DANIEL TETSUO HUZIOKA

UM PROTOCOLO ALM BASEADO EM DESIGUALDADE TRIANGULAR PARA DISTRIBUIÇÃO DE CONTEÚDO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Elias P. Duarte Jr.

CURITIBA

2010

DANIEL TETSUO HUZIOKA

UM PROTOCOLO ALM BASEADO EM DESIGUALDADE TRIANGULAR PARA DISTRIBUIÇÃO DE CONTEÚDO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Elias P. Duarte Jr.

CURITIBA

2010

SUMÁRIO

Resumo

A comunicação por difusão seletiva, ou *multicast*, permite que uma única transmissão seja feita para um grupo de destinatários. Aplicações de distribuição de conteúdo em larga escala na Internet demandam este tipo de comunicação. Por não haver amplo suporte à comunicação *multicast* na camada de rede da Internet, alguns protocolos (chamados protocolos ALM *Application Layer Multicast*) simulam a camada de rede através de redes sobrepostas na camada de aplicação, permitindo o uso da comunicação *multicast* sobre essas redes. Neste trabalho apresentamos o PALMS, um protocolo ALM voltado para a distribuição de conteúdo contínuo proveniente de uma única fonte. Os objetivos principais do PALMS são a distribuição de conteúdo utilizando poucas mensagens de controle e a possibilidade de implantação na estrutura atual da Internet. A arquitetura do PALMS consiste de um servidor responsável pela geração de conteúdo, um *tracker* responsável pelo auxílio à entrada de nós na rede e pelos *peers*, que além de consumirem o conteúdo também são responsáveis por propagá-lo pela rede. Os *peers* se organizam em grupos, e acordos de compartilhamento nos grupos determinam de quem o *peer* recebe e para quais *peers* o fluxo deve ser transmitido.O PALMS conta com um mecanismo que permite aos *peers* realizarem trocas de acordos a fim de diminuir o tempo de recebimento do conteúdo. Este mecanismo utiliza o *round-trip-time* (RTT) entre os *peers*, e baseia-se no conceito de desigualdades triangulares. Quando um *peer* atesta que, ao trocar os acordos de recebimento e transmissão com seu *peer* transmissor, o tempo de recebimento das mensagens diminui, os acordos são trocados assim como e as posições dos *peers* no grupo. O protocolo foi implementado no simulador PeerSIM utilizando a linguagem Java, e comparado a outros protocolos de distribuição de conteúdo. Experimentos demonstram que, além do PALMS apresentar-se como uma solução factível na Internet hoje, ele também utiliza menor largura de banda para mensagens de controle que as alternativas apresentadas.

Abstract

Multicast communication allows a single transmission to be received by a group of nodes. Large-scale content distribution applications running on the Internet demands this kind of communication. As there is no broad support to multicast at the Internet network layer, some protocols (called ALM protocols) simulate the network layer through overlay networks created at the application layer, allowing the use of multicast over those networks. This paper presents PALMS, an ALM protocol focused on single-sourced continuous content distribution. The main objectives of PALMS are the distribution of content using fewer control messages than other protocols and the ability of deployment in the current structure of the Internet. PALMS architecture consists of a server, responsible for content generation, a tracker responsible for peer joining bootstrap mechanism and the peers, the final clients of the network and also response for content propagation. Peers are organized into groups, and sharing agreements in groups determines who the peer should receive from and to which peers it should propagate the content. This paper also proposes a mechanism that allows peers to swap agreements in order to reduce the content receiving delay. This mechanism uses the round-trip-time (rtt) among peers, and is based on the concept of triangular inequalities. When a peer verifies that, upon agreement swap with it's sending node, the content receiving delay is lower to all nodes involved in the process,their agreements are changed and so their positions in the group. The protocol was implemented in the PeerSIM simulator using JAVA language, and compared to other content distribution protocols. Experiments show that, besides PALMS presents a solution that can be implemented on the Internet today, it also incurs less overhead bandwidth among peers and server.

Capítulo 1

Introdução

A distribuição de fluxo (do inglês *stream*) vem ganhando espaço na Internet à medida que as redes permitem a transferência de conteúdo a taxas de transmissão cada vez maiores. Atualmente, é possível assistir a vídeos online ou ouvir uma rádio com qualidade considerável, sendo o conteúdo solicitado sob demanda ou transmitidos em tempo real. Há diversas aplicações para a distribuição de conteúdo, variando desde portais de distribuição a redes totalmente descentralizadas. Entretanto, a estrutura atual da Internet impõe algumas restrições no envio e recebimento de mensagens. Este trabalho apresenta um novo protocolo de distribuição de conteúdo contínuo utilizando *multicast* em nível de aplicação. O protocolo é descrito, especificado e foi implementado no ambiente de simulação PeerSim.

Esta introdução descreve a motivação e os conceitos básicos necessários ao trabalho, estando organizada da maneira a seguir. A seção 1.1 descreve métodos utilizados para a distribuição de conteúdo. A seção 1.2 descreve a importância da comunicação *multicast*. A seção 1.3 apresenta *multicast* utilizado na camada de aplicação. A seção 1.4 apresenta os objetivos e a contribuição esperada deste trabalho. A seção 1.5 apresenta a organização do restante deste trabalho.

1.1 Distribuição de Conteúdo

A distribuição de conteúdo é uma das principais atividades na Internet, sendo responsável por grande parte de seu tráfego [1]. O modelo mais básico constitui-se na distribuição de conteúdo através de um servidor [2], que se encarrega de transmitir o conteúdo a cada cliente que o requisita. Para distribuições maiores que envolvam um maior número de

usuários, o modelo cliente-servidor passa a não ser uma boa alternativa, sendo necessário um modelo mais escalável. Desta demanda, surgiu o modelo *Peer-to-Peer* (P2P) [3], que utiliza-se dos nós presentes na rede para auxiliar na transmissão e propagação do conteúdo, diminuindo a carga imposta ao servidor.

Outra abordagem utilizada para categorizar sistemas de distribuição de conteúdo diz respeito ao tipo conteúdo distribuído. Conteúdo estático diz respeito a distribução de conteúdo disponível desde o início da transmissão. Quando o conteúdo não encontra-se disponível no início da transmissão, sendo compartilhado a medida que é gerado [4], dá-se o nome de conteúdo contínuo. Este trabalho apresenta uma nova solução para distribuição de conteúdo contínuo, considerando um sistema com características do modelo P2P.

1.2 Comunicação Multicast

Nas redes de computadores, é possível classificar a comunicação de acordo com a forma de roteamento das mensagens e quantidade de destinatários. Na comunicação ponto-a-ponto (do inglês, *unicast*), um transmissor envia uma mensagem para apenas um receptor. Na difusão (do inglês, *broadcast*), um transmissor envia uma mensagem a todos os outros nós da rede. Já na difusão seletiva [5] (do inglês, *multicast*), um transmissor envia uma mensagem para um grupo de receptores.

Para a distribuição de conteúdo contínuo, a transmissão *unicast* pode não ser a mais eficiente, quando empregada de maneira arbitrária. Em transmissões de conteúdo contínuo ao vivo, por exemplo, diversos receptores requisitam o mesmo conteúdo à fonte. Utilizando comunicações *unicast*, o transmissor envia múltiplos pacotes idênticos, um para cada receptor. No *multicast*, o transmissor enviaria apenas uma mensagem, sendo esta replicada apenas quando necessário.

Uma alternativa apresentada reside na transmissão *multicast* presente no IP (*Internet Protocol*) *Multicast* [6], onde a replicação das mensagens ocorre apenas quando necessário a critério dos roteadores. Roteadores experimentais pertencentes ao *Multicast Backbone* (MBone) [7] suportam IP *Multicast*, mas sua implantação hoje se resume a uma pequena escala de organizações e instituições.

1.3 Multicast em Nível de Aplicação

No modelo da Internet [2], a camada responsável pelo roteamento das mensagens é a camada de rede. Como na Internet a maioria dos roteadores possui suporte apenas à comunicação *unicast*, resta buscar alternativas à comunicação *multicast*.

Na camada de aplicação, podem ser implementadas todas as lacunas das outras camadas. É possível simular o comportamento de outras camadas neste nível, inclusive o modelo de transmissão de mensagem *multicast*. Uma série de protocolos utiliza a camada de aplicação para simularem a comunicação *multicast* (que serão apresentados no capítulo 3). Estes protocolos, chamados protocolos ALM (Difusão Seletiva em Nível de Aplicação, do inglês *Application Layer Multicast*), criam uma rede sobreposta na camada de aplicação, criando conexões virtuais entre os nós como as encontradas em [8]. Com essa rede, é possível aos nós simularem transmissões *multicast*, sem a necessidade de suporte da camada de rede. Este trabalho utiliza os conceitos de ALM para a criação do protocolo, assim como apresenta algumas abordagens na área.

1.4 Objetivos e Contribuição deste Trabalho

O objetivo deste trabalho é criar um protocolo escalável que permita a distribuição de conteúdo contínuo na Internet, mas que utilize poucas mensagens de controle. Para o refinamento da topologia, utiliza-se de um mecanismo baseado em desigualdades triangulares. Esse conceito é baseado no fato que, dado um triângulo qualquer, o comprimento da aresta formada por dois vértices quaisquer é inferior à soma dos comprimentos das outras duas arestas.

Nas redes de computadores, a topologia de uma rede pode ser representada por um grafo com peso nas arestas. Neste trabalho os pesos representam o tempo de ir-e-vir(RTT, do inglês *round-trip-time*) entre nodos da rede, sendo possível aplicar resultados da geometria sobre eles. Um conjunto de três arestas num grafo podem não formar um triângulo, e o mecanismo apresentado neste trabalho procura este conceito para o refinamento da topologia da rede.

Além da definição do protocolo proposto, é objetivo deste trabalho sua implementação em um ambiente de simulação que permita sua comparação com outras alternativas.

1.5 Organização do Trabalho

O restante deste trabalho está organizado da seguinte maneira. O capítulo 2 descreve alguns dos principais modelos de distribuição de conteúdo na Internet, assim como apresenta uma distinção entre os modelos utilizados para fins semelhantes. O capítulo 3 apresenta a comunicação *multicast*, sua condição atual na Internet e alguns protocolos relacionados. O capítulo 4 apresenta um novo protocolo ALM para distribuição de conteúdo na Internet, o PALMS, assim como um protótipo e sua avaliação experimental. O capítulo 5 conclui este trabalho, apresentando algumas considerações sobre a avaliação experimental e alguns trabalhos futuros.

Capítulo 2

Distribuição de Conteúdo

A distribuição de conteúdo diz respeito ao fornecimento e disponibilização de conteúdo na rede. Em relação ao tipo de conteúdo, podemos dividí-los em conteúdo estático ou conteúdo contínuo [9]. Conteúdo estático é aquele que se encontra previamente disponível e que pode, a priori, ser acessado de maneira assíncrona, sendo sua distribuição realizada sob demanda pelos usuários. Conteúdo gerado à medida que a transmissão ocorre é chamado de conteúdo contínuo, e sua distribuição se dá por difusão para os usuários. Há grandes diferenças entre as duas categorias, sobretudo na disponibilidade do conteúdo, nas técnicas utilizadas para a melhoria da distribuição e em suas restrições.

Neste capítulo, são apresentados as duas categorias de conteúdo, estático e contínuo, assim como dois modelos empregados para sua distribuição, o modelo cliente-servidor e o modelo *Peer-to-peer*.

2.1 Distribuição de Conteúdo Estático

A distribuição de conteúdo estático refere-se à transmissão onde o conteúdo encontra-se totalmente disponível no início da transmissão. Sendo assim, ao menos o distribuidor inicial do conteúdo precisa ter disponibilidade do conteúdo na íntegra. Essa asserção, caso não haja restrições específicas da aplicação, faz com que o conteúdo possa ser recebido fora de ordem, permitindo até a manipulação do conteúdo recebido pelo receptor. Um exemplo de manipulação por parte do receptor está presente na distribuição de vídeo sob demanda, onde é possível ao receptor manipular e controlar o fluxo recebido, de forma similar ao controle existente na reprodução de um DVD [10]. Estes controles podem incluir, por exemplo, pausar a reprodução, comandos de avançar e retroceder, dando maior liberdade

ao receptor quanto ao conteúdo recebido. Entretanto, essa liberdade pode resultar em um aumento na complexidade de transmissão do conteúdo.

Como os receptores possuem controle total sobre o conteúdo, qualquer mudança de conteúdo pelo receptor deve ser atendida pela fonte de dados. Um problema que surge na distribuição sob demanda que não está presente na distribuição de conteúdo contínuo é a disparidade do conteúdo reproduzido por diferentes usuários. Na distribuição de conteúdo contínuo todos os usuários estão interessados no mesmo conjunto de dados sendo disponibilizados pela fonte de dados. Em sistemas de distribuição sob demanda, cada usuário é livre para escolher a janela de conteúdo que o convém (através das funções descritas no parágrafo anterior). No pior caso, cada usuário estará interessado em uma janela específica, sendo necessário responder a cada uma das requisições de forma independente. Em grande parte dos casos, há uma parcela de usuários interessados na mesma janela, ou em janelas próximas. Algumas técnicas, como o *Batching* [11] e o *Bridging* [12] foram criadas para tirar vantagens dessas situações.

2.2 Distribuição de Conteúdo Contínuo

A distribuição de conteúdo contínuo refere-se a transmissão onde o conteúdo não se encontra totalmente disponível no início da transmissão. Para sua distribuição, uma fonte de dados gera o conteúdo de forma contínua, como um fluxo de dados, disponibilizando o conteúdo a medida que ele é gerado. Sendo assim, os receptores que recebem o conteúdo podem apenas consumi-lo, sem grandes possibilidades de manipulação ou controle de *playback* como na distribuição de conteúdo sob demanda, apenas sobre o conteúdo previamente recebido [9]. Esta técnica requer menor complexidade de transmissão e encaminhamento das mensagens, uma vez que os receptores estarão interessados no mesmo conteúdo, tendo apenas uma pequena diferença de tempo (janela) no conteúdo reproduzido devido ao atraso ocasionado pela rede.

Caso um receptor conecte-se na rede após o início da transmissão, vários cenários podem ocorrer. Trivialmente, assim como ocorre em transmissões de TV por difusão, o receptor passa a receber o conteúdo gerado a partir do momento de sua inserção, sem

ter acesso ao conteúdo previamente transmitido. Outra alternativa é o receptor passar a receber o conteúdo gerado a partir do momento de sua inserção, mas também receber toda a transmissão anterior à sua entrada, permitindo assim a reprodução do conteúdo como um histórico. Uma alternativa pouco utiliza, mas factível, é o início da transmissão do zero, onde o receptor, ao entrar na rede, começa a receber o conteúdo do início da transmissão.

Nos casos em que o receptor adquire conteúdo previamente transmitido, a distribuição assemelha-se a distribuição de conteúdo estático até que o receptor alcance o ponto de transmissão atual (através de pulos na execução, por exemplo), onde passa então a receber o conteúdo corrente.

Apesar de não possibilitar a manipulação em tempo real do conteúdo, o receptor pode armazenar o conteúdo em um *buffer*, que pode ser utilizado posteriormente para manipulação do conteúdo em histórico. O emprego do *buffer* é importante pois quase todas as redes, em especial a Internet, estão sujeitas a falhas e oscilações. Não é possível garantir que o conteúdo recebido será contínuo e entregue em períodos constantes, podendo haver variações entre a entrega de um pacote e outro [13]. Sendo assim, a existência de um *buffer* é importante não apenas para uma possível manipulação do conteúdo, mas para prover maior robustez ao protocolo [14].

Quando a transmissão é puramente contínua, sem mecanismos específicos para uma possível distribuição sob demanda, a complexidade do sistema tende a diminuir. Como visto anteriormente, quando há apenas a distribuição por difusão todos os receptores encontram-se no mesmo ponto de execução e estão interessados em receber o mesmo conteúdo, facilitando a distribuição pela fonte. Este é o cenário ideal para a utilização da comunicação *multicast*, vista em detalhes no próximo capítulo.

2.3 Modelo Cliente-Servidor

O modelo cliente-servidor é um modelo de comunicação onde os integrantes da rede, clientes e servidores, possuem papéis bem definidos e distintos entre si [2, 15]. Neste modelo, cabe ao servidor a tarefa de receber requisições e enviar dados aos clientes. Ao cliente,

cabe a tarefa de requisitar dados do servidor e consumi-los. Sua grande utilização deve-se principalmente a baixa complexidade na implementação de uma rede de distribuição. Seu funcionamento baseia-se na manipulação de cada cliente individual e independentemente. Por independente, entende-se que cada cliente recebe o conteúdo completo, em conexão direta com o servidor. Essa independência garante que cada cliente receba o conteúdo na íntegra e de maneira confiável, já que ele é enviado diretamente do servidor para o cliente sem intermediários. Essa independência por parte dos clientes permite que diferentes tipos de clientes, com diferentes capacidades de processamento e banda, conectem-se na rede, ao contrário do modelo P2P (visto a seguir) onde os clientes necessitam de certa capacidade de processamento e banda adicionais.

A estrutura lógica de uma rede baseada no modelo-cliente servidor é bem simples, sendo constituída por um (ou mais) servidores, distribuindo conteúdo a diversos clientes. Um exemplo da estrutura lógica criada para a distribuição de conteúdo no modelo cliente-servidor está exemplificada na figura 2.1.

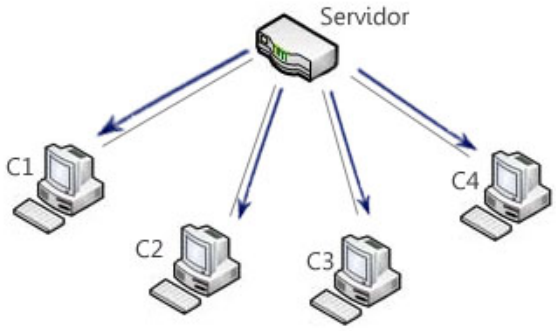

Figura 2.1: Estrutura de distribuição de conteúdo no modelo cliente-servidor.

Em diversas aplicações comerciais, essas características são interessantes, pois o objetivo é que não haja grandes atrasos no recebimento do conteúdo pelos clientes. Entretanto, essa independência também impõe uma grande carga ao servidor, uma vez que cada novo cliente significa uma nova transmissão. Quando o limite da capacidade de processamento ou banda do servidor são atingidos, novos clientes não poderão obter o conteúdo, ou a transmissão aos clientes conectados na rede será afetada.

Diversas técnicas foram propostas para diminuir a carga de transmissão imposta ao servidor no modelo cliente-servidor sem perder suas características. Uma das técnicas que mais se destaca envolve o uso de servidores *proxy* [16], onde além do servidor central, há diversos outros servidores secundários que armazenam temporariamente o conteúdo distribuído. Caso um cliente requisite um conteúdo presente em um dos *proxy cache*, este fica responsável por transmitir a informação, desonerando assim o servidor. Vários sistemas utilizam esta técnica, que possui como principais desvantagens o custo de implantação de novos servidores e o *overhead* na busca e redirecionamento de conteúdo por parte dos próprios servidores *proxy*.

2.4 Modelo Peer-to-Peer

O modelo *Peer-to-Peer* (P2P) foi criado para a distribuição de conteúdo na rede, visando o aumento da escalabilidade das redes de distribuição de conteúdo [3, 4, 17, 1, 18]. O princípio básico dessas redes é utilizar recursos disponíveis nos receptores, chamados pares ou *peers*, para retrasmitirem o conteúdo a outros receptores. Esta técnica permite que *peers* conectem-se na rede sem necessariamente imporem ao servidor o ônus de uma nova conexão, podendo o conteúdo ser obtido tanto do servidor quanto de outros *peers*, que já possuem o conteúdo desejado. A figura 2.2 exemplifica uma rede de distribuição de conteúdo no modelo P2P.

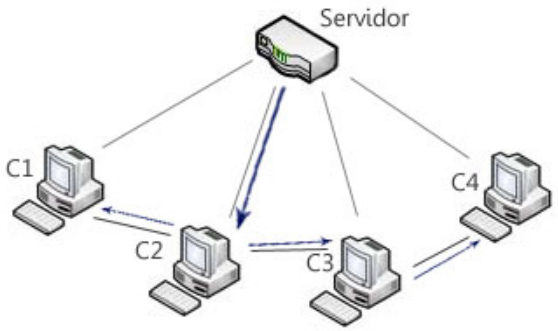

Figura 2.2: Estrutura de distribuição de conteúdo no modelo P2P.

Embora mais escalável que o modelo cliente-servidor, este modelo possui algumas

desvantagens. A adição de *peers* no caminho da transmissão adiciona pontos adicionais de falhas que podem ocasionar no não recebimento de mensagens caso não hajam mecanismos de tolerância a falhas.

Em relação ao desempenho, a comparação entre os dois modelos depende das configurações da rede. Para redes com um número reduzido de receptores, uma rede cliente-servidor pode obter maior taxa de entrega de pacotes para cada receptor, dependendo da largura de banda do servidor. Já em uma rede P2P, a dependência em relação a banda do servidor tende a ser menor, mas há dependência de disponibilidade tanto de banda como de processamento por parte dos *peer* [15]. Apesar disso, redes P2P vêm sendo responsável por grande parte do tráfego da Internet [19].

Em relação à organização da rede, as redes P2P distribuem as tarefas realizadas exclusivamente pelo servidor no modelo cliente-servidor, sendo altamente dependente do comportamento dos mesmos para funcionar corretamente. Por serem dependentes dos *peers*, protocolos que implementam o modelo P2P precisam avaliar as consequências das saídas dos *peers* ou do comportamento errôneo de algum deles, sendo necessária a inclusão de mecanismos de tolerância a falhas que não estão presentes no modelo cliente-servidor.

Um problema que surge nas redes P2P devido à descentralização da rede é a organização dos *peers*. No modelo cliente-servidor, a organização da rede se dá diretamente com o servidor, e requisições de qualquer espécie ocorrem do sentido cliente para servidor ou servidor para cliente. No modelo P2P, descentralização da rede faz necessária a adoção de métodos de organização dos *peers*, pois para um *peer* realizar uma requisição qualquer é necessário descobrir para qual *peer* essa requisição deve ser feita. Sobre os modelos utilizados para a organização dos *peers*, podemos distinguir as redes P2P em estruturadas e não-estruturadas.

Em redes P2P estruturadas, a organização dos *peers* é feita utilizando como apoio tabelas *hash*. Para permitir a implementação de forma distribuída, a tabela é implementada como uma DHT (Tabela Hash Distribuída, do inglês *Distributed Hash Table*), onde cada *peer* fica responsável por um conjunto de chaves. O desafio dos protocolos que utilizam uma arquitetura estruturada é implementar um método eficiente de distribuição da DHT

e um método eficiente de mapeamento e descobrimento dos *peers* e suas chaves. Soluções como [20, 21, 22] implementam mecanismos de atribuição e descoberta de chaves em redes P2P.

Em redes P2P não estruturadas, a organização dos *peers* se dá de maneira arbitrária. Por não possuir um processo determinístico, não é possível, na maior parte dos casos, prever a organização dos *peers* resultante. Quando há a necessidade de buscas de conteúdo nessas redes, utilizam-se usualmente processos semelhantes a inundações na rede ou propagação epidêmica [23]. Por serem menos complexas que as redes P2P estruturadas, o modelo não-estruturado é normalmente utilizado quando há uma entidade de apoio, como por exemplo o *tracker* nas redes torrent [18], ou quando não há interesse no mapeamento de chaves.

Capítulo 3

Comunicação Multicast

A comunicação por difusão seletiva ou *multicast* é um dos diversos métodos de comunicação utilizados nas redes de computadores, e consiste na entrega de uma mesma mensagem para um grupo de receptores. Seu nome deriva de outro método de comunicação chamado difusão ou *broadcast*, que consiste no envio de uma mesma mensagem a todos nós da rede. Como o próprio nome sugere, o *multicast* seleciona um grupo de receptores e envia a mensagem a este grupo específico.

Este capítulo é organizado da seguinte maneira. A seção 3.1 apresenta uma comparação entre a comunicação *unicast* e a comunicação *multicast*. A seção 3.2 apresenta o protocolo baseado em IP para comunicação *multicast*. A seção 3.3 apresenta os conceitos sobre *multicast* em nível de aplicação (ALM, do inglês *Application Layer Multicast*), que simulam a comunicação *multicast* utilizando a camada de aplicação, assim como alguns protocolos que o implementam. A seção 3.4 apresenta outros protocolos relacionados a este trabalho.

3.1 Unicast vs. Multicast

A comunicação *unicast* é a única suportada por todos os roteadores da Internet [24]. Seu uso se deve principalmente à simplicidade no redirecionamento das mensagens, permitindo o roteamento rápido das mensagens pelos roteadores. A comparação entre o *unicast* e o *multicast* é importante pois mostra os principais desafios para a implementação do *multicast* em nível de rede.

A principal diferença entre a comunicação *unicast* e *multicast* diz respeito à distribuição da carga de envio e ponto de replicação das mensagens. Quando há o envio de

uma mesma mensagem a múltiplos receptores, no *unicast* há a replicação das mensagens na origem pelo transmissor, enquanto no *multicast* essa replicação só ocorre quando necessário pelos roteadores. Em relação à distribuição da carga de envio, ao considerarmos um mesmo conjunto de nós transmissores, intermediários e receptores, a comunicação *unicast*, por enviar uma mensagem diferente a cada receptor, impõe carga de n mensagens a cada nó intermediário, onde n é o número total de receptores. Já a comunicação *multicast*, por adiar a replicação das mensagens, impõe carga de no máximo n, igualando-se à comunicação *unicast* apenas quando cada nó receptor não possui nós intermediários em comum.

A figura 3.1 mostra a diferença de carga imposta a um nó intermediário r_1 na utilização do *unicast* e *multicast* para o envio de uma mensagem de s_1 para os nós c_1, c_2 e c_3. Como na comunicação *unicast* uma mensagem específica deve ser enviada a cada destinatário, s_1 deverá enviar 3 mensagens. Caso a comunicação *multicast* seja utilizada, s_1 enviará apenas 1 mensagem, cabendo a r_1 a tarefa de replicar estas mensagens aos destinatários.

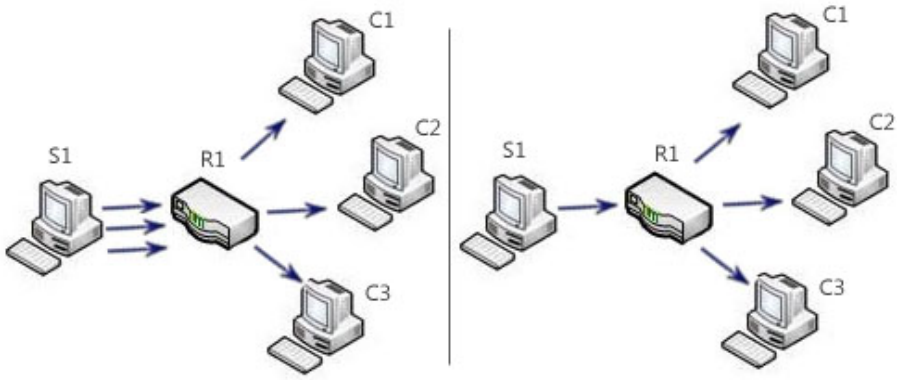

Figura 3.1: Comparação da comunicação *unicast* (à esquerda) e *multicast* (à direita), possuindo nós intermediários na transmissão.

É possível inferir que, a medida que mais nós intermediários são inseridos no caminho a diferença de utilização de banda entre a comunicação *unicast* e *multicast* tende a aumentar, apenas comprovando a superioridade no uso do *multicast* para o envio simultâneo à múltiplos destinatários. No restante deste capítulo são apresentadas soluções para o desenvolvimento de protocolos *multicast* compatíveis com a Internet.

3.2 IP Multicast

O IP Multicast é um conjunto de regras adicionais ao protocolo IP, de modo a permitir a comunicação para múltiplos destinos simultâneos. Cada versão possui uma maneira distinta de suporte à comunicação *multicast*.

Na versão 6, especificada pelo RFC 2460 [25], o *multicast* é suportado nativamente através de um block especial no datagrama, chamado *multicast block*. O identificador de um *multicast block* corresponde a 1/256 do espaço destinado ao endereçamento das máquinas, onde todos os valores correspondem a 1 (*1111 1111* ou *FF* em hexadecimal). Sendo assim, todos os endereços IPv6 iniciados por esses valores são endereços multicast.

Embora nativamente suportado pela versão 6, o termo *IP Multicast* usualmente refere-se à implementação da comunicação *multicast* sobre a versão 4 do protocolo, por esta ser a versão atualmente suportada pelos roteadores na Internet.

Na versão 4, o RFC 1112 [26] propõe um método de endereçamento dos grupos *multicast*. Os identificadores de grupos multicast utilizam 1/16 do espaço destinado ao endereçamento IP e correspondem aos endereços iniciados por *1110*, os endereços da antiga Classe D [27]. Endereços IPv4 para grupos *multicast* são atribuídos de acordo com o RFC 5771 [28], dependendo principalmente de seu escopo de utilização.

A inserção ou remoção de membros em um grupo *multicast* é especificada pelo protocolo IGMP (*Internet Group Management Protocol*), que atualmente encontra-se em sua versão 3 [6]. Quando possível, é utilizado um mapeamento especial de endereço IP para endereço MAC (*Medium Access Control*), definido pelo RFC 1112 [26]. Isso evita o envio de múltiplas mensagens *unicast* até mesmo na camada de enlace.

Embora factível, a comunicação *multicast* não é implementada nos roteadores da Internet por diversos problemas. Problemas como o envio de mensagens de destinatários não solicitados já possuem solução [6], mas ainda há diversos problemas em aberto [24].

3.3 Protocolos ALM

Os protocolos ALM são protocolos voltados à comunicação *multicast* utilizando apenas recursos da camada de aplicação da Internet e sendo independentes da camada de rede. Essa independência se dá através da simulação de transmissões *multicast* no nível de aplicação para mensagens *unicast* em nível de rede. Para simular uma transmissão *multicast*, usa-se a capacidade de envio dos membros participantes, de forma semelhante ao modelo P2P visto anteriormente, que são responsáveis por propagarem a mensagem. Diversos protocolos [29, 30, 31, 32, 33, 34, 35, 36] utiliza esta técnica, como veremos a seguir. A figura 3.2, mostra um exemplo de funcionamento dos protocolos ALM, onde uma mensagem enviada por w_1 deve alcançar todos os nós presentes na rede. Para isso, w_1 transmite à w_2, que recebe a mensagem e propaga-a para n_1, que ao recebê-la propaga à n_2, e assim por diante. A figura 3.3 mostra como a mesma transmissão seria efetuada utilizando-se apenas comunicações *unicast*. Por último, a figura 3.4 demonstra o cenário ideal onde a comunicação *multicast* é implementada pela camada de rede, onde a replicação ocorre apenas quando estritamente necessária pelos roteadores.

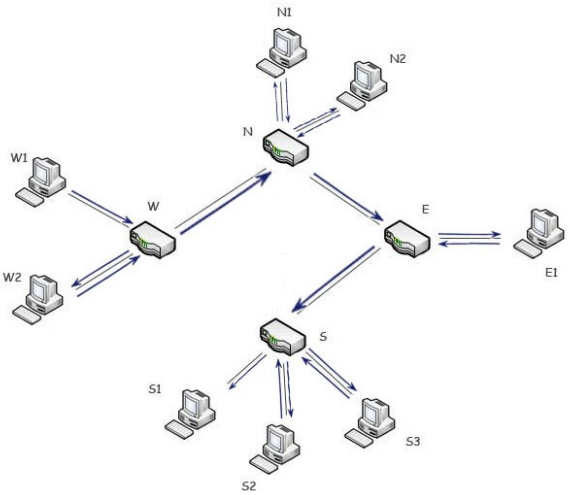

Figura 3.2: Exemplo de propagação de mensagem em protocolos ALM.

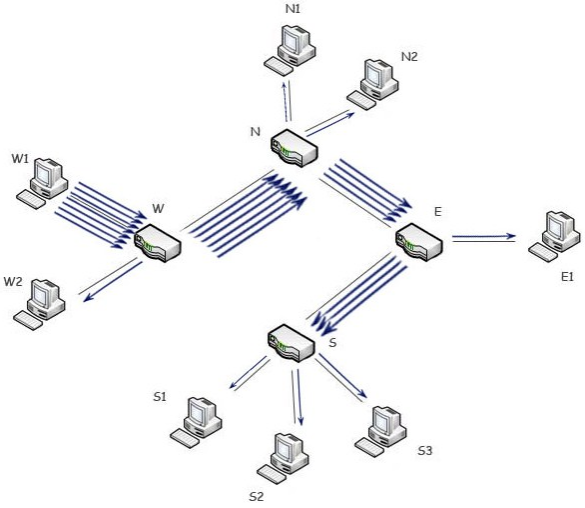

Figura 3.3: Exemplo de propagação de mensagem utilizando exclusivamente comunicações *unicast*.

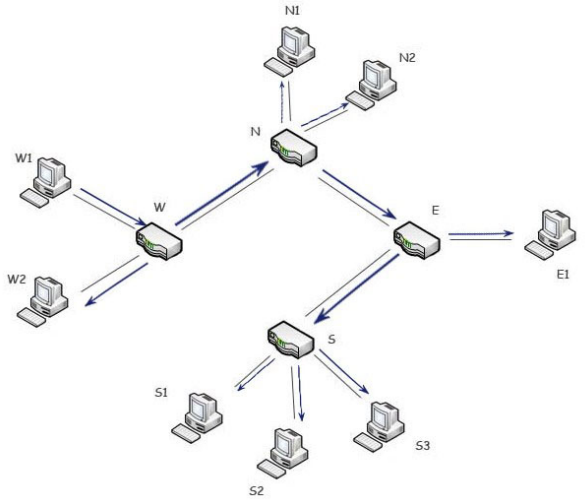

Figura 3.4: Exemplo de propagação de mensagem utlizando comunicação *multicast* com suporte da camada de rede.

Por não necessitar do suporte da camada de rede, os protocolos ALM podem ser utilizados na estrutura atual da Internet sem a necessidade de modificações. Apesar disso, os protocolos ALM usualmente modificam ou adicionam elementos à camada de aplicação. Esses elementos constituem-se de camadas sobrepostas às camadas existentes na Internet, responsáveis por proverem informações sobre os nós conectados à rede e também por distribuir o conteúdo.

As métricas comumente utilizadas para a comparação de protocolos ALM dizem res-

peito à qualidade da rede sobreposta criada, como por exemplo *stress* e *stretch* [37]. O *stress* mede a replicação de pacotes transmitidos por um mesmo nó na rede, sendo sempre de valor 1 para os membros de uma rede cujo *multicast* é suportado à nivel de rede. O *stretch* é a razão entre o tamanho do caminho percorrido pelo pacote no protocolo ALM sobre o tamanho do caminho de uma mensagem *unicast* entre os nós, sendo sempre de valor 1 nas mensagem *unicast*. Embora muito utilizadas, essas métricas dependem de informações da camada de rede nem sempre disponíveis. Outras métricas mais subjetivas, como a sobrecarga imposta aos nós e a diferença entre a latência fim-a-fim e utilizando o caminho ALM, são mais comumente utilizadas por poderem ser avaliadas em nível de aplicação.

Nesta seção, são apresentados os funcionamentos dos protocolos EMMA [32], YOID [30], NICE [31] e Narada [29].

3.3.1 EMMA/EMMA-QoS

O protocolo EMMA [32] (Multicast em nível de usuários para aplicações multilaterais, em inglês*End-user Multicast for Multi-party Applications*) é um protocolo ALM destinado a aplicações com diversas fontes de conteúdo. Apesar de funcionar em sistemas com apenas uma fonte de conteúdo, os recursos providos pelo protocolo são melhores utilizados em aplicações que efetivamente utilizem várias fontes, como por exemplo uma reunião virtual ou uma vídeo conferência.

Funcionamento Básico

O funcionamento básico do EMMA consiste em manter uma topologia de controle capaz de derivar árvores de abrangência específicas para cada membro da rede. Tendo conhecimento de todos os nós, uma entidade chamada *lobby server* mantém informações como IP e porta dos nós, assim como informações sobre o conteúdo produzido por cada uma delas (taxa de bits, codificação, por exemplo). Utilizando alguns mecanismos preemptivos, os *peers* ficam encarregados de escolherem quais conteúdos ele ficará responsável por propagar.

Inserção na Rede

A entrada de nós na rede é dado através de um *lobby server*, que mantém informações de todos os nós presentes na rede. Essas informações compreendem IP, porta, identificação do conteúdo produzido, entre outros. O *lobby server* se assemelha ao *rendezvous point* encontrado em outros protocolos ALM.

Comunicando-se com o *lobby server*, um nó que deseja conectar-se à rede requisita a lista de nós presentes na rede, verificando o tempo de ir-e-vir das mensagens para cada um desses nós e requisitando conexões com os nós com os tempos mais baixos. Para uma requisição ser aceita, duas limitações (grau e capacidade) não podem ser violadas. Limitação de grau restringe quantas conexões um nó pode ter, e limitação de capacidade diz respeito a quantas propagações de fluxo um nó pode realizar. Essas limitações são determinadas pelos próprios nós, e não havendo violação de nenhuma delas, a requisição de inserção é aceita.

Manutenção e Gerenciamento das Camadas

Para o roteamento das mensagens, cada nó possui sua própria árvore de abrangência, construída a partir da camada de controle cuja única restrição é um valor máximo (em número de saltos ou atraso médio) que não pode ser excedido. Mensagens de atualização periódicas são utilizadas para a manutenção ou reconstrução da árvore quando necessário, sendo propagadas via inundação da rede. Essa técnica não apenas notifica a inserção de um novo nó a partir de seus vizinhos como também permite a descoberta de caminhos diferentes para os nós, podendo ser útil na saída ou falha de um nó.

Como no EMMA é possível a transmissão multilateral, é necessária uma política de gerenciamento de conteúdo para auxiliar os nós na escolha em relação à retransmissão do conteúdo de diversas fontes. Cada nó mantém uma tabela de prioridades, que contém informações sobre a origem do conteúdo recebido e qual sua respectiva prioridade de recebimento. Após um nó decidir sobre a prioridade do conteúdo, este valor é então transmitido até o nó transmissor utilizando-se do caminho inverso da transmissão original.

Características Específicas

Por se tratar de uma rede interligada de vários nós, um nó intermediário do caminho pode ser responsável pela retransmissão de mais de um conteúdo. Caso a capacidade de retransmissão desse nó intermediário seja excedida, ele compara a soma das preferências de todos os nós que estão recebendo o conteúdo em questão à soma das preferências de todos os nós requisitando o novo conteúdo. Caso a segunda seja maior que a primeira, o nó cessa a propagação do conteúdo corrente e o novo conteúdo passa a ser transmitido.

A figura 3.5 mostra um exemplo do sistema de propagação de mensagens com prioridade. Nesta figura, têm-se inicialmente os transmissores S_A e S_B transmitindo os conteúdos A e B, respectivamente. A partir do transmissor S_A, os nós C_1 e C_2 recebem o conteúdo A, com prioridade 5 cada um. Esta transmissão passa por C_5, que possui um limite de envio equivalente a um único conteúdo. Sendo assim, o nó C_3, requisitante do conteúdo B com prioridade 6, cuja transmissão obrigatoriamente passa por C_5, terá sua requisição negada pois as prioridades de C_1 e C_2 somam 10 no total. Entretanto, na entrada do membro C_4, requisitante do conteúdo B com prioridade 5, a soma das prioridades do conteúdo B (11) no nó C_5 ultrapassa a coma das prioridades do conteúdo A (10). Neste caso, C_5 interrompe a propagação de A e passa a retransmitir B.

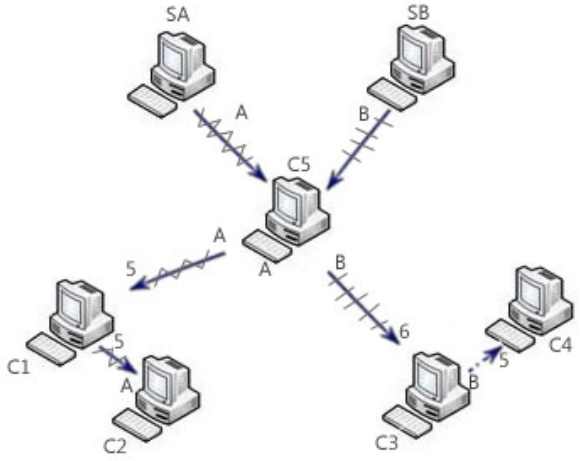

Figura 3.5: Sistema de propagação de mensagens com prioridade.

A tolerância a falhas no EMMA se restringe à correção de falhas resultantes de um nó falho. Caso um nó n deixar a rede, ou simplesmente deixar de repassar seus conteúdos

correspondentes, cada nó f ligado ao nó n que depende de n para receber conteúdo liga-se a outro nó que também possui o conteúdo em questão, mantendo a redistribuição. Este processo é possível devido às atualizações proativas do estado da camada de controle.

3.3.2 YOID

YOID [30] (Seu próprio distribuidor de internet, do inglês *Your Own Internet Distribution*), é na verdade um conjunto de protocolos, que provêem meios de distribuição de conteúdo pela Internet utilizando a capacidade de compartilhamento natural dos clientes. Adotando regras de funcionamento bastante versáteis, o YOID permite, mas não se limita a, utilização de servidores de distribuição de conteúdo e a utilização do IP Multicast [6]. Foi um dos primeiros protocolos que utilizaram a comunicação *multicast* em nível de aplicação, e é formado pelos protocolos YTMP, protocolo de gerenciamento da árvore, YDP, protocolo de distribuição e YIDP, protocolo de identificação. Uma rede YOID pode compreender um ou mais grupos de distribuição *multicast*, e iremos considerar o comportamento das entidades envolvidas em apenas um grupo.

Funcionamento Básico

O conjunto de protocolos YOID, chamados daqui em diantes apenas de YOID, utiliza-se de um *rendezvous Point* que possui informações de nós presentes na rede para auxiliar a entrada de nós na rede. O YOID procura criar inicialmente uma rede sobreposta em forma de árvore, criando uma rede sobreposta *mesh* para recuperação de eventuais falhas ou saídas de nós. Essa abordagem traz algumas vantagens como a escolha explícita de quais nós serão responsáveis por transmistir o fluxo e o controle da quantidade de envios que um nó deve realizar (calculado a partir de seu número de filhos na árvore).

Inserção na Rede

A inserção na rede YOID se dá através do *rendezvous point*, que mantém uma lista de alguns membros da rede. O endereço e porta do *rendezvous point* deve ser de conhecimento

geral dos nós a priori de sua entrada. Para ser inserido na rede, um nó x requisita sua entrada ao *rendezvous point*, recebendo uma lista de nós arbitrários. Para que um nó y, presente na lista, possa aceitar x como filho, este precisa possuir capacidade livre de compartilhamento e a inserção de x não pode criar *loops* na rede. A capacidade de compartilhamento é determinada por cada nó, e caso o nó x não encontre nenhum nó disponível, ele pode requisitar outra lista ao *rendezvous point*, ou declarar-se o nó raiz de uma árvore, processo visto a seguir. Caso o nó x encontre mais de um possível pai, ele escolherá o melhor entre eles utilizando métricas pré-definidas.

Manutenção e Gerenciamento das Camadas

Com o passar do tempo, vários nós que não conseguiram ligar-se a um pai no momento de sua inserção na rede criam novas árvores. Ainda neste cenário, nós que deixam de receber informações sobre seus pais e não conseguem ligar-se a outro pai na rede também criam novas árvores. Devido à volatilidade e dinamicidade das redes de distribuição de conteúdo [1], este cenário possui bastante semelhança com a utilização real, necessitando de um mecanismo para evitar o particionamento da árvore.

De maneira proativa, todo membro da rede busca por alguns outros membros que não são vizinhos/parentes seus na árvore, e tenta estabelecer conexões entre eles. Essas conexões, juntas com as conexões da árvore, formam a rede *mesh*, responsável pela recuperação das partições na árvore. Ainda de maneira proativa, o *rendezvous point* busca por partições da árvore e gerencia o intercalamento das mesmas, transformando a raiz de uma árvore no filho de um nó existente em outra árvore, da mesma maneira que ocorre a inserção de um novo nó.

Cada nó mantém uma lista (chamada de *rootpath*) que contém o caminho entre o nó raiz (que realiza a transmissão) até o nó em questão. Sempre que há mudanças, essa lista é retransmitida a todos os nós ao longo da árvore.

Características Específicas

Devido a inserção do nó pelo *rendezvous point* diretamente na árvore de compartilhamento, o YOID possui informações suficientes para gerenciar a estrutura da árvore de forma direta, uma vez que o *rendezvous point* possui conhecimento global de todos os nós ligados à rede. Devido à aleatoriedade dos nós enviados pelo *rendezvous point* na inserção dos nós, a estrutura resultante da árvore pode não ser a mais otimizada, podendo ser melhorada através de mecanismos de balanceamento ou redistribuição de nós.

Para melhorar o desempenho da distribuição de conteúdo, os nós periodicamente buscam por outros possíveis pais que beneficiariam a distribuição de acordo com métricas estabelecidas como latência, vazão, entre outros, simulando sua inserção em outros pontos da árvore e conectando-se a novos pais quando for vantajoso.

Como a lista recebida é arbitrária, um nó pode escolher outro nó para o qual ele indiretamente envia o fluxo (um nó abaixo dele na mesma árvore de compartilhamento). Em grande parte dos casos a lista de *rootpath* detectará esse laço e a inserção não será efetuada. Entretanto, devido ao paralelismo das mensagens, ainda assim laços podem ser formados. Para a detecção tardia de laços, mensagens de atualização de *rootpath* contém ainda uma informação sobre os saltos presentes no caminho, chamado *switchstamp*. Quanto maior o *switchstamp*, mais alta sua posição na árvore. Ao detectar a existência de um laço através de uma mensagem de *rootpath*, caso o *switchstamp* maior da lista seja o seu, ele será o responsável por quebrar o laço.

Outro ponto importante no YOID é seu suporte nativo ao *IP Multicast*. Para isso, o YOID cria um *cluster* englobando todo o grupo *Multicast*, promovendo um dos integrantes do grupo a cabeça do grupo. Para a rede YOID, apenas a cabeça do grupo está efetivamente conectada à rede, e toda transmissão de algum nó do grupo *Multicast* para a rede YOID precisa, necessariamente, ser transmitida pela cabeça da rede, assim como todo recebimento da rede YOID para o grupo.

3.3.3 NICE

O NICE [31] (NICE é o Ambiente de Cooperação na Internet, do inglês *NICE is the Internet Cooperative Environment*), é um projeto desenvolvido pela universidade de Maryland com o intuito de criar um *framework* cooperativo para a implementação escalável de aplicações distribuídas pela Internet. A idéia geral do projeto é permitir a criação de aplicações que compartilhem recursos das máquina dos cliente pelos demais clientes presentes na rede. Este *framework* não necessita de nenhum recurso adicional da camada de rede, permitindo o compartilhamento de conteúdo utilizando a estrutura atual da rede. Este projeto inclui diversos protocolos, entre eles o protocolo para comunicação *multicast* em nível de aplicação, chamado neste trabalho apenas de *protocolo NICE*.

O protocolo NICE é um protocolo de aplicação, que tem como idéia principal a redução de complexidade e o ganho de desempenho (*stress* e *stretch*, definidos no início desta seção). O protocolo utiliza uma estrutura hierárquica para sua rede, organizada em níveis. Uma das características principais do NICE é justamente essa estrutura hierárquica, que o difere dos demais protocolos previamente vistos. No NICE, a camada adicional de controle não constitui uma rede *mesh*, mas sim uma rede estruturada em clusters, utilizada tanto para o controle quanto para a transmissão de conteúdo. Sendo assim, cria-se apenas uma camada adicional, e não duas como na maioria dos outros protocolos.

Funcionamento Básico

O funcionamento básico do protocolo NICE depende em grande parte da estrutura hierárquica de sua rede. Esta rede, composta pos diversos níveis hierárquicos, tem em cada nível um conjunto de *clusters*. Esses *clusters* são formados por nós da rede, a partir de métricas de aproximação dos nós, como por exemplo baixa latência entre os nós. Para estabeler a hierarquia da rede, têm-se a princípio que todos os nós presentes na rede estejam inclusos em um, e apenas um, *cluster* do nível mais baixo da hierarquia. O tamanho desse *cluster* varia entre k e $3k$ -1, onde k é uma constante que determina a quantidade de nós próximos uns aos outros. De cada *cluster* no nível mais baixo, escolhe-se, através de métricas de distância, o integrante mais equidistante de todos os nós, chamado *cluster leader*. O

conjunto de *cluster leaders* de um nível mais baixo são promovidos a um nível hierárquico maior, embora continuem fazendo parte do nível hierárquico inferior. Esses *cluster leaders* então formam um novo *cluster* na hierarquia superior, escolhem-se os *cluster leaders* do novo *cluster* e cria-se uma nova hierarquia, até haver um nível hierárquico com apenas um integrante.

Da definição acima, temos que (i) se um nó está presente em um determinado nível hierárquico, ele estará presente em todos os níveis hierárquicos inferiores. (ii) um nó só está presente em um *cluster* por nível hierárquico. (iii) Se um nó não está presente em um nível hierárquico j, ele não estará presente em nenhum nível hierárquico acima de j. (iv) Há, no máximo, $\log_k$n níveis hierárquicos. Utilizando estas regras, cada nó mantém informações sobre cada um de seus vizinhos de *cluster*, assim sobre o *cluster* um nível hierárquico acima de seu maior nível. Este modelo é exemplificado na figura 3.6, onde todos os nós A's estão no primeiro nível, os nós B's são os líderes de seus respectivos *clusters*, e fazem parte do primeiro e segundo nível, e o nó C0 (único, neste exemplo), é o líder do *cluster* superior, e pertence ao primeiro, segundo, e terceiro nível. Esta figura ainda exemplifica o método de propagação de uma mensagem enviada por C0, que é propagada através dos líderes de *clusters* para seu *cluster* inferior, e assim por diante.

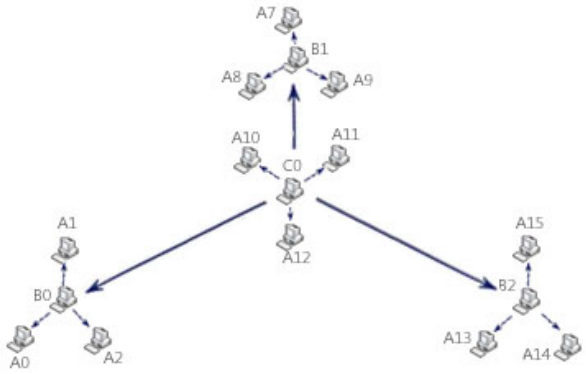

Figura 3.6: Exemplo ilustrativo das redes sobrepostas do protocolo NICE.

Inserção na Rede

O protocolo NICE, assim como outros protocolos ALM, utilizam uma unidade centralizadora conhecida por todos os nós. Esta unidade, normalmente a origem do conteúdo a ser

distribuído, é responsável pela entrada dos nós na rede, iniciando o processo de inserção do nó.

A inserção do nó se dá através da busca de uma posição em um *cluster* do nível mais baixo, iniciando a busca pelo nível mais alto. Como os *cluster leader* possuem características semelhantes aos *clusters* aos quais pertencem, a busca têm por objetivo procurar um *cluster leader* que mais se assemelhe ao novo nó. Para isso, um nó que deseja entrar na rede primeiramente contacta a unidade centralizadora, que envia uma lista do cluster do primeiro nível hierárquico. Desta lista, o nó verifica qual mais se assemelha a ele, requisitando uma lista de membros do *cluster* inferior no qual o nó semelhante é um *cluster leader*, e assim recursivamente até o nível mais baixo.

Manutenção e Gerenciamento das Camadas

Devido ao fato do *cluster leader* ser sempre o centro equidistante de todos os membros do *cluster*, mudanças nos membros do *cluster* podem ocasionar a demoção de um *cluster leader*. Quando um *cluster leader* é demovido, ele retira-se de todos os *clusters* hierarquicamente superiores ao qual ele foi efetivamente demovido, e novos *cluster leaders* devem ser promovidos em cada *cluster* afetado. Iniciando pelo menor nível hierárquico onde houve mudança de *cluster leader*, promove-se um novo *cluster leader*, que toma o lugar do antigo *cluster leader* também no *cluster* de nível imediatamente superior, que poderá ter seu *cluster leader* modificado e assim sucessivamente. A figura 3.7 mostra a mudança ocorrida na rede quando, devido á saída do nó A3 e a entrada de A4, o *cluster leader* do *cluster* passa a ser o membro A2. Neste caso, o antigo *cluster leader* A1 remove-se do *cluster* superior (A1-B1-C1), sendo substituído por A2. Na entrada de A2 no *cluster* superior, verifica-se que ele é o maior indicado a ser o novo *cluster leader* deste *cluster* também, sendo B1 substituído por A2 no primeiro *cluster*.

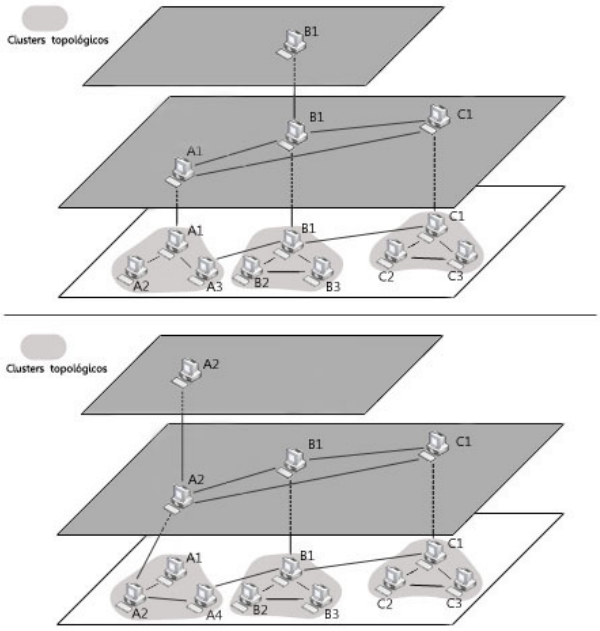

Figura 3.7: Exemplo de alteração na rede na troca de um *cluster leader*.

Periodicamente, cada nó envia uma mensagem, chamada *Heartbeat*, da distância estimada entre e seus vizinhos no *cluster*, para todos os vizinhos do *cluster*. O *cluster leader* envia ainda uma lista de todos os membros do *cluster*, para auxiliar na integração do *cluster* caso um novo nó tenha sido adicionado. Ainda, o *cluster leader* envia uma lista de todos os participantes do *cluster* hierárquico superior, ao qual ele mesmo pertence, para auxiliar na recuperação da rede caso o *cluster leader* venha a falhar.

Ainda periodicamente, o *cluster leader* verifica o tamanho de seu *cluster*. Caso o tamanho do *cluster* seja superior ao limite de *3k-1*, o *cluster leader* reparte o *cluster* atual em dois *clusters* de modo que a distância máxima entre os nós do mesmo *cluster* seja minimizada, e ainda determina os novos nós centrais dos dois *clusters* através de mensagens de transferência de liderança. Esta pode ser uma operação recursiva, já que cada metade do *cluster* anterior pode vir a ser maior que *3k-1*, e neste caso, o processo continua. No outro oposto, caso o tamanho do *cluster* for inferior a k, seu *cluster leader* busca, no *cluster* de nível superior, pelo vizinho mais próximo. O *cluster leader* requisita a este vizinho, que é um *cluster leader* no nível inferior, que receba os novos nós em seu *cluster*, através de uma mensagem de requisição de união de *clusters*. Após isso, ambos os

cluster leaders enviam mensagens de atualização sobre os novos componentes do *cluster*. O antigo *cluster leader* é demovido e retirado do nível hierárquico superior, pois não é mais um *cluster leader*.

Características Específicas

Ao iniciar o processo de inserção, pode-se levar algum tempo para que o nó encontre um *cluster* adequado para se inserir. Enquanto o processo não é completo, o nó temporariamente inclui-se na rede como membro direto do *cluster* de maior hierarquia, que recebe o conteúdo diretamente da origem do conteúdo, ou do nó presente no nível hierárquico mais alto. Este mecanismo pode aumentar a sobrecarga no nó de maior nível hierárquico, mas é particularmente útil quando a inserção de um nó encadeia a divisão e criação de vários *clusters* em diferentes níveis hierárquicos.

Tolerância a Falhas

Um mecanismo de Tolerância a Falhas está presente na manutenção dos *clusters*. Quando um novo *cluster leader* é escolhido, ele pode não conseguir ingressar no *cluster* superior por diversos motivos. Quando isto ocorre, o novo *cluster leader* inicia um processo de inserção em algum cluster no nível especificado. Este processo envolve percorrer o nível hierárquico superior e modificá-lo caso necessário, dependendo da atuação da unidade centralizadora.

Um nó pode deixar de funcionar sem antes poder informar aos outros membros de sua saída. Caso o nó não seja um *cluster leader*, sua saída é detectada através da não-entrega das mensagens de *HeartBeat* e o *cluster* se estabiliza como visto anteriormente. Entretanto, quando o nó faltante é um *cluster leader*, todos os outros nós do *cluster* devem escolher, de maneira independente, quem é o novo *cluster leader*, baseado nas informações de distância das mensagens de *Heartbeat*. Quando dois ou mais *cluster leaders* forem promovidos no mesmo *cluster*, eles devem entrar em consenso utilizando um campo específico das mensagens dos *cluster leaders*, até que o nó mais equidistante dos outros torne-se o único *cluster leader*.

3.3.4 Narada

O Narada é um protocolo ALM voltado à distribuição de conteúdo, cuja rede de distribuição funciona de forma descentralizada. Sua rede sobreposta é dividida em duas, sendo uma delas uma rede *mesh*, responsável pela transmissão de informações de controle, e a outra delas representada uma árvore de abrangência, criada a partir das ligações existentes na rede *mesh*. Cada nó possui sua própria árvore de abrangência, que deve possuir todos os nós presentes na rede. A figura 3.8 demonstra a rede *mesh* (mais clara), assim como a árvore de abrangência de C_1. Os objetivos principais do Narada são a auto organização das camadas de maneira distribuída, a eficiência dessas camadas, a auto-otimização das mesmas utilizando quebras e inserções de ligações entre nós existentes e a adaptatividade à dinamicidade das redes P2P.

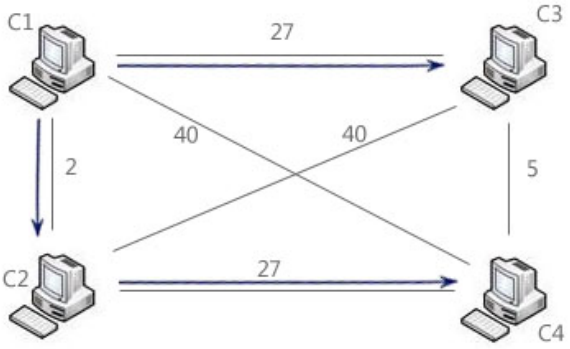

Figura 3.8: Exemplo ilustrativo das redes sobrepostas criadas pelo protocolo Narada.

Funcionamento Básico

O Narada pressupõe a existência de um mecanismo para a obtenção de uma lista parcial de *peers* ativos na rede, comumente implementado como um *rendezvous point* ou por listas disponíveis fora do contexto do protocolo. O Narada constrói primeiramente a rede *mesh* de controle, derivando a partir desta uma árvore de abrangência específica para cada nó. Esta abordagem é especialmente importante para o desempenho de aplicações com várias fontes de conteúdo, uma vez que a rede *mesh* dá mais robustez à arvore de abrangência.

Como a árvore de abrangência é construída a partir da rede *mesh*, o caminho da transmissão está diretamente ligada à qualidade das ligações existentes na camada de

controle, que deve ser balanceada para que não haja sobrecarga de nós na rede. Para isso, o Narada utiliza alguns mecanismos de refinamento da rede *mesh*, vistos nas seções a seguir.

Inserção na Rede

Quando um nó deseja entrar na rede, ele deve obter uma lista de membros ativos de alguma forma (seja através de um *rendezvous point* ou uma lista externa). O nó envia requisições de inserção a todos os membros presentes na lista. Apenas uma aceitação é necessária para o processo ser bem sucedido, tornando o nó vizinho do membro que o aceitou na rede. Assim que ele é adicionado, o novo nó envia mensagens de atualização para seu vizinho, que irá repassar essa atualização aos demais nós da rede. O mecanismo de atualização dos nós é visto na seção seguinte.

Manutenção e Gerenciamento das Camadas

No Narada, o gerenciamento da camada de controle é feita de maneira distribuída. Para isto, cada nó mantém uma lista atualizada contendo todos os outros nós conhecidos da rede. Para cada nó k, são guardadas informações sobre o endereço de k , o último número de sequência sk enviado por k e o *timestamp* em que sk foi recebido. De maneira pró-ativa, cada nó envia para seus vizinhos sua lista completa de nós conhecidos contendo inclusive a si mesmo, incrementando seu próprio número de sequência. Ao receber a mensagem, um vizinho j pode prosseguir de duas maneiras. Caso não haja uma entrada de k na lista, j adiciona k à sua própria lista, adicionando um campo correspondente ao *timestamp* local em j quando este recebeu a mensgem de atualização de k. Caso haja uma entrada, seja i o vizinho de j que enviou a lista, j então verifica o número de sequência s_{ki}. Caso o campo s_{ki} da mensagem seja menor que s_{kj} (último número de sequência de k recebido por j), então a mensagem é descartada, caso contrário o número de sequência s_{kj} recebe o valor de s_{ki} e o *timestamp* é atualizado para o *timestamp* local. Para evitar problemas de *loops* infinitos nas atualizações, o *Narada* utiliza-se de uma estratégia similar à empregada no protocolo [38] sobre a rede *mesh*. Cada membro mantém uma tabela de

custo de roteamento para todos os membros, e também o caminho relacionado a aquele custo, sendo a atualização feita sobre esses dois campos. Também periodicamente, cada nó envia a todos os seus vizinhos suas tabelas de roteamento, contendo o custo relacionado à cada nó e o caminho utilizado para chegar até ele.

Para o encaminhamento dos dados, o Narada utiliza um mecanismo igual ao DVMRP [5]. Um nó m que recebe uma mensagem de n, cujo transmissor inicial foi s, irá propagar a mensagem apenas se n for o próximo salto na árvore de roteamento de m para s, e enviará a mensagem apenas para os nós que utilizarem m como próximo salto para s. Para isso, além de manter uma tabela de roteamento com custos e caminhos para cada outro nó da rede, ele deve também manter uma tabela de roteamento com custos e caminhos de cada vizinho para cada nó da rede, atualizas periodicamente.

Características Específicas

Uma das características específicas do protocolo Narada é que os nós periodicamente buscam por novas conexões vantajosas na rede *mesh*, além de derrubar conexões inúteis. De maneira proativa, cada nó busca por nós aleatórios aos quais ainda não é vizinho e calcula, no caso de uma conexão entre os dois nós existir, quais outros nós presentes na rede terão seu desempenho com o nó iniciador do processo melhorado. Para determinar a melhora no desempenho, utiliza-se uma função de *utilidade*. Esta função de *utilidade* é definida a partir de métricas e do objetivo da otimização da malha, seja ela latência, tamanho do caminho, etc. O valor da função de utilidade de um nó é dado em relação ao tamanho total da rede, e a adição de uma nova conexão é feita caso esse valor exceda um limite (dado como parâmetro do protocolo).

De modo semelhante, o nó verifica o *custo de consenso* das arestas existentes. Para o cálculo do custo, utiliza-se uma superestimação da *utilidade* da conexão. Dado um nó i, que esteja testando o *custo de consenso* com um vizinho atual j. O custo de consenso é o valor máximo entre quantos outros nós i utiliza j como próximo salto e quantos outros nós j, a partir da tabela de roteamento local de i, utiliza i como próximo salto. Para a conexão ser desfeita (e os vizinhos serem separados), esse custo deve ser inferior a um

limite em relação ao tamanho total da rede, determinado como parâmetro do protocolo. Como exemplo, caso j tenha como único vizinho na rede o nó i, o custo da conexão de i para o nó j será máximo (pois j utiliza i como próximo salto para todos os demais nós), sendo mínimo para i (pois i utiliza j apenas para alcançar o próprio j). Neste caso, o custo de consenso da conexão ij é máximo.

É importante notar que o custo de consenso entre dois nós deve ser pequeno o suficiente para não acarretar a re-adição da conexão pelo mecanismo de utilidade.

Tolerância a Falhas

Quando um nó deixa a rede sem aviso prévio, o principal problema para a rede é a possibilidade de seu particionamento. Para contornar esse problema, cada nó possui um tempo de espera máximo de mensagem de atualização tm que, quando expirado, adiciona o nó cujo tm expirou em uma fila de membros de tm expirado. O nó então executa um algoritmo probabilístico e periódico para remover o nó cabeça da fila, que tem sua existência sondada, resultando na adição de uma conexão entre os nós ou a confirmação de que destino deixou a rede. A importância do algoritmo ser executado de forma probabilística se dá devido à prevenção da adição de diversas conexões para o mesmo nó, uma vez que uma conexão entre dois nós de partições diferentes reestabelece o contato entre todos os nós das duas partições. A checagem por mensagens de atualizações perdidas se dá de maneira periódica, sendo o intervalo de tempo entre as checagens um parâmetro do protocolo, cujo valor é relacionado à taxa de envio das mensagens de atualização.

Outro mecanismo de tolerância a falhas faz com que nós saindo de maneira não espontânea permaneçam na rede até nenhum outro nó utilizá-lo como rota. Esse resultado é obtido atribuindo-se um custo elevado para qualquer rota utilizando o nó que irá sair, até que as rotas seram atualizadas para outras rotas de menor custo.

3.4 Outros Protocolos

Vários protocolos ALM propostos recentemente procuram descobrir informações da topologia de rede para melhorar seu desempenho [33, 34, 35, 36]. Esses protocolos necessitam de uma arquitetura mais complexa, comumente envolvendo *peers* confiáveis ou então *landmarks* para definir a posição de um *peer* na rede. O problema principal nesta abordagem é a dependência dessas entidades (além do *rendezvous point* presente na maioria dos protocolos ALM), criando mais pontos críticos de falhas. Outro problema secundário se deve ao fato da informação sobre a topologia de rede se basear no RTT entre os *peer*. Em uma rede dinâmica como a Internet, onde o RTT entre diversos nós pode mudar durante a execução do protocolo, esses protocolos estão sujeitos à picos de variaçoes na rede, resultando em um processo ineficiente de localização.

Alguns protocolos possuem asserções incomuns em um ambiente dinâmico (como a Internet). Em alguns casos, utiliza-se a asserção de que todos os nós possuem banda de envio suficiente para os parâmetros do protocolo, mas segundo [39] nem todos os *peers* contribuem de modo eficaz para a rede, não havendo portanto garantias de banda de envio disponível em todos os nós.

Nesta seção, são apresentados os protocolos *Highways* [34] e o Sistema Híbrido Cliente-Servidor Assistido por Peers Não-Confiáveis [40]. O Highways é um exemplo de protocolo que utiliza em sua rede *overlay super-peers* como *landmarks* para definir a posição de um *peer* na rede. Já o sistema híbrido cliente-servidor assistido por peers não-confiáveis é um sistema que, apesar de não implementar explicitamente o *multicast* em nível de aplicação, possibilita a distribuição de conteúdo na estrutura atual da Internet, baseando-se no conceito de *peers* não confiáveis, servindo como exemplo de um sistema cujas asserções encontram-se presentes no comportamento dos nós na Internet.

3.4.1 Highways

O protocolo descrito em [34] baseia-se no uso de *overlay* hierárquico geométrico utilizando *super-peers* e *peers*, proposto em [41]. Nesta rede, utiliza-se um grupo de *super-peers* para

criar um overlay confiável, simulando a infraestrutura de um backbone. Utilizando os *super-peers* como *landmarks*, todos os *peers* e *super-peers* são mapeados em um espaço geométrico euclidiano, de acordo com a medição de seus RTT's. Uma eleição de um *super-peer* leva em consideração não somente a quantidade de recursos disponíveis no *peer*, informação facilmente obtida, mas também a confiabilidade do *peer*. Caso um *peer* esteja apenas em trânsito na rede, saindo e entrando frequentemente, ele não poderá ser um *super-peer*. Entretanto, a confiabilidade de um *peer* é de difícil detecção, pois no momento da inserção não é possível afirmar se um *peer* será confiável ou não.

Esse protocolo impõe alta carga nos *super-peers*, pois eles estão presentes num grafo altamente conexo, sendo cada *super-peer* conectado a outros 6 *super-peers*. Os *super-peers* são divididos em 3 clusters (simulando os continentes das Américas, África/Europa e Ásia) de acordo com seus RTT's entre si. Um *peer* encontra sua posição geométrica na rede calculando seu RTT em relação a todos os *super-peers*. Além disso, cada *peer* possui uma posição local (referente ao seu cluster) e uma posição global (referente à medições inter-cluster). Internamente, os *peers* calculam suas distâncias entre si através de seus RTT's, e a partir desta informação derivam uma matriz de transição entre o cálculo do RTT e a posição geométrica de cada nó presente em seu cluster. Para *peer* presentes em outros clusters, utiliza-se essa matriz de transição de algum *peer* presente no cluster externo, e a partir desta matriz têm-se a posição global dos *peers* pertencentes a clusters externos.

O processo utilizado para encontrar a posição geográfica de um *peer* através de *landmarks* é conhecido como triangulação, e depende apenas de que os *landmarks* não sejam colineares para as coordenadas (dimensões) a serem trianguladas.

3.4.2 Hybrid Client-Server Multimedia Streaming Assisted by Unreliable Peers

O sistema de *streaming* multimídia híbrido cliente-servidor assistido por *peers* não confiáveis apresentado em [40] é um sistema voltado à distribuição de conteúdo contínuo proveniente de uma única fonte na Internet. Utilizando o modelo híbrido, este sistema possui tanto

um servidor quanto *peers*.

O servidor é responsável pela geração e distribuição inicial do fluxo, armazenando-o temporariamente em um *buffer* local. Diferente da maioria dos protocolos vistos até agora, neste sistema cabe ao servidor a tarefa de atender a requisições de entrada dos *peers*, distribuir uma lista de *peers* bem como informar a todos os *peers* quando o conteúdo encontra-se disponível.

À medida que o fluxo é gerado, ele é dividido em *fatias* (de tamanhos iguais), que por sua vez são divididos em *blocos* também de tamanhos iguais, sendo então enviados aos *peers*. Esta implementação permite que durante uma janela de tempo, especificada pela frequência com que o conteúdo é consumido, os blocos possam ser recebido em qualquer ordem. Utilizando as informações de sequência de produção das fatias, os *peers* realizam *acordos de compartilhamento* entre si (ou entre o servidor). A divisão das fatias em blocos permite a descentralização do recebimento (um *peer* pode receber blocos diferentes de *peers* diferentes). No caso de falha, apenas os blocos faltantes são requisitados.

Os *acordos de compartilhamento* possuem um período de duração fixo em número de fatias, que determinam quais *peers* irão receber e transmitir quais blocos das fatias compreendidas na duração do acordo. Os acordos são feitos com uma duração máxima pois prevê-se que a rede é dinâmica, e os *peers* nela presentes não são confiáveis podendo subitamente deixar a rede. Como os blocos e fatias precisam ser recebidos completamente no momento de execução do conteúdo, os acordos são feitos *a priori* do recebimento dos mesmos. Quando um *peer* alcança o tempo limite (através do número de sequência das fatias recebidas) de recebimento de uma fatia sem recebê-la completamente, os blocos faltantes são requisitados diretamente ao servidor, não comprometendo a execução do conteúdo. Durante a vigência de um acordo, o servidor realiza acordos espontâneos iniciais com alguns *peers*. Após um intervalo de tempo fixo,o servidor informa a todos os *peers* que novos acordos estão disponíveis. Ao receber a notificação, cada *peer* verifica quais blocos do próximo acordo ele ainda não possui, buscando entre seus *peers* vizinhos por acordos de compartilhamento. Caso um *peer* não consiga acordos até o prazo limite de realização de acordos, ele requisita a transmissão daquele bloco para o servidor, pela

duração do acordo em questão. A duração do acordo, o intervalo entre o início do acordo e a realização de acordos espontâneos para o acordo seguinte,a quantidade de acordos espontâneos realizados, o intervalo para a notificação de novos acordos e o prazo limite de realização de acordos são parâmetros do protocolo.

Cada *peer* mantém uma lista de outros *peers* para os quais possa requisitar acordos quando notificado pelo servidor. Periodicamente, os *peers* verificam se possuem conhecimento de outros *peers* suficientes para haver uma grande chance de realizar um acordo com sucesso. Também periodicamente, os *peers* verificam o tempo de ida e vinda (RTT) entre os *peers*, e calculam também o tempo de ida e vinda em relação ao servidor. A escolha dos acordos espotâneos realizados pelo servidor e a escolha dos *peers* para requisição de acordos se dá em ordem crescente dessas estimativas.

Por este ser um sistema completo, há a implementação de um *buffer* para o armazenamento do conteúdo, normalmente não incluso nos protocolos ALM por serem de responsabilidade de uma aplicação consumidora superior ao protocolo. Este *buffer* é utilizado para amenizar a perda de pacotes proveniente da saída de um *peer* da rede. Um *peer* cujo acordo de recebimento foi rompido só perceberá a falha quando seu *buffer* de consumo estiver vazio, quando ele irá então requisitar o conteúdo ao servidor.

Esse sistema não implementa detectores de falhas em nível de aplicação, sendo responsabilidade do *socket* a detecção de falha em um *peer*. Sendo assim, faz-se necessária a implantação deste sistema utilizando protocolos da camada de rede que suportem a detecção de falhas, como o TCP [2].

Capítulo 4

O Protocolo ALM Proposto

Com o rápido aumento da utilização de sistemas de distribuição de conteúdo na Internet, diversos protocolos para esta tarefa vêm surgindo, como os descritos no capítulo 3. Esse aumento se deve em grande parte devido à popularização das redes P2P que são utilizadas para a distribuição de conteúdo, já sendo responsáveis por grande parte do tráfego de dados na rede [19, 1]. Em grande parte dos sistemas P2P, principalmente em sistemas P2P estruturados, há uma grande sobrecarga imposta aos *peers* devido à utilização de mensagens de controle. Projetando protocolos mais eficientes, é possível diminuir a carga imposta aos *peers*.

O protocolo desenvolvido neste trabalho é um protocolo ALM voltado a redes de distribuição de conteúdo contínuo que utiliza desigualdades triangulares para refinar sua topologia. O protocolo tem por objetivo a distribuição de fluxo contínuo proveniente de uma única fonte inicial através de uma árvore de fonte específica, reduzindo as tarefas de controle e utilizando a desigualdade triangular para auxilixar no refinamento da topologia. Este capítulo é organizado da seguinte maneira. A seção 4.1 descreve como o conteúdo é produzido e distribuído no sistema. A seção 4.2 descreve a arquitetura do sistema, descrevendo cada entidade e suas funções. A seção 4.3 descreve o mecanismo de desigualdade triangular utilizado para refinamento da topologia. A seção 4.4 apresenta os resultados da avaliação experimental realizada.

4.1 Organização do Conteúdo

A organização do conteúdo compreende o seu mecanismo de geração e distribuição. À medida que o conteúdo é gerado, ele é dividido em pedaços (fatias) de tamanho fixo, dado

como parâmetro do sistema ao servidor. Denomina-se um *acordo de envio* o acordo que um *peer* realiza para enviar uma fatia a outros *peers*, e *acordo de recebimento* o acordo que um *peer* realiza para receber uma fatia de outro *peer*. Um *peer* p_r que possui um acordo de recebimento com outro *peer* p_s recebe os dados de p_s, sendo que p_s possui um acordo de envio para p_r. Os acordos possuem duração indeterminada, sendo interrompidos apenas quando uma das partes falha ou quando há o dispardo do mecanismo de desigualdade triangular. O conjunto de acordos de compartilhamento forma a rede P2P. A priori, cada *peer* possui quantos acordos de envio quanto permite sua largura de banda, e um acordo de recebimento. A Figura 4.1 demonstra um exemplo desse conjunto de acordo entre os *peers*. Nesta figura, pode-se observar a propagação da fatia 1, transmitida pelo servidor aos *peers* $p_{1.1}$ e $p_{2.1}$. Os peers $p_{1.1}$, $p_{1.2}$ e $p_{1.3}$ pertencem ao grupo g_1, enquanto os *peers* $p_{2.1}$ e $p_{2.2}$ pertencem ao grupo g_2. Quando $p_{1.1}$ recebe a fatia 1, ele a propaga para $p_{1.2}$, que por sua vez propaga para $p_{1.3}$. O mesmo ocorre no grupo g_2, onde $p_{2.1}$, ao receber a fatia 1, propaga-a para $p_{2.2}$.

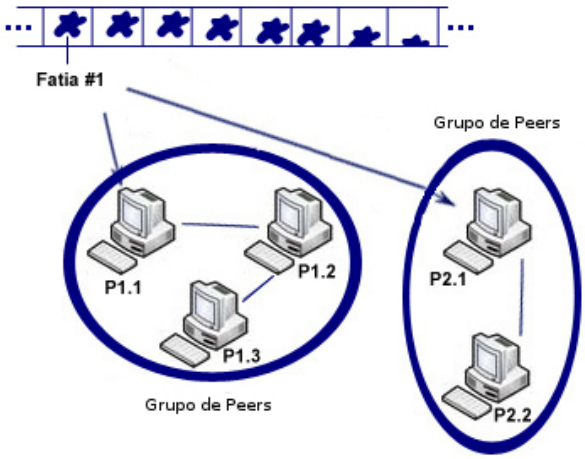

Figura 4.1: Distribuição de fluxo segundo acordos realizados.

4.2 Arquitetura do Sistema

O PALMS configura-se como um protocolo híbrido em relação aos modelos P2P e cliente-servidor, possuindo tanto a figura de um servidor quanto a figura dos *peers*. A arquitetura do PALMS é composta por três entidades, *servidor*, *tracker* e *peer*. Ao *servidor*, cabe a

tarefa de gerar e transmitir os dados que serão distribuídos no sistema. Ao *tracker*, cabe a tarefa de permitir a entrada organizada de novos *peers* na rede. Aos *peers*, cabe a tarefa de recepção, execução e propagação do conteúdo a outros *peers*. A Figura 4.2 mostra a estrutura geral da arquitetura do protocolo. A seguir, cada um dos componentes da arquitetura é descrito.

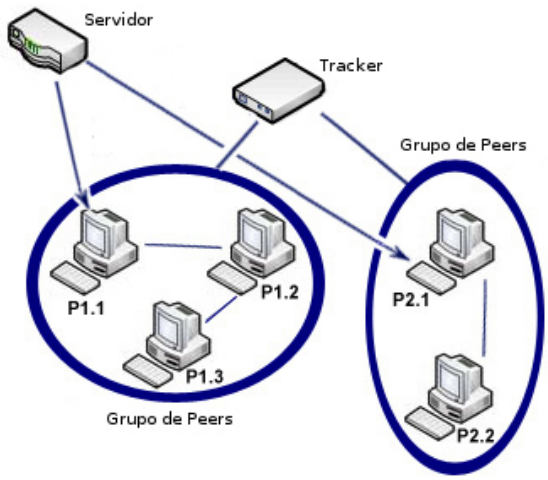

Figura 4.2: Arquitetura do protocolo PALMS.

4.2.1 O Servidor de Fluxo

O *servidor*, também chamado de fonte, é a entidade responsável pela produção e disponibilização do conteúdo na rede. O conteúdo é gerado como um fluxo contínuo (em inglês, *stream*), e é descrito em detalhes na seção 4.1.

É função do servidor determinar o intervalo de tempo em que cada novo elemento do fluxo será disponibilizado. Este intervalo é fixado no início da transmissão, mantendo-se constante até seu término.

Em relação à distribuição de conteúdo, o servidor desempenha a função de *seeder* das redes *torrent* [18], que diferencia-se de um *peer* comum tanto por não precisar receber conteúdo de nenhum outro *peer* como por não consumi-lo. O servidor pode enviar o conteúdo produzido a qualquer *peer* requisitante, respeitando seu limite de banda. As requisições de conteúdo são descritas na seção 4.2.3.

Em relação à topologia da rede, o servidor não possui conhecimento de todos os nós presentes na rede, como também não participa da organização dos *peers*. Apenas *peers*

que recebem conteúdo diretamente do servidor são conhecidos por ele.

4.2.2 O Tracker

O *tracker* desempenha funções semelhantes às encontradas nos *trackers* das redes *torrent* [18] e aos *rendezvous Point* de alguns protocolos ALM [31, 29]. No PALMS, o *tracker* desempenha duas funções: (i) auxiliar na inserção de *peers* na rede e (ii) auxiliar na organização da rede.

É função do *tracker* disponibilizar o endereço do sistema na rede, possibilitando a entrada de novos *peers* a partir deste endereço. Ou seja, um novo *peer* conecta-se ao sistema através do *tracker*, e não através do servidor como nos sistemas baseados no modelo cliente-servidor. Essa mudança retira a carga de inserção de novos *peers* do servidor, permitindo que ele execute apenas a função de distribuição de conteúdo.

O *tracker* mantém uma lista de *peers* ativos na rede, com informações em relação à disponibilidade de retransmissão de cada *peer*. Informações sobre disponibilidade de retransmissão podem ser enviadas pelo próprio *peer* ou por outros *peers* que detectem essa condição. Um *peer* é adicionado à lista quando ele é inserido com sucesso na rede (descrito em detalhes na seção 4.2.3), sendo responsabilidade do *peer* reportar o sucesso de sua inserção. Um *peer* é removido da lista quando informa sua saída ao *tracker*. Um *peer* pode também ser marcado como indisponível quando reportado por si mesmo ou através de outro *peer*. O *tracker* não realiza verificações ativas nos *peers*, distribuindo essa função para os *peers* presentes na rede. Essa implementação pode causar uma inconsistência temporária na lista de *peers* do *tracker*, que se aproximará mais do estado real da rede a medida que os *peers* ativos reportarem inconsistências.

Para auxiliar a inserção de novos *peers*, o *tracker* envia a cada novo *peer* três informações iniciais: o endereço do servidor, informações sobre o fluxo e um conjunto de *peers*. O endereço do servidor faz-se necessário pois na ocorrência de falhas na transmissão entre os *peers*, é feita uma requisição do conteúdo diretamente ao servidor, visando evitar interrupções significantes na execução do conteúdo. As informações sobre o fluxo visam auxiliar os *peers* no correto recebimento do conteúdo e na detecção de problemas relaci-

onados ao seu recebimento. Informações sobre o fluxo compreendem a taxa do conteúdo (relevante quando o conteúdo é um vídeo ou áudio, por exemplo) e o intervalo de produção. O conjunto de *peers* é um subconjunto da lista de *peers* mantida pelo *tracker*. Esse subconjunto é composto apenas por *peers* que o *tracker* considera disponíveis na rede, e seus integrantes são escolhidos aleatoriamente dentre todos os *peers* da lista. A escolha por um subconjunto aleatório baseia-se na premissa otimista que haverá pelo menos um *peer* disponível que consiga propagar o conteúdo ao novo *peer* e, como a cada requisição um subconjunto aleatório diferente é escolhido, a carga de propagação fica distribuída entre um grande número de *peers* na rede. O tamanho do conjunto de *peers* enviado pelo *tracker* é um parâmetro do sistema.

Para auxiliar na organização da rede, o *tracker* mantém um contador de *número de grupos* (*groupID*), que é incrementado sempre que um novo grupo é formado (os grupos são descritos na seção 4.2.3). Quando um *peer* inicia sua inserção na rede, ele pode não conseguir se conectar a nenhum *peer* do conjunto de *peers* enviado pelo *tracker* (ou o conjunto pode estar vazio ou todos os *peers* estão indisponíveis). Neste caso, o *peer* requisita ao *tracker* a criação de um novo grupo. Esse processo assegura a distinção entre os grupos, sem a necessidade de atualizações constantes pelo *tracker*.

A escolha pelo desenvolvimento do *tracker* foi feita para não haver riscos de erros de inserção de *peers* na rede. Caso este papel seja desempenhado por um outro *peer*, a complexidade do sistema aumentaria pois cada *peer* precisa ter uma visão consistente da rede. Uma das vantangens da existência do *tracker* é a possibilidade de implementar novas funções sobre o *tracker*, que não poderiam ser desempenhadas por *peers* num sistema totalmente distribuído. Um exemplo de funcionalidade que pode, caso necessário, ser implementada são as funções de atravessamento de NAT [42].

4.2.3 Os Peers

Os *peers* são os processos dos usuários finais do sistema, e são responsáveis não somente por consumir, mas também por auxiliar na propagação do conteúdo na rede. Há duas maneiras de um *peer* receber o conteúdo, diretamente do servidor ou de outro *peer*. Em

ambas as situações o *peer* poderá enviar o conteúdo recebido à outro *peer* caso possua disponibilidade de banda, formando assim a rede P2P. Nesta seção, são apresentados todos os mecanismos compreendidos em um *peer*.

Mecanismo de Entrada

Quando um *peer* requisita sua entrada na rede, ele inicialmente recebe do *tracker* as informações tanto sobre o servidor (endereço IP e porta) quanto sobre o fluxo(intervalo de produção do conteúdo e *stream rate*), como também recebe uma lista de *peers*, descrita na seção 4.2.2. O *peer* então dispara simultaneamente mensagens de requisição de inserção a todos os *peers* da lista. A aceitação ou não da requisição de inserção depende da disponibilidade de banda para retransmissão do conteúdo por parte dos *peers* receptores da mensagem. Apenas uma única resposta positiva é necessária para completar a inserção do *peer* na rede, sendo as demais descartadas. O disparo simultâneo paralelo se justifica pois, o *peer* que aceitar mais rapidamente a requisição será o *peer* mais indicado para a propagação do conteúdo ao novo *peer* naquele momento, pois apresenta o menor RTT (*round-trip-time*) dentre os *peers* da lista recebida.

Quando o *peer* p_i dispara requisições a todos os *peers* da lista recebida pelo *tracker*, aqueles que se encontram disponíveis enviam seus respectivos identificadores de grupos a p_i. A primeira resposta recebida por p_i, proveniente do *peer* p_j pertencente ao grupo g_x, informa a p_i que há um *peer* p_j disponível no grupo g_x. p_i então requisita sua inserção ao *peer* p_j, que informa a p_i o sucesso de sua inserção no grupo g_x e passa a retransmitir as fatias do conteúdo ao *peer* p_i.

Após receber cada fatia, o *peer* determina o *timeout* para o recebimento da próxima fatia. Esse *timeout* é baseado no intervalo de produção do servidor (enviado pelo *tracker* na inserção do *peer*) e é atribuído como um parâmetro do protocolo. Ao receber a próxima fatia, o *timeout* para a fatia recebida é suspenso e um novo *timeout* é disparado para a próxima fatia.

Periodicamente, os *peers* atualizam os tempos de ir-e-vir com os *peers* com os quais possui acordos, através de mensagens de atualização. A *taxa de atualização periódica*

determina a periodicidade em que essas atualizações acontecem e é dada como parâmetro do protocolo.

Mecanismo de Saída

Quando um *peer* p_k pretende deixar a rede, ele pode fazê-lo de duas maneiras distintas, de modo súbito ou não. De maneira não súbita, p_k informa sua saída a todos os *peers* com os quais possuem acordos. Os *peers* cujos acordos de recebimento com p_k foram rompidos reiniciam seus processos de descobrimento de um novo grupo, criando um novo quando necessário. O *peer* cujo acordo de envio foi rompido apenas remove p_k de sua lista de acordos e libera a banda utiliza para o envio, permitindo a realização de um novo acordo de envio.

Caso um *peer* deixe a rede sem prévia notificação, caracterizando uma saída súbita, inicialmente nenhum integrante do sistema é informado de sua saída. Um parâmetro do protocolo indica quantas mensagens de atualização um *peer* deve deixar de receber até considerar outro *peer* falho. Caso um *peer* p_f seja considerado por outro *peer* p_x, p_x então encerra todos os acordos existentes com p_f. Caso p_x possuísse um acordo de envio para p_f, o acordo é simplesmente desfeito por p_x, liberando-o para realização de novos acordos de envio. Caso p_x possuísse um acordo de recebimento de p_f, esse acordo é encerrado e p_x torna-se temporariamente órfão. Um *peer* órfão não possui ID de grupo válido (visto em detalhes na seção 4.2.3), e portanto não aceita requisições de acordo para envio. Apesar disso, um *peer* órfão mantém seus acordos de envio, recebendo o fluxo temporariamente do servidor.

Assim que detectada a condição de órfão, um *peer* busca entre sua lista de *peers* conhecidos (enviado pelo *tracker*) por um novo acordo (de recebimento). Para um *peer* desta lista ser candidato a realização deste acordo, além de todas as restrições presentes em um acordo inicial (quando um novo *peer* entra na rede), ele deve também ser de um grupo diferente do último grupo ao qual pertenceu o *peer* órfão (motivado pela não formação de laços, explicados mais adiante nesta seção). Caso não haja *peers* passíveis de acordo, um novo grupo é criado. A criação de novos grupos pelos *peers* agora órfãos faz

com que eles recebam conteúdo diretamente do *servidor* (ver seção 4.2.3 para detalhes). A atualização para um novo grupo ocorre do *peer* orfão a todos com os quais possui acordos de envio, e cada um destes *peers* por sua vez enviam a atualização a todos os *peers* com os quais possuem acordos de envio. Essa técnica permite a rápida recuperação do sistema a uma saída súbita de um *peer*.

Para não haver perdas no recebimento do conteúdo quando um *peer* deixa a rede entre os intervalos de atualização dos tempos de ir-e-vir, foi desenvolvido um mecanismo chamado *notify-emergency* que utiliza o *timeout* de recebimento das mensagens descrito na seção anterior. Caso o *timeout* dispare, o *peer* requisita a fatia diretamente ao servidor, informa a todos os *peers* com os quais possui acordos de envio que aquela fatia será entregue com atraso, e identifica-se como o *peer* que notificou o atraso através de uma mensagem de notificação de atraso (*notify*). Além disso, o *peer* se coloca em estado de emergência (*emergency*). Ao receber um *notify*, caso o *peer* não se encontre em emergência, ele adia seu disparo de *timeout* para a fatia em questão e propaga a mensagem de *notify* aos *peers* com os quais possui acordos de envio. Caso ele se encontre em estado de emergência, esse estado é removido. Caso o *timeout* estoure e ele se encontre em estado de emergência por mais de x estouros de *timeout* (sendo x um parâmetro do protocolo), ele considera que seu transmissor direto não se encontra apto a retransmitir o fluxo, mudando seu estado para órfão. A recepção de uma fatia retira o *peer* do seu estado de emergência.

Em todos os casos, a detecção de um *peer* falho (através dos mecanismos descritos acima e dos limites de não recebimento) ocasiona na notificação ao *tracker* da falha de um *peer*. A utilização de *buffer* é fundamental para acomodar oscilações no desempenho do sistema. O dimensionamento desses *buffer* está fora do escopo deste trabalho.

Organização dos Peers em Grupos

Os grupos são utilizados para separarem os *peers* em árvores de compartilhamento distintas. A relação pai-filho corresponde a um acordo de envio de pai para filho (e consequente acordo de recebimento do filho com o pai). Cada grupo possui um *peer* que recebe os dados diretamente do servidor (o primeiro *peer* do grupo), responsável por propagá-los

aos seus *peers* receptores, que por sua vez propagam para seus próprios *peers* receptores.

Quando um *peer* p_x deseja criar um novo grupo, por exemplo o primeiro *peer* de todo o sistema, ele envia uma requisição ao *tracker*. O *tracker* então incrementa seu contador de *id de grupos*, anteriormente gid_x, para gid_{x+1} e envia esse novo valor a p_x. Ao receber a resposta, p_x muda seu próprio *id de grupo* para gid_{x+1}. A modificação de *id de grupo* é propagada para qualquer *peer* em sua lista de acordos de envio. Após a modificação, qualquer outro *peer* que se conectar a p_x será pertencente ao grupo gid_{x+1}.

Quando um *peer* conecta-se a um grupo já existente, ele se conecta a apenas um único *peer* do grupo (ao qual ele enviou a requisição), firmando um acordo de recebimento diretamente com esse *peer*. O *peer* recém inserido pode possui banda suficiente para a propagação de conteúdo, informando ao *tracker* de sua disponibilidade e possibilitando a realização de acordos de envio a novos *peers*. A topologia da rede resultante desse processo forma uma árvore interna aos *grupos* para o compartilhamento de conteúdo. A topologia de compartilhamento em árvore é apresentada na Figura 4.3. As setas incidentes nos grupos indicam quais *peers* recebem fluxo diretamente do servidor, sendo estes os únicos *peers* conhecidos pelo servidor. Internamente aos grupos, o *peer* $p_{2.1}$ está conectado diretamente ao servidor, e não está conectado nem possui informações sobre nenhum outro *peer*. Já o *peer* $p_{1.1}$ possui acordos de envio com os peers $p_{1.5}$ e $p_{1.3}$. $p_{1.3}$ possui um acordo de recebimento de $p_{1.1}$ e um acordo de envio para $p_{1.2}$. $p_{1.5}$ possui um acordo de recebimento com $p_{1.1}$ e dois acordos de envio, um com $p_{1.6}$ e outro com $p_{1.4}$.

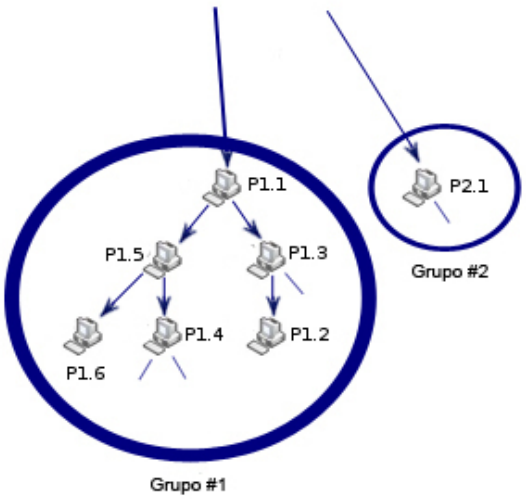

Figura 4.3: Topologia interna ao grupo.

Como os *grupos* representam as árvores de transmissão do conteúdo, a ocorrência de laços deve ser evitada ou detectada e resolvida. O PALMS procura evitar a ocorrência de laços através da não aceitação de requisições de acordos provenientes de um mesmo *grupo*. Quando ocorre o cenário incomum de um *peer* cujo identificador de grupo está desatualizado realizar um acordo de envio com um *peer* do mesmo grupo (mas cujo identificador está atualizado), um laço pode ser formado. A formação de laços resulta em falha na entrega do conteúdo, ocasionando o estouro do *timeout* de recebimento de mensagens de *peers* afetados pela formação do laço. Ao transmitir a mensagem de notificação (proveniente do mecanismo de *notify-emergency*), um *peer* detectará a formação de um laço ao receber sua própria notificação. Este *peer* encerra seu acordo de recebimento e o processo de inserção (ou criação de grupo) é iniciado, interrompendo o laço.

4.3 Desigualdade Triangular

A aleatoriedade na inserção de um novo *peer* possibilita a rápida alocação e recebimento do fluxo, além de garantir que o melhor *peer* dentre os enviados pelo *tracker* foi escolhido. Entretanto, à medida que os grupos vão crescendo, um *peer* inserido em um grupo pode ter melhores condições de propagação do conteúdo que o *peer* com o qual possui o acordo de recebimento. A natureza dinânima das redes P2P impede o uso eficiente de estratégias

que utilizem a visão global do sistema. Outro problema da dinamicidade é a variação de comportamento (como por exemplo a ocorrência de *jitter*) dos *peers*. Processos de refinamento da topologia que são disparados apenas na inserção de um *peer* podem ser influenciados por anomalias temporárias da rede, tornando a topologia resultante ineficiente.

O mecanismo desenvolvido para este protocolo visa o refinamento contínuo da topologia, utilizando o conhecimento local dos *peers*. Este mecanismo realiza trocas de posições no grupo entre os *peers* que enviam e recebem fluxo, utilizando um conceito baseado na desigualdade triangular geométrica presente no cálculo do RTT entre os *peers*. A figura 4.4 mostra um exemplo de como as arestas com peso representado pelos RTT dos *peers* podem formar um triângulo.

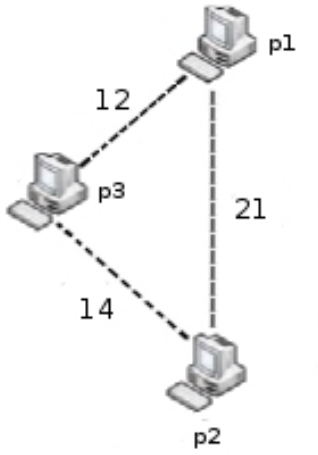

Figura 4.4: Arestas representando um triângulo.

A desigualdade triangular diz que dado um triângulo qualquer, um lado é menor que a soma dos outros dois lados. Na topologia em árvore do PALMS, considere o cenário onde três *peers* p_1, p_2 e p_3 possuem acordos de envio e recebimento entre si (p_1 envia para p_2 que envia para p_3), formando duas arestas no caminho dos dados (de p_1 para p_2 e de p_2 para p_3). É dado o nome de *peer transmissor inicial* a p_1, *peer intermediário* a p_2 e *peer receptor final* a p_3. A latência entre os *peers* envolvidos no processo representam os pesos nas arestas. Utilizando essas informações, a desigualdade triangular é verificada. Caso a desigualdade não se apresente, e a aresta maior for a correspondente ao acordo

de p_1 para p_2, a troca é realizada, não sendo mais utilizada a aresta de p_1 para p_2 na propagação dos dados. No caso de troca, os acordos entre os três *peers* são modificados. No mesmo cenário, p_1 passa a enviar para p_3, e p_3 passa a enviar para p_2. A inexistência da desigualdade triangular entre esses três pontos (e a exclusão da aresta maior da figura 4.4) faz com que mesmo com um *peer* intermediário entre p_1 e p_3 (que agora recebe de p_2), p_3 passe a receber o conteúdo mais rapidamente. A condição que deve ser satisfeita para a troca ser efetuada corresponde a $RTT_{(p1,p2)} > RTT_{(p1,p3)} + RTT_{(p3,p2)}$.

A figura 4.5 (1) mostra um exemplo de acordos antes do mecanismo ser disparado. A figura 4.5 (2) mostra que, após as medições de RTT, não há desigualdade triangular entre as três arestas. A figura 4.5 (3) mostra os acordos resultantes do disparo do mecanismo.

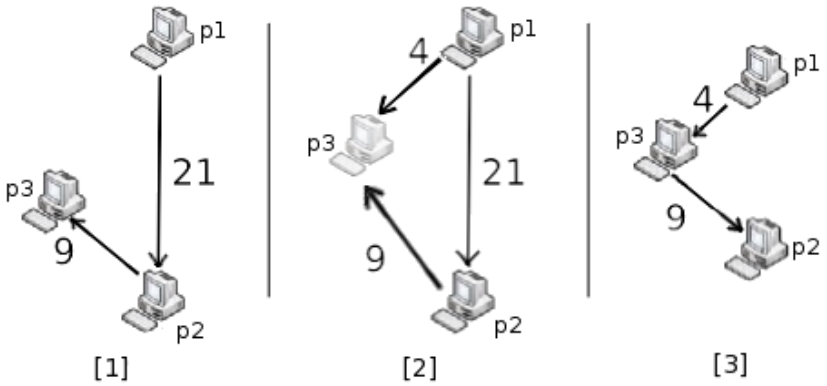

Figura 4.5: Execução do mecanismo de desigualdade triangular por p_3.

O mecanismo baseado na desigualdade triangular compreende várias etapas entre os três *peers*, dadas pelo pseudo-código abaixo. É importante notar que antes do início do processo, o peer inicialmente receptor da mensagem (no cenário anterior, p_3) deve certificar-se que possui banda suficiente para uma transmissão nova (para p_2 no caso do mecanismo ser disparado), caso contrário o processo não é iniciado. O pseudo-código abaixo apresenta as ações tomadas por todos os *peers* envolvidos.

Se p_3 não possuir capacidade de envio adicional então

Não realiza ação nenhuma e sai do algoritmo.

Senão

Mede $RTT_{(p3,p2)}$.

Mede $RTT_{(p3,p1)}$.

Mede $RTT_{(p2,p1)}$.

Se $(RTT_{(p1,p2)} > RTT_{(p1,p3)} + RTT_{(p3,p2)})$ então

p_1 troca o acordo de envio de p_2 para p_3;

p_2 cancela acordo de envio para p_3;

p_3 realiza acordo de envio para p_2.

Algoritmo 1: Pseudo-código do mecanismo de desigualdade triangular.

Quando um *peer* possui vários acordos de envio, múltiplas requisições de realização de troca podem ocorrer em um curto período de tempo. A realização de múltiplas trocas simultâneas, utilizando um mesmo *peer* intermediário ou envolvendo um *peer* receptor pode resultar em acordos inválidos ao término das trocas de acordos. O método adotado para garantir a exclusividade temporária dos *peers* é através de mecanismos de travamento dos *peers* enquanto estiverem com trocas de acordos pendentes. Quando a verificação ocorre, antes das requisições de troca ocorrerem, o *peer* receptor e intermediário verificam se não se encontram travados (ou seja não possuem acordos pendentes), recusando pedidos de troca de acordos em caso afirmativo. Um *peer* travado cumpre todos os acordos estabelecidos, apenas negando a realização de novos acordos (sejam eles devido ao mecanismo de desigualdade triangular ou inserção de novos *peers*). Como o processo envolve diversas etapas, falhas entre os *peers* podem ocorrer antes que o processo se finalize. No caso de falha, os *peers* restantes permanecem travados por um período fixo de tempo (determinado por um parâmetro do protocolo), que deve ser grande o suficiente para permitir que um processo completo possa ser finalizado caso os *peers* não falhem. O tempo máximo

de travamento é dado em função da taxa de atualização periódica dos *peers* (descrito na seção 4.2.3), e caso o destravamento ocorra com o estouro do tempo e haja acordos inválidos, a atualização periódica detectará essa situação e novos acordos serão feitos (quando cabíveis).

O mecanismo de desigualdade triangular pode ser executado em diversos momentos. Duas versões foram desenvolvidas, reativa e proativa, sendo explicadas e comparadas a seguir.

A versão reativa é disparada continuamente até que o *peer* encontre um *peer intermediário* com o qual o mecanismo baseado em desigualdade triangular não é executado. Nesta versão, um *peer* testa a desigualdade triangular quando; (i) seu transmissor direto mudar ou (ii) quando seu transmissor direto notificar que seu próprio transmissor direto foi modificado.

Considere o mesmo cenário descrito acima, onde p_3 é o *peer receptor final*, p_2 o *peer intermediário* e p_1 o *peer transmissor inicial*. Quando p_3 mede seu RTT com p_2 e p_1 e confirma a não existência da desigualdade triangular, p_3 passa a receber o fluxo de p_1 e enviar o fluxo para p_2. p_3 imediatamente mede seu RTT em relação a p_1 e o *peer* com o qual p_1 possui acordo de recebimento (p_0). p_2, por sua vez, informa a todos os *peers* com os quais possui acordos de envio que agora possui um acordo de recebimento com p_3, permitindo a todos eles a medição de seus RTT com p_3. Além disso, quando p_3 realizar um acordo de recebimento com um *peer* onde a desigualdade triangular esteja presente, p_3 informa a todos os *peers* com os quais possui um acordo de recebimento que seu transmissor mudou.

É possível perceber que a condição para o mecanismo reativo ser disparado é uma consequência do sucesso de troca envolvendo seu transmissor direto. Sendo assim, um *peer* realiza sucessivas trocas de acordos até encontrar uma posição estável no grupo (quando a condição de desigualdade triangular estiver presente), não executando o mecanismo baseado em desigualdade triangular até que o *peer* com o qual possui um acordo de recebimento mude.

A versão proativa é uma versão que dispara medições periodicamente durante a execução

do protocolo. Ao invés de executar o mecanismo sempre que um nó transmissor é substituído por outro, na versão proativa cada *peer* checa os *peers* com os quais possui acordos de recebimento em intervalos de tempo (determinados como parâmetro do sistema).

Enquanto na versão reativa a convergência é rápida, na versão proativa essa mesma convergência pode levar vários intervalos de tempo. Esta característica é um ponto positivo para a implementação reativa caso em algum momento a rede se estabilize, mas caso o desempenho dos *peers* varie à medida que a transmissão ocorra, a versão proativa é mais eficiente. Na existência de variações constantes na rede, o algoritmo reativo irá realizar trocas sempre que uma modificação for detectada, podendo submeter os *peers* a diversas trocas influenciadas por anomalias na rede. Caso a rede se estabilize, então a versão proativa irá submeter os *peers* a verificações iniciais desnecessárias, uma vez que a estabilidade da rede foi encontrada e qualquer tentativa de troca será negada.

Neste trabalho, propõe-se que seja utilizada uma combinação da versão reativa com a versão proativa. Para não incorrer em grande sobrecarga na inserção ou no momento imediato após uma troca, *peers* que realizam trocas não informam aos receptores, cabendo a esses *peers*, ao dispararem sua própria versão proativa, a detecção de mudanças. Esta abordagem permite uma convergência rápida inicial (quando um novo acordo é estabelecido), sem impactos muito grandes devido a anomalias da rede.

4.4 Avaliação Experimental

Esta seção descreve a avaliação experimental realizada através de simulação do protocolo proposto. As simulações foram realizadas utilizando o simulador Java PeerSim [43], no qual as entidades do sistema são representadas por objetos Java. As simulações ocorreram no modelo de disparo de eventos, onde uma mensagem enviada de uma entidade a outra é representada como um evento escalonado no simulador. Essa abordagem garante que não há perdas de mensagens durante o transporte. A topologia da rede aplica os conceitos de redes *small world* apresentados em [44]. Como falhas durante o transporte foram desconsideradas, a rede de roteadores gerada apenas auxilia na medição de latência entre os nós terminais.

Em todas as simulações, o comportamento dos nós foi simulado baseando-se em estudos sobre distribuição de fluxos presente em [45]. Sobre esse estudo, foi considerado que grande parte dos *peers* se conectam no início da transmissão, grande parte deles permanecem durante toda a transmissão e as inserções ocorrem segundo o efeito *flash crowd*. Esse efeito descreve o comportamento de inserções de *peers* em rajadas, e não linearmente no decorrer da simulação. As saídas dos *peers* se deram de maneira arbitrária durante a execução, e foram consideradas falhas do tipo *crash*, onde um nó deixa de funcionar permanentemente e sem notificação prévia.

Em relação à capacidade de recebimento, foi estipulada banda suficiente para os nós receberem o fluxo integralmente, assim como foi atribuído a cada nó banda suficiente para o recebimento e envio das mensagens de controle. Já a capacidade de envio foi definida de acordo com medições presentes em [39], atribuindo o limite máximo de 8 envios (pois uma conexão cujo *upload* é 512kbps (*kilobits* por segundo) suporta no máximo 8 envios de fluxo a 64kbps (*kilobits* por segundo)). Esse cenário incluiu também *peers* que contribuem pouco ou não contribuem para a propagação do conteúdo, e são muito comuns em redes Torrent [46]. Estes *peers* apresentam pouca capacidade de *upload*, não apresentando, no entanto, comportamento malicioso.

As mesmas redes foram simuladas para execução de todos os protocolos avaliados, sendo também as mesmas características consideradas em todas as medidas para os diversos protocolos, por exemplo banda de envio, latência entre os nós, e entrada e saída da rede. Essa abordagem permite a avaliação do comportamento dos protoolos sujeitos às mesmas condições da rede.

Os protocolos escolhidos para as avaliações experimentais em relação ao PALMS foram os protocolos Narada [29] (descrito na seção 3.3.4) e o sistema proposto em [40], através do *Streaming POTS* (descrito na seção 3.4.2). A escolha pelo protocolo Narada se deve ao fato de ser um dos pioneiros na utilização de *multicast* em nível de aplicação, além de representar de forma mais fiel a camada de rede na camada de aplicação (simulando cada *peer* como um roteador da camada de rede). O Streaming POTS foi escolhido por permitir a disseminação de conteúdo para *peers* não confiáveis, e apesar de não ser explicitamente

um protocolo ALM, mantém uma rede *mesh* (acoplada) para tolerância a falhas. Nos protocolos PALMS e Narada, o papel do *tracker/rendezvous point* foi desempenhado por uma entidade separada do servidor, embora sua junção seja possível.

Nos experimentos, o mesmo membro da rede que desempenha a função de servidor de um protocolo desempenha também a função de servidor nos outros protocolos (o mesmo ocorrendo com o *tracker* e o *rendezvous point*). Isso permite a sincronização de ações concomitantes dos diferentes protocolos, como por exemplo a produção do fluxo. Quando uma fatia é criada no servidor do Streaming POTS, uma fatia é criada pelo servidor do PALMS assim como um fluxo é distribuído pelo servidor do Narada. Como o protocolo Narada não é voltado especificamente para a distribuição de conteúdo proveniente de uma única fonte, o servidor é implementado em um *peer*, cuja banda de envio foi modificada para permitir o envio ilimitado de conteúdo, sendo ele o único transmissor inicial de mensagens. O experimento 1 comparou o comportamento do PALMS com e sem o mecanismo de desigualdade triangular. O experimento 2 comparou o comportamento do PALMS em relação ao NARADA. O experimento 3 comparou o comportamento do PALMS em relação ao Streaming POTS.

4.4.1 Experimento 1

Este experimento teve como objetivo atestar a efetividade do mecanismo utilizando a desigualdade triangular no PALMS. Para isso, foi desenvolvido um protótipo do protocolo sem o mecanismo de troca [47], e seus resultados foram comparados. Como visto na seção 4.3, na versão com o mecanismo de desigualdade triangular foi utilizada uma combinação das versões proativa e reativa.

Cada simulação teve duração de 1 hora, sendo executadas 10 simulações no total. Os resultados do experimento correspondem aos valores médios (juntamente com os intervalos de confiança) dessas execuções. As medições ocorreram em intervalos de 6 segundos. Quando cabível, os valores representados resultam na soma dos valores durante o intervalo de 6 segundos posteriormente divididos por 6, resultando em uma estimativa de valores por segundo.

Como parâmetros da rede, foram utilizados 1024 nós (incluindo servidor e *tracker*). Entre 0 e 200 segundos, foram inseridos 128 *peers* em rajadas em intervalos de tempo de 8 segundos até a capacidade da rede ser atingida (1022 *peers*). 300 segundos após o início da simulação, 128 *peers* arbitrários foram removidos, e a simulação continuou com os 894 *peers* restantes.

Em relação aos parâmetros do PALMS com e sem o mecanismo de desigualdade triangular, ambos utilizaram os mesmos parâmetros (exceto os parâmetros utilizados no mecanismo de desigualdade triangular). Cada lista de *peers* enviada pelo *tracker* conteve 32 *peers*, sendo este tamanho suficiente para não serem necessárias reenvios da lista de *peers* na maioria dos casos. A taxa de atualização periódica dos RTT foi de 6 segundos; após 2 atualizações consecutivas sem resposta os acordos com o *peer* são encerrados. O tempo máximo de recebimento de uma fatia foi fixado em 1.5 segundos, dando aos *peers* tempo suficiente para a detecção e requisição das fatias antes da produção de uma nova fatia na maioria dos casos. Em relação ao mecanismo de travamento presente na desigualdade triangular, o tempo máximo para a finalização de uma troca de acordo foi fixado em 10 segundos, embora não foram detectados casos do estouro deste tempo. Estes parâmetros foram definidos experimentalmente visando o balanceamento entre a sobrecarga nos *peers* e a manutenção da rede em tempo hábil, sem que ocorresem perdas de conteúdo. Para os gráficos a seguir, considere PALMS SDT a versão do PALMS sem o mecanismo de desigualdade triangular, e PALMS CDT a versão com o mecanismo proposto.

A figura 4.6 mostra a distribuição do tempo de inicialização dos *peers*. O tempo de inicialização compreende o intervalo de tempo decorrido desde o primeiro contato do *peer* com o *tracker* até o recebimento da primeira fatia. A figura representa curvas de distribuição *kernel*, que são representações contínuas de um histograma, representando a frequência com que os diferentes intervalos tempos de inicialização ocorreram. No gráfico, é possível perceber que grande parte dos *peers* teve seu tempo de inicialização próximo de 600 milisegundos. Observando as duas curvas, é possível perceber uma pequena variação entre os tempos de inserção, ocasionados pela execução inicial do mecanismo de desigualdade triangular.

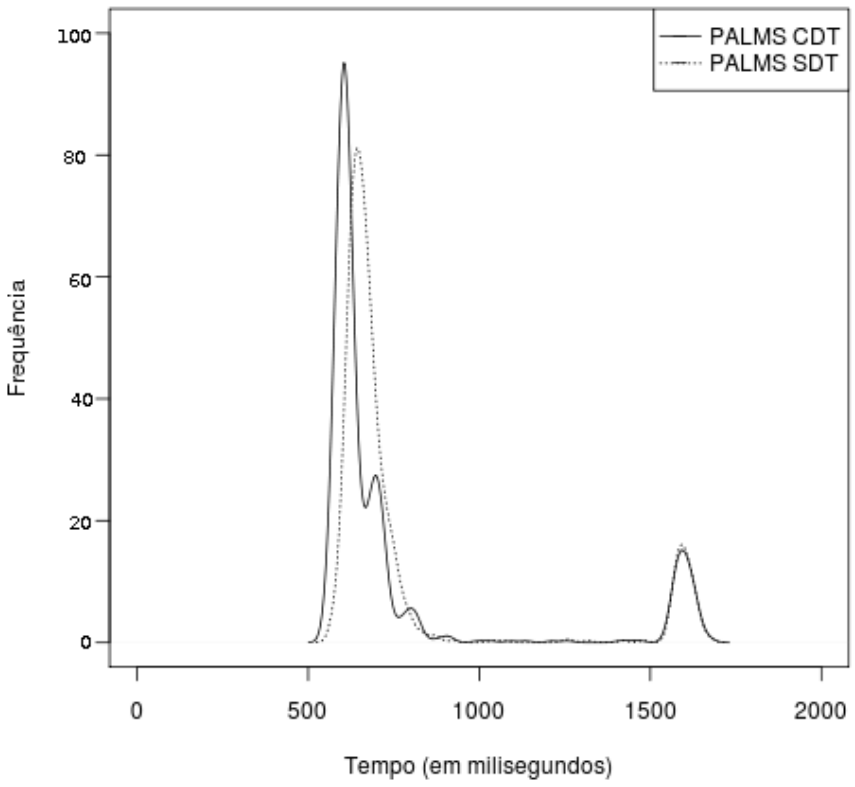

Figura 4.6: Tempo de inicialização média dos *peers*.

A figura 4.7 mostra o tempo médio e o intervalo de confiança no recebimento das fatias, do momento em que o servidor distribui as fatias ao momento em que elas são recebidas pelos *peers*. Tanto o CDT quanto o SDT são representado por três curvas cada. O valor da média, apresentado pelas linhas pontilhadas, e seus respectivos intervalos de confiança. Devido às trocas de acordos periódicas dos *peers*, é possível notar que há um ganho na presença do mecanismo de desigualdade triangular pois este permite pequenas adequações às variações da rede. Como a rede é dinâmica, a tendência é o aumento da diferença entre os dois protocolos. A figura 4.7 ainda mostra o comportamento dos *peers* na ocorrência de falha súbita de um *peer*. No momento em que há a saída, na fatia 300, há um aumento temporário subito no tempo de recebimento, em decorrência do estouro do *timeout* de atraso. A falha na entrega é rapidamente solucionada, em ambos os casos.

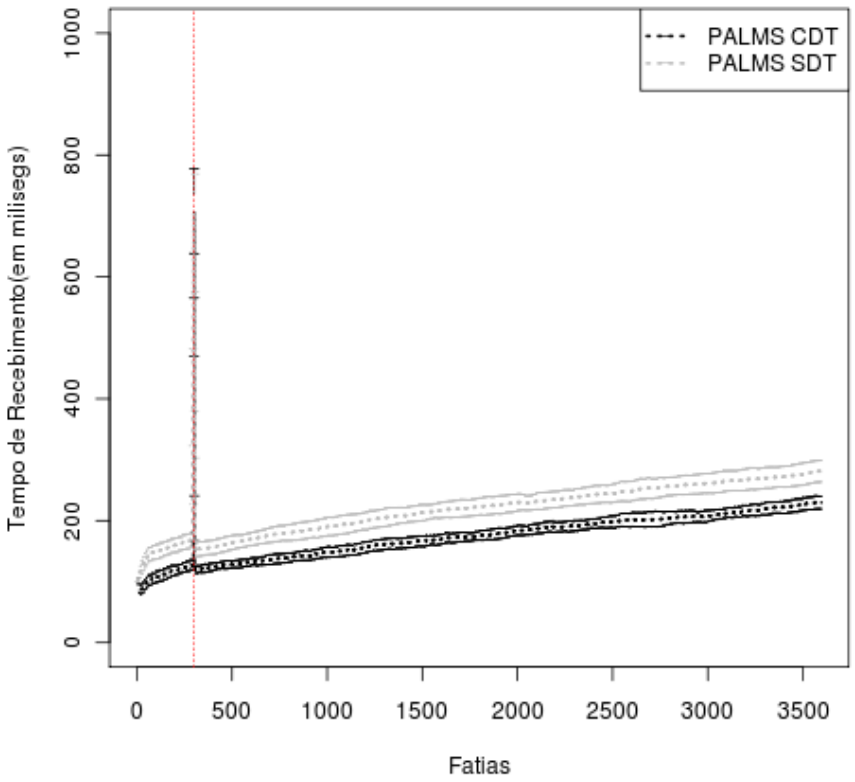

Figura 4.7: Tempo médio de recepção de fatias.

O ganho obtido no recebimento das fatias visto na figura 4.7 tem um custo que pode ser medido através das mensagens de controle adicionais utilizadas pelo mecanismo de desigualdade triangular. A figura 4.8 demonstra a sobrecarga de controle em *kilobytes* enviados por segundo. A diferença apresentada pelo gráfico, decorrente do mecanismo de desigualdade triangular, é mínima se comparada ao *overhead* imposto pelos outros protocolos, como é observado nos experimentos a seguir. É importante lembrar que, como as medições ocorreram a cada 6 segundos, oscilações presentes durante os 6 segundos foram amenizados pela média dos valores no intervalo. Pela figura, é possível observar que em ambas versões do protocolo, o uso de banda para mensagens de controle se mantém quase constantes. Isso se deve ao fato de cada *peer* manter atualizações apenas dos *peers* com os quais possui acordo, além da sobrecarga adicional devido ao mecanismo de desigualdade triangular.

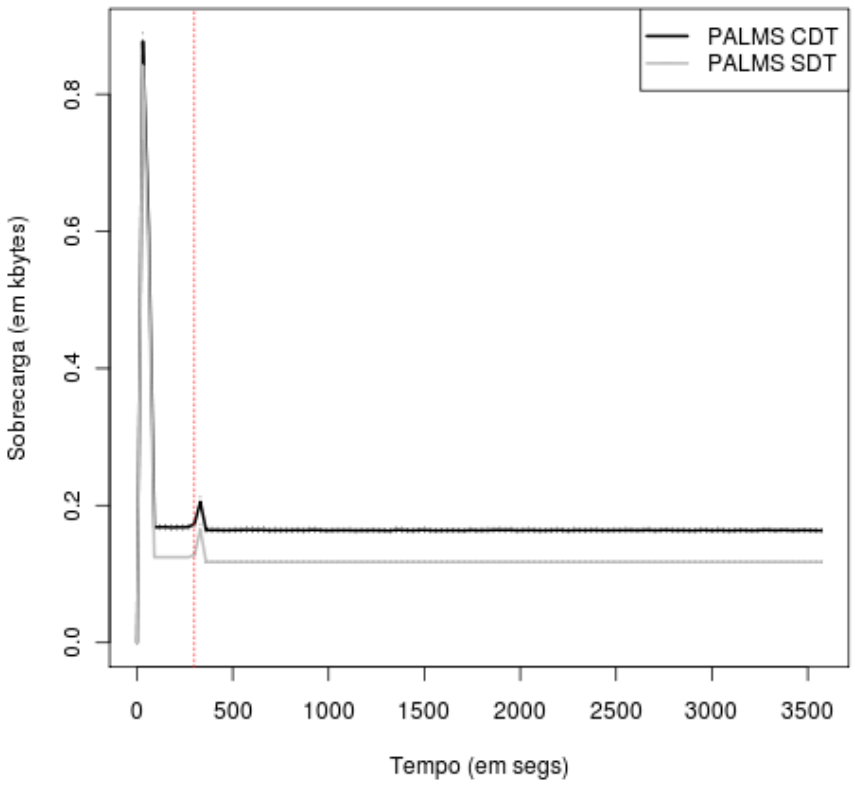

Figura 4.8: Uso de banda no envio de mensagens de controle pelos *peers*.

A figura 4.9 mostra o acréscimo na banda utilizada pelo servidor para o envio de mensagens de controle. Um acréscimo semelhante ocorre por parte do servidor, uma vez que ele pode ser o transmissor final em uma processo de troca de acordos que envolva seu receptor direto. Entretanto, como o transmissor final participa apenas do último estágio da troca de acordos, o impacto no servidor é menor que nos demais *peers*. O intervalo de confiança deste experimento foi de aproximadamente 1 kilobyte/segundo para mais e para menos, e não foi representado no gráfico para facilitar sua visualização.

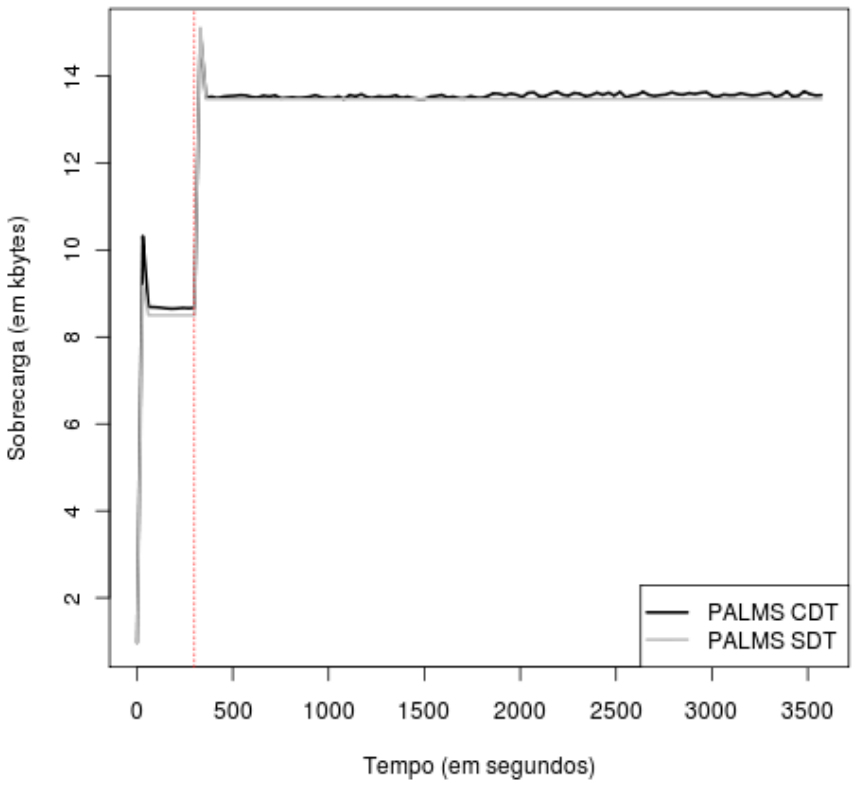

Figura 4.9: Uso de banda em mensagens de controle enviadas pelo servidor.

Como não houve restrições em relação à quantidade de banda de envio do servidor, toda requisição de criação de grupo foi aceita. Em um cenário real, após o servidor atingir sua capacidade máxima, requisições de acordo posteriores seriam negadas, e ao invés de criar um novo grupo os *peers* requisitantes buscariam por grupos existentes. Como houveram diversas falhas simultâneas 300 segundos após o início da simulação, diversos novos grupos foram formados.

A figura 4.10 demonstra o aumento do número de grupos na ocorrência de falhas quando não há restrição de banda no servidor. Após a falha dos *peers*, podem ser necessárias a criação de novos grupos para acomodar os *peers* órfãos. É possível notar que a quantidade de envios realizados pelo servidor ficou bem próximo, não sendo influenciado de maneira significativa pelo mecanismo de desigualdade triangular. O intervalo de confiança presente neste experimento foi de 5 acordos para mais ou para menos, e não foi representado para facilitar a visualização do gráfico.

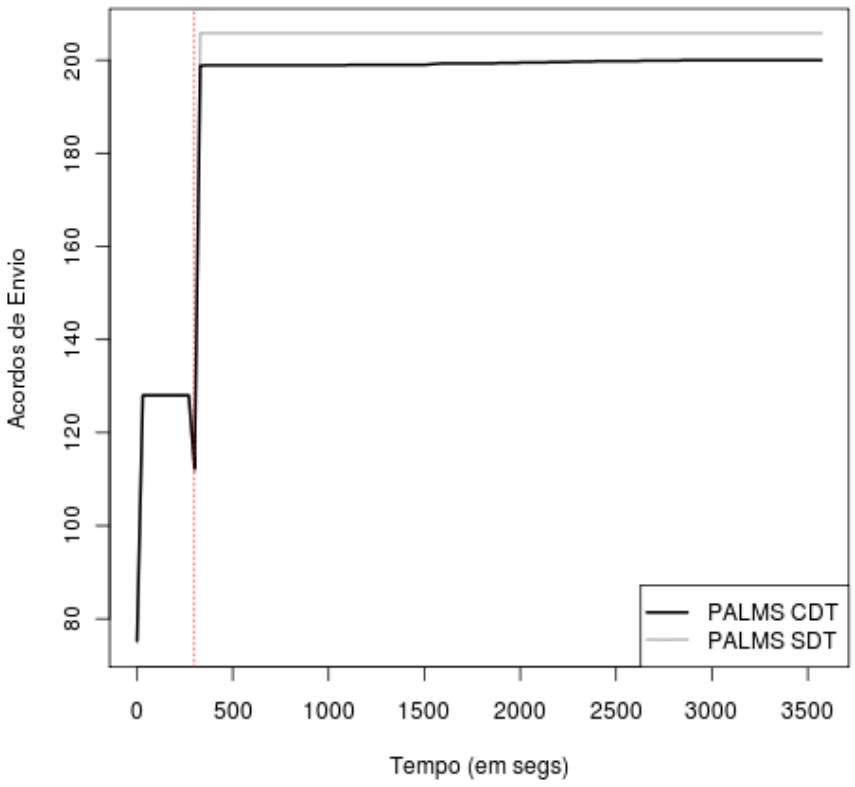

Figura 4.10: Acordos de envio mantidos pelo servidor.

Este experimento demonstrou o ganho obtido pelo mecanismo de desigualdade triangular a um baixo custo em mensagens de controle. Em ambientes dinâmicos como a Internet, podem haver trocas de acordos influenciadas por estados temporários na rede, que serão desfeitas quando o mecanismo de desigualdade triangular for testado após retomada da normalidade.

4.4.2 Experimento 2

Este experimento tem como objetivo comparar o comportamento do PALMS em relação ao protocolo Narada. As métricas utilizadas foram o tempo médio de inserção de um nó, tempo médio de recebimento do conteúdo, *overhead* causado pelas mensagens de controle, quantidade de transmissões realizadas pelo *servidor* e quantidade de fluxo perdido. Neste experimento, o tempo de inserção de um nó compreende o tempo decorrido entre a requisição de inserção ao *tracker* ou *rendezvous* e a primeira parte do fluxo recebida.

Cada simulação teve duração de 10 minutos, sendo executadas 10 simulações no total. Os resultados do experimento correspondem aos valores médios e intervalos de confiança de 95% entre essas execuções. As medições ocorreram em intervalos de 5 segundos, sendo os valores representados iguais às somas dos valores obtidos durante esse intervalo e posteriormente divididos por 5, resultando em uma estimativa de valores por segundo.

Como parâmetros gerais da rede, foram utilizados 128 nós (incluindo servidor e *tracker* ou *rendezvou*). O número baixo de nós, bem como o tempo reduzido de simulação, devem-se ao fato das trocas de mensagens entre os *peers* do Narada resultarem em $O(N^2)$ mensagens de controle agregadas [31], cada uma delas contendo uma lista quase completa dos *peers* presentes na rede. Para este experimento, foi atribuído um tempo inicial de até 150 segundos para a inserção dos 128 *peers*, em rajadas de 16 *peers*. Após esse tempo, os *peers* puderam convergir suas tabelas de roteamento e informações de atualização de outros *peers* até 200 segundos do início da simulação, onde 16 *peers* arbitrários falharam. A simulação foi encerrada 600 segundos após seu início.

Em relação aos parâmetros dos protocolos, os seguintes valores foram utilizados. Tanto no PALMS quanto no Narada, a lista de *peers* de tamanho 16 foi suficiente para para não incorrer em grandes re-envios da lista pelo *tracker*. A taxa de atualização periódica de RTT no PALMS foi de 5 segundos, mesmo intervalo utilizado para as mensagens de atualização de RTT no Narada. O tempo máximo de finalização do mecanismo de desigualdade triangular foi estipulado em 5 segundos, mas não houve destravamentos devido ao estouro do *timeout*. O tempo máximo de recebimento das fatias no PALMS foi estipulado em 1.3 segundos, estipulado de modo a restringir o tempo de recebimento máximo dos *peers*, permitindo até mesmo o cancelamento de acordos caso um dos *peers* não entregue a fatia em tempo hábil. A quantidade de mensagens de atualização não-recebidas para um *peer* ser considerado falho foi estipulada em 1, pois a taxa de atualização de 5 segundos é grande o suficiente para não exceder a latência entre dois *peers* arbitrários.

Em relação ao Narada, foram considerados 3 atualizações não recebidas para que um *peer* seja considerado falho (pois na ocorrência de uma partição uma atualização será perdida, sendo outra atualização necessária para o reparo da partição, restanto um

intervalo para a mensagem de atualização após o reparo ser recebida). O tempo de permanência de *peers* falhos nas tabelas de atualização foi de 12 intervalos de atualização. Como nesta rede não há reinserções de *peers* após o período de entrada, esse valor foi grande o suficiente para não haver reinserções errôneas de *peers* já considerados falhos. Para haver uma rápida convergência entre as tabelas de roteamento, foi considerado um intervalo de atualização das tabelas igual ao de atualização dos *peers*, sendo portanto 5 segundos. Em relação à checagem de particionamento, um *peer* realiza a checagem a cada 2 atualizações, podendo detectar e unir partições antes do estouro do *timeout* (que é de 3 atualizações). Para não sobrecarregar os *peers*, o intervalo de refinamento da rede foi de 60 segundos, onde um *peer* descarta outro quando o utiliza para alcançar menos de 10% da rede (cerca de 6 nós). O grau de utilidade foi estipulado de modo a não permitir a imediata re-inserção do último *peer* cuja conexão foi suspensa. A cada intervalo de verificação de utilidade (que ocorre ao mesmo tempo que o refinamento), um *peer* realiza a verificação em outros 10 *peers*, que ainda não fazem parte de sua vizinhança.

Com intervalos de produção de fluxo de 1 segundo, quando um *peer* é inserido na rede ele não passa a receber o fluxo imediatamente após sua inserção, pois depende pelo menos da produção da próxima fatia. No PALMS, como visto no experimento 1, é necessária apenas a inserção com sucesso do *peer* em um grupo, e a fatia subsequente à essa inserção já será repassada ao *peer*. No Narada, a recepção do fluxo depende da inserção do *peer* na rede *mesh* de controle e da atualização das tabelas de roteamento de seus vizinhos. O cálculo de uso de banda de envio no Narada não é direto como no PALMS (pois no PALMS a árvore de propagação é explícita), e a quantidade de envios de um *peer* depende do remetente da mensagem e da tabela de roteamento de outros *peers*. Como o único transmissor de dados é o servidor (embora hajam transmissões de mensagens de controle provenientes de outros *peers*), um *peer* pode aceitar outro *peer* como vizinho na rede quando, no momento da requisição de inserção, ele não esteja enviando mensagens provenientes do servidor a mais *peers* que sua capacidade máxima. Isso pode ocasionar a aceitação de um *peer* e posterior recusa (quando for detectado que o *peer* deveria enviar mais mensagens que sua capacidade). A figura 4.11 mostra um histograma de frequência

de tempo de inserção, onde grande parte dos *peers* recebe uma fatia em até 2 intervalos de produção após sua entrada na rede, enquanto alguns *peers* do Narada têm a recepção atrasada devido ao intervalo de atualização das tabelas. O comportamento do PALMS se assemelha ao do experimento 1. No caso do Narada, alguns *peers* demoraram a receber o conteúdo devido ao tempo de convergência das tabelas de roteamento (definido como 5 segundos). Por haver inserções em rajadas, alguns *peers* necessitaram de até duas atualizações nas tabelas de roteamento, recebendo o conteúdo após cerca de 10 segundos (2 intervalos de atualização de tabelas de roteamento) de sua inserção.

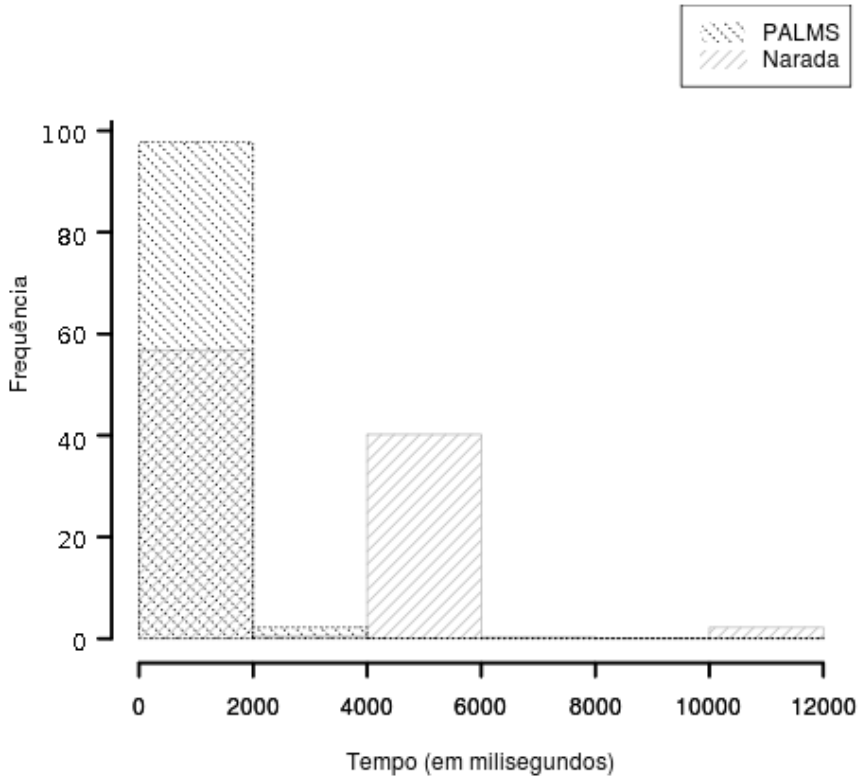

Figura 4.11: Tempo de inicialização média dos *peers*.

A figura 4.12 apresenta um gráfico de tempo de recebimento das fatias. As linhas mais fortes representam o tempo médio, sendo representados seus intervalos de confiança (de mesma cor). É possível notar que o tempo médio de recepção dos dois protocolos é bastante semelhante, embora o mecanismo de adição de rotas por utilidade do Narada resultar em uma menor variação entre os experimentos. Além disso, na ocorrência da

saída de *peers* aos 200 segundos não aumentou a latência média dos *peers* no Narada. Esse comportamento é explicado na figura 4.13.

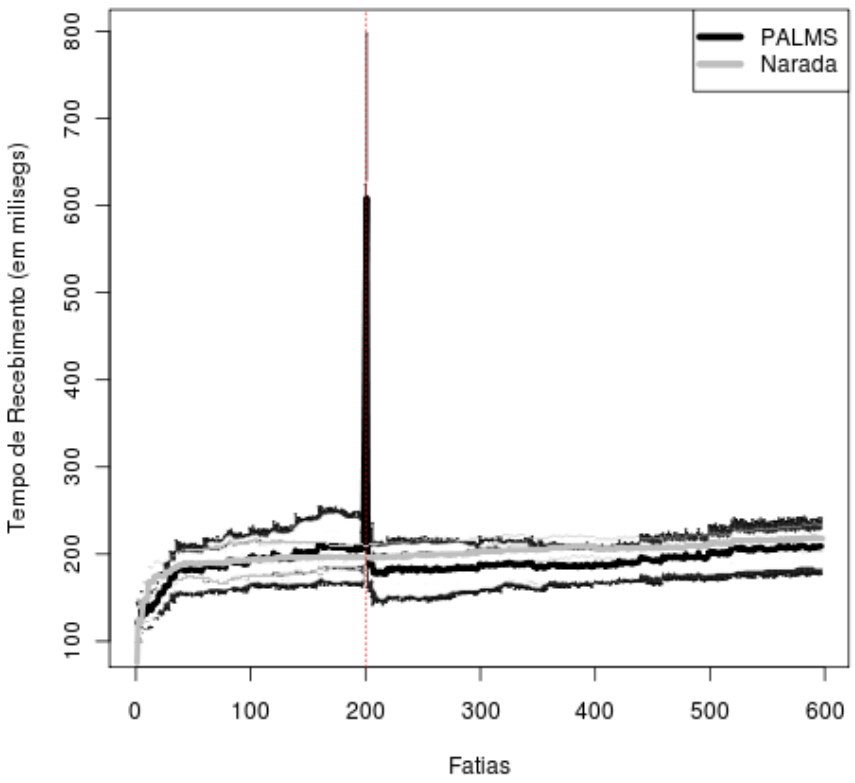

Figura 4.12: Tempo médio de recepção de fatias.

Como é possível observar na figura 4.12, no momento da falha dos *peers* há um pico no tempo de recebimento no PALMS. Isso se deve à quebra dos acordos de envio pelos *peers* faltosos, assim como no experimento 1, e o atraso de 1 ou 2 fatias para os *peers* dependentes deles nos grupos. Esse pico não ocorre no Narada, pois a falha nos *peers* não é detectada até o estouro do tempo limite de recebimento de atualização, quando o processo de verificação de *peers* é iniciado. Enquanto as tabelas de roteamento não forem atualizadas para refletir as falhas, há perda de mensagens por parte dos *peers*. A figura 4.13 mostra o número de *peers* que não receberam a fatia.

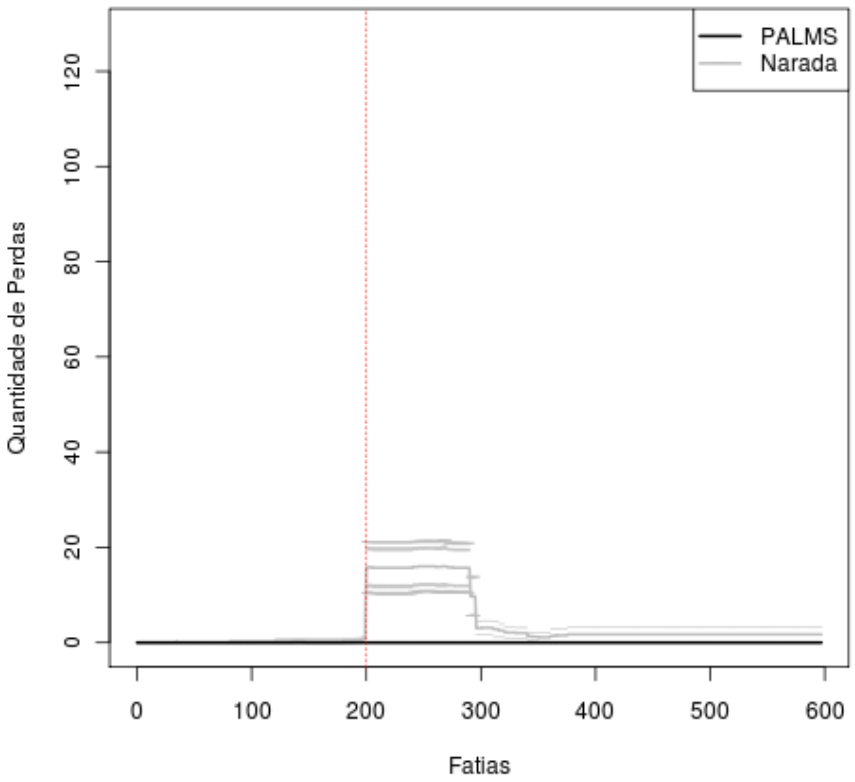

Figura 4.13: Quantidade de fatias não recebidas.

A figura 4.14 mostra a utilização de banda devido às mensagens de controles. O Narada foi um dos protocolos escolhidos para a comparação por melhor representar a topologia de rede na camada de aplicação, onde cada *peer* corresponde a um *roteador* na camada de rede. Entretanto, o envio das tabelas de roteamento e as mensagens de atualização de todos os demais membros da rede podem gerar um tráfego considerável, como pode ser observado na figura 4.14, onde mesmo após a estabilidade da rede, o consumo de banda devido às mensagens de controle no Narada se manteve alto, enquanto o PALMS manteve seu comportamento semelhante ao encontrado no experimento 1.

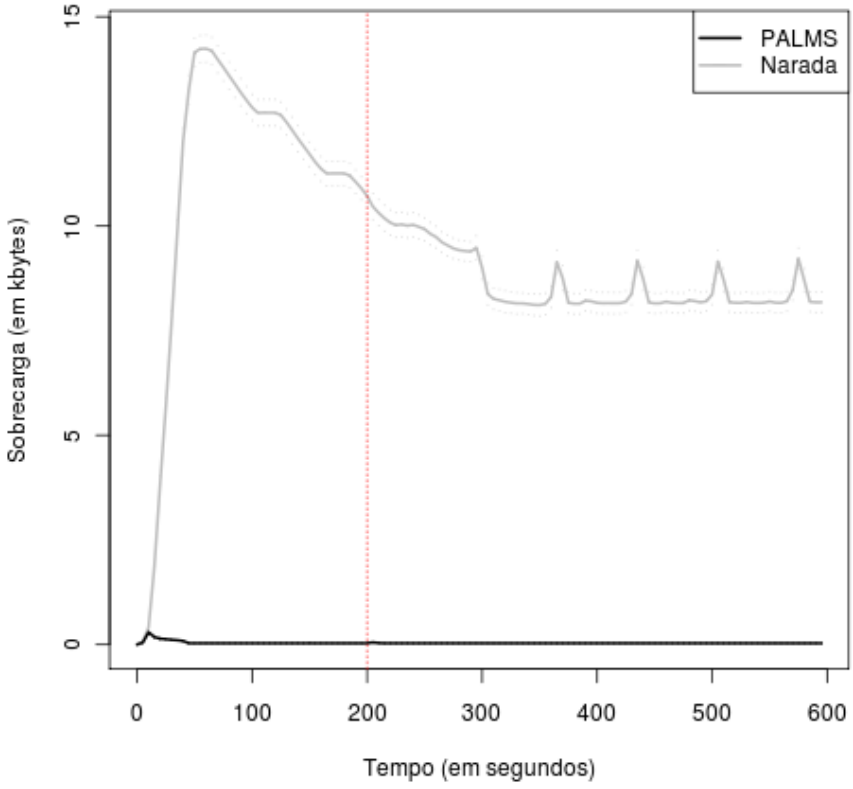

Figura 4.14: Uso de banda no envio de mensagens de controle pelos *peers*.

Em relação ao servidor, foi atribuída uma capacidade de banda ilimitada para envio de fluxos. A figura 4.15 demonstra o número de envios diretos do servidor por fatia. Percebe-se que o PALMS tem o mesmo comportamento do Experimento 1, criando novos grupos na saída de *peers* e realizando novos acordos com o servidor. No Narada, a saída de *peers* só é detectada pelo servidor caso os *peers* faltosos influenciem em sua tabela de roteamento (seja pelo servidor enviar fluxo diretamente para esses *peers* ou pela falha deles ocasionar a adição de novos vizinhos ao servidor). A defesagem de tempo entre a falha nos *peers* e sua repercusão nos envios do servidor deve-se à estimativa de tempo máximo que um *peer* deva ser considerado falho antes de sua remoção da rede. Enquanto a falha não é detectada, rotas inválidas podem estar em funcionamento ocasionando a perda de fatias (visto na figura 4.13). Além disso, os mecanismos de checagem de utilidade e consenso dos *peers* pode ocasionar aumento ou decréscimo nas transmissões diretas do servidor, como visto nos instantes finais da simulação.

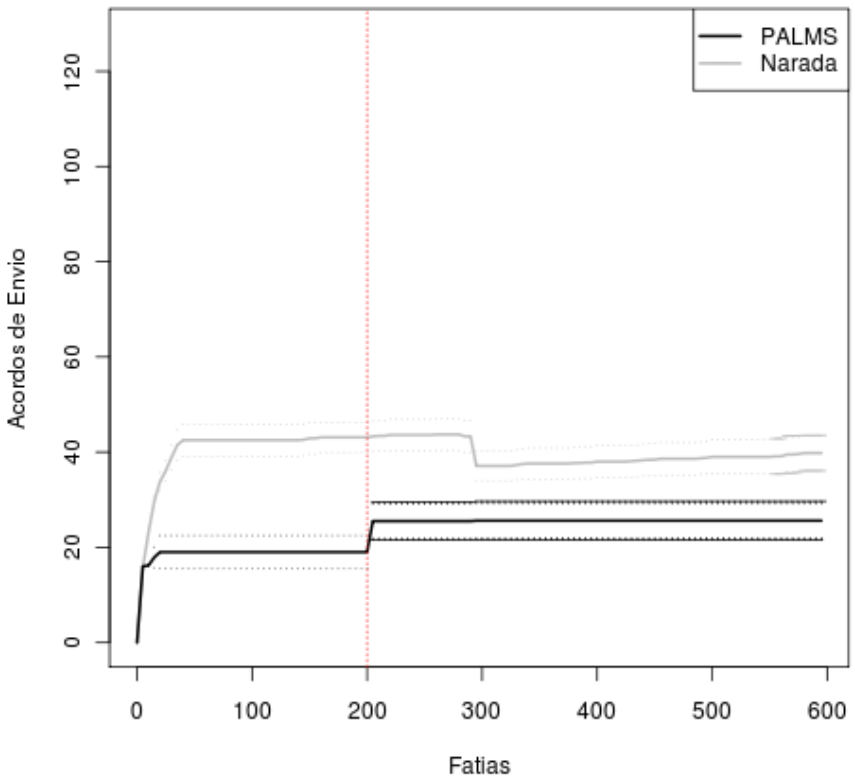

Figura 4.15: Número de envios do servidor por fatia.

Pelo fato do servidor ser representado como um *peer* qualquer no Narada, a utilização de banda em mensagens de controle segue o mesmo comportamento. Entretanto, pelo servidor possuir banda de envio infinita, a requisição de inserção por um *peer* arbitrário sempre será aceita, resultando em um maior número de vizinhos do servidor que a média dos demais *peers*. A figura 4.16 mostra uma comparação de uso de banda em mensagens de controle pelo servidor no PALMS e no Narada, tendo grande variação entre as simulações no último caso mas sendo sempre superior ao PALMS.

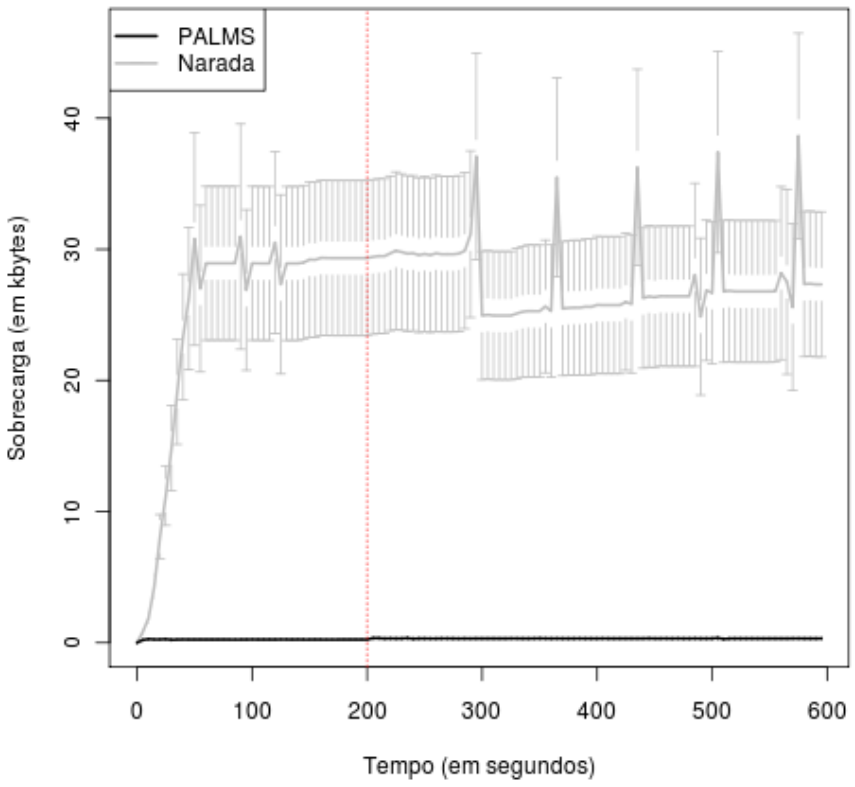

Figura 4.16: Uso de banda em mensagens de controle enviadas pelo servidor.

Este experimento permitiu demonstrar que, apesar de semelhantes no tempo de recebimento médio das fatias, o PALMS consegue utilizar a rede de maneira mais eficaz não submetendo os *peers* e o servidor a um alto consumo de banda pelas mensagens de controle. O tempo de recuperação de falhas no Narada depende da taxa de atualização dos nós. Uma taxa de atualização alta resulta em menor tempo de recuperação, mas ocasiona alto consumo de banda em mensagens de controle, sendo o oposto é verdadeiro. No PALMS, o tempo de recuperação dura em média 2 a 3 fatias (através do mecanismo de *notify-emergency*). Uma implementação do Narada sugere a transmissão de uma mensagem pelo *peer* falho inviabilizando seu uso em qualquer rota, mas como o tipo de falha adotado nas simulações foi o de falha súbita, essa abordagem não pode ser utilizada.

4.4.3 Experimento 3

O experimento 3 visa demonstrar o comportamento do PALMS em relação ao Streaming POTS do trabalho apresentado em [40]. As métricas utilizadas foram o tempo de inserção de um nó (tempo decorrente entre o primeiro contato com servidor, *tracker* ou *rendezvous point*, e o primeiro fluxo recebido), tempo médio de recebimento do fluxo em relação ao servidor, o *overhead* causado pelas mensagens de controle, a carga imposta ao servidor (*controle* e *dados*) e a quantidade de fluxo perdido.

Cada simulação teve duração de 20 minutos, sendo executadas 10 simulações com sementes aleatórias diferentes. Os resultados obtidos são valores que representam a média (com o respectivo intervalo de confiança) dos experimentos, quando cabíveis. As medições ocorreram em intervalos de 5 segundos, sendo os valores representados iguais às somas dos valores obtidos durante esse intervalo e posteriormente divididos por 5, resultando em uma estimativa de valores por segundo.

Em relação às configurações da rede, os estágios foram divididos entre período de inserção, quebra e reestabelecimento/normalização. Este cenário permite avaliar o comportamento dos nós quando os eventos pré-definidos ocorrem. No período de inserção, que durou do instante 0.5 segundos até 3 minutos, 128 *peers* arbitrários foram inseridos na rede em intervalos de 8 segundos, até atingir-se a capacidade máxima de 1024 *peers* na rede. No instante de 6 minutos, 128 nós arbritrários foram removidos da rede de maneira abrupta, sendo então iniciada a fase de reestabelecimento e normalização com os *peers* restantes até o final da simulação.

Em relação às configurações dos protocolos, admitiu-se 16 *peers* enviados a cada lista do *tracker*. Para as janelas de recepção de fatias do POTS, foram consideradas 20 fatias. Em relação às taxas de atualização, o PALMS teve sua taxa de atualização imposta em 6 segundos (com *timeout* de 2 intervalos), enquanto no POTS essa taxa foi de 30 segundos para a atualização de RTT e 60 segundos para a atualização do tamanho da vizinhança. O tamanho mínimo da vizinhança de um *peer* no Streaming POTS foi de 16 *peers* (o tamanho da lista enviada pelo *tracker*), enquanto o tamanho máximo foi estipulado em 128 nós, próximo de 10 por cento da rede. O tempo máximo de espera no PALMS foi de

1.3 segundos, assim como o intervalo de consumo do bloco no Streaming POTS. O tempo de espera máximo do Streaming POTS no esvaziamento do buffer foi de 0.5 segundos, para possibilitar o recebimento da fatia em tempo hábil. O tempo máximo de finalização do mecanismo de desigualdade triangular foi estipulado em 5 segundos, mas não houveram destravamentos devido ao estouro do *timeout*. No Streaming POTS, foram utilizados 216 acordos espontâneos (para se igualar ao número de grupos obtidos no PALMS experimentalmente). Acordos espontâneos foram concedidos na fatia 10, notificações de novos acordos ocorreram na fatia 20, e o tempo máximo para finalização dos acordos foi de 90 segundos após o início dos acordos. A duração dos acordos foi de 120 fatias. Os parâmetros foram definidos experimentalmente para refletir o comportamento esperado dos protocolos, sendo explicados nos gráficos a seguir.

Em ambos os protocolos, os servidores produziam fatias a cada 1 segundo. No Streaming POTS, cada fatia foi composta por apenas um bloco.

A figura 4.17 mostra a distribuição do tempo de inicialização dos *peers*. A métrica é a mesma adotada no primeiro experimento. Como pode-se notar pelo histograma, todos os *peers* no Streaming POTS possuem tempo de inicialização entre 500 milisegundos e 1 segundo. Isso se explica pois no Streaming POTS a entrada dos *peers* se dá pelo servidor, e ao receber a primeira requisição de entrada um acordo temporário (de vigência igual a um acordo espotâneo) é estabelecido. Já no PALMS, é possível ver uma pequena porcentagem de *peers* têm a inicialização entre 1,5 e 2 segundos, devido ao atraso ocasionado pela necessidade de descoberta de um grupo disponível. A solução adotada pelo Streaming POTS diminui substancialmente o tempo de inserção de um *peer*, mas na ocorrência de inserções em rajadas pode haver sobrecarga temporária no servidor.

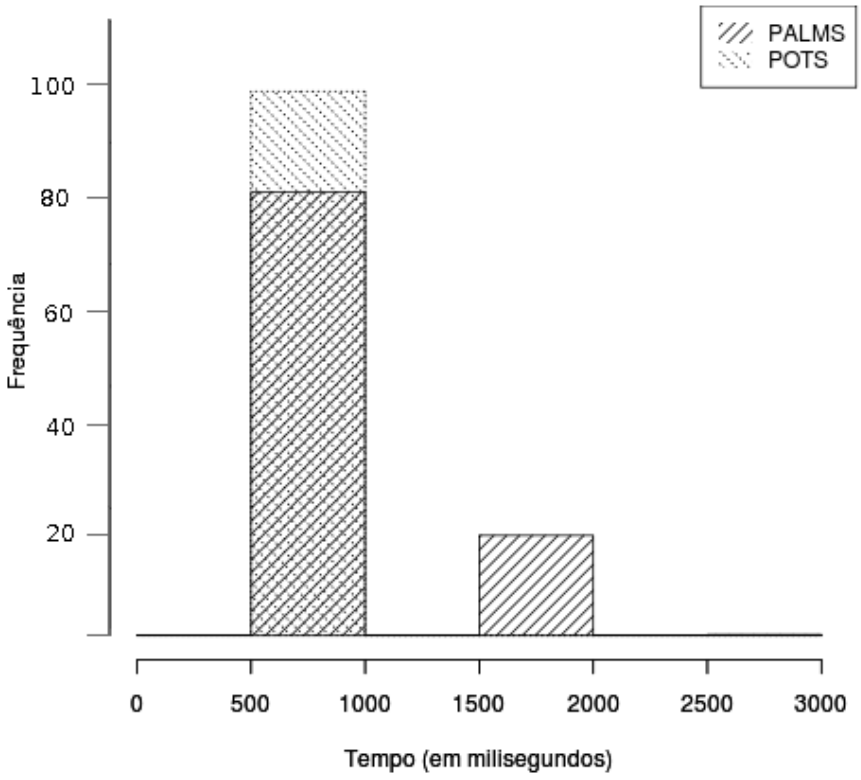

Figura 4.17: Tempo de inicialização média dos *peers*.

A figura 4.18 apresenta um gráfico de tempo médio e intervalo de confiança de recebimento das fatias. No período de inserção a latência média de recebimento no Streaming POTS é semelhante ao de um sistema cliente-servidor, até o encerramento dos acordos vigentes. No período de falha, é possível notar o mesmo pico de atraso presente no experimento 1. No Streaming POTS, esse pico não ocorre pois há perda de algumas fatias, devido à configuração de janela de slice e *buffer* interno de consumo dos *peers*. Com a configuração utilizada, os *peers* no Streaming POTS não detectam a falha nos *peers* até o esvaziamento de seu *buffer*, quando irão então requisitar o bloco ao servidor. Além disso, o PALMS teve um tempo médio geral menor devido ao mecanismo de troca, enquanto no Streaming POTS a árvore de distribuição era modificada (não necessariamente para melhor) a cada intervalo de acordo.

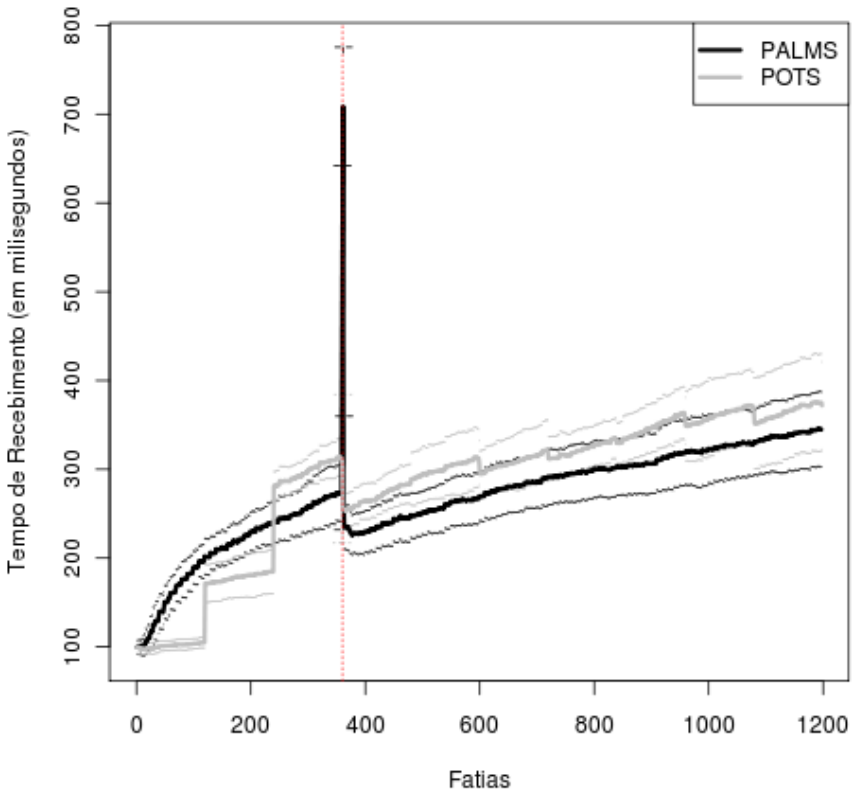

Figura 4.18: Tempo médio de recepção de fatias.

A janela de fatias de tamanho 5 foi suficiente para entregar o conteúdo a todos os requisitantes no PALMS. Inicialmente, foi executada uma simulação com janelas de 5 fatias para o Streaming POTS, mas verificaram-se várias perdas em momentos específicos da simulação (após o período de inserção, por exemplo). A janela foi então modifcada para 20 fatias. Entretanto, um intervalo de consumo menos restritivo (2 segundos nas simulações em [40]) pode criar uma defasagem entre o tempo de recebimento do fluxo (levado em consideração para o tempo médio de recebimento) e o consumo do mesmo. Quando o *peer* verifica que seu *buffer* encontra-se vazio, o bloco/fatia requisitado pode não estar mais disponível no servidor. O aumento na janela de tempo diminuiria o número de perdas, mas aumentaria consideravelmente o tempo de recebimento das fatias.

A figura 4.19 demonstra quantidade total de fatias perdidas devido ao recebimento fora da janela.

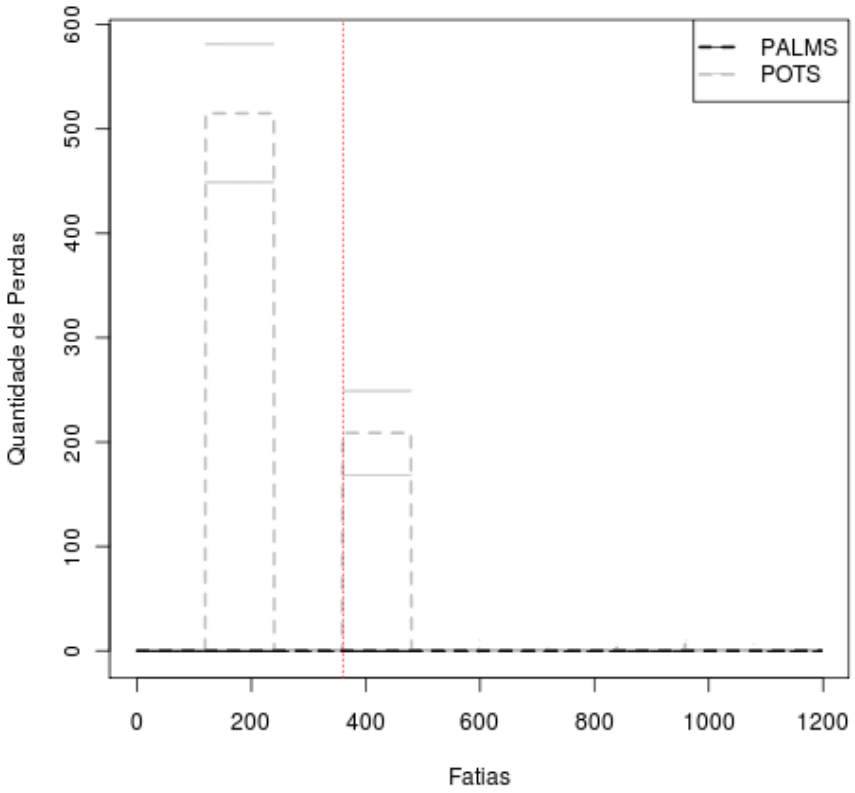

Figura 4.19: Quantidade de fatias recebidas fora da janela de prazo.

A taxa de acerto na obtenção de fatias é bastante sensível aos parâmetros destinados à fatia de realização de acordos espontâneos, notificação de novos acordos e limite de acordo. Quando os acordos espotâneos são realizados próximo da fatia-base do acordo, *peers* que são inseridos na rede após esse tempo certamente não terão acordos espontâneos. Caso um *peer* seja inserido após a notificação de novos acordos, ele receberá todas as fatias do acordo corrente, mas como não houve notificação de um novo acordo, ao término do acordo corrente, o *peer* irá requisitar ao servidor emergencialmente todas as fatias do acordo seguinte, normalizando seu comportamento no acordo posterior (ao receber a notificação de novos acordos). Esse problema seria solucionado atrasando a realização e notificação de acordos para o término do acordo vigente. Entretanto, isso daria aos *peers* menos tempo hábil para a realização de acordos com outros *peers*, resultando em uma maior taxa de requisição ao servidor. Os parâmetros utilizados neste experimento foram os que permitiam a entrada de *peers* o mais tardar possível e ainda assim permitindo aos *peers* a

realização de acordos. As configurações utilizadas tiveram como intuito uma aproximação da realização dos acordos com o tempo de refinamento do PALMS, permitindo um maior intervalo de tempo aos *peers* do Streaming POTS para a negociação de acordos com outros *peers* para não ser necessário a negociação direta com o servidor.

Para atestar a realização de acordos temporários e verificar a utilização do servidor na transmissão de dados, foi realizado um teste sobre a carga de envio de dados imposta no servidor. A figura 4.20 demonstra quantos envios de uma determinada fatia o servidor efetuou. Pode-se notar que durante o período de inserção a carga imposta ao servidor no Streaming POTS foi elevada, enquanto se manteve relativamente estável no PALMS. Pode-se notar também que no período de falha o número de envios no POTS se manteve constante, pois os *peers* não haviam consumido todo o *buffer* recebido.

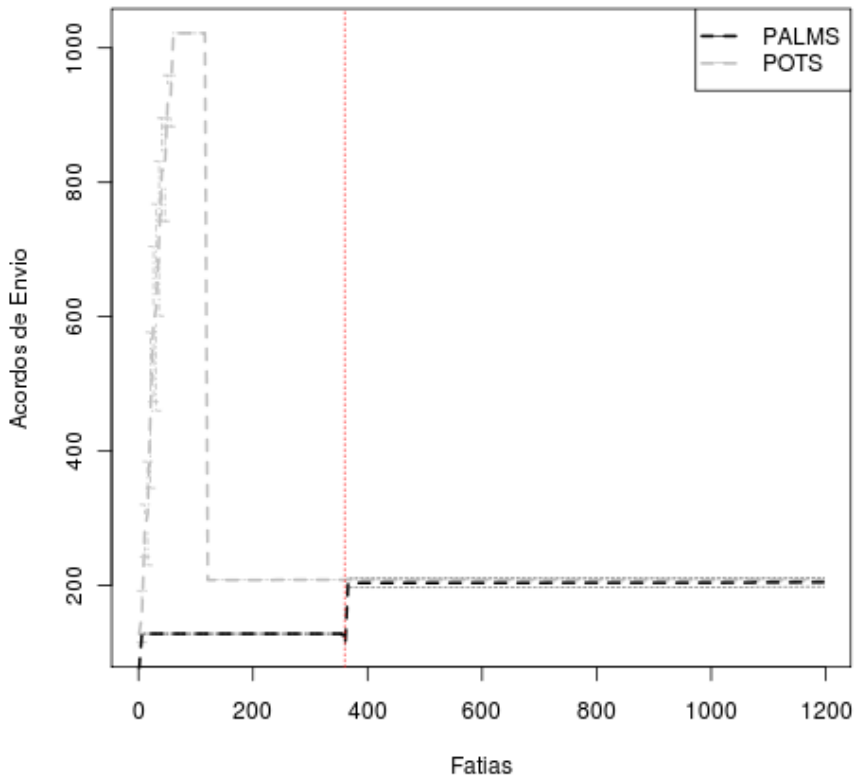

Figura 4.20: Número de envios do servidor por fatia.

Para selecionar os melhores acordos entre os *peers* disponíveis, cada *peer* do Streaming POTS mantém uma lista de outros *peers* presentes na rede, cujos tempos de ir-e-vir

(RTT) são atualizados periodicamente. No PALMS, essa atualização ocorre apenas nos dois conjuntos descritos em 4.2.3. Além das mensagens de atualização, o Streaming POTS possui ainda o *overheah* causado pelas requisições de acordo. O PALMS possui também *overhead* ocasionado pelo mecanismo de troca. A figura 4.21 apresenta a carga de transmissão de mensagens de controle provenientes de todos os tipos de mensagem por período de tempo.

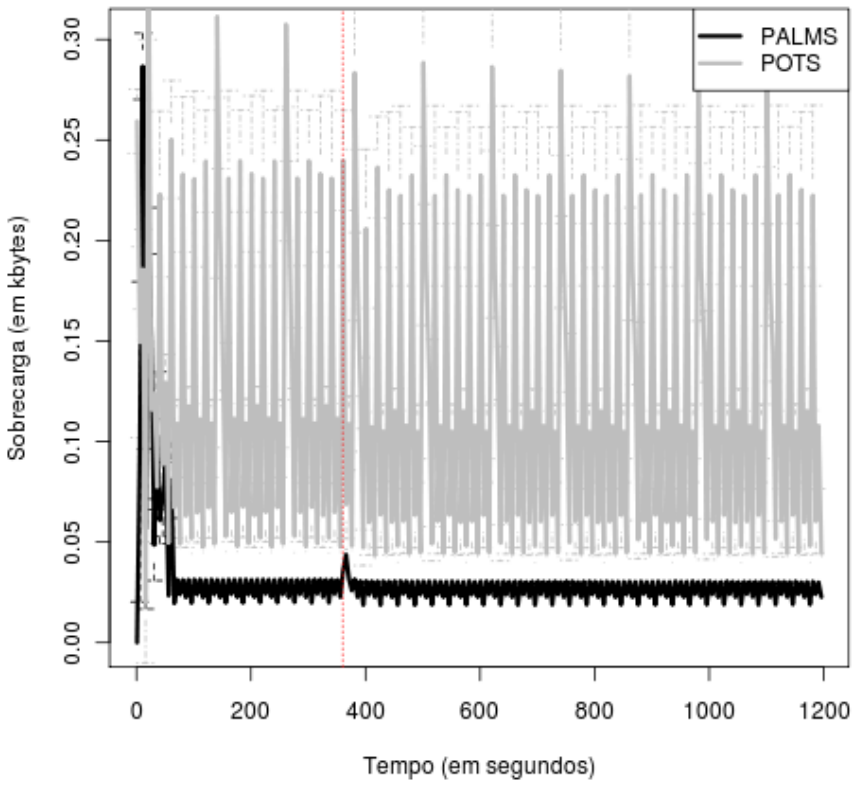

Figura 4.21: Uso de banda no envio de mensagens de controle pelos *peers*.

Apesar de taxas de atualização 10 vezes maiores para o PALMS, há quase o mesmo *overhead* de controle imposto pelo Streaming POTS e PALMS. Apesar disso, a diferença maior no uso de mensagens de controle se dá por parte do servidor. A figura 4.22 mostra o uso de banda em mensagens de controle. Enquanto no PALMS as mensagens se restringem a atualizações dos líderes de grupo (*peers* com os quais o servidor mantém acordos de envio) e respostas a requisições de troca provenientes do mecanismo de desigualdade triangular, no Streaming POTS, além de respostas de mensagens de atualização, o servidor

precisa também enviar mensagens de acordos espontâneos a alguns *peers* e notificações de novos acordos a todos os *peers* presentes na rede.

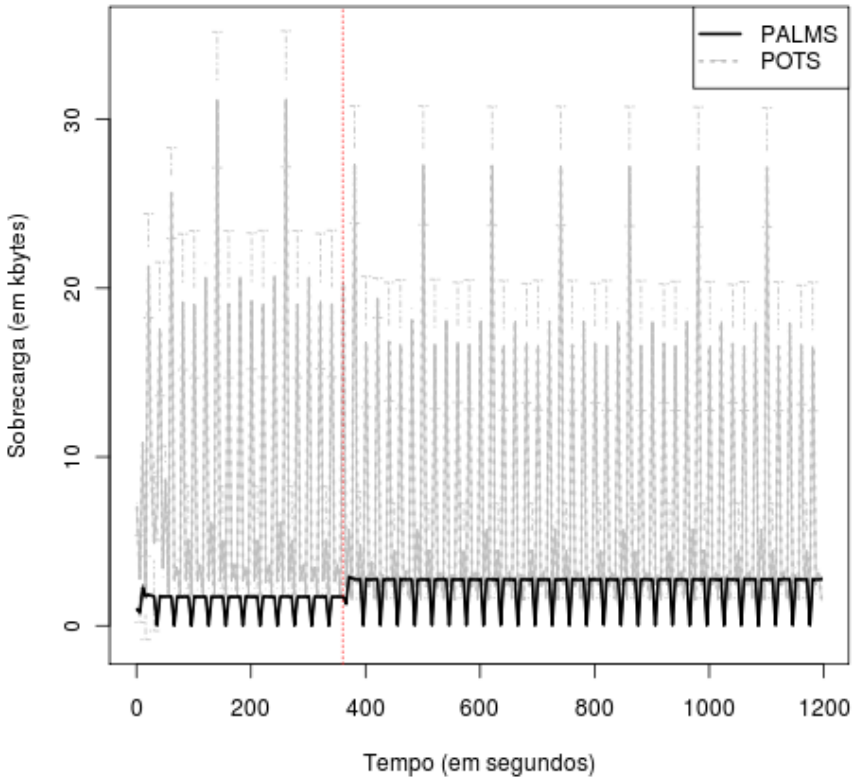

Figura 4.22: Uso de banda em mensagens de controle enviadas pelo servidor.

Este experimento demonstrou que o PALMS pode entregar fatias com latência semelhante e até menor que o sistema descrito em [40]. Além disso, pode-se observar um menor uso do servidor, tanto para mensagens de controle quanto para a propagação dos dados. Pode-se argumentar que o servidor do Streaming POTS desempenha também o papel de *tracker*, mas esta entidade é utilizada apenas na entrada dos *peers* e no caso de falhas no PALMS, não influenciando no uso da banda constante da rede.

Capítulo 5

Conclusão

Neste trabalho foi apresentado um novo protocolo para distribuição de conteúdo contínuo proveniente de uma única fonte utilizando difusão seletiva em nível de aplicação. Foram apresentadas diversas alternativas para a utilização da comunicação *multicast* para distribuição de conteúdo na Internet, incluindo alguns dos principais protocolos ALM, que implementam *multicast* na camada de aplicação.

O protocolo proposto no trabalho, PALMS, utiliza uma rede sobreposta com topologia em árvore. O objetivo foi desenvolver um protocolo simples com baixa sobrecarga em termos de mensagens de controle. O protocolo adota um mecanismo, baseado em desigualdade triangular para reorganizar a topologia de forma a aumentar a eficiência no sistema. O PALMS foi implementado no simulador PeerSim, comparando os comportamentos da versão com e sem o mecanismo de desigualdade triangular e comparando também a versão final do protocolo com o protocolo Narada e o sistema proposto em [40], utilizando como métricas tempo de inserção de *peers*, sobrecarga causada pelas mensagens de controle nos *peers* e no *servidor*, tempo de recebimento do conteúdo na rede simulada e carga imposta ao servidor no envio do conteúdo. Experimentos realizados mostram que o mecanismo de desigualdade triangular necessita de poucas mensagens de controle, mostrando que PALMS consegue atingir um desempenho semelhante aos demais protocolos analisados utilizando menos mensagens de controle

Trabalhos futuros incluem a aplicação de outras métricas no refinamento dos *peers* (como banda de envio e participação na rede), a implantanção de um mecanismo para permitir a entrada de nós por trás de NAT no *tracker* e a realização de experimentos em um ambiente real, utilizando também as métricas dependentes da topologia da rede.

Bibliografia

[1] SEN, S.; WANG, J. "Analyzing Peer-to-peer Traffic Across Large Networks". *IEEE/ACM Transactions on Networking*, Piscataway, NJ, USA, v. 12, n. 2, p. 219–232, 2004.

[2] COMER, D. E. "Interligação de Redes com TCP/IP". 5th. ed. Campus, 2006.

[3] ANDROUTSELLIS-THEOTOKIS, S.; SPINELLIS, D. "A Survey of Peer-to-peer Content Distribution Technologies". *ACM Computing Surveys*, New York, NY, USA, v. 36, n. 4, p. 335–371, 2004.

[4] LI, J. "Peer-to-peer Multimedia Applications". *MULTIMEDIA '06: Proceedings of the 14th annual ACM International Conference on Multimedia*. New York, NY, USA: , 2006. p. 3–6.

[5] DEERING, S. E.; CHERITON, D. R. "Multicast Routing in Datagram Internetworks and Extended LANs". *TOCS: ACM Transactions on Computer Systems*, New York, NY, USA, v. 8, n. 2, p. 85–110, 1990.

[6] HOLBROOK, H.; CAIN, B.; HABERMAN, B. "Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast". (Request for Comments, 4604), RFC 4604, 2006.

[7] ERIKSSON, H. "MBONE: The Multicast Backbone". *Communications of the ACM*, New York, NY, USA, v. 37, n. 8, p. 54–60, 1994.

[8] ANDERSEN, D. et al. "Resilient Overlay Networks". *ACM SIGOPS Operating Systems Review*, New York, NY, USA, v. 35, n. 5, p. 131–145, 2001.

[9] MORAES, I. M. et al. “Distribuição de Vídeo sobre Redes Par-a-Par: Arquiteturas, Mecanismos e Desafios”. *SBRC ’08: Minicursos do Simpósio Brasileiro de Redes de Computadores*. Rio de Janeiro, Rj, Brazil: , 2008. p. 115–171.

[10] GHOSE, D.; KIM, H. J. “Scheduling Video Streams in Video-on-Demand Systems: A Survey”. *Multimedia Tools and Applications*, Hingham, MA, USA, v. 11, n. 2, p. 167–195, 2000.

[11] DAN, A.; SITARAM, D.; SHAHABUDDIN, P. “Scheduling Policies For an On-Demand Video Server with Batching”. *MULTIMEDIA ’94: Proceedings of the second ACM International Conference on Multimedia*. New York, NY, USA: , 1994. p. 15–23.

[12] SHI, W.; GHANDEHARIZADEH, S. “Controlled Buffer Sharing in Continuous Media Servers”. *Multimedia Tools and Applications*, Hingham, MA, USA, v. 23, n. 2, p. 131–159, 2004.

[13] ZHAO, W.; WILLEBEEK-LEMAIR, M.; TIWARI, P. “Efficient Adaptive Media Scaling and Streaming of Layered Multimedia in Heterogeneous Environment”. *ICMCS ’99: Proceedings of the IEEE International Conference on Multimedia Computing and Systems*. Washington, DC, USA: , 1999. p. 377.

[14] HUANG, C.-M.; LIU, P.-C. “Robust Audio Transmission Over Internet with Self-adjusted Buffer Control”. *Information Sciences: Informatics and Computer Science: An International Journal*, New York, NY, USA, v. 124, n. 1-4, p. 1–28, 2000.

[15] LEIBNITZ, K. et al. “Peer-to-peer vs. Client/Server: Reliability and Efficiency of a Content Distribution Service”. *ITC20 ’07: Proceedings of the 20th International Teletraffic Conference on Managing Traffic Performance in Converged Networks*. Berlin, Heidelberg: , 2007. p. 1161–1172.

[16] VENKATRAMANI, C. et al. “Optimal Proxy Management for Multimedia Streaming in Content Distribution Networks”. *NOSSDAV ’02: Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. New York, NY, USA: , 2002. p. 147–154.

[17] LEE, D. L.; ZHAO, D. J.; LUO, Q. “Information Retrieval in a Peer-to-peer Environment”. *InfoScale ’06: Proceedings of the 1st international Conference on Scalable Information Systems*. New York, NY, USA: , 2006. p. 48.

[18] COHEN, B. “Incentives Build Robustness in BitTorrent”. *In Proceedings of the 1st Workshop on Economics of Peer-to-peer Systems*, 2003.

[19] BASHER, N. et al. “A Comparative Analysis of Web and Peer-to-peer Traffic”. *WWW ’08: Proceeding of the 17th International Conference on World Wide Web*. New York, NY, USA: , 2008. p. 287–296.

[20] ROWSTRON, A. I. T.; DRUSCHEL, P. “Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems”. *MIDDLEWARE ’01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*. London, UK: , 2001. p. 329–350.

[21] STOICA, I. et al. “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”. *SIGCOMM ’01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: , 2001. p. 149–160.

[22] XU, Z.; MIN, R.; HU, Y. “Reducing Maintenance Overhead in DHT Based Peer-to-peer Algorithms”. *P2P ’03: Proceedings of the 3rd International Conference on Peer-to-peer Computing*. Washington, DC, USA: , 2003. p. 218.

[23] BOYD, S. et al. “Gossip Algorithms: Design, Analysis and Applications”. *INFOCOM ’05: Proceedings of IEEE INFOCOM 2001*. Miami, FL, USA: , 2005. p. 1653–1664.

[24] GANJAM, A.; ZHANG, H. “Internet multicast video delivery”. *Proceedings of the IEEE*, v. 93, n. 1, p. 159 –170, 2005.

[25] ABLEY, J.; SAVOLA, P.; NEVILLE-NEIL, G. “Deprecation of Type 0 Routing Headers in IPv6”. (Request for Comments, 5095), RFC 5095, 2007.

[26] DEERING, S. E. “Host extensions for IP multicasting”. (Request for Comments, 1112), RFC 1112, 1989.

[27] POSTEL, J. “Internet Protocol - DARPA Internet Program, Protocol Specification”. (Request for Comments, 791), RFC 791, 1981.

[28] M.COTTON; L.VEGODA; D.MEYER. “IANA Guidelines for IPv4 Multicast Address Assignments”. (Request for Comments, 5095), RFC 5771, 2010.

[29] CHU, Y.-h.; RAO, S. G.; ZHANG, H. “A Case for End System Multicast (keynote address)”. *SIGMETRICS ’00: Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. New York, NY, USA: , 2000. p. 1–12.

[30] FRANCIS, P. “Yoid: Extending the Internet Multicast Architecture”. Http://www.aciri.org/yoid, 1999. Acesso em 13 dez. 2009.

[31] BANERJEE, S.; BHATTACHARJEE, B.; KOMMAREDDY, C. “Scalable Application Layer Multicast”. *SIGCOMM ’02: Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. New York, NY, USA: , 2002. p. 205–217.

[32] NAKAMURA, Y. et al. “On Designing End-User Multicast for Multiple Video Sources”. *ICME ’03: Proceedings of the 2003 International Conference on Multimedia and Expo - Volume 3*. Washington, DC, USA: , 2003. p. 497–500.

[33] ZHANG, X.; LI, Z.; WANG, Y. “A Distributed Topology-Aware Overlays Construction Algorithm”. *MG ’08: Proceedings of the 15th ACM Mardi Gras conference*. New York, NY, USA: , 2008. p. 1–6.

[34] LUA, E. K. et al. “Scalable Multicasting with Network-aware Geometric Overlay”. *Computer Communications*, Newton, MA, USA, v. 31, n. 3, p. 464–488, 2008.

[35] XU, Z.; TANG, C.; ZHANG, Z. "Building Topology-Aware Overlays Using Global Soft-State". *ICDCS: Proceedings of the 23rd International Conference on Distributed Computing Systems*, Los Alamitos, CA, USA, v. 0, p. 500, 2003.

[36] ZHANG, X. et al. "An Application Layer Multicast Approach Based on Topology-Aware Clustering". *Computer Communications*, Newton, MA, USA, v. 32, n. 6, p. 1095–1103, 2009.

[37] HOSSEINI, M. et al. "A Survey of Application-Layer Multicast Protocols". *IEEE Communications Surveys & Tutorials, IEEE*, v. 9, n. 3, p. 58–74, 2007.

[38] BELLOVIN, S.; ZININ, A. "Standards Maturity Variance Regarding the TCP MD5 Signature Option (RFC 2385) and the BGP-4 Specification". (Request for Comments, 4278), RFC 4278, 2006.

[39] SRIPANIDKULCHAI, K. et al. "The Feasibility of Supporting Large-scale Live Streaming Applications with Dynamic Application End-points". *SIGCOMM '04: Proceedings of the 2004 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. New York, NY, USA: , 2004. p. 107–120.

[40] MELLO, S. L. V. "Hybrid Client-server Multimedia Streaming Assisted by Unreliable Peers". *The 15th International Conference on Distributed Multimedia Systems*, San Francisco, USA, p. 1–6, 2009.

[41] LUA, E. K.; ZHOU, X. "Network-Aware SuperPeers-Peers Geometric Overlay Network". *ICCCN '97: International Conference on Computer Communication Networks*. Las Vegas, NV , USA: , 2007. p. 141–148.

[42] WACKER, A. et al. "A NAT Traversal Mechanism for Peer-To-Peer Networks". *P2P '08: Proceedings of the 2008 Eighth International Conference on Peer-to-peer Computing*. Washington, DC, USA: , 2008. p. 81–83.

[43] JELASITY, M. et al. "The Peersim Simulator". Http://peersim.sf.net. Acesso em 3 mar. 2010.

[44] WATTS, D. J.; STROGATZ, S. H. “Collective Dynamics of ’Small-world’ Networks”. *Nature*, v. 393, n. 6684, p. 409–10, 1998.

[45] SRIPANIDKULCHAI, K.; MAGGS, B.; ZHANG, H. “An analysis of live streaming workloads on the internet”. *IMC ’04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: , 2004. p. 41–54.

[46] JUN, S.; AHAMAD, M. “Incentives in BitTorrent Induce Free Riding”. *P2PECON ’05: Proceedings of the 2005 ACM SIGCOMM Workshop on Economics of Peer-to-peer Systems*. New York, NY, USA: , 2005. p. 116–121.

[47] HUZIOKA, D.; JR., E. D. “PALMS: Um Protocolo ALM Simples para Distribuição de Conteúdo”. *WP2P SBRC ’2010: IV Workshop de Redes Dinâmicas e Sistemas P2P*, Gramado, RS, BR, p. 1–14, 2010.