism`, 2013. [Internet; acessado em 07-Junho].

[21] GHODSI, A., ZAHARIA, M., HINDMAN, B., et al. "Dominant resource fairness: fair allocation of multiple resource types". In: *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, pp. 24–24, Berkeley, CA, USA, 2011. USENIX Association.

[22] WU, X., CHIEN, A. A. "A distributed algorithm for max-min fair bandwidth sharing", *University of California, San Diego, Tech. Rep*, 2005.

[23] FOONG, A. P., HUFF, T. R., HUM, H. H., et al. "TCP performance revisited". In: *Performance Analysis of Systems and Software, 2003. ISPASS. 2003 IEEE International Symposium on*, pp. 70–79. IEEE, 2003.

[24] NACE, D., PIÓRO, M. "Max-min fairness and its applications to routing and load-balancing in communication networks: A tutorial", *IEEE Communications Surveys and Tutorials*, v. 10, n. 1-4, pp. 5–17, 2008.

[25] ALLALOUF, M., SHAVITT, Y. "Centralized and distributed algorithms for routing and weighted max-min fair bandwidth allocation", *IEEE/ACM Trans. Netw.*, v. 16, n. 5, pp. 1015–1024, out. 2008. ISSN: 1063-6692. doi: 10.1109/TNET.2007.905605.