THOR: UM SISTEMA DE DISTRIBUIÇÃO DE VÍDEO PAR-A-PAR BASEADO NA TÉCNICA D1HT

Anderson Borges da Silva

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: Claudio Luis de Amorim

Rio de Janeiro
Setembro de 2011

THOR: UM SISTEMA DE DISTRIBUIÇÃO DE VÍDEO PAR-A-PAR BASEADO NA TÉCNICA D1HT

Anderson Borges da Silva

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. Claudio Luis de Amorim, Ph.D.

Prof. Felipe Maia Galvão França, Ph.D.

Prof. Orlando Gomes Loques Filho, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2011

Silva, Anderson Borges da

Thor: Um Sistema de Distribuição de Vídeo Par-a-Par baseado na Técnica D1HT/Anderson Borges da Silva. – Rio de Janeiro: UFRJ/COPPE, 2011.

XII, 64 p.: il.; 29, 7cm.

Orientador: Claudio Luis de Amorim

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2011.

Referências Bibliográficas: p. 52 – 55.

1. Tabelas Hash Distribuídas. 2. Distribuição de vídeo P2P. I. Amorim, Claudio Luis de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Em memória de Tales Vinícius Brandão Sanches

Agradecimentos

Em primeiro lugar agradeço a Deus por estar vivo e ter a oportunidade de realizar este trabalho.

Agradeço a minha esposa Evelin por me incentivar e dar suporte durantes esses longos anos de mestrado. Pelo carinho e por me compreender nos momentos em que não estive disponível para a família.

A minha mãe Terezinha por ter me criado como uma pessoa correta que sou. Por ter me incentivado nos estudos e por ter conseguido realizar a árdua tarefa de ser mãe e pai.

A minha sogra Rosiane e minha cunhada Melissa por serem as pessoas especiais que são, e por fazerem parte da minha vida.

Ao professor Edil Fernandes que sem a sua ajuda e incentivos talvez não tivesse retornado a percorrer essa grande jornada de conhecimento.

A toda a COPPE/UFRJ por tudo que aprendi nestes anos de estudos. Agradeço especialmente a toda a equipe do LCP, em particular a Diego Dutra, Gabriel Lima, Héberte Moraes e Lauro Whately por todo o suporte e ajuda.

Agradeço muito ao meu orientador Claudio Amorim, por me direcionar durante o meu trabalho, pela paciência e compreensão nos momentos em que não pude estar em dia com minhas tarefas de aluno.

Agradecimentos especiais ao meu cachorro Thor pelo carinho e pela inspiração. Também pelos momentos em que simplesmente esteve lá ao meu lado me fazendo companhia.

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

THOR: UM SISTEMA DE DISTRIBUIÇÃO DE VÍDEO PAR-A-PAR BASEADO NA TÉCNICA D1HT

Anderson Borges da Silva

Setembro/2011

Orientador: Claudio Luis de Amorim

Programa: Engenharia de Sistemas e Computação

Neste trabalho apresenta-se o Thor, um sistema *peer-to-peer* de distribuição de vídeo sob demanda baseada na técnica D1HT. A D1HT é uma Tabela *Hash* Distribuída que resolve as consultas em apenas um salto e com baixo custo de manutenção de suas tabelas. Utilizando-se desta técnica, desenvolveu-se uma camada chamada de D1HThor que é responsável pelo gerenciamento dos segmentos de dados que formam o vídeo distribuído. Esse gerenciamento é feito de forma a garantir que se algum membro da rede possuir um vídeo, este será sempre encontrado por todos os demais. O Thor utiliza-se então desta camada para realizar a busca e distribuição dos vídeos. Foram realizados experimentos com alterações de tamanho de buffer e taxa de chegada dos nós. Os resultados demonstram que a aplicação possui uma alta escalabilidade garantindo uma boa qualidade de serviço.

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

THOR: A PEER-TO-PEER DISTRIBUTION VIDEO SYSTEM BASED ON D1HT TECHNIQUE

Anderson Borges da Silva

September/2011

Advisor: Claudio Luis de Amorim

Department: Systems Engineering and Computer Science

This work presentes Thor, a peer-to-peer distribution video system based on D1HT technique. The D1HT is a Distributed Hash Table that solves the lookups in just one hop with low maintenance cost of its tables updates. Using this technique, it was developed a layer called D1HThor, which is responsible for managing data segments that form the distributed video. This management is done to ensure that if any member of the network has a video, it will always be found by everyone else. Thor uses this layer to perform the search and distribution of videos. Experiments were performed with changes in buffer size and arrival rate at nodes. Results show that the application has a high scalability, ensuring a good quality of service.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Nos últimos anos, aplicações de vídeo sobre IP têm atraído um grande número de usuários na Internet. A solução básica em geral, e em particular para *streaming* de vídeo é o modelo cliente/servidor, onde um cliente estabelece uma conexão com o servidor e o conteúdo do vídeo é enviado diretamente. Uma variação do modelo cliente/servidor são as Redes de Distribuição de Conteúdo[2] (CDN - *Content Distribution Network*). Nesta solução, o servidor de origem do vídeo primeiro envia o conteúdo do vídeo para um conjunto servidores de distribuição de conteúdo localizados estrategicamente próximos às bordas das redes. Desta forma, um cliente não pré-carrega mais o vídeo diretamente do servidor de origem, sendo direcionado para o servidor de distribuição de conteúdo mais próximo de sua localização. Esse mecanismo consegue efetivamente reduzir o atraso inicial e o tráfego na rede e também consegue servir a um número maior de usuários. O Youtube [3] é um exemplo de empresa que emprega CDNs para fornecer vídeos para o usuário final.

A escalabilidade torna-se um grande desafio para a solução de *streaming* baseada em servidores. Uma sessão de vídeo com boa qualidade requer uma grande banda de rede, e o fornecimento de banda, nos servidores de vídeo deve crescer proporcionalmente com o número de clientes. Isto faz com que a solução de *streaming* de vídeo baseada em servidores tenha um custo muito elevado para grandes audiências de centenas de milhares de pessoas.

As redes P2P[4] (*peer-to-peer*) surgiram como um novo paradigma para desenvolvimento de aplicações distribuídas de grande escala. A filosofia básica de uma rede P2P é de incentivar os usuários a atuarem tanto como clientes como servidores, esses usuários são então chamados de pares. Em uma rede P2P, um par não somente pré-carrega dados da rede utilizando sua banda de recepção, mas também os fornece para outros usuários utilizando sua banda de transmissão. A banda de transmissão dos usuários é eficientemente utilizada para reduzir a banda que poderia então ser utilizada pelo servidor para outros usuários. Desta forma, as soluções P2P apresentam-se também como uma alternativa de distribuição de conteúdo que

diminui os gastos com infraestrutura.

Diversas pesquisas vêm sendo desenvolvidas para melhorar o desempenho das redes P2P. Nesta direção estão as pesquisas que visam reduzir o tempo de busca de informações nessas redes. Vários estudos promissores envolvem a utilização das Tabelas *Hash* Distribuídas.

Redes como a Pastry [5], Chord [6] e CAN [7] já provaram seus méritos quando se trata de buscas envolvendo múltiplos saltos, que são redes onde a busca pela informação passa por mais de um nó. No entanto, novas pesquisas desenvolveram técnicas para resolver as consultas em somente um salto, onde cada nó tem conhecimento de todos os demais participantes da rede. Essas técnicas, em geral, exigem um alto consumo de banda de rede para manter as tabelas atualizadas.

Neste trabalho, utiliza-se a D1HT [8] como base. A D1HT é uma Tabela Hash Distribuída que resolve as consultas em apenas um salto com um baixo custo de manutenção de suas tabelas.

1.1 Contexto

Tipicamente, os dois modos de envio de vídeo para um cliente são *download* e *streaming*[9]. No modo *download*, o usuário recebe toda a informação requisitada antes de proceder com a exibição. Este modo tende a produzir uma latência inaceitável no início da exibição devido ao elevado tempo despendido na transferência total do vídeo.

Embora sistemas de distribuição de arquivos como o BitTorrent possam ser usados para baixar todo o conteúdo da mídia antes de ser reproduzido, tais sistemas não podem ser utilizados para visualização de *streaming* de vídeo, pois introduzem um grande atraso inicial[4], já que todo o conteúdo terá que ser baixado antes de iniciar sua reprodução.

Contrastando com esta estratégia, o modelo *streaming* ou pré-carregamento progressivo propõe que a exibição inicie tão logo uma porção mínima do vídeo chegue até o cliente. Desta forma, os demais trechos vão sendo recebidos, decodificados e exibidos em sequência, caracterizando-se uma aplicação de tempo real. Apesar de praticamente eliminar a latência no início de exibição, esta abordagem introduz maior complexidade na transmissão já que os trechos do vídeo devem chegar ao cliente a tempo de serem exibidos, de modo a não provocar interrupções na exibição.

1.2 Contribuições

As principais contribuições deste trabalho são:

- Implementação e avaliação preliminar de uma aplicação de distribuição de vídeo em modo *streaming* utilizando a técnica de D1HT;
- Desenvolvimento de disponibilização de uma biblioteca de gerenciamento de segmento de dados que pode ser utilizada livremente por outras aplicações de distribuição P2P de dados;
- Demonstração de que é possível implementar uma aplicação de distribuição de vídeos em uma rede P2P de larga escala, utilizando-se D1HT, que garanta a qualidade de serviço.

1.3 Organização da Dissertação

O conteúdo dessa dissertação está organizado da seguinte forma: No Capítulo 2 é apresentado o conhecimento básico dentro do escopo deste trabalho. No Capítulo 3 a aplicação de distribuição de vídeo é detalhada, apresentado sua arquitetura e os módulos que a compõe. O Capítulo 4 detalha o algoritmo utilizado na camada D1HThor que é utilizada pela aplicação como uma abstração para gerenciar os segmentos de um vídeo. No Capítulo 5 são apresentados e discutidos os resultados experimentais obtidos. Os trabalhos relacionados são apresentados no Capítulo 6 e o Capítulo 7 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Conceitos Básicos

Neste capítulo são apresentados os conceitos que serviram de base para a realização deste trabalho. Serão apresentadas as características do VoD, modelos cliente/servidor e modelo par-a-par (P2P). Neste capítulo serão também apresentadas as Tabelas *Hash* Distribuídas (DHT), e em particular será apresentada a D1HT.

2.1 Vídeo sob Demanda

Vídeo sob demanda, também chamado VoD (*Video-on-Demand*), permite que usuários assistam qualquer ponto de um vídeo a qualquer momento. Tais sistemas oferecem grande flexibilidade e conveniência para os usuários, haja vista que o cliente começa a assistir o conteúdo requisitado imediatamente, sendo a transmissão realizada no modo *streaming*. Demais disso, o cliente deve ser capaz de interagir com a exibição, podendo, entre outra operações, parar, avançar e recuar dentro da sequência do vídeo.

Todavia, implementar um sistema de VoD com todas estas funcionalidades, capaz de atender uma grande quantidade de clientes, não é tarefa trivial. Isto porque existe um compromisso entre a ocupação de largura de banda de rede do servidor, latência de início de exibição e o oferecimento de capacidade de interação ao usuário.

2.1.1 Categorias

De acordo como grau de interatividade que oferecem aos clientes, os sistemas tradicionais de VoD podem assim ser categorizados:

- **No VoD (NoVoD)**: trata-se de sistema baseado em *broadcast* similar ao oferecido pelas emissoras de televisão, onde o usuário fica restrito à grade de programação estática e, ainda, todos os usuários recebem o vídeo pelo mesmo canal;

- **Near VoD (NVoD)**: sistema onde múltiplos canais são usados para transmitir o mesmo vídeo com diferenças pré-definidas de início de transmissão, possibilitando ao usuário avançar ou recuar a exibição através da troca do canal; Cada canal geralmente atende a um grupo de usuários;
- **True VoD (TVoD)**: sistema em que o usuário ao solicitar o vídeo, é atendido prontamente, ou seja, sem ter que esperar por um instante de início de transmissão pré-definido. Nele os usuários recebem fluxos por canais diferentes quando as requisições são feitas em momentos diferentes;
- **Interactive VoD (IVoD)**: extensão de TVoD onde os usuários possuem controle individual da exibição, sendo capazes de empreender operações de VCR em qualquer ponto do vídeo; Cada usuário ocupa um canal do servidor.

Saliente-se que, quando da análise das características das categorias, é possível notar que conforme vão sendo oferecidos mais recursos ao usuário, maior é a demanda por recursos do servidor, sendo que NoVoD e IVoD representam os casos extremos. A NoVoD é a categoria que necessita menos recursos do servidor, podendo assim atender a inúmeros clientes, o que facilita sua escalabilidade, com o ponto negativo de não proporcionar ao usuário liberdade de horário e interatividade ao assistir o vídeo. Já IVoD permite que o usuário assista o vídeo em qualquer tempo e que interaja na exibição, ao custo de levar o servidor a saturação com poucos clientes.

Encontrar um nível intermediário entre as referidas abordagens é a questão chave para obter um sistema de VoD eficiente, de forma que a oferta de interatividade e baixa latência de início de exibição não comprometam a escalabilidade do sistema.

2.1.2 Abordagem Proativa X Reativa

Quanto à transmissão dos fluxos de vídeo, duas abordagens distintas podem ser adotadas na construção de sistemas de VoD, quais sejam: proativa e reativa [10].

Na abordagem proativa, as transmissões ocorrem em tempos pré-determinados, sem que um usuário realmente requisite seu início. Ao analisar as categorias acima expostas, nota-se que tanto NoVoD quanto NVoD adotam a abordagem proativa.

Em contrapartida, na abordagem reativa a transmissão se inicia apenas quando uma requisição proveniente de um cliente chegar ao servidor. Esta é a abordagem usada pelas categorias TVoD e IVoD anteriormente discriminadas.

2.1.3 Modelo Push X Pull

Além das possíveis abordagens de transmissão expostas acima, outra questão relevante em sistemas VoD é a forma como o vídeo é requisitado ao servidor, podendo

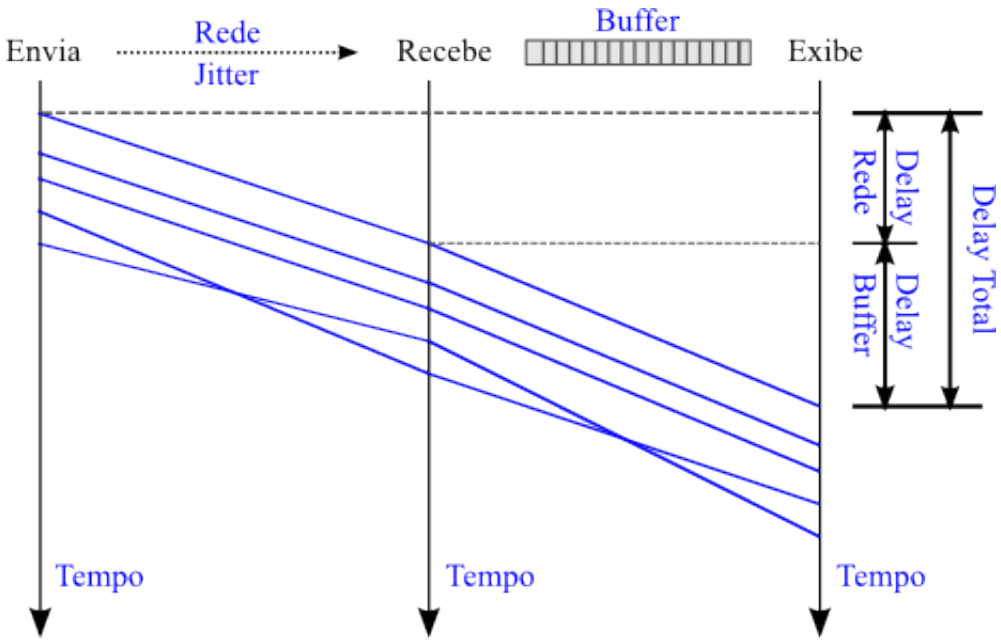

Figura 2.1: Jitter de rede.

ser feita através do modelo *push* ou *pull*. No modelo *push* (ou *Server-push*), para que a transmissão inicie o cliente sinaliza o servidor. Após esta solicitação inicial, o cliente recebe sequencialmente os trechos do vídeo, ficando sob responsabilidade do servidor temporizar o envio de forma que a exibição por parte do cliente proceda de forma contínua. Ao revés, no modelo *pull* (ou *Client-pull*) o cliente solicita trechos do vídeo conforme sua necessidade, repetindo este procedimento até o final do vídeo.

2.1.4 Componentes Básicos de Sistemas VoD

São três os componentes que formam um sistema VoD tradicional, quais sejam: servidor de vídeo, responsável por armazenar e transmitir mídia; clientes, que são aqueles que solicitam e consomem conteúdos; e rede de comunicação, que serve como meio de interconexão entre os dois anteriores. No que concerne ao cliente, algumas considerações adicionais merecem ser expostas.

Cliente

O cliente pode ser um computador ou um set-top-box, capaz de executar as tarefas básicas de receber, decodificar e exibir o vídeo. Tais tarefas são suficientes quando a transmissão do conteúdo for síncrona, ou seja, o trecho a ser decodificado chega sempre no instante exato. Para que isto ocorra, tanto o servidor quanto a rede devem oferecer alta qualidade de serviço (QoS). Na prática, onde redes de pacotes são largamente difundidas, torna-se necessário adotar um mecanismo auxiliar para garantir o correto funcionamento do sistema.

O comportamento da transmissão em uma rede de pacotes, representado pela Figura 2.1, demostra que ocorre um atraso entre o envio dos blocos e o seu recebimento, intrínseco ao tipo da rede, conhecido como *delay* da rede. Além disso, a rede tende a fazer com que os pacotes enviados com frequência constante sejam recebidos em intervalos heterogêneos, e, em certas ocasiões, em uma ordem diferente da estabelecida no envio, impondo uma variação no atraso denominada *jitter*. Como solução para tal obstáculo, é introduzido no cliente um *buffer* local. Desta forma, na hipótese de ocorrência de atraso (*delay*) na chegada de novos trechos do vídeo, o decodificador pode consumir os blocos já presentes no *buffer* até que se normalize a transmissão, evitando-se assim, congelamentos na exibição.

No caso oposto, onde trechos chegam antes do previsto, o *buffer* os acumula a fim de que o decodificador possa consumi-los no momento futuro apropriado, evitando o descarte do conteúdo transmitido. Contudo, para que o *buffer* suporte estas variações, a exibição só é iniciada quando uma quantidade mínima de blocos proporcional ao *jitter* da rede é atingida, sendo o intervalo compreendido entre o recebimento do primeiro bloco e o início da exibição denominado *delay* do buffer, que somado ao atraso do recebimento do primeiro bloco, definem a latência de início de exibição, aqui chamada de *delay* total. Por oportuno, é importante salientar que quanto maior o *jitter*, maior a capacidade necessária no *buffer*.

2.2 Modelo Cliente/Servidor x P2P

O problema da escalabilidade no acesso a um conjunto de dados está diretamente ligado a forma como este é provido. Os modelos tradicionais baseiam-se no paradigma cliente/servidor, no qual existe um cliente que deseja uma determinada informação e um servidor responsável por armazená-la e transmiti-la quando for requisitado. Nessa abordagem, o número de clientes simultaneamente atendidos, fica limitado pelos diversos recursos finitos existentes no caminho crítico a ser percorrido pelos dados entre o local onde estão armazenados no servidor até o cliente que solicitou a informação.

Dependendo da aplicação, os próprios clientes podem compartilhar seus dados com os demais, transformando-se em possíveis provedores de conteúdo. Esta forma de troca de conteúdo estabelecida entre clientes é demonimada de *peer-to-peer* (P2P), onde o desempenho tende a crescer com a chegada de novos clientes, visto que agregam novos recursos ao sistema.

Para descobrir em quais clientes os dados desejados estão armazenados, torna-se necessário oferecer um serviço de busca apropriado, seja de forma centralizada ou distribuída. Neste trabalho utiliza-se a forma distribuída de buscas de informação utilizando-se DHT. Em análise do conteúdo, é possível aferir que a Figura 2.2 a)

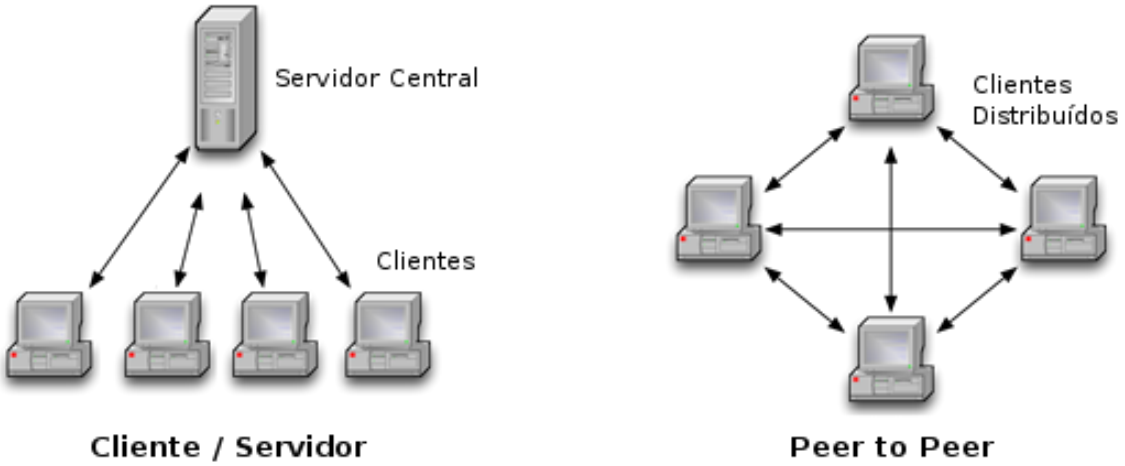

Figura 2.2: Modelos de arquiteturas Cliente/Servidor e Peer-to-Peer.

reflete um sistema Cliente/Servidor tradicional, em quanto que a Figura 2.2 b) representa um sistema P2P.

2.3 Tabelas *Hash* Distribuídas

Tabelas *hash* distribuídas (em inglês, DHT - *Distributed Hash Table*) fornecem uma solução prática e escalável para armazenar, localizar e recuperar informações que estejam amplamente espalhadas em um ambiente distribuído de grande escala. Por essa razão, DHTs foram propostas como base para uma variedade de aplicações par-a-par, mostrando a grande aceitabilidade da DHT como uma ferramenta distribuída útil.

Sistemas DHT são implementados com as mesmas facilidades de uma tabela *hash* convencional, onde as chaves (informações) são distribuídas entre os nós participantes. Para rotear uma busca por uma dada chave a partir de nó de origem até o nó que contém a chave desejada, as DHTs implementam uma Rede Sobreposta (*Overlay Network*) com informações de roteamento armazenadas em cada nó participante. A menos que uma DHT seja grande o suficiente para armazenar os endereços IPs de todos os nós participantes, o roteamento de uma busca acabará requerendo múltiplos saltos até encontrar a chave desejada. Ou seja, a busca terá uma alta latência, visto que passará por um número de nós antes de atingir o nó destino.

Enquanto grandes DHTs permitem buscas rápidas, elas também requerem uma alta utilização de banda a fim de manter-se atualizadas conforme os nós entram e saem do sistema, especialmente em sistemas muito dinâmicos, onde existe uma alta taxa de entrada e saída de nós. Como resultado, existe um compromisso entre diminuir a latência das buscas (número de saltos) e diminuir utilização de banda

(para manter as informações de roteamento atualizadas).

A maioria das DHTs propostas resolvem as buscas com múltiplos saltos com o intuito de minimizar o tráfego de manutenção (para atualização das informações de roteamento). Nesta categoria, enquadram-se as seguintes redes: Chord [6] , CAN [7], Pastry [5] e Tapestry [11].

Para resolver buscas com somente um salto, a DHT de cada participante conterá a informações sobre todos os demais pares. Com isso, a latência é reduzida, com o custo de aumentar o tráfego para atualizar a informações de roteamento. Nesse cenário, diversos algoritmos têm sido propostos com o objetivo de diminuir o tráfego necessário para atualização das tabelas, como o OneHop [12], 1h-Calot [13] e o D1HT [14] que será descrito a seguir.

Recentes estudos [15] mostraram que em alguns casos, DHTs com apenas um salto podem gerar menos tráfego que as de múltiplos saltos, mesmo para sistemas dinâmicos. D1HT foi apresentada em [8] como uma solução de um salto capaz de: 1) garantir que grande fração das buscas são resolvidas somente em um salto (99%); 2) requer um baixo consumo de banda; 3) oferece bom balanceamento do tráfego de manutenção entre os nós; e 4) se adapta às modificações em sistemas dinâmicos.

2.3.1 D1HT

D1HT é uma uma Tabela *Hash* Distribuída que visa responder as buscas com somente um salto e com baixa latência. Para reduzir o custo de atualização das tabelas de roteamento, D1HT utiliza o algoritmo EDRA que garante a atualização das tabelas, com um baixo custo.

Definição

Um sistema D1HT[1] é composto de um conjunto $\mathbb{D}$ de n pares e associa itens (ou chaves) aos pares com uso das técnicas de *consistent hashing* [16], onde tanto as chaves quanto os pares são convertidos para identificadores inteiros (IDs) no mesmo intervalo $[0 : N]$, com $N \gg n$. Tipicamente, o ID de uma chave é obtido aplicando-se a função criptográfica SHA-1 [17] ao valor da chave, o ID de um par é obtido aplicando-se a função SHA-1 ao seu endereço IP, e $N = 2^{160} - 1$. Para simplificar, refere-se aos IDs dos pares e das chaves como se fossem respectivamente os próprios pares e chaves.

Os diversos IDs dos pares e chaves de um sistema D1HT são dispostos em um mesmo anel, onde o identificador 0 sucede o identificador N, e o sucessor e o predecessor de um identificador i são respectivamente os seus vizinhos no sentido horário e anti-horário. Cada chave é associada ao seu sucessor e pode, opcionalmente, ser replicada nos k pares seguintes no anel, sendo que o valor do parâmetro k pode ser

ajustado pela aplicação.

Cada par em um sistema D1HT mantém uma tabela de roteamento com os endereços IP de todos os participantes do sistema. Desta maneira, qualquer consulta é trivialmente resolvida com apenas um salto, desde que esta tabela local de roteamento esteja atualizada. Caso um par p não tome conhecimento de um evento causado pela adesão ou partida de algum outro par do sistema, a sua tabela de roteamento ficará desatualizada, de maneira que p poderá encaminhar uma consulta a um par errado ou a um par que já tenha saído do sistema. Desde que cada par tenha conhecimento do seu verdadeiro sucessor [6], em ambos os casos a consulta deverá ser resolvida após algumas tentativas, mas irá demorar mais do que inicialmente esperado. Uma vez que esta consulta será resolvida com perda de desempenho, mas não irá falhar, esse tipo de ocorrência é chamada de falha de roteamento, ao invés de um erro de consulta.

As falhas de roteamento devem ser minimizadas para garantir o desempenho do sistema. Para isso, o D1HT utiliza de um algoritmo chamado de EDRA que dissemina a ocorrência dos eventos sem exigir grandes demandas de rede ou causar desbalanceamento de carga entre os pares.

O armazenamento da tabela de roteamento requer memória local dos diversos pares do sistema. Em D1HT estes requerimentos são minimizados com o uso de uma tabela *hash* local para armazenamento dos endereços IP, já que o espaço de identificadores (IDs) é esparsamente ocupado (uma vez que $N \gg n$). O índice desta tabela *hash* é formado pelos próprios identificadores dos pares, evitando assim a necessidade de armazenamento destes IDs. Desta maneira, cada tabela de roteamento irá requerer $6n$ octetos para armazenar os endereços IPv4 (incluindo porta) dos n pares participantes do sistema, além de uma área adicional para tratamento das colisões. Assim, para aplicações em ambientes corporativos, como CPDs HPC ou ISP, a tabela de roteamento de cada par irá requerer no máximo algumas centenas de milhares de octetos. Para um vasto sistema com um milhão de pares, cada tabela de roteamento irá consumir cerca de seis milhões de octetos, o que é simplesmente desprezível para computadores domésticos, e plenamente aceitável mesmo para pequenos dispositivos móveis individuais modernos, como telefones celulares, tocadores de música e máquinas fotográficas.

Para aderir a um sistema D1HT, o novo par deverá conhecer ao menos um outro par que já faça parte do sistema, e então seguir um protocolo de adesão (*joining protocol*). Os detalhes do protocolo de adesão devem ser definidos pela implementação, e devem atender aos seguintes requisitos: (i) A adesão de um par só poderá ser considerada como completa, e então iniciada a sua disseminação, quando o novo par tiver recebido a tabela de roteamento completa e todas as suas chaves; (ii) A disseminação da adesão de um par deverá ser feita segundo o algoritmo EDRA;

(iii) O sucessor (ou predecessor) do novo par deverá garantir que este receba todas as notificações de eventos que ocorram enquanto a sua adesão está sendo disseminada; (iv) O protocolo de adesão deverá ser capaz de receber extensões como, por exemplo, o mecanismo de Quarentena.

Protoloco de adesão

O mecanismo de adesão de novos pares utilizado nesse trabalho foi o mesmo utilizado na implementação original da D1HT apresentado em [1]. O protocolo desenvolvido inspirou-se no empregado em Chord [6] com as devidas modificações para atender os requisitos do D1HT.

A adesão de um par é sempre tratada por seu sucessor, que lhe transfere toda a tabela de roteamento e então inicia a propagação da sua adesão segundo EDRA. Para assegurar que o novo par irá receber todas as notificações de eventos enquanto a sua adesão não for conhecida por todo o sistema, o seu sucessor irá lhe enviar todos os eventos que receba até que o novo par receba ao menos uma mensagem de manutenção com cada TTL.

Manutenção das Tabelas de Roteamento

Como cada par em um sistema D1HT deve saber o endereço IP de todos os outros pares do sistema, qualquer evento deverá ser conhecido por todos os pares de uma maneira rápida, de modo a minimizar a existência de endereços desatualizados nas tabelas de roteamento. Por outro lado, tendo-se como objetivo o suporte a sistemas grandes e dinâmicos, deve-se evitar mecanismos rápidos porém ingênuos (p.ex. broadcast) de propagação das informações sobre os eventos, de maneira a evitar sobrecargas na rede. Deste modo, a detecção e propagação dos eventos impõem três importantes desafios para D1HT: (a) minimizar as demandas de rede, (b) prover bom balanceamento de carga, e (c) garantir um limite superior para a fração de entradas desatualizadas nas tabelas de roteamento (de modo a permitir que a grande maioria das consultas seja resolvida com um único salto). Para atender estes desafios, foi desenvolvido o algoritmo EDRA, que é capaz de notificar qualquer evento a todos os pares do sistema em tempo logarítmico, tem ótimas características de balanceamento de carga, e baixa demanda de rede se comparado com outras DHTs de um salto. Adicionalmente, EDRA é capaz de se adaptar a mudanças no comportamento do sistema de maneira a continuamente satisfazer a um limite pré-definido de falhas de roteamento, usando uma arquitetura puramente P2P e auto-reorganizável.

Capítulo 3

Thor: Uma infraestrutura de distribuição de vídeos baseado na técnica D1HT

Neste capítulo será apresentado o sistema Thor com sua arquitetura e funcionamento. Serão também detalhados os passos necessários para a distribuição e busca das informações.

3.1 Visão Geral

Redes DHTs vêm sendo propostas como substrato para uma vasta gama de aplicações distribuídas, incluindo grades computacionais [18] [19] [20], resolução de nomes [21] [22], sistemas de armazenamento [23] [5], vídeo sob demanda e streaming [24] [25], soluções de backup [26], bibliotecas de referências bibliográficas [27], busca na Internet [28], telefonia sobre IP (VoIP) [29] [30], jogos na Internet [31], banco de dados [32] e outras.

Dentre as diversas aplicações de uma rede DHT, neste trabalho foi desenvolvida uma aplicação de localização e busca de vídeo e seus segmentos de dados. Numa rede de distribuição de vídeos assume-se que a rede é formada por um acervo de vídeos e um grande número de usuários conectados ou membros da rede.

No sistema Thor, um vídeo é dividido em um determinado número de segmentos. Esses segmentos são as unidades básicas que serão compartilhadas entre os membros da rede. Cada segmento é registrado na rede como uma entidade isolada, e o sistema se encarrega de agrupá-los novamente para formar o conteúdo desejado.

Esta mesma tecnologia poderia ser utilizada para compartilhar outros tipos de arquivos. No entanto, quando escolhe-se trabalhar com vídeos, têm-se algumas restrições que devem ser atendidas. Primeiro, os blocos devem ser buscados em

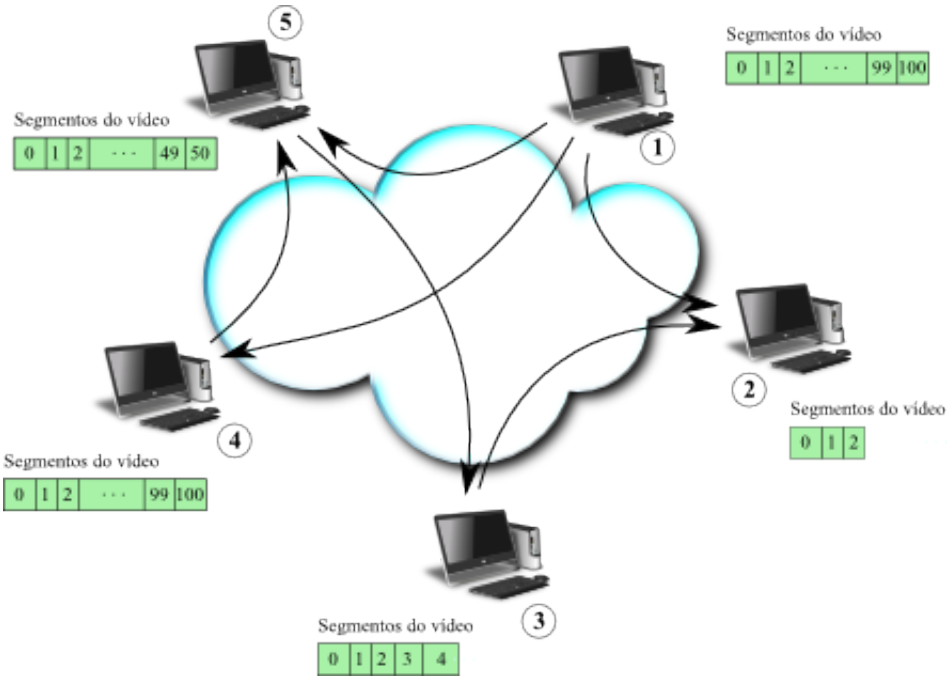

Figura 3.1: Visão geral: O nó 1 é o nó inicial contendo o arquivo original. O nó 2 contém até o segmento 2 e está recebendo o próximo segmentos dos nós 1 e 3. O nó 3 contém até o segmento 4, está recebendo o próximo conteúdo do nó 5 e já está fornecendo o segmento ao nó 2. O nó 4 completou o arquivo. O nó 5 contém até o segmento 50 está recebendo segmentos de 1 e 4 e está fornecendo o segmento ao nó 3.

sequência, pois a busca de um bloco que não será exibido naquele momento pode acarretar a perda de um bloco que efetivamente seria exibido, e isso degradaria a qualidade do vídeo exibido. Os blocos também devem chegar aos clientes dentro de um intervalo inferior ao tempo de sua exibição, caso contrário, as aplicações clientes sofrerão atrasos, e a exibição dos vídeos não se dará de forma contínua.

Na Figura 3.1 tem-se uma configuração de uma rede de distribuição de vídeos. Pode-se observar que cada nó está em um momento distinto de exibição. Para um nó iniciar o compartilhamento, basta que o segmento de vídeo desejado esteja disponível num nó. Não é necessário aguardar todo o conteúdo de um vídeo estar disponível para iniciar o compartilhamento, reduzindo assim o tempo inicial de exibição. Após o recebimento de todos os segmentos por um nó, ele possuirá uma cópia completa do arquivo original, pois todo segmento recebido é armazenado e exibido.

3.2 Arquitetura

Devido à complexidade de desenvolvimento de aplicações distribuídas, adotou-se uma abordagem de camadas de software, a fim de minimizar o esforço de manutenção e de tornar as camadas reutilizáveis por outras aplicações. Na Figura 3.2 podemos

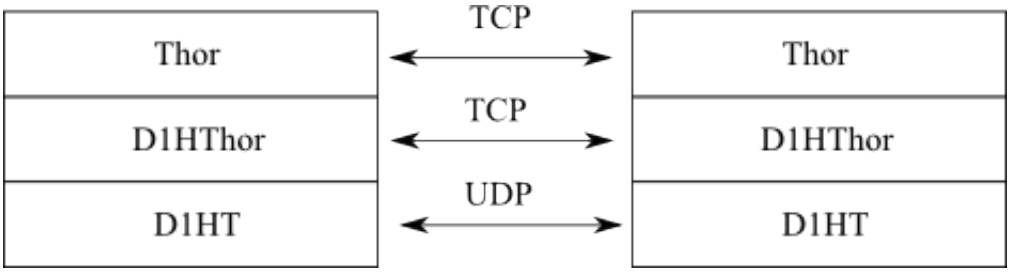

Figura 3.2: Arquitetura da Aplicação

ver as três camadas que formam o sistema desenvolvido. Na camada mais baixa temos a implementação do D1HT disponível que foi utilizada neste trabalho. Na camada intermediária está a implementação do D1HThor e somente por último está a camada de aplicação P2P de distribuição de vídeos.

A camada D1HThor permite que a aplicação seja responsável somente pelo registro e busca de chaves dos segmentos de um arquivo. Todo o gerenciamento que é necessário para o deslocamento das chaves é feito pela D1HThor. Esse gerenciamento é necessário, visto que, com a entrada e saída aleatória dos nós, as chaves dos segmentos devem ser migradas e as tabelas atualizadas.

Por fim, a camada D1HT é onde está implementado o algoritmo da tabela hash distribuída. Vale salientar que, como a aplicação não se comunica diretamente com a camada D1HT, a implementação da DHT pode ser modificada sem impacto na camada de aplicação, sendo necessário somente ajustes na camada intermediária D1HThor.

Conforme a arquitetura apresentada, a camada de aplicação comunica-se com os demais pares da rede utilizando o protocolo TCP. A camada D1HThor também utiliza o TCP para troca de informações e gerenciamento de suas tabelas. Já a D1HT manteve a sua implementação original e faz uso do protocolo UDP para disseminação dos eventos.

3.3 Arquivos

Para a disponibilização dos arquivos de vídeo na Rede de Cobertura, são gerados metadados com as informações necessárias a criação e localização dos segmentos que compõem o arquivo original. As informações necessárias são: a) a chave, que é o *hash* do nome do arquivo; b) o tamanho total do arquivo; e c) tamanho do segmento.

Com a informação contida no metadado, o arquivo pode então ser dividido nos seus respectivos segmentos. Para que todos os pares da rede tenham acesso aos dados, tanto o metadado como os segmentos de dados são registrados na Rede de Cobertura utilizando-se a função $publish(key)$ da interface de aplicação do D1HThor.

Na implementação utilizada, somente os segmentos de dados são entidades físicas

que podem ser encontradas em arquivo. As informações de metadado são geradas quando um arquivo é publicado e se mantém em memória enquanto o arquivo estiver disponível. Assim que todos os nós que possuem um determinado arquivo saírem da rede, o conteúdo do seu metadado também será retirado da rede, visto que é através da localização do metadado que o conteúdo de um arquivo é encontrado.

Quando o arquivo de vídeo, ou seja, os metadados e segmentos, é publicado na rede, nenhuma modificação é feita no arquivo original. Note que a divisão dos segmentos é apenas lógica, pois já se tornará real conforme esses segmentos são enviados pela rede, até constituir um arquivo integral no nó que solicitou o arquivo.

3.3.1 Publicação de Arquivos

A camada de aplicação é responsável por listar os arquivos nos diretórios, dividir em segmento de dados, registrar os segmentos na Rede de Cobertura D1HT através da interface de aplicação da camada D1HThor. A aplicação é responsável por dar significado aos metadados e segmentos. Pois somente esta sabe reunir os segmentos de forma a reconstituir o arquivo de vídeo original. Desta forma, os metadados e segmentos registrados não possuem nenhum significado para a camada D1HThor. Para esta camada, são apenas chaves e valores. Para a publicação dos arquivos, cada aplicação gera os metadados dos arquivos disponibilizados e então realiza sua publicação na rede destes metadados e dos segmentos que compõem os arquivos.

3.3.2 Busca por Arquivo

Quando uma busca é realizada, o nome do arquivo é convertido em sua correspondente chave, usando-se SHA-1. O conteúdo desta chave será buscado na rede através da função de interface *IP = get(key)* fornecida pela interface de aplicação da D1HThor. Esta função retorna o IP do nó que possui o metadado do aquivo procurado. Uma vez descoberto o IP, o nó solicitante deve conectar-se ao IP e solicitar o conteúdo associado à esta chave (que na primeira busca será sempre um metadado).

Desta forma, quando o par que iniciou a busca possuir o metadado, ele será capaz de calcular as chaves de todos os segmentos que formam o arquivo. A chave de um segmento será formada pelo nome original do arquivo adicionando-se um sequencial.

Com as informações contidas no metadado, pode-se calcular a quantidade de segmentos e as suas respectivas chaves. As chaves dos segmentos serão estão buscadas na Rede de Cobertura utilizando-se também a funcão *IP = get(key)*, que desta vez retorna o IP do nó que possui o conteúdo do segmento procurado. A aplicação então conecta-se ao IP e solicita a transferência do segmento desejado.

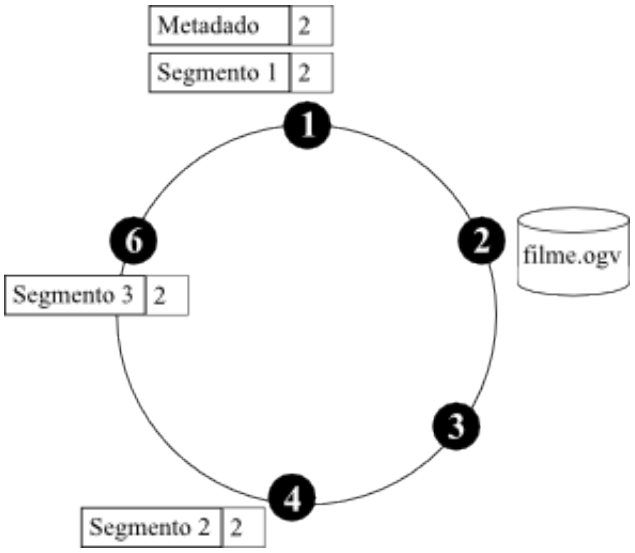

Figura 3.3: Rede após o nó 2 publicar o arquivo.

3.3.3 Caso Prático

Dadas as informações do arquivo contidas na Tabela 3.1, teremos que para esse conteúdo, serão gerados três segmentos distintos. Onde os dois primeiros segmentos terão tamanho 10k e o terceiro tamanho 5k.

Nome do Arquivo	**Tamanho**	**Tamanho do Segmento**
filme.ogv	25k	10k

Tabela 3.1: Exemplo de arquivo.

Na Tabela 3.2 mostra-se os três segmentos, com seus respectivos tamanhos e os valores hipotéticos para a função *hash* de cada um deles.

Número do Segmento	**Tamanho**	**Resultado da função** *hash*
1	10k	1
2	10k	4
3	5k	6

Tabela 3.2: Segmentos do arquivo da tabela 3.1.

Considerando o referido arquivo, vamos analisar o caso em que o nó 2 (Figura 3.3) é o proprietário do arquivo e responsável por sua publicação. Esse nó deverá então gerar e registrar o metadado. O nó também publicará os segmentos conforme a Tabela 3.2, onde o segmento 1 será registrado no nó 1, o segmento 2 será registrado no nó 4 e o segmento 3 será publicado no nó 6. Podemos ver o resultado desta distribuição na Figura 3.3.

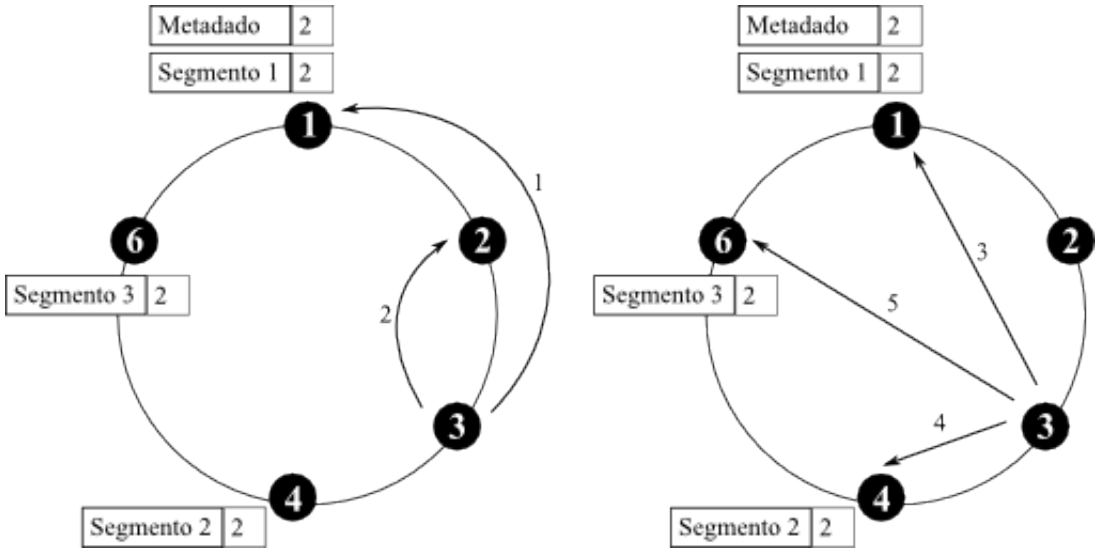

Figura 3.4: a) Busca o endereço do metadado no nó 1 e conecta-se ao nó 2 para buscar o conteúdo; b) Busca pelos endereços dos segmentos 1, 2 e 3 nos nós 1, 4 e 6.

Considerando então uma busca realizada pelo nó 3, os seguintes passos serão executados: (1) a primeira informação pesquisada será a localização do metadado. Conforme a Figura 3.4 (a), o metadado foi registrado no nó 1 e o seu conteúdo encontra-se no nó 2; (2) O nó 3 então deve conectar-se ao nó 2 e solicitar as informações sobre o arquivo; (3) após conhecer o conteúdo do arquivo, o nó 3 inicia a busca pelos segmentos, para isso, a Tabela 3.2 é utilizada para verificar o *hash* de cada segmento. Na Figura 3.4 (b) podemos verificar que o nó realizou a busca pelos segmentos, e em todas as buscas indicam que o conteúdo do segmento encontra-se no nó 2, visto que esse é o único nó que possui o arquivo no momento. Para cada segmento pesquisado, o nó realiza a busca e então conecta-se ao nó 2 para buscar o conteúdo.

3.4 Implementação

O sistema foi desenvolvido em máquinas rodando a distribuição Ubuntu do sistema operacional Linux para arquitetura 32 bits do x86. A linguagem de programação adotada foi a C++, com utilização de bibliotecas da Boost [33], a qual é uma coleção de bibliotecas multiplataforma que estendem a funcionalidade da linguagem de programação C++.

Os módulos D1HThor e a a aplicação são ambos multithread, utilizando-se para isso as funcionalidades fornecidas pela Boost. O protocolo TCP/IP é utilizado pela camada D1HThor e pela camada de aplicação. As funcionalidades originais do D1HT foram mantidas nesta implementação.

A implementação do D1HThor não utiliza o D1HT como um processo estan-

que conforme a implementação original. O código do D1HT [34] foi incorporado à aplicação na forma de uma biblioteca estática que é ligada à aplicação. Note que a tarefa de incorporar o código do D1HT à aplicação foi mencionada em [1] como sendo um trabalho a ser desenvolvido.

Desta forma, o sistema Thor é composto por três componentes distintos: 1) biblioteca com o susporte a D1HT; 2) biblioteca D1HThor e 3) código da aplicação propriamente dito. As duas primeiras camadas do sistema, encapsulamento do D1HT e desenvolvimento do D1HThor, foram desenvolvidas e testadas de forma independente. O desenvolvimento do sistema também se deu de forma que fosse facilmente identificada a camada que causasse algum erro.

Capítulo 4

D1HThor: Camada de Gerenciamento de Segmentos de Dados

O D1HThor é responsável por localizar e informar ao cliente para qual IP deve ser solicitada a informação desejada. Também é sua função, validar a autenticidade dos dados, mover a informações de um nó para outro conforme estes entram e saem da rede sobreposta D1HT.

Diferente do que foi implementado para o DHash[35], onde é armazenado o par chave/valor, o D1HThor armazena o par chave/"endereço do nó que contém o valor". O D1HT é o substrato utilizado para a localização do nó responsável pelos dados.

O objetivo da camada é garantir que se um segmento de dados estiver registrado na rede por qualquer um dos seus participantes, esse conteúdo sempre será localizado por todos os demais. No entanto, se o conteúdo estiver somente em um nó, e este sair da rede, a informação não estará mais disponível, visto que o conteúdo nunca é copiado. Para o gerenciamento das informações, o D1HThor utiliza três tabelas que serão atualizadas conforme os nós registrarem segmentos, realizarem buscas ou com a entrada e saída dos pares.

4.1 Tabelas do D1HThor

Nesta seção serão apresentados detalhes de funcionamento das tabelas do D1HThor.

4.1.1 Tabela de Registro

Quando um nó qualquer solicita o registro de um bloco, esse bloco será armazenado na Tabela de Registro (TR) do nó responsável pela chave naquele momento. Essa

TR conterá então a lista de todos os pares que solicitaram o registro de blocos enquanto o nó for responsável pela chave.

A TR é implementada de forma a conter o par Chave/IP. Onde para essa TR, o significado do conteúdo Chave/IP indica em que IP está armazenado o conteúdo dos dados com aquela chave.

Um exemplo de TR pode ser visto Figura 4.1 b), onde o nó 1 possui uma TR contendo a chave E e indicando que o conteúdo desta chave encontra-se no nó 2. O nó 2 possui uma TR contendo uma chave B e indicando que o conteúdo desta chave encontra-se no nó 1. Já o nó 4 contém uma TR contendo os valores F e D apontando para o nó 1, uma outra entrada de D apontando para o nó 2 e a chave C apontando para o nó 4.

No exemplo apresentando, nota-se que o nó 4 possui duas entradas para a chave D. Isso indica que existem duas cópias daquela chave, uma com o conteúdo no nó 1 e uma outra cópia no nó 2.

4.1.2 Tabela Inicial

Sempre que um nó qualquer realiza o registro de uma chave, este armazena o endereço do nó ao qual a chave foi registrada na Tabela Inicial (TI) local. A TI é implementada de forma a conter o par Chave/IP. Para esta implementação, o par Chave/IP informa que a Chave foi registrada no nó com o endereço IP. Sempre que um nó sair da rede, todos os demais verificam se aquele nó estava em suas TI. Se a chave se encontrar na tabela, então o nó irá registrar novamente as chaves. Isso é feito para garantir que a informação registrada não se perca, visto que o nó que conhecia a sua localização original não pertence mais à rede.

4.1.3 Tabela de Rastreamento ($Tracker$)

Esta tabela é responsável por manter a informações atualizadas conforme os nós entram e saem da rede. A implementação utiliza uma estrutura de dados que aceita chaves repetidas e contém o par Chave/IP. A informação contida nessa tabela é uma indireção que indica qual nó possui a chave k registrada.

Quando esta Tabela de Rastreamento (TT) é utilizada, isso indica que houve uma alteração do nó responsável por uma chave já registrada anteriormente. A TT é então utilizada para buscar o nó responsável pela chave no momento em que ela foi registrada. Uma vez encontrada a informação, ela é copiada para o atual nó responsável pela chave. Isso faz com que futuras pesquisas não utilizem mais a TT para buscar as informações. No entanto, por exemplo, se outro nó entrar na rede, e este for o novo responsável pela chave, ele receberá uma TT e ela será utilizada da próxima vez que a chave for buscada.

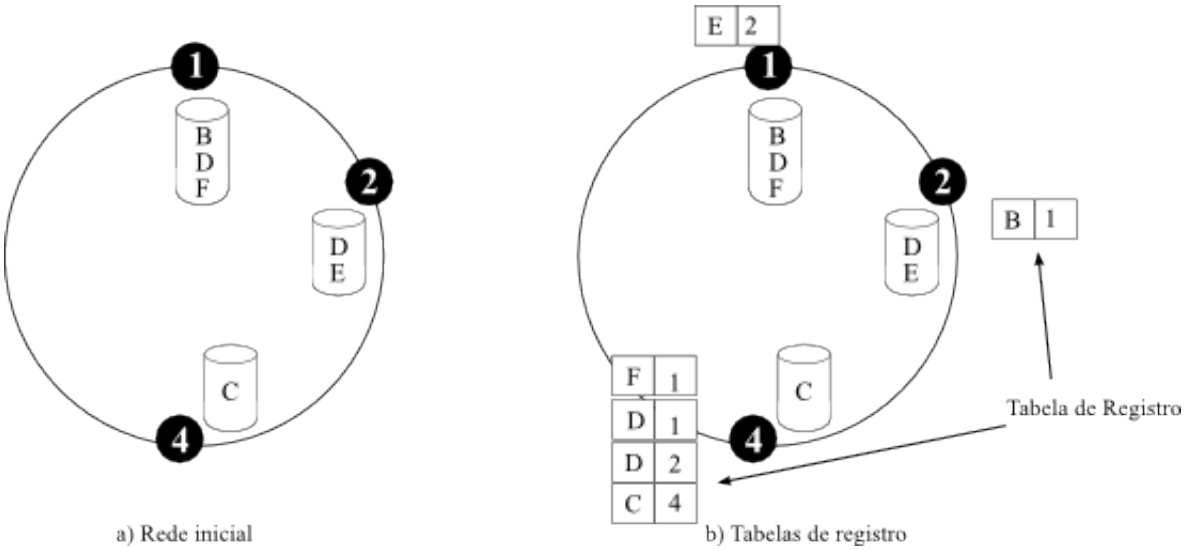

Figura 4.1: (a) Rede inidial; (b) Rede com os nós registrados.

4.1.4 Uso das Tabelas

Na Figura 4.1 (a) temos a rede de cobertura inicial com os nós 1, 2 e 4. A Figura 4.1 (b) mostra como ficou a Tabela de Registro de cada nó após a publicação dos dados. Os dados que foram registrados encontram-se na Tabela 4.1.

A Tabela 4.1 informa o número do nó, as chaves dos dados que estão disponíveis para o compartilhamento, e um exemplo do que seria um resultado da SHA-1 aplicada à chave. Por exemplo, o nó 1 possui a chave B, e teria o nó 2 como responsável por esta chave.

Após o registro das chaves, podemos então conferir na Figura 4.1 (b) que o nó 1 ficou responsável pela chave E, o nó 2 ficou responsável pela chave B e o nó 4 ficou responsável por sua chave D e mais as chaves que seriam destinadas ao nó 3, isto é, as chaves C e F.

Nó	**Chave**	**Resultado da função** *hash*
1	B	2
1	D	4
1	F	3
2	D	4
2	E	1
4	C	3

Tabela 4.1: Tabela com o registro inicial. Informa o número do nó, as chaves que o nó possui e a função *hash* para cada chave

Inclusão do nó 3

Na Figura 4.2 temos a nova configuração das tabelas com a entrada do nó 3. Com a entrada do novo nó, algumas chaves que eram responsabilidade do nó 4 devem ser migradas para o nó recem inserido. A migração dessas chaves se dá através da utilização da Tabela de Rastreamento.

Assim que o nó 3 é inserido, este deve solicitar a Tabela de Rastreamento ao seu sucessor. Esta tabela será formada por todos os nós que já se encontram na tabela de rastreamento do sucessor, mais as suas próprias chaves. Note na Figura 4.2 que todas as chaves da Tabela de Rastreamento estão apontando para o nó 4, que é o sucessor que enviou a tabela.

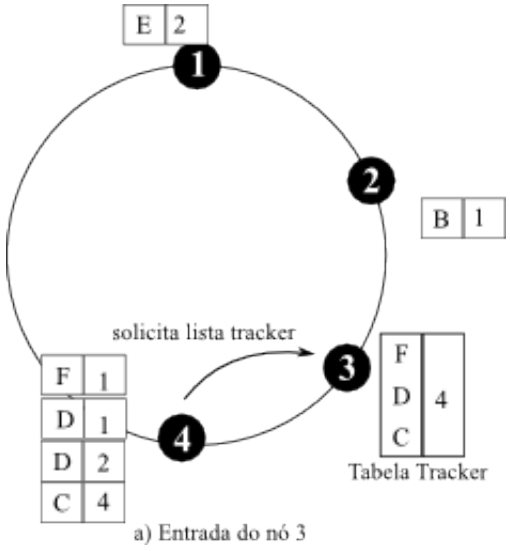

Figura 4.2: Inclusão do nó 3. A tabela tracker é solicitada ao seu sucessor.

Busca pela chave D

Para buscar a chave D na Figura 4.2, deve-se consultar o seu *hash* na Tabela 4.1. Como pode ser verificado, a tabela informa que a chave D tem como *hash* o valor 4. Ou seja, o nó 4 é o responsável pela chave D. Com isso, o D1HThor conecta-se ao nó 4 e solicita o endereço do nó que possui efetivamente os dados. Conforme a Figura 4.2, existem duas entradas para a chave D. Uma das entradas é então selecionada conforme o algoritmo implementado. Neste caso, os valores retornados para o chamador seria o nó 1 ou o 2.

Busca pela chave C

Um segundo cenário, seria a busca pela chave C. Para buscar essa chave, consulta-se o valor do seu *hash*, que neste caso é o 3. O D1HThor então conecta-se ao nó 3 e solicita o endereço do nó responsável pelos dados associados a esta chave. Note no

entanto, que a Tabela de Registro (TR) do nó 3 está vazia, visto que este nó ainda não fazia parte da rede quando a chave C foi registrada. Neste momento, a Tabela de Rastreamento é então utilizada para saber quem foi o nó que registrou a chave.

Esta busca na Tabela de Rastreamento é realizada internamente pelo próprio nó 3, e não pelo nó que iniciou a busca. O nó 3 então identifica que o responsável pela chave era o nó 4, ele conecta-se a esse nó e solicita as informações. A informação recebida é inserida na Tabela de Registro para que uma segunda busca não utilize mais a Tabela de Rastreamento. A entrada referente a chave C é apagada da Tabela de Rastreamento. E por fim, o nó 4 é retornado ao solicitante.

Ao final das buscas realizadas, teremos a Figura 4.3 com o estado final das tabelas.

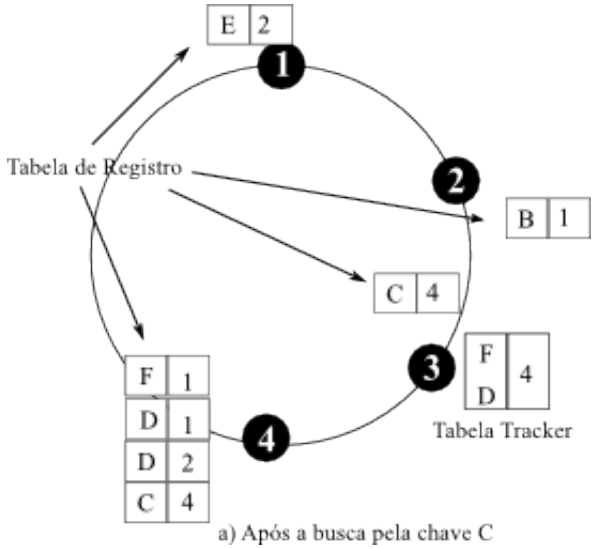

Figura 4.3: Estado final da tabela após consultas por D e C. A busca pela chave C modifica o estado das tabelas do nó 3.

4.2 Interface de Aplicação

O D1HThor fornece a seguinte API a ser utilizada pelo cliente.

4.2.1 Armazenar Informação: publish(key)

Esse método deve ser chamado sempre que um nó desejar publicar uma informação para os demais pares na rede. Esse método inclui o nó chamador na lista dos locais onde o conteúdo pode ser encontrado, o conteúdo dos dados não é copiado para a rede.

4.2.2 Recuperar Informação: IP = get(key)

Esta interface é responsável por buscar o endereço IP de um nó que contém o dado associado àquela chave. Caso exista mais de um nó que contenha a informação desejada, a escolha do nó destino será feita de forma aleatória ou em uma segunda implementação, poderia retornar o nó que estiver mais próximo de quem solicitou a informação.

Uma aplicação cliente que utilize essa interface será responsável por conectar ao nó destino e solicitar o conteúdo da informação.

4.3 Algoritmo

A apresentação dos algoritmos da implementação do D1HThor será dividido em três partes. Na primeira, serão apresentados os passos executados internamente pela interface de aplicação. Já na segunda parte, serão apresentados os serviços *multithread* que foram implementados.

Tais serviços formam o coração do D1HThor e foram desenvolvidos de forma que qualquer um dos participantes da rede tenha acesso.

Por último, serão apresentadas as chamadas de retorno (callbacks) que foram desenvolvidas a fim de capturar os eventos ocorridos na rede D1HT.

4.3.1 Interface de Aplicação

1. Ao publicar um segmento, os seguintes passos são seguidos: *publish(key k)*
   (a) Localizar o nó n responsável pela chave k;
   (b) Conectar ao nó n;
   (c) Requisitar ao nó n o registro de k;
   (d) Registrar na Tabela Inicial o nó n como sendo o nó inicial de k.
2. Ao buscar um segmento, os seguintes passos são tomados: *IP = get(key k)*
   (a) Localizar o nó n responsável pela chave k;
   (b) Conectar ao nó n;
   (c) Requisitar ao nó n os endereços dos nós que possuem o conteúdo associado à chave k;
   (d) Selecionar o melhor endereço e que possui o conteúdo de k;
   (e) Retornar o endereço e;

4.3.2 Serviços

A seguir temos os algoritmos dos serviços implementados em cada nó, e que são utilizados pela API:

1. **Recebeu de n requisição para registar a chave k.**

   (a) Se k estiver na Tabela de Rastreamento então

   - Conectar ao nó encontrado na Tabela de Rastreamento;
   - Requisitar os endereços dos nós que possuem o conteúdo associado à chave k;
   - Para cada endereço e retornado;
     - Adicionar k à Tabela de Registro com o IP e;
   - fim-para;
   - Remover k da Tabela de Rastreamento;

   fim-se

   (b) Se não existir uma entrada na Tabela de Registro com a associação da chave k para o nó n então

   - Adicionar uma entrada na Tabela de Registro com a chave k e o nó n;

   fim-se;

2. **Recebeu de n a requisição dos endereços dos nós que possuem o conteúdo associado à chave k.**

   (a) Se k estiver na Tabela de Registro então

   - Retornar os endereços de k contidos na Tabela de Registro

   (b) senão se k estiver na Tabela de Rastreamento então

   - Conectar ao nó m associado a chave k na Tabela de Rastreamento;
   - Requisitar a m os endereços dos nós que possuem o conteúdo associado a chave k;
   - Para cada endereço e retornado
     - Adicionar uma entrada na Tabela de Registro com a chave k e o endereço e

     fim-para
   - Remover k da Tabela de Rastreamento;
   - Retornar os endereços de k contidos na Tabela de Registro;

(c) Senão

- Retornar não encontrado

fim-se

3. **Recebeu de n a requisição da Tabela de Rastreamento**.

(a) Para cada chave k na Tabela de Registro

- Adicionar k e o endereço do nó local a uma Tabela de Rastreamento temporária;

fim-para

(b) Enviar ao o nó n a Tabela de Rastreamento temporária;

(c) Enviar ao o nó n a Tabela de Rastreamento local;

(d) Apagar a Tabela de Rastreamento temporária.

4.3.3 Chamadas de Retorno (*callbacks*)

As funções de *callback* foram implementadas para tratar os eventos de inclusão e remoção de um nó.

Cada nó de uma rede de cobertura D1HT contém informação sobre todos os demais participantes da rede. Com isso, cada um dos participantes da rede deve ser notificado da entrada e da saída de qualquer nó.

A implementação original do D1HT foi modificada a fim de permitir a instalação de uma função de *callback*, que é chamada sempre que o evento registrado ocorrer. Tal característica foi utilizada na implementação do D1HThor para realizar a manutenção de suas tabelas.

1. **callback: Inclusão de um nó n**

(a) Conectar ao nó sucessor s

(b) Se sucessor s não for o próprio nó então

- Solicitar a Tabela de Rastreamento

fim-se

2. **callback: Notificação de remoção de um nó n**

(a) Remover todas as entradas de n da Tabela de Registro

(b) Remover todas as entradas de n da Tabela de Rastreamento

(c) Se n estiver na Tabela Inicial então

- Deletar n da Tabela Inicial

- Localizar o nó m responsável pela chave k
- Requisitar ao nó m o registro de k
- Registrar na Tabela Inicial o nó m como sendo o nó inicial de k

fim-se

Capítulo 5

Avaliação do Thor

Este capítulo descreve os aspectos relacionados ao ambiente experimental e os testes efetuados para analisar o comportamento da aplicação desenvolvida.

5.1 Ambiente Experimental

O ambiente experimental constitui-se de 9 máquinas (Tabela 5.1), sendo 5 Intel Pentium D de 3 GHz e 4 Intel Core 2 Quad de 2.40 GHz. As máquinas encontram-se conectadas a uma rede privada de 1 Gigabit Ethernet.

Máquina	**Processador**	**Memória**	**Kernel**
Host 1	Pentium(R) D CPU 3.00GHz	2GB	Linux 2.6.35-22
Host 2	Pentium(R) D CPU 3.00GHz	2GB	Linux 2.6.35-22
Host 3	Pentium(R) D CPU 3.00GHz	2GB	Linux 2.6.35-22
Host 4	Pentium(R) D CPU 3.00GHz	2GB	Linux 2.6.35-22
Host 5	Pentium(R) D CPU 3.00GHz	2GB	Linux 2.6.35-22
Host 6	Core(TM)2 Quad Q6600 @ 2.40GHz	3GB	Linux 2.6.35-22
Host 7	Core(TM)2 Quad Q6600 @ 2.40GHz	2GB	Linux 2.6.35-22
Host 8	Core(TM)2 Quad Q6600 @ 2.40GHz	2GB	Linux 2.6.38-8
Host 9	Core(TM)2 Quad Q6600 @ 2.40GHz	2GB	Linux 2.6.35-22

Tabela 5.1: Máquinas utilizadas nos experimentos

5.2 Metodologia Utilizada

Este trabalho apresenta uma solução escalável para a distribuição de vídeos a fim de maximizar a utilização dos recursos de rede e atendendo o maior número de clientes numa rede local. Com o objetivo de demonstrar os méritos da aplicação desenvolvida, neste capítulo serão apresentados os resultados experimentais obtidos

a partir da comparação entre a aplicação desenvolvida e as soluções tradicionais de distribuição de vídeo baseadas em servidor centralizado.

Primeiramente, para ter-se uma plataforma inicial de comparação, foram realizados experimentos com a aplicação sendo executada em um servidor centralizado, modelo no qual todos os nós solicitam os segmentos do vídeo ao servidor central. Para fins práticos, no cenário avaliado, o nó atuante como servidor central foi executado em uma máquina com a largura de banda limitada a 100 Mbps. Já na solução distribuída, que não se baseia em servidor centralizado, todos os clientes são também servidores, utilizando-se de sua banda de *upload*, atingindo assim uma maior banda de rede agregada.

Em ambos os cenário, foram executados diversos experimentos com o intuito de avaliar a quantidade máxima de clientes suportados garantindo a qualidade do serviço. Para isso, foi utilizado um vídeo com uma taxa de 5 Mbps. Qualidade do serviço, no caso de streaming de vídeo, significa que o usuário deve assistir a um certo conteúdo sem interrupções e sem degradação da qualidade do vídeo causado pelo processo de distribuição. Nos experimentos realizados, o processo de distribuição é constante e não existe nenhum mecanismo de adaptação, ou seja, qualquer degradação do serviço se dá unicamente pelo atraso na distribuição dos segmentos devido à utilização da rede.

5.3 Carga Utilizada

Os testes foram realizados simulando a chegada de clientes ao sistema[1] a uma taxa λ, modelado por processo de Poisson [36]. Os intervalos de tempo entre chegadas são exponencialmente distribuídos com média $\frac{1}{\lambda}$.

5.4 Métricas Avaliadas

5.4.1 Qualidade do serviço de streaming de vídeo

Como estratégia utilizada para aferição da qualidade do serviço, neste trabalho, os vídeos são distribuídos em segmentos de 1 segundo de duração. Considera-se também um buffer no cliente capaz de armazenar somente um segundo de vídeo. Desta forma, garantir a qualidade do serviço significa que os segmentos devem chegar ao cliente sempre em um intervalo de tempo inferior a 1 segundo e qualquer atraso no recebimento dos segmentos trará impacto na qualidade do vídeo.

[1] O procedimento de geração de carga pode ser encontrado no Apêndice A

5.4.2 Balanceamento de Carga

Nesse item avalia-se como se dá a distribuição dos segmentos entre os nós. A aplicação foi instrumentada de forma a sempre informar o nó de origem do segmento recebido. Com essas informações pode-se plotar um gráfico indicando quantos segmentos cada nó forneceu aos demais nós da rede. Nesta avaliação, quanto mais disperso for a distribuição dos segmentos melhor será o desempenho da aplicação.

5.4.3 Taxa de Provimento

Tendo-se a análise do balanceamento de carga, será apresentada também a taxa máxima de provimento, que é o número máximo de clientes que também atuaram como servidores.

5.5 Avaliação dos Resultados

Nesta seção são apresentados os resultados obtidos experimentalmente e ainda as métricas avaliadas.

5.5.1 Parâmetros utilizados

Tamanho do Buffer

Utilizou-se dois tamanhos de buffer distintos: buffer de tamanho 1, capaz de armazenar um segmento de vídeo, e buffer de tamanho 4, capaz de armazenar 4 segmentos de vídeo. A utilização do buffer de tamanho 1 indica que a aplicação possui 1 espaço de armazenamento utilizado para a exibição do vídeo (buffer de exibição) e também existirá um buffer de transferência de segmento (buffer de transferência), local onde o próximo segmento do vídeo, que está sendo transferido, será armazenado. Sempre que o buffer de exibição for consumido, o conteúdo do buffer de transferência será utilizado como buffer de exibição e o próximo segmento será buscado. Já a utilização do buffer de tamanho 4 indica que a aplicação possui 4 espaços de armazenamento utilizados para a exibição do vídeo, e da mesma forma, existirá um buffer de transferência de segmento, local onde o próximo segmento do vídeo será armazenado. O conteúdo do buffer de transferência será utilizado para ocupar o local do espaço consumido do buffer de exibição. É importante notar que para ambos os tamanhos de buffer, a aplicação somente buscará o próximo segmento quando o buffer de transferência for consumido, e enquanto isso não ocorrer a aplicação ficará aguardando.

Taxa de chegada

Nos experimentos foram utilizadas diferentes taxas de chegadas, que indicam o momento de início de exibição do vídeo. As taxas utilizadas foram de 1 cliente por segundo (λ=1) e 1 cliente a cada 4 segundos (λ=0,25).

5.5.2 Avaliação do Servidor Centralizado

Para avaliação do servidor centralizado, realizou-se experimentos com a distribuição de um vídeo com taxa de 5 Mbps e com duração de 10 minutos. Avaliou-se o impacto do tamanho do buffer e da taxa de chegada dos clientes. Em todos os experimentos aqui apresentados, considerou-se uma rede de 100 Mbps limitando-se, assim, a largura de banda do servidor.

A considerar as limitações impostas, o servidor utilizado teria um limite máximo de 20 clientes, utilizando assim toda a sua largura de banda. Os experimentos demonstram os valores reais atingidos. Nos cenários avaliados, aferiu-se o número máximo de clientes que podem ser atendidos sem que haja alteração na qualidade do serviço, ou seja, sem perda de segmentos.

As latências médias de download de segmentos serão demonstradas com as figuras pertinentemente apostadas nos tópicos que se seguem. O percentual de perda de segmentos, calculado com base nos valores obtidos, são apresentados para cada experimento e em tabela comparativa. Já os gráficos de variância podem ser encontrados no Apêndice B.

Experimentos com buffer tamanho 1 e taxa de chegada de 1 cliente por segundo ($\lambda = 1$)

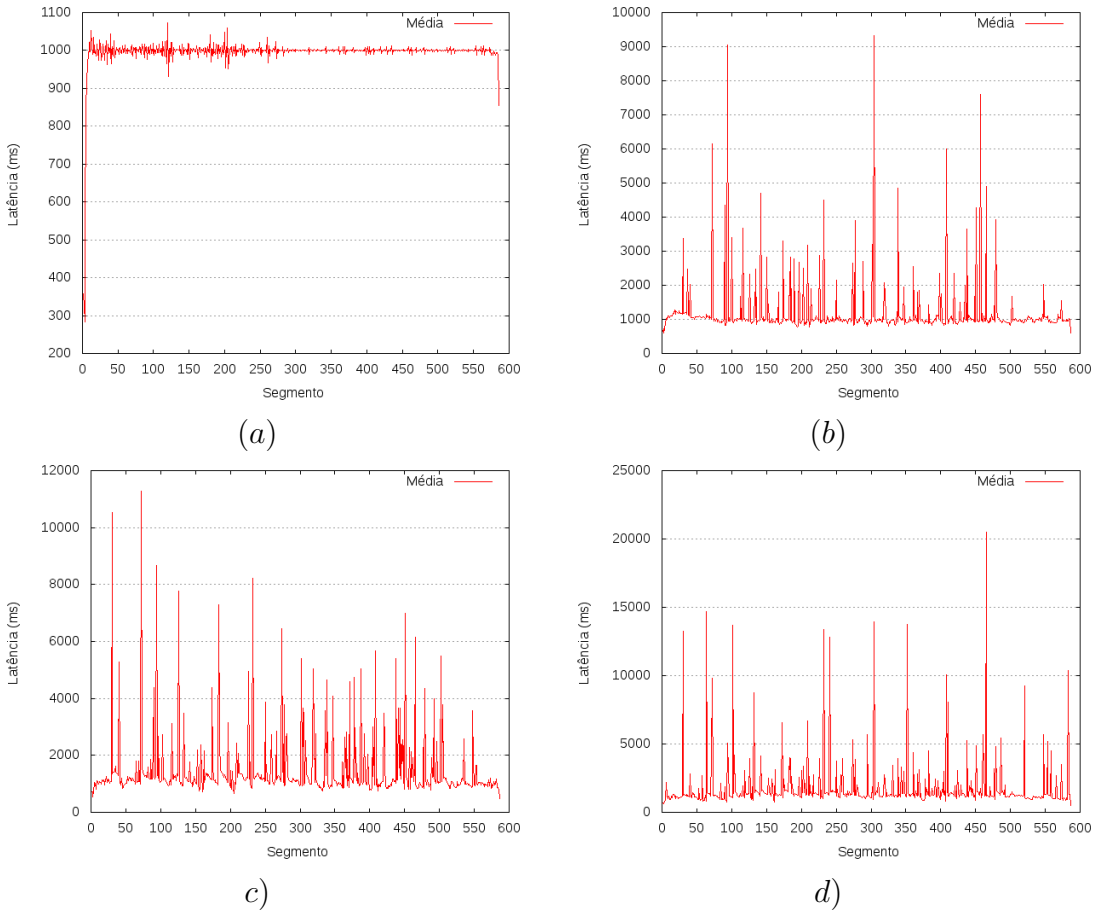

Figura 5.1: Latência média de download dos segmentos para um servidor centralizado. (a) 15 clientes; (b) 20 clientes; (c) 25 clientes; (d) 30 clientes.

Número de clientes	Percentual de perda
15	0,00%
20	18.50%
25	36.69%
30	55.19%

Tabela 5.2: Percentual de perda de segmentos para um servidor centralizado com buffer de tamanho 1 e taxa de chegada de 1 cliente por segundo.

Experimentos com buffer tamanho 1 e taxa de chegada de 1 cliente a cada 4 segundos (λ=0,25)

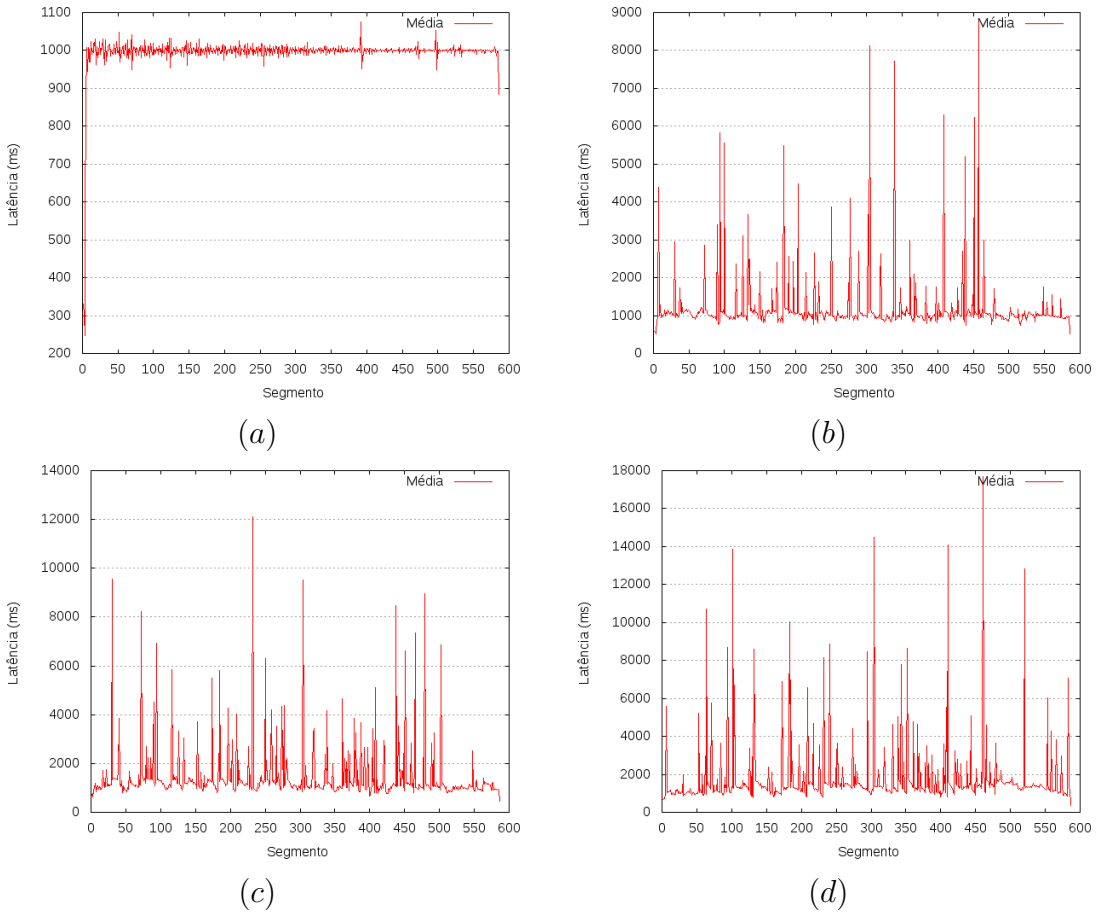

Figura 5.2: Latência média de download dos segmentos para um servidor centralizado. (a) 15 clientes; (b) 20 clientes; (c) 25 clientes; (d) 30 clientes

Número de clientes	Percentual de perda
15	0,01%
20	21.96%
25	38.21%
30	55.43%

Tabela 5.3: Percentual de perda de segmentos para um servidor centralizado com buffer de tamanho 1 e taxa de chegada de 1 cliente a cada 4 segundos.

Experimentos com buffer tamanho 4 e taxa de chegada de 1 cliente por segundo ($\lambda = 1$)

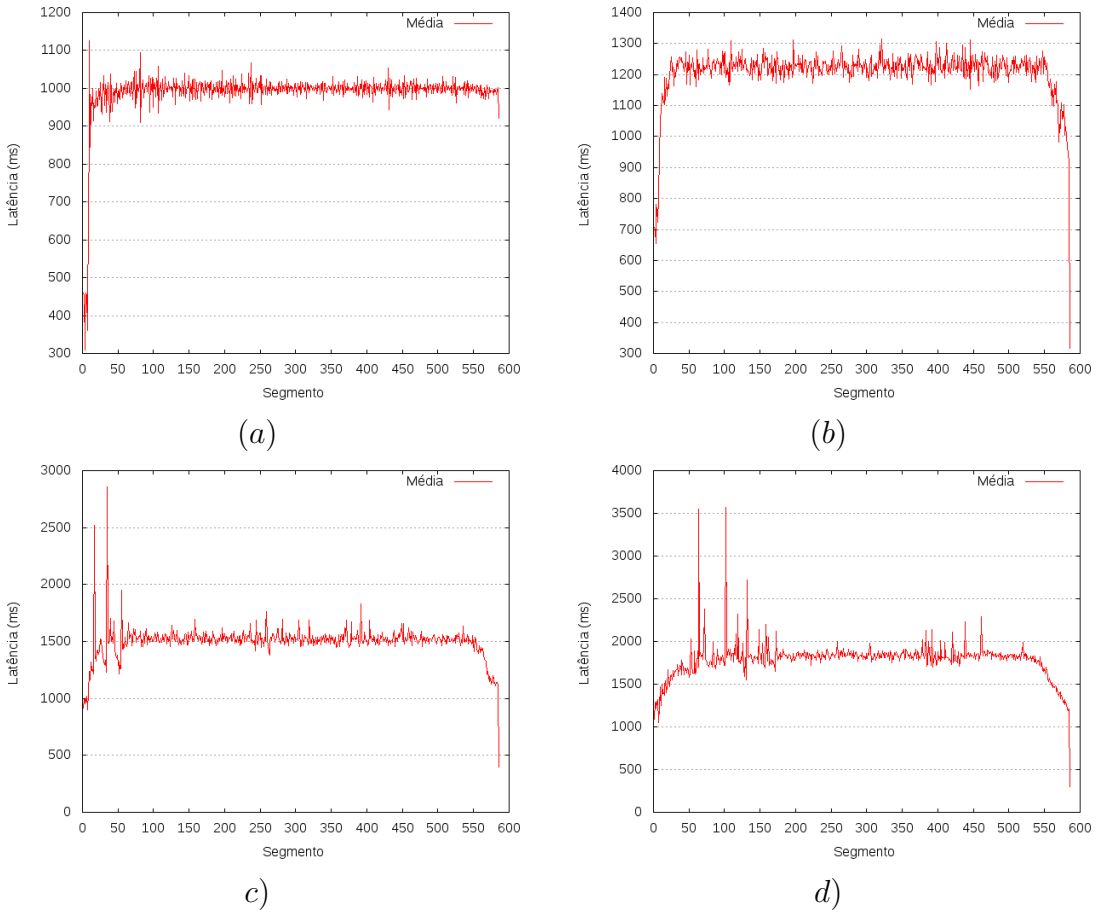

Figura 5.3: Latência média de download dos segmentos para um servidor centralizado. (a) 15 clientes; (b) 20 clientes; (c) 25 clientes; (d) 30 clientes.

Número de clientes	Percentual de perda
15	0,01%
20	8.76%
25	36.46%
30	43.41%

Tabela 5.4: Percentual de perda de segmentos para um servidor centralizado com buffer de tamanho 1 e taxa de chegada de 1 cliente por segundo.

Experimentos com buffer tamanho 4 e taxa de chegada de 1 cliente a cada 4 segundos (λ=0,25)

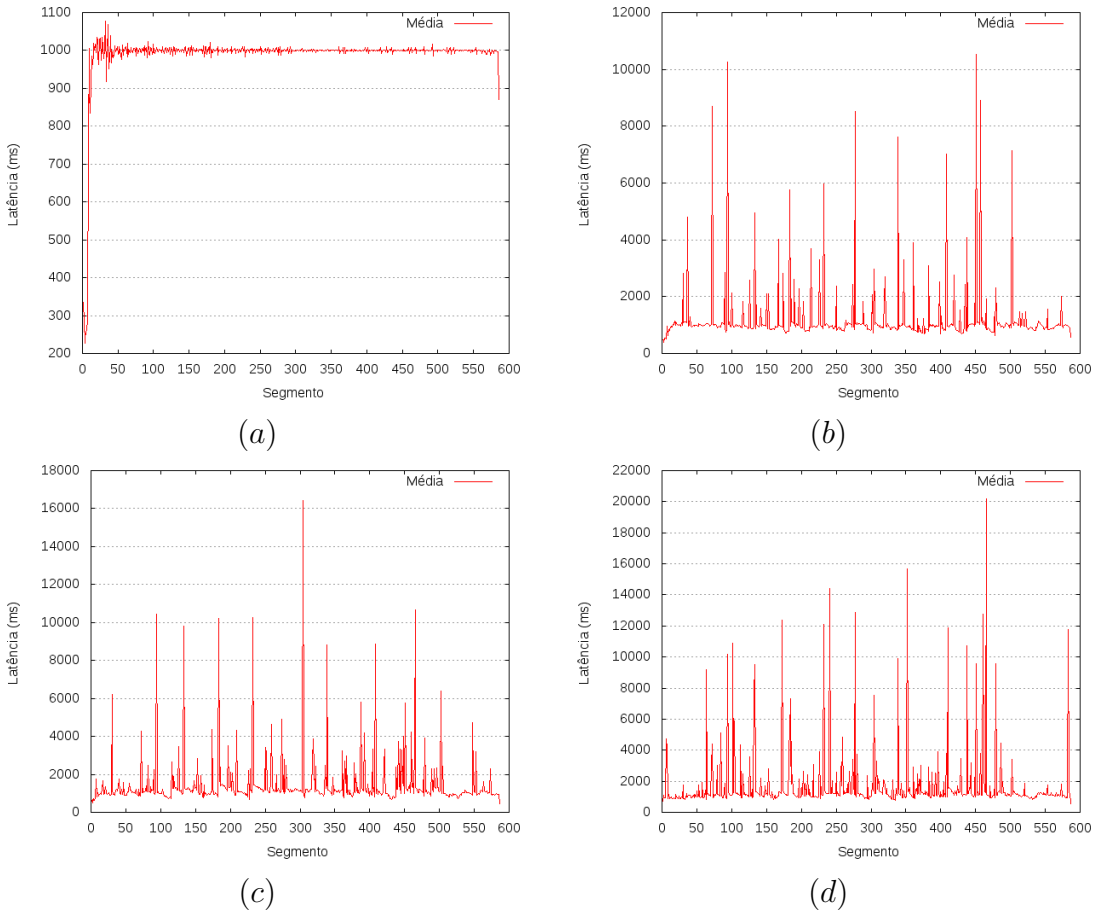

Figura 5.4: Latência média de download dos segmentos para um servidor centralizado. (a) 15 clientes; (b) 20 clientes; (c) 25 clientes; (d) 30 clientes.

Número de clientes	Percentual de perda
15	0,01%
20	14.35%
25	28.65%
30	45.14%

Tabela 5.5: Percentual de perda de segmentos para um servidor centralizado com buffer de tamanho 4 e taxa de chegada de 1 cliente a cada 10 segundos.

Discussão dos Resultados para Servidor Centralizado

Da análise dos dados anteriormente demonstrados, nota-se que o número de 15 clientes conectados foi adequadamente suportado pela rede, haja vista que a taxa de perda de segmentos foi de 0,00%. A latência média ficou em torno de 1000 ms, sendo este o valor esperado, tendo em vista a taxa de exibição de segmentos.

Todavia, diverso foi o resultado quando utilizou-se o número de 20 clientes, vez que neste contexto o serviço não foi completamente atendido em nenhum dos casos em razão da perdas de segmentos. Ainda, o aumento gradativo do número de clientes implicou no incremento das taxas de perda de segmentos, que alcançam percentuais elevados e, portanto, inadequados ao serviço de distribuição de vídeo.

A Tabela 5.6 resume os resultados obtidos nos experimentos realizados.

	Buffer = 1		**Buffer = 4**	
Clientes	$\lambda = 1$	$\lambda = 0,25$	$\lambda = 1$	$\lambda = 0,25$
15	0,00%	0,01%	0,01%	0,01%
20	18.50%	21.96%	8.76%	14.35%
25	36.69%	38.21%	36.46%	28.65%
30	55.19%	55.43%	43.41%	45.14%

Tabela 5.6: Comparativo dos percentuais de perda de segmentos para um servidor centralizado.

5.5.3 Avaliação do Thor

Conforme utilizado nos experimentos com servidor centralizado, nesta avaliação também utilizou-se um vídeo com taxa de 5 Mbps e com duração de 10 minutos. Avaliou-se o impacto do tamanho do buffer e da taxa de chegada dos clientes. Para todos os experimentos, considerou-se uma rede de 100 Mbps. Para isso, a largura de banda foi limitada em software para cada cliente.

Nos cenários avaliados, aferiu-se o número máximo de clientes que podem ser atendidos sem que haja alteração na qualidade do serviço, ou seja, sem perda de segmentos. As latências médias de download de segmentos serão demonstradas com as figuras pertinentemente apostadas nos tópicos que se seguem. O percentual de perda de segmentos, calculado com base nos valores obtidos, são apresentados para cada experimento e em tabela comparativa. Já os gráficos de variância podem ser encontrados no Apêndice B.

Número de clientes avaliados

Serão apresentados os resultados para um número de 54, 72 e 81 clientes. Sendo deste último, o valor máximo atingido no ambiente de experimentação. Com um valor de 81 nós, cada uma das 9 máquinas é encarregada de executar 9 instâncias da aplicação Thor.

Os testes com 10 instâncias em cada máquina não foram bem sucedidos, visto que nem todos os segmentos foram recuperados com sucesso. Isso deveu-se a super utilização da largura de banda da máquina, fazendo com que as mensagens de atualização das tabelas D1HT tomassem um tempo excessivo. Tal demora na atualização das tabelas gerou descarte de segmentos por parte da aplicação.

Experimentos com buffer de tamanho 1 e taxa de chegada de 1 cliente por segundo (λ=1)

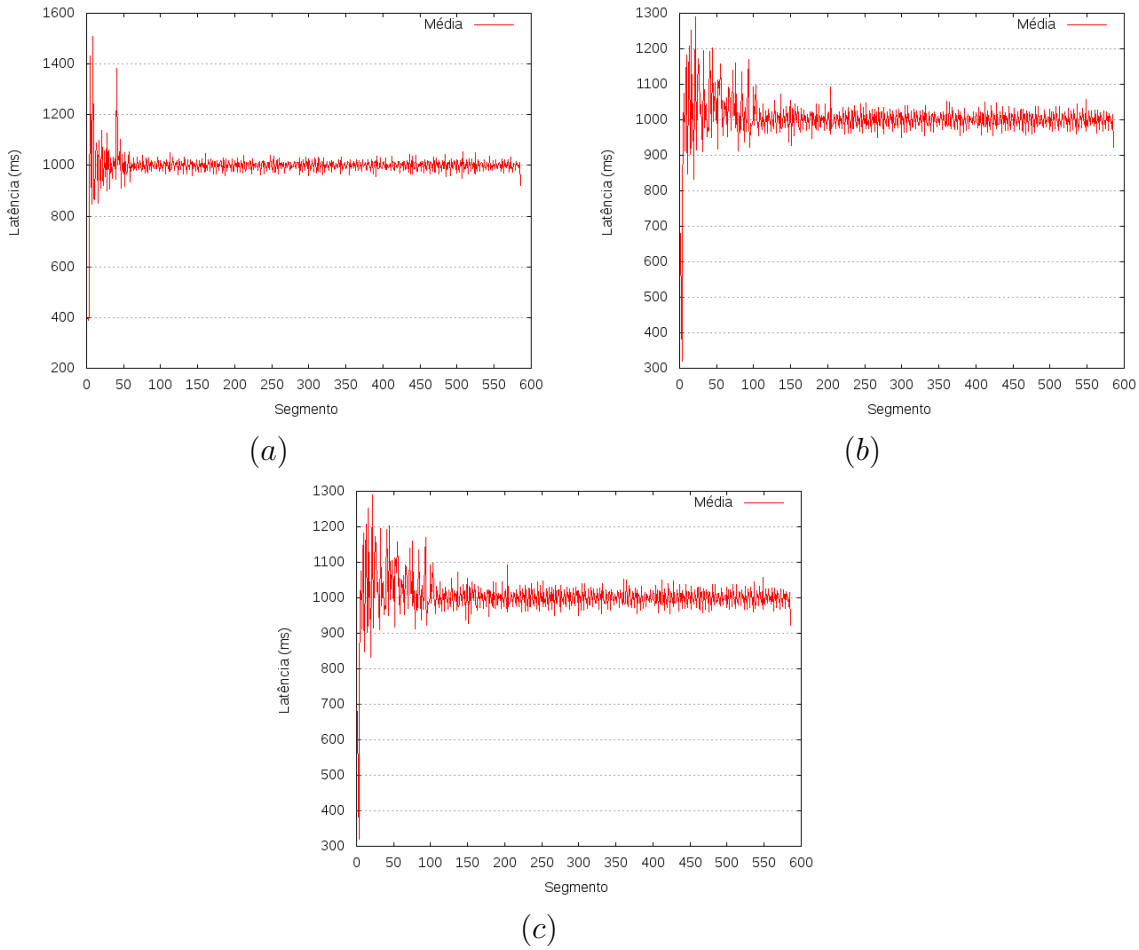

Figura 5.5: Latência média de download. (a) 54 nós; (b) 72 nós; (c) 81 nós.

Número de clientes	Percentual de perda
54	0.08%
72	0.22%
81	0.59%

Tabela 5.7: Percentual de perda de segmentos.

O balanceamento da distribuição dos vídeos pode ser visto na Figura 5.6.

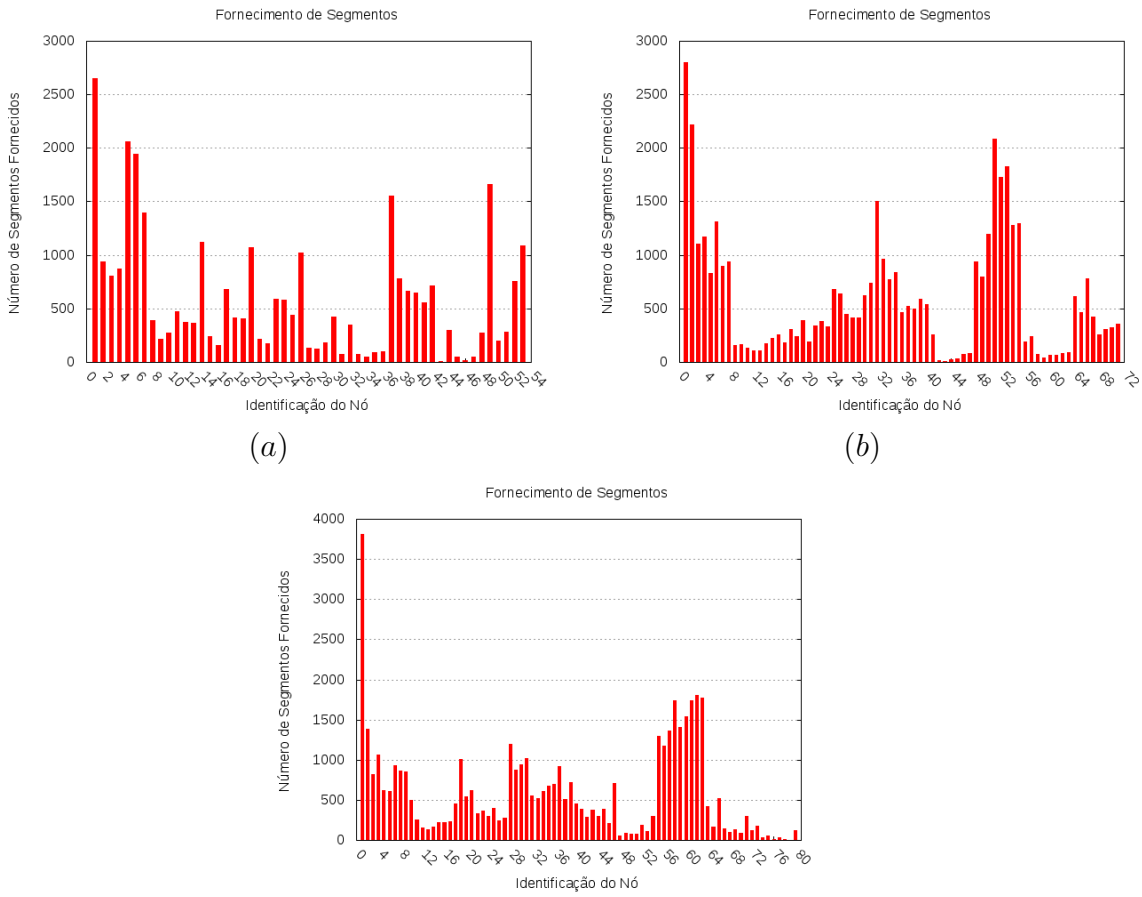

Figura 5.6: Provimento de segmentos. (a) 54 nós; (b) 72 nós; (c) 81 nós.

Experimentos com buffer de tamanho 1 e taxa de chegada de 1 cliente a cada 4 segundos (λ=0,25)

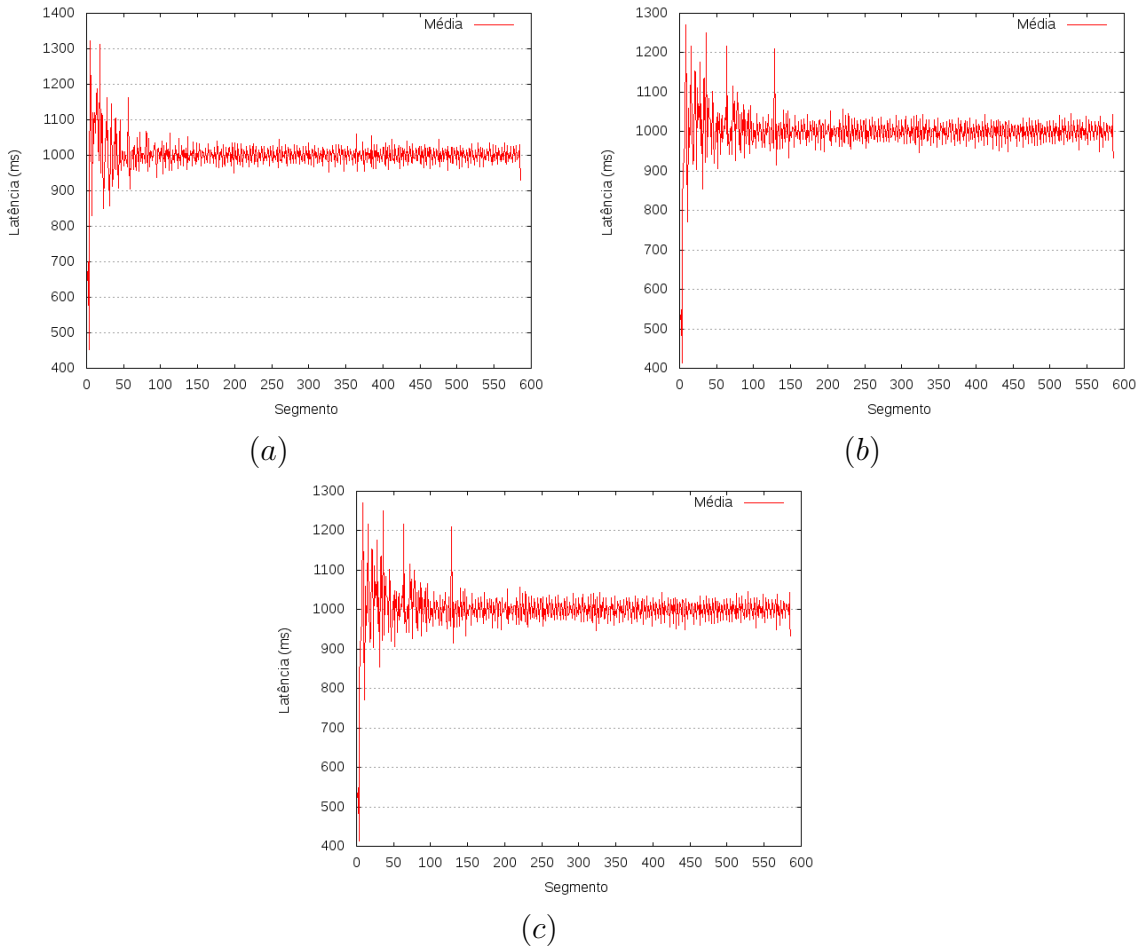

Figura 5.7: Latência média de download. (a) 54 nós; (b) 72 nós; (c) 81 nós.

Número de clientes	Percentual de perda
54	0.26%
72	0.20%
81	0.66%

Tabela 5.8: Percentual de perda de segmentos.

O balanceamento da distribuição dos vídeos pode ser visto na Figura 5.8.

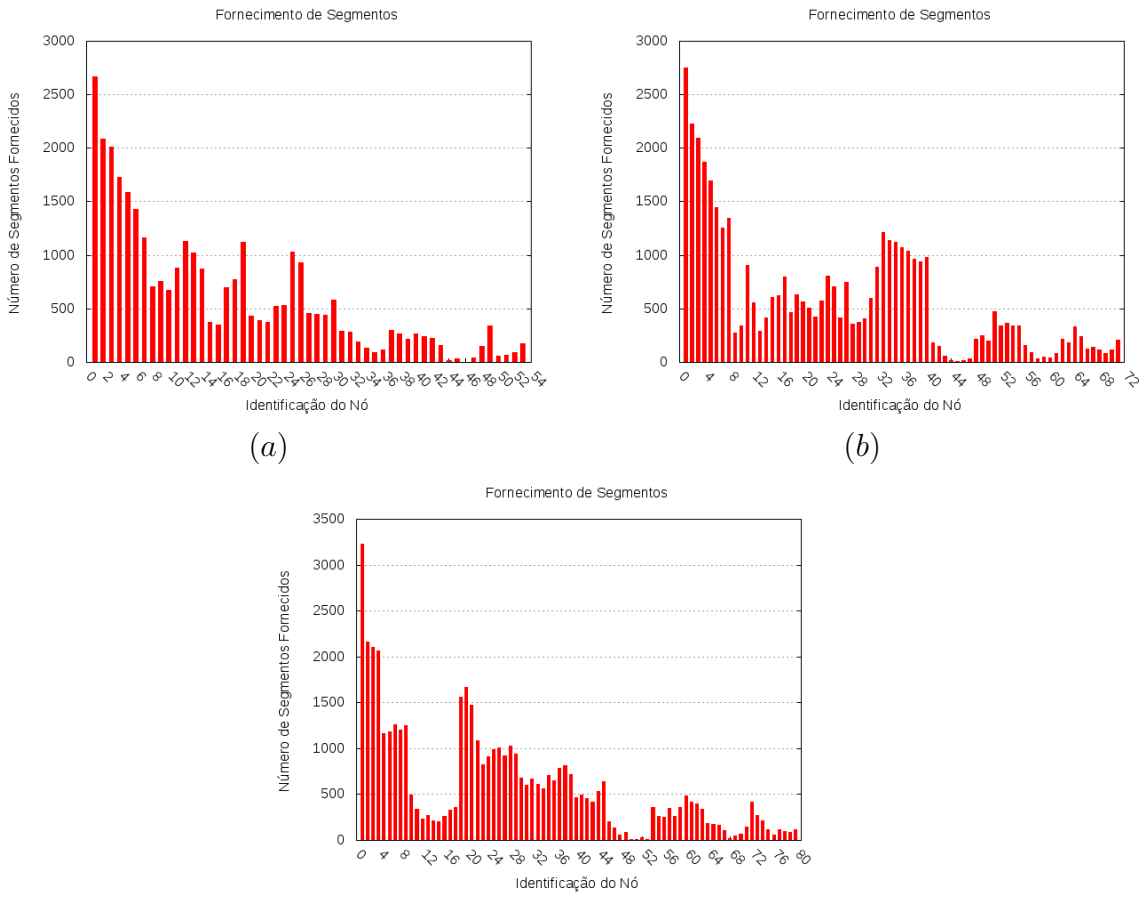

Figura 5.8: Provimento de segmentos. (a) 54 nós; (b) 72 nós; (c) 81 nós.

Experimentos com buffer de tamanho 4 e taxa de chegada de 1 cliente por segundo (λ=1)

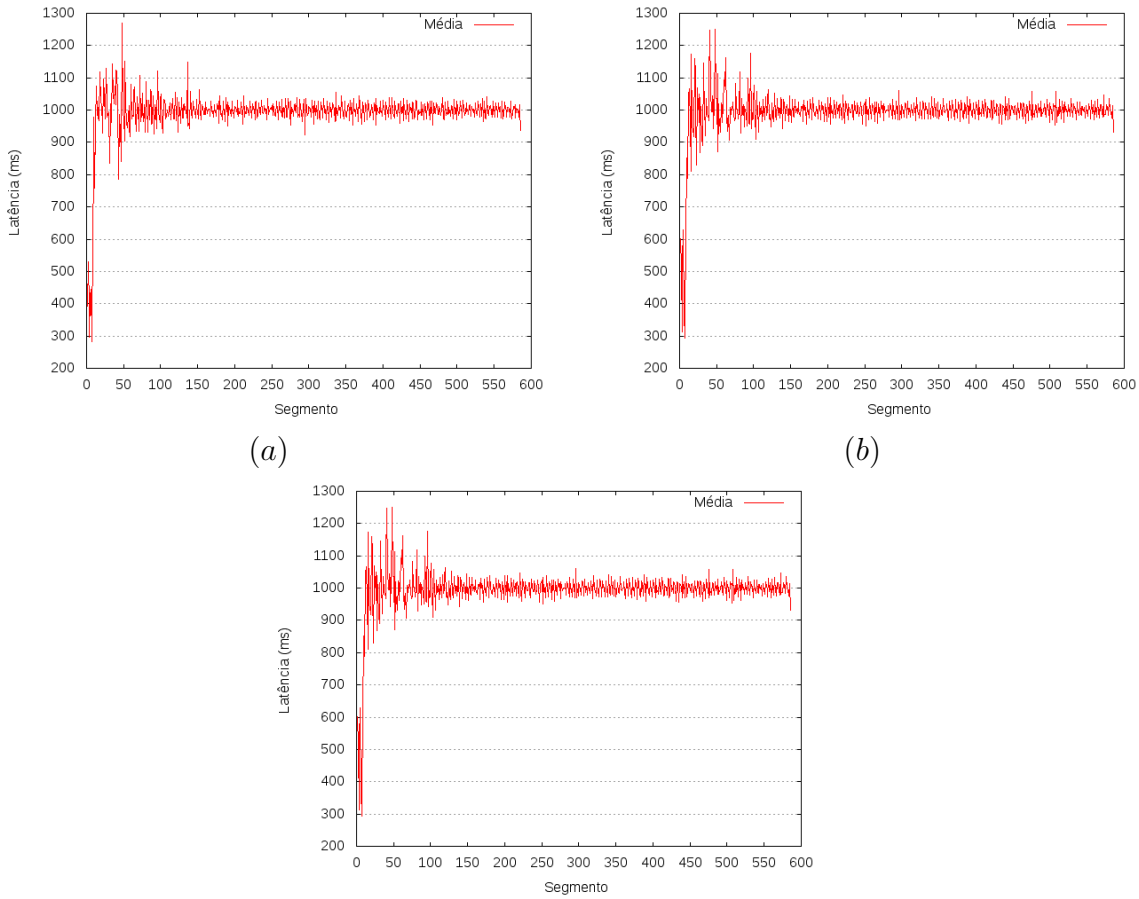

Figura 5.9: Latência média de download. (a) 54 nós; (b) 72 nós; (c) 81 nós.

Número de clientes	Percentual de perda
54	0.06%
72	0.12%
81	0.34%

Tabela 5.9: Percentual de perda de segmentos.

O balanceamento da distribuição dos vídeos pode ser visto na Figura 5.10.

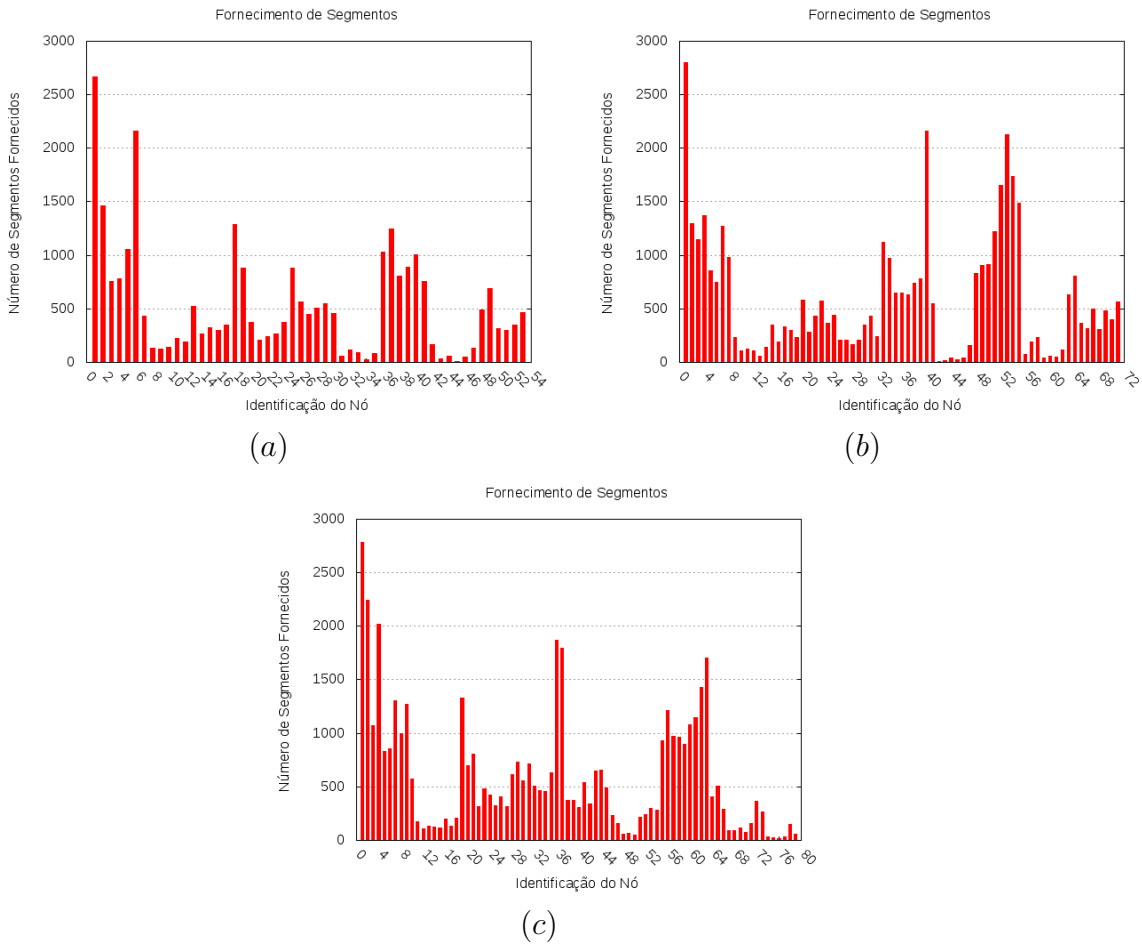

Figura 5.10: Provimento de segmentos. (a) 54 nós; (b) 72 nós; (c) 81 nós.

Experimentos com buffer de tamanho 4 e taxa de chegada de 1 cliente a cada 4 segundos (λ=0,25)

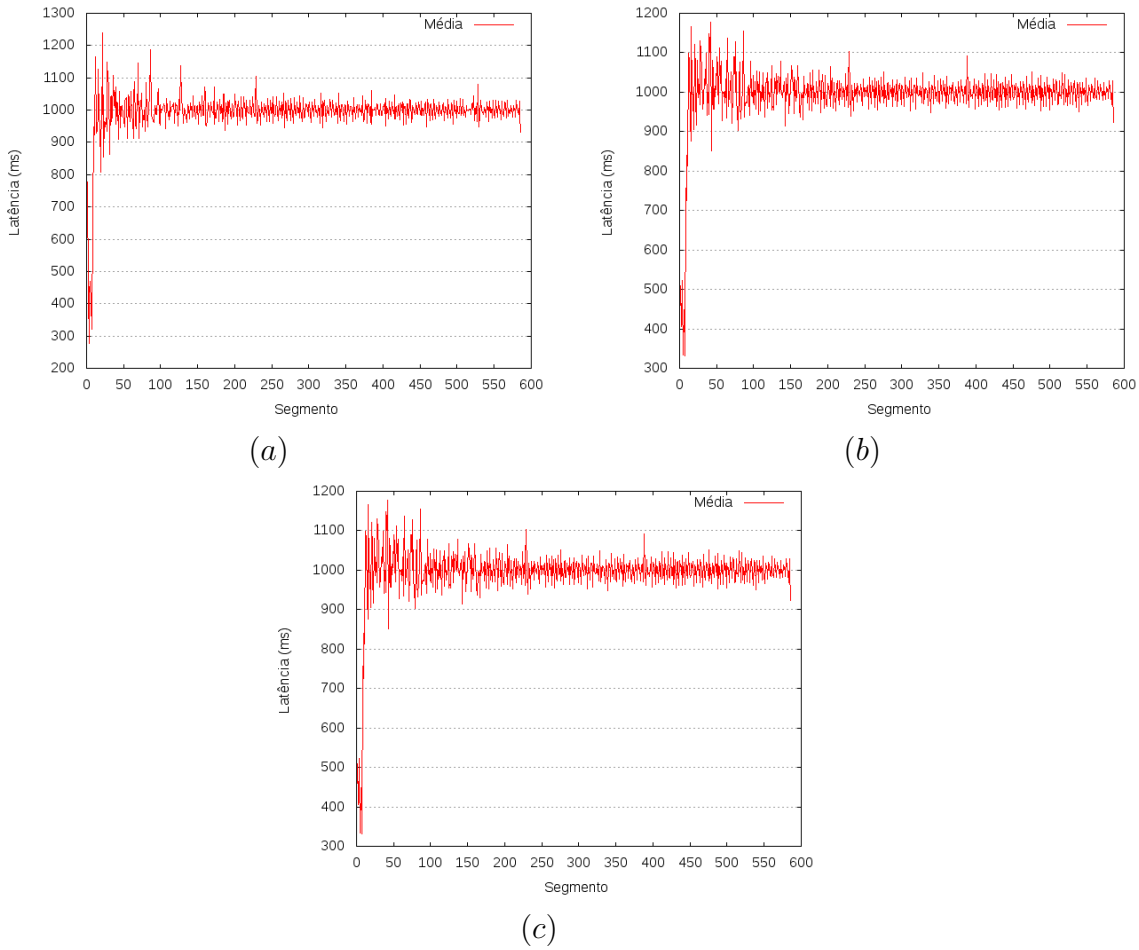

Figura 5.11: Latência média de download. (a) 54 nós; (b) 72 nós; (c) 81 nós.

Número de clientes	Percentual de perda
54	0.09%
72	0.19%
81	0.32%

Tabela 5.10: Percentual de perda de segmentos.

O balanceamento da distribuição dos vídeos pode ser visto na Figura 5.12.

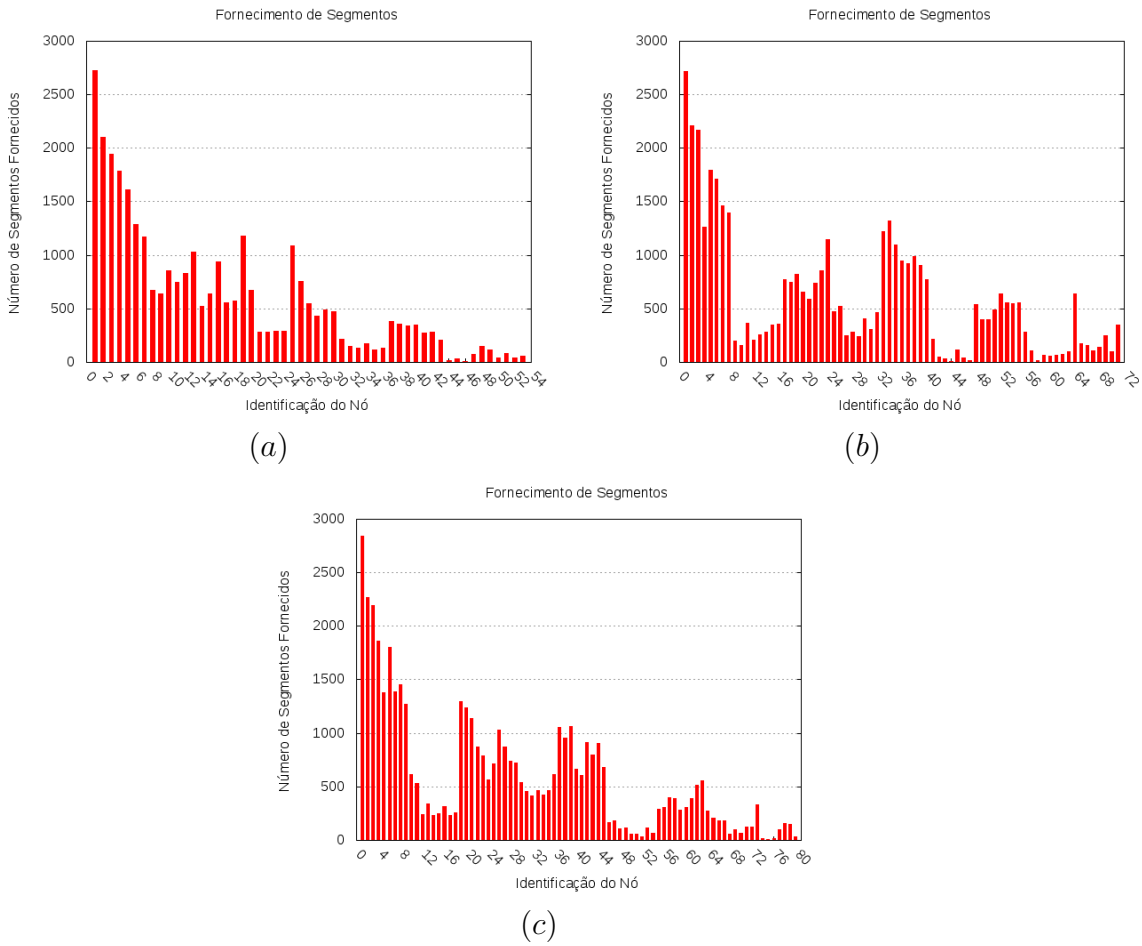

Figura 5.12: Provimento de segmentos. (a) 54 nós; (b) 72 nós; (c) 81 nós.

Análise da Qualidade do Serviço

A tabela 5.11 agrupa os percentuais de perda de segmentos para cada um dos cenários acima analisados. Os dados apresentados demonstram um excelente resultado quanto a qualidade de serviço, visto que para o pior caso analisado teve-se uma perda de somente 0,66% dos segmentos com uma escalabilidade de 81 nós.

	Buffer = 1		**Buffer = 4**	
Clientes	$\lambda = 1$	$\lambda = 0,25$	$\lambda = 1$	$\lambda = 0,25$
54	0.08%	0.26%	0.06%	0.09%
72	0.22%	0.20%	0.12%	0.19%
81	0.59%	0.66%	0.34%	0.32%

Tabela 5.11: Percentual de perda de segmentos.

O tamanho do buffer teve influência nos resultados. Isso se dá, devido a realização de buscas antecipadas dos segmentos com vistas a preencher o buffer. Desta sorte, os dados contidos no buffer são utilizados para esconder atrasos existentes na rede.

Tendo-se os dados analisados, notou-se que quase que a totalidade dos segmentos perdidos pertencem aos momentos iniciais de execução, enquanto a rede não está estabilizada. Esse fenômeno pode ser percebido nos gráficos de latência média de download.

Análise do Balanceamento de Carga

Nos testes realizados, o sistema apresentou um bom balanceamento da carga de trabalho. Tal balanceamento está refletido na tabela da taxa de provimento.

Em uma análise detalhada entre dois cenários, por exemplo, com buffer de tamanho 1 e taxas de chegada λ=1 (Figura 5.6) e λ=0,25 (Figura 5.8), temos que: pode-se verificar que existe uma diferença em como os segmentos foram fornecidos. No primeiro caso, uma taxa de chegada de 1 nó por segundo, faz com que todos os nós entrem no sistema em um curto espaço de tempo, ficando a distribuição dos segmentos mais dispersa.

Para o segundo caso, como a taxa de chegada é de 1 a cada 4 segundos, existe um intervalo de tempo maior entre a chegada dos nós ao sistema, fazendo com que os nós que entrem no sistema só tenham como pedir segmentos aos nós anteriores, pois estes já se encontram na rede. Nesse passo, o gráfico se apresenta de forma mais uniforme, concentrando o provimento dos segmentos nos primeiros nós da rede.

Tem-se no entanto, que mesmo com essa diferenciação na distribuição dos segmentos, isto não influencia na taxa total de provimento, que se mantém com pouca variação.

Análise da Taxa de Provimento

A Figura 5.12 apresenta as taxas máximas de provimento obtidas com os experimentos realizados. Com as taxas de chegadas analisadas, não foram detectadas alterações significativas quanto ao provimento de segmentos.

	Buffer = 1		**Buffer = 4**	
Clientes	$\lambda = 1$	$\lambda = 0,25$	$\lambda = 1$	$\lambda = 0,25$
54	55,55%	58,73%	60,31%	55,55%
72	54,16%	51,38%	56,94%	55,55%
81	56,79%	54,32%	53,08%	58,02%

Tabela 5.12: Taxa de Provimento.

Durante os experimentos notou-se que a taxa de provimento não ultrapassou o valor de 60,31%. Este fato deve-se à pouca diversidade de segmentos na rede, visto que o sistema utiliza uma busca sequencial de segmentos. Com essa estratégia de busca, todos os nós requisitam sempre os mesmos segmentos.

5.6 Discussão

Conforme os resultados aqui apresentados, nota-se de forma clara a limitação de escalabilidade de um servidor centralizado, que, nos experimentos aqui realizados, não conseguiu nem mesmo atender o seu limite de banda teórico, ficando assim abaixo das expectativas. Já o sistema Thor tira proveito das técnicas de D1HT para a busca e distribuição dos segmentos de vídeo, técnica esta que a aplicação utiliza através da camada de gerenciamento de segmentos D1HThor. Camada esta que mostrou-se eficiente no manuseio dos segmentos garantindo a localização e recuperação dos mesmos. Assim, pode-se verificar que sistema Thor, apresentou-se de forma eficiente na distribuição de vídeos sob demanda em uma rede P2P local, mostrando ser um sistema escalável e com um bom balanceamento de carga.

Os experimentos realizados neste trabalho ficaram limitados à utilização de uma rede local e com uma limitação no número de máquinas. No entanto, a D1HT foi projetada para atender milhões de nós em uma rede na Internet. Desta forma, experimentos adicionais em um ambiente de Internet devem ser realizados com o Thor, a fim de avaliar seu comportamento e com o intuito de confirmar os resultados obtidos em uma rede local.

Capítulo 6

Trabalhos Relacionados

Neste capítulo são abordados os principais trabalhos relacionados ao tema.

6.1 Streaming em Redes P2P

GloVE[37] (*Global Video Environment*) utiliza técnicas de reuso de fluxo chamada de Cache de Vídeo Cooperativa (CVC) para agregar escalabilidade a servidores de VoD de baixo custo. Nesta técnica, os buffers locais dos clientes integram uma memória global distribuída, usada como fonte prioritária de conteúdo, fazendo com que os clientes se tornem provedores de outros clientes, segundo um modelo P2P baseado em conceitos das técnicas de *Chaining* e *Patching*. O GloVE então coordena a CVC através de um gerente centralizado com diferentes políticas de escolha do cliente responsável por responder a uma determinada requisição.

Esse trabalho é o que mais se relaciona com o Thor em se tratando de ambiente de desenvolvimento, pois ambos atacam o problema de distribuição de vídeo utilizando-se de uma rede local. As diferenças são encontradas no que diz respeito ao buffer, que no Thor o arquivo inteiro é copiado para os clientes, sem preocupação com limite de espaço. Também as técnicas de *Chaining* e *Patching* não são utilizadas pelo Thor.

O BiToS [38] é provavelmente o primeiro trabalho a usar um sistema P2P VoD. Utiliza um protocolo desenvolvido com base no BitTorrent. O mecanismo original do BitTorrent utiliza o critério de buscar primeiro os segmentos mais raros na rede. No entanto, esse mecanismo não é adequado a transmissão de dados que possuam uma janela de tempo restrita para serem recebidos. Desta forma, foram realizadas modificações no mecanismo de seleção de segmentos, a fim de suportar streaming de vídeo. O mecanismo de seleção possui três componentes: partes recebidas, um conjunto com partes de alta prioridade e um conjunto de partes que estão faltando. No mecanismo desenvolvido, o par seleciona com um probabilidade p para baixar uma parte do vídeo pertencente ao conjunto de alta prioridade, e seleciona com uma

probabilidade $1 - p$ uma parte contida no conjunto de partes que estão faltando. Desta forma, as partes que se encontram no conjunto de alta prioridade, que são as partes mais próximas que serão utilizadas, serão buscadas antes das partes que não estão próximas de sua utilização.

Com relação ao sistema apresentado neste trabalho, a maior diferença se dá em relação as buscas dos segmentos. O Thor utiliza a busca sequencial dos segmentos, já o BiToS utiliza uma busca baseada em critérios estatísticos com o intuito de criar uma maior diversidade nos segmentos distribuídos. Outra diferença fundamental está relacionada ao ambiente, que para o sistema Thor utilizou-se redes locais.

P2VoD [39] tira vantagem dos nós intermediários que encaminham o conteúdo multimídia mais recentemente recebido, e que tenha sido armazenado em cache. Clientes existentes na rede P2VoD podem encaminhar o *stream* de vídeo para um novo cliente assim que possuir banda da saída suficiente e que ainda tenha o primeiro bloco do arquivo de vídeo armazenado no buffer. Um mecanismo de *caching* é usado para permitir a um grupo de clientes, que tenham chegado no sistema em momentos distintos, a armazenar o mesmo conteúdo de vídeo em uma área específica de seus $buffers$. Nesse trabalho também é proposto um eficiente protocolo de controle, baseado em árvores *multicast*, para facilitar o gerenciamento de entradas de novos nós e o processo de recuperação de falha.

O mecanismo de *caching* empregado nesse sistema possui um tamanho limitado, diferindo do que é empregado no Thor, que se propõe a transferir todo o conteúdo do vídeo, e não somente em exibi-lo. O mecanismo empregado no P2VoD utiliza-se de uma rede estruturada em árvores, já o Thor emprega uma rede P2P desestruturada.

6.2 Trabalhos envolvendo DHTs de um salto[1]

Este trabalho utilizou uma DHT de um salto chamada de D1HT. No entanto, existem outras DHTs que resolvem consultas em um salto.

O sistema OneHop [12] foi a primeira DHT proposta a garantir que a maior parte das consultas sejam resolvidas em um único salto, mesmo em ambientes dinâmicos [40]. Em contraste com a arquitetura puramente P2P de D1HT, a propagação das informações sobre eventos em OneHop é baseada em uma hierarquia, onde os nós são agrupados em unidades (ou slices), que por sua vez são agrupadas em blocos. Uma vez que cada unidade e cada bloco têm um líder, a hierarquia imposta por OneHop divide os nós participantes em três níveis distintos: nós comuns, líderes de unidades e líderes de blocos. Esta topologia hierárquica é construída com o objetivo de viabilizar o agrupamento de várias notificações de eventos em uma mesma mensagem, procurando-se assim minimizar as demandas de rede para manutenção das tabelas de roteamento. Para tanto, cada líder de unidade é responsável por

acumular as informações sobre todos os eventos em sua própria unidade e periodicamente propagá-las ao líder do seu bloco. Cada líder de bloco agrupa os eventos ocorridos em suas diversas unidades e periodicamente os encaminha para todos os outros líderes de bloco. Os diversos líderes de bloco irão então tomar conhecimento de todos os eventos ocorridos no sistema, e vão encaminhar estas informações para todos os líderes de unidades do seu bloco. Cada líder de unidade será então responsável por iniciar a propagação destes eventos por todos os nós de sua respectiva unidade.

Enquanto a hierarquia imposta por OneHop facilita o agrupamento de eventos de maneira a minimizar os seus custos de manutenção das tabelas de roteamento, D1HT consegue atingir o mesmo objetivo usando uma arquitetura puramente P2P. Deste modo, D1HT não só tem custos de manutenção inferiores, como também evita vários problemas intrínsecos ao sistema hierárquico usado por OneHop.

O sistema 1h-Calot [13], desenvolvido de maneira simultânea e independente ao D1HT, guarda algumas similaridades, já que ambos os sistemas procuram resolver as consultas com um único salto e fazem a propagação de evento com uso de árvore logarítmica. Há, no entanto, diferenças fundamentais entre estes dois sistemas. Primeiro, por que D1HT constrói suas árvores de propagação de eventos a partir de TTLs, enquanto 1h-Calot baseia-se em intervalos de identificadores. Além disto, 1h-Calot não tem suas propriedades de desempenho e correção formalmente provadas. Mas a principal diferença entre estas duas DHTs reside no fato de 1h-Calot não ser capaz de agrupar eventos para minimizar seus custos de manutenção, e assim as suas demandas de banda passante são muito superiores às de D1HT, tornando inviável o seu uso em muitas aplicações.

Capítulo 7

Conclusões e Trabalhos Futuros

Nesta dissertação foi proposto, implementado e avaliado um sistema de vídeo sob demanda P2P para distribuição de vídeos baseados na técnica D1HT denominado Thor. Foram descritas a arquitetura e as camadas que formam a aplicação Thor. Descreveu-se como a camada D1HThor utiliza as suas tabelas para garantir que todos os segmentos de dados registrados sejam sempre encontrados na rede e que nenhuma informação seja perdida com a entrada e saída aleatórias dos nós.

Foram realizados experimentos com um número diverso de clientes e com tamanho de buffer e taxa de chegadas variados. O sistema apresentou um bom desempenho em todos os testes realizados. Mostrou também possuir uma ótima escalabilidade mantendo a qualidade de serviço.

Os experimentos realizados neste trabalho ficaram limitados à utilização de uma rede local e com uma limitação no número de máquinas. No entanto, a D1HT foi projetada para atender milhões de nós em uma rede na Internet. Desta forma, experimentos adicionais em um ambiente de Internet devem ser realizados com o Thor, a fim de avaliar seu comportamento e com o intuito de confirmar os resultados obtidos em uma rede local.

Uma boa proposta de alteração do sistema Thor seria a implementação de formas alternativas de buscas pelos segmentos como, por exemplo, a implementação de busca antecipada de forma aleatória a outros segmentos além daquele que será o próximo utilizado. Poder-se-ia também criar duas listas de segmentos com prioridades distintas, uma de alta prioridade para os próximos segmentos a serem utilizados e, outra de baixa prioridade de onde seriam buscados os demais segmentos. Tais modificações propostas aumentariam a diversidade dos segmentos e possivelmente traria um impacto na melhora da taxa de provimento.

Referências Bibliográficas

[1] MONNERAT, L. R. *Tabelas Hash Distribuídas com Consultas de um Salto com Baixa Carga de Manutenção*. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2010.

[2] BUYYA, R., PATHAN, M., VAKALI, A. "Content Delivery Networks". Springer, 2008.

[3] "YouTube". . Disponível em: <`http://www.youtube.com`>. Último acesso em Agosto de 2011.

[4] RISSON, J., MOORS., T. "Survey of research towards robust peer-to-peer networks: Search methods." *Computer Networks*, v. 50, n. 17, pp. 3485–3521, 2006.

[5] ROWSTRON, A., DRUSCHEL, P. "Pastry: Scalable, decentralized object location, and routing for largescale peer-to-peer systems", *IFIP/ACM Middleware*, v. 42, n. 2, pp. 145–163, nov. 2001.

[6] STOICA, I., MORRIS, R., LIBEN-NOWELL, D., etal. "Chord: a scalable peer-to-peer lookup protocol for Internet applications", *IEEE/ACM Transactions on Networking*, fev. 2003.

[7] RATNASAMY, S., FRANCIS, P., HANDLEY, M., etal. "A scalable content-addressable network", *ACM SIGCOMM*, ago. 2001.

[8] MONNERAT, L. R., AMORIM, C. L. "D1HT: A Distributed On Hop Hash Table". In: *Proceedings of the 20th IEEE International Parallel and Distributed Processing Sysmposium (IPDPS)*, maio 2006.

[9] WU, D., HOU, Y. T., ZHU, W., etal. "Streaming Video over the Internet: Aproaches and Directions", *IEEE Trans. Circuits and Systems for Video Technology*, v. 11, n. 3, pp. 282–300, mar. 2001.

[10] HU, A. "Video-on-Demand Broadcasting Protocols: A Comprehensive Study". In: *Proceedings of the IEEE INFOCOM 2001*, Anchorage, Alaska, abr. 2001.

[11] ZHAO, B. Y., KUBIATOWICZ, J., JOSEPH, A. D. *Tapestry: An infrastructure for fault-tolerant widearea location and routing.* UC Berkeley UCB/CSD-01-1141, 2001.

[12] FONSECA, P., RODRIGUES, R., GUPTA, A., etal. "Full-Information Lookups for Peer-to-Peer Overlays", *IEEE/ACM Transactions on Parallel and Distributed Systems*, set. 2009.

[13] TANG, C., BUCO, M. J., CHANG, R. N., etal. "Low traffic overlay networks with large routing tables." In: *Proceedings of the International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS)*, pp. 14–25, jun. 2005.

[14] MONNERAT, L., AMORIM, C. "Peer-to-Peer Single Hop Distributed Hash Tables". In: *Proceedings of IEEE GLOBECOM*, nov. 2009.

[15] LI, J., STRIBLING, J., MORRIS, R., etal. "A performance vs. cost framework for evaluating DHT design tradeoffs". In: *Proc. of INFOCOM*, mar. 2005.

[16] KARGER, D., LEHMAN, E., LEIGHTON, T., etal. "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web". In: *Proceedings of the Symposium on Theory of Computing*, maio 1997.

[17] NIST. "Secure Hash Standard (SHS)", *FIPS Publication*, pp. 180–1, abr. 1995.

[18] RILEY, C., SCHEIDELER, C. "A distributed hash table for computational grids". In: *Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.

[19] ROCHA, V., NETTO, M. A. S., KON, F. "Descoberta de Recursos em Grades Computacionais Utilizando Estruturas P2P". In: *Proceedings of the 24th Brazilian Symposium on Computer Networks (SBRC)*, maio 2006.

[20] SCHINTKE, F., SCHUTT, T., REINEFELD, A. "A framework for self-optimizing Grids using P2P components". In: *Proceedings of the 14th IEEE DEXA*, 2003.

[21] COX, R., MUTHITACHAROEN, A., MORRIS, R. "Serving DNS using Chord". In: *Proceedings of IPTPS*, mar. 2002.

[22] RAMASUBRAMANIAN, V., SIRER, E. "The design and implementation of a next generation name service for the Internet". In: *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, ACM, set. 2004.

[23] KUBIATOWICZ, J., BINDEL, D., CHEN, Y., etal. "OceanStore: An Architecture for Global-scale Persistent Storage". In: *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, nov. 2000.

[24] CALVAGNA, A., LORENZO, L. D., TROPEA, G. "Video-on-Demand for P2P Communities". In: *Proceedings of The Tenth International Conference on Distributed Multimedia Systems (DMS)*, set. 2004.

[25] HEFEEDA, M., HABIB, A., BOTEV, B., etal. "PROMISE: Peer-to-peer media streaming using CollectCast". In: *Proceedings of the 11th ACM International Conference on Multimedia (MM-03)*, pp. 45–54, nov. 2003.

[26] COX, L., NOBLE, B. "Pastiche: Making backup cheap and easy". In: *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, dez. 2002.

[27] STRIBLING, J., COUNCILL, I. G., LI, J., etal. "OverCite: A Cooperative Digital Research Library". In: *Proceedings of IPTPS*, fev. 2005.

[28] GYLFASON, H. I., KHAN, O., SCHOENEBECK, G. "Chora: Expert-based P2P web search". In: *Proceedings of the Workshop on Agents and P2P Computing (AP2PC)*, 2006.

[29] BRYAN, D. A., LOWEKAMP, B. B., JENNINGS, C. "SOSIMPLE: A Serverless, Standards-based, P2P SIP Communication System". In: *Proceedings of the 2005 International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications (AAA-IDEA)*, jun. 2005.

[30] SINGH, K., SCHULZRINNE, H. "Peer-to-peer internet telephony using SIP". In: *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, pp. 63–68, jun. 2005.

[31] KNUTSSON, B., LU, H., XU, W., etal. "Peer-to-Peer support for massively multiplayer games". In: *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, mar. 2004.

[32] HUEBSCH, R., HELLERSTEIN, J., BOON, N., etal. "Querying the Internet with PIER". In: *Proceedings of the International Conference on Very Large Databases*, set. 2003.

[33] "BOOST". . Disponível em: <`http://www.boost.org/`>. Último acesso em Agosto de 2011.

[34] MONNERAT, L. R. "D1HT source code available under GPL license from http://lcp.coppe.ufrj.br/D1HT", .

[35] DABEK, F., LI, J., SIT, E., etal. "Designing a DHT for low latency and high throughput". In: *Proc. of the 1st Symposium on Networked Systems Design and Implementation*, mar. 2004.

[36] ROSS, S. M. *Simulation*. Elsevier Academic Press, 2006.

[37] DE PINHO, L. B. *Implementação e Avaliação de um Sistema de Vídeo sob Demanda Baseado em Cache de Vídeo Cooperativa.* M.Sc. dissertation, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2002.

[38] VLAVIANOS, A., ILIOFOTOU, M., FALOUTSOS, M. "Bitos: Enhancing bittorrent for supporting streaming applications." *In 9th IEEE Global Internet Symposium*, abr. 2006.

[39] DO, T. T., HUA, K. A., TANTAOUI, M. A. "P2PVoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment". In: *Proceedings of the IEEE International Conference on Communications (ICC 2004)*, Paris, France, jun. 2004.

[40] GUPTA, A., LISKOV, B., RODRIGUES, R. "Efficient Routing for Peer-to-Peer Overlay". In: *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, mar. 2004.

Apêndice A

Procedimento de Geração de Carga

Para realizar os experimentos, foi desenvolvido um programa gerador de tempos de chegadas de n clientes segundo o processo de Poisson. O programa recebe como parâmetro a taxa de chegada λ e escreve os tempos de chegada em um arquivo, uma linha para cada nó que será executado. Após esse passo, o programa encarregado de iniciar a aplicação lê o arquivo gerado e inicia a requisição do arquivo de acordo como tempos lidos. O seguinte procedimento é utilizado pela na máquina encarregada de iniciar os clientes:

1. Lê o arquivo com as máquinas onde a aplicação será executada;
2. Lê o arquivo gerado com os tempos de início de cada nó;
3. Inicia a aplicação em todos os nós;
4. Faz um ssh para cada nó solicitando que o nó requisite o arquivo no momento t lido do arquivo;

Apêndice B

Gráficos de Variância

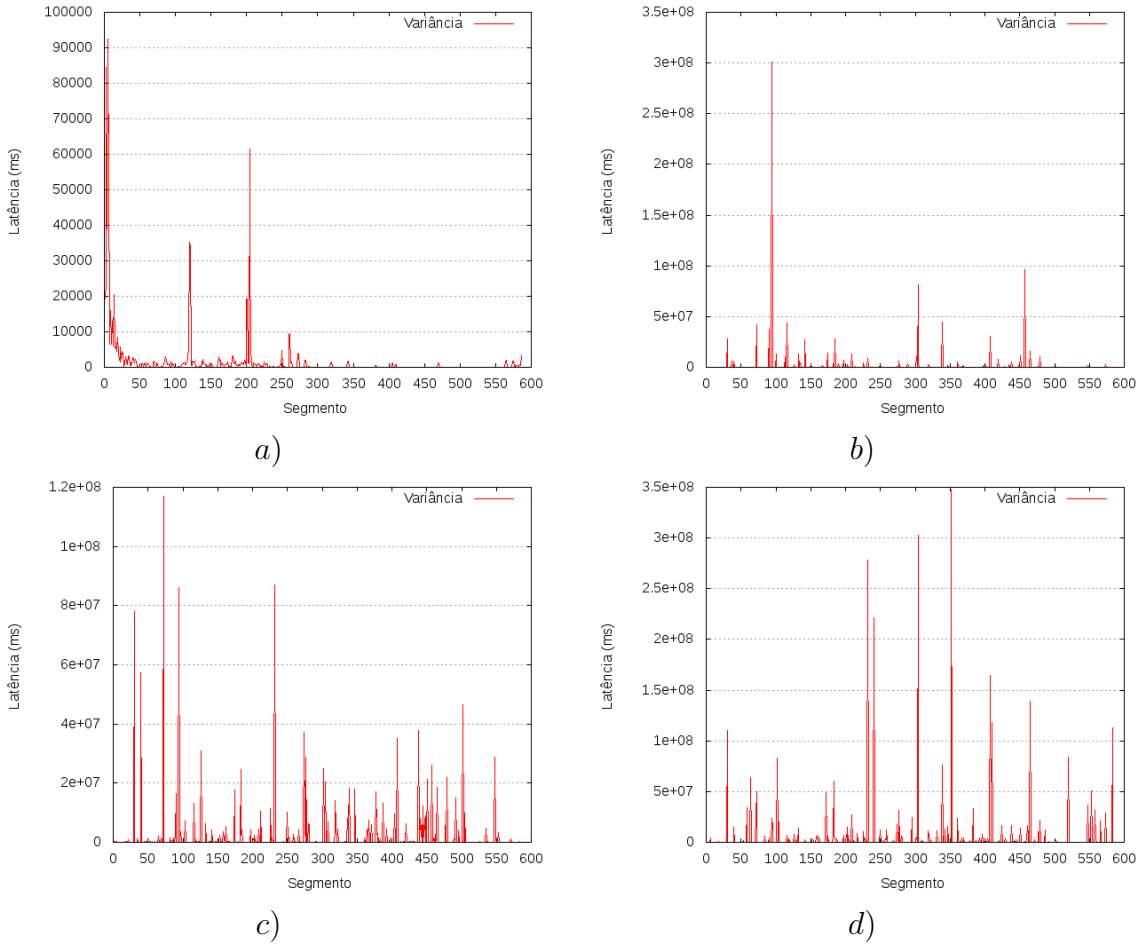

Figura B.1: Variância para um servidor centralizado com buffer tamanho 1 e taxa de chegada de 1 cliente por segundo. a) Rede com 15 clientes; b) Rede com 20 clientes; c) Rede com 25 clientes; d) Rede com 30 clientes

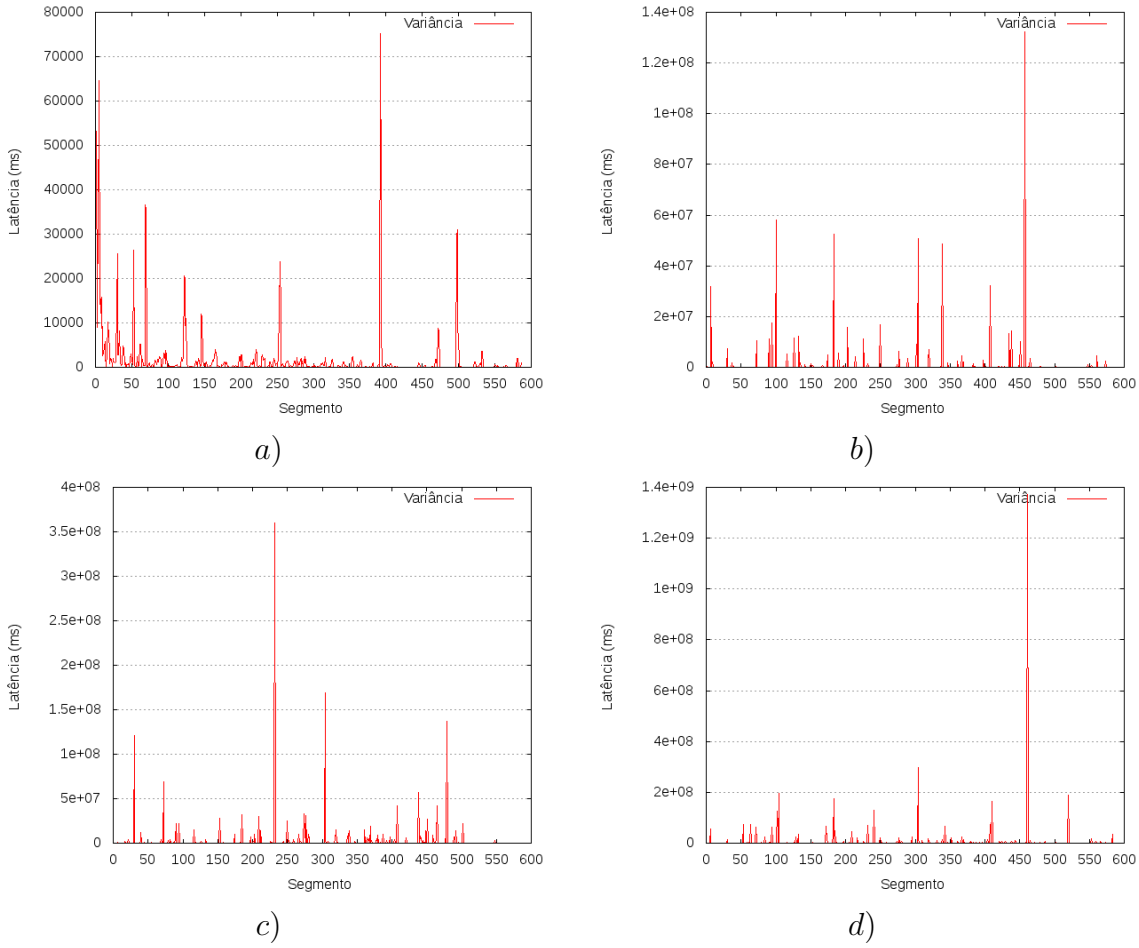

Figura B.2: Variância para um servidor centralizado com buffer tamanho 1 e taxa de chegada de 1 cliente a cada 4 segundos. a) Rede com 15 clientes; b) Rede com 20 clientes; c) Rede com 25 clientes; d) Rede com 30 clientes

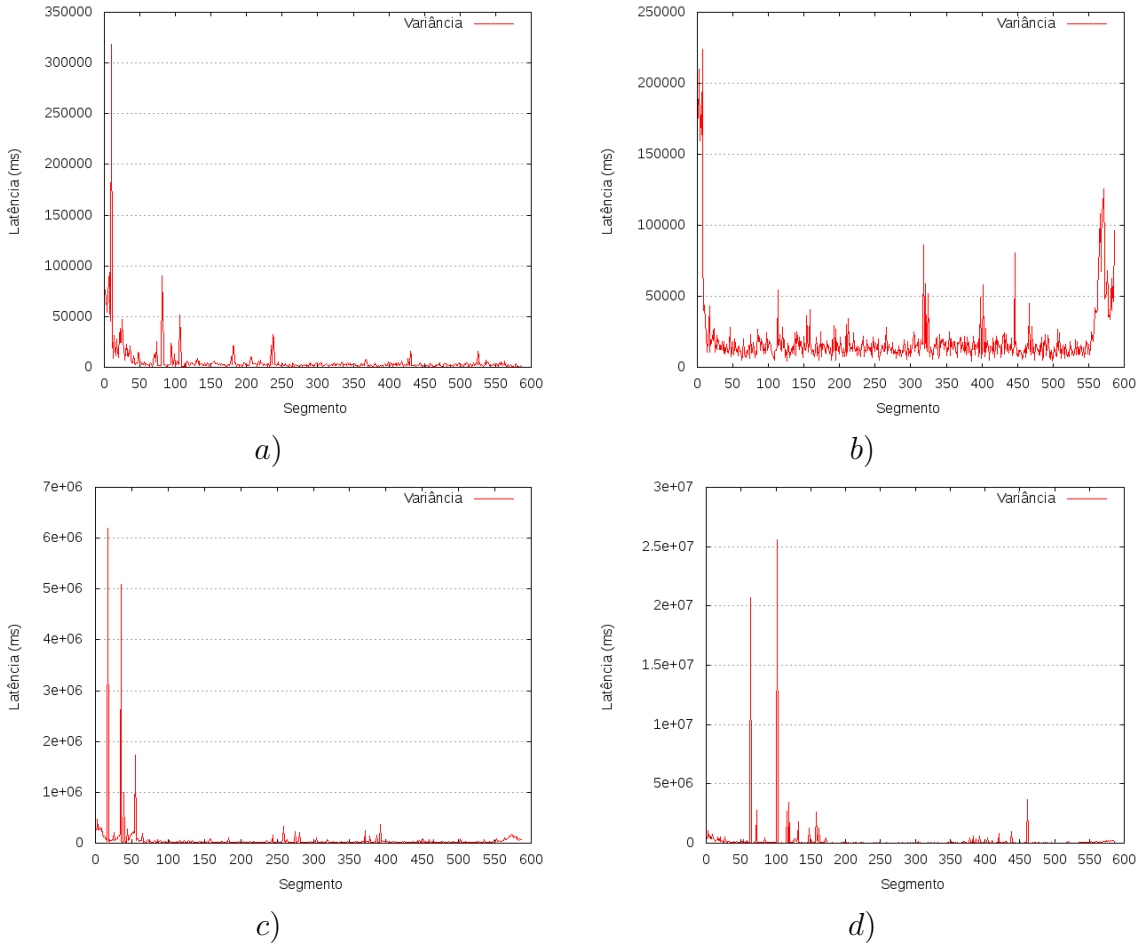

Figura B.3: Variância para um servidor centralizado com buffer tamanho 4 e taxa de chegada de 1 cliente por segundo. a) Rede com 15 clientes; b) Rede com 20 clientes; c) Rede com 25 clientes; d) Rede com 30 clientes

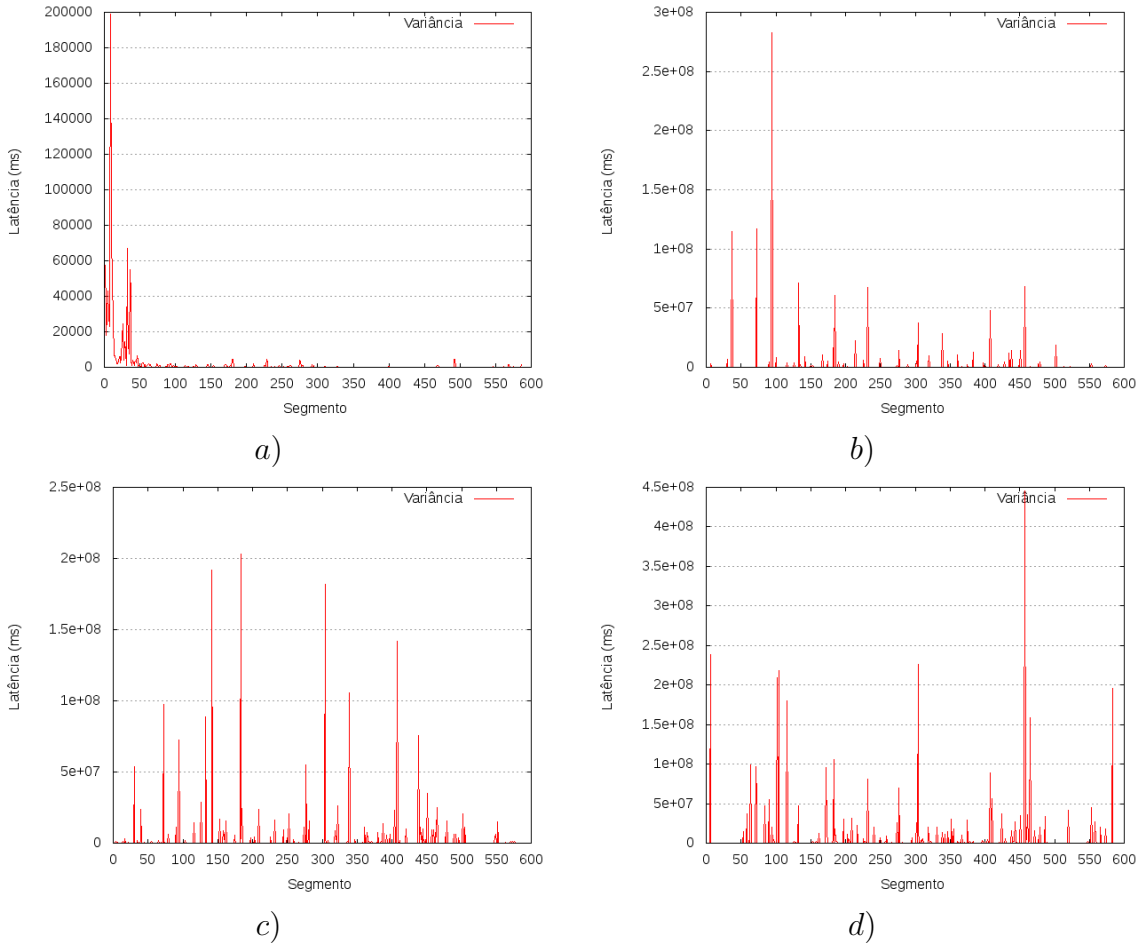

Figura B.4: Variância para um servidor centralizado com buffer tamanho 4 e taxa de chegada de 1 cliente a cada 4 segundos. a) Rede com 15 clientes; b) Rede com 20 clientes; c) Rede com 25 clientes; d) Rede com 30 clientes

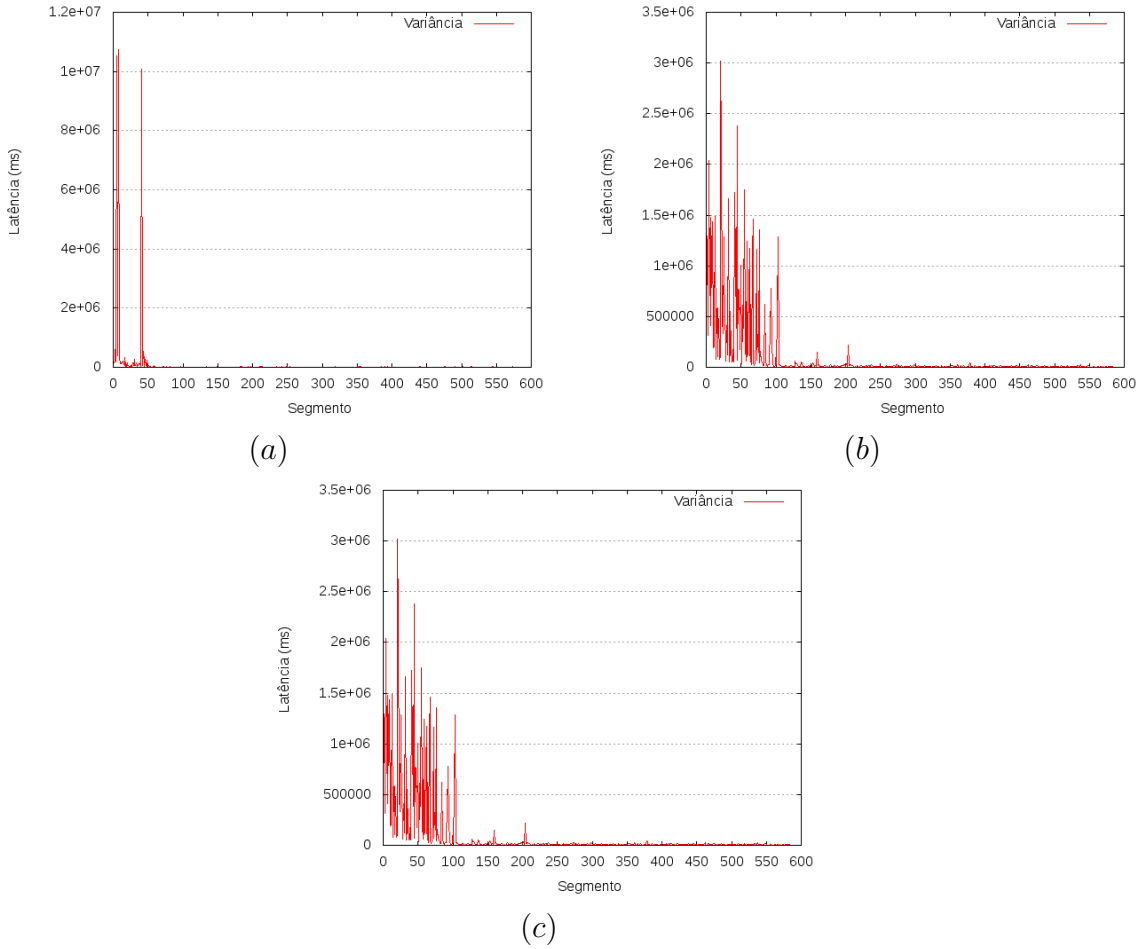

Figura B.5: Variância para a aplicação distribuída com buffer tamanho 1 e taxa de chegada de 1 cliente por segundo. (a) 54 nós; (b) 72 nós; (c) 81 nós.

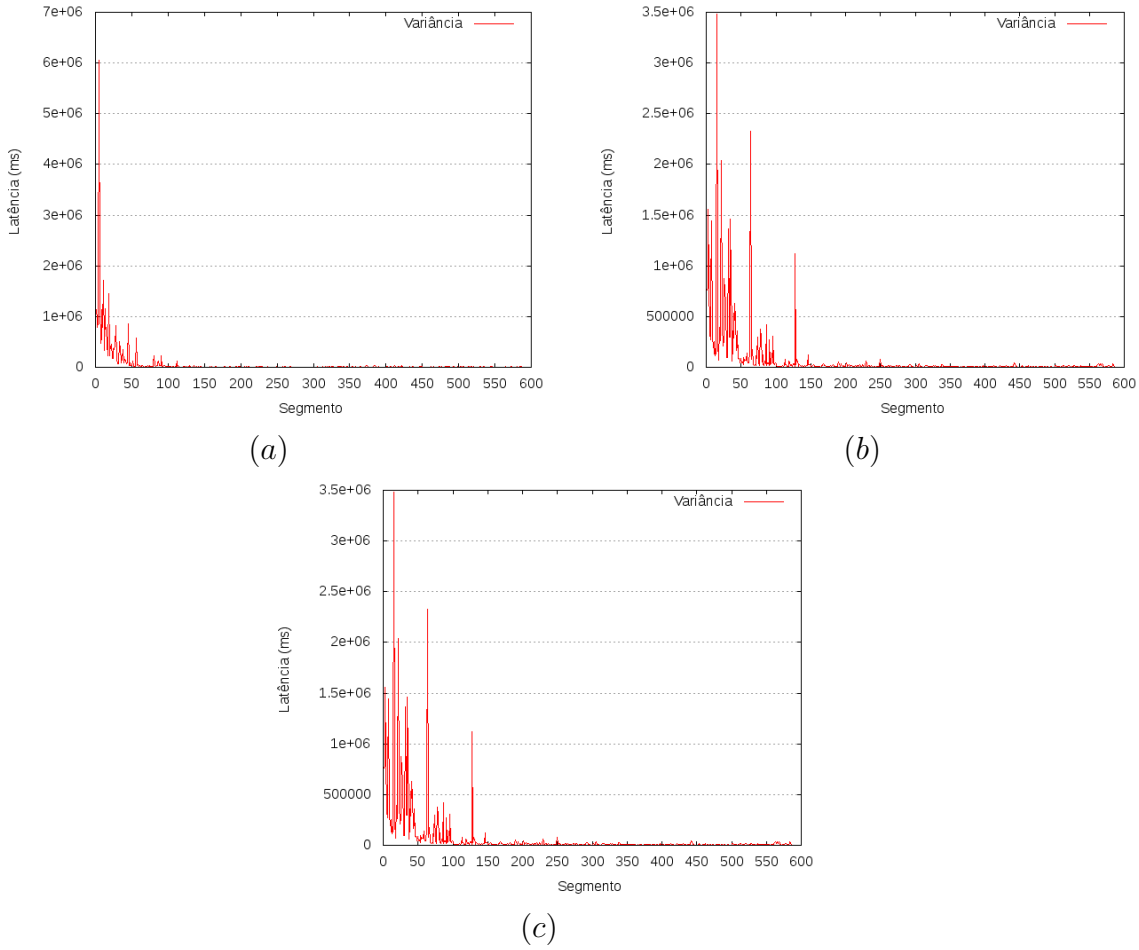

Figura B.6: Variância para a aplicação distribuída com buffer tamanho 1 e taxa de chegada de 1 cliente a cada 4 segundos. (a) 54 nós; (b) 72 nós; (c) 81 nós.

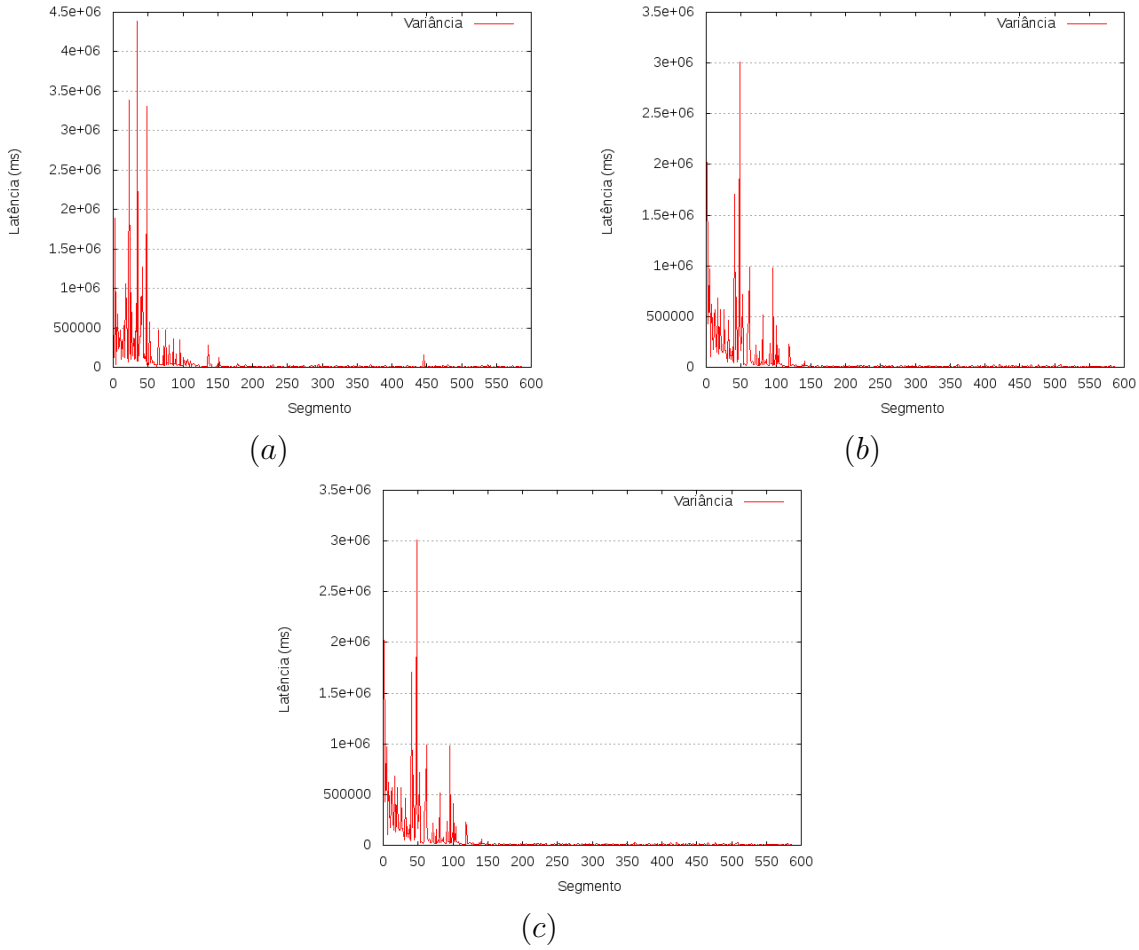

Figura B.7: Variância para a aplicação distribuída com buffer tamanho 4 e taxa de chegada de 1 cliente por segundo. (a) 54 nós; (b) 72 nós; (c) 81 nós.

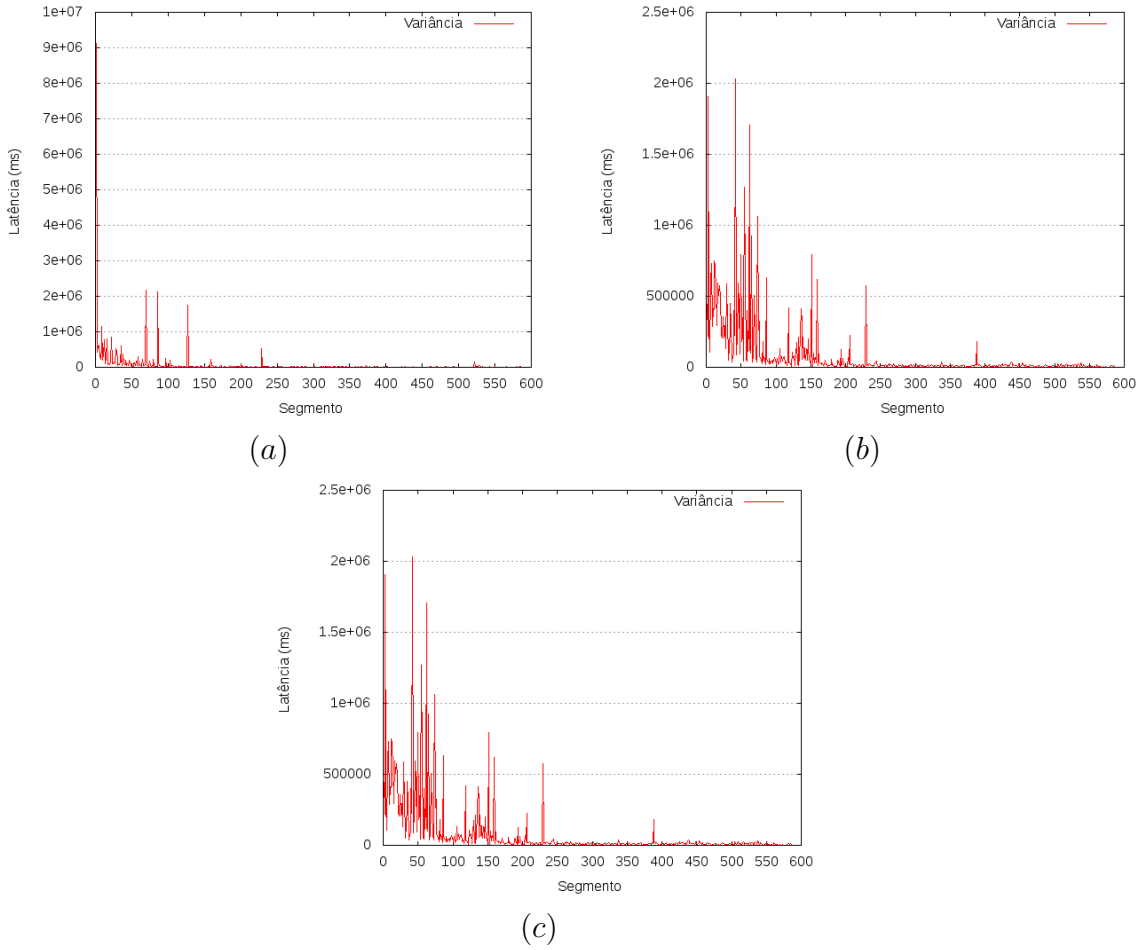

Figura B.8: Variância para a aplicação distribuída com buffer tamanho 4 e taxa de chegada de 1 cliente a cada 4 segundos. (a) 54 nós; (b) 72 nós; (c) 81 nós.