Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma Abordagem Colaborativa de Cache em Redes Ad Hoc

Marcos Fagundes Caetano

Monografia apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientador
Prof. Dr. Jacir Luiz Bordim

Brasília
2008

Universidade de Brasília – UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Informática

Coordenador: Prof. Dr. Li Weigang

Banca examinadora composta por:

Prof. Dr. Jacir Luiz Bordim (Orientador) – CIC/UnB
Prof. Dr. Mario Antônio Ribeiro Dantas – INE/UFSC
Prof. Dr. Paulo Roberto de Lira Gondim – ENE/UnB

CIP – Catalogação Internacional na Publicação

Marcos Fagundes Caetano.

Uma Abordagem Colaborativa de Cache em Redes Ad Hoc/ Marcos Fagundes Caetano. Brasília : UnB, 2008.
84 p. : il. ; 29,5 cm.

Tese (Mestre) – Universidade de Brasília, Brasília, 2008.

1. Cache Colaborativo, 2. Cache Cooperativo, 3. Cache, 4. Wireless, 5. Redes Ad-Hoc, 6. MANET

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro – Asa Norte
CEP 70910–900
Brasília – DF – Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma Abordagem Colaborativa de Cache em Redes Ad Hoc

Marcos Fagundes Caetano

Monografia apresentada como requisito parcial
para conclusão do Mestrado em Informática

Prof. Dr. Jacir Luiz Bordim (Orientador)
CIC/UnB

Prof. Dr. Mario Antônio Ribeiro Dantas
INE/UFSC

Prof. Dr. Paulo Roberto de Lira Gondim
ENE/UnB

Prof. Dr. Li Weigang
Coordenador do Mestrado em Informática

Brasília, 01 de agosto de 2008

Dedicatória

Dedico este trabalho à minha amada Gisele.

Agradecimentos

Agradeço ao meu orientador e amigo, prof. Jacir Luiz Bordim, por ter acreditado no meu potencial e ter me ajudado a descobrir o caminho da pesquisa. Agradeço pelas excelentes discussões que tivemos, muitas vezes fervorosas, mas sempre focadas em um objetivo maior.

Agradeço à minha mulher Gisele, por todo apoio, paciência e compreensão. Sua presença em minha vida tem feito todo esforço valer a pena.

Agradeço aos meus pais, Ivani e Cláudio, e às minhas irmãs, Patrícia e Cláudia, pelo carinho e torcida.

Agradeço aos meus sogros, Ana e Mozart, pelas orações e pelo apoio dedicados.

Agradeço à prof^a. Célia Ghedini Ralha, chefe do Departamento de Computação, pelo apoio dado no decorrer do mestrado. Seus conselhos e considerações, sempre pertinentes, ajudaram a melhorar este trabalho.

Gostaria de agradecer a todos que, diretamente ou indiretamente contribuíram para a conclusão deste trabalho.

Resumo

O avanço das tecnologias de rede sem fio permitiu o surgimento de redes *ad-hoc*. A partir de um ambiente não infra-estruturado é possível o estabelecimento de comunicação entre dispositivos espalhados em uma região. Esses dispositivos estabelecem comunicação entre si, de forma dinâmica e em tempo real, criando topologias que permitam o roteamento de pacotes entre os membros da rede. Entretanto, algumas limitações inerentes à tecnologia geram problemas que contribuem para a degradação da vazão na rede. De acordo com Gupta et al. [28], quanto maior é o número de nós em uma rede, menor será a sua vazão. Para esse contexto, o modelo tradicional de *cache* não se apresenta como uma boa opção. A penalidade imposta à rede, após um local *cache miss*, é alta e sobrecarrega tanto os nós intermediários que participam do roteamento, quanto o servidor da rede.

Com objetivo de diminuir essa penalização, diversos trabalhos implementam o conceito de *cache* colaborativo. Essa política consiste em tentar obter a informação, após um *local miss*, a partir dos nós vizinhos mais próximos. Entretanto, seu uso pode ser considerado limitado. As políticas colaborativas de *cache* restringem-se apenas a disponibilizar, aos demais membros da rede, as informações locais armazenada no *cache* de cada cliente. Nenhuma política global para gerenciamento dessas informações é proposta. O objetivo desse trabalho é propor um mecanismo de *cache* colaborativo que permita o compartilhamento de informações, entre nós de uma rede, de forma a diminuir a carga de trabalho tanto no servidor quanto na rede. A partir de uma área de *cache* global, compartilhada entre um grupo de nós, é possível a diminuição do tempo médio de resposta e do número médio de saltos durante o processo de obtenção de dados em uma rede. Para validação da proposta, um modelo foi implementado utilizando o simulador de redes *ad-hoc*, *GloMoSim* [50]. Os resultados experimentais demonstram uma redução de 57.77% no número de requisições submetidas ao servidor para grupos de 8 nós, e 72.95% para grupos de 16 nós. Observou-se uma redução de aproximadamente 16 vezes no tempo médio gasto para responder a uma requisição (*Round Trip Time*).

Palavras-chave: Cache Colaborativo, Cache Cooperativo, Cache, Wireless, Redes Ad-Hoc, MANET

Abstract

The advance of wireless tecnologies has allowed the appearing of ad hoc networks. From a unstructured environment, it is possible to stablish communication among devices. These devices set up communication among themselves, in a dinamic way and in real time, creating topologics that allow the packages flow among the network members. However, some limitations intrinsic to the tecnology generate problems that contribute to the degradation of the network flow. According with Gupta et al. [28], as bigger is the number of nodes in a network, as smaller will be its throughput. The penalty imposed to the network, after a local cache miss, is high and overloads not just the intermediate nodes that participate in the routing, but also the network server.

With the intent of decrease this penalization, several works implement the concept of colaborative cache. This policy consists in trying to get the information from the nearest nodes, after a local miss. Nevertheless, its use can be considered limitated. The colaborative cache policies restrain to give just the local information stored in each client's cache to the other network members. There's no proposition for a global policy to manage such information. The objective of this work is to propose a colaborative cache mechanism that allows the information sharing, among nodes of a network, in a way to decrease the load of work in the server and in the network. From a global chache area, shared by a group of nodes, it's possible to reduce the average response time and the average number of hops during the process of getting data in a network. To validate the proposal, a model was implemented using the GloMoSim [50] ad hoc network simulator. The experimental results show a 57.77% reduction in the number of requests submited to the server for groups of 8 nodes, and a 72,95% reduction for groups of 16 nodes. It was noticed a decrease of 16 times in the average time spent to answer to a request (Round Trip Time).

Keywords: Colaborative Cache, Cooperative Cache, Cache, Wireless, Ad Hoc Networks, MANET

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

ACK	Acknowledges
AODV	Ad Hoc On-Demand Distance Vector
CBR	Constant Bit Rate
CSMA	Carrier Sense Multiple Access
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CTS	Clear To Send
CW	Contention Window
DCF	Distributed Coordination Function
DIFS	DCF Inter Frame Spacing
DSDV	Destination-Sequenced Distance Vector Routing
DSR	Dynamic Source Routing
DSSS	Direct Sequence Spread Spectrum
FTP	File Transfer Protocol
GloMoSim	Global Mobile Information Systems Simulation Library
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IFS	Inter Frame Spacing
IP	Internet Protocol
LFU	Least Frequently Used
LLC	Logical Link Control
LRU	Least Recently Used
MAC	Medium Access Control
MANET	Mobile Ad hoc Networks
NAV	Network Allocation Vector
OLSR	Optimized Link State Routing
PARSEC	Parallel Simulation Environment for Complex Systems
PC	Point Coordinator
PCF	Point Coordination Function
PDA	Personal digital assistants
PHY	Physical Layer
RREP	Route Reply
RREQ	Route Request
RRER	Route Error
RTS	Request To Send
SIFS	Shorter Inter Frame Spacing

SLRU	Segmented Least Recently Used
TCP	Transmission Control Protocol
TELNET	Telecommunication Network
TTL	Time To Live
UDP	User Datagram Protocol

Capítulo 1

Introdução

Nos últimos anos, observa-se o uso cada vez mais difundido de tecnologias de redes sem fio. Sua utilização é possível a partir de dispositivos tais como: *laptops*, *PDA*, celulares, *Desktops*, dentre outros. Em casa ou no trabalho, normalmente, essa tecnologia está associada ao modo de acesso infra-estruturado. Neste modelo, um ponto de acesso realiza o controle do canal e todo dado trafegado, necessariamente, passa por ele.

De acordo com Basagni [5], redes sem fio podem ser categorizadas em dois grupos distintos: redes baseadas em infra-estrutura e redes sem infra-estrutura ou *ad hoc*. A figura 1.1 apresenta uma topologia de rede sem fio baseada em infra-estrutura. De acordo com a figura, é possível observar que alguns nós não estão conectados ao ponto de acesso por não estarem próximos o suficiente do mesmo. Em um modelo de rede não infra-estruturada, não existe a necessidade de infra-estrutura pré-estabelecida para a comunicação. Ou seja, os nós comunicam-se diretamente sem o intermédio de um ponto de acesso. Este tipo de rede, conhecida como rede *ad-hoc*, possui topologia dinâmica e alto grau de mobilidade.

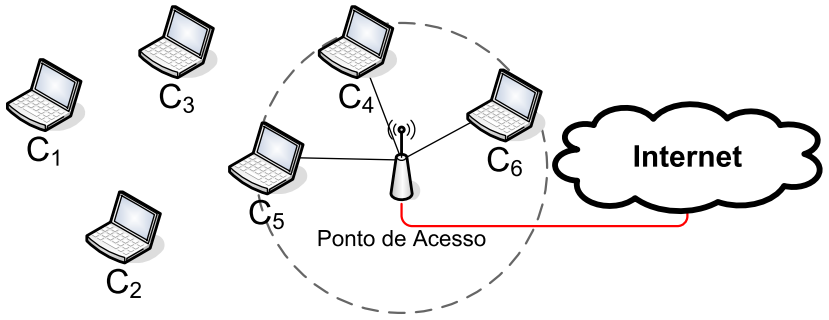

Figura 1.1: Topologia tradicional de rede sem fio para o acesso a *internet*.

.

Historicamente, redes *ad hoc* foram desenvolvidas com a finalidade de emprego militar. As diversas características de um campo de batalha, exigem um modelo de comunicação não infra-estruturado. A partir da década de 90, o desenvolvimento e barateamento da tecnologia proporcionou a sua disseminação na área civil. Hoje, é possível de uma forma simples, o estabelecimento de comunicação descentralizado entre laptops. A comunicação de um salto é definida pelo padrão

802.11 [24] e implementada pelos fabricantes de placa de rede sem fio. Para comunicação em múltiplos saltos, é necessário a utilização de algum protocolo de roteamento para redes *ad hoc*, um exemplo seria o protocolo AODV [41]. A implementação do protocolo AODV, o qual será apresentado no capítulo 2, para ambiente GNU/Linux [26], pode ser obtida no sítio da universidade de Basel [47].

Apresentados os elementos necessários, a topologia apresentada na figura 1.2 é possível de ser implementada. Utilizando o modo de comunicação *ad hoc*, os nós C_1, C_2 e C_3 passam a ter acesso à internet através dos nós C_4 e C_5. Neste contexto, a rede passa a ter uma maior área de cobertura, pois os nós localizados na borda do sinal do ponto de acesso passam a rotear as requisições dos nós mais afastados.

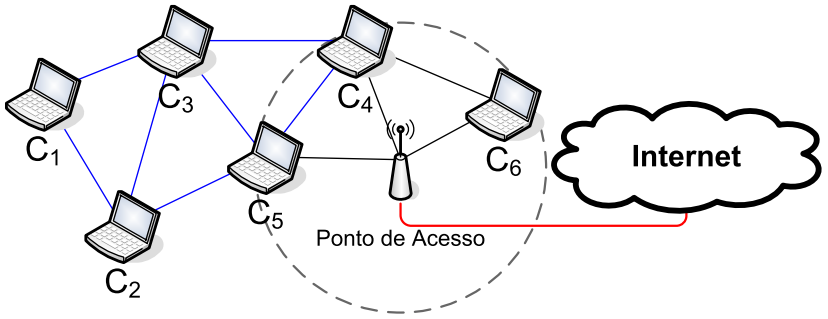

Figura 1.2: Topologia mista de rede sem fio. Modelo tradicional e modelo *Ad Hoc* operando juntos.

A topologia apresentada pode ser implementada em um ambiente real tal como um cibercafé. Neste tipo de ambiente, vários usuários utilizam diversos tipos de dispositivos para ter acesso a *internet*. Entretanto, mesmo que todo ambiente esteja iluminado pelo sinal de rede, outros problemas podem ser identificados em um modelo apenas infra-estruturado. Requisições de nós diferentes para páginas iguais sempre serão submetidas ao ponto de acesso. Caso não exista um servidor proxy[27] instalado logo após o ponto de acesso, o custo dessas requisições será ainda maior. Todavia, o uso da tecnologia de rede *ad hoc* permite que as páginas armazenadas no *cache* dos clientes sejam compartilhadas entre os demais membros da rede, fazendo com que nem toda requisição seja submetida ao ponto de acesso e conseqüentemente diminuindo o seu gargalo. Essa política é conhecida como *cache* cooperativo ou *cache* colaborativo.

A política de *cache* colaborativo não é nova. No passado, foi empregada largamente em redes cabeadas. Conforme apresentado por Dahlin [19], foi utilizada em sistema de arquivos distribuídos com objetivo de diminuição do gargalo e melhoramento de performance. Atualmente, diversos trabalhos abordam esse tipo de política aplicada ao contexto de redes *ad-hoc* [6, 2, 15, 17, 16, 31, 31, 46, 49]. O principal argumento no seu uso, parte da necessidade em se obter a informação desejada em um menor número de saltos possíveis. Conforme demonstrado por Gupta [28], quanto maior o número de nós envolvidos, menor será a vazão do canal. Esta característica está relacionada com os problemas conhecidos como: *problema do terminal exposto* e *problema do terminal escondido*. Os quais serão abordados no capítulo 2.

1.1 Justificativa

As políticas colaborativas de *cache*, apresentadas nos artigos referenciados na revisão bibliográfica, limitam-se apenas a disponibilizar aos demais membros da rede as informações armazenadas no *cache* de cada cliente. Cada nó mantém apenas uma política individual de *cache*, armazenando somente dados de seu interesse. No sistema tradicional de *cache*, somente haverá um ganho no sistema caso um dado requisitado encontre-se armazenado em outro membro da rede. Nesse modelo não existe uma política de cache global com objetivo de explorar o armazenamento de páginas de interesse dos demais membros da rede.

Voltando ao contexto do *cybercafe*, o acesso a páginas Web segue o padrão da distribuição *zipf-like* conforme identificado por Breslau[9]. De acordo com esse trabalho, armazenar 1% do universo de páginas representa responder entre 20% a 35% de todos acessos a esse universo, já armazenar 10% representa entre 45% a 55% dos acessos. Por fim, 70% dos acessos são obtidos com o armazenamento de 25% a 40% das páginas.

Em uma política individual de cache, dependendo do tamanho do universo de páginas, é factível para um cliente armazenar 1% das páginas e responder até 35% das requisições. Entretanto, um maior número de acertos só será possível caso haja mais espaço para o armazenamento de uma quantidade maior de páginas. Essa capacidade pode ser obtida através de um sistema global de *cache*. Supondo que em uma rede com 10 nós, todos com o mesmo tamanho de *cache*, cada nó reserve a metade do seu *cache* para o armazenamento de páginas de interesse dos demais membros. Neste exemplo, a área global de *cache* terá tamanho 5 vezes maior do que o tamanho do *cache* individual de cada cliente. Sendo possível, para este caso, o armazenamento de uma quantidade maior de páginas e conseqüentemente diminuição na quantidade de requisições submetidas ao ponto de acesso.

1.2 Objetivos

Esse trabalho tem como objetivo a proposta de um sistema colaborativo de *cache* para redes *ad-hoc*. Não limitando-se apenas a disponibilização do cache individual de cada cliente, e sim, explorando o conceito de *cache* global para páginas de interesse coletivo. Esta área global de *cache* é formada a partir da reserva de parte do *cache* de cada cliente, que forma a estrutura, e será utiliza para armazenar as páginas de interesse desse grupo. A utilização de todos os membros da rede torna muito onerosa as operações de consulta. Este problema foi abordado em [22] e será melhor discutido no capítulo 3.

Como contribuição secundária, está a disponibilização da implementação de um sistema colaborativo de *cache* para ambientes de rede *ad-hoc*.

1.3 Objetivos Específicos

O presente trabalho apresenta os seguintes objetivos específicos:

- Levantamento do estado da arte no que se refere aos conceitos de *cache* colaborativo;
- Proposta de um modelo cooperativo de *cache*, baseado em um sistema de *cache* global, destinado ao armazenamento e a disponibilização de páginas de interesse coletivo;
- Estudo do ambiente de simulação para redes *ad-hoc* conhecido como *GloMoSim*[50];
- Modelagem para implementação do modelo cooperativo de *cache* proposto;
- Implementação do modelo proposto;
- Estudo e implementação da distribuição *zipf-like* [9] para modelagem do padrão de acesso a páginas Web;
- Implementação do modelo tradicional de *cache* para comparação com o modelo proposto;
- Definição dos cenários para simulação dos modelos implementados;
- Simulação dos modelos e coleta de resultados;
- Análise dos resultados coletados;

1.4 Estrutura do Documento

Esta dissertação está dividida em duas partes. A primeira parte apresenta uma revisão bibliográfica abordando os principais tópicos relacionados à redes *ad-hoc*, sistema de *cache* e sistema de *cache* colaborativo. A segunda parte apresenta o modelo proposto, a simulação e os resultados obtidos, juntamente com uma respectiva avaliação. As duas partes mencionadas estão divididas em 6 capítulos, conforme descritos a seguir:

- O capítulo 2 apresenta os principais conceitos envolvendo redes *ad-hoc* e como o trabalho proposto está relacionado;
- O capítulo 3 apresenta os conceitos envolvendo sistema de *cache* e sistema colaborativo de *cache*. Também é apresentado uma revisão do estado da arte no que se refere aos conceitos de *cache* colaborativo;
- O capítulo 4 apresenta o modelo proposto, sua modelagem e implementação no simulador *GloMoSim*;
- O capítulo 5 apresenta os resultados das simulações e a análise dos valores obtidos;
- Por fim, o capítulo 6 apresenta as considerações finais do trabalho, as conclusões obtidas, limitações e dificuldades encontradas e sugestões para trabalhos futuros.

- No apêndice é apresentado os *scripts* de configuração do simulador *GloMo-Sim*;

Capítulo 2

Redes *Ad Hoc*

Neste capítulo serão introduzidos conceitos envolvendo redes *ad-hoc*, bem como suas características. Será apresentado o padrão IEEE 802.11 e a pilha de protocolos TCP/IP no contexto de rede *ad-hoc*. Por fim, será apresentado o contexto em que essa proposta de trabalho se encaixa.

2.1 Definições

O termo rede sem fio refere-se a um tipo de rede onde a forma de comunicação entre dois ou mais dispositivos é feita por meio de ondas eletromagnéticas [5]. Redes sem fio podem ser classificadas sob diferentes aspectos. Neste trabalho, iremos adotar a classificação de acordo com a formação e arquitetura da rede. Seguindo esse critério, de acordo com Basagni[5], as redes sem fio podem ser classificas em dois grupos distintos:

- **Com infra-estrutura** – Possui como característica básica a necessidade da existência de infra-estrutura pré-estabelecida. Toda a comunicação externa é intermediada através de um ponto de acesso. Apresenta mobilidade restrita;
- **Sem Infra-estrutura ou *ad hoc*** – São redes formadas de maneira dinâmica. Não necessitam de infra-estrutura pré-estabelecida e apresentam como principal característica a mobilidade e autonomia limitada dos seus nós. Cada nó desempenha o papel de *host* e roteador. As informações fluem na rede a partir da cooperação entre os nós;

Diferentemente das redes com fio, as redes *ad-hoc* sem fio apresentam as seguintes características:

- **Interferência do meio** – as ondas de rádio utilizadas para comunicação em redes *ad hoc* são suscetíveis a interferências geradas por outros dispositivos eletrônicos, como microondas. Enquanto que a taxa de erro em fibras óticas é da ordem de 10^{-9}, em redes sem fio a taxa de erro é da ordem de 10^{-4} [38];

- **Recursos de *hardware* limitados** – em alguns casos, baixo poder de processamento, memória, armazenamento e autonomia de bateria;
- **Largura de banda** – apresentam taxas inferiores quando comparadas com redes cabeadas;
- **Topologia** – apresentam auto grau de mobilidade o que acarreta em uma topologias dinâmica. Perdas podem ocorrer devido a mobilidade dos nós;
- **Meio de transmissão** – o canal é compartilhado e a distância máxima de transmissão não pode ser definida com acurácia. Existe uma oscilação causada pelas inúmeras fontes de interferência externa;
- **Vulnerabilidades** – como o sinal de rádio encontra-se espalhado pelo meio, a implementação de mecanismos de segurança eficientes apresentam maior complexidade quando comparados com mecanismos disponíveis para redes cabeadas;

As características destacadas tornam-se ainda mais sensíveis quando a comunicação acontece em múltiplos saltos. De acordo com Gupta[28], a vazão máxima do canal pode ser definida a partir da fórmula: $\theta(\frac{W}{\sqrt{(n \log n)}})$, onde W é taxa de transmissão e n é o número de nós tentando transmitir. Ou seja, para um grupo de 10 nós com uma taxa de transmissão de 2 Mbps, a vazão máxima obtida será de um pouco mais que 600 Kbps. A partir da fórmula apresentada por Gupta, é possível verificar que quanto maior o número de nós, menor será a vazão máxima obtida. Nesse contexto, justifica-se o uso de um sistema de *cache* que permita ao nós da rede, recuperar a informação em um menor número de saltos.

Na subseção 2.3.2.2 serão abordadas as características de implementação de uma rede sem fio que levam a este tipo de comportamento.

2.2 Arquitetura TCP/IP

A pilha de protocolos TCP/IP foi desenvolvida com a finalidade de possibilitar a comunicação entre computadores com arquiteturas distintas e sistemas operacionais diferentes. O modelo TCP/IP foi projetado em quatro camadas, as quais podem ser observadas na figura 2.1 e estão descritas abaixo [20]:

- **Host/Rede** – responsável pelos detalhes de *hardware* e a *interface* física com o meio de comunicação. Como serviços definidos para esta camada estão funções de acesso físico e lógico ao meio físico;
- **Inter-Rede** – esta camada trata do roteamento dos pacotes. Responsável pelo envio dos datagramas de um computador qualquer para outro computador, independente da localização do computador;
- **Transporte** – esta camada é responsável pelo estabelecimento de conexão fim a fim. Destacam-se dois protocolos TCP (*Transmission Control Protocol*) e UDP (*User Datagram Protocol*). O protocolo TCP disponibiliza a

camada de aplicação um fluxo de dados confiável entre nó origem e nó destino. O protocolo UDP não possui esta garantia, transferindo para camada de aplicação essa responsabilidade;

- **Aplicação** – camada de mais alto nível na pilha TCP/IP, é responsável por tratar os detalhes da aplicação em si. São exemplo de protocolos da camada de aplicação: FTP (*file transfer Protocol*), HTTP (*Hypertext Transfer Protocol*), dentre outros.

A figura 2.1 apresenta a pilha de protocolos TCP/IP, descrita acima, e a sua disposição quando mapeada em rede *ad-hoc*. A camada *Host/Rede* é subdividia em camada física e camada *MAC*, ambas descritas pelo padrão IEEE 802.11[32]. Na camada *Inter-Rede*, destacam-se os protocolos de roteamento *AODV*[41] (*Ad-hoc On-Demand Distance Vector Routing*) e *DSR*[33] (*Dynamic Source Routing Protocol*), os quais serão examinados na seção 2.4.

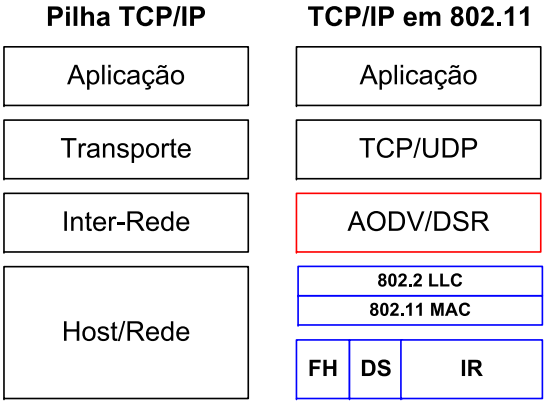

Figura 2.1: Adaptações na pilha TCP/IP quando aplicada na arquitetura 802.11.

Este trabalho em questão concentra-se na camada de aplicação. O sistema de cache utiliza a *interface* disponibilizada pela camada de transporte para o envio de requisições e mensagens de controle aos demais nós da rede. Para geração de algumas estatísticas, tais como número médio de saltos por requisição, foi necessário a obtenção de informações a partir da camada de rede. Essa é um técnica conhecida como *Cross-Layer*[44]. Informações mais detalhadas a respeito da modelagem e implementação do modelo proposto serão apresentadas no capítulo 4.

2.3 Padrão IEEE 802.11

Atualmente, o padrão *IEEE 802.11* é o padrão implementado pela maioria das redes *ad-hoc*. Sendo este um documento muito extenso, apenas os conceitos relevantes serão abordados nesta dissertação. Maiores informações podem ser obtidas a partir do documento original [32].

Os protocolos IEEE 802.11 definem a primeira camada no modelo TCP/IP, conforme correspondência apresentada pela figura 2.1. A camada de *enlace* é

dividida em subcamada MAC (*Media Access Control*) e subcamada LLC (*Logical Link Control*). A subcamada LLC é responsável pelo controle de erros e pelo controle de fluxo. Enquanto a subcamada MAC é responsável pelo endereçamento, divisão dos dados em *frames*, e o controle de acesso ao meio. Além dessas funcionalidades, no padrão 802.11 a camada MAC também é responsável por funções tipicamente desempenhadas por protocolos da cama acima como fragmentação, retransmissão de pacotes e *acknowledges* (ACK) [32]. A camada física, por sua vez, é dividida em duas partes menores: subcamada dependente do meio físico (*Physical Medium Dependent Layer*) e o protocolo de convergência da camada física (*Physical Layer Convergence Protocol*). A primeira é responsável por realizar funções de modularização, codificação e decodificação dos sinais que serão transmitidos pelo meio físico. A segunda camada é responsável por fornecer uma interface entre a camada MAC e a camada física logo abaixo.

Conforme definido pelo padrão, na camada de *enlace* o protocolo *IEEE 802.11 MAC* é o responsável pelo controle de acesso ao meio. De uma maneira geral, os protocolos MAC podem ser divididos em dois grupos: **determinísticos** e **aleatórios**. Nos protocolos MAC de acesso aleatório, o acesso ao meio físico ocorre sem agendamento prévio entre os clientes. Já os determinísticos necessitam de acordo prévio, podendo ser feito de forma centralizada ou descentralizada. No padrão 802.11, o protocolo MAC pode operar em dois modos distintos: modo determinístico PCF (*Point Coordination Function*) e modo aleatório DCF (*Distributed Coordination Function*).

- No modo de operação PCF (*Point Coordiantion Function*), a rede trabalha em modo infra-estruturado e o controle de acesso ao canal é feito através do ponto de acesso. Cada cliente está associado ao ponto de acesso, e este desempenha o papel de *Point Coordination* (PC). O acesso ao canal é feito em rodas definidas pelo PC. A cada rodada, o ponto de acesso questiona cada cliente a respeito do conteúdo a ser transmitido. Como o ponto de acesso tem maior prioridade de acesso ao meio, ele consegue monopolizar o canal e gerenciar a ordem de quem deve transmitir.

- O modo DCF (*Distributed Coordination Function*) habilita a rede a trabalhar de forma não infra-estruturada, permitindo aos nós conversarem diretamente entre si. Este comportamento gera colisões devido há disputa entre vários clientes por um mesmo canal. Para tentar resolver este problema, vários protocolos de acesso ao meio foram propostos [48]. Dentre eles, destacam-se: ALOHA, CSMA, Non-Persistent CSMA, P-Persistent CSMA, 1-Persistent CSMA, CMSA/CA.

2.3.1 CSMA

O protocolo CSMA (*Carrier Sense Multiple Access*) apresenta uma evolução significativa quando comparado ao protocolo ALOHA. Walke [48], nos mostra que o protocolo CSMA consegue 80% de aproveitamento da vazão total do canal. Essa evolução, partiu da capacidade dos nós conseguirem identificar se o canal está livre ou ocupado. Naturalmente, se o canal estiver ocupado o nó deve esperar este ficar livre. Caso o canal esteja livre, o nó pode iniciar sua transmissão.

Esta abordagem permite que mais de um nó verifique o estado livre do canal e tente transmitir. Neste caso, colisões ainda irão correr. Para poder resolver esse problema, surgiu a necessidade da implantação de uma política coordenada de acesso ao canal. Diversos trabalhos foram publicados nesse sentido, algumas variações do protocolo CSMA serão apresentadas na seqüencia.

2.3.2 CSMA/CA

O protocolo de acesso ao meio implementado pelo padrão 802.11 é o CSMA/CA (*Carrier Sense Multiple Access With Collision Avoidance*). Este protocolo apresenta as mesmas características do protocolo CSMA com o recurso de *Collision Avoidance*. Quando dois nós verificam simultaneamente que o canal está livre, eles não iniciam imediatamente a transmissão do dado. Cada um irá esperar um tempo mínimo, representado por DIFS (*Distributed Inter-Frame Space*), para executar o algoritmo exponencial de *backoff*. No padrão 802.11a o tempo DIFS é de $34\mu s$ [48].

2.3.2.1 Algoritmo Exponencial de *Backoff*

O algoritmo de *backoff* é utilizado para diminuir a probabilidade de colisões entre nós que desejam iniciar uma transmissão. Ele consiste em um sorteio aleatório da quantidade de tempo que um nó deve esperar, antes de voltar a tentar transmitir. A faixa de valores possíveis de serem sorteadas é incrementada a medida que colisões voltam a ocorrer e retransmissões são necessárias.

$$BackoffTime = Random() * SlotTime \tag{2.1}$$

Onde,

- Random() – é um inteiro pseudorandômico distribuído no intervalo entre $[0, CW]$ e $CWMin \leqslant CW \leqslant CWMax$;
- SlotTime – *slot* de tempo definido pelas características físicas;

A janela de contenção (CW: *Contention Window*) é incrementada a cada nova retransmissão do dado. Ela inicia com o valor de $CWMIN = 31$ e a cada nova retransmissão seu valor é dobrado até o valor máximo de $CWMAN = 1023$.

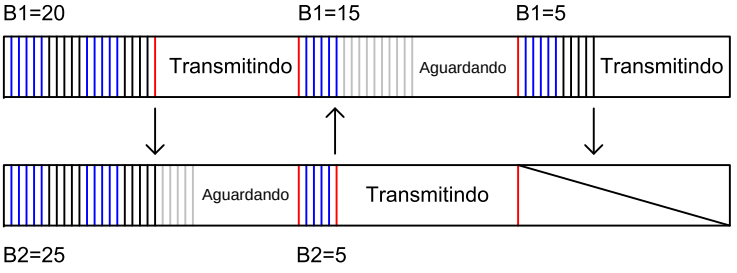

Figura 2.2: Um exemplo do funcionamento do Algoritmo Exponencial de *Backoff*.

No exemplo apresentado na figura 2.2, é possível verificar o funcionamento do algoritmo de *backoff*. Inicialmente, dois nós executam o algoritmo de *backoff* com valor de $CW = 31$. O primeiro nó obteve o valor 20 e o segundo o valor 25. A cada *slot* de tempo os nós irão decrementar o seu valor de *backoff*. O primeiro nó irá chegar primeiro em zero e iniciará a transmissão. Nesse momento, o segundo nó irá interromper o decremento do valor de *backoff* até o canal ficar novamente livre. Ao terminar de transmitir, o primeiro nó irá sortear novamente um valor de *backoff*. Quando o canal fica livre, ambos os nós voltam a decrementar o valor de *backoff*. O segundo nó irá chegar primeiro a zero e iniciará a transmissão, enquanto o primeiro nó irá aguardar o canal ficar livre para continuar a decrementar seu valor novo de *backoff*.

O algoritmo exponencial de *backoff* é utilizado pelo padrão 802.11. Os nós executam esse algoritmo toda vez que [48]:

- quando os nós verificam que o canal está ocupado antes da primeira transmissão;
- antes de cada transmissão;
- depois de uma transmissão bem sucedida;

2.3.2.2 RTS e CTS

As características do rádio *half-duplex* utilizado na comunicação entre nós, acarreta nos problemas conhecidos como: Problema do Terminal Exposto e Problema do Terminal Escondido. Ambos os problemas acarretam na diminuição da vazão da rede. Entretanto, apenas o Problema do Terminal Escondido é possível de ser resolvido utilizando este tipo de tecnologia.

- No Problema do Terminal Exposto apresentado na figura 2.3, a área de transmissão dos nós C_2 e C_3 estão delimitadas pelos círculos menores. Os círculos maiores representam, respectivamente, as áreas de interferência geradas quando os nós C_2 e C_3 transmitem [36]. Ou seja, para o nó C_2 o círculo maior em azul representa o ruído detectado no canal quando o nó C_3 transmite para o nó C_4. Esse sinal recebido por C_2 não é suficiente para ele conseguir identificar o conteúdo dos dados que estão sendo transmitidos, mas é suficiente para faze-lo decidir não utilizar o canal para transmissão. No entanto de acordo com essa topologia, C_2 e C_3 podem transmitir simultaneamente sem perigo de colisão pois suas áreas de transmissão são independentes.
- O problema do Terminal Escondido apresentado na figura 2.4 ocorre quando dois nós não conseguem detectar a presença um do outro, gerando assim colisão. Neste exemplo, C_1 não recebeu à solicitação de transmissão de C_3 para C_2. C_3 também não recebeu a solicitação de C_1 para C_2. Tanto C_1 quanto C_3 irão transmitir para C_2 e este não será capaz de receber nenhum dos dados. Este problema ocorre porque ambos transmissores não compartilham a mesma área de transmissão.

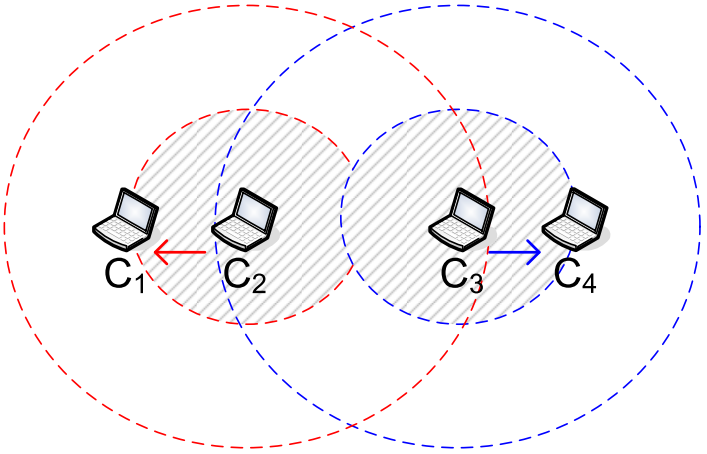

Figura 2.3: Problema do Terminal Exposto (*Exposed Terminal Problem*).

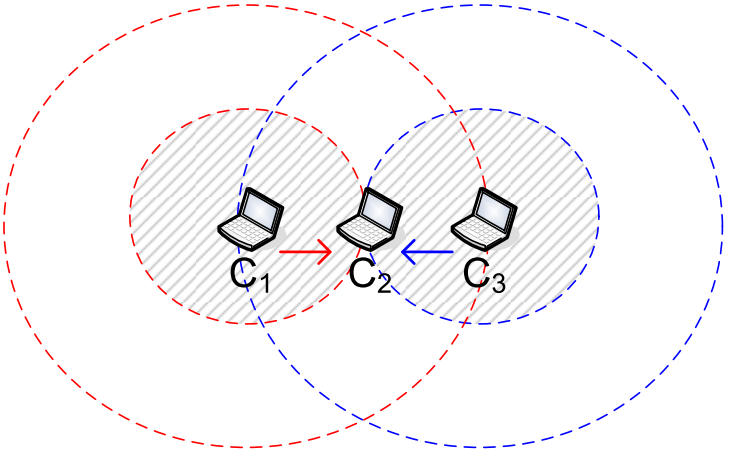

Figura 2.4: Problema do Terminal Escondido (*Hidden Terminal Problem*).

Para resolver o problema de redução de vazão causado pelo problema do terminal escondido, o padrão 802.11 (CSMA/CA) implementa o conceito de alocação de espaço aéreo. Seu uso é opcional e consiste no envio de pacotes RTS (*Request to Send*) e CTS (*Clear to Send*). De acordo com o exemplo apresentado na figura 2.5 (WALKE, [48]), o primeiro passo para os nós que desejam disputar o canal é executar o algoritmo exponencial de *backoff*. Após um tempo igual a DIFS, todos os nós começam a decrementar o valor de *backoff* obtido. Como o nó 2 foi o primeiro a chegar o seu valor em zero, ele irá transmitir um pacote RTS para o nó 1. Após um tempo igual a SIFS (*Short Interframe Space*) o nó 1 irá enviar como resposta um pacote CTS. Note que a seqüencia RTS/CTS/DADO/ACK é separada por um espaço de tempo com duração SFIS. O espaço SFIS é bem menor que o DIFS, o que coloca a seqüencia de pacotes CTS/DADO/ACK com maior prioridade de acesso ao canal do que o início da transmissão RTS.

Dentre os campos contidos nos pacotes RTS e CTS, é possível calcular o tempo total em que a seqüencia RTS/CTS/DADO/ACK irá durar. Com base nessa informação, os demais nós podem ajustar o seu valor de NAV (*Network Allocator Vector*). O valor de NAV estipula a quantidade de tempo que o nó não

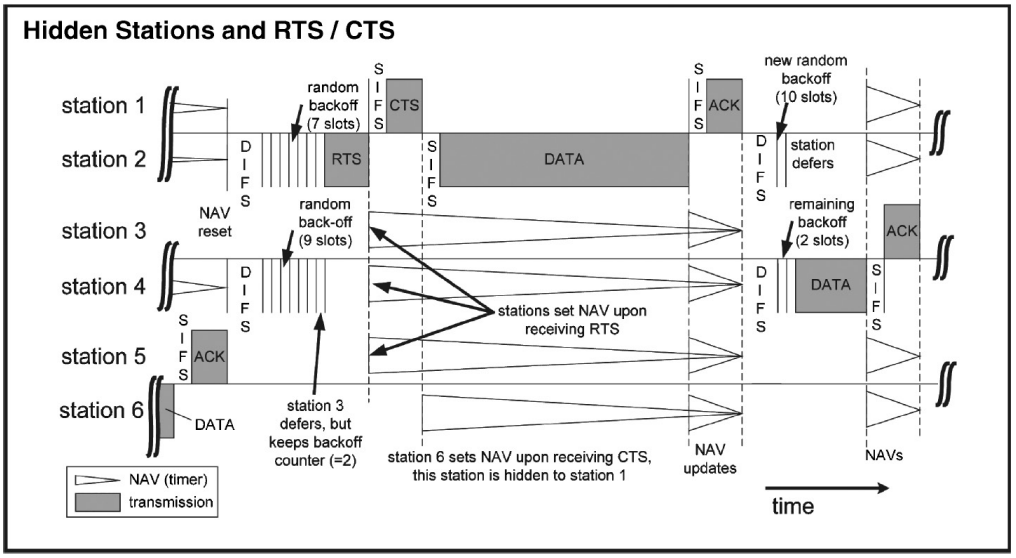

Figura 2.5: Resolução do Problema do Terminal Escondido utilizando pacotes RTS e CTS (WALKE, [48]).

precisa acessar o canal. Após o envio do pacote de ACK pelo nó 1, todos os nós irão continuar a decrementar os seus respectivos valores *backoff*. Nesse ponto o protocolo continua normalmente. É interessante ressaltar que de acordo com o exemplo, o nó 6 não conseguiu receber o pacote RTS por estar muito distante. Entretanto, ele conseguiu ajustar o valor do seu NAV através do pacote CTS enviado pelo nó 2.

2.4 Protocolos de Roteamento

O roteamento eficiente de pacotes em uma rede *ad-hoc* não é uma tarefa trivial. O Padrão IEEE 802.11 define o acesso em modo não infra-estruturado apenas para um salto. O roteamento em múltiplos saltos é possível mediante a utilização de protocolo de roteamento específico da camada de rede na pilha TCP/IP (figura 2.1). Um estudo mais aprofundado de vários protocolos para roteamento pode ser encontrado em [5]. Sua natureza dinâmica torna complexa o gerenciamento das rotas. Uma rota recém criada pode deixar de ser válida em pouco tempo. Alguns problemas relacionados são:

- movimentação de alguns nós;
- degradação do sinal gerada por interferência;
- desligamento do radio para economia de bateria;

O uso de protocolos de roteamento permite o envio de requisições a nós situados a vários saltos de distância. Quanto a criação e manutenção de rotas, os protocolos de roteamento podem ser divididos em [43]: *Table Driven* (pró-ativos) e *On-demand* ou (Reativos).

2.4.1 Protocolos Reativos

Os protocolos de roteamento reativos estabelecem e gerenciam as rotas conforme surge a necessidade. Durante o processo de estabelecimento de comunicação entre dois nós as rotas são criadas, a medida que as rotas são utilizadas sua validade é atualizada. Como o processo de criação e gerenciamento de rotas é feito sob demanda, um *delay* está associado a comunicação e o uso de rotas inválidas pode ocorrer. Ao contrário dos protocolos pró-ativos, estes apresentam bom desempenho em redes grandes e redes com alta mobilidade, já que não há necessidade de manter uma tabela contendo rotas para cada nós da rede.

Como exemplo de protocolos reativos tempos: *AODV* (*Ad-Hoc On-Demand Distance Vector Routing*) [41] e o *DSR* (*Dynamic Source Routing*) [33].

2.4.1.1 AODV

O protocolo de roteamento AODV (*Ad-Hoc On-Demand Distance Vector Routing*) trabalha em um escopo local da rede, ou seja, os nós não tem uma visão completa da rede e das rotas que estão gerenciando. Com isso, o impacto de manutenção de rotas é menor, pois a troca de tabelas não é feita com todos os nós da rede [43].

Cada nó possui uma tabela de roteamento onde é armazenado informações de roteamento dos nós vizinhos que estejam ao seu alcance (*single-hop*). Existe um tempo máximo em que uma determinada rota é considerada válida. Caso a rota não seja utilizada antes do término desse período, a rota é considerada inválida e excluída. Cada vez que a rota é utilizada, dentro do prazo de validade corrente, a sua marcação de tempo é renovada.

A figura 2.6 apresenta o funcionamento do protocolo AODV. Supondo que o nó B deseja estabelecer comunicação com o nó E. Caso ele não conheça uma rota válida até o nó E, ele irá fazer *broadcast* de uma requisição de rota (*RReq - Route Request*) a seus vizinhos, os seus vizinhos irão repassar para os vizinhos deles e assim por diante. A medida que o pacote de requisição é repassado aos demais membros da rede, a cada salto, um contador no pacote é incrementado. Esse processo irá terminar quando for alcançado o nó destino, ou um nó que conheça uma rota válida até o destino. A requisição contendo o menor valor do contador que chegar no nó E será utilizada para o envio da resposta de rota (*RREP - request replay*).

Como este é um algoritmo com visão local da rede, o nó B saberá que quando ele quiser falar com o nó E ele precisa utilizar a rota através do seu vizinho C. Para evitar loops, cada rota é marcada com um número seqüencial que é incrementado a cada nova tentativa de descobrimento de rota. Requisições de rota para o mesmo destino que tenham número seqüencial menor que o armazenado são descartadas. Uma mensagem RRER (*Route Error*) será enviada ao nó remetente quando um nó intermediário não conseguir encontrar o próximo nó indicado na tabela de rota.

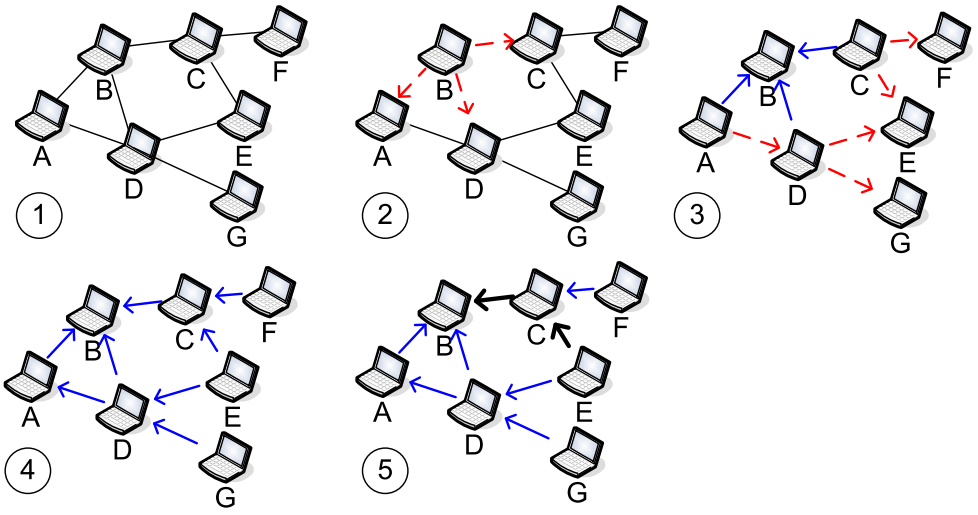

Figura 2.6: Protocolo de roteamento reativo *AODV* (*Ad-hoc On-Demand Distance Vector Routing*). De 2 a 4 representa o *broadcast* de requisição de rota feita pelo nó *B* para o nó *E*. Em 5 é representado o envio da resposta de rota pelo nó *E*.

2.4.1.2 DSR

O protocolo DSR (*Dynamic Source Routing*) é bem semelhando ao protocolo AODV. Dentre as diferenças significativas, pode-se destacar a marcação referente a rota completa em um pacote RREQ. O que permite ao nós destinatário saber exatamente todos os nós intermediários por onde o pacote passou. O protocolo DSR realiza cache de rotas, inclusive mais de uma rota para um mesmo destino. O que permite o uso de rotas alternativas em caso de rotas desativadas. Ao contrário do AODV, rotas inativas não são excluídas. Uma comparação mais aprofunda entre esse dois protocolos reativos pode ser obtida em [8].

2.4.2 Protocolos Pró-ativos

Os protocolos de roteamento pró-ativos estabelecem e gerenciam as rotas de forma automática. Sua principal característica é a existência de uma tabela com as rotas para todos os nós alcançáveis da rede. Com este comportamento, o tempo para o estabelecimento de comunicação entre dois nós não sofre *delay* gerado pelo processo de criação ou uso de rotas inválidas. Em contra partida, caso a rede apresente um comportamento muito dinâmico, o custo para manutenção das rotas de maneira pró-ativa pode ser muito elevado, faz-se necessária a troca constante de tabelas de roteamento entre os nós. Este tipo de protocolo também não apresenta um bom desempenho para redes muito grandes, já que será necessária a reserva de uma quantidade grande de memória para o armazenamento das rotas grandes.

Como exemplo de protocolos pró-ativo temos: OLSR (*Optimized Link State Routing Protocol*) [1] e DSDV (*Destination-Sequenced Distance Vector Routing*) [40];

Neste capítulo foram introduzidos alguns conceitos e características envolvendo redes móveis sem fio *ad-hoc*. Foram apresentados os protocolos para pilha

TCP/IP no ambiente *ad-hoc*. Utilizando a arquitetura da pilha TCP/IP, foi apresentado o padrão IEEE 802.11 o qual define o funcionamento para camada *Host/Rede*. Foram apresentados o problema do terminal exposto e o problema do terminal escondido, e como o padrão IEEE 802.11 resolve este último através da alocação de espaço aéreo. Também foram apresentados dois protocolos reativos de roteamento para multi-salto. No próximo capítulo serão introduzidos os conceitos envolvendo sistema de *cache* individual e colaborativo, bem como, a distribuição zipf-like. Também será apresentada uma revisão atual do estado da arte no que se refere a sistemas colaborativos de cache.

Capítulo 3

Sistema de *Cache*

O uso de políticas de *cache* para melhorar o gerenciamento dos recursos computacionais está bastante difundido. Dentre as vantagens obtidas com sua utilização, é possível citar [21]:

- redução no uso da largura de banda;
- redução na carga de trabalho dos servidores;
- diminuição do tempo de resposta médio.

Em ambientes de rede sem fio, o emprego de políticas de *cache* é fundamental. Além dos benefícios citados, a utilização de um sistema de *cache* eficiente permite um melhor aproveitamento no uso de bateria dos dispositivos móveis [37]. Na pilha TCP/IP apresentada na figura 2.1, o sistema de *cache* é implementado na camada de aplicação. Para o sistema de *cache*, as camadas de baixo irão disponibilizar um interface transparente de serviços para permitir a comunicação entre cliente e servidor. O objetivo desse capítulo é fornecer um embasamento teórico sobre as políticas de *cache* existentes. Baseado no ambiente de rede *ad-hoc*, abordar as questões envolvendo a recuperação e manutenção de dados. Por fim, apresentar como se dá o funcionamento do sistema de *cache* em uma rede *ad-hoc*.

3.1 Sistema de *Cache* Individual

O conceito de *cache*, utilizado neste trabalho, é definido como uma área local e individual destinada ao armazenamento de informações de interesse do dispositivo. As informações armazenadas em *cache* são obtidas a partir de um servidor remoto. Recuperar um dado armazenado localmente é mais rápido e possui um custo menor do que um dado localizado remotamente.

Quando um dado é recuperado a partir do seu *cache* local, dizemos que ocorreu um *cache-hit*. Quando o dado não encontra-se armazenado localmente, dizemos que ocorreu um *cache-miss*. Um *cache-miss* acarreta no uso do sistema de comunicação para o envio de requisições ao servidor. Além do *delay* associado a recuperação do dado, temos o envolvimento de outros dispositivos para fazer o

roteamento da informação. Em redes *ad-hoc* o custo gerado por um *cache-miss* reflete não apenas no dispositivo que fez a requisição, mas em todos os nós envolvidos com o roteamento das informações até o servidor.

Como os recursos dos dispositivos móveis são limitados, existe a necessidade do gerenciamento do espaço de *cache*. Uma política de substituição (*Replacement Algorithm*) é acionada toda vez que o *cache* está cheio e um novo dado precisa ser armazenado. A escolha de uma *vítima* a ser substituída não é uma tarefa trivial. Conforme apresentado por Podlipning at el. [42], os seguintes fatores são importante e devem ser observados:

- o tempo transcorrido desde a última vez que o dado foi referenciado;
- a quantidade de vezes que o dado foi referenciado;
- o tamanho do dado;
- o custo para recuperar o dado a partir do servidor;
- o tempo transcorrido desde a última atualização do dado;
- o tempo empírico previsto para que o dado continue válido.

Baseado nas características descritas, diversos algoritmos de substituição foram propostos [42]. Dentre eles, podemos citar: LRU (*Least Recently Used*), Size, LFU (*Least Frequently Used*) [3], LFU-Aging [3], SLRU (*Segmented Least Recently Used*) [34] e RAND.

A quantidade de algoritmos de substituição apresentados na literatura é muito extensa [42]. Entretanto, no contexto de *cache* colaborativo aplicado a redes *ad-hoc*, a grande maioria dos trabalhos utilizam apenas a polícia de substituição *LRU*. O foco desses trabalhos concentram-se em maneiras eficientes de distribuir e acessar os dados, de formar a gerar o menor impacto na rede. Já trabalhos específicos em algoritmos de substituição, como publicado por LI [37], exploram e propõem políticas de substituição que levem em conta as características físicas das redes sem fio. Neste trabalho, como política de substituição individual de *cache* iremos implementar o algoritmo *LRU*. O algoritmo LRU além de apresentar bons resultados, é utilizado pelos trabalhos relacionados na revisão bibliográfica.

3.1.1 Algoritmo *LRU*

O algoritmo LRU (*Least Recently Used*) é um dos algoritmos de substituição mais utilizados [42]. Baseia-se no conceito de localidade temporal. Ou seja, um dado recentemente acessado tem maiores chances de ser referenciado em um futuro próximo do que um dado mais antigo. Com isso, ele mantém em *cache* os dados mais recentes e substitui os mais antigos.

A figura 3.1 apresenta um exemplo do funcionamento do algoritmo *LRU*. Para um *cache* de tamanho dois, dada a seguinte seqüência de consultas: 1, 2, 2 e 3. Observa-se que em *C* o dado 2 é recuperado a partir do *cache* local. Em *D* o dado *3*, recuperado remotamente, precisa ser armazenado em *cache*. A política *LRU* irá selecionar o dado *1* como *vítima* a ser substituída, pois esse é o dado menos referenciado recentemente.

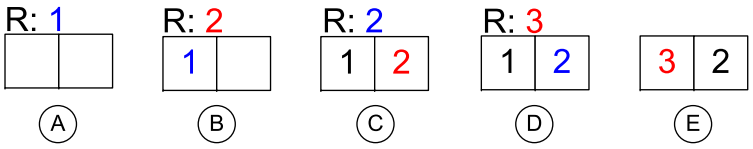

Figura 3.1: Algoritmo de substituição *LRU* aplicado em uma *cache* de tamanho dois a partir da seqüência de requisições: 1, 2, 2 e 3.

3.1.2 Algoritmo LFU

O algoritmo LFU (*Least Frequently Used*) [3] associa a cada dado armazenado em *cache* um contador. O contador é incrementado a medida que os dados são referenciados. O Algoritmo de substituição LFU escolhe sua *vítima* baseado no dado com menor freqüência obtida.

A figura 3.2 apresenta um exemplo do funcionamento do algoritmo *LFU*. Em *D*, o dado *1* foi escolhido como vítima por possuir o menor valor de contador.

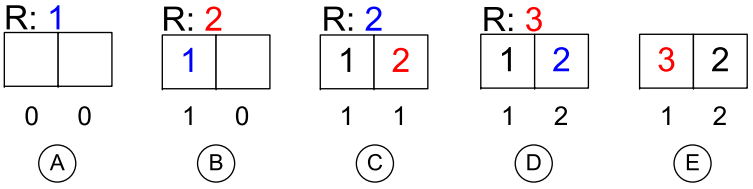

Figura 3.2: Algoritmo de substituição *LFU* aplicado em uma *cache* de tamanho dois a partir da seqüência de requisições: 1, 2, 2 e 3.

3.1.3 Algoritmo LFU-Aging

O algoritmo LFU-Aging [3] foi proposto como um melhoramento para o algoritmo LFU. No algoritmo original, os dados que foram muito populares por um período de tempo no passado, mesmo não sendo mais requisitados, podem permanecer em *cache* por um longo período de tempo. Para evitar este tipo de poluição do *cache*, a abordagem LFU-Aging utiliza um mecanismo de envelhecimento. É estipulado um valor de *threshold* para os contadores. Quando a média dos contadores ultrapassar o valor de *threshold*, todos os contadores acima do limite são divididos por dois.

3.1.4 Algoritmo SLRU

O algoritmo SLRU (*Segmented Least Recently Used*) [34] divide a área para armazenamento dos dados em duas partes: segmento protegido (*protected segment*) e segmento não protegido (*unprotected segment*). A idéia básica do algoritmo é reservar uma área em disco para armazenamento dos objetos mais populares.

A primeira vez que um dado é referenciado, ele será armazenado no segmento não protegido no *cache*. Quando ocorre um *cache-hit*, o dado é movimentado para

área protegida de *cache*. Ambos os segmentos são gerenciados pela política LRU. Entretanto, os dados armazenados na área protegida não podem ser removidos do *cache*. Quando o segmento protegido está cheio e um novo dado precisa ser armazenado, o dado menos recentemente referenciado é movido para área de *cache* não protegida.

3.1.5 Algoritmo RAND

O algoritmo de substituição RAND (*Random*) escolhe suas vítimas de forma aleatória. Vários algoritmos aleatórios foram propostos, variando apenas a forma como calcular as probabilidades. É uma classe de algoritmo que não precisa de estruturas adicionais. Entretanto, uma das grandes desvantagens das abordagens aleatórias é o fornecimento de saídas diferentes para um mesmo conjunto de entrada [42].

3.2 *Cache* Colaborativo

O conceito de *cache* cooperativo utilizado pelos trabalhos encontrados na literatura, consiste no compartilhamento das informações armazenadas em *cache* com os demais membros da rede. Neste modelo, cada nó mantém suas preferências e políticas individuais de gerenciamento de *cache*. Quando ocorre um *cache-miss* local, o nó submeterá aos demais membros da rede uma requisição solicitando o dado não encontrado em seu *cache*. Se existir algum nó na rede que tenha a informação, este encaminhará o dado ao nó requisitante. Se nenhum nó tiver o dado, dizemos que ocorreu um *cache-miss* global. Nesse caso, a requisição será enviada diretamente ao servidor.

Obter um dado entre os vizinhos exige um custo bem maior do que reaver o dado a partir do *cache* local. Entretanto, o custo pode ser bem menor do que submeter a requisição ao servidor de origem. Essa é a idéia básica apresentada nos trabalhos correlatos. Contudo, cada trabalho possui um foco específico e trata problemas pertinentes envolvendo a colaboração de dados em um ambiente de rede *ad-hoc*. Dentre os focos de pesquisa, é possível citar:

- realizar consultas no *cache* dos demais nós [22, 46, 15, 17, 16];
- diminuir o número de mensagens em operações globais [15, 17, 16];
- criação de grupos de nós (*clusters*) [16, 17, 6];
- redistribuição de informação entre os nós de uma rede. [29]

3.2.1 Localização dos Dados

Em redes *ad-hoc*, o uso de *flooding* é uma prática comum quando um nó deseja disseminar uma mensagem a todos os membros da rede. Entretanto, o uso deste recurso deve ser ponderado devido ao custo elevado gerado pela "inundação"de pacotes. O número de pacotes aumenta consideravelmente devido a política seguida pelos nós durante um processo de *flooding*. Cada nó quando recebe uma

mensagem deve repassa-la a todos os seus vizinhos, os vizinhos irão repassar para os vizinhos deles e assim por diante. O processo termina quando o nó destino é alcançado, ou quando todos os nós receberam a mensagem.

Flooding é uma das técnicas utilizadas durante o processo de procura por um dado, após um local *miss*. Se a taxa de local *miss* for muito elevada, o uso de *flooding* pode tornar impraticável o uso de políticas colaborativas de *cache*. Para tentar melhorar esse cenário, DU [22] propôs a utilização de *flooding* com no máximo 4 saltos de profundidade. Após 4 saltos se o dado não for encontrado, os demais nós não irão mais repassar a consulta para seus vizinhos. A segunda medida, é o armazenamento das respostas, pelos nós intermediários, para que novas consultas possam ser respondidas em um número menor de saltos.

CHAND [2] limita a procura aos vizinhos situados apenas há um salto de distância. Caso ocorra um global *miss*, o nó irá submeter a consulta ao servidor. No caminho até o servidor, os demais nós irão questionar aos seus vizinhos situados há um salto de distância. O objetivo é tentar recuperar o dado armazenado o mais próximo nó requisitante, tentando com isso evitar o envio de requisições em grandes profundidades.

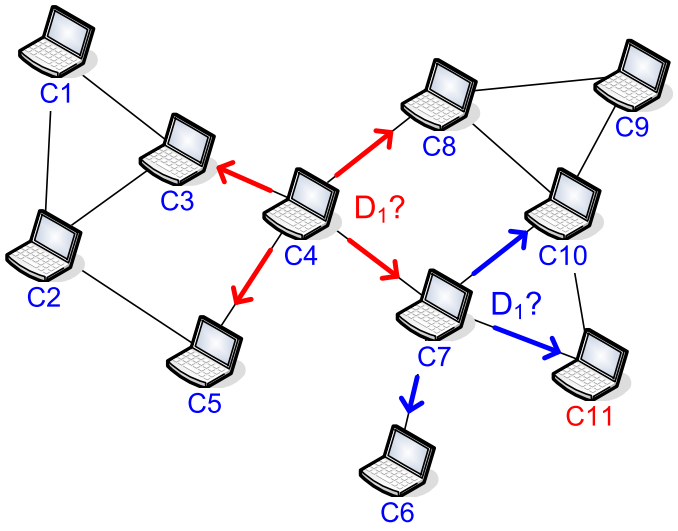

Figura 3.3: Inicialmente C_4 procura o dado D_1 entre seus vizinhos situados há um salto de distância. No caminho até o servidor cada nó procura em sua vizinhança.

Na figura 3.3, é apresentado um exemplo de funcionamento do algoritmo. Após um local *miss*, o nó C_4 submete a requisição do dado D_1 aos seus vizinhos (C_3, C_5, C_8 e C_7). Como nenhum dos nós possuem o dado, ele irá submeter a requisição ao servidor C_{11}. A requisição será roteada, e no caminho os nós intermediários irão consultar os seus vizinhos para verificar a existência do dado D_1. O objetivo do algoritmo é tentar recuperar o dado o mais perto possível de quem está requisitando.

3.3 Coerência de *Cache*

A replicação de dados em um ambiente móvel sem fio permite o acesso rápido as informações distribuídas na rede. Entretanto, a introdução de cópias acarreta no problema de gerenciamento da consistência entre elas.

3.3.1 Modelos de Consistência de *Cache*

Modelos de consistência de *cache* são políticas adotadas, pelos nós de uma rede, para gerenciar à validade das informações armazenadas no *cache* dos clientes. Essas informações possuem um período de tempo associado, em que o seu conteúdo pode ser considerado válido. Dependendo do tipo de dado gerenciado esse período pode ser maior ou menor. Um dado que é constantemente atualizado no servidor possui um tempo de validade pequeno, assim como um dado que quase nunca é atualizado irá possuir um tempo de validade grande. Normalmente, esse período de tempo é conhecido como TTL (*Time to Live*) [23].

Conforme apresentado por Cao [12], os modelos de consistência de *cache* podem ser classificados em: modelo de consistência forte (CF), modelo de consistência Δ (CD) e modelo de consistência fraca (CFr). Seja $S_0^{t_{D_i}}$ a versão do dado D_i armazenado no servidor S_0 e $C^t_{D_i,N_j}$ o *timestamp* com valor t do dado D_i armazenado na *cache* do nó N_j.

Modelo de Consistência Forte – Toda atualização do dado D_i no servidor S_0 é repassada as demais cópias do dado. Tal que para $dado(\forall_t, \forall_i, C^{t_{D_i,N_j}} = S_0^{t_{D_i}})$ o modelo *CF* é satisfeito;

Modelo de Consistência Δ – O dado D_i pode ser lido pelo nó N_j não mais que Δ tempo desatualizado. Tal que para $dado(\forall_t, \forall_i, \exists_\pi, 0 \leq \pi \leq \Delta, C^{t_{D_i,N_j}} = S_0^{t-\pi_{D_i}})$ o modelo *CD* é satisfeito;

Modelo de Consistência Fraca – O dado D_i pode ser lido pelo nó N_j desatualizado. Tal que para $dado(\forall_t, \forall_i, \exists_\pi, C^{t_{D_i,N_j}} = S_0^{t-\pi_{D_i}})$ o modelo *CFr* é satisfeito.

O custo de manutenção do modelo de consistência forte é muito oneroso em ambientes de rede sem fio. Esse custo está associado diretamente a taxa de atualização do dado e a quantidade de cópias distribuídas na rede. Quanto maior a taxa de atualização e o número de cópias, maior será o impacto na rede. Por esse motivo, a maioria dos trabalhos em ambiente de redes *ad-hoc* trabalham com modelo de consistência fraca [22, 17, 46].

3.3.2 Controle de Consistência de *Cache*

Baseado nos modelos de consistência de *cache* apresentados é possível definir qual abordagem de consistência de *cache* deve ser utilizada. Conforme apresentado por CAO [11], existem duas abordagens distintas: *Stateful* e *Stateless*.

3.3.2.1 Abordagem *Stateful*

Na abordagem *Stateful*, o servidor mantém uma lista contendo o nó e o dado fornecido. A medida que os dados são atualizados em seu *cache*, o servidor está apto a notificar os nós associados ao dado [11]. Esta abordagem permite a invalidação ou atualização das cópias utilizando *unicast* ou *multicast*, reduzindo assim, o impacto gerado por *flooding* [22]. Entretanto, está abordagem além de adicionar ao servidor uma maior carga de trabalho, possibilita à notificação desnecessária de dados que já não encontram-se armazenados nos *cache* dos nós. Huang [31], aborda este problema e propõe um algoritmo adaptativo para notificação de dados modificados.

3.3.2.2 Abordagem *Stateless*

Na abordagem *Stateless* o servidor não possui um controle das requisições. As notificações de atualização do dados podem ser através de *flooding* (*Push Based*) ou transferindo ao cliente a responsabilidade de questionar o servidor quanto a validade do dado (*Pull Based*) [11].

Em ambientes em que os dados possuem uma alta taxa de atualização, a abordagem *Pull Based* pode diminuir a carga de manutenção dos dados. Entretanto, se muitos nós passarem a fazer requisições, haverá uma degradação da rede. Trabalhos como de Bjornsson utilizam a criação de *clusters de nós* para contornar esse problema [6].

3.4 Distribuição Zipf-Like

Em trabalho apresentado em [9], Breslan et al. demonstra que o padrão de acesso à páginas Web segue o comportamento descrito pela lei *Zipf* [51] acrescido de um fator de correção α. Esse trabalho é conhecido como distribuição *zipf-like*, sendo atualmente utilizado por diversos trabalhos na área de *cache* colaborativo [17, 22, 15, 2, 16]. Partindo da observação de arquivos de logs gerados por seis instituições acadêmicas, foi possível a verificação do comportamento de acesso e a definição dos valores de α.

Para definição do comportamento de acesso, os seguintes ambientes foram analisados:

- **DEC traces** – *Digital Equipment Corporation*, *proxy* Web, servindo 17.000 estações de trabalho, foram utilizadas 3.543.968 requisições;
- **UCB traces** – *UC Berkeley*, *home ip service*, servindo toda a comunidade acadêmica, foram utilizadas 1.907.762 requisições;
- **UPisa traces** – *Universita di Pisa*, *proxy* Web, servindo departamento de ciência da computação, foram utilizadas 2.833.624 requisições;
- **Questnet traces** – *ISP Questnet from Australia*, foram utilizadas 2.885.285 requisições;

- **NLANR traces** – *National Lab for Applied Networking Research*, foram utilizadas 1.766.409 requisições;
- **FuNet traces** – *FuNet from Finland*, servindo comunicade acadêmica, foram utilizadas 4.815.551 requisições.

Algumas observações interessantes foram obtidas com esse trabalho. Entre elas, é possível observar:

- a distribuição de acesso a páginas web, a partir de um grupo fixo de usuários, segue a distribuição *zipf-like*, $\frac{\Omega}{i^{\alpha}}$. O valor de α varia entre 0.64 até 0.83;
- a regra "10/90", definida para execução de programas, não se aplica ao acesso a páginas. A concentração no acesso, aos documentos mais requisitados, está associado ao valor definido para α, sendo necessário entre 25% a 40% dos documentos para se obter 70% das requisições;
- é baixa a relação estatística entre a freqüência de acesso a uma página e o seu tamanho. Ou seja, não foi verificado que o tamanho do documento influencia na sua freqüência de acesso;
- é baixa a relação estatística entre a freqüência de acesso a uma página e a sua taxa de atualização;

3.4.1 Cálculo da Distribuição de Probabilidade *Zipf-Like*

A distribuição probabilidade *zipf-like* é definida por uma função f, tal que:

$$f(i) = \frac{\Omega}{i^{\alpha}}; i = 1, 2, 3, ..., N. \tag{3.1}$$

Onde:

- N – é o número total de objetos da distribuição. Ou seja, o número total de páginas distintas;
- Ω – é a freqüência do objeto mais freqüente da distribuição;
- α – é a popularidade relativa dos objetos da distribuição;

Como:

$$\sum_{i=1}^{N} f(i) = 1 \tag{3.2}$$

Substituindo 3.1 em 3.2, temos:

$$\sum_{i=1}^{N} \frac{\Omega}{i^{\alpha}} = 1 \tag{3.3}$$

Desenvolvendo a expressão, temos:

$$\Omega \sum_{i=1}^{N} \frac{1}{i^{\alpha}} = 1 \tag{3.4}$$

$$\Omega = (\sum_{i=1}^{N} \frac{1}{i^{\alpha}})^{-1} \tag{3.5}$$

Sendo:

$$(\sum_{i=1}^{N} \frac{1}{i^{\alpha}})^{-1} \cong \frac{1-\alpha}{N^{1-\alpha}}, \alpha \neq 1 \tag{3.6}$$

Temos que:

$$\Omega = \frac{1-\alpha}{N^{1-\alpha}}, \alpha \neq 1 \tag{3.7}$$

Sendo a equação 3.7 uma aproximação da equação 3.5, a mesma não apresenta precisão para o cálculo com valores pequenos para N. O comportamento da distribuição *zipf-like* pode ser visto no gráfico da figura 3.4. Para este cálculo, foi escolhido um valor grande de N, $N = 10^{23}$. Conforme pode ser visto no gráfico, para $\alpha = 0.66$, 10% das páginas (eixo x) representam 45.70% da requisições (eixo y).

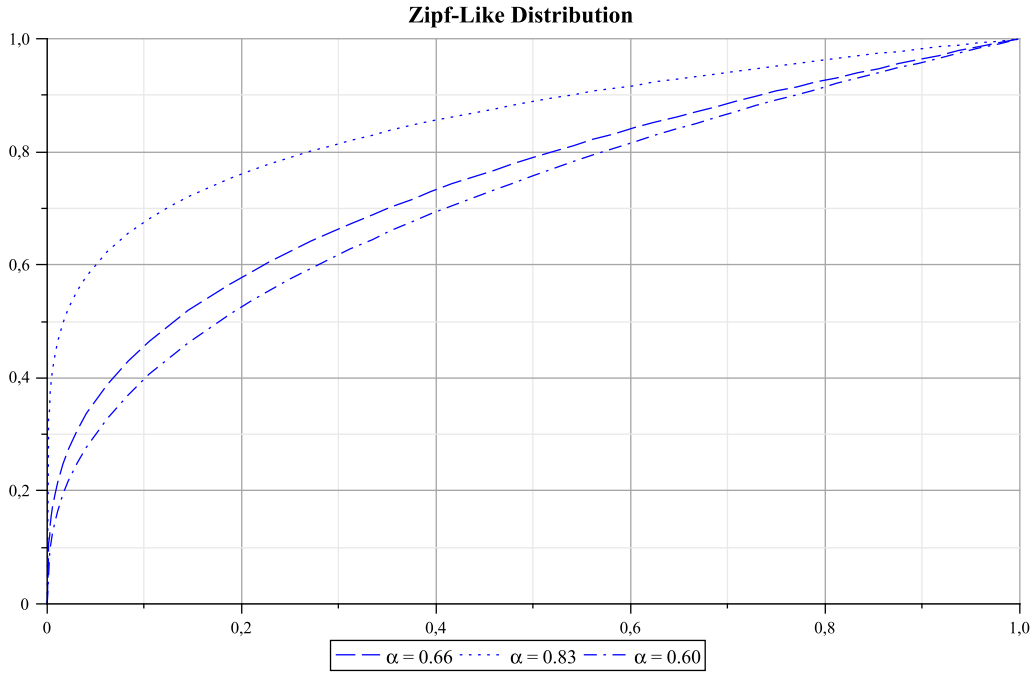

Figura 3.4: Distribuição acumulativa de requisições de Páginas Web. No eixo x temos a quantidade de páginas que equivalem a quantidade requisições no eixo y.

De acordo com o padrão de requisições de páginas descrito por Breslan et al. em [9], foi verificado que a implementação da distribuição *zipf-like*, neste trabalho, aproxima-se dos valores descritos no artigo quando $\alpha = 0.66$. É interessante

observar que este valor está dentro da faixa apresentada no artigo, onde: $0.64 \leq \alpha \leq 0.83$. Uma outra característica a ser ressaltada, é o comportamento da função de acordo com a variação de α. Para valores grandes de α, uma quantidade maior de requisições está concentra em um grupo menor de dados. Por exemplo, para $\alpha = 0.83$, 20% dos dados são mapeados para quase 80% das requisições. Enquanto que para $\alpha = 0.60$, 20% dos dados são mapeados para um pouco mais que 50% das requisições.

3.4.2 Uso das Equações

Conforme abordado na subseção 3.4.1, para valores pequenos de N, a equação 3.7 não apresenta uma boa aproximação. Para as simulações deste trabalho, foram utilizadas a equação 3.5 e 3.3, tendo em vista que os cálculos necessários para geração da distribuição de probabilidade foram executados apenas na fase de inicialização da simulação, não representando assim um gargalo ao sistema.

3.5 Trabalhos Correlatos

Yin et al. [10, 49] propuseram três políticas cooperativas de *cache*, são elas: *cacheData*, *cachePath* e *HybridCache*. A figura 3.5 apresenta o cenário onde essas políticas foram apresentadas. Nas três políticas, cada nó sempre submete sua requisição ao nó servidor, neste caso N_{11}, após um *local miss*. Como a idéia central do trabalho é fornecer uma resposta em um menor número de saltos possível, cada requisição sempre será analisada, pelos nós intermediários, antes de serem roteadas. Essa abordagem sobrecarrega o sistema e impõe um carga extra de trabalho aos nós que fazem parte do tronco principal de roteamento. Estas políticas são aplicadas aos nós intermediários e apresentam os seguintes comportamentos:

- *cacheData* – recebendo pelo menos duas requisições, de nós diferentes para um mesmo dado, será tomada a iniciativa de fazer *cache* do dado, se esse nó intermediário estiver há um salto de distância do nó requisitante;
- *cachePath* – baseado na mesma condição da política *cacheData*, realiza *cache* do caminho para a cópia do dado que será entregue e não do dado em si;
- *hybridCache* – essa é um híbrido das duas políticas anteriores. Para dados muito grandes faz *cache* do caminho e para dados com alta taxa de atualização realiza *cache* do dado;

Além do custo de processamento imposto pelas políticas, não é feito qualquer controle sobre as cópias que são geradas indiscriminadamente. Por esse motivo, utiliza política de coerência simples (TTL). Não implementa conceito de *cluster*. Entretanto, minimiza o número de mensagens utilizando apenas mensagens do tipo *unicast*.

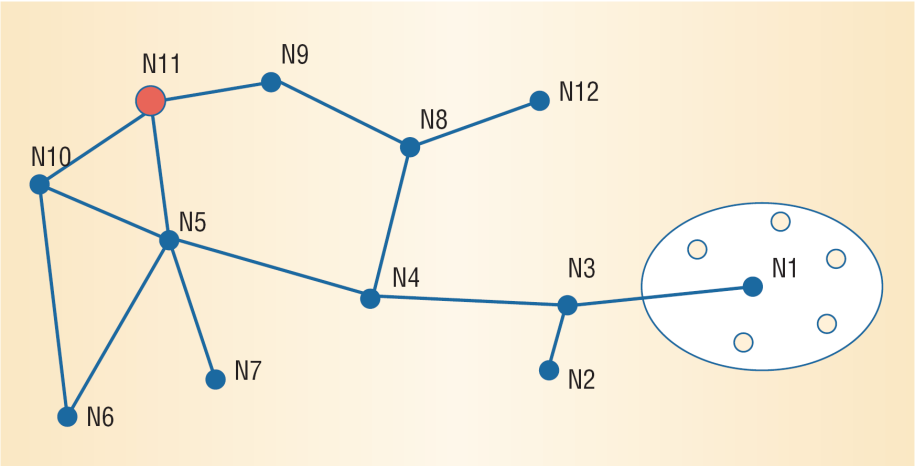

Figura 3.5: Rede *ad-hoc* composta por 11 nós. Nó N_{11} desempenha o papel de servidor e os demais nós desempenham o papel de *router* ou *host* (exemplo fonte: [10]).

Em uma política colaborativa de *cache*, a quantidade de mensagens trocadas entre nós é um fator abordado por vários trabalhos da área. O uso indiscriminado de *floading*, como técnica para localização e obtenção de dados, degrada o desempenho da rede e pode inviabilizar o seu uso [39]. Neste sentindo, Du et al. [22] demonstraram que consultas do tipo *flooding* devem limitar-se a no máximo 4 saltos. Acima desse valor, o custo para encontrar o dado entre os vizinhos mais distantes é maior do que simplesmente submeter a requisição diretamente para o servidor. Outro ponto, é a distinção em *cache* de dois conceitos: tipo de dado primário e tipo de dado secundário. Dados do tipo primário são aqueles que se encontram distantes a 4 saltos de sua cópia, já os do tipo secundário encontram-se apenas a 2 saltos. Como os dados do tipo primário possuem maior precedência sobre os secundários, durante a política de substituição local, a tendência do sistema é manter cópias de dados há mais de 4 saltos de distância, tornando o sistema mais heterogênico até 3 saltos.

Outra abordagem para diminuição no número de mensagens na rede, é a criação *clusters* de nós. A utilização de *clusters* cria uma hierarquia lógica entre os nós e confina a troca de mensagens a regiões específicas da rede. Cao et al. [12] adaptaram o modelo de *cluters* apresentado em [6] para o ambiente de rede *ad hoc*. Neste novo contexto, os *cluters* passaram a ser criados, assim como em [16], baseados na proximidade dos nós e no monitoramento contínuo dos seguintes atributos: *nível de energia*, *taxa de acesso* e *tempo de presença*. O objetivo deste trabalho, foi a criação de *clusters* para diminuição de mensagens geradas pela política de consistência forte: *invalidation* [13]. Nessa abordagem, o número de mensagens necessárias para invalidar um determinado dado é menor, pois na visão do servidor a rede é formada por apenas alguns nós (*relay peers* ou *clusters head*). O que torna a política de controle *statefull* menos onerosa e, conforme defendido pelo trabalho, justifica o emprego de políticas de coerência forte no contexto de redes sem fio.

Chow et al. [17] utilizam conceito de *clusters* e assinaturas [7] para diminuição no número de mensagens. Os *clusters* são formados baseados no padrão de acesso e movimentação dos nós. O controle dos grupos é feito pelo servidor, que recebe periodicamente informações sobre a posição dos nós através do uso de GPS (*Global Positioning System*). Essa abordagem além de tornar todo o sistema dependente de apenas um nó, no caso o servidor, obriga-o a manter tabelas referente ao que está sendo acessado e referente ao posicionamento de cada membro. O uso de assinaturas, assim como proposto em [15], permite a representação de todo o conteúdo em *cache* através de um vetor *bits*. Esses vetores são trocados entre os nós pertencentes ao grupo, o que torna possível a inferência, com um certo grau de precisão, do que está sendo armazenado pelos demais nós. Diminuindo assim, a quantidade de mensagens necessárias para recuperar as informações contidas no *cluster*. Como esse assunto foge do escopo do trabalho, seus conceitos não serão aqui abordados em profundidade. Fica apenas as referências citadas para uma investigação futura por parte do leitor.

Chand et al. [2] e Ting et al. [46] também utilizam conceito de *cluster* em seus trabalhos. Ambos restringem a 1 salto a distancia permitida entre os membros. Essa restrição, além de diminuir o tempo de resposta a consultas submetidas ao *cluster*, simplifica a comunicação entre os nós membros. Após um *global miss*, as consultas submetidas ao servidor serão analisadas, pelos nós que compõe a rota, antes de serem roteadas. Cada nó submete a requisição a sua vizinhança e só prossegue no roteamento, caso o dado não seja encontrado. A figura 3.6 apresenta essa consulta para rota formada pelo nós: $B \rightarrow D \rightarrow K \rightarrow L \rightarrow P$

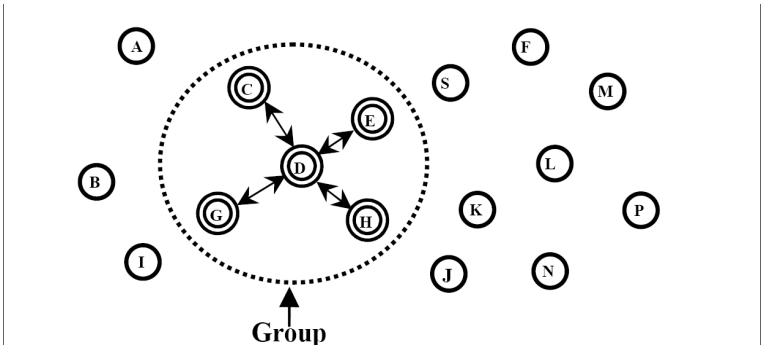

Figura 3.6: As consultas são submetidas aos vizinhos de cada nó que compõe a rota até o servidor. (exemplo fonte: [46]).

Com relação a política de substituição, Chand et al. dão preferencia por manter em *cache* os dados com as seguintes características: provenientes de nós distantes, que possuam menor taxa de atualização e com menor tamanho. Essas características possibilitam o armazenamento em cache de uma maior quantidade de dados, cujo benefício para sua recuperação local é maior e sua taxa de atualização é baixa.

A figura 3.7 apresenta uma tabela comparativa entre as políticas de *cache* apresentadas. Os seguintes pontos foram considerados:

- A maioria dos trabalhos implementa as técnicas de agrupamento de nós

(*clustering*) como forma de diminuir a quantidade de mensagens trafegadas na rede;

- o uso de *clustering* permite ao trabalho apresentado por Cao et al. ([12]) utilizar política de consistência forte (*invalidation*) no contexto de rede *ad hoc*;
- Todos os trabalhos implementam o algoritmo de substituição LRU. Esse fato é justificável pois este algoritmo apresenta, na média, um bom desempenho;
- A maioria dos trabalhos utiliza consistência fraca (TTL) como forma de gerenciamento dos dados. Essa é uma maneira de simplificação do modelo;
- Apenas Chand et al [2] e Du et al [22] implementaram algum tipo de distinção quanto aos dados armazenados em *cache*;
- Nenhum dos trabalhos apresentados implementaram qualquer política com decisão colaborativa sobre o que entra ou sai do *cache*. O conceito de colaboração restringe-se a cada nó ter suas políticas individuais de *cache* e apenas permitir que outros nós da rede possam consultar o seu conteúdo.;

	Ying-2006	**Du-2005**	**Cao-2005**	**Chow-2007**	**Ting-2006**	**Chand-2006**
Modelo de Consistência	Fraca TTL	Fraca TTL	Forte Invalidation	Fraca TTL	Fraca TTL	Fraca TTL
Política de Substituição	LRU	LRU Inter/Intra	LRU	LRU	LRU	LRU (dist, TTL, Tamanho)
Controle de Consistência	Stateless	Stateless	Statefull	Stateless	Stateless	Stateless
Uso de Cluster	Não	Não	Sim	Sim	Sim	Sim
Decisão Colaborativa	Não	Não	Não	Não	Não	Não
Distinção do Tipo de Dado	Não	Sim	Não	Não	Não	Sim

Figura 3.7: Comparativo entre as políticas de *cache* apresentadas.

Neste capítulo foram apresentados conceitos envolvendo sistema de *cache* individual e sistemas colaborativos de *cache*. Em seguida, foi apresentada a distribuição *zipf-like*. Por fim, foi apresentado um levantamento do estado da arte na área de *cache* colaborativo, seguido de uma comparação entre esses trabalhos. No próximo capítulo será apresentada a proposta para esse trabalho.

Capítulo 4

Modelo Proposto

Neste capítulo será apresentado o ambiente experimental, suas principais características e implementação. Em seguinte, será apresentado o algoritmo colaborativo de *cache* proposto e as características gerais do simulador *GloMoSim*. Será abordada a modelagem de todo sistema implementado e a relação entre as estruturas implementadas e o simulador. Por fim, serão apresentados os parâmetros configurados no GloMoSim para efetuar a simulação da prosposta.

4.1 Descrição do Ambiente Experimental

Conforme apresentado na seção 1.2, este trabalho tem como objetivo a proposta de um sistema colaborativo de *cache* para ambientes de redes *ad-hoc*. Permitindo que nós pertencentes a rede possam organizar-se entre si, formando grupos e compartilhando informações. Diferente dos trabalhos apresentados na seção 3.5, este apresenta a proposta de uma área global de *cache* destinada ao armazenamento das páginas mais acessadas pelos nós que formam esta área. Nesta proposta, os nós organizam-se formando estruturas lógicas conhecidas como *clusters*. Cada nó contribui com espaço em *cache* para o armazenamento de páginas de interesse do *cluster*. O objetivo dessa abordagem é aumentar o número de *cache global hit* para diminuir a penalização, associada a cada *cache local miss*, oriunda do modelo de *cache* tradicional.

A abordagem de *cluster* implementada neste trabalho segue o descrito por Bjornsson et al [6]. A partir da figura 4.1 é possível verificar os componentes que formam esta estrutura de organização lógica. O *cluster* exemplificado na figura é formado por quatro nós, cuja distância entre si é de apenas um salto. Todo nó pertencente ao *cluster* distancia-se, obrigatoriamente, um salto do *cluster head* e no máximo dois saltos de um outro membro. O *cluster head* é o nó responsável por toda comunicação externa ao *cluster*, desempenhado o papel de *gateway* para os demais membros. Os nós são conhecidos como *cluster nodes* e comunicam-se entre si através de *broadcast* simples. Toda comunicação interna ao *cluster* é capturada pelo *cluster head*, cuja análise fornece elementos para execução das operações de gerenciamento da área *global de cache*. De acordo com a figura 4.1, a área de *cache* de cada cliente é dividida em duas partes:

- *área global* – é a área de *cache* destinada ao armazenamento de páginas

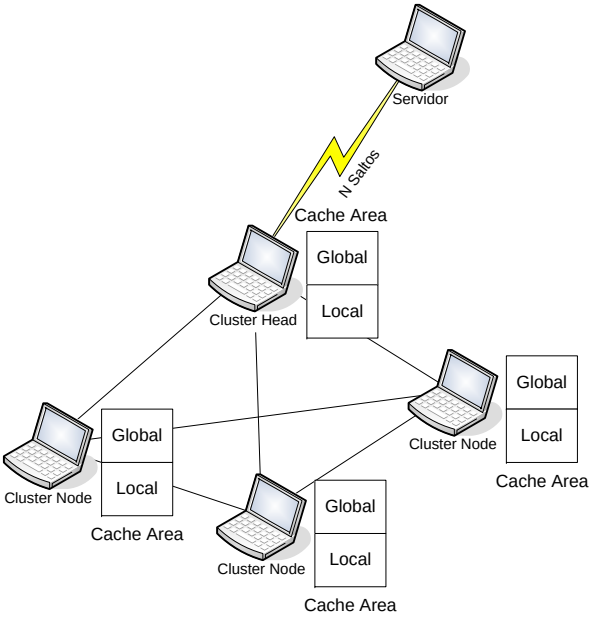

Figura 4.1: Disposição do Sistema de *Cache* Colaborativo Proposto Dentro do *Cluster* Formado.

de interesse dos demais membros do *cluster*. Esta área é gerenciada pelo *cluster head*. A soma das áreas globais, de todos os *cluster nodes* , compõe a área total de *cache* da política colaborativa;

- *área local* – é a área de *cache* destinada ao armazenamento das páginas de interesse do nó.

De acordo com o modelo colaborativo proposto, as páginas armazenadas em *cache* são classificadas em:

- *páginas globais* – são páginas identificadas pelo *cluster head* como sendo do interesse de dois ou mais *cluster nodes*. São armazenadas na área global de cada *cluster node* e o seu gerenciamento é realizado pelo *cluster head*.

- *páginas locais* – são páginas do interesse individual de cada *cluster node*. São armazenadas na área local e o gerenciamento é feito de maneira individual e independente por cada *cluster node*;

O tamanho máximo do *cache* destinado ao armazenamento das páginas, de interesse global, é de 50% do tamanho total do *cache* de cada *cluster node*. Cada *cluster node* pode utilizar esse espaço para o armazenamento de páginas de seu interesse, caso o mesmo não esteja sendo utilizado. Entretanto, nesta área existe uma precedência das páginas globais sobre as locais. Ou seja, os 50% de espaço global deve ser respeitado no momento em que uma página de interesse global seja recebida para ser armazenada. Uma página será identificada como sendo de interesse global a partir do momento que o *cluster head* verifica que mais de um nó mostrou-se interessado por ela. Quando esse evento ocorrer, o *cluster head* irá

notificar o *cluster* a respeito de qual nó será responsável por armazenar a página em sua área global.

Para validação do modelo, algumas restrições foram feitas com a finalidade de simplificar e facilitar a demonstração do funcionamento da proposta. No entanto, é importante ressaltar que a solução aqui proposta pode ser generalizada e as restrições eliminadas sem a perda das características de funcionamento do sistema. As restrições aqui apresentadas são abordadas pelos trabalhos referenciados na revisão bibliográfica, as quais são:

- a topologia do *cluster* é definida, de maneira estática, antes do início da simulação;
- os nós membros do *cluster* estão a um saltos de distância do *cluster head*;
- toda comunicação *intra-cluster* é capturada pelo *cluster head*;
- toda a comunicação para fora do *cluster* é feita por intermédio do *cluster head*;
- a topologia é estática durante o período de simulação;
- os nós não ficam indisponíveis;
- é implementado o modelo de consistência fraco TTL;

Na validação do modelo proposto, os seguintes valores foram utilizados para configuração do ambiente:

- a quantidade de páginas distintas utilizadas foram: 256 páginas, 512 páginas e 1024 páginas;
- cada cliente possui espaço suficiente em seu *cache* para armazenar até 12 páginas distintas;
- para política colaborativa, ficou definido que 50% do *cache* de cada cliente será utilizado para armazenar as páginas de interesse local e os outros 50% será utilizado para armazenar páginas de interesse global;

O gráfico da figura 4.2 apresenta a quantidade de páginas que o sistema de cache colaborativo consegue armazenar, quando se é variado o tamanho do *cluster* e a quantidade de páginas distintas. Observando o gráfico, é possível verificar que quando o número de nós é igual a um e o número de páginas distintas é igual a 256, é possível armazenar no *cache* do cliente aproximadamente 5% do total de páginas distintas. Quando o número de páginas é de 512, esse número cai para 2.5% e 1.25% para 1024 páginas. A partir de 2 nós, o gráfico representa a porcentagem de *cache* referente a área colaborativa. Como cada cliente reserva 50% do *cache* para o armazenamento de páginas globais, 2 clientes proporcionam o armazenamento dos mesmos 5% referentes as 256 páginas. Para 4 clientes esse número sobe para 10%, chegando no máximo de 40% quando o número de clientes é 16. Nota-se que quando o número do nós é igual a um, não faz sentindo

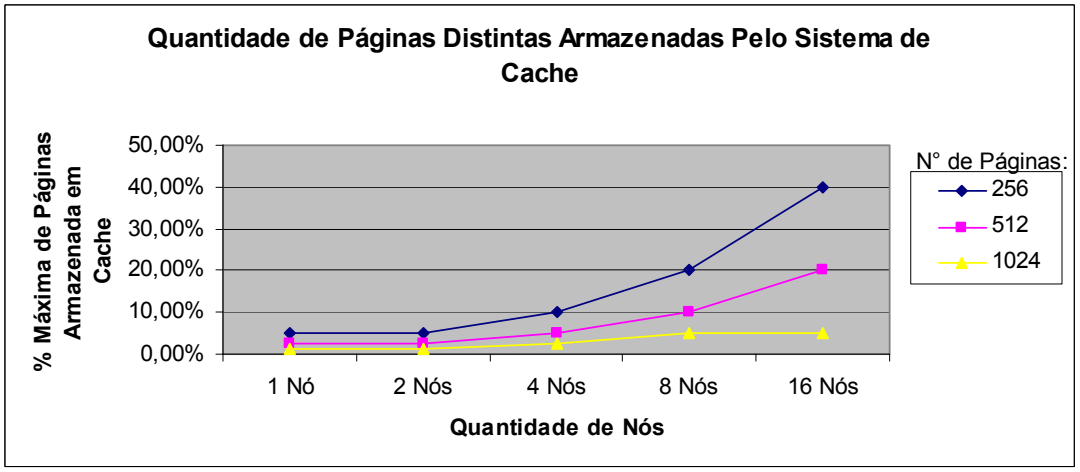

Figura 4.2: Quantidade de Páginas Distintas Armazenadas Pelo Sistema de *Cache*.

implementar uma política colaborativa de *cache*. Para este caso, 100% do *cache* do cliente é utilizando para armazenar páginas de interesse local do nó.

Para validação e comparação do modelo de *cache* colaborativo proposto, também foi implementado um sistema individual e tradicional de *cache*. Nesse sistema, os nós de uma rede não colaboram entre si e toda requisição que não pode ser respondida localmente é enviada, inevitavelmente, ao servidor de páginas da rede. Nas próximas subseções será apresentado o algoritmo colaborativo proposto e na seqüência a modelagem e implementação desse sistema de *cache* colaborativo.

4.2 Protocolo Colaborativo de Cache

Conforme apresentado na seção 3.5, diversos trabalhos foram propostos na área de *cache* colaborativo para ambientes de redes *ad hoc*. Conforme visto, a figura 3.7 apresenta uma tabela comparativa entre os trabalhos apresentados pela revisão bibliográfica. A maioria dos trabalhos apresenta as seguintes características:

- utilizam modelo de consistência fraca para simplificação do modelo;
- utilizam a política de substituição LRU. Na média essa política apresenta um bom resultado;
- utilizam modelo de controle de consistência *stateless*. Dos modelos analisados que implementam esse modelo de controle, não foi encontrado necessidade da utilização do modelo *statefull*;
- utilizam técnica de *clustering* como forma de redução e concentração das mensagens na rede;
- não é feita nenhuma distinção do tipo dos dados que são armazenados em *cache*;

- os dados são armazenados de acordo com os interesses individuais de cada nó. Não existe uma política colaborativa para armazenamento dos dados;

O objetivo deste trabalho é a redução no número de mensagens encaminhadas para o servidor, através de uma proposta colaborativa de *cache*. Assim como os trabalhos referenciados, esta proposta também implementa o conceito de *clustering* como forma de redução e concentração do número de mensagens. Assim como em [17], os *clusters* são formados a partir de requisitos mínimos: espaço de armazenamento e localização na rede. Entretanto, para simplificação do modelo, os *clusters* são formados de maneira estática antes do início da simulação do sistema. O comportamento do sistema de *cache* colaborativo proposto é apresentado na seqüência em forma de tratamento para casos específicos do ambiente.

4.2.1 Tratamento de um *Cache* Local *Hit*

Um *cache* local *hit* ocorre no escopo dos *cluster nodes* e *cluster head*, quando esse último desempenha o papel de *cluster node*. A aplicação *Browser*, localizada no *cluster node*, submete uma requisição de página ao sistema de *cache* local. O sistema de *cache* local, ao verificar que exite uma cópia válida da página armazenada, identifica esta como sendo recentemente acessada e encaminha a resposta para a aplicação. Nesse caso, nenhum acesso a rede foi necessário.

4.2.2 Tratamento de um *Cache* Local *Miss*

O fluxo de execução do algoritmo é o mesmo do apresentado anteriormente. Entretanto, um local *miss* ocorre quando a página solicitada não encontra-se armazenada localmente, ou quando a cópia existente está inválida. Se a página estiver inválida, ela será excluída do *cache*. Nesse ponto, o *cluster node* irá submeter uma requisição ao demais membros do *cluster*. A requisição é enviada através de um *broadcast*. Ao receber a requisição, o *cluster head* irá aguardar por alguns estantes na esperança que outro membro da rede responda a requisição. Esse tempo faz-se necessário pois o *cluster head* só possui a relação dos dados armazenados na área global de *cache*. É possível que outro nó possua uma cópia válida do dado em sua área local. Durante esse período, se outro nó responder a requisição, o *cluster head* saberá que a página é de interesse de mais de um nó e nesse caso submeterá a página para área global de *cache* (caso ela não esteja).

Passado o tempo de espera, se nenhum nó responder, o *cluster head* irá verificar na sua tabela de *cache global* se o dado está armazenado na área global de *cache*. É importante ressaltar que nesse ponto, o *cluster head* não tem como saber com certeza se a página não está no *cluster*, pois alguns nós podem estar a dois saltos do nó requisitante. Caso a página esteja relacionada na área global de *cache*, o tratamento será o descrito na subseção 4.2.3, caso contrário, o *cluster head* precisa submeter a requisição ao *cluster* para certificar-se de que a página não está armazenada na área local de algum nó não alcançado pelo *broadcast* inicial. Após o segundo *broadcast*, o *cluster head* irá aguardar por mais algum tempo e irá executar o mesmo procedimento descrito anteriormente caso algum nó responda. Ao final do segundo tempo de espera, o *cluster head* irá considerar

que ocorreu um *global miss* e o tratamento para esse evento é o apresentado na subseção 4.2.4.

4.2.3 Tratamento de um *Cache* Global *Hit*

Neste ponto, o *cluster head* sabe que a página solicitada encontra-se armazenada na área global de *cache* e também é sabido em qual nó encontra-se armazenado essa página. O primeiro passo é verificar se a página armazenada no *cache* global encontra-se válida. Através de um *tag* é possível verificar essa informação. Se a página estiver desatualizada, o *cluster head* irá submeter uma requisição ao servidor central da rede. Quando o *cluster head* receber a resposta, a página será submetida ao *cluster*. Associado a resposta, será anexado uma ordem para atualização da página armazenada na área global de cache. Caso a página não encontre-se desatualizada, o *cluster head* irá submeter uma requisição ao nó que possui a página armazenada na sua região de *cache* global. Neste ponto, é possível verificar que esse nó encontra-se a dois saltos do nó requisitante. O nó que possui a página irá enviar a reposta endereçada diretamente ao nó requisitante (*unicast*).

4.2.4 Tratamento de um *Cache* Global *Miss*

Neste ponto, o *cluster head* sabe que a página não foi encontrada no *cluster* e irá submeter uma requisição ao servidor da rede. Ao receber a responsta, o *cluster head* irá submete-la aos membros da rede através de uma operação de *broadcast*.

4.2.5 Modelo de Consistência de *Cache*

Conforme verificado nos trabalhos correlatos, como forma de simplificação do modelo, a maioria dos trabalhos implementam um modelo de consistência fraca TTL (*Time To Live*). Como decisão de projeto, optou pelo uso de consistência fraca com TTL, com abordagem de controle *Stateless*.

4.3 Arquitetura do Ambiente

Para validação das idéias apresentadas, optou-se pela implementação do modelo proposto utilizando o simulador de redes sem fio *GlomoSim* [50]. O simulador GloMoSim é largamente utilizado para validação de trabalhos associados a redes *MANET*. Por ser um simulador modularizado, estável e de fácil implementação, é bastante difundido entre a comunidade acadêmica, conforme pode ser observado através dos trabalhos apresentados na revisão bibliográfica. Nas próximas subseções serão apresentadas algumas características desse simulador e como a implementação deste modelo está associada ao simulador.

4.3.1 GloMoSim

Global Mobile Information System Simulator (*GloMoSim*) é definido como um ambiente de simulação escalar para comunicação de grandes redes sem fio. Glo-

MoSim utiliza simulação de eventos discretos paralelos provido pelo compilador paralelo Parsec [4]. Uma das suas principais características é a capacidade de trabalhar com centenas de nós conectados através de um ambiente de comunicação heterogêneo. Maiores informações a respeito do simulador podem ser encontrados em [35]. A tabela 4.1 apresenta a relação, por camada, dos principais protocolos implementados pelo simulador GloMoSim.

Camada	**Protocolo**
Physical	Free space, Two-Ray
Data Link (MAC)	CSMA, MACA, TSMA, 802.11
Network (Routing)	Bellman-Ford, FSR, OSPF, DSR, WRP, LAR e AODV
Transport	TCP, UDP
Application	Telnet, FTP, CBR, HTTP, ...

Tabela 4.1: Relação de Protocolos por Camada Implementados pelo GloMoSim.

O simulador GloMoSim apresenta estrutura modularizada e bem definida, o que permite a inclusão de novos protocolos em qualquer uma das camadas apresentadas na tabela 4.1. O núcleo do simulador trabalha, basicamente, processando as mensagens trocadas entre camadas. Cada camada está associada a uma estrutura conhecida como *GlomoNode* e cada instância dessa estrutura representa um nó na rede. Após a fase de inicialização do sistema, o escalonador interno do simulador irá tratar as mensagens geradas e encaminha-las para serem tratadas por suas respectivas camadas. Durante o período de simulação, cada camada associada a cada nó irá gerar mensagens conforme o comportamento dos protocolos implementados. Ao final da simulação, funções de finalização serão invocadas e estatísticas de simulação serão geradas. Basicamente, para a incorporação de um novo protocolo, faz-se necessário a implementação das seguintes funções:

- **Função de Inicialização** – função responsável pela alocação e inicialização das estruturas utilizadas durante a simulação;
- **Função de Finalização** – liberação de memória alocada e criação das estatísticas de simulação;
- **Função de Tratamento de Evento** – função de tratamento para toda mensagem que deve ser tratada pelo protocolo.

Em cada camada do simulador estão definidas as três funções citadas. Como o modelo proposto foi implementado na camada de aplicação, as seguintes funções foram alteradas para integração com o simulador: *GLOMO_AppInit*(), *GLOMO_AppFinalize*() e *GLOMO_AppLayer*(). A figura 4.3 apresenta uma visão macro dos principais módulos implementados, os quais são descritos abaixo:

- **Zip-Like Requests** – módulo responsável por fornecer requisições de páginas de acordo com a distribuição de probabilidade *zipf-like*. Aceita os seguintes parâmetros de configuração: α, nós requisitantes, número de páginas distintas e quantidade total de requisições;

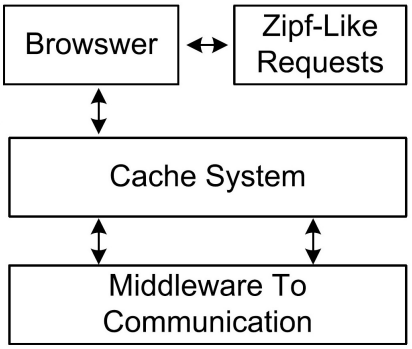

Figura 4.3: Componentes básicos do modelo proposto.

- **Browser** – módulo que emula requisições de páginas web baseado na distribuição de requisições fornecida pelo módulo *Zip-Like Requests*;
- **Cache System** – módulo principal do sistema proposto. Implementa o comportamento de *cache* tradicional, bem como o modelo de *cache* colaborativo proposto.
- **Middleware For Communication** – módulo implementado com objetivo de criar uma relação de dependência fraca entre o modelo proposto e o simulador GloMoSim.

4.3.2 Implementação do Sistema Proposto

Na pilha de protocolos TCP/IP, a camada de aplicação corresponde a última camada. Nela encontram-se todos os protocolos que dão suporte às aplicações dos usuários. Nesse sentindo, é nessa camada em que a proposta de um sistema colaborativo de *cache* se enquadra. A figura 4.4 apresenta uma visão macro dos módulos que compõe o sistema, bem como sua relação com os outros protocolos da pilha TCP/IP.

Diferente de algumas proposta implementadas em simuladores, o sistema proposto apresenta todas características de uma aplicação real. Durante à simulação, cada nós aloca um espaço de memória próprio, onde estruturas de dados reais são utilizadas para representar um sistema real de *cache*. Nesse sistema, por exemplo, existe a representação de estruturas do tipo páginas Web com as seguintes características: tamanho, conteúdo, *time stamp* e url. Durante o processo de modelagem do sistema, tomou-se o cuidado para que a implementação se aproximasse o máximo possível de um ambiente real. Neste trabalho apenas a parte de comunicação entre os nós segue a metodologia tradicional de simulação.

Com objetivo de criar um relação de dependência fraca entre o sistema proposto e o ambiente de simulação utilizado, foi implementado uma camada intermediária (*middlerware communication*) entre a *interface* da camada de transporte e o sistema de *cache*. Com isso, o sistema proposto não está amarrado a interface disponibilizada pelo simulador e sim pela nova interface disponibilizada pela camada intermediária. Neste caso, o esforço para portar o ambiente proposto

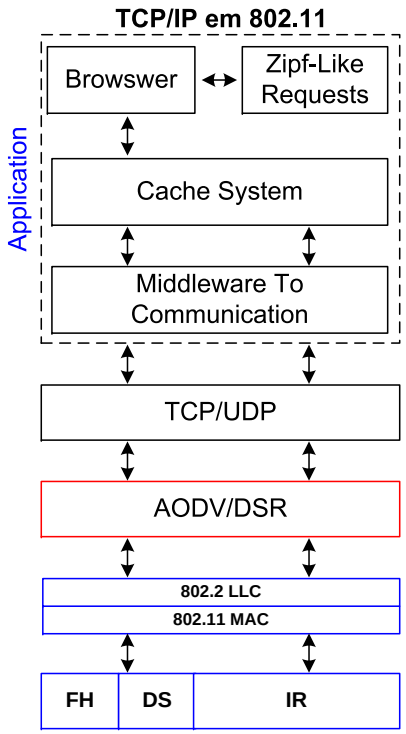

Figura 4.4: Disposição do modelo proposto de acordo com a pilha de protocolos TCP/IP no contexto de redes *ad-hoc*.

para um outro simulador ou até mesmo um ambiente real, seria reimplementar a interface da camada intermediária para o outro ambiente desejado.

Para verificar o funcionamento do sistema de *cache* proposto, foi implementado uma aplicação, referida no texto como *browswer*, a qual submete requisições de páginas ao sistema de *cache* proposto. Para isso, o *browser* utiliza o módulo *zipf-like requests* o qual fornece o padrão de requisições que deve ser executado por cada nó participante da simulação. Este módulo gera requisições de acordo com a distribuição *zipf-like* [9], a qual será apresentada na seção 3.4.

A figura 4.5 apresenta a relação entre as principais estruturas implementadas por este modelo colaborativo. Em uma rede *ad-hoc*, dois ou mais nós podem se associar para compor um *cluster*. Este grupo é composto, obrigatoriamente, por um nó que desempenha o papel de *cluster head* do grupo. Cada *cluster* está associado a um servidor. Neste modelo, o servidor é definido como um nó que possui a capacidade de fornecer, aos demais nós da rede, as informações consumidas por eles.

Conforme apresentado, o *cluster head* trabalha como *gateway* do *cluster*. Como todos os nós membros estão a um salto de distância do *cluster head*, este consegue identificar as mensagens que estão sendo trocadas internamente no *cluster*. Este tipo de informação permite a identificação das páginas mais acessadas pelo cluster e, desta forma, o seu armazenamento na área global de *cache*. O gerenciamento da área global de *cache* é de responsabilidade do *cluster head*. O fato do *cluster head* concentrar todas as informações referentes a área global de *cache* torna-o um ponto único de falha. Este problema tem sido exaustivamente

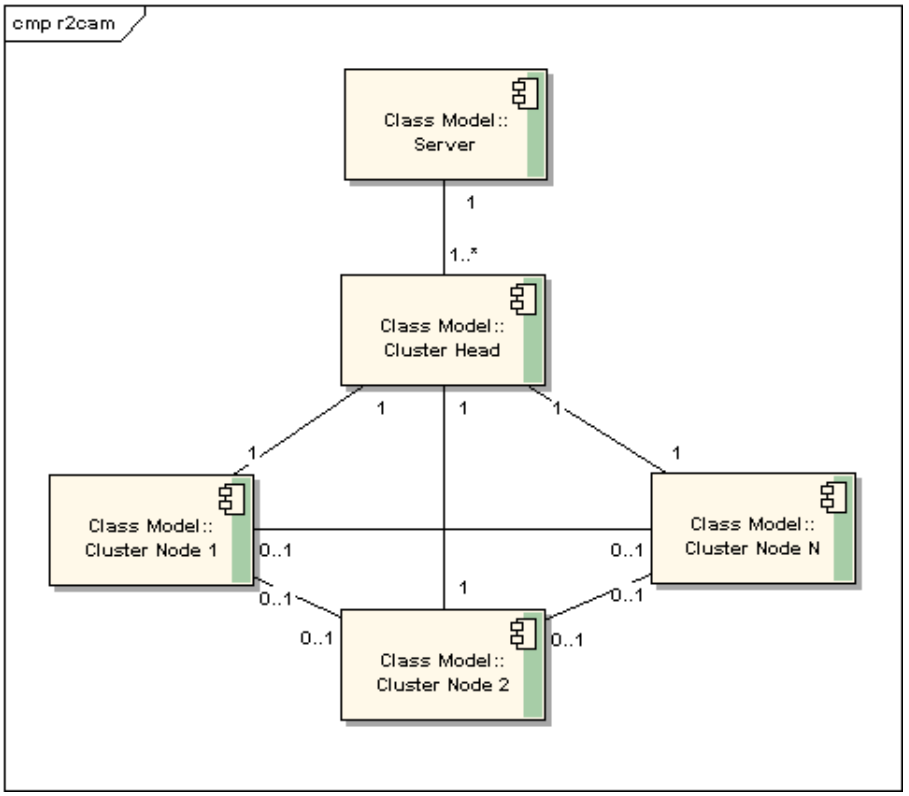

Figura 4.5: Relação de Associação Entre os Principais Módulos da Política Colaborativa de *Cache* Proposta.

abordado na literatura [18], tornando-o fora do escopo do trabalho.

A figura 4.6 apresenta a estrutura de dados utilizada pelo *cluster head* para gerênciar a área global de *cache*. O compilador PARSEC [4] utilizado pelo GloMoSim não permite linkar, junto ao código do simulador, bibliotecas como a GLIB [45]. Esta limitação impôs um ônus ao projeto, obrigando a implementação de todo o tipo de estrutura de dados utilizada. Cada nó membro do *cluster* contribui com espaço em *cache* para o armazenamento de páginas do tipo global. O controle do espaço total disponível na área global de cada nó, bem como a quantidade de espaço alocado em cada nó é feito através da estrutura do tipo **r2NodeList**. Esta é uma estrutura de dados do tipo lista e cada posição representa um nó do *cluster*. As páginas armazenadas na área global são inseridas na estrutura do tipo **r2GlobalStackNode**. Esta também é uma estrutura de lista que armazena, em cada elemento, as seguintes informações:

- **nodeAddr** – endereço do nó que armazena a página em sua área global de *cache*;
- **size** – tamanho da página;
- **ttl** – tempo estimado para expiração da página;
- **url** – url da página que está sendo armazenada;

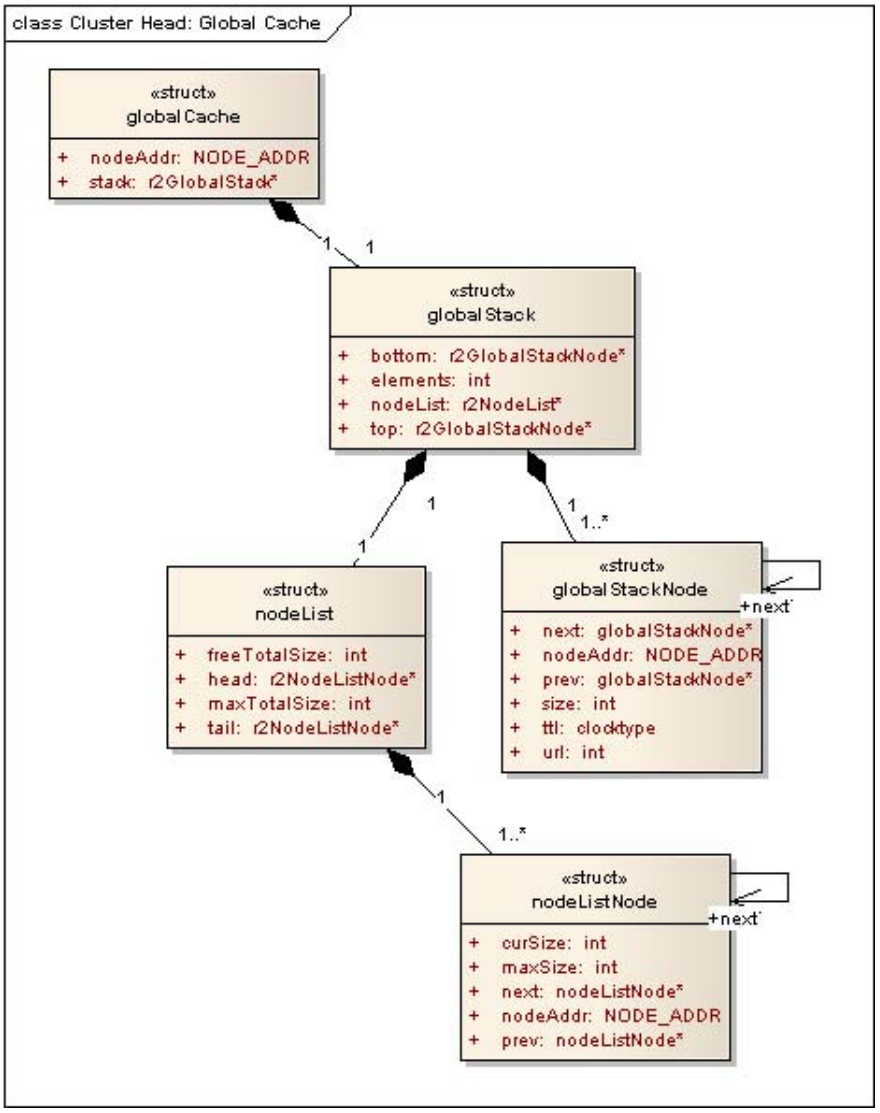

Figura 4.6: Estruturas Relacionadas ao Módulo de Cache Colaborativo implementado Pelo *Cluster Head*.

A figura 4.7 apresenta a implementação do sistema de *cache* utilizado por cada *cluster node*. Na estrutura do tipo **r2Stack**, são armazenadas as seguintes informações:

- **curGlobalSize** – quantidade de memória atual utilizada para armazenar páginas de interesse global;
- **curLocalSize** – quantidade de memória atual utilizada para armazenar páginas de interesse local;
- **maxGlobalSize** – quantidade máxima de memória a ser utilizada para área global de *cache*;
- **maxTotalSize** – tamanho total do sistema de cache;

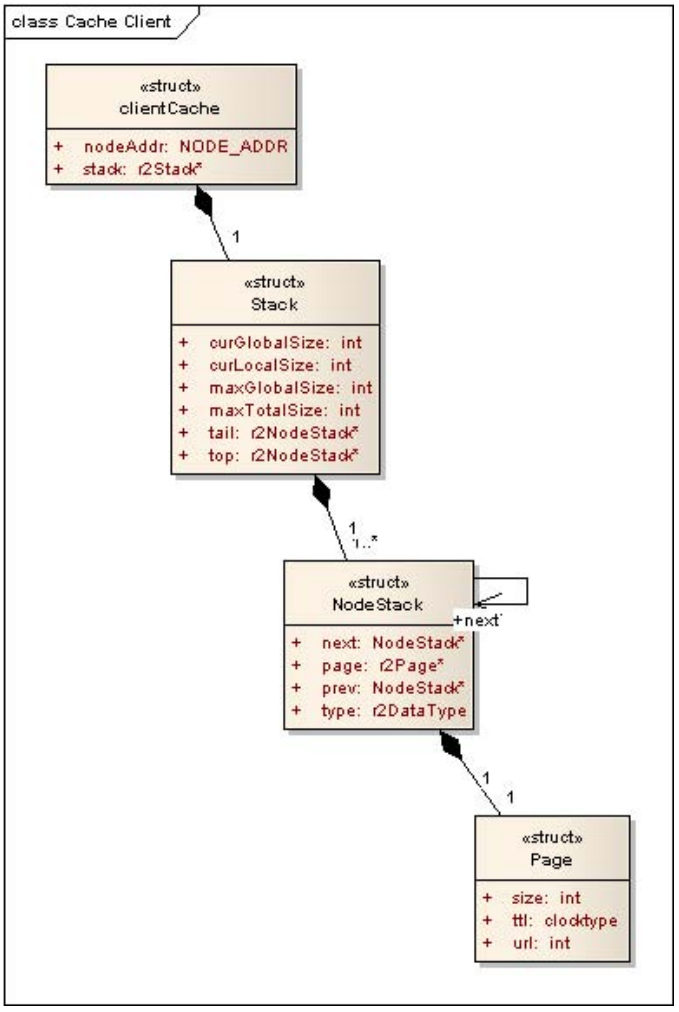

Figura 4.7: Estruturas Relacionadas ao Módulo de Cache Implementado Pelo *Cluster Node*.

Todas as páginas armazenadas pelo sistema de *cache* são do tipo **r2Page**. A estrutura de dados do tipo **r2Stack** apresenta o comportamento de uma pilha, sendo o elemento do topo o mais recentemente acessado e o da calda o menos recentemente acessado. Cada elemento da pilha é do tipo **r2NodeStack** e serve para armazenar uma estrutura do tipo **r2Page**. A distinção entre páginas globais e locais é feita através do campo **r2DataType**. Quando uma página é recupera do *cache*, a mesma é colocada no topo da pilha. Quando há necessidade de liberar espaço em *cache*, para o armazenamento de uma página proveniente de fora, os elementos a partir da calda são removidos. Esses procedimentos fazem com que o sistema apresente o comportamento da política LRU. Páginas marcadas como sendo do tipo global são gerenciadas pelo *cluster head* e só podem ser excluídas pelo nó local se este receber uma mensagem de gerenciamento proveniente do *cluster head*.

O sistema de *cache* é otimizado para dar preferência ao armazenamento de páginas locais caso o espaço destinado ao armazenamento global não esteja sendo utilizado. A medida que as páginas globais forem sendo enviadas para armaze-

namento pelo *cluster head*, elas passam a ter maior prioridade sobre as páginas locais e antigas armazenadas em *cache*. É importante destacar que o valor definido em **maxGlobalSize** deve ser respeitado, o que impede o armazenamento de páginas globais acima do valor acordado entre os membros do *cluster*.

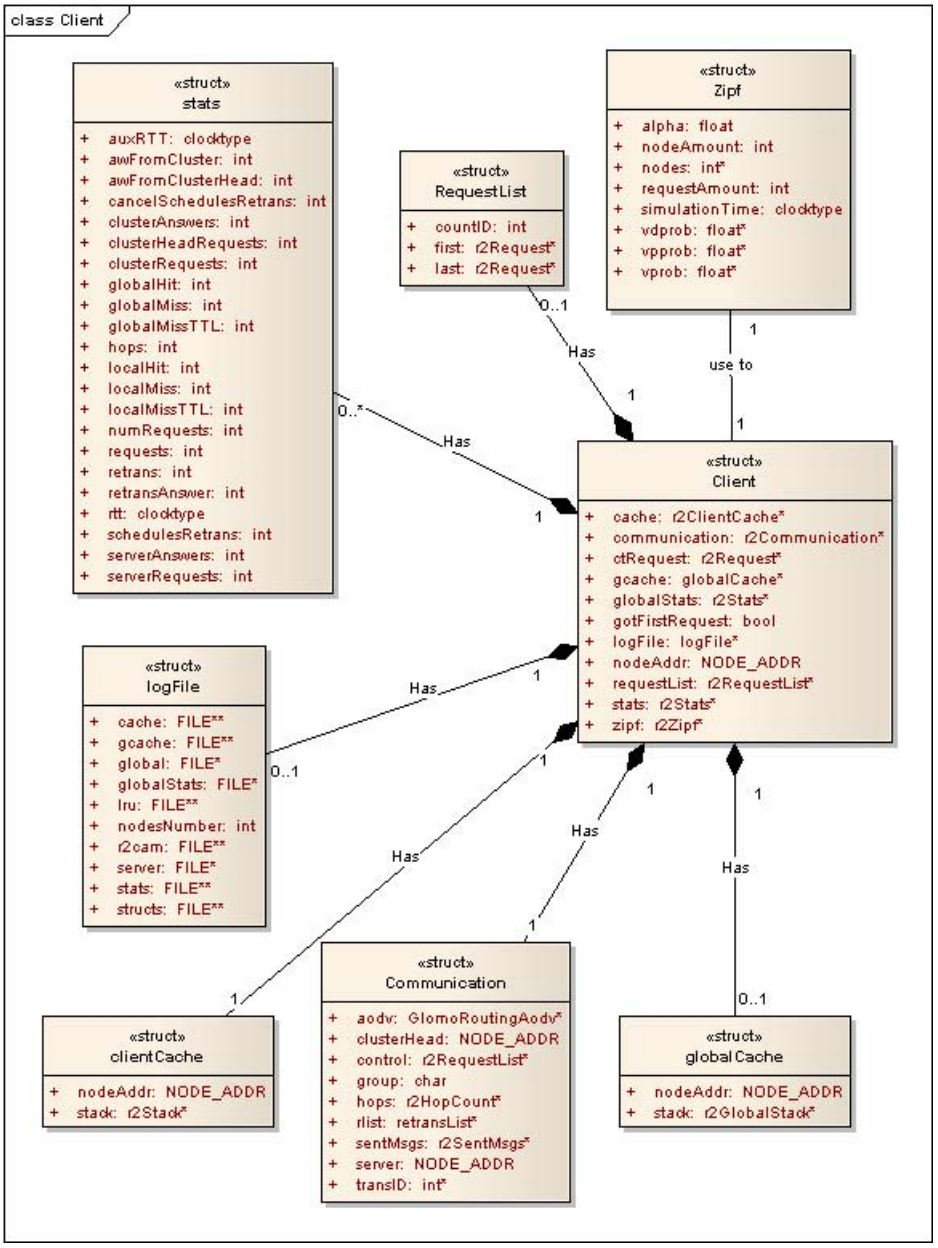

Figura 4.8: Estruturas Relacionadas ao Módulos Implementados Pelo Cliente.

A figura 4.8 apresenta todas as estruturas de dados implementadas pelos nós clientes. Em uma visão macro, elas apresentam as seguintes características:

- **r2Client** – estrutura principal que reúne todas as outras estruturas utilizadas pelo sistema;
- **r2Stats** – estrutura responsável pelo armazenamento das estatísticas colhidas durante a simulação;

- **r2RequestList** – estrutura referencia todas as requisições submetidas por um determinado nó;
- **r2Zipf** – armazena os atributos necessários para criação do padrão de requisições a serem submetidas durante a simulação do modelo;
- **r2LogFile** – estrutura responsável pelas operações realizadas nos arquivos de log na fase final da simulação;
- **r2ClientCache** – estrutura de *cache* utilizada pelos *cluster nodes*;
- **r2Communication** – estrutura utilizada pela camada de comunicação com a finalidade de criar uma dependência fraca entre o GloMoSim e o modelo implementado. Dentre seus atributos, destacam-se:

  aodv – estrutura definida e utilizada na camada do roteamento do simulador. É referenciada na camada de aplicação pois faz-se necessário o acesso da tabela de roteamento para fins de estatística;

  cluster head – indica quem é o *cluster head* do *cluster* o qual o nó faz parte;

  group – defini qual é o ID do *cluster* o qual o nó faz parte. Toda mensagem endereçada ao grupo leva esse identificador e apenas os nós pertencentes ao mesmo irão ler a mensagem;

  r2RetransList – dado as característica da aplicação, é utilizado na camada de transporte o protocolo UDP. Como a entrega de pacotes não é garantida por esse protocolo, esta estrutura realiza o controle das mensagens enviadas e aciona o protocolo de retransmissão da camada de aplicação quando for necessário. A entidade **r2RetransList** implementa esse protocolo;

  server – campo utilizado apenas pelos nós que desempenham o papel de *cluster head* no sistema. Nele é armazenado o endereço do servidor geral do sistema;
- **r2GlobalCache** – estrutura utilizada pelo *cluster head* para o gerenciamento da área global de *cache*;

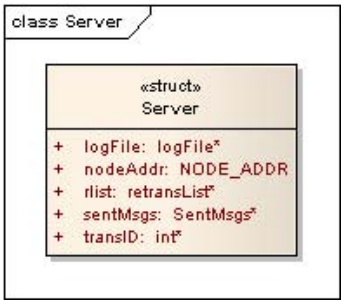

Figura 4.9: Estrutura Relacionada ao Módulo Implementado Pelo Nó Servidor da Rede.

A figura 4.9 apresenta a estrutura que representa o servidor de páginas de todo o sistema. Quando uma página não encontra-se no *cache global*, o *cluster head* encaminha a requisição para este servidor. O mesmo trabalha no modelo descrito como *stateless*, ou seja, as requisições chegam e o servidor apenas precisa encaminhar as páginas como resposta. Sendo este um modelo de gerenciamento simples, não se faz necessário a implementação de estruturas complexa. Dentre os atributos relevantes, encontra-se os listados abaixo:

- **r2LogFile** – referência para estrutura global responsável pela manipulação dos arquivos de log. Os logs gerados por esta estrutura são apenas para facilitar o processo de verificação do funcionamento do sistema simulado;
- **NodeAddr** – Armazena o endereço local do nó;
- **r2RetransList** – dado as característica da aplicação, é utilizado na camada de transporte o protocolo UDP. Como a entrega de pacotes não é garantida por este protocolo, esta estrutura realiza o controle das mensagens enviadas e aciona o protocolo de retransmissão da camada de aplicação quando for necessário. Este protocolo de retransmissão encontra-se implementado por esta entidade;

4.3.3 Parametros do Simulador *Glomosim*

Para a simulação do modelo de *cache* colaborativo proposto no simulador *GloMoSim*, foram utilizados os valores descritos na tabela 4.2.

Parâmetro	**Valor**
NUMBER-OF-NODES	24
MOBILITY	NONE
PROPAGATION-PATHLOSS	TWO-RAY
RADIO-TYPE	RADIO-ACCNOISE
RADIO-FREQUENCY	2.4e9
RADIO-BANDWIDTH	2000000
RADIO-TX-POWER	1.1
RADIO-ANTENNA-GAIN	0.0
RADIO-RX-SENSITIVITY	-91.0
RADIO-RX-THRESHOLD	-79.0
MAC-PROTOCOL	802.11
NETWORK-PROTOCOL	IP
ROUTING-PROTOCOL	AODV

Tabela 4.2: Valores dos parâmetros utilizados no *GloMoSim* durante a simulação do modelo proposto.

Neste capítulo foi apresentado o modelo de *cache* colaborativo proposto, bem como o ambiente experimental utilizado. Considerações e restrições sob o modelo foram discutidas. Em seguinda, foi abordado detalhes quanto a implementação do

modelo e a relação entre as estruturas implementadas e o simulador *GloMoSim*. Por fim, foi apresentado as configurações utilizadas no *GloMoSim* para simulação da proposta. No próximo capítulo será apresentado as simulações realizadas, bem como os resultados obtidos e as suas respectivas análises.

Capítulo 5

Resultados Experimentais e Análise

Neste capítulo serão apresentados os primeiros resultados experimentais obtidos a partir da simulação do modelo colaborativo de *cache* proposto, o qual foi apresentado no capítulo 4. Também será apresentado uma análise dos resultados obtidos, com o objetivo de validar a proposta apresentada.

5.1 Ambiente de Simulação

A proposta apresentada foi comparada com modelo tradicional e individual de *cache*. O objetivo é demonstrar o funcionamento, e possível benefício, de uma abordagem colaborativa sobre um individual. No modelo tradicional implementado, cada nó possui uma área individual de *cache* a qual é consultada toda vez que a sua aplicação deseja uma página. Neste modelo, *local miss* são tratados com o envio da consulta para o servidor central da rede.

A figura 5.1 apresenta a topologia de rede utilizada durante as baterias de testes de validação. O servidor de páginas da rede, representado pelo nó mais afastado do *cluster*, disponibiliza, de uma maneira transparente, todas as páginas requistadas pelos demais nós da rede. Os nós não pertencentes ao *cluster* realizam apenas roteamento das requisições durante a execução das baterias de testes. Toda bateria foi repetida 6 vezes e os resultados obtidos tratados. Os valores discrepantes foram retirados e a média ponderada foi obtida, formando os valores finais apresentados no gráfico. O número total de nós na rede permaneceu sempre constante, contudo a mesma bateria de teste foi repetida para o *cluster* formado por: 2, 4, 8 e 16 nós.

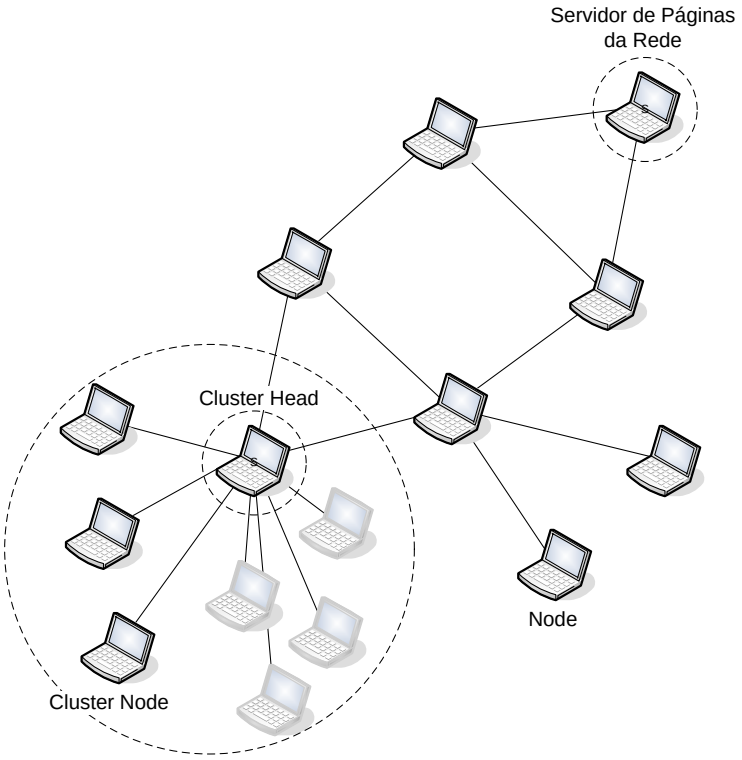

Figura 5.1: Topologia de Rede Utilizada no Ambiente de Simulação.

A modelagem da capacidade de armazenamento do sistema de *cache* é apresentada na figura 4.2. Um descrição detalhada foi apresentada no capítulo anterior. Nas próxima seções serão apresentados os resultados obtidos com as baterias de testes executadas.

5.2 Verificação da Carga no Servidor

O objetivo dessa primeira bateria de testes foi verificar a quantidade de requisições que foram enviadas ao servidor. Comparando com a política individual de *cache*, espera-se uma redução na quantidade de requisições enviadas. A figura 5.2 apresenta a porcentagem de mensagens submetidas ao servidor quando a quantidade de membros do *cluster* aumenta. Com uma quantidade maior de membros o tamanho da área de *cache* global também aumenta, e com isso, a quantidade de global *hits*. A quantidade de *hits* é inversa a quantidade de requisições submetidas ao servidor.

O melhor caso ocorre quando o número de páginas distintas é igual a 256. De acordo com a figura 4.2, nesse contexto, cada nó consegue armazenar em *cache* até 5% das 256 páginas. Nesse caso, com 16 nós a área global de *cache* consegue armazenar 40% das páginas. Os 16 nós utilizando políticas individual de *cache* enviam ao servidor, cada um, 77.83% de suas requisições. Enquanto que utilizando a política colaborativa de *cache*, cada um envia apenas 27.05% da requisições. O pior caso ocorre para 1024 páginas distintas. A política colaborativa, para 16 nós, envia 55.75% das requisições, enquanto a política individual envia

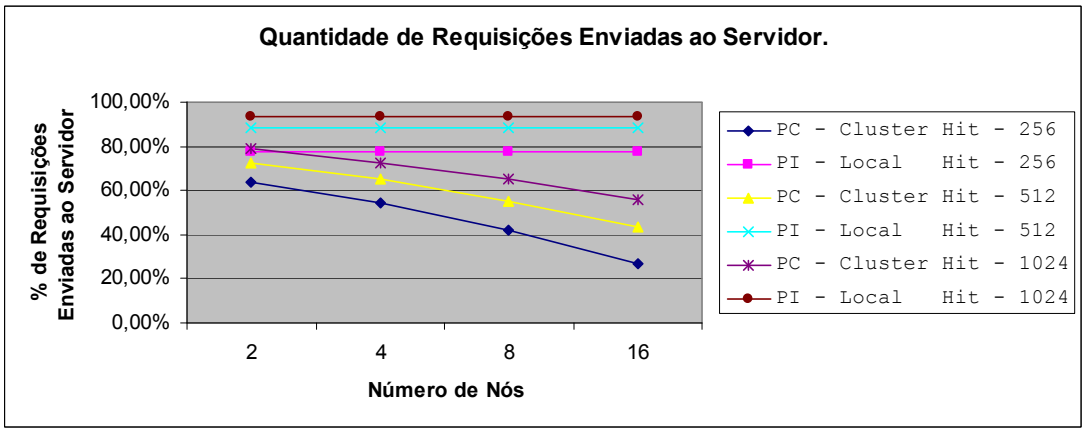

Figura 5.2: Quantidade de Requisições Enviadas ao Servidor.

93.77% das requisições.

5.3 Verificação da Eficiência do Sistema de *Cache*

O objetivo dessa segunda e terceira baterias de testes é verificar se a política colaborativa de *cache* comporta-se da maneira correta. De acordo com a figura 5.3, para o melhor caso da política colaborativa, 256 páginas e 16 nós, 72.95% das requisições foram respondidas dentro do *cluster* (*cluster hit*). Enquanto que a política individual obteve apenas 22.18% de *local hit*.

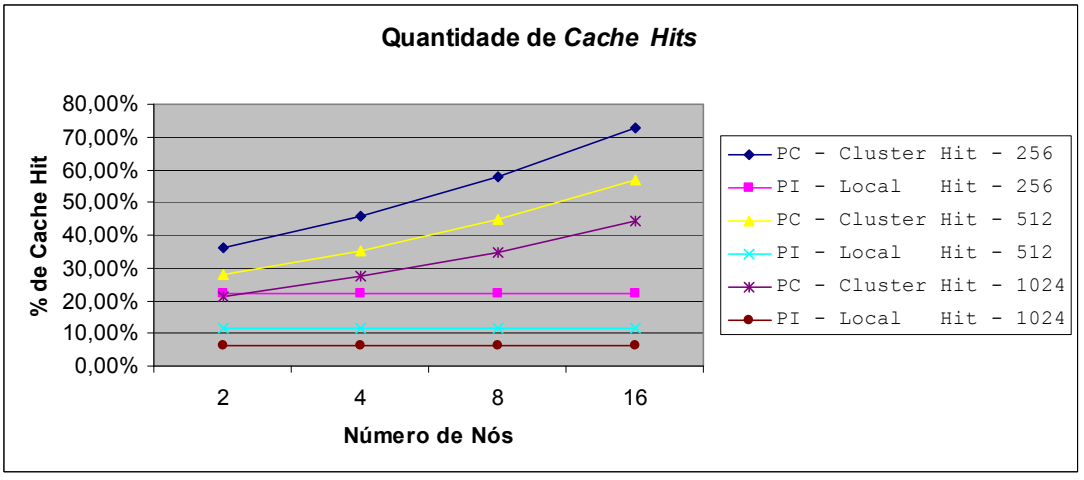

Figura 5.3: Quantidade de *Cluster Hit* e Local *Hit*.

A figura 5.4 apresenta a mesma situação descrita anteriormente, acrescida dos respectivos limites superior e inferior para ambas políticas. O limite superior para política individual de *cache* é calculado com base na distribuição *zipf-like*. De acordo com a distribuição, figura 3.4, 5% dos dados corresponde a 36.10% das

requisições. É possível observar que a política individual, mesmo armazenando 5% das páginas, conseguiu apenas 22.18% das requisições. Esse comportamento ocorre por dois motivos:

- Em [30] é descrito o problema *cold-start misses* ou *first-reference misses* que é a penalidade compulsória associada ao sistema de *cache* pelo fato deste iniciar vazia.
- A política de substituição LRU, mesmo apresentando um bom desempenho na média, não é a política mais indicada no contexto de páginas web [14].

Os dois problemas descritos são responsáveis pela diferença entre o valor esperado pela distribuição e o obtido através das simulações. Com objetivo de validar a explicação apresentada, uma quarta simulação foi realizada. Nesta, o sistema de *cache* é inicializado com 5% das páginas mais acessadas. Durante a simulação, nenhuma política de substituição é executada. Ou seja, o *cache* permanece com as mesmas páginas armazenadas durante toda a simulação, servindo a aplicação com apenas esse conjunto estático de páginas armazenadas. Nesse processo, apenas é feita a pontuação entre *local miss* e *local hit*. Ao final da simulação, foi verificado que os 5% das páginas foram responsáveis por 36.10% das requisições respondidas. Ou seja, a pre-inicialização do sistema de *cache* com 5% das páginas mais acessadas eliminou o problema de *cold-stat misses* e aproximou o resultado ao modelo teórico apresentado pela distribuição *zipf-like*.

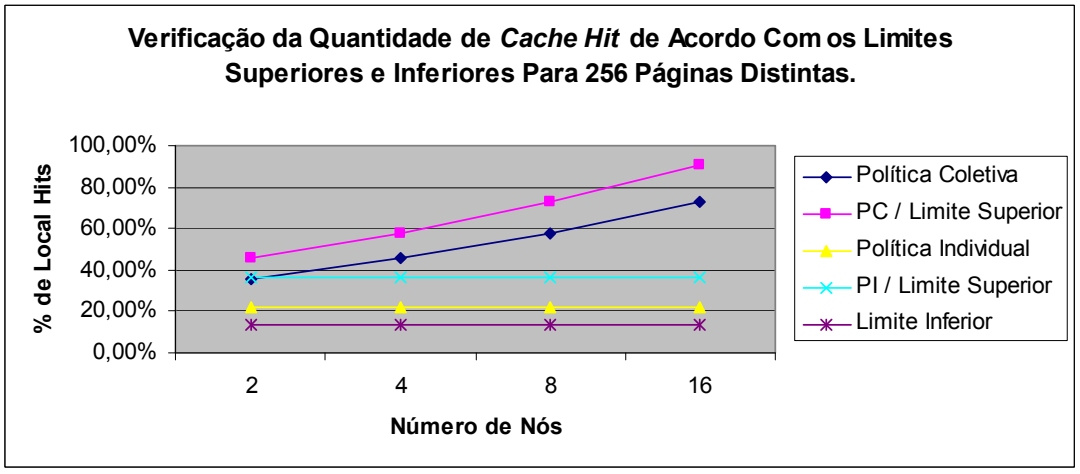

Figura 5.4: Verificação da Quantidade de *Cache Hit* de Acordo Com os Limites Superiores e Inferiores para 256 Páginas Distintas.

Por fim, o limite inferior foi calculado baseado nas características do sistema colaborativo de *cache*. Nesse sistema, cada nó reserva uma porcentagem do *cache* individual para o armazenar páginas globais e a outra porcentagem é dedicado ao armazenamento de páginas do interesse do nó. Ficou definido como limite inferior para política individual de *cache* como sendo a quantidade de requisições respondidas com o tamanho de cache correspondente a porcentagem destinada ao armazenamento de páginas de interesse individual do nó. No caso apresentado

pela figura 5.4, esse valor é de 13.43%. Por fim, o limite inferior da política colaborativa é considerado como sendo o valor obtido pela política individual de *cache*. Se o valor obtido com a política colaborativa for inferior ao valor obtido com a política individual, isso significa que a política coletiva não é necessária.

5.4 Verificação de Desempenho do Sistema de *Cache*

A quarta e a quinta bateria de teste tem como objetivo verificar o desempenho da política colaborativa proposta. Na figura 5.5 é possível observar a média de saltos necessária para responder a uma requisição em ambos sistemas: colaborativo e individual. De acordo com o gráfico, o melhor caso para política individual de *cache* (PI, 256 páginas) é praticamente pior que o pior caso para política colaborativa de *cache* (PC, 1024 páginas). Esse comportamento é justificado pois, após um *local miss*, o nó sempre submete a requisição ao servido, o que torna a penalização muito severa. Na abordagem colaborativa, essa penalidade é atenuada toda vez que ocorre um *cluster hit*.

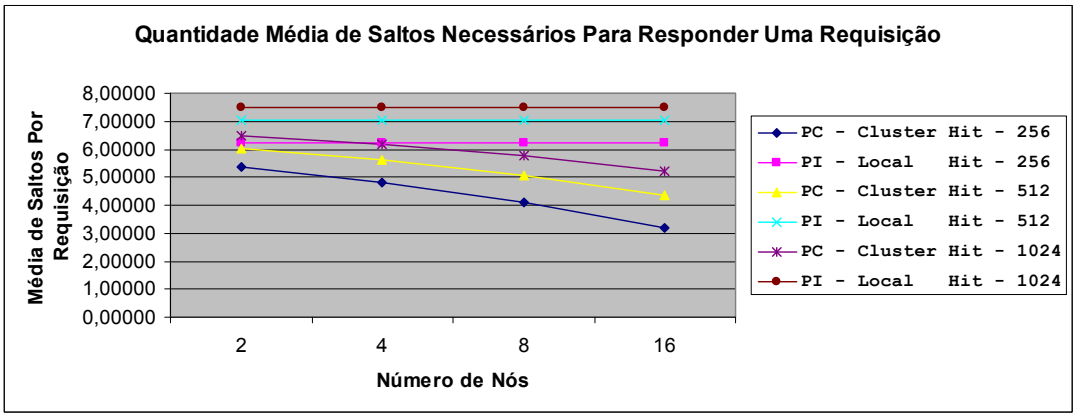

Figura 5.5: Quantidade Média de Saltos Necessários Para Responder Uma Requisição.

No modelo proposto, todos os membros de um *cluster* alcançam o *cluster head* em um salto. No melhor caso, um membro alcança outro em um salto e no pior caso em dois saltos através do *cluster head*. Considerando requisição e resposta, temos respectivamente: dois e quatro saltos. De acordo com a figura 5.5, no melhor caso para política colaborativa (256 páginas) a média de saltos por requisição é de 3,20 saltos/requisição para 16 nós e 4,11 saltos/requisição para 8 nós. O que sinaliza que, na média, a maioria das requisições estão sendo respondidas no interior do *cluster*. De fato esse valor é condizente. De acordo com os *logs* gerados na simulação, para 16 nós, 103.947 requisições (51.97%) foram respondidas dentro do *cluster* e 54.103 (27.05%) requisições foram enviadas ao servidor. Analisando o pior, caso temos: $Saltos_{piorcaso} = \frac{103.947*4+54.103*6}{200000} = 3.702$ requisições/saltos. Como nem sempre o pior caso ocorre, o valor 3,20 saltos/requisição capturados pelo sistema está condizente ao esperado.

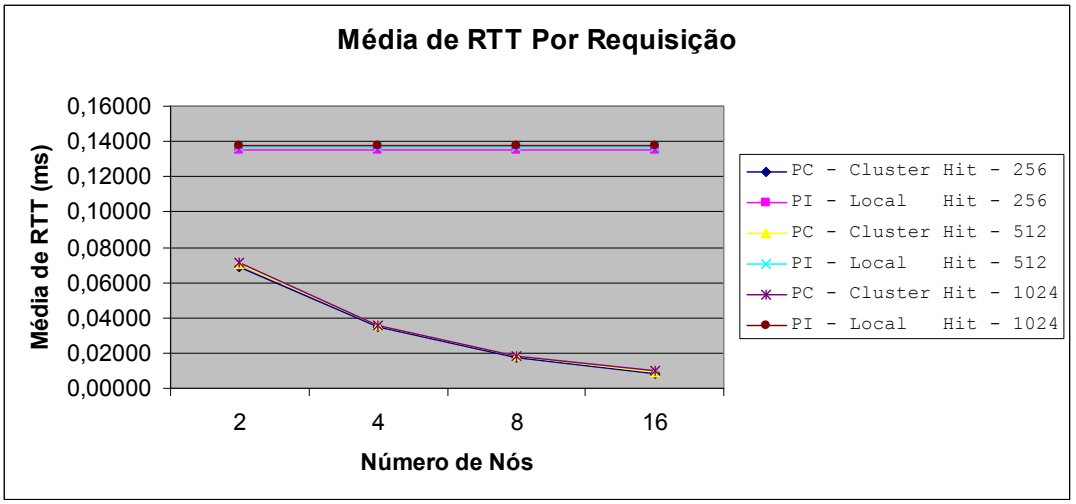

Figura 5.6: Média de RTT Por Requisição.

A figura 5.6 apresenta os resultados obtidos com a quinta bateria de testes. Essa bateria teve como objetivo verificar o tempo médio de resposta para uma requisição (*Roud Trip Time*). É possível observar que os tempos médios entre as simulações para 256, 512 e 1024 páginas distintas foram muito próximos. O maior número de páginas distintas reflete no aumento do universo de páginas possíveis de serem requisitadas, o que força uma maior rotatividade dos dados armazenados em *cache*. Esse comportamento aumenta o número de *cache local miss* e *cache global miss*, o que força a obtenção da informação em um número maior de saltos. Como a distância, entre nós requisitantes, até o servidor de páginas da rede não é significativa, para topologia utilizada, o aumento no número de páginas distintas não reflete na alteração no tempo RTT.

5.5 Verificação do Funcionamento do Sistema Colaborativo de *Cache*

A sexta e sétima bateria de testes, figuras 5.7 e 5.8, tem como objetivo verificar a proporção de *local hit* e *cluster hit* na composição de um *global hit*. Ou seja, da quantidade total de requisições respondidas pelo *cluster* será verificado a porcentagem de páginas respondidas localmente pelo cliente e a porcentagem de páginas que foi efetivamente respondida pela área global de *cache*.

De acordo com a figura 5.7, é possível observar que a medida que o tamanho do *cluster* aumenta, a porcentagem de páginas respondidas pelo *cache global* (*cluster hit*) também aumenta. Esse comportamento condiz com o esperado. No melhor caso, para 256 páginas e 16 nós, obteve-se 72.95% de *global hit*. Desses, 20.98% eram *local hit* e 51.97% eram *cluster hit*. Comparando esses valores com os apresentado na figura 5.7, para 1024 páginas e 16 nós, observa-se 44.25% de *global hit*, sendo desses, 6.59% eram *local hit* e 44.25% eram *cluster hit*. Dois pontos devem ser analisados a partir dos valores apresentados:

- conforme o número de páginas distintas aumenta, a porcentagem de global *global hits* (política colaborativa) e *local hit* (política individual) diminui (figuras 5.7, 5.8 e 5.3);

- conforme o tamanho do *cluster* aumenta (número de nós) a quantidade de *local hit* (política colaborativa) diminui (figuras 5.7, 5.8).

O primeiro ponto levantado é um comportamento esperado. Se o número de páginas distintas acessadas aumenta e o tamanho *cache* permanece constante, é esperado que a quantidade de *hits* diminua, já que a rotatividade de páginas em *cache* será maior. O segundo ponto está relacionado com o modelo proposto. No *cache* de cada cliente, existe uma área reservada para política global de *cache*. Esta área é gerenciada pelo *cluster head* e contém páginas de interesse do *cluster*. Conforme o tamanho do *cluster* aumenta, também aumenta a probabilidade de um determinado nó armazenar em seu *cache* (área global) uma página que não seja, necessariamente, do seu interesse. Esse problema está relacionado com a política de distribuição de páginas implementada pelo *cluster head*. Uma forma de otimizar o sistema é, baseado no limite máximo de *local hit* possível, buscar melhorias no algoritmo de distribuição de páginas e no algoritmo de substituição implementado pelos nós (LRU).

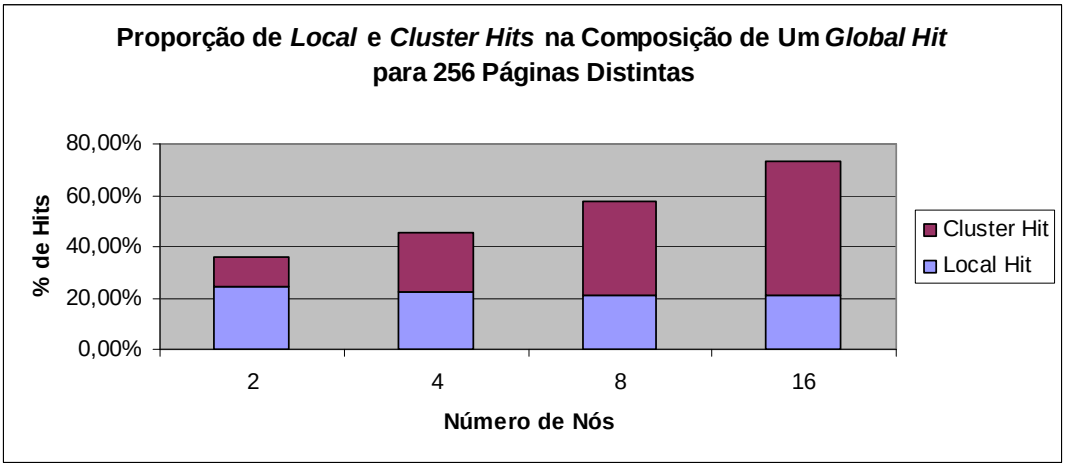

Figura 5.7: Proporção de *Local* e *Cluster Hit* na Composição de *Global Hit* para 256 páginas distintas.

Neste capítulo foram apresentados os primeiros resultados obtidos a partir de sete bateria de testes. Os objetivos dessas baterias eram a demonstração do funcionamento do sistema e a avaliação dos resultados obtidos. Os seguintes pontos foram verificados: carga no servidor, eficiência do sistema de *cache*, desempenho do sistema de *cache* e funcionamento do sistema colaborativo de *cache*. Por fim, foram apresentadas explicações quanto ao comportamento dos resultados e identificados alguns pontos do sistema que devem ser melhorados. Esses últimos, serão melhor apresentados na seção 6.3.

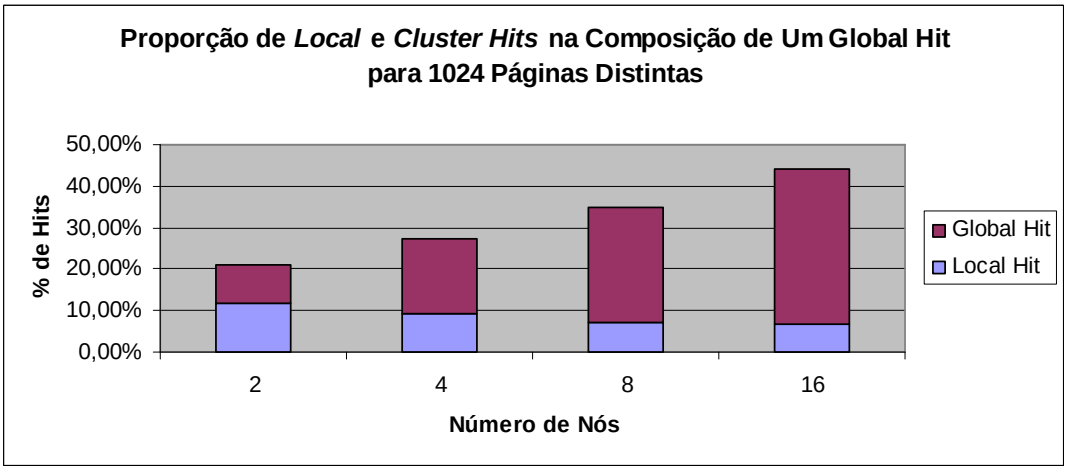

Figura 5.8: Proporção de *Local* e *Cluster Hit* na Composição de *Global Hit* para 1024 páginas distintas.

Capítulo 6

Considerações Finais

Neste capítulo serão apresentadas as considerações finais do trabalho, assim como as limitações e dificuldades encontradas. Por fim, serão apresentados a relação de trabalhos futuros propostos.

6.1 Conclusões

Nesta dissertação foi abordado o problema relacionado a quantidade de mensagens submetidas a rede após um *local miss*. Foi discutido que quanto maior o número de saltos necessário para obter uma resposta, maior será as chances de ocorrerem colisões e maior será a degradação da rede. Conforme apresentado por Gupta et al [28], a quantidade de nós está relacionada diretamente com a capacidade máxima de vazão da rede. Tendo como objetivo de diminuir o número de saltos durante o processo de recuperação de informação da rede, este trabalho propôs um modelo colaborativo de *cache*.

Assim como a maioria dos trabalhos referenciados na bibliografia apresentada, esta proposta utiliza o conceito de *cluster* para o agrupamento de nós. Restrições quanto ao modelo foram apresentadas na seção 4.1. Contudo, diferente dos trabalhos propostos, este trabalho propõe o uso de uma área de *cache* global destinada ao armazenamento de páginas de interesse de um determinado grupo de nós. Estes nós fazem parte do *cluster* e são denominados *cluster nodes*. Conforme apresentado, esta área global de *cache* é formada a partir da reserva de espaço em cada *cluster node*. O gerenciamento das páginas armazenada nessa região global é feita pelo *cluster head*.

Uma implementação foi feita utilizando o simulador de redes *ad hoc GloMoSim*. Conforme mencionado, diferente das implementações convencionais em simulador, esta implementação apresenta características reais de um ambiente real. Um *middleware* de comunicação foi implementado para criar uma dependência fraca entre o modelo implementado e o simulador *GloMoSim*. Para comparação com o sistema proposto, uma política individual e convencional de *cache* também foi implementada.

Foram executadas 8 baterias de testes como forma de validação da proposta. Cada bateria foi executada 6 vezes e os resultados finais foram obtidos a partir da média. Os seguintes aspectos foram avaliados: carga no servidor, eficiência do

sistema de *cache*, desempenho do sistema de *cache* e funcionamento do sistema colaborativo. Os principais resultados obtidos foram: para um *cluster* contendo 16 nós, obteu-se uma redução de aproximadamente 73% no número de mensagens enviadas ao servidor, contra 22% obtidos com a política convencional de *cache*; quando analisado o tempo médio de resposta a uma requisição, o modelo colaborativo obteve um tempo médio de 16 vezes menor que o tempo médio da política convencional.

6.2 Dificuldades Encontradas

As principais dificuldades enfrentadas durante a implementação do trabalho, estão relacionadas com o ambiente de simulação utilizado (*GloMoSim* [50]). Dentre elas, podemos citar:

- O simulador *GloMoSim* utiliza o compilador (código fechado) *PARSEC* [4] para gerar o arquivo executável de simulação. Como o *PARSEC* não aceita importar bibliotecas como a *GLIB* [45], todas as estruturas de dados utilizadas no código foram obrigatoriamente implementadas;

- Como parte do código do *GloMoSim* utiliza funções específicas do *PARSEC*, e este está disponível apenas através de bibliotecas binárias, não foi possível a utilização de ferramentas como *GDB* [25] para debugar o código. O que acarretou em um esforço extra na para debugar o código em busca de erros;

- Alguns *bugs* do *GloMoSim* foram identificados. Como o código do simulador é muito extenso, a localização de erros do tipo *segmentation fault* atrasaram a implementação em semanas. Erros lógicos do simulador foram os priores enfrentado. Entretanto, é importante ressaltar que todos os erros encontrados foram corridos;

- Outro problema foi a falta de documentação específica para questões avançadas de implementação do *GloMoSim*. Foram gastos alguns meses de trabalho para se entender as partes mais avançadas do código.

6.3 Trabalhos Futuros

A partir deste trabalho, foi possível identificar os seguintes pontos a serem melhorados:

- otimização do algoritmo de distribuição de páginas globais executado pelo *cluster head*. O objetivo é aumentar a proporção de *local hits* e diminuir a de *cluster hit* de forma a melhorar o desempenho do sistema;

- Comparação do modelo proposto com outras políticas colaborativas de *cache* [22, 46, 49];

- verificar o comportamento do sistema quando o número de saltos até o *cluster head* aumenta. Verificar a relação entre a quantidade de saltos máxima até o *cluster head* e a distância mínima que o *cluster head* precisa estar do servidor central;
- realizar a avaliação do modelo proposto sobre o ponto de vista das cópias disponíveis na rede durante a simulação;
- verificar o desempenho do sistema quando a variação no tempo TTL (*Time to Live*) de cada página;

Apêndice A

Arquivo de configuração do *GloMoSim*: config.in

```
# ***** GloMoSim Configuration File *****

# Glomosim is COPYRIGHTED software.  It is freely available without fee for
# education, or research, or to non-profit agencies. No cost evaluation
# licenses are available for commercial users. By obtaining copies of this
# and other files that comprise GloMoSim, you, the Licensee, agree to abide
# by the following conditions and understandings with respect to the
# copyrighted software:
#
# 1.Permission to use, copy, and modify this software and its documentation
#   for education, research, and non-profit purposes is hereby granted to
#   Licensee, provided that the copyright notice, the original author's names
#   and unit identification, and this permission notice appear on all such
#   copies, and that no charge be made for such copies. Any entity desiring
#   permission to incorporate this software into commercial products or to use
#   it for commercial purposes should contact:
#
#   Professor Rajive Bagrodia
#   University of California, Los Angeles
#   Department of Computer Science
#   Box 951596
#   3532 Boelter Hall
#   Los Angeles, CA 90095-1596
#   rajive@cs.ucla.edu
#
# 2.NO REPRESENTATIONS ARE MADE ABOUT THE SUITABILITY OF THE SOFTWARE FOR ANY
#   PURPOSE. IT IS PROVIDED "AS IS" WITHOUT EXPRESS OR IMPLIED WARRANTY.
#
# 3.Neither the software developers, the Parallel Computing Lab, UCLA, or any
#   affiliate of the UC system shall be liable for any damages suffered by
#   Licensee from the use of this software.
#
# $Id: config.in,v 1.32 2001/04/12 18:35:00 jmartin Exp $
#
# Anything following a "#" is treated as a comment.
#
###############################################################################
#
# The folowing parameter represents the maximum simulation time. The numberd
# portion can be followed by optional letters to modify the simulation time.
# For example:
#       100NS   - 100 nano-seconds
#       100MS   - 100 milli-seconds
#       100S    - 100 seconds
#       100     - 100 seconds (default case)
#       100M    - 100 minutes
#       100H    - 100 hours
```

```
#        100D    - 100 days
#

SIMULATION-TIME     24H

#
# The following is a random number seed used to initialize  part of the seed of
# various randomly generated numbers in the simulation. This can be used to vary
# the seed of the simulation to see the consistency of the results of the
# simulation.
#

SEED                1

#
# The following two parameters stand for the physical terrain in which the nodes
# are being simulated. For example, the following represents an area of size 100
# meters by 100 meters. All range parameters are in terms of meters.
#
# Terrain Area we are simulating.
#

TERRAIN-DIMENSIONS  (2000, 2000)

#
# The following parameter represents the number of nodes being simulated.
#

NUMBER-OF-NODES    12

#
#
#The following parameter represents the node placement strategy.
#- RANDOM: Nodes are placed randomly within the physical terrain.
#- UNIFORM: Based on the number of nodes in the simulation, the physical
#  terrain is divided into a number of cells. Within each cell, a node is
#  placed randomly.
#- GRID: Node placement starts at (0, 0) and are placed in grid format with
#  each node GRID-UNIT away from its neighbors. The number of nodes has to be
#  square of an integer.
#- FILE: Position of nodes is read from NODE-PLACEMENT-FILE. On each line of
#  the file, the x and y position of a single node is separated by a space.
#

NODE-PLACEMENT      FILE
NODE-PLACEMENT-FILE ./nodes.input
# NODE-PLACEMENT      GRID
# GRID-UNIT           30
# NODE-PLACEMENT      RANDOM
#NODE-PLACEMENT      UNIFORM

#
# The following represent parameters for mobility. If MOBILITY is set to NO,
# than there is no movement of nodes in the model. For the RANDOM-DRUNKEN model,
# if a node is currently at position (x, y), it can possibly move to (x-1, y),
# (x+1, y), (x, y-1), and (x, y+1); as long as the new position is within the
# physical terrain. For random waypoint, a node randomly selects a destination
# from the physical terrain. It moves in the direction of the destination in
# a speed uniformly chosen between MOBILITY-WP-MIN-SPEED and
# MOBILITY-WP-MAX-SPEED (meter/sec). After it reaches its
# destination, the node stays there for MOBILITY-WP-PAUSE time period.
# The MOBILITY-INTERVAL is used in some models that a node updates its position
# every MOBILITY-INTERVAL time period. The MOBILITY-D-UPDATE is used that a node
# updates its position based on the distance (in meters).
#

MOBILITY  NONE

# Random Waypoint and its required parameters.
```

```
#MOBILITY RANDOM-WAYPOINT
#MOBILITY-WP-PAUSE       30S
#MOBILITY-WP-MIN-SPEED   0
#MOBILITY-WP-MAX-SPEED   10

#MOBILITY TRACE
#MOBILITY-TRACE-FILE ./mobility.in

#MOBILITY PATHLOSS-MATRIX

# The following parameters are necessary for all the mobility models

MOBILITY-POSITION-GRANULARITY 0.5

#########################################################################
#
# PROPAGATION-LIMIT:
#   Signals with powers below PROPAGATION-LIMIT (in dBm)
#   are not delivered. This value must be smaller than
#   RADIO-RX-SENSITIVITY + RADIO-ANTENNA-GAIN of any node
#   in the model. Otherwise, simulation results may be
#   incorrect. Lower value should make the simulation more
#   precise, but it also make the execution time longer.
#
PROPAGATION-LIMIT      -111.0

#
# PROPAGATION-PATHLOSS: pathloss model
#   FREE-SPACE:
#     Friss free space model.
#     (path loss exponent, sigma) = (2.0, 0.0)
#   TWO-RAY:
#     Two ray model. It uses free space path loss
#     (2.0, 0.0) for near sight and plane earth
#     path loss (4.0, 0.0) for far sight. The antenna
#     height is hard-coded in the model (1.5m).
#   PATHLOSS-MATRIX:
#
#PROPAGATION-PATHLOSS   FREE-SPACE
PROPAGATION-PATHLOSS   TWO-RAY
#PROPAGATION-PATHLOSS   PATHLOSS-MATRIX

#
# NOISE-FIGURE: noise figure
#
NOISE-FIGURE    10.0

#
# TEMPARATURE: temparature of the environment (in K)
#
TEMPARATURE     290.0


########################################
#
# RADIO-TYPE: radio model to transmit and receive packets
#   RADIO-ACCNOISE: standard radio model
#   RADIO-NONOISE: abstract radio model
#   (RADIO-NONOISE is compatible with the current version (2.1b5)
#    of ns-2 radio model)
#
RADIO-TYPE              RADIO-ACCNOISE
#RADIO-TYPE              RADIO-NONOISE

#
# RADIO-FREQUENCY: frequency (in heltz)  (Identifying variable for multiple
#  radios)
#
RADIO-FREQUENCY     2.4e9
```

```
#
# RADIO-BANDWIDTH: bandwidth (in bits per second)
#
RADIO-BANDWIDTH     2000000

#
# RADIO-RX-TYPE: packet reception model
#   SNR-BOUNDED:
#     If the Signal to Noise Ratio (SNR) is more than
#     RADIO-RX-SNR-THRESHOLD (in dB), it receives the signal
#     without error. Otherwise the packet is dropped.
#     RADIO-RX-SNR-THRESHOLD needs to be specified.
#   BER-BASED:
#     It looks up Bit Error Rate (BER) in the SNR - BER table
#     specified by BER-TABLE-FILE.
#
RADIO-RX-TYPE           SNR-BOUNDED
RADIO-RX-SNR-THRESHOLD  10.0
#RADIO-RX-SNR-THRESHOLD  8.49583

#RADIO-RX-TYPE           BER-BASED
#BER-TABLE-FILE          ./ber_bpsk.in

#
# RADIO-TX-POWER: radio transmition power (in dBm)
#
#RADIO-TX-POWER        15.0
RADIO-TX-POWER        1.1

#
# RADIO-ANTENNA-GAIN: antenna gain (in dB)
#
RADIO-ANTENNA-GAIN  0.0

#
# RADIO-RX-SENSITIVITY: sensitivity of the radio (in dBm)
#
RADIO-RX-SENSITIVITY -91.0

#
# RADIO-RX-THRESHOLD: Minimum power for received packet (in dBm)
#
#RADIO-RX-THRESHOLD -81.0
RADIO-RX-THRESHOLD -79.0
#
###############################
#

MAC-PROTOCOL        802.11
#MAC-PROTOCOL        CSMA
#MAC-PROTOCOL        MACA

#MAC-PROTOCOL        TSMA
#TSMA-MAX-NODE-DEGREE      8

#MAC-PROPAGATION-DELAY 1000NS

#
# PROMISCUOUS-MODE defaults to YES and is necessary if nodes want
# to overhear packets destined to the neighboring node.
# Currently this option needs to be set to YES only for DSR is selected
# as routing protocol.  Setting it to "NO" may save a trivial amount
# of time for other protocols.

#PROMISCUOUS-MODE     NO

###############################
#
# Currently the only choice.
```

```
NETWORK-PROTOCOL    IP
NETWORK-OUTPUT-QUEUE-SIZE-PER-PRIORITY 100

#RED-MIN-QUEUE-THRESHOLD 150
#RED-MAX-QUEUE-THRESHOLD 200
#RED-MAX-MARKING-PROBABILITY 0.1
#RED-QUEUE-WEIGHT .0001
#RED-TYPICAL-PACKET-TRANSMISSION-TIME 64000NS


##############################
#


#ROUTING-PROTOCOL    BELLMANFORD
ROUTING-PROTOCOL    AODV
#ROUTING-PROTOCOL    DSR
#ROUTING-PROTOCOL    LAR1
#ROUTING-PROTOCOL    WRP
#ROUTING-PROTOCOL    FISHEYE

#ROUTING-PROTOCOL    ZRP
#ZONE-RADIUS         2

#ROUTING-PROTOCOL    STATIC
#STATIC-ROUTE-FILE   ROUTES.IN


#
# The following is used to setup applications such as FTP and Telnet.
# The file will need to contain parameters that will be use to
# determine connections and other characteristics of the particular
# application.
#


APP-CONFIG-FILE   ./app.conf


#
# The following parameters determine if you are interested in the statistics of
# a a single or multiple layer. By specifying the following parameters as YES,
# the simulation will provide you with statistics for that particular layer. All
# the statistics are compiled together into a file called "GLOMO.STAT" that is
# produced at the end of the simulation. If you need the statistics for a
# particular node or particular protocol, it is easy to do the filtering. Every
# single line in the file is of the following format:
# Node:      9, Layer:  RadioNoCapture, Total number of collisions is 0
#

APPLICATION-STATISTICS          YES
TCP-STATISTICS                  NO
UDP-STATISTICS                  YES
ROUTING-STATISTICS              YES
NETWORK-LAYER-STATISTICS        YES
MAC-LAYER-STATISTICS            YES
RADIO-LAYER-STATISTICS          YES
CHANNEL-LAYER-STATISTICS        YES
MOBILITY-STATISTICS             NO

#
#
# GUI-OPTION: YES allows GloMoSim to communicate with the Java Gui Vis Tool
#             NO does not

GUI-OPTION   NO
GUI-RADIO    NO
GUI-ROUTING  NO
```

Apêndice B

Arquivo de configuração do *GloMoSim*: nodes.input

```
#(R2CAM) Glomosim format file.
0  0 (432, 79, 0.0)
1  0 (244, 222, 0.0)
2  0 (336, 222, 0.0)
3  0 (168, 273, 0.0)
4  0 (168, 220, 0.0)
5  0 (335, 312, 0.0)
6  0 (422, 260, 0.0)
7  0 (400, 162, 0.0)
8  0 (295, 148, 0.0)
9  0 (346, 96, 0.0)
10  0 (204, 201, 0.0)
11  0 (230, 279, 0.0)
```

Apêndice C

Arquivo de configuração do *GloMoSim*: app.conf

```
# The traffic generators currently available are FTP,
# FTP/GENERIC, TELNET, CBR, and HTTP.
#
# ------------------------------------------------------------
# 1. FTP
#
# FTP uses tcplib to simulate the file transfer protocol.  In order to use
# FTP, the following format is needed:
#
#     FTP <src> <dest> <items to send> <start time>
#
# where
#
#     <src> is the client node.
#     <dest> is the server node.
#     <items to send> is how many application layer items to send.
#     <start time> is when to start FTP during the simulation.
#
# If <items to send> is set to 0, FTP will use tcplib to randomly determine
# the amount of application layer items to send.  The size of each item is
# will always be randomly determined by tcplib.  Note that the term "item"
# in the application layer is eqivalent to the term "packet" at the network
# layer and "frame" at the MAC layer.
#
#
# EXAMPLE:
#
#     a) FTP 0 1 10 0S
#
#        Node 0 sends node 1 ten items at the start of the simulation,
#        with the size of each item randomly determined by tcplib.
#
#     b) FTP 0 1 0 100S
#
#        Node 0 sends node 1 the number of items randomly picked by tcplib
#        after 100 seconds into the simulation.  The size of each item is
#        also randomly determined by tcplib.
#
#
# ------------------------------------------------------------
# 2. FTP/GENERIC
#
# FTP/GENERIC does not use tcplib to simulate file transfer.  Instead,
# the client simply sends the data items to the server without the server
# sending any control information back to the client.  In order to use
# FTP/GENERIC, the following format is needed:
#
```

```
#     FTP/GENERIC <src> <dest> <items to send> <item size> <start time> <end time>
#
# where
#
#     <src> is the client node.
#     <dest> is the server node.
#     <items to send> is how many application layer items to send.
#     <item size> is size of each application layer item.
#     <start time> is when to start FTP/GENERIC during the simulation.
#     <end time> is when to terminate FTP/GENERIC during the simulation.
#
# If <items to send> is set to 0, FTP/GENERIC will run until the specified
# <end time> or until the end of the simuation, which ever comes first.
# If <end time> is set to 0, FTP/GENERIC will run until all <items to send>
# is transmitted or until the end of simulation, which ever comes first.
# If <items to send> and <end time> are both greater than 0, FTP/GENERIC will
# will run until either <items to send> is done, <end time> is reached, or
# the simulation ends, which ever comes first.
#
# EXAMPLE:
#
#     a) FTP/GENERIC 0 1 10 1460 0S 600S
#
#        Node 0 sends node 1 ten items of 1460B each at the start of the
#        simulation up to 600 seconds into the simulation.  If the ten
#        items are sent before 600 seconds elapsed, no other items are
#        sent.
#
#     b) FTP/GENERIC 0 1 10 1460 0S 0S
#
#        Node 0 sends node 1 ten items of 1460B each at the start of the
#        simulation until the end of the simulation.  If the ten
#        items are sent the simulation ends, no other items are
#        sent.
#
#     c) FTP/GENERIC 0 1 0 1460 0S 0S
#
#        Node 0 continuously sends node 1 items of 1460B each at the
#        start of the simulation until the end of the simulation.
#
#
# ---------------------------------------------------------------
# 3. TELNET
#
# TELNET uses tcplib to simulate the telnet protocol.  In order to use
# TELNET, the following format is needed:
#
#     TELNET <src> dest> <session duration> <start time>
#
# where
#
#     <src> is the client node.
#     <dest> is the server node.
#     <session duration> is how long the telnet session will last.
#     <start time> is when to start TELNET during the simulation.
#
# If <session duration> is set to 0, FTP will use tcplib to randomly determine
# how long the telnet session will last.  The interval between telnet items
# are determined by tcplib.
#
#
# EXAMPLE:
#
#     a) TELNET 0 1 100S 0S
#
#        Node 0 sends node 1 telnet traffic for a duration of 100 seconds at
#        the start of the simulation.
#
#     b) TELNET 0 1 0S 0S
#
```

```
#        Node 0 sends node 1 telnet traffic for a duration randomly
#        determined by tcplib at the start of the simulation.
#
#
# ------------------------------------------------------------
# 4. CBR
#
# CBR simulates a constant bit rate generator.  In order to use CBR, the
# following format is needed:
#
#     CBR <src> <dest> <items to send> <item size>
#         <interval> <start time> <end time>
#
# where
#
#     <src> is the client node.
#     <dest> is the server node.
#     <items to send> is how many application layer items to send.
#     <item size> is size of each application layer item.
#     <interval> is the interdeparture time between the application layer items.
#     <start time> is when to start CBR during the simulation.
#     <end time> is when to terminate CBR during the simulation.
#
# If <items to send> is set to 0, CBR will run until the specified
# <end time> or until the end of the simuation, which ever comes first.
# If <end time> is set to 0, CBR will run until all <items to send>
# is transmitted or until the end of simulation, which ever comes first.
# If <items to send> and <end time> are both greater than 0, CBR will
# will run until either <items to send> is done, <end time> is reached, or
# the simulation ends, which ever comes first.
#
# EXAMPLE:
#
#     a) CBR 0 1 10 1460 1S 0S 600S
#
#        Node 0 sends node 1 ten items of 1460B each at the start of the
#        simulation up to 600 seconds into the simulation.  The interdeparture
#        time for each item is 1 second.  If the ten items are sent before
#        600 seconds elapsed, no other items are sent.
#
#     b) CBR 0 1 0 1460 1S 0S 600S
#
#        Node 0 continuously sends node 1 items of 1460B each at the start of
#        the simulation up to 600 seconds into the simulation.
#        The interdeparture time for each item is 1 second.
#
#     c) CBR 0 1 0 1460 1S 0S 0S
#
#        Node 0 continuously sends node 1 items of 1460B each at the start of
#        the simulation up to the end of the simulation.
#        The interdeparture time for each item is 1 second.
#
#
# ------------------------------------------------------------
# 5. HTTP
#
# HTTP simulates single-TCP connection web servers and clients.  Bruce Mah
# has gathered packet traces of HTTP network conversations, and produced
# CDFs for "the size of HTTP items retrieved, number of items per 'Web page',
# think time, and user browsing behavior."
# (http://www.ca.sandia.gov/~bmah/Software/HttpModel/)
#
# This model has been implemented for GloMoSim, and the following format
# describes its use for servers:
#
#     HTTPD <address>
#
# where
#
#     <address> is the node address of a node which will be serving
```

```
#               Web pages.
#
# For HTTP clients, the  following format is used:
#
#     HTTP <address> <num_of_server> <server_1> ... <server_n> <start> <thresh>
#
# where
#
#     <address> is the node address of the node on which this client resides
#     <num_of_server> is the number of server addresses which will follow
#     <server_1>
#     .
#     .
#     <server_n> are the node addresses of the servers which this client
#                will choose between when requesting pages.  There must
#                be "HTTPD <address>" lines existing separately for each of
#                these addresses.
#     <start> is the start time for when the client will begin requesting
#             pages
#     <thresh> is a ceiling (specified in units of time) on the amount of
#              "think time" that will be allowed for a client.  The
#              network-trace based amount of time modulo this threshhold
#              is used to determine think time.
#
# EXAMPLE:
#
#     HTTPD 2
#     HTTPD 5
#     HTTPD 8
#     HTTPD 11
#     HTTP 1 3 2 5 11 10S 120S
#
#     There are HTTP servers on nodes 2, 5, 8, and 11.  There is an HTTP
#     client on node 1.  This client chooses between servers {2, 5, 11} only
#     when requesting web pages.  It begins browsing after 10S of simulation
#     time have passed, and will "think" (remain idle) for at most 2 minutes
#     of simulation time, at a time.

#CBR 0 1 10000 512   5S      70S 10800S


R2CAM_SERVER 0
#R2CAM_ZIPF D 0.8 4 <3,4,10,11> 256 400000
#R2CAM_ZIPF D 0.8 1 <3> 512 200000

R2CAM_ZIPF D 0.8 2 <3,4> 256 200000

#R2CAM_LRU_CLIENT 3 0 600
#R2CAM_LRU_CLIENT 4 0 600
#R2CAM_LRU_CLIENT 10 0 600
#R2CAM_LRU_CLIENT 11 0 600

#First Cluster
R2CAM_CLIENT 1   1 A 300 150
R2CAM_CLIENT 3   1 A 300 150
R2CAM_CLIENT 4   1 A 300 150
R2CAM_CLUSTERHEAD 1 0 A 2 <3,4> <150,150>

#R2CAM_CLIENT 10  1 A 600 300
#R2CAM_CLIENT 11  1 A 600 300
#R2CAM_CLUSTERHEAD 1 0 A 4 <3,4,10,11> <300,300,300,300>
```

Referências

[1] Optimized link state routing protocol (olsr). 2003.

[2] *Efficient Cooperative Caching in Ad Hoc Networks.*, 2006.

[3] Martin Arlitt, Ludmila Cherkasova, John Dilley, Rich Friedrich, and Tai Jin. Evaluating content management techniques for web proxy caches. *SIGMETRICS Perform. Eval. Rev.*, 27(4):3–11, 2000.

[4] Parallel Computing Laboratory at UCLA. Parsec: Parallel simulation environment for complex systems. Disponível em: <http://pcl.cs.ucla.edu/projects/parsec/>. Acessado em: 30/07/2008.

[5] Stefano Basagni, Ivan Stojmenovic, and Silvia Giordano. *Mobile Ad Hoc Networking*. Wiley-IEEE, 2004.

[6] Magnus E. Bjornsson and Liuba Shrira. Buddycache: high-performance object storage for collaborative strong-consistency applications in a wan. In *OOPSLA '02: Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 26–39, New York, NY, USA, 2002. ACM Press.

[7] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[8] Azzedine Boukerche. Performance evaluation of routing protocols for ad hoc wireless networks. *Mob. Netw. Appl.*, 9(4):333–342, 2004.

[9] Lee Breslan, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: evidence and implications. In *Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE INFOCOM.*, volume Volume 1, pages 126 – 134, March 1999.

[10] Guohong Cao, Liangzhong Yi, and Chita R Das. Cooperative cache-based data access in ad hoc networks. *Computer*, 37:32 – 39, 2004.

[11] Jiannong Cao, Yang Zhang, Guohong Cao, and Li Xie. Data consistency for cooperative caching in mobile environments. *Computer*, 40(4):60–66, 2007.

[12] Jiannong Cao, Yang Zhang, Li Xie, and Guohong Cao. Consistency of cooperative caching in mobile peer-to-peer systems over manet. pages 573–579, 2005.

[13] Pei Cao and Chengjie Liu. Maintaining strong cache consistency in the world wide web. *IEEE Trans. Comput.*, 47(4):445–457, 1998.

[14] Ludmila Cherkasova. Improving www proxies performance with greedy-dual-size-frequency caching policy. In hp technical report, Computer Systems Laboratory HP, Palo Alto, November 1998.

[15] Chi-Yin Chow, Hong Va Leong, and A A. Chan. Cache signatures for peer-to-peer cooperative caching in mobile environments. *18th International Conference on Advanced Information Networking and Applications*, 1:96–101, 2004.

[16] Chi-Yin Chow, Hong Va Leong, and Alvin T. S. Chan. Distributed group-based cooperative caching in a mobile broadcast environment. In *MDM '05: Proceedings of the 6th international conference on Mobile data management*, pages 97–106, New York, NY, USA, 2005. ACM Press.

[17] Chi-Yin Chow, Hong Va Leong, and Alvin T.S. Chan. Grococa: group-based peer-to-peer cooperative caching in mobile environment. *IEEE Journal on Selected Areas in Communications*, 25:179–191, January 2007.

[18] George F. Coulouris and Jean Dollimore. *Distributed systems: concepts and design.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, third edition, 2001.

[19] Michael Dahlin, Randolph Wang, Thomas E. Anderson, and David A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Operating Systems Design and Implementation*, pages 267–280, 1994.

[20] Mario Antônio Ribeiro Dantas. *Tecnologia de Redes de Comunicação e Computadores.* Axcel Books do Brasil Editora, 2002.

[21] Brian D Davision. A web caching primer. *IEEE Internet Computing*, 5:38–45, 2001.

[22] Yu Du and Sandeep K. S. Gupta. Coop - a cooperative caching service in manets. In *ICAS-ICNS 2005. Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services, 2005.*, pages 58–58. IEEE, October 2005.

[23] Yuguang Fang, Zygmunt J. Haas, Ben Liang, and Yi-Bing Lin. Ttl prediction schemes and the effects of inter-update time distribution on wireless data access. *Wirel. Netw.*, 10(5):607–619, 2004.

[24] IEEE 802.11 The Working Group for WLAN Standards. Ieee 802.11 wireless local area networks. IEEE 802.11 WIRELESS LOCAL AREA NETWORKS. Disponível em: <`http://ieee802.org/11/`>. Acessado em: 22/07/2008.

[25] Free Software Foundation. Gdb: The gnu project debugger. Disponível em: <`http://http://sourceware.org/gdb/`>. Acessado em: 30/07/2008.

[26] Free Software Fundation. The linux kernel archive. The Linux Kernel Archive. Disponível em: <http://kernel.org>. Acessado em: 22/07/2008.

[27] NSF grant (NCR-9796082). Squid: Optimising Web Delivery. Disponível em: <http://www.squid-cache.org>. Acessado em: 28/11/2007.

[28] P. Gupta and P.R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46:388–404, 2000.

[29] Takahiro Hara. Replica allocation methods in ad hoc networks with data update. *Mob. Netw. Appl.*, 8(4):343–354, 2003.

[30] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 2007.

[31] Yu Huang, Jiannong Cao, and Beihong Jin. A predictive approach to achieving consistency in cooperative caching in manet. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 50, New York, NY, USA, 2006. ACM Press.

[32] IEEE. Ieee standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. Technical report, Institute of Electrical and Electronics Engineers, 2007.

[33] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[34] Ramakrishna Karedla, J. Spencer Love, and Bradley G. Wherry. Caching strategies to improve disk system performance. *Computer*, 27(3):38–46, 1994.

[35] UCLA Parallel Computing Laboratory. Glomosim: Global mobile information systems simulation library. In *GloMoSim: Global Mobile Information Systems Simulation Library*. Disponível em: <http://pcl.cs.ucla.edu/projects/glomosim/>. Acessado em: 25/08/2008.

[36] Jinyang Li, Charles Blake, Douglas S.J. De Couto, Hu Imm Lee, and Robert Morris. Capacity of ad hoc wireless networks. pages 61–69, 2001.

[37] Wenzhong Li, Edward Chan, and Daoxu Chen. Energy-efficient cache replacement policies for cooperative caching in mobile ad hoc network. In *IEEE Wireless Communications and Networking Conference, WCNC.*, pages 3347 – 3352, March 2007.

[38] C.S.R. Murthy and B.S. Manoj. *Ad Hoc Wireless Networks: Architecture and Protocols.* Prentice Hall, 2004.

[39] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 151–162, New York, NY, USA, 1999. ACM.

[40] Charles E. Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. pages 234–244, 1994.

[41] Charles E. Perkins and Elizabeth M. Royer. Ad-hoc on-demand distance vector routing. *wmcsa*, 00:90, 1999.

[42] Stefan Podlipnig and Laszlo Boszormenyi. A survey of web cache replacement strategies. *ACM Comput. Surv.*, 35(4):374–398, 2003.

[43] E. Royer and C. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. 1999.

[44] Vineet Srivastava and Mehul Motani. Cross-layer design: a survey and the road ahead. *IEEE Communications Magazine*, 43:112 – 119, 2005.

[45] The Gnome Project Team. Glib reference manual. In *GNOME Documentation Library*. Disponível em: <`http://library.gnome.org/devel/glib/`>. Acessado em: 30/07/2008.

[46] Yi-Wei Ting and Yeim-Kuan Chang. A novel cooperative caching scheme for wireless ad hoc networks: Groupcaching. In IEEE, editor, *NAS 2007: International Conference on Networking, Architecture and Storage*, pages 62–68, july 2007.

[47] Uppsala University and University of Basel. Aodv-uu - ad-hoc on-demand distance vector routing. AODV-UU - Ad-hoc On-demand Distance Vector Routing. Disponível em: <`http://core.it.uu.se/core/index.php/AODV-UU`>. Acessado em: 22/07/2008.

[48] Bernhard H. Walke, Stefan Mangold, and Lars Berlemann. *IEEE 802 Wireless Systems: Protocols, Multi-Hop Mesh/Relaying, Performance and Spectrum Coexistence*. John Wiley & Sons, January 2007.

[49] Liangzhong Yin and Guohong Cao. Supporting cooperative caching in ad hoc networks. *IEEE Transactions on Mobile Computing*, 5:77– 89, 2006.

[50] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. In *PADS'98: Proceedings of the 12th Workshop on Parallel and Distributed Simulations*, May 1998.

[51] George Kingsley Zipf. Relative frequency as a determinant of phonetic change. *Harvard Studies in Classical Philology*, 40:1–95, 1929.