COPPE/UFRJ

SIMULADOR PARA UM MODELO DE BUSCA OPORTUNÍSTICA DE CONTEÚDOS EM REDES ORIENTADAS A INFORMAÇÃO

Diego Vargas Jannibelli

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

Orientador: Rosa Maria Meri Leão

Rio de Janeiro

Setembro de 2014

SIMULADOR PARA UM MODELO DE BUSCA OPORTUNÍSTICA DE CONTEÚDOS EM REDES ORIENTADAS A INFORMAÇÃO

Diego Vargas Jannibelli

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

Examinada por:

Prof. Rosa Maria Meri Leão, Dr.

Prof. José Ferreira de Rezende, Dr.

Prof. Daniel Sadoc Menasché, Ph.D.

Prof. Carlos Alberto Vieira Campos, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2014

Jannibelli, Diego Vargas

Simulador para um Modelo de Busca Oportunística de Conteúdos em Redes Orientadas a Informação/Diego Vargas Jannibelli. – Rio de Janeiro: UFRJ/COPPE, 2014.

XII, 72 p.: il.; 29, 7cm.

Orientador: Rosa Maria Meri Leão

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2014.

Referências Bibliográficas: p. 64 – 69.

1. Redes Orientadas a Informação. 2. Random Walking. 3. ICN. I. Leão, Rosa Maria Meri. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Dedico todo esse trabalho aos meus familiares e amigos.

Agradecimentos

Primeiramente agradeço a Profª Rosa pela forma com que orientou esse trabalho. Ela precisou de muita paciência e compreensão. Agradeço também ao Guilherme pelos conselhos fornecidos durante o desenvolvimento desse trabalho. Não teria terminado sem sua ajuda. Não posso esquecer aos alunos do LAND, como Gaspare, Jefferson e Raphael por terem cedido e configurado as máquinas do laboratório para as simulações. Eles têm grande participação no resultado final desse trabalho.

Agradeço a minha mãe por me dar força e carinho, além das refeições enquanto implementava o trabalho. Estou vivo porque ela me lembrava que era hora de comer. Ao meu irmão que sempre esteve ao meu lado e me alegrava quando as coisas não estavam dando certo. Ao meu pai que mesmo longe sempre meu deu força a continuar o meu caminho.

E por fim, aos meu amigos que mesmo ausente nos eventos sociais faziam questão de me lembrar que era por um bom motivo.

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

SIMULADOR PARA UM MODELO DE BUSCA OPORTUNÍSTICA DE CONTEÚDOS EM REDES ORIENTADAS A INFORMAÇÃO

Diego Vargas Jannibelli

Setembro/2014

Orientador: Rosa Maria Meri Leão

Programa: Engenharia de Sistemas e Computação

Redes orientadas a informação (ICN), uma nova abordagem em disseminação de informação, vêm ganhando atenção dos pesquisadores nos últimos anos por causa do aumento do volume de tráfego na Internet. Essa arquitetura é uma alternativa ao modelo tradicional da Internet, o TCP/IP, pois ao contrário do TCP/IP onde o roteamento é baseado no endereço do provedor do conteúdo, as redes orientadas a informação possuem como objeto principal de roteamento o nome da informação.

Várias propostas de arquitetura ICN vêm sendo elaboradas. A arquitetura estudada nesse trabalho é baseada em busca aleatória na rede para encontrar o conteúdo armazenado mais próximo ao usuário, diminuindo o tempo de resposta e a carga nos servidores no topo da hierarquia. O armazenamento do conteúdo na rede é baseado na sua popularidade. O objetivo desse trabalho é desenvolver um simulador para avaliar esta arquitetura. Duas métricas foram estimadas: o percentual de requisições por conteúdo que são atendidas pelos roteadores da rede, denominado carga filtrada, e o tempo para encontrar o conteúdo, denominado tempo de resposta. Nos cenários estudados usamos a topologia de Rede Nacional de Ensino e Pesquisa (RNP) e duas políticas de substituição dos conteúdos armazenados nos roteadores.

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

INFORMATION CENTRIC NETWORK SIMULATOR BASED ON AN OPPORTUNISTIC SEARCH MODEL

Diego Vargas Jannibelli

September/2014

Advisor: Rosa Maria Meri Leão

Department: Systems Engineering and Computer Science

Information centric networks (ICN), a new approach for content and information distribution networks, have gained significant attention from researchers in the last years because of the increased volume of traffic in the Internet. This architecture is an alternative to the traditional model of the Internet, TCP/IP, because unlike TCP/IP where routing is based on the address of the content provider, ICN routing is based on the name of the information.

Several proposals for ICN architectures have been developed. The architecture studied in this work is based on random walks to find content stored closest to the user, reducing the response time and load on servers at the top of the hierarchy. Content is stored on the network based on its popularity. The aim of this work is to develop a simulator to evaluate this architecture. Two metrics were estimated: the percentage of requests that are served by network routers, called filtered load, and the time to find the content called response time. In the scenarios studied we consider the topology of National Education and Research Network (RNP) and two replacement policies for the contents stored in the routers.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A Internet foi originalmente concebida para permitir a comunicação entre computadores remotos por intermédio da comutação de pacotes, onde a multiplexação estatística de pacotes é usada para permitir o compartilhamento dos recursos. Há cerca de 50 anos atrás a teoria de redes de filas [1] foi usada para analisar e mostrar as vantagens da tecnologia de *redes de comutação de pacotes*. Com o passar dos anos novas tecnologias surgiram, como por exemplo, a banda larga e a evolução dos dispositivos móveis, tornando a Internet um meio para conectar usuários a conteúdos. Hoje em dia os usuários estão interessados em conteúdos ou informação, seja um vídeo no YouTube ou um arquivo no BitTorrent. A disseminação de conteúdos provocou um crescimento exponencial no volume de tráfego da Internet. De acordo com *Cisco's Visual Networking* [2], em 2012, 64% de todo o tráfego mundial era relativo a visualização de vídeos. É previsto que em 2017, 51% de todo o tráfego da Internet será relativo a redes de distribuição de conteúdos. Desde então, os pesquisadores e engenheiros têm tido um crescente interesse na área de distribuição de conteúdos e começaram a projetar uma gama variada de redes orientadas a disseminação de conteúdos.

Uma das propostas é a de redes de distribuição de conteúdos CDN (*Content Delivery Networks*), como por exemplo, a Akamai [3]. Em CDNs, servidores-origem armazenam pelo menos uma réplica de todo conteúdo publicado. Além disso, existem servidores CNDs espalhados pela rede que replicam o conteúdo e contém um ponteiro para toda cópia na rede.

A infraestrutura de CDN está sendo levada ao limite de sua capacidade de serviço

pelo aumento significativo de tráfego de vídeo [4] e pelo fato do conteúdo ser gerado a taxas cada vez mais altas, por exemplo, em redes sociais. Para aliviar o problema, têm sido implementadas federações de CDNs que podem redirecionar uma solicitação de conteúdo para o servidor de armazenamento mais próximo do usuário, após a determinação do endereço do conteúdo [4]. Sistemas híbridos P2P-CDN também têm ampliado seu espaço na indústria recentemente. O problema de escalabilidade de infraestrutura CDN e suas variações tende a se agravar com o constante aumento de tráfego de vídeo e em cenários onde milhares de usuários de dispositivos móveis passam a requisitar conteúdo. A centralização do controle das CDNs tende a não escalar [5] com o aumento dramático da demanda de conteúdo. Por outro lado, os sistemas P2P são sistemas onde os usuários (os *peers*) agem como clientes e servidores ao mesmo tempo. Até muito recentemente esses sistemas eram considerados perfeitamente escaláveis. Entretanto, têm surgido trabalhos nos últimos anos que mostram que sistemas P2P também têm limitações de vazão e não escalam com a demanda dos usuários [6, 7, 8, 9].

A segunda proposta são as redes orientadas a informação ICN (*Information-Centric Networks*) que pressupõe que os elementos da rede sejam capazes de processar, rotear e armazenar os conteúdos. Nas ICNs, os conteúdos compõem as unidades de informação e podem ser armazenadas nos elementos de rede, sendo que os mesmos são identificados por seu nome ou atributos. As réplicas dos conteúdos espalham-se pela rede de acordo com a demanda formando uma rede de armazenamento. Esta nova concepção objetiva otimizar métricas de desempenho como disponibilidade da informação e tempo para recuperar a informação e ainda balancear os recursos de rede entre as requisições [10]. Em maiores detalhes, na concepção ICN, o conteúdo a ser recuperado por clientes têm um nome e um mantenedor, isto é, um local conhecido de onde o objeto pode ser recuperado. Uma vez gerada uma requisição por um conteúdo este flui em direção ao mantenedor através dos roteadores de conteúdo. Cada roteador de conteúdo tem uma *cache* que pode armazenar o conteúdo (ou parte dele) quando este é enviado do mantenedor ao solicitante. Uma outra requisição para o mesmo conteúdo poderá encontrá-lo em uma das *caches* dos roteadores e assim recuperá-lo sem necessidade de acessar o mantenedor. Note que esse cenário pode ser entendido como uma rede de *caches*.

A principal diferença entre as CDNs e as ICNs, é que em CDNs, servidores entregam os conteúdos na forma *tradicional* da Internet, ou seja, através de uma conexão entre o cliente o servidor, enquanto que em ICNs os pedidos de conteúdo são roteados e encaminhados baseados no nome ou em um atributo do conteúdo.

As ICNs ainda apresentam vários desafios sendo estudados pela comunidade científica, porém já existe um consenso de quais são as funções básicas que devem ser atendidas. A principal diferença entre o paradigma ICN e as demais propostas de redes de distribuição de conteúdo é o uso de identificação por nome do dado/conteúdo ao invés da localização geográfica, como por exemplo o endereço IP. Como mencionado, as requisições dos conteúdos são endereçadas pelo nome do conteúdo e os componentes da rede usam esta identificação para rotear as requisições até os computadores que possuem uma cópia autorizada de tal pedido.

Outro fator que diferencia as ICNs são o uso de autenticação e proteção do conteúdo. Um dos maiores esforços da Internet é a autenticação de receptores e transmissores, bem como manter a segurança na conexão. Em ICNs os conteúdos são autenticados para evitar alteração e *espionagem*, por isso apenas cópias autorizadas podem circular na ICN. Roteadores ICN são equipados com memórias caches que permitem o armazenamento de pedaços de conteúdos para serem retransmitidos, diminuindo a carga nos provedores e o tempo de resposta aos clientes. O roteador ICN apenas encaminha o pedido do conteúdo, caso o conteúdo não esteja armazenado em sua memória cache. Uma outra funcionalidade de ICNs é o uso do modelo de *publish/subscribe* [11], onde o usuário gera requisições para um determinado conteúdo e a rede provê ao usuário tal conteúdo quando disponível.

O paradigma ICN tem sido considerado como candidato promissor para uma arquitetura da Internet do futuro [10, 12]. Inúmeros desafios entretanto ainda tem que ser investigados para que os novos conceitos possam ser colocados em prática de uma forma eficiente.

O objetivo deste trabalho é desenvolver um simulador para redes orientadas a informação baseado na arquitetura proposta em [13, 14]. Em [13, 14] foi proposto um modelo analítico para calcular métricas de desempenho da arquitetura proposta. O modelo é baseado em algumas hipóteses, como por exemplo, um determinado tipo de topologia e que a cache dos roteadores é infinita. Usamos o simulador com o

objetivo de avaliar uma topologia real. Para tal, consideramos a topologia da Rede Nacional de Ensino e Pesquisa (RNP), e analisamos o caso da cache do roteador ser finita, considerando duas políticas de substituição dos conteúdos da cache.

O trabalho está estruturado da seguinte forma. No capítulo 2 iremos apresentar os principais conceitos das redes orientadas a informação e resumir alguns dos trabalhos realizados na área de ICNs. Além disso, descreveremos o modelo de busca oportunística, no qual este trabalho se baseia. No capítulo 3 iremos descrever como o simulador foi desenvolvido, assim como sua arquitetura e validação. Os resultados obtidos a partir dos experimentos realizados serão apresentados no capítulo 4. Por fim, no capítulo 5 iremos realizar as considerações finais e conclusões.

Capítulo 2

Redes Orientadas a Informação - ICN

Como descrito no capítulo anterior, a crescente busca por distribuição de conteúdos e informações pela Internet motivou pesquisadores a projetar arquiteturas baseadas em objetos de dados denominadas redes orientadas a informação ICN (*Information-Centric Networks*). Hoje, existem várias iniciativas focadas em desenvolver esse tipo de arquitetura para que no futuro, esta nova arquitetura possa provavelmente vir a substituir o modelo atual da Internet.

Nesse capítulo, iremos definir esse novo modelo de disseminação de informação e descrever algumas iniciativas tais como, DONA (*Data-Oriented Network Architecture*) um projeto da universidade de Berkeley [15], PSIRP ou PURSUIT (*Publish-Subscribe Internet Routing Paradigm ou Publish-Subscribe Internet Tecnology*) [16], CCN ou NDN (*Content Centric Networking ou Named Data Networking*) [17] e NetInf (*Networking Information*) [18].

Além disso, ao final do capítulo, iremos descrever a arquitetura proposta para redes orientadas a informação [13, 14] investigada neste trabalho. A arquitetura é oriunda de uma tese de doutorado em curso atualmente no laboratório LAND/PESC. Essa arquitetura é estruturada em domínios, onde cada domínio é subdividido em subdomínios e, por sua vez, cada domínio contém um conjunto de roteadores com capacidade de armazenamento de conteúdos requisitados pelos usuários, ou seja, possui uma *cache* local.

2.1 Conceito

A principal diferença entre ICN e as demais propostas em redes IP para disseminação de conteúdos é o uso de identificação por nome do dado/conteúdo ao invés da localização geográfica (endereço IP). Como mencionado, as requisições dos conteúdos são endereçadas pelo nome do conteúdo e os componentes da rede usam esta identificação para rotear as requisições até os computadores que possuem uma cópia autorizada de tal pedido.

Outro fator que diferencia as redes orientadas a informação são o uso de autenticação e proteção do conteúdo. Um dos maiores esforços da Internet é a autenticação de receptores e transmissores, bem como manter a segurança nas conexões. Em ICNs os conteúdos são autenticados para evitar alteração e *espionagem*, por isso apenas cópias autorizadas podem circular na rede.

Os roteadores das ICNs são equipados com memórias caches que permitem o armazenamento de pedaços de conteúdos para serem retransmitidos, diminuindo assim a carga nos provedores e o tempo de resposta aos clientes. O roteador ICN apenas encaminha o pedido do conteúdo, caso o conteúdo não esteja armazenado em sua memória cache.

2.2 Principais componentes de ICNs

Detalhes de ICNs ainda são discutidos pela comunidade de pesquisa, porém já existe um consenso de quais são as funções básicas que devem ser atendidas. Essas funcionalidades serão descritas abaixo.

Nomeação ou Identificação: Como mencionado acima, a informação é o principal objeto de ICNs, por isso a identificação desses objetos que são transmitidos através da rede é de suma importância para que o modelo funcione. A forma de nomeação/identificação varia de acordo com a arquitetura, podendo ser uma nomeação hierárquica, similar às URLs utilizadas na Internet, ou uma nomeação plana (*flat*).

Resolução de Nomes e Roteamento: Resolução de nomes é o ato de correlacionar o nome da informação com o servidor fonte da informação. Enquanto o roteamento é o ato de gerar um caminho mais próximo ao usuário para transmitir a informação do servidor até o usuário. A questão-chave é se essas duas funcionalidades são dependentes, ou seja, estão acopladas ou se são independentes. Dizer que são dependentes, significa que a requisição da informação é roteada até o servidor fonte que subsequentemente envia pelo caminho reverso a informação requerida. Já no caso de serem independentes, a resolução de nomes não determina ou restringe o caminho que o dado irá percorrer do servidor até o usuário.

Armazenamento na Rede - *Caching* Existem duas abordagem para o armazenamento na rede, uma é o armazenamento no caminho das requisições dos conteúdos (*on-path*) e outra é fora do caminho das requisições (*off-path*). No primeiro caso, a mensagem com o conteúdo percorre o caminho reverso daquele percorrido pelas requisições, até ser entregue ao usuário. Neste caminho, o conteúdo pode ou não ser armazenado nos roteadores. No segundo caso, o conteudo é entregue ao usuário através de um caminho diferente daquele percorrido pelas requisições, por exemplo, usando o protocolo IP. Da mesma forma, o conteúdo pode ou não ser armazenado nos roteadores do caminho percorrido até o usuário.

2.3 Modelos na Literatura

2.3.1 Data-Oriented Network Architecture - DONA

É um projeto da universidade de Berkeley e foi uma das primeiras arquiteturas ICN completas [15], à medida que altera radicalmente a atribuição de nomes trocando a nomeação hierárquica pela nomeação plana (*flat*). Ao contrário das URLs que fazem uso do DNS e portanto dependem da localização geográfica da informação, os nomes *flats* na arquitetura DONA podem permanecer iguais mesmo que as informações se movam. Com isso, as informações podem ser armazenadas e replicadas na camada de rede, aumentando assim a disponibilidade da informação.

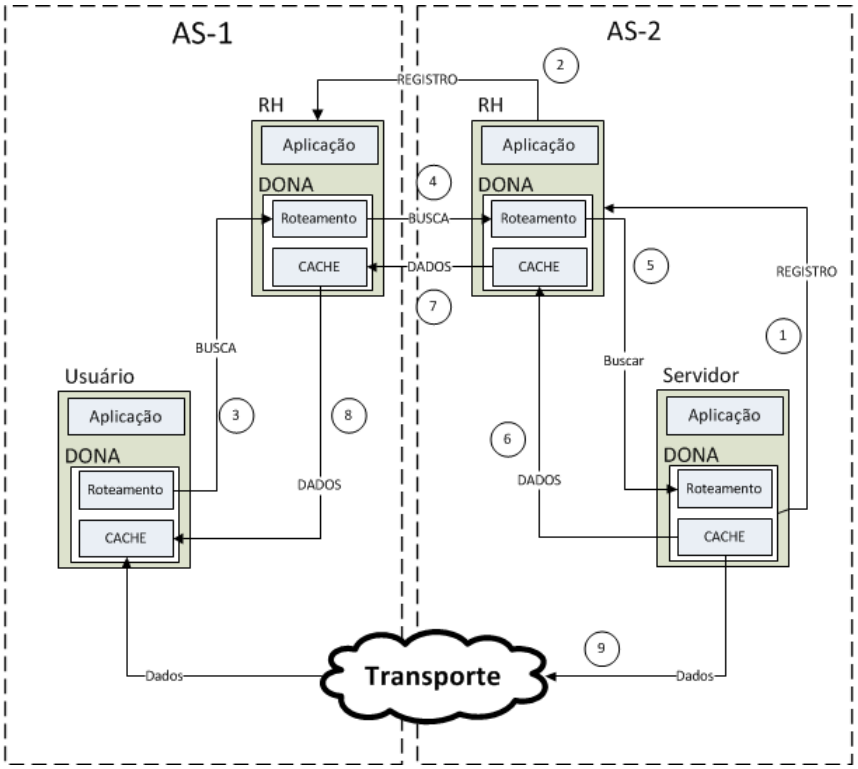


Figura 2.1: Exemplo de um cenário utilizando a arquitetura DONA.

Nomeação ou Identificação: Em DONA um conteúdo publicado é subdividido em fragmentos, onde cada fragmento do conteúdo publicado é associado a um *gerenciador*. Os nomes são uma função *hash* da chave pública P do gerenciador e de um label L que identifica o conteúdo de forma única com relação ao seu gerenciador. A granularidade dos nomes fica a cargo do gerenciador que é considerado o proprietário da informação. Por exemplo, os gerenciadores podem nomear tanto um web site inteiro, quanto cada página dentro dele. Resumindo, os nomes são *flat*, independentes da aplicação, independentes da localização e globalmente únicos.

Resolução de Nomes e Roteamento: A resolução de nomes em DONA é realizada por servidores especializados chamados de manipuladores de resolução (*Resolution Handlers* - *RH*). Existe pelo menos um *RH* lógico em cada AS. Os *RHs* são interconectados formando um serviço de resolução de nomes hierárquico, como ilustrado na 2.1, permitindo assim a resolução de nomes e roteamento de dados entre AS's.

Para tornar a informação disponível o servidor fonte (*gerenciador*) envia uma mensagem REGISTRO com o nome do objeto para o *RH* local que armazena um

ponteiro para o *gerenciador* (passo 1). A partir de então o *RH* propaga este registro para os seus *RHs* pais e os *RHs* do mesmo domínio (passo 2), fazendo com que cada *RH* do caminho de propagação armazene o mapeamento do nome do objeto e o endereço do *RH* que encaminhou o registro.

Para o usuário localizar uma informação, este envia uma mensagem BUSCA para o seu *RH* local (passo 3) que propaga esta mensagem para os seus *RHs* pais até encontrar o registro da informação. Após encontrar, o pedido segue os ponteiros até encontrar o servidor fonte (passos 4 e 5).

O roteamento de dados pode ser acoplado ou desacoplado. Na opção desacoplado, quando a mensagem BUSCA encontra o servidor fonte apropriado, o dado pode ser enviado diretamente ao usuário através do roteamento e encaminhamento IP. Na opção acoplado, as mensagens BUSCA coletam o caminho percorrido, enquanto se movem de *RH* para *RH*, indicando a sequência de AS's percorridas pela requisição. Quando o pedido encontra o servidor fonte, os *RHs* do caminho percorrido são utilizados para encaminhar os dados até o usuário (passos de 6 a 8) ou alternativamente o servidor fonte pode mandar o dado diretamente ao usuário (passo 9).

Armazenamento na Rede: A arquitetura DONA suporta o armazenamento no caminho (*on-path*) através da infraestrutura dos *RHs*. Um *RH* que decide armazenar um objeto de dados requisitado pode substituir o endereço IP da fonte do pedido pelo seu próprio endereço IP antes de encaminhar a mensagem FIND para o próximo *RH*. Assim qualquer resposta será encaminhada para o *RH* corrente e poderá ser armazenada por ele. Na opção acoplada, os dados retornam pelo caminho reverso, logo os *RHs* intermediários podem então decidir se irão armazenar a informação ou não. Caso uma mensagem BUSCA encontre a informação no *cache* de um *RH*, os dados podem ser enviados diretamente ao usuário.

2.3.2 Publish-Subscribe Internet Techonology - PURSUIT

O projeto PURSUIT é a continuação do projeto *PSIRP (Publisher-Subscribe Internet Routing Paradigm)* [16]. Ambos foram desenvolvidos pelo programa *EU Framework 7* que produziu uma arquitetura que substitui completamente o proto-

colo IP.

A arquitetura PURSUIT é composta por três funções separadas: *rendezvous*, gerenciador de topologia e encaminhamento. Quando a função *rendezvous* correlaciona um pedido a um conteúdo, envia para o gerenciador de topologia para criar uma rota entre o servidor fonte e o usuário. Esta rota é usada pela função de encaminhamento para executar a transferência de dados.

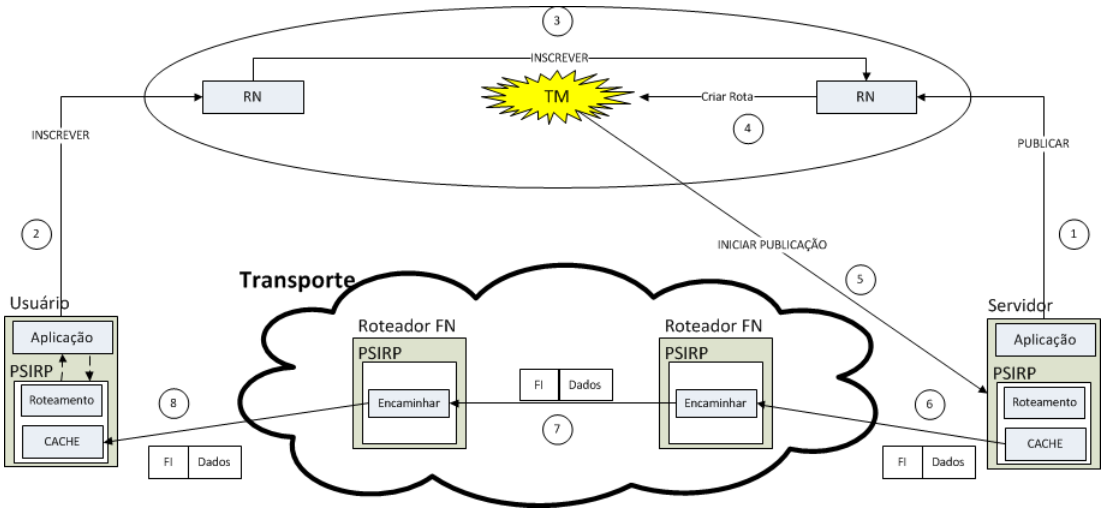


Figura 2.2: Exemplo de um cenário utilizando a arquitetura PSIRP.

Nomeação ou Identificação: Os objetos de informação no PURSUIT são identificados por um único par de IDs, composto por um ID do escopo e por um ID do *rendezvous*. O ID do escopo agrupa objetos que estão relacionados de alguma forma, enquanto que o ID do *rendezvous* é o real identificador para um determinada informação. Enquanto os nomes no PURSUIT são *flat* como na arquitetura DONA, os escopos podem ser organizados em hierarquia, portanto um nome completo consiste de uma sequência de IDs de escopos e um único ID de *rendezvous*.

Resolução de Nomes e Roteamento: Resolução de nomes em PURSUIT é gerenciado pela função *rendezvous* que é implementada por uma coleção de nós *Rendezvous* (RNs - *Rendezvous Nodes*) chamada de rede de *Rendezvous*-RENE implementada por uma DHT hierárquica, como ilustrado na figura 2.2.

Para tornar uma informação disponível, o servidor fonte envia uma mensagem PUBLICAR para o RN local que é roteado por DHT para o RN com o ID do escopo correspondente (passo 1). Um usuário que deseja a informação, envia uma

mensagem INSCREVER para o RN local que é roteado por DHT até o mesmo RN (passos 2 e 3).

A partir de então, o RN informa ao gerenciador de topologia para criar uma rota que conecte o usuário ao servidor fonte para a transferênciada do conteúdo desejado (passo 4). Após criar a rota, o gerenciador de topologia envia para o servidor fonte uma mensagem INICIAR PUBLICAÇÃO (passo 5) que finalmente utiliza a rota recebida para enviar o objeto de informação através dos nós de encaminhamento (FNs - *Forwarding Nodes*) (passos 6 a 8).

A resolução de nomes e o roteamento dos pedidos são desacoplados em PURSUIT. A resolução de nomes é realizada na rede de *Rendezvous*-RENE enquanto o roteamento dos pedidos é organizado pelos gerenciadores de topologia (TMs) e executado pelos FNs.

Armazenamento na Rede: PURSUIT suporta tanto o armazenamento no caminho quanto fora (*on-path* e *off-path*). No caso do armazenamento no caminho, os pacotes encaminhados são armazenados nos FNs para que estes sejam capazes de servir os pedidos subsequentes. Entretanto, este tipo de armazenamento pode não ser eficaz devido ao roteamento e a resolução de nomes serem mecanismos desacoplados. Pedidos para um mesmo objeto de informação podem ser encaminhados para o mesmo RN, enquanto que a transferência de dados podem ser feita usando diferentes caminhos. No caso de armazenamento fora do caminho (*off-path*), as *caches* funcionam como publicadores de informação, notificando uma informação disponível para a RENE.

2.3.3 Named Data Networking - NDN

O projeto *Named Data Networking* (NDN) é uma continuação do projeto *Content-Centric Networking*, que foi desenvolvido pelo programa *US Future Internet Architecture* [17]. Uma característica importante de NDN é que os nomes são hierárquicos, permitindo assim que a resolução de nomes e as informações sobre o roteamento dos dados sejam agregadas através de nomes similares, o que é um fator crítico para a escalabilidade da arquitetura.

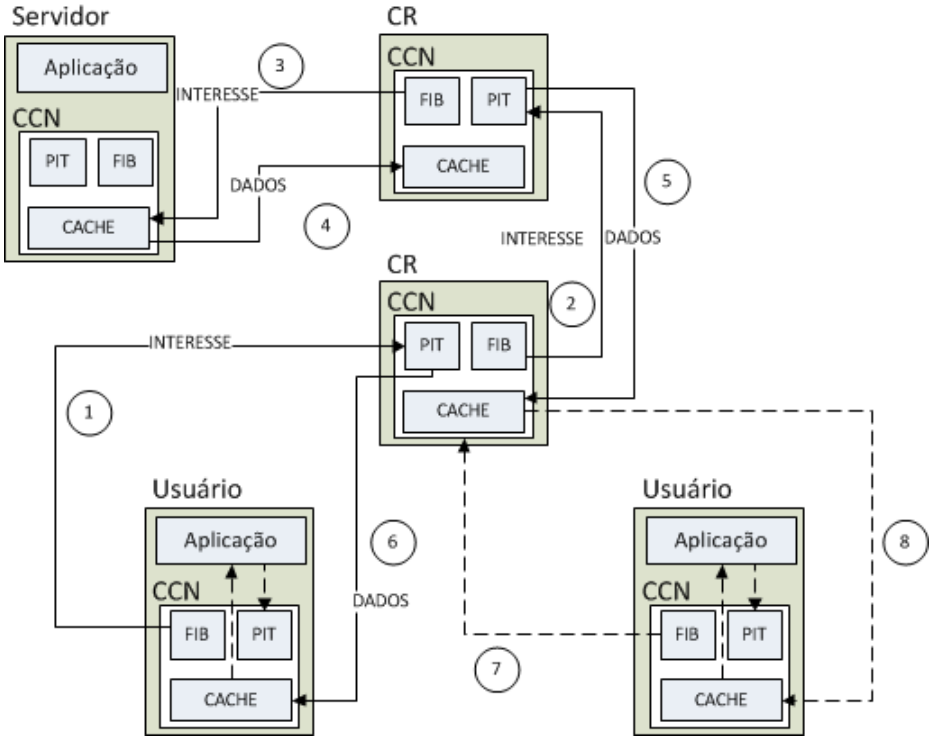


Figura 2.3: Exemplo de um cenário utilizando a arquitetura NDN.

Nomeação ou Identificação: Nomes em NDN são hierárquicos e podem ser similares às URLs. Entretanto, nomes NDN não são necessariamente URLs pois a primeira parte não é um nome de DNS ou um endereço IP e eles não são necessariamente legíveis. Ao invés disso, cada componente do nome pode ser qualquer coisa, incluindo uma palavra legível ou um valor *hash*.

Se é feito um pedido para um certo nome cujo prefixo é **A**, todos os objetos de informação que possuírem o prefixo **A** no seu nome poderão atender o pedido. Por exemplo, um pedido para o nome $/ufrj.br/pesc/main.html$ pode corresponder a um objeto de informação nomeado como $/ufrj.br/pesc/main.html/\_v1/\_s1$, que poderia significar o primeiro segmento da primeira versão do dado requisitado. Depois de receber esse objeto, o usuário pode pedir o próximo segmento de dados através da requisição $/ufrj.br/pesc/main.html/\_v1/\_s2$.

É esperado que a aplicação do usuário conheça a forma com que os objetos são segmentados. No entanto, o fato de vários objetos com o mesmo prefixo corresponderem a um determinado pedido (regra do *prefix matching*) permite que a aplicação saiba quais dados estão disponíveis. Além disso, possibilita que o usuário possa pedir uma informação que ainda não foi produzida. Isto pode ser usado para implementar

aplicações que criam os objetos dinamicamente.

Resolução de Nomes e Roteamento: Na NDN, os usuários enviam mensagens de *Interesse* (*INTEREST messages*) para requisitar objetos de informação. Os objetos são encaminhados através de mensagens de *dados* (*DATA messages*). Ambas as mensagens carregam o nome do objeto.

Como ilustrado na figura 2.3, todas as mensagens são encaminhadas passo-a-passo através dos roteadores de conteúdos (*CRs - Content Routers*). Cada CR mantêm três estruturas de *dados*: a base de informação de encaminhamento (*FIB - Forwarding Information Base*), a tabela de interesses pendentes (*PIT - Pending Interest Table*) e a área de armazenagem de conteúdo (*CS - Content Store*). A FIB mapeia os nomes das informações para as interfaces de saída que são utilizadas para encaminhar as mensagens de *Interesse* para as fontes de dados apropriadas. A PIT identifica a interface de chegada das mensagens de *Interesse* pendentes, ou seja, para as mensagens de *Interesse* que estão aguardando as mensagens de *dados*. Por fim, o CS serve como uma *cache* local para os objetos de informação que passam pelo CR.

Quando uma mensagem de *Interesse* é recebida, o CR extrai o nome da informação e procura no CS se existe algum objeto que possua um nome com o prefixo requisitado. Se algum objeto for encontrado localmente, este é imediatamente enviado de volta através da interface de chegada na mensagem de *dados* e a mensagem de *Interesse* é descartada. Caso contrário, o roteador faz uma busca na FIB pelo prefixo que melhor combina com o nome extraído da mensagem (*longest prefix matching*) a fim de decidir para qual interface de saída a mensagem de *Interesse* será encaminhada. Caso o roteador encontre alguma entrada na FIB para esse prefixo, ele grava a interface de entrada da mensagem de *Interesse* na PIT e envia a mensagem para o CR indicado na FIB (passos 1 a 3). Se a PIT já possuir um registro com o mesmo nome, o roteador adiciona mais um registro para a respectiva interface de entrada e descarta a mensagem INTERESSE, formando assim uma árvore multicast.

Quando uma informação é encontrada tanto no servidor fonte quanto no CS, a mensagem de *Interesse* é descartada e a informação retorna na mensagem de *dados*. A mensagem de *dados* é encaminhada de volta para o usuário baseada nos estados contidos nas PITs ao longo do caminho. Especificamente, quando um CR recebe uma mensagem de *dados*, a primeira coisa a ser é feita é armazená-la no CS e a

partir de então procurar na PIT se existe um registro correspondente aquele objeto de informação contido na mensagem de *dados*. Caso a PIT possua mais de um registro, a mensagem de *dados* é enviada por todas as interfaces e o CR deleta o registro da PIT (passos 4 a 6).

Armazenamento na Rede: NDN suporta naturalmente armazenamento no caminho (*on-path*), já que sempre que uma mensagem de *Interesse* é recebida, o CR verifica se o objeto já se encontra na cache (CS) e armazena todos os objetos contidos na mensagem de *dados*.

2.3.4 Network Information - NetInf

O projeto NetInf faz parte de um projeto maior conhecido como *SAIL-Scalable and Adaptive Internet Solutions* [18]. SAIL tem o objetivo de investigar arquiteturas para a Internet do futuro e meios para tornar a transição entre a Internet de hoje e a Internet do futuro o mais suave possível. NetInf é uma das áreas do projeto SAIL dedicada ao estudo de ICNs. Esta arquitetura é muito genérica, pois combina elementos presentes nas arquiteturas NDN e PURSUIT e pode até operar no modo híbrido. Além disso, pode ser implementada por diferentes tecnologias de roteamento e encaminhamento, introduzindo uma camada de convergência para fazer essa comunicação entre as mensagens NetInf e os pacotes da rede de suporte.

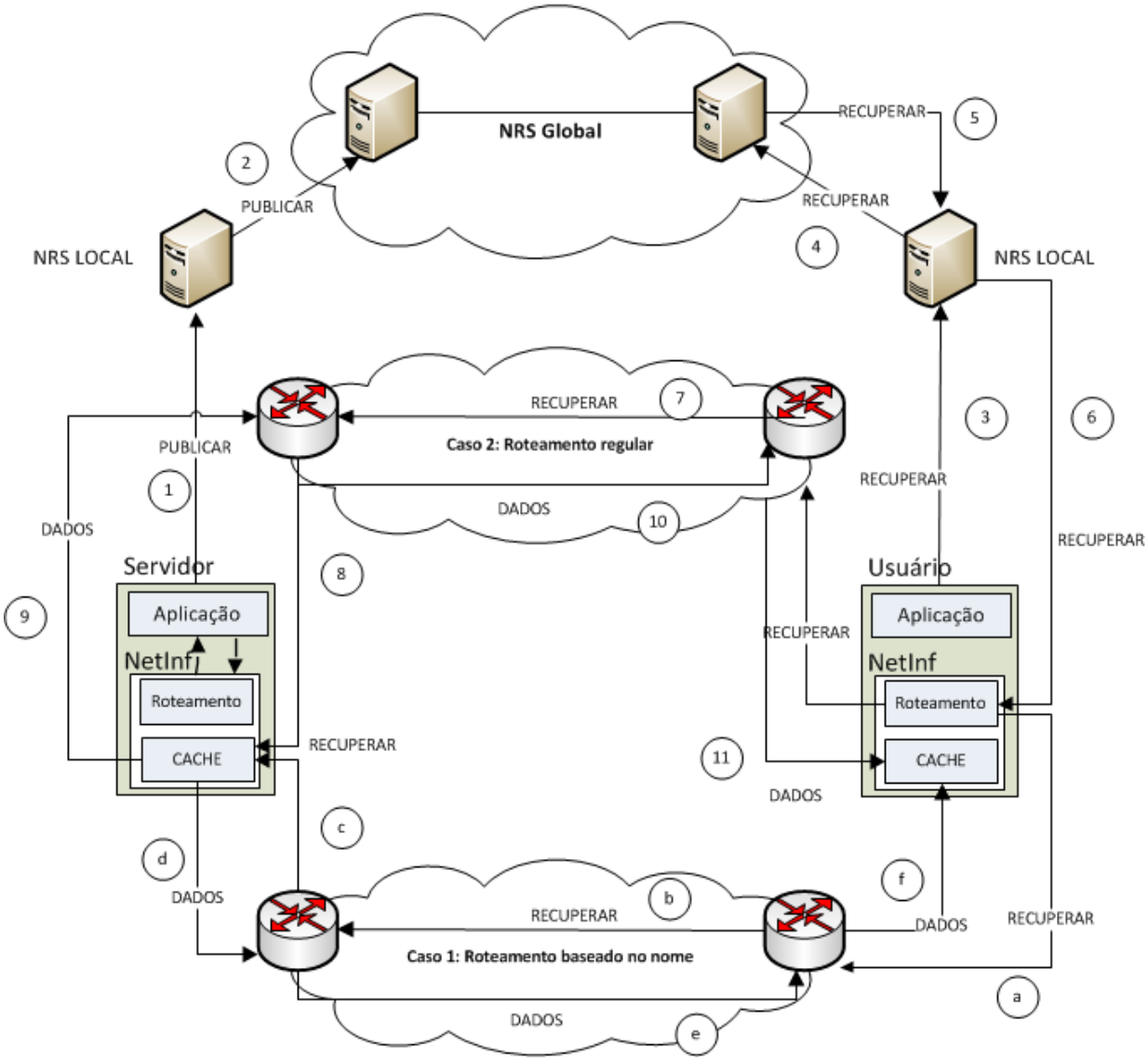


Figura 2.4: Exemplo de um cenário utilizando a arquitetura NetInf.

Nomeação ou Identificação: Os nomes no NetInf seguem um padrão do tipo URI *ni://A/L*, onde *A* é a parte do nome referente a autoridade e *L* é a parte referente ao local (com relação a autoridade). Cada uma das partes pode ser um valor *hash* ou uma sequência de caracteres.

Resolução de Nomes e Roteamento: No NetInf a resolução de nomes e o roteamento podem ser tanto acoplados quanto desacoplados, como ilustrado na figura 2.4. No caso desacoplado, o *Sistema de Resolução de Nomes* (*NRS - Named Resolution System*) é usado para mapear os nomes dos objetos para um identificador da

localização onde a informação está armazenada, como por exemplo o endereço IP. O NRS é uma DHT. Para tornar uma informação disponível, o servidor fonte envia uma mensagem PUBLICAR (*PUBLISH message*) com seu localizador para o NRS local que armazena a parte L do nome. O NRS local agrega todos os nomes com endereços L diferentes e mesmo endereço A e envia para o NRS global (passos 1 e 2). Para localizar uma informação, um usuário envia uma mensagem RECUPERAR (*GET message*) para o seu NRS local que por sua vez consulta o NRS global e retorna a localização do objeto desejado (passos 3 a 6). Por fim, o usuário envia para o servidor fonte uma mensagem RECUPERAR que por sua vez retorna com o dado para o usuário em uma messagem DADOS (*DATA message*) (passos 7 a 12).

No caso acoplado, o protocolo de roteamento é usado para anunciar os nomes dos objetos e preencher as tabelas de roteamento dos roteadores de conteúdos (*CRs - Content Routers*), como na arquitetura NDN. Um usuário envia uma mensagem RECUPERAR para o CR local que é propagada nó-a-nó até o servidor fonte ou até uma *cache* (passos A a C). Quando a informação é encontrada, essa retorna através da mensagem *DADOS* pelo caminho reverso da mensagem RECUPERAR (passos D a F).

Armazenamento na Rede: A arquitetura NetInf, além de armazenar no caminho (*on-path*) através dos CRs, prevê a implementação de armazenamento de objetos de informação em larga escala e mecanismos de replicação em cooperação com o NRS. Essas caches são tratadas como publicadores de informação.

2.4 Modelo Oportunístico baseado em Passeios Aleatórios(*Random Walk*) e Contadores RC

Esse é o modelo utilizado nas simulações desse trabalho. É um modelo baseado em domínios, onde cada domínio é subdividido em subdomínios e, por sua vez, cada subdomínio contém um conjunto de roteadores com capacidade de armazenamento de conteúdos requisitados pelos usuários, ou seja, possui uma *cache* local.

A figura 2.5 ilustra a arquitetura proposta: roteadores encaminham requisições dos usuários, por entre os níveis hierárquicos (setas verdes na figura), em direção à

área de publicação. Ao menos uma cópia de cada conteúdo publicado na rede está localizada na área de publicação. Em cada domínio, passeios aleatórios (*random walks*), com fins de busca de conteúdo, são realizados em apenas um dos domínios de cada nível (setas amarelas na figura). Tal estratégia tem como objetivo reduzir a carga de requisições que chegam na área de publicação, bem como diminuir o tempo de resposta da entrega de conteúdo aos usuários. Um conteúdo requisitado é encaminhado pelo caminho reverso percorrido pela requisição, através da hierarquia (setas azuis na figura). À medida que o conteúdo é encaminhado pela rede, os roteadores decidem se vão ou não armazenar o conteúdo localmente (*cache*). A arquitetura proposta possui dois mecanismos importantes: (i) o passeio aleatório (*random walk*), utilizado para o roteamento do pedido do usuário até a fonte da informação que pode estar na área de publicação ou armazenada nos roteadores da rede e (ii) o mecanismo de armazenamento na rede baseado na popularidade dos conteúdos.

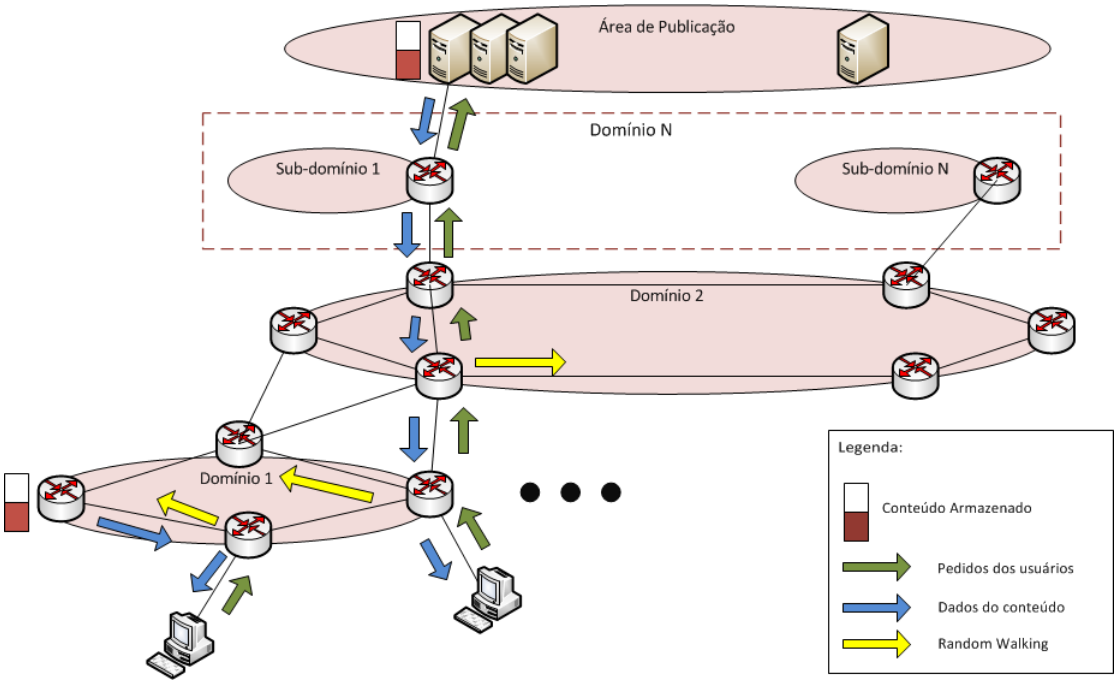


Figura 2.5: Arquitetura ICN proposta.

2.4.1 Armazenamento na Rede baseado na popularidade

O armazenamento de conteúdo nos roteadores da rede (*cache*) é realizado com base na popularidade dos conteúdos. Cada conteúdo é identificado por uma chave

única. Cada roteador da rede possui um conjunto de contadores (RCs - Reinforced Counters), com contadores distintos para conteúdos distintos. Esses contadores possuem dois limitantes: um limitante superior e um limitante inferior.

A finalidade destes contadores é possibilitar que os roteadores decidam se um conteúdo deve ser armazenado/removido da cache local. Sempre que uma requisição chega a um roteador, proveniente de um usuário ou de um roteador em um nível hierárquico inferior, o contador do conteúdo associado a essa requisição, nesse roteador, é incrementado de uma unidade. Quando um contador atinge o limitante superior, o roteador armazena localmente o conteúdo associado a este contador quando o dado for encaminhado para o usuário. Para evitar que o conteúdo mantenha-se armazenado após sua popularidade diminuir, tem-se que todos os contadores são decrementados em uma unidade, com o decurso do tempo. Assim, quando um contador atinge um limitante inferior, se o conteúdo associado a este contador estiver armazenado localmente em um roteador, este conteúdo é removido da cache deste roteador. Cabe ressaltar que, durante o passeio aleatório (*random walk*), os contadores não são incrementados.

Em [13, 14] foi proposto um modelo para representar o comportamento do contador RC. No modelo o comportamento do RC é representado por um processo de nascimento e morte (*birth-death process*) em cada roteador, onde a taxa de nascimento é igual a taxa de entrada dos pedidos no roteador (λ) e a taxa de morte representa a taxa de decréscimo do contador (γ) como ilustrado na figura 2.6. Definiremos então como $\pi_{up}(k)$ a probabilidade no estado estacionário do contador ser maior do que o limite superior k. A probabilidade $\pi_{up}(k)$ pode ser obtido pela equação 2.1.

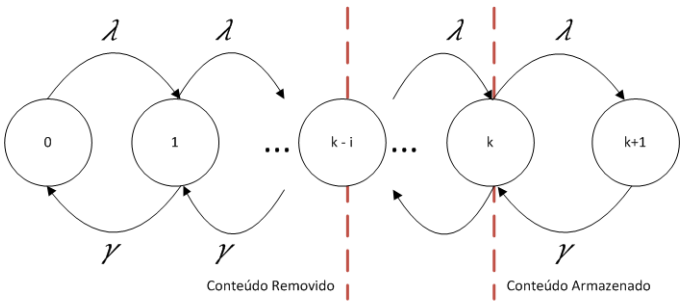


Figura 2.6: Processo de *Birth-Death*.

$$\pi_{up}(k) = \sum_{i=k}^{\infty} \rho^i(1-\rho) = 1 - \sum_{i=0}^{k-1} \rho^i(1-\rho) \tag{2.1}$$

2.4.2 Transferência de Dados

As requisições enviadas pelos usuários são encaminhadas para o topo da hierarquia, ou seja, para a área de publicação. Ponteiros para o caminho que as requisições percorrem são mantidos nos roteadores, possibilitando que o conteúdo seja encaminhado para os usuários seguindo o caminho reverso das requisições. Esse mecanismo é conhecido como as migalhas de pão (*bread crumbs*). Estes ponteiros são removidos à medida que os conteúdos são encaminhados para os usuários.

2.4.3 Roteamento das requisições utilizando passeios aleatórios

O *random walk* é um mecanismo de busca aleatória utilizada pela arquitetura proposta para encontrar a informação desejada na rede, sem precisar ir até a área de publicação para recuperar a informação. Sempre que uma requisição vinda do domínio inferior ou do usuário, chega em um roteador, este roteador verifica se a informação desejada está armazenada na *cache* local. Caso esteja armazenada, o pedido é descartado e o dado é enviado para o usuário pelo caminho reverso percorrido pela requisição do usuário a este servidor. Caso contrário, um passeio aleatório é iniciado a partir desse roteador, para buscar o conteúdo dentro do domínio ao qual este roteador pertence.

O passeio aleatório possui um número máximo de passos. Tal condição evita que seja gasto tempo de busca excessivo em um domínio, após um dado número de passos de um passeio ter ocorrido e o conteúdo não ter sido encontrado. Quando este número máximo de passos é atingido, o pedido é passado para o domínio acima na hierarquia e um novo ciclo se inicia. Se o pedido não é encontrado em nenhum domínio, ele chegará na área de publicação onde estará armazenada ao menos uma cópia do conteúdo sendo requisitado.

Capítulo 3

Simulador ICN

Como mencionado anteriormente, o simulador desenvolvido para esse trabalho é baseado na arquitetura proposta em [13, 14].

O objetivo principal do simulador é avaliar o desempenho da arquitetura proposta considerando diversos cenários.

3.1 Arquitetura

O simulador foi desenvolvido na linguagem C/C++ para plataforma Linux. A figura 3.1 ilustra as relações entre os módulos descritos na seção 3.1.1.

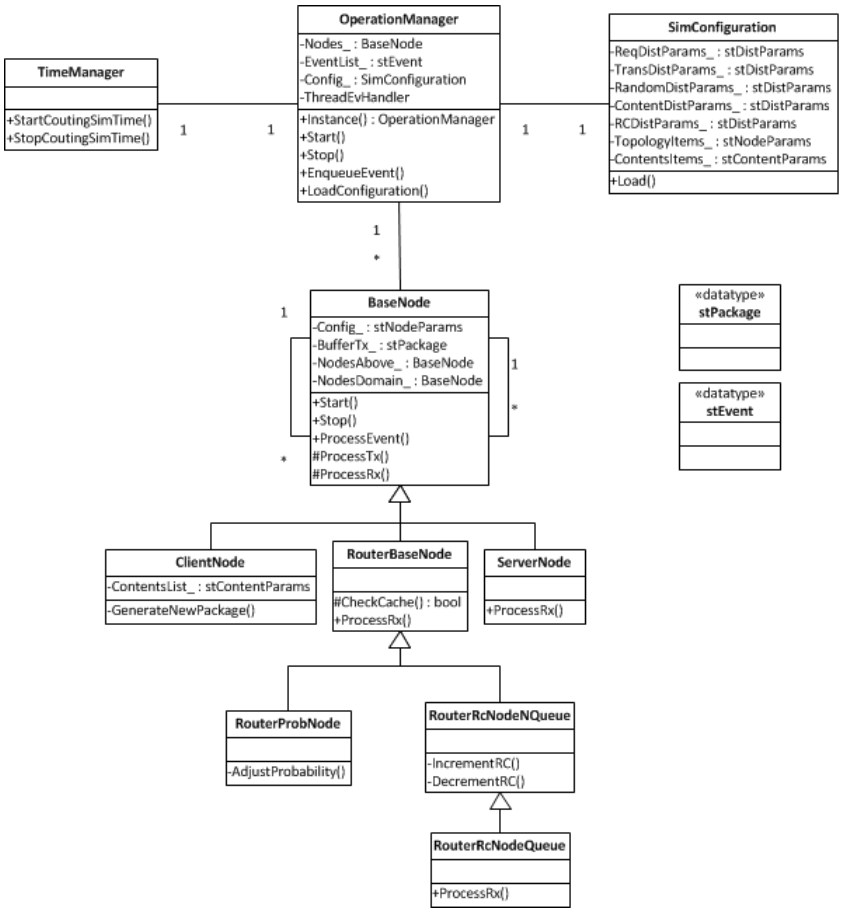


Figura 3.1: Arquitetura do Simulador.

3.1.1 Módulos

ICNSimulator

É a classe de entrada do simulador, ou seja, é a classe responsável por executar e terminar a simulação.

OperationManager

É a classe principal do simulador pois é responsável por criar, conectar e controlar os objetos da topologia carregada no arquivo de configuração. Além disso, é responsável por controlar a lista de eventos gerados pelos objetos e realizar o cálculo das métricas principais do sistema, como por exemplo, a carga filtrada e o tempo de resposta de uma requisição.

BaseNode: É a classe base que pode representar qualquer nó da rede. Ela contém os métodos básicos que são utilizados ou sobrescritos (*override*) pelos nós. Os nós podem ser do tipo ClienteNode, RouterBaseNode ou ServerNode (Área de publicação) que serão descritos a seguir.

ClientNode: É a classe que faz o papel dos clientes na rede, ou seja, é responsável por gerar e transmitir os pedidos dos conteúdos pela rede.

ServerNode: É a classe que faz o papel da Área de Publicação, ou seja, é responsável por contabilizar os pacotes que não foram filtrados nos domínios.

RouterBaseNode: É a classe base que pode representar qualquer tipo de roteador. Os roteadores podem ser do tipo RouterProbNode, RouterRcNodeNQueue e RouterRcNodeQueue que serão descritos a seguir.

1. RouterProbNode: é a classe que faz o papel de um roteador que filtra os pedidos baseado em uma probabilidade fixa, ou seja, a probabilidade do conteúdo estar na *cache* do roteador é dada por um valor fixo.

2. RouterRcNodeNQueue: é a classe que faz o papel de um roteador que não possui fila de entrada, ou seja, todo pacote recebido é imediatamente atendido. Além disso, a classe utiliza o valor do contador RC para verificar se o conteúdo está ou não armazenado na *cache*.

3. RouterRcNodeQueue: a única diferença dessa classe para o RouterRcNodeNQueue, é que existe uma fila de entrada, ou seja, todo pacote recebido, entra em uma fila para ser atendido.

TimeManager: É a classe responsável por controlar o tempo de simulação.

ConfigManager: É a classe responsável por interpretar e carregar a configuração a partir de um arquivo de entrada.

Utilitários: Existem algumas classes e estruturas que são utilitárias, ou seja, não participam ativamente da execução.

1. DisUtility: é responsável por gerar valores que seguem algum tipo de distribuição. Podendo ser exponencial, uniforme, normal ou ZPIF.

2. stEvent: é uma estrutura de dados utilizada pelos objetos para gerar eventos durante a simulação.

3. stPackage: é uma estrutura de dados utilizada pelos objetos para transmitir as informações do pedido do conteúdo.

3.2 Geração de Amostras de Variáveis Aleatórias

O simulador é capaz de gerar amostras de variáveis aleatórias com as seguintes distribuições: exponencial, normal, uniforme e Zipf.

A seguir descreveremos os métodos usados para gerar as amostras assim como um gráfico que valida a implementação da função usada para a geração das amostras.

3.2.1 Distribuição Exponencial

Amostras de uma variável aleatória exponencial com parâmetro λ são geradas a partir do algoritmo da transformada inversa [19], através da equação 3.1.

A figura 3.2 mostra o histograma obtido a partir de 1000 amostras geradas pela função implementada no simulador considerando $\lambda = 10$.

$$E = -\frac{1}{\lambda}\log(U) \tag{3.1}$$

onde U tem distribuição uniforme com parâmetros 0 e 1.

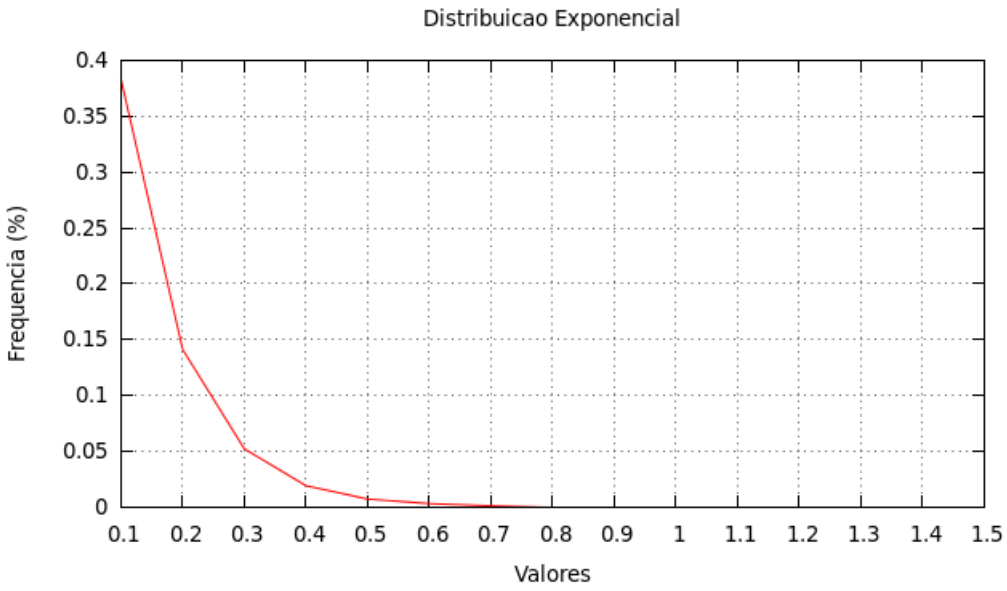


Figura 3.2: Histograma gerado com amostras da variável aleatória exponencial.

3.2.2 Distribuição Normal

Para gerar amostras de uma variável aleatória normal foi utilizado o método de Box-Muller [20].

As amostras foram geradas a partir da equação 3.2.

A figura 3.3 mostra o histograma obtido a partir de 1000 amostras geradas pela função implementada no simulador para $\mu = 0$ e $\sigma = 1$.

$$N = ((\sqrt{-2\log(U)}\cos(2\phi V))\sigma) + \mu \tag{3.2}$$

onde U e V tem distribuição uniforme com parâmetros 0 e 1.

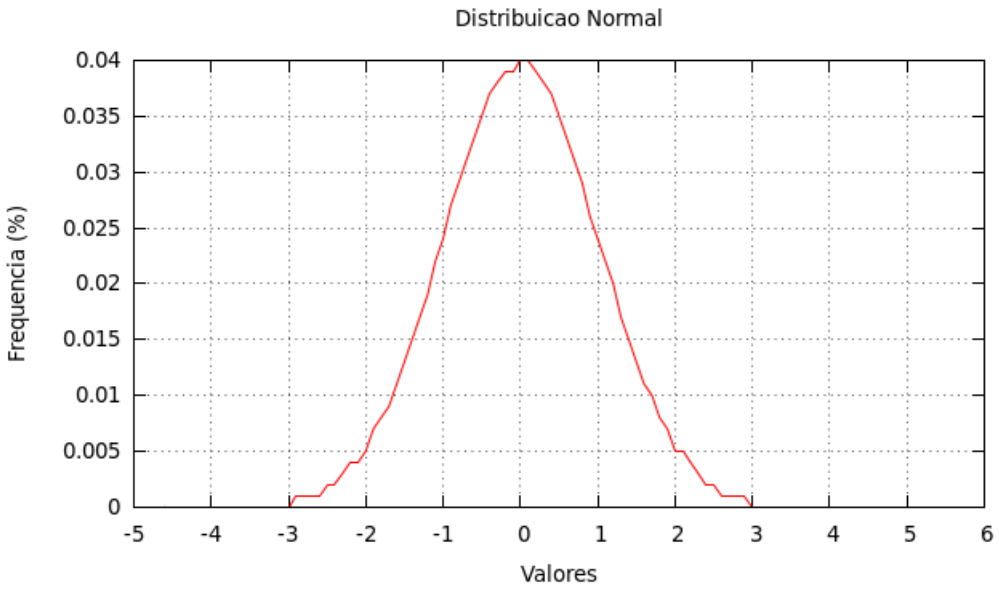


Figura 3.3: Histograma gerado com amostras da variável aleatória normal.

3.2.3 Distribuição Zipf

Consideramos a distribuição Zipf para atribuir popularidade aos conteúdos. Suponha que N diferentes conteúdos são particionados em i classes de popularidade, isto é, um conteúdo de classe i é requisitado com a probabilidade definida pela equação 3.3.

$$p(i) = \frac{(\sum_{i=1}^{N} 1/i^{\alpha})^{-1}}{i^{\alpha}} \tag{3.3}$$

O índice i será selecionado como valor da distribuição Zpif, caso o valor de U, uma variável aleatória uniforme $[0, 1]$, seja menor ou igual a $p(i)$.

A figura 3.4 mostra o histograma da Zpif para os parâmetros $\alpha = 1.2$ e $N = 10$.

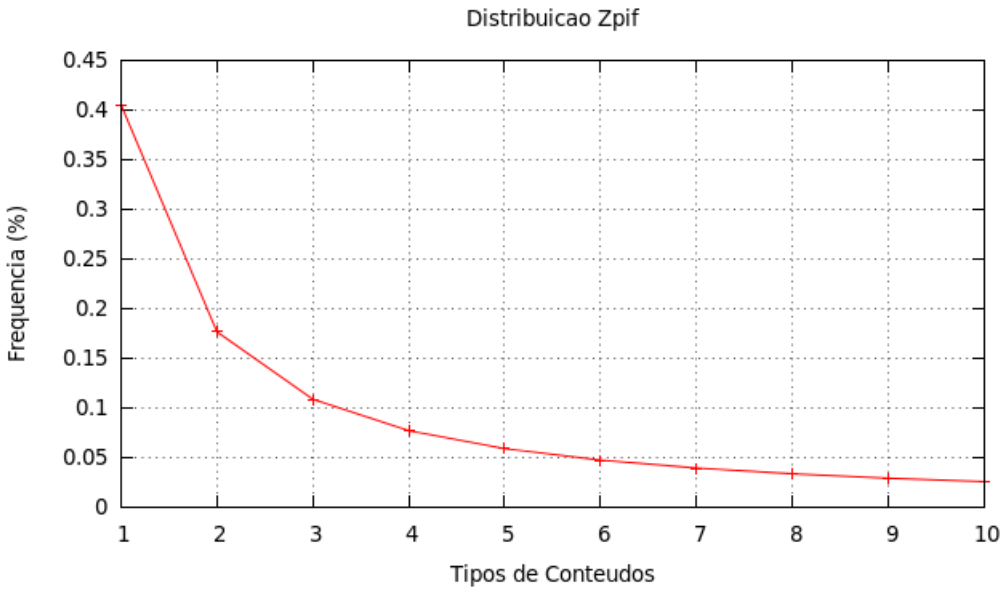


Figura 3.4: Histograma gerado com amostras da variável aleatória Zpif.

3.3 Validação do Simulador

A avaliação do simulador foi realizada em etapas, de forma incremental. Foram elaboradas diversas versões e em cada uma das versões, foram adicionadas novas características ao simulador. A validação foi realizada comparando-se os resultados obtidos pelo simulador com os resultados analíticos. Foi calculado o erro relativo para cada uma das métricas. Seja o $result_{sim}$ o resultado obtido pela simulação e $result_{ana}$ o resultado analítico. Então, definimos o erro relativo a partir de $|result_{sim} - result_{ana}|/result_{ana}$.

Os principais resultados analíticos usados foram a Lei de Little e a fila M/M/1. Um sistema com filas, sob certas condições [1], deve obedecer a Lei de Little:

$$E[N_{sistema}] = \lambda E[T_{sistema}] \tag{3.4}$$

onde $E[T_{sistema}]$ é o tempo médio de permanência no sistema, $E[N_{sistema}]$ é o número médio de usuários no sistema e λ é a taxa de chegada no sistema.

Os principais resultados da fila M/M/1 que foram usados são a utilização da fila (ρ), o tempo médio ($E[T_{fila}]$) e o número médio de usuários na fila ($E[N_{fila}]$).

$$\rho = \lambda/\mu \tag{3.5}$$

$$E[N_{fila}] = \frac{\rho}{1-\rho} \tag{3.6}$$

onde λ é a taxa de chegada na fila e μ é a taxa de serviço.

Os parâmetros utilizados nas validações foram baseados nos seguintes critérios:

- Fixamos a taxa de transmissão (μ) e variamos a taxa de chegada das requisições (λ) para que tenhamos uma utilização (ρ) baixa, média e alta, ou seja, 30%, 50% e 90%;
- O valor do TTL foi baseado na quantidade de roteadores nos domínios, ou seja, o valor do TTL não podia ser alto já que simulamos uma topologia pequena;
- O parâmetro da Zpif foi retirado do artigo [21], a popularidade de um vídeo na internet pode ser representado por uma Zpif de parâmetro $1,2$;
- A taxa de decréscimo do RC (γ) foi estimada a partir do cálculo reverso do ϕ_{up}. Onde fixamos um valor de probabilidade de encontrar o conteúdo no roteador (10%) e uma utilização do processo do RC. Assim, a partir da fórmula 2.1 podemos estimar o valor de k, ou seja, o valor do limite do RC.

3.3.1 Versão 1.0: Roteadores com filas de espera, sem RC, sem Random Walking e sem Conteúdos

Na primeira versão foram implementadas a geração de eventos e as filas nos objetos da rede. O objetivo foi coletar os resultados das filas e comparar com as fórmulas da fila M/M/1. O módulo *ClientNode* gera requisições, onde o intervalo entre as requisições possui distribuição exponencial. As requisições entram na fila do buffer de saída para serem transmitidas para o *RouterBaseNode* a ele conectado. O *RouterBaseNode* apenas retira a requisição da fila e transmite para um roteador do próximo domínio, até a requisição chegar no *ServerNode* (Área de Publicação) e sair do sistema. O tempo de transmissão em todos os roteadores da rede é exponencial. Se um roteador está conectado a diversos outros, ele escolhe uniformemente um deles para transmitir a requisição.

Tabela 3.1: Configuração utilizada na versão 1.0

Parâmetro	Valor
Tempo de simulação	10000 segundos
Tempo entre requisições (λ)	Utilizamos uma utilização de 30%, 50% e 90%, ou seja, $\lambda = 4,5$; $7,5$ e $13,5$
Tempo entre transmissões (μ)	15
Topologia	Topologia simétrica: 4 roteadores no primeiro domínio, todos conectados entre si. Cada um conectado a um cliente.

Como mencionado anteriormente, o nosso objetivo nessa versão é verificar se as medidas calculadas por todos os objetos da rede estão condizentes com os resultados da fila M/M/1, ou seja, $E[N]$, $E[T]$ e ρ.

A configuração utilizada para essa simulação está descrita na tabela 3.3.1.

Usando a configuração apresentada acima e as fórmulas da M/M/1 obtivemos os seguintes resultados:

1. Para uma utilização de 30%: $E[N] = 0,42$ e $E[T] = 0,09$
2. Para uma utilização de 50%: $E[N] = 1$ e $E[T] = 0,13$
3. Para uma utilização de 90%: $E[N] = 9$ e $E[T] = 0,66$

A tabela 3.3.1 apresenta os resultados obtidos a partir da simulação. Enquanto a tabela 3.3.1 apresenta o erro relativo.

3.3.2 Versão 2.0: Roteadores sem filas de espera, sem RC, com Random Walking e sem Conteúdos

Nessa versão implementamos o passeio aleatório (*Random Walking*) sem as filas nos nós, ou seja, quando uma requisição é gerada, ela é imediatamente transmitida sem entrar em fila. Como ainda não existem conteúdos armazenados dentro dos roteadores, a requisição irá percorrer todos os passos até estourar o TTL (*time-to-live*) e passar para o próximo domínio até chegar na área de publicação.

Tabela 3.2: Resultados da versão 1.0.

Nó	Medidas	Utilização		
		30%	50%	90%
Cliente	Requisições Geradas	44976	74756	135597
	Requisições Enviadas	44976	74756	135597
	Utilização	0,2982	0,4971	0,9046
	E[N]	0,4257	0,9887	0,90613
	E[T]	0,0946	0,1322	0,6682
Router	Requisições Recebidas	44976	74756	135597
	Requisições Enviadas	44976	74756	135597
	Utilização	0,2980	0,4995	0,9025
	E[N]	0,4217	0,9883	0,9025
	E[T]	0,0937	0,1322	0,6639

Tabela 3.3: Erro Relativo da versão 1.0.

Nó	Medidas	Erro		
		30%	50%	90%
Cliente	Utilização	0,0019	0,0029	0,0046
	E[N]	0,0057	0,0013	0,0613
	E[T]	0,0046	0,0022	0,0082
Roteador	Utilização	0,0020	0,0005	0,0025
	E[N]	0,0017	0,0017	0,0027
	E[T]	0,0037	0,0022	0,0039

Tabela 3.4: Resultados da versão 2.0.

Nó	Medidas	Utilização		
		30%	50%	90%
	λ	4,5 0	7,5	13,5
	μ	15	15	15
Domínio 1	Tempo de permanência	0,333325	0,33341	0,333043
	λ	4,5	7,5	13,5
	μ	15	15	15
Domínio 2	Tempo de permanência	0,333404	0,3332	0,3334

Tabela 3.5: Erro Relativo da versão 2.0.

Nó	Medidas	Erro		
		$\rho = 0,3$	$\rho = 0,5$	$\rho = 0,9$
Domínio 1	Tempo de permanência	$8 * 10^{-5}$	$7 * 10^{-5}$	$2 * 10^{-4}$
Domínio 2	Tempo de permanência	$7 * 10^{-5}$	$1 * 10^{-4}$	$1 * 10^{-4}$

Estamos interessados em calcular o tempo médio de permanência de uma requisição dentro do domínio. Como a fila foi removida dos roteadores, temos que o tempo médio de permanência dentro do roteador é igual ao tempo médio que o roteador gasta para transmitir uma requisição, ou seja $\frac{1}{\mu}$. Além disso, como não há conteúdos armazenados dentro do domínio, o pedido irá percorrer TTL roteadores.

Assim:

$$E[T_{dominio}] = TTL * E[T_{roteador}] = TTL * \frac{1}{\mu} \tag{3.7}$$

Utilizamos a mesma configuração da versão 1.0 descrita na tabela 3.3.1, porém incluimos o valor do $TTL = 5$. Assim temos que o tempo médio de permanência no domínio para $\mu = 15$, é $E[T_{dominio}] = 0,33$. A tabela 3.3.2 apresenta os resultados obtidos a partir da simulação e a tabela 3.3.2 apresenta o erro relativo.

Podemos notar que independente da variação do λ o tempo não se altera. Isso se deve ao fato de que como o sistema é sem fila de espera, o tempo de permanência no roteador só depende da taxa de transmissão (μ).

3.3.3 Versão 2.5: Roteadores sem filas de espera, sem RC, com Random Walking e com Conteúdos

Antes de incluirmos a fila de espera nos roteadores, criamos uma versão para simular o armazenamento dos conteúdos no domínio, ou seja, toda vez que o roteador receber um pedido, irá filtrar o pedido com uma probabilidade p.

Estamos interessados em computar o tempo médio de permanência do conteúdo dentro do domínio. Na versão anterior, a quantidade de saltos era fixa, já que não tinha nenhum conteúdo armazenado no domínio. Nessa versão, temos que o conteúdo pode estar armazenado no roteador com probabilidade p. Assim, podemos mapear essa quantidade de saltos em uma variável aleatória.

Suponha um caso simples, onde o TTL máximo é 3. Então temos que a probabilidade do pedido achar o conteúdo em zero saltos é dada por p, em um salto é dada por $(1-p)p$, em dois saltos é $(1-p)^2p$ e finalmente em três saltos é $1-(p+(1-p)p+(1-p)^2p)$.

A partir do exemplo simples acima, é fácil ver que o valor esperado do número de saltos dentro de um domínio pode ser calculado a partir de:

$$E[N_{saltos}] = \sum_{i=0}^{T-1} i(1-p)^i p + (T * (1 - \sum_{i=0}^{T-1} i(1-p)^i p)) \tag{3.8}$$

A partir de (3.8) temos que:

$$E[T_{dominio}] = \sum_{i=1}^{T} E[T_{roteador}] i p_{N_{saltos}}(i) = E[T_{roteador}] E[N_{saltos}] \tag{3.9}$$

onde $p_{N_{saltos}}(i)$ é a função probabilidade massa da variável aleatória discreta N_{saltos} e T é o TTL máximo.

Iremos utilizar a mesma configuração da versão 1.0, descrita na tabela 3.3.1, porém temos novos parâmetros a serem considerados, o TTL e a probabilidade p do conteúdo estar armazenado no roteador. Para os resultados apresentados vamos utilizar $TTL = 5$ e $p = 10\%$. Assim temos que o número esperado de saltos dentro do domínio é $E[N_{saltos}] = 3,675$ e o tempo médio de permanência é $E[T_{domnio}] = 0,245$.

A tabela 3.3.3 apresenta os resultados obtidos a partir da simulação, enquanto a tabela 3.3.3 apresenta os valores do erro relativo.

Tabela 3.6: Resultados da versão 2.5.

Nó	Medidas	Utilização		
		30%	50%	90%
	λ	4,5 0	7,5	13,5
	μ	15	15	15
Domínio 1	Tempo de permanência	0,246222	0,24632	0,24689
	λ	4,5 0	7,5	13,5
	μ	15	15	15
Domínio 2	Tempo de permanência	0,246172	0,24596	0,24675

Tabela 3.7: Erro Relativo da versão 2.5.

Nó	Medidas	Erro		
		$\rho = 0,3$	$\rho = 0,5$	$\rho = 0,9$
Domínio 1	Tempo de permanência	0,00012	0,00132	0,00189
Domínio 2	Tempo de permanência	0,001172	0,00096	0,0047

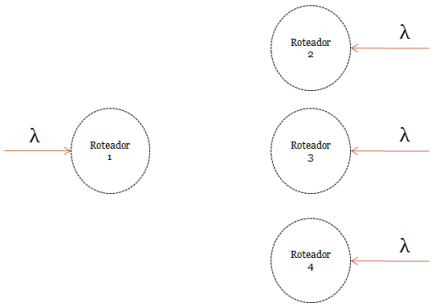
3.3.4 Versão 3.0: Roteadores com filas de espera, sem RC, com Random Walking e sem Conteúdos

As versões 2.0 e 2.5 foram criadas para validar o algoritmo de passeios aleatórios em roteadores sem fila. Agora que já validamos o algoritmo de passeios aleatórios, vamos inserir a fila nos roteadores e usar o resultado de Little e da fila M/M/1 para avaliar os resultados. Nessa versão, os roteadores estão com filas de entrada, ou seja, cada requisição recebida entra em uma fila para ser atendida. Além disso, os roteadores não armazenam nenhum conteúdo sendo assim as requisições vão percorrer o domínio até o valor máximo do TTL ser alcançado.

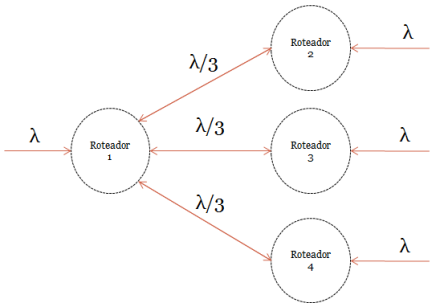
Definiremos duas taxas em um domínio: uma taxa exógena que chega em um roteador proveniente de um outro domínio, e uma taxa endógena que é gerada pelos roteadores do próprio domínio devido ao passeio aleatório.

Para melhor entendimento do cálculo da taxa endógena, vamos imaginar um cenário simétrico e simples com quatro roteadores no domínio, totalmente interconectados entre si, todos com TTL igual a 3 e recebendo uma taxa exógena λ, como ilustrado na figura 3.5.

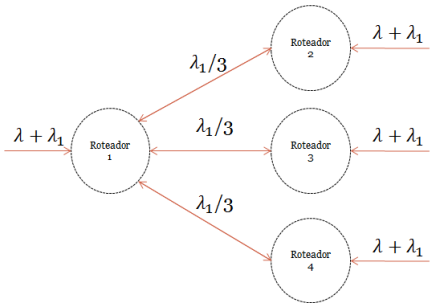
A ideia para calcularmos a taxa endógena para esse cenário é colocar em cada requisição uma estampa identificando quantos passos foram dados no *Random Walking*.



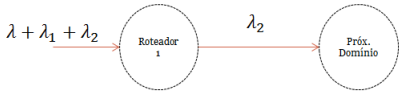
(a) *Random Walking* no passo 1.



(b) *Random Walking* no passo 2.



(c) *Random Walking* no passo 3.



(d) *Random Walking* no passo 4.

Figura 3.5: Exemplo para o cálculo da taxa total (endógena + exógena).

Sendo que o λ_i é a identificação da taxa λ no passo i.

Assim, temos:

1. No passo antes de iniciar o *Random Walking* como ilustrado na figura 3.5(a), todas as requisições chegam aos roteadores com a estampa 0;

2. Ao iniciar o *Random Walking* as requisições com estampa 0 são encaminhadas com estampas 1 com taxa $\frac{\lambda}{3}$ para os roteadores a ele conectado, como ilustrado pela figura 3.5(b);

3. No segundo passo do *Random Walking*, temos em cada roteador chegando λ dos clientes mais $\frac{\lambda}{3}$ de cada roteador, ou seja, 2λ $(\lambda + \lambda_1)$. As requisições com estampa 1 são encaminhadas novamente com estampa 2, como ilustrado na figura 3.5(c);

4. No último passo do *Random Walking*, temos em cada roteador chegando λ dos clientes mais λ com estampa 1 e mais λ com estampa 2, ou seja 3λ ($\lambda+\lambda_1+\lambda_2$). Nesse passo as requisições de estampa 2 são encaminhadas com estampa 3, porém 3 é o TTL máximo. Então, essas requisições são encaminhadas para o próximo domínio com taxa λ, como ilustrado na figura 3.5(d).

A seguir iremos demonstrar qual a taxa total que chega em um roteador.

Proposição 1 *A taxa total (endógena + exógena) nos roteadores em uma topologia simétrica é dada por:*

$$\lambda_{total} = TTL * \lambda, \ onde \ TTL > 0. \tag{3.10}$$

Prova 1 *Seja K o número de roteadores na rede e λ a taxa de chegada de requisições em cada roteador. Logo a taxa total exógena é igual a $\lambda * K$. Seja $1/\mu$ o tempo médio de serviço de cada requisição, logo o tempo médio que cada requisição fica no domínio é $TTL * 1/\mu$. Pela lei de Little temos:*

E[número de requisições por roteador] $= \lambda_{total} * \frac{1}{\mu} =$

E[número de requisições no domínio] $* \frac{1}{K} = \frac{\lambda * K * TTL}{\mu} * \frac{1}{K}$

■

Iremos manter a configuração utilizada nas versões anteriores, porém com $\lambda = 2,5$. Então, para as configurações apresentadas, temos $\lambda_{total} = 5 * 2,5 = 12,5$. Agora já podemos calcular os resultados de Little e da fila M/M/1: $\rho = \frac{12,5}{15} = 0,8333$, $E[N] = 4,99$, $E[T_{roteador}] = 0,3999$ e $E[T_{dominio}] = 5 * 0,3999 = 1,999$. As tabelas 3.3.4 e 3.3.4 apresentam os resultados obtidos a partir dessa versão. Enquanto as tabelas 3.3.4 e 3.3.4 apresentam os valores do erro relativo.

3.3.5 Versão 3.5: Roteadores com filas de espera, sem RC, com Random Walking e com Conteúdos

Mantendo a mesma linha de raciocínio das versões 2.0 e 2.5, temos que a versão 3.5 é a mesma que a 3.0, porém incluimos uma probabilidade p do conteúdo estar armazenado no roteador. Para o cálculo do λ_{total}, usaremos a mesma estratégia

Tabela 3.8: Resultados de Little e M/M/1 da versão 3.

Nó	Medidas de Interesse			
	ρ	$E[N]$	$E[T_{roteador}]$	$E[T_{dominio}]$
Roteador 1	0,831587	4,97080	0,398394	1,99854
Roteador 2	0,833310	5,00223	0,400548	1,99854
Roteador 3	0,833449	5,02399	0,402101	1,99854
Roteador 4	0,831483	4,96461	0,397783	1,99854

Tabela 3.9: Resultados das taxas da versão 3.

Nó	Taxa Endógena	Taxa Exógena	Taxa Total (endo + exo)
Roteador 1	2,49551	2,49363	12,4707
Roteador 2	2,50496	2,49778	12,4884
Roteador 3	2,49668	2,49848	12,4942
Roteador 4	2,49108	2,49811	12,4806

Tabela 3.10: Erro Relativo para os valores Little e M/M/1 da versão 3.

Nó	Medidas de Interesse			
	ρ	$E[N]$	$E[T_{roteador}]$	$E[T_{dominio}]$
Roteador 1	0,00141	0,0282	0,00061	0,001459
Roteador 2	0,00031	0,00323	0,001548	0,001459
Roteador 3	0,000449	0,02499	0,003101	0,001459
Roteador 4	0,00152	0,03439	0,00122	0,001459

Tabela 3.11: Erro Relativo para os valores das taxas da versão 3.

Nó	Taxa Endógena	Taxa Exógena	Taxa Total (endo + exo)
Roteador 1	0,00449	0,00637	0,02293
Roteador 2	0,00496	0,00222	0,01156
Roteador 3	0,00332	0,00152	0,0058
Roteador 4	0,00892	0,00189	0,01932

Tabela 3.12: Resultados de Little e M/M/1 da versão 3.5.

Nó	Medidas de Interesse			
	ρ	$E[N]$	$E[T_{roteador}]$	$E[T_{dominio}]$
Roteador 1	0,615072	1,58564	0,172151	0,6353
Roteador 2	0,616464	1,60325	0,173435	0,6353
Roteador 3	0,614756	1,58309	0,171535	0,6353
Roteador 4	0,615155	1,59854	0,173313	0,6353

utilizada na versão 3.0 porém, agora, para cada transição temos que a taxa de saída para o próximo passo do *Random Walking* será multiplicada pela probabilidade $1-p$.

Assim, temos que:

$$\lambda_{total} = \lambda \sum_{i=1}^{TTL} (1-p)^{i-1} \tag{3.11}$$

Simplificando a equação 3.11 temos:

$$\lambda_{total} = \lambda(\frac{1-(1-p)^{TTL}}{p}) \tag{3.12}$$

A partir da equação (3.11) temos que a última parcela do somatório é a taxa de saída do roteador para o outro domínio, ou seja, é a quantidade de pacotes com estampa 3. Esta taxa é dada por $\lambda_{out} = \lambda(1-p)^{TTL-1}$.

Iremos manter a configuração utilizada nas versões anteriores, porém com $\lambda = 2,5$ e a probabilidade de encontrar o conteúdo $p = 0,1$. A partir da equação (3.12), temos que $\lambda_{total} = 9,2139$. Com isso, podemos calcular os resultados de Little e da fila M/M/1: $\rho = \frac{9,2139}{15} = 0,6142$, $E[N] = 1,5924$, $E[T_{roteador}] = 0,1728$ e $E[T_{dominio}] = E[N_{saltos}] * E[T_{roteador}] = 0.6353$. Note que o número esperado de saltos só depende da probabilidade de encontrar o conteúdo no roteador e do número máximo de saltos (TTL). Assim, o valor esperado é o mesmo do apresentado na versão 2.5.

As tabelas 3.3.5 e 3.3.5 apresentam os resultados obtidos a partir dessa versão. Enquanto as tabelas 3.3.4 e 3.3.4 apresentam os valores do erro relativo.

Tabela 3.13: Resultados das taxas da versão 3.5.

Nó	Taxa Endógena	Taxa Exógena	Taxa Total (endo + exo)
Roteador 1	2,50128	1,47849	9,22816
Roteador 2	2,50760	1,48033	9,24410
Roteador 3	2,50106	1,47620	9,22893
Roteador 4	2,50560	1,48158	9,22339

Tabela 3.14: Erro Relativo: Resultados de Little e M/M/1 da versão 3.5.

Nó	Medidas de Interesse			
	ρ	$E[N]$	$E[T_{roteador}]$	$E[T_{dominio}]$
Roteador 1	0,000812	0,00376	0,00065	0,0076
Roteador 2	0,002204	0,01085	0,00063	0,0076
Roteador 3	0,000496	0,00931	0,00127	0,0076
Roteador 4	0,000895	0,00614	0,00051	0,0076

Tabela 3.15: Erro relativo para os valores das taxas da versão 3.5.

Nó	Taxa Endógena	Taxa Exógena	Taxa Total (endo + exo)
Roteador 1	0,00128	0,00229	0,01426
Roteador 2	0,0076	0,00413	0,0302
Roteador 3	0,00106	$2 * 10^{-4}$	0,01503
Roteador 4	0,0056	0,00538	0,00949

3.3.6 Versão 4.0: Roteadores sem filas de espera, com RC e com Random Walking

Nessa versão, implementamos a última funcionalidade da arquitetura proposta, o contador (RC) em cada roteador. Além disso, incluimos a opção do cache ser infinito ou finito com duas políticas de substituição: aleatória ou pelo menor RC. Onde na primeira, o conteúdo a ser retirado da cache é escolhido aleatoriamente e na segunda o conteúdo retirado é aquele com o menor valor do RC.

Como mencionado no capítulo 2, a função dos contadores é fazer com que o roteador armazene um determinado conteúdo caso o valor do RC tenha atingido o limite superior pré-configurado.

Lembrando que os valores dos RCs somente serão incrementados caso a requisição recebida seja de outro domínio ou do cliente a ele conectado, ou seja, os pacotes gerados pelo *Random Walking* não incrementam os RCs.

Para validar o simulador, iremos calcular a probabilidade de encontrar o conteúdo no primeiro roteador do domínio ($p(hit)$), ou seja, o roteador que recebe a requisição do cliente ou de um roteador de outro domínio. Esta probabilidade é igual ao valor de $\pi_{up}(k)$, que é a probabilidade no estado estacionário do contador ser maior do que o limite superior k (Ver definição no capítulo 2). Neste primeiro roteador, $\pi_{up}(k)$ só depende do valor do RC estar maior do que o limite superior.

Iremos manter a configuração das versões anteriores, porém iremos incluir o RC com $k = 5$ e $\gamma = 28,75$. Note que para facilitar o cálculo consideramos o limite inferior do RC igual ao limite superior, ou seja, $\pi_{up}(k)$, a probabilidade de encontrar o conteúdo na cache só irá depender do valor do RC estar ou não acima do limite superior. Com a configuração acima temos que $\pi_{up}(k)$ é dado por:

$$\pi_{up}(k) = 1 - \frac{((1-\rho_{RC}) - (1-\rho_{RC})\rho_{RC}^{k})}{1-\rho_{RC}} = 0,64 \tag{3.13}$$

onde k é o limite do RC e ρ_{RC} é dado por $\frac{\lambda}{\gamma}$. Lembrando que λ é a taxa de chegada de requisições e γ é a taxa de decréscimo do RC.

A figura 3.6 mostra o valor da probabilidade de encontrar o conteúdo no primeiro domínio obtida com o simulador. Note que para TTL=0, ou seja, quando a requisição chega no domínio, $p(hit) \approx 0,64$.

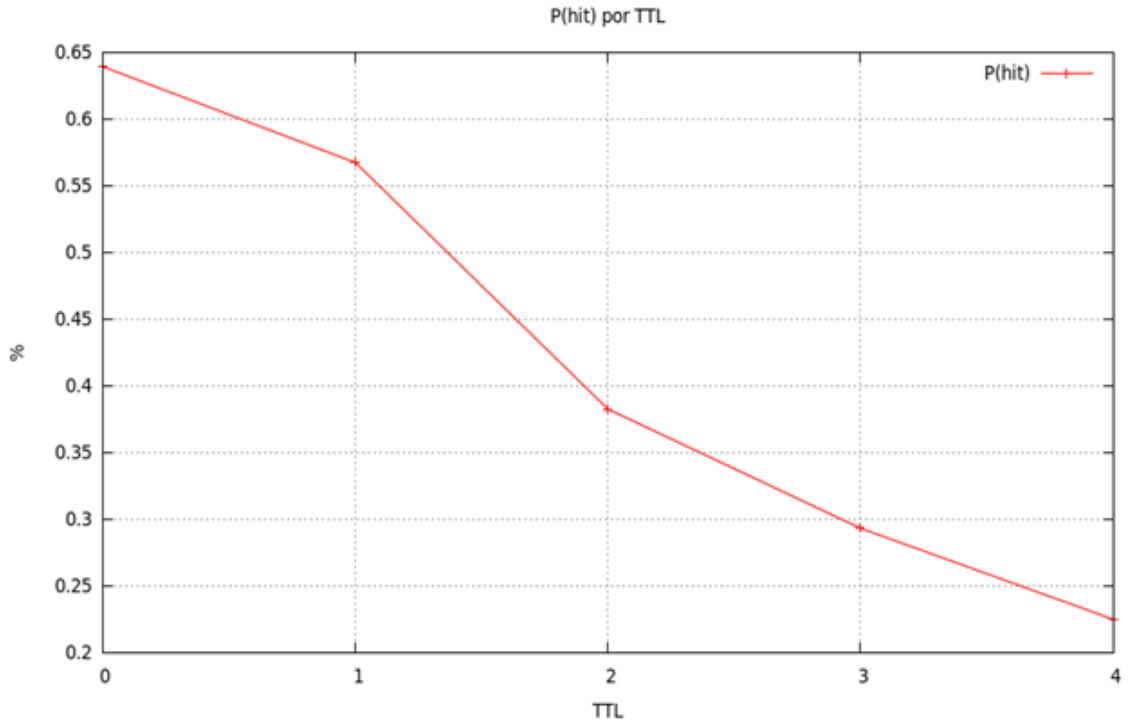


Figura 3.6: Probabilidade de encontrar o conteúdo ($p(hit)$) por TTL.

No próximo capítulo iremos mostrar mais resultados dessa versão, utilizando outras topologias na simulação.

Capítulo 4

Resultados

Neste capítulo apresentamos os resultados obtidos com o simulador. Consideramos duas topologias: uma topologia simétrica (que é uma das hipóteses consideradas no modelo proposto para a arquitetura estudada [13, 14]) e uma topologia real baseada na topologia da Rede Nacional de Ensino e Pesquisa (RNP). As principais métricas estimadas pelo simulador são a carga filtrada e o tempo de resposta para o atendimento de uma requisição.

A carga filtrada em um domínio é o percentual de requisições que encontram o conteúdo naquele domínio e portanto não são encaminhadas para outro domínio. Porém, esse percentual é calculado pelo número de pacotes que são filtrados no domínio sobre o total de pacotes gerados pelo sistema e não pelo total de pacotes que chegam no domínio. Enquanto a carga filtrada total é o percentual de requisições que encontram o conteúdo em um dos domínios, ou seja, que não são encaminhadas para a área de publicação. Escolhemos calcular a carga filtrada pois existem duas vantagens importantes em filtrar as requisições: uma é que esta filtragem diminui a carga na área de publicação o que diminui o custo; a segunda é que a filtragem pode diminuir significativamente o tempo de resposta.

Para todos os cenários foram realizadas 5 rodadas e foi estimado um intervalo de confiança de 90% com uma variação de no máximo 12% do valor da média.

4.1 Cenários de Simulações

Com relação ao mecanismo de passeio aleatório para buscar o conteúdo, consideramos dois casos: um passeio aleatório lento (*slow walker*) e um passeio aleatório rápido (*fast walker*). A característica do passeio aleatório lento é que o tempo médio que uma requisição permanece em um nó é quase igual ao tempo médio para o decréscimo do valor do RC. Isto significa que enquanto o passeio é executado em um domínio, como o valor do RC pode mudar, o conteúdo também pode ser retirado da cache do roteador, logo o estado das caches dos roteadores pode mudar durante o passeio.

A característica do passeio aleatório rápido é que o tempo médio que uma requisição permanece em um nó é muito menor que o tempo médio para o decréscimo do valor do RC. Isto significa que enquanto o passeio é executado em um domínio, como o valor do RC não deve mudar, os conteúdos que estão armazenados nas caches devem permanecer durante o percurso da requisição no domínio. O objetivo do estudo dos passeios aleatórios rápido e lento é avaliar qual a influência de manter o conteúdo *fixo* nos roteadores durante o passeio ou permitir que esses conteúdos se alterem durante o passeio em um domínio. No modelo proposto em [13, 14] foi considerada a hipótese de passeio aleatório rápido, ou seja, os conteúdos pemanecem fixos durante o passeio. Portanto, achamos importante avaliar os dois casos com o simulador.

Foram avaliados também cenários com cache finita e cache infinita. No caso da cache finita, duas políticas foram consideradas para remoção dos conteúdos da cache: retirar um conteúdo aleatoriamente ou retirar o conteúdo com menor valor do contador RC, que supostamente é o conteúdo menos requisitado dentre todos os existentes na cache. A substituição de um conteúdo ocorre toda vez que a cache está cheia e o valor do RC de um outro conteúdo atingir o limite superior. O principal objetivo desses cenários foi avaliar como a limitação da cache pode influenciar as métricas obtidas dado que no modelo proposto em [13, 14] foi considerada a hipótese de cache infinita.

A seguir enumeramos os cenários avaliados:

Cenário Cache Infinita e Passeio Aleatório Lento: Todos os roteadores possuem uma cache de tamanho infinito, ou seja, todo o conteúdo para o qual o limite superior do contador RC for atingido, será armazenado. Além disso, possui a característica de passeio aleatório lento.

Cenário Cache Infinita e Passeio Aleatório Rápido: Todos os roteadores possuem uma cache de tamanho infinito e é realizado um passeio aleatório rápido.

Cenário Cache Finita com Substituição Aleatória e Passeio Lento: Todos os roteadores possuem uma cache de tamanho limitado, ou seja, apenas uma quantidade limitada de conteúdos pode ser armazenada. A política de substituição de conteúdos é aleatória e o passeio aleatório é lento.

Cenário Cache Finita com Substituição Aleatória e Passeio Rápido: Todos os roteadores possuem uma cache de tamanho limitado, a política de substituição de conteúdos é aleatória e o passeio aleatório é rápido.

Cenário Cache Finita com Substituição pelo menor RC e Passeio Lento: Todos os roteadores possuem uma cache de tamanho limitado, a política de substituição é a do conteúdo com menor valor de RC e o passeio aleatório é lento.

Cenário Cache Finita com Substituição pelo menor RC e Passeio Rápido: Todos os roteadores possuem uma cache de tamanho limitado, a política de substituição é a do conteúdo com menor valor de RC e o passeio aleatório é rápido.

A tabela 4.1 mostra quais os parâmetros utilizados nas simulações.

4.2 Resultados da Topologia Simétrica

Nessa topologia consideramos dois domínios: o primeiro domínio é composto por dois sub-domínios, e o segundo por um único sub-domínio, como ilustrado na figura 4.1.

Tabela 4.1: Parâmetros de Simulação

Parâmetro	Símbolo	Valor
Taxa de chegada dos pedidos	λ	25
Taxa de transmissão dos roteadores	μ	26 (lento) e 150 (rápido)
Taxa de decréscimo do RC	γ	14,47
Parâmetro da Zipf para a escolha dos conteúdos	α	1,2
Limite do RC	-	5
Tamanho da cache (quando consideramos cache finita)	-	3
Quantidade de tipos de conteúdos	-	10

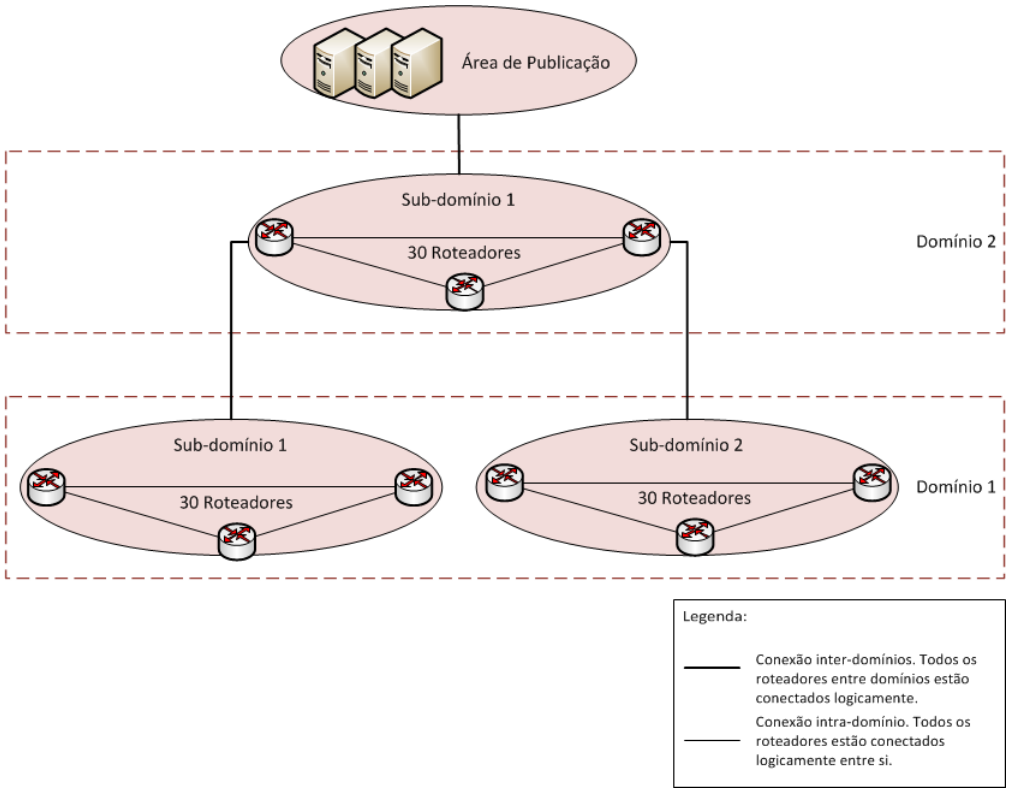


Figura 4.1: Topologia Simétrica.

Com esta configuração é possível observar como a agregação de fluxos no segundo domínio pode aumentar a chance do conteúdo ser armazenado e recuperado mais rapidamente. Cada sub-domínio possui um total de 30 roteadores, onde todos os roteadores do mesmo sub-domínio estão conectados entre si e conectados com todos os roteadores do sub-domínio superior, ou seja, topologia *Full-mesh*. Cabe ressaltar que a conexão entre os roteadores é lógica.

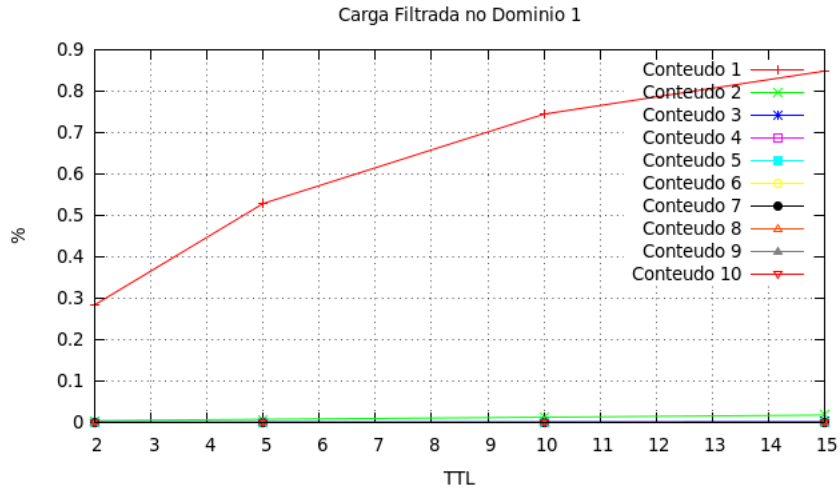
O objetivo principal desse experimento é avaliar o *trade-off* entre o tempo que uma requisição percorre um domínio, ou seja, o TTL (*Time To Live*), a carga filtrada e o tempo para achar um conteúdo (tempo de resposta total) para uma topologia semelhante a considerada no modelo proposto para a arquitetura estudada [13, 14]. Vamos considerar a cache infinita, o passeio aletório rápido e o TTL igual a 2, 5, 10 e 15.

As figuras 4.2 e 4.3 apresentam a carga filtrada em cada domínio, a carga filtrada total e o tempo de resposta total.

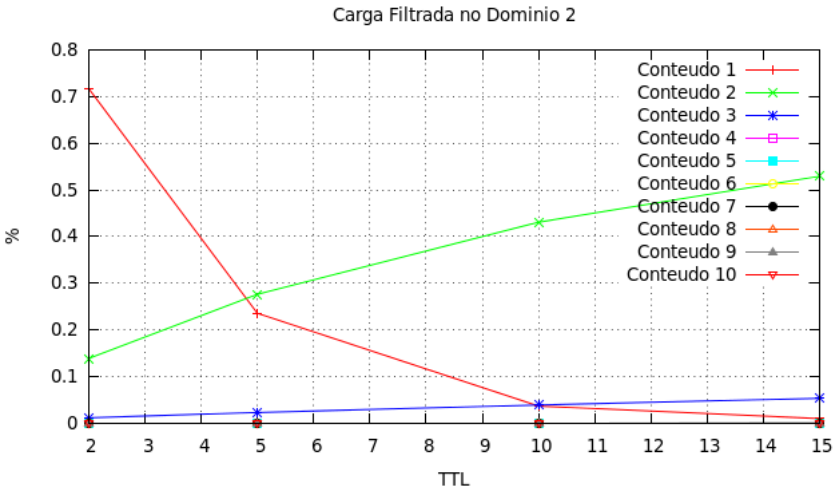
Na figura 4.2(a), podemos notar que apenas o conteúdo #1 está armazenado no primeiro domínio e portanto é filtrado, pois ele é o mais popular. Já na figura 4.2(a), podemos notar que no segundo domínio, devido a agregação de fluxo dos dois sub-domínios inferiores, os conteúdos #2 e #3 passam a ser armazenados e portanto parte das requisições é filtrada e o restante é encaminhado para a área de publicação.

Na figura 4.3(a) podemos notar que a carga filtrada dos conteúdos diminui à medida que a popularidade dos conteúdos diminuem. E na figura 4.3(b) podemos notar que a partir do conteúdo #4 o tempo de resposta se mantém constante. Esse tempo de resposta constate se deve ao fato de que as requisições dos conteúdos acima do #4 são resolvidos na Área de Publicação, assim mantendo o mesmo tempo de resposta.

Os conteúdos do #4 ao #10 não são armazenados em nenhum domínio pois são pouco populares. Como era esperado a carga filtrada do conteúdo #1 aumenta com o TTL para o domínio 1. Já para o domínio 2, a carga filtrada do conteúdo #1 diminui com o TTL pois somente as requisições que não forem filtradas no domínio 1 serão encaminhadas para o domínio 2.

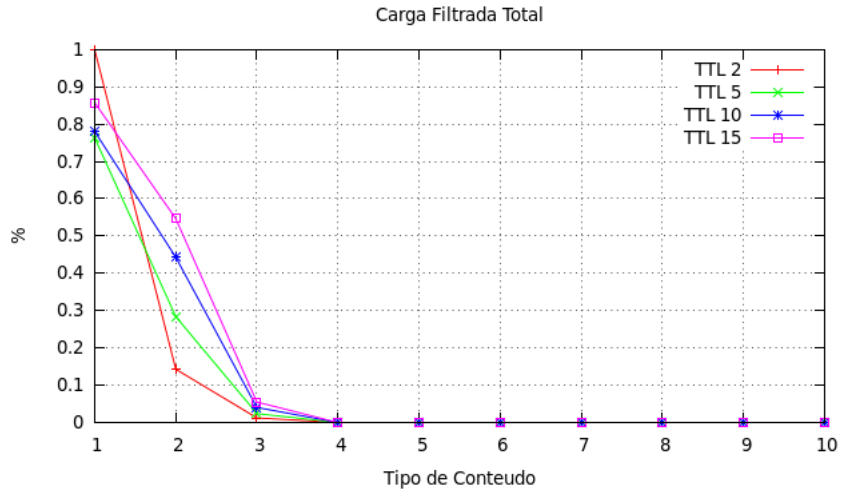


(a) Carga filtrada no primeiro domínio.

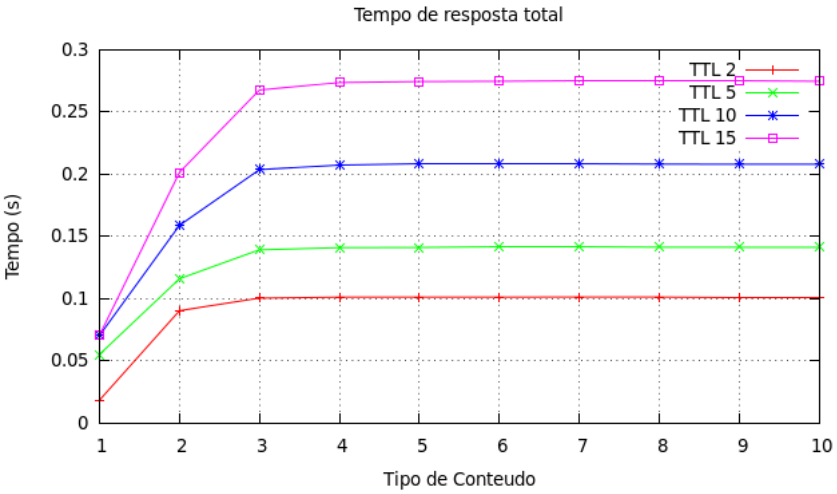


(b) Carga filtrada no segundo domínio.

Figura 4.2: Carga filtrada por domínio.



(a) Carga filtrada total.



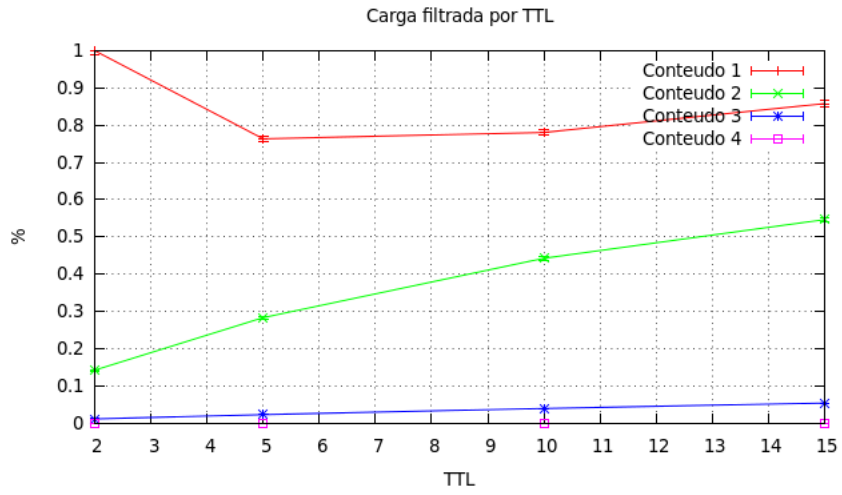
(b) Tempo de resposta total.

Figura 4.3: Carga filtrada e tempo de resposta.

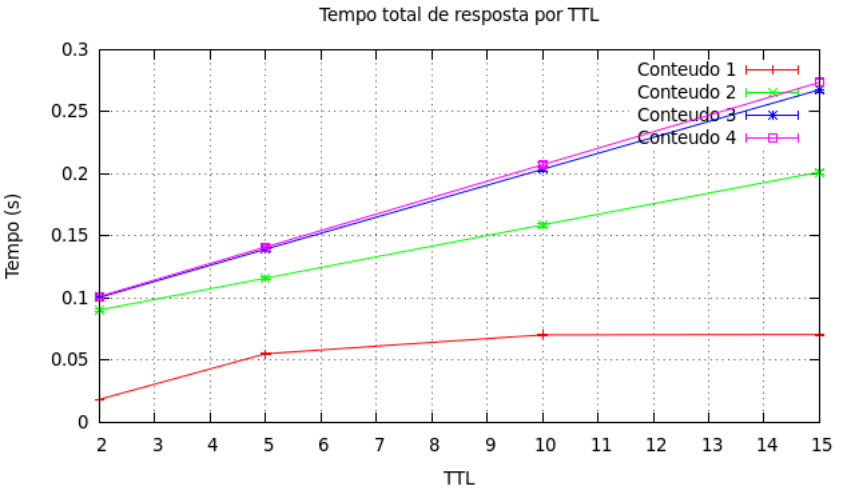
A figura 4.4 mostra com mais detalhes a carga filtrada total e o tempo de resposta para os conteúdos mais populares, com a variação do TTL. Podemos notar que a carga filtrada total do conteúdo #1 para o TTL=2 é igual a 100%. Já para os outros TTLs maiores do que 2, a carga filtrada teve uma diminuição significativa, o que não é esperado. Porém este comportamento pode ser explicado pelos gráficos da carga filtrada por domínio (Figura 4.2). Para o caso do TTL=2, somente 30% das requisições para o conteúdo #1 são filtradas no primeiro domínio. Com isso, a probabilidade do conteúdo #1 ser armazenado no segundo domínio é função da agregação de 70% das requisições de cada sub-domínio do domínio 1. Para o caso do TTL=5, um pouco mais de 50% das requisições para o conteúdo #1 são filtradas

no primeiro domínio. Com isso, a probabilidade do conteúdo #1 ser armazenado no segundo domínio é função da agregação de aproxidamente 50% das requisições de cada sub-domínio do domínio 1. E esse mesmo comportamento se repete para outros valores de TTL: quanto maior é o percentual de carga filtrada no primeiro domínio, menor é taxa de requisições agregadas que chega no segundo domínio e portanto menor é a probabilidade do conteúdo ser armazenado no segundo domínio.

A partir dos resultados desse experimento, podemos concluir que não é vantajoso aumentar o TTL para as cargas mais populares que são filtradas no primeiro domínio pois isso pode diminuir a carga filtrada total além de aumentar o tempo de resposta.



(a) Carga filtrada total por TTL.



(b) Tempo de resposta total por TTL.

Figura 4.4: Carga e tempo de resposta para os conteúdos mais populares em uma topologia simétrica.

4.3 Resultados da Topologia da RNP

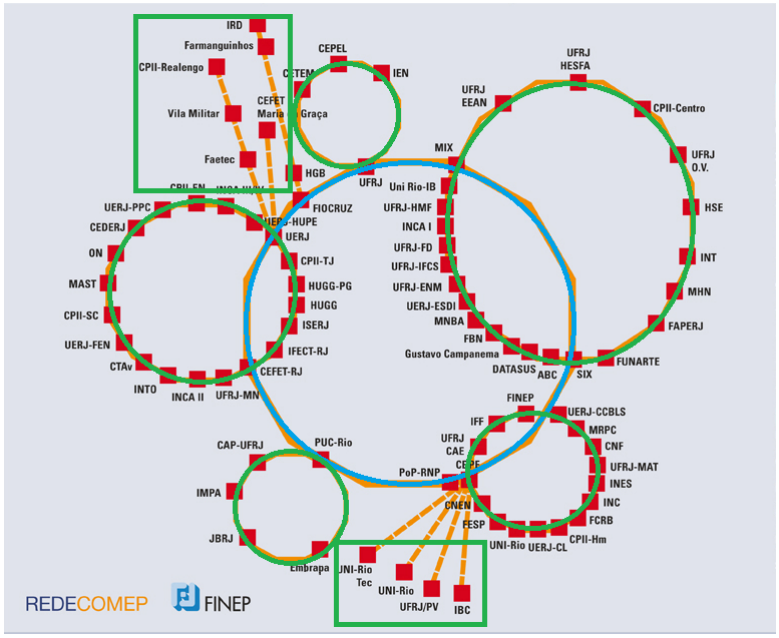
Nestes experimentos iremos considerar a topologia de uma rede real. Escolhemos a Rede Nacional de Ensino e Pesquisa (RNP) [22] por ser a rede usada pelas instituições de ensino e pesquisa do Brasil.

Para as simulações utilizamos apenas uma parte da RNP que compreende as regiões metropolitanas do Rio de Janeiro, São Paulo e Belo Horizonte, como ilustrado na figura 4.5. A topologia ICN foi criada da seguinte maneira:

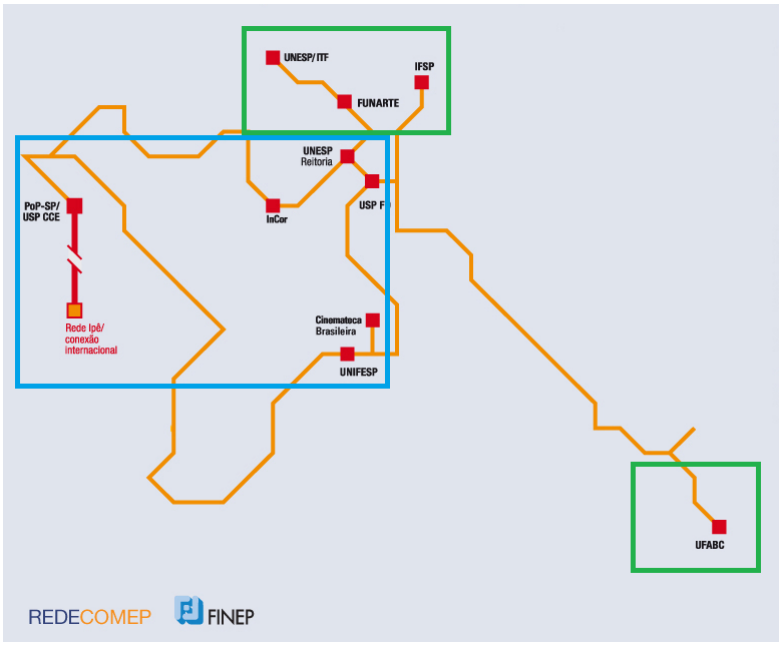
- Para cada marcação em verde foi criado um sub-domínio do primeiro domínio;
- Para cada marcação em azul foi criado um sub-domínio do segundo domínio;
- O terceiro domínio é composto pelos PoPs da RNP do Rio de Janeiro, São Paulo e Belo Horizonte;
- O terceiro domínio está conectado diretamente com a Área de Publicação.

Podemos notar que a topologia de algumas partes da rede é em anel. O algoritmo de passeios aleatórios pode ser muito ineficiente para este tipo de topologia, portanto incluímos alguns atalhos (conexões lógicas) entre os roteadores, de forma a criar uma topologia em malha. Os nós origem e destino de cada atalho foram escolhidos de forma aleatória.

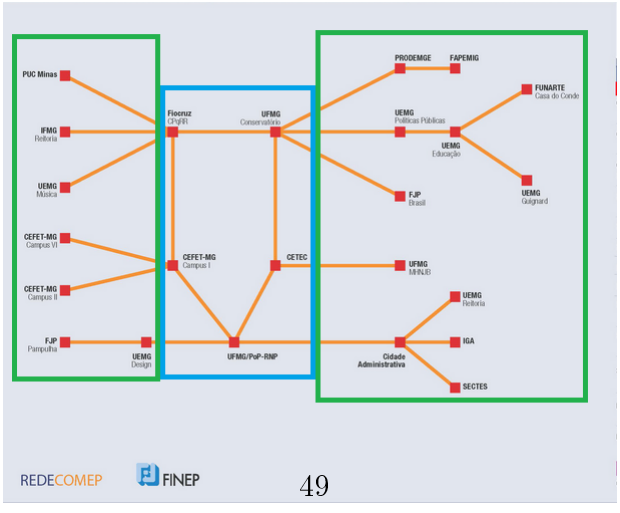
O objetivo principal desses experimentos é avaliar o *trade-off* entre o tempo que uma requisição percorre um domínio, ou seja, o TTL (*Time To Live*), a carga filtrada e o tempo para achar um conteúdo (tempo de resposta total) para uma topologia real. Iremos também avaliar como a limitação da cache do roteador e as políticas de substituição de conteúdo nas caches afetam as métricas de desempenho do sistema.



(a) Rede Metropolitana do Rio de Janeiro



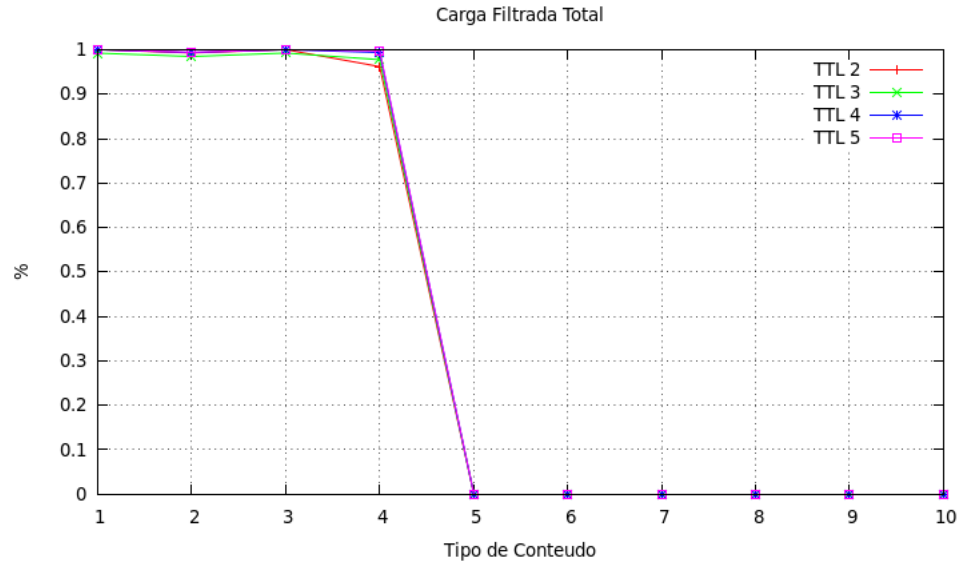
(b) Rede Metropolitana de São Paulo



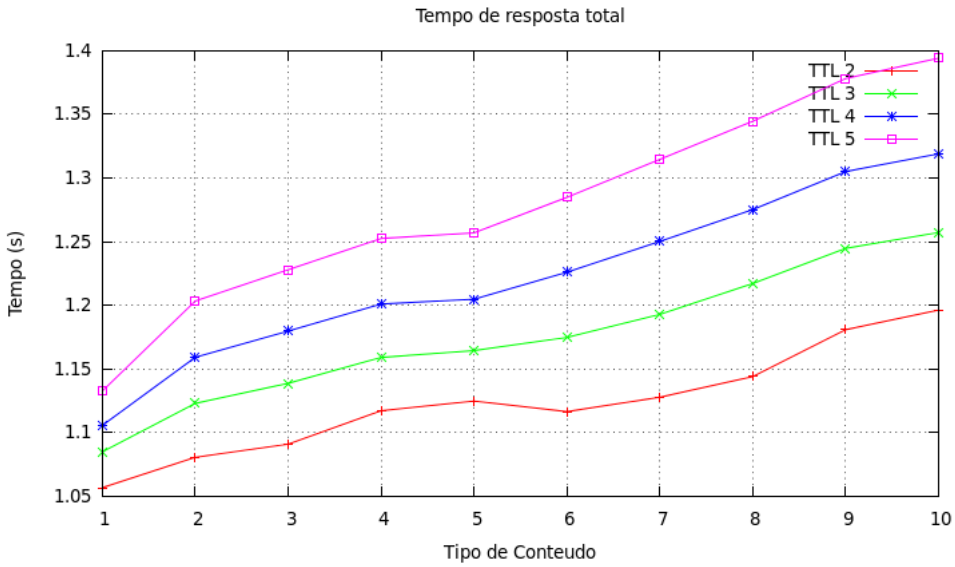
(c) Rede Metropolitana de Belo Horizonte

4.3.1 Cenários para Cache Infinita

As figuras 4.6 e 4.7 mostram os resultados para o passeio rápido e passeio lento. A partir dos gráficos 4.6(a) e 4.7(a) observamos que a grande maioria das requisições pelos conteúdos #1, #2, #3 e #4 é filtrada nos domínios e não chega a área de publicação. Esses conteúdos possuem alta probabilidade de estarem armazenados nas caches dos roteadores pois a taxa de chegada de requisições para esses conteúdos é bastante alta. Cabe observar que não existe diferença significativa entre a carga filtrada para o cenário de passeio aleatório lento e a carga filtrada para o passeio aleatório rápido. Só existe diferença entre os dois cenários com relação ao tempo de resposta pois no passeio lento as requisições levam mais tempo para percorrer o domínio, portanto o tempo de resposta é maior.

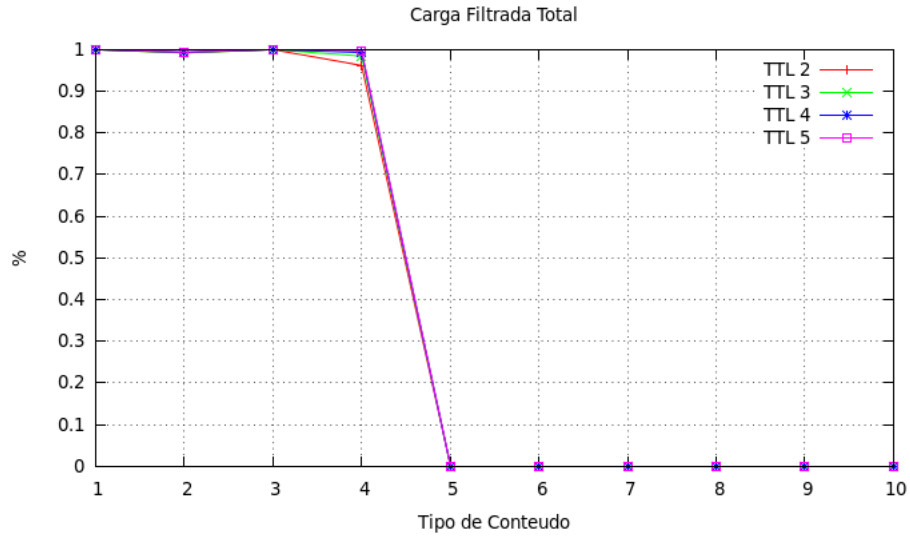


(a) Carga filtrada total

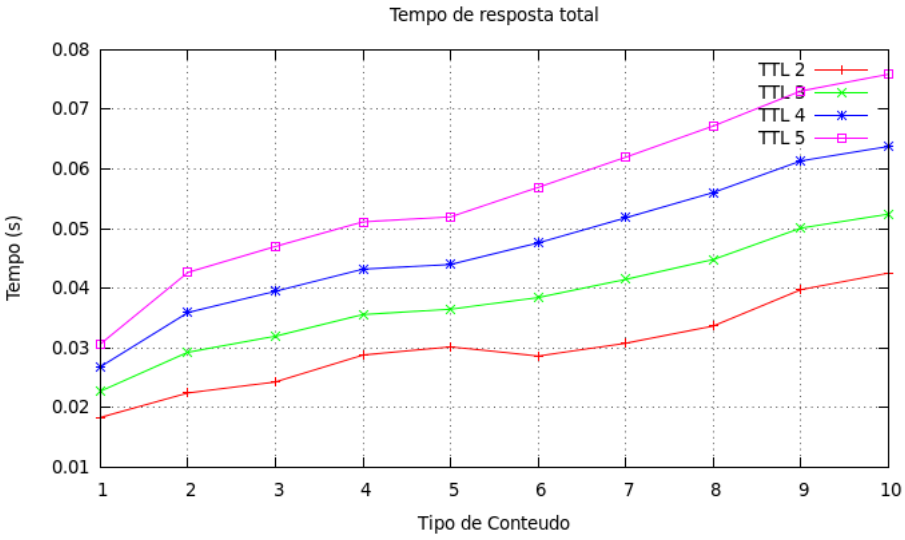


(b) Tempo total de resposta ao usuário

Figura 4.6: Cenário de cache infinita e passeio aleatório lento.



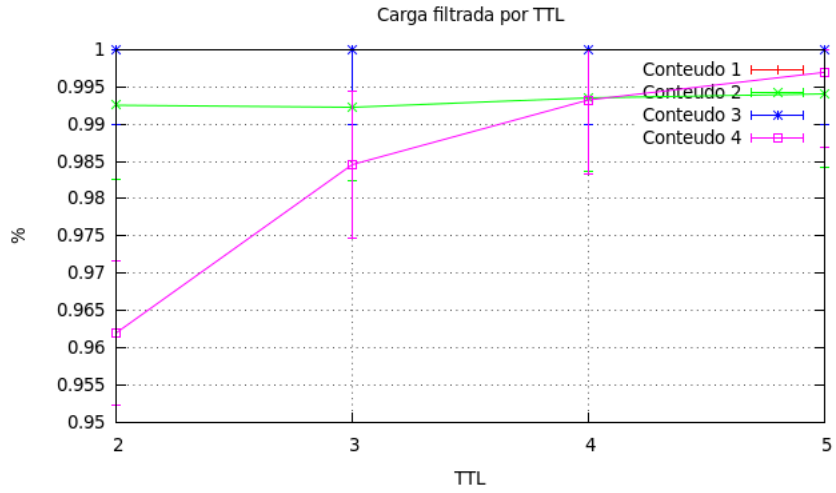
(a) Carga filtrada total



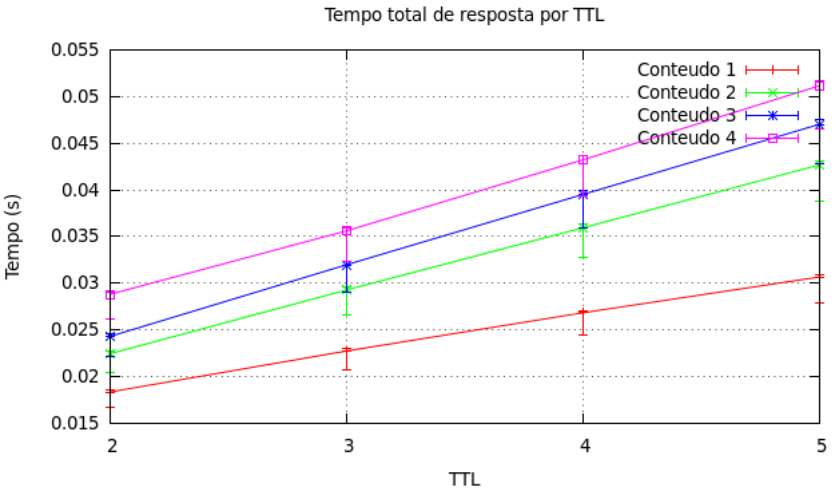
(b) Tempo total de resposta ao usuário

Figura 4.7: Cenário de cache infinita e passeio aleatório rápido.

A figura 4.8 permite visualizar melhor as métricas obtidas para os conteúdos mais populares. Podemos observar que para os conteúdos #1, #2, #3, o melhor TTL é o menor pois a carga filtrada não aumenta com o TTL. Neste caso, aumentar o TTL, aumentaria o tempo de resposta sem nenhum ganho com relação a carga filtrada. Já para o conteúdo #4, a carga filtrada aumenta com o TTL. Logo, existe vantagem no aumento do TTL, pois, menos requisições serão encaminhadas para a área de publicação.



(a) Carga filtrada total para os conteúdos mais populares



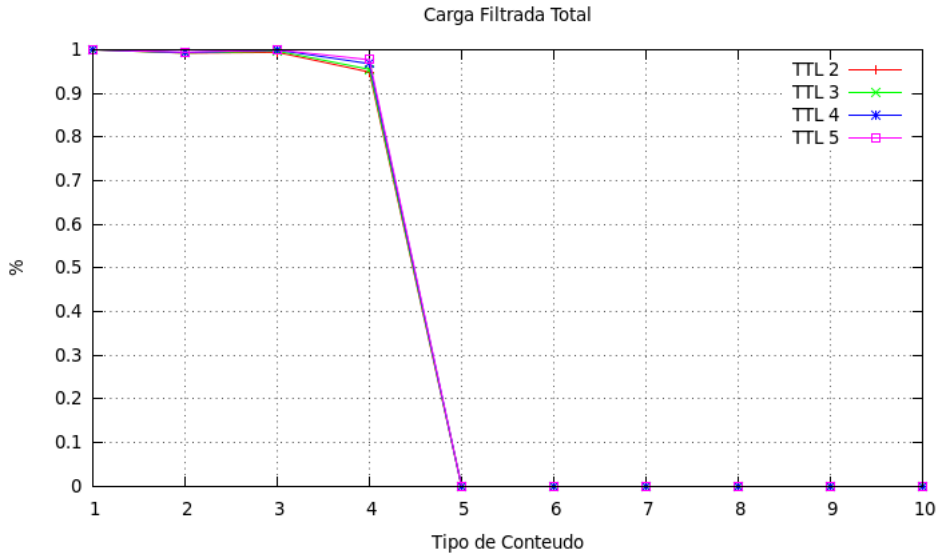
(b) Tempo total de resposta ao usuário para os conteúdos mais populares

Figura 4.8: Cenário de cache infinita e passeio aleatório rápido: conteúdos mais populares.

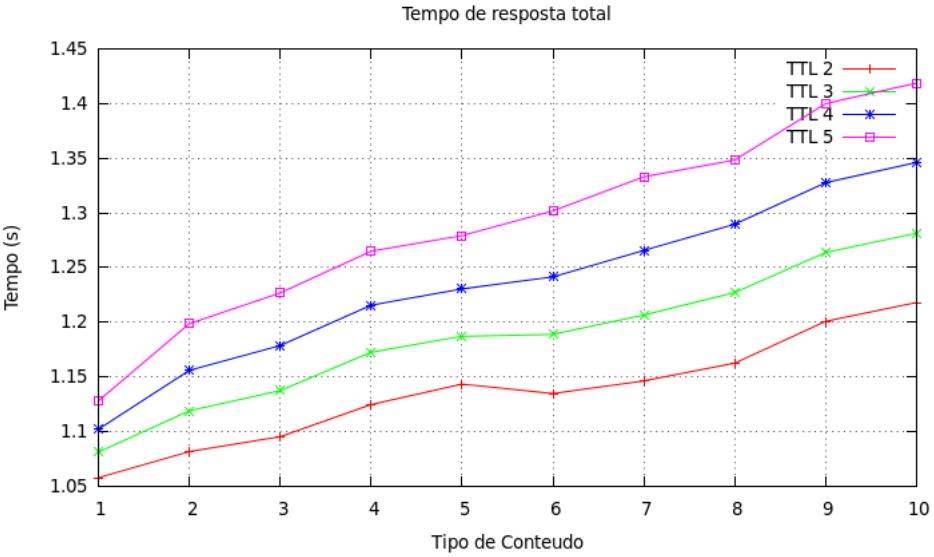
4.3.2 Cenário para Cache Finita

Os gráficos 4.9 e 4.10 mostram os resultados para o passeio rápido e passeio lento para uma política de substituição aleatória. E os gráficos 4.11 e 4.12 mostram os resultados para o passeio rápido e passeio lento para uma política de substituição baseada no menor RC. Pode-se notar que a carga filtrada total é praticamente igual para todos os cenários, indicando que as políticas de substituição não fazem diferença para esses cenários. Comparando-se a carga filtrada total quando a cache é infinita com a carga filtrada total quando a cache é finita, não observa-se dife-

rença significativa nos valores. Isto significa que limitar o tamanho da cache não diminuiu a carga filtrada conforme esperado. Este comportamento pode ser explicado através da seguinte observação: consideramos uma cache com três posições nos roteadores e somente quatro conteúdos possuem popularidade que justifique o seu armazenamento nos domínios. Logo, as caches nos roteadores possuem espaço para armazenar os conteúdos mais populares durante a maior parte do tempo. Esta observação também explica o fato da carga filtrada ser a mesma para as duas políticas de substituição consideradas.

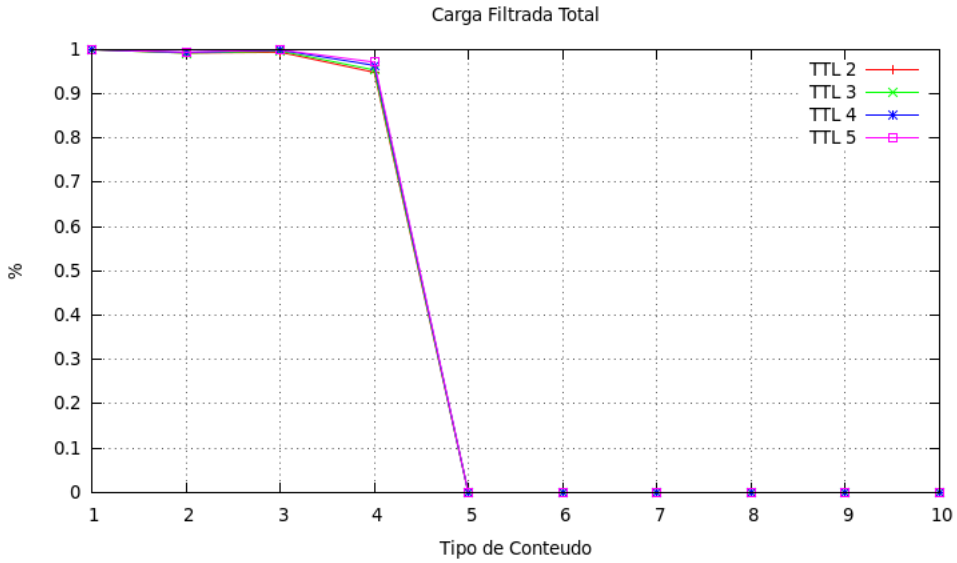


(a) Carga filtrada total

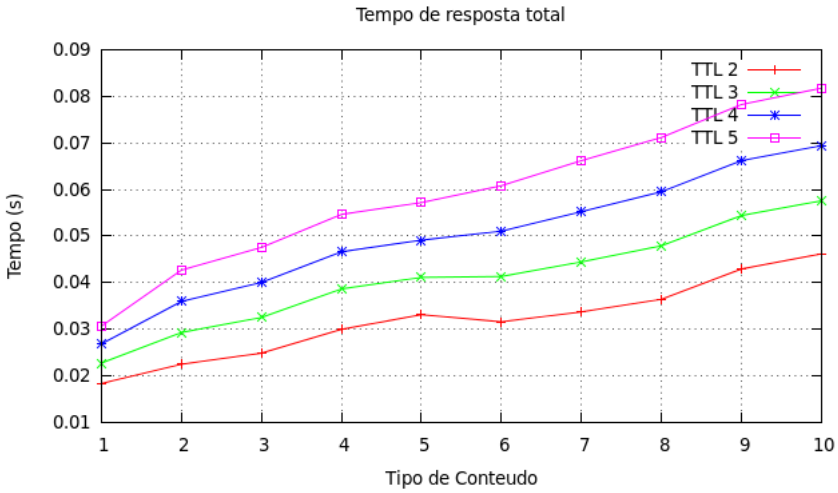


(b) Tempo total de resposta ao usuário

Figura 4.9: Cenário de cache finita com política de substituição aleatória para passeio aleatório lento.

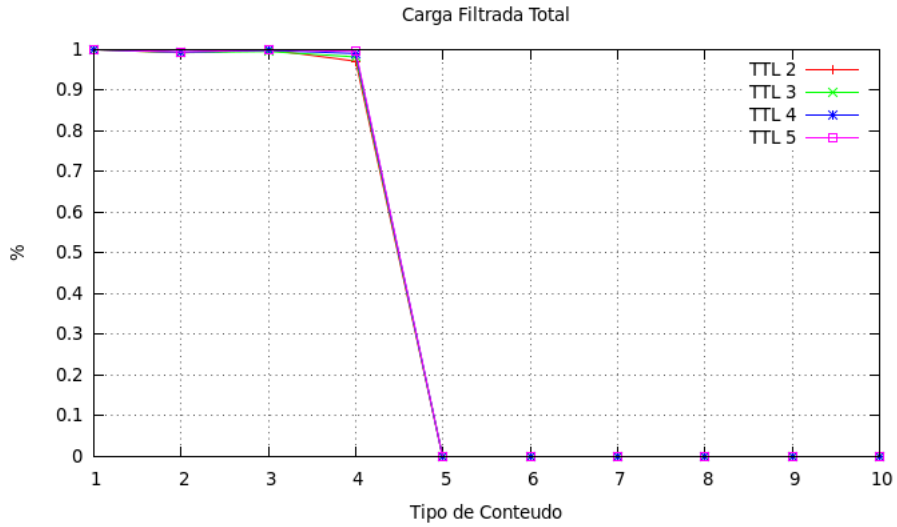


(a) Carga filtrada total

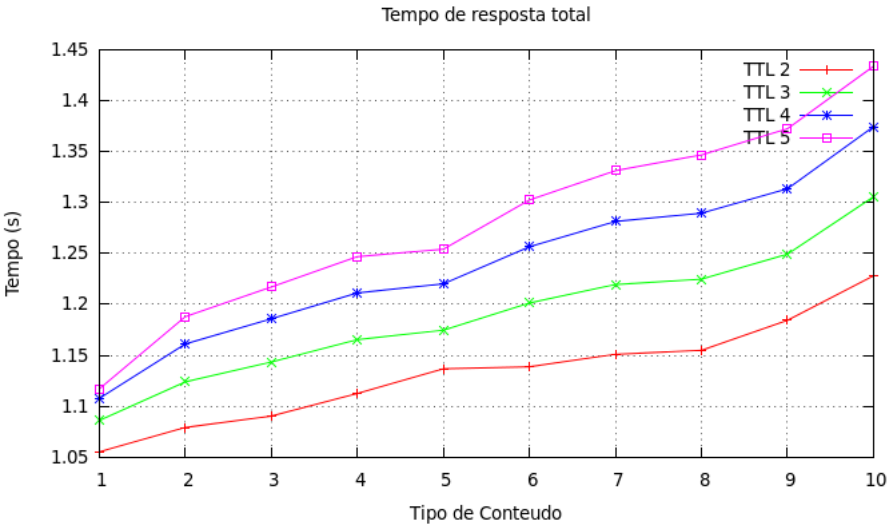


(b) Tempo total de resposta ao usuário

Figura 4.10: Cenário de cache finita com política de substituição aleatória para passeio aleatório rápido.

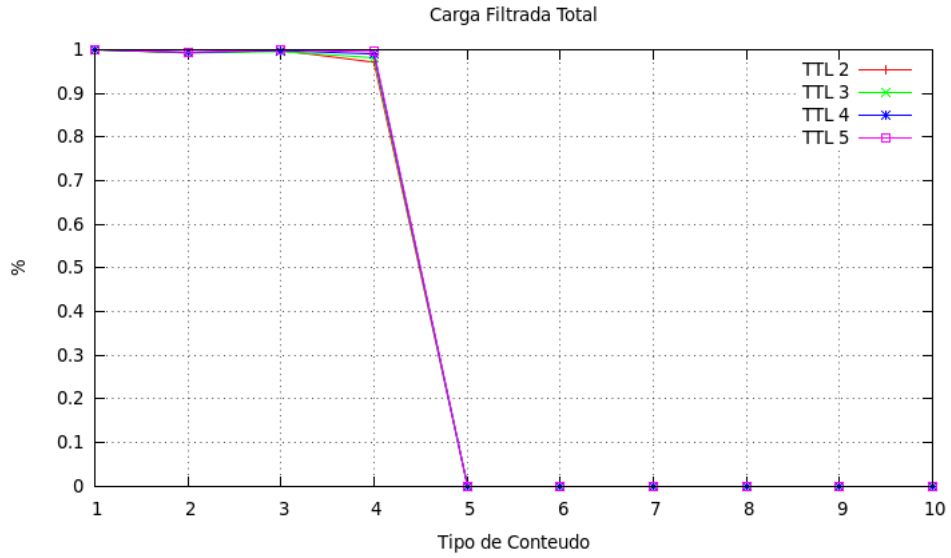


(a) Carga filtrada total

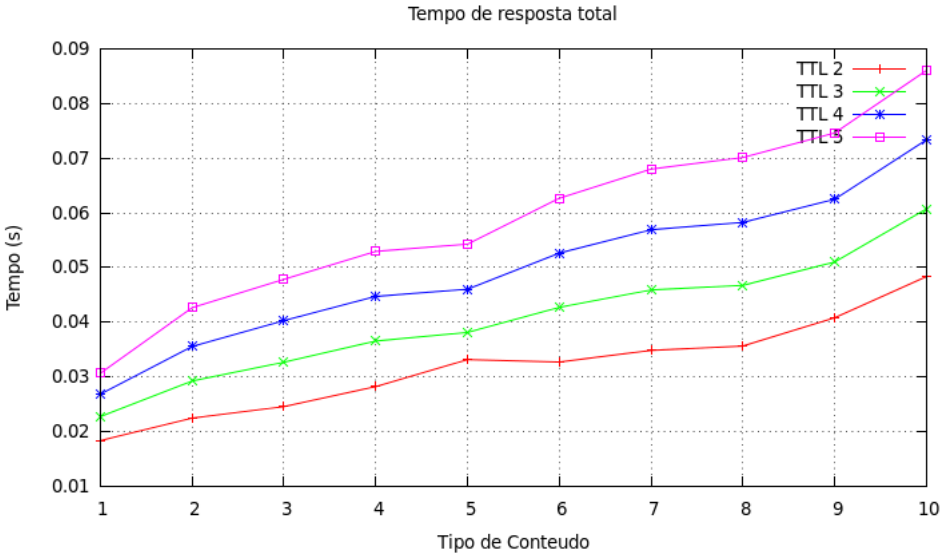


(b) Tempo total de resposta ao usuário

Figura 4.11: Cenário de cache finita com política de substituição do menor RC para passeio aleatório lento.



(a) Carga filtrada total



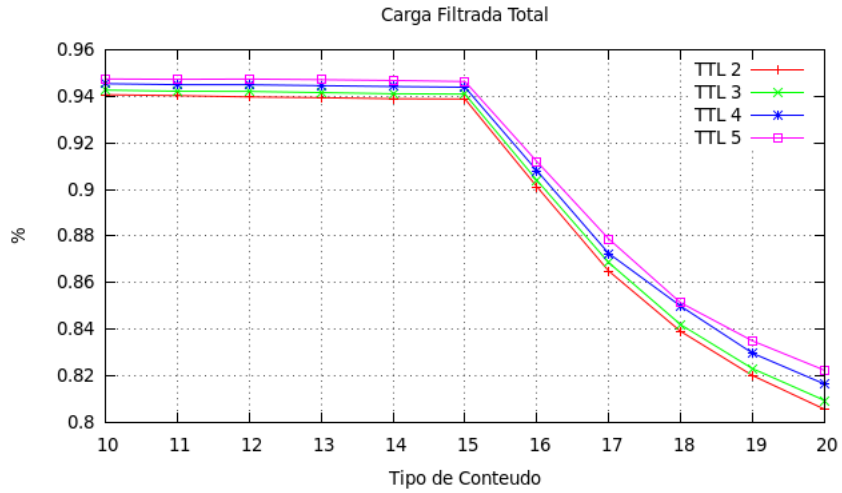
(b) Tempo total de resposta ao usuário

Figura 4.12: Cenário de cache finita com política de substituição do menor RC para passeio aleatório rápido.

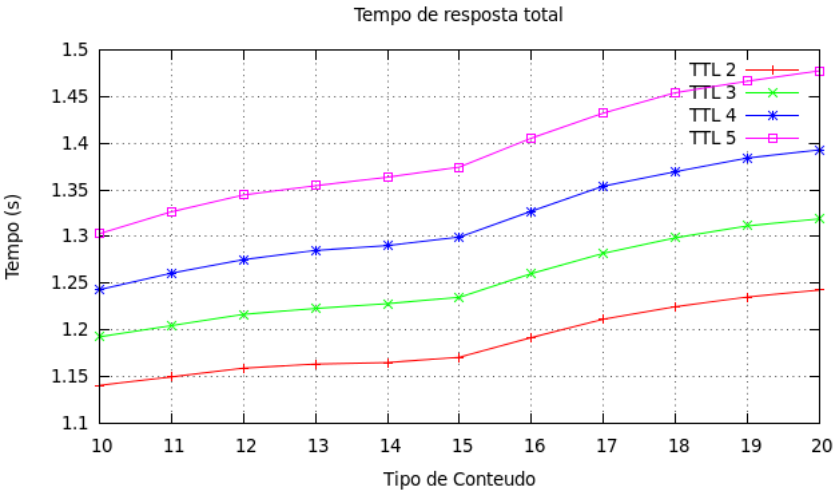
Para observar melhor qual o impacto da limitação da cache, simulamos mais dois cenários: um com a cache de tamanho 1 e política de substituição do menor RC e outro com a cache infinita, todos dois para passeio aleatório lento. Diferentemente dos cenários anteriores, neste cenário diminuímos o parâmetro da Zipf para 1, com o objetivo de distribuir melhor a popularidade entre os conteúdos. Com esta modificação diminuímos a popularidade dos primeiros conteúdos e por consequên-

cia aumentamos a dos últimos. Logo a quantidade de conteúdos com popularidade que justifique o armazenamento na cache irá aumentar. Além disso, aumentamos a quantidade de conteúdos para 20. Ambas as alterações tem o objetivo de aumentar o número de conteúdos a serem armazenados na cache.

As figuras 4.13 e 4.14 mostram os resultados obtidos para os conteúdos menos populares pois para os mais populares não existe diferença significativa entre a carga filtrada para cada um dos cenários. Podemos notar que a carga filtrada dos conteúdos menos populares diminui com a limitação da cache, como era esperado. Esta diminuição da carga aumenta com a diminuição da popularidade. Ou seja, quanto menos popular é um conteúdo, maior impacto terá a limitação da cache na carga filtrada. Por exemplo, o conteúdo #20 que é o menos popular, tem 82% da carga filtrada quando a cache é limitada e 99% da carga filtrada quando a cache é infinita. Este comportamento pode ser explicado através da política de substituição que retira o conteúdo da cache com o menor valor de RC, ou seja, o menos popular. Quanto ao tempo de resposta, ele é maior para o cenário de cache finita pois um número maior de requisições serão encaminhadas para a área de publicação, aumentando o tempo de resposta.

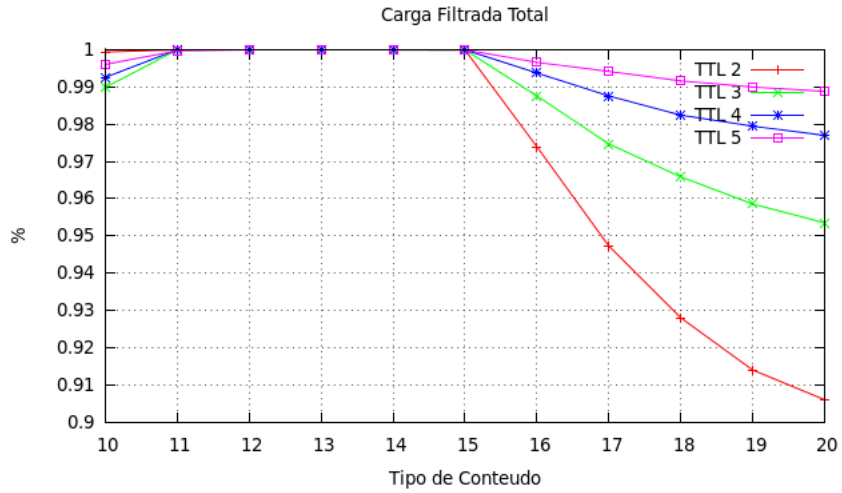


(a) Carga filtrada total.

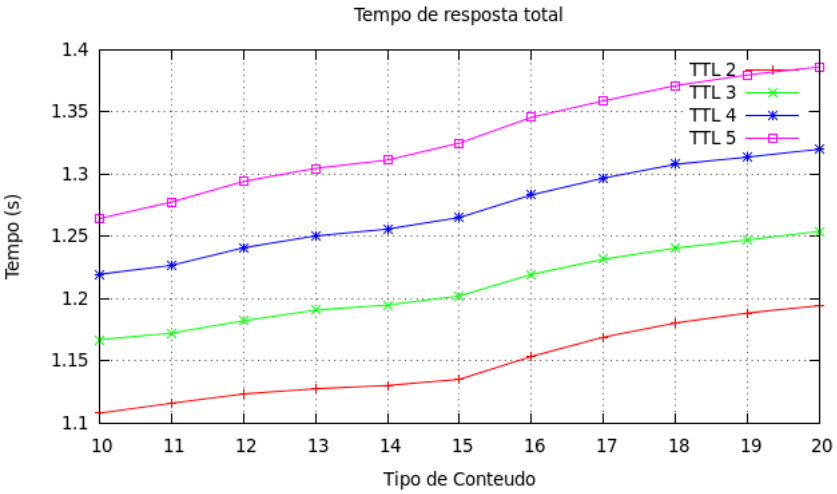


(b) Tempo de resposta total.

Figura 4.13: Cenário de cache finita de tamanho 1.



(a) Carga filtrada total.



(b) Tempo de resposta total.

Figura 4.14: Cenário de cache infinita.

Capítulo 5

Conclusão

Redes orientadas a informação estão no foco das pesquisas atualmente pois os estudos da literatura têm mostrado o potencial dessa arquitetura para resolver alguns dos problemas da Internet atual.

Neste trabalho desenvolvemos um simulador para uma proposta de arquitetura ICN onde a busca pelo conteúdo é aleatória e o armazenamento do conteúdo é baseado na sua popularidade. Esse simulador foi desenvolvido e validado de forma modular, onde para cada versão um módulo do algoritmo de busca foi implementado e validado.

As principais métricas estimadas pelo simulador são: o percentual de requisições por conteúdo atendidas pelos roteadores da rede (carga filtrada) e o tempo para encontrar um conteúdo (tempo de resposta).

Consideramos diversos cenários de simulação onde procuramos *relaxar* as hipóteses consideradas no modelo analítico da arquitetura estudada. Dessa forma foi possível avaliar o impacto das hipóteses consideradas no desempenho da arquitetura, como topologia simétrica, cache infinita e passeio aleatório rápido.

Usamos a topologia da RNP nos nossos experimentos considerando cache finita e cache infinita nos roteadores e também passeio aleatório rápido e lento. No caso da cache finita, duas políticas foram consideradas para remoção dos conteúdos da cache: retirar um conteúdo aleatoriamente ou retirar o conteúdo com menor valor do contador RC, que supostamente é o conteúdo menos requisitado dentre todos os existentes na cache.

Através dos resultados obtidos com os experimentos pudemos observar que a

carga filtrada dos conteúdos menos populares diminui com a limitação da cache. Observamos também que quanto menos popular é um conteúdo, maior impacto terá a limitação da cache na carga filtrada. Quanto ao tempo de resposta, ele é maior para o cenário de cache finita pois um número maior de requisições serão encaminhadas para a área de publicação, aumentando o tempo de resposta. Portanto, o fato da cache ser limitada impacta as métricas dos conteúdos menos populares. Quanto ao passeio aleatório lento ou rápido, não observamos nenhuma diferença significativa nas métricas obtidas com ambos para os cenários simulados.

Além disso, para a topologia simétrica pudemos observar que, em alguns casos, o aumento do TTL pode diminuir a carga filtrada total, pois o aumento da filtragem de um conteúdo no primeiro domínio pode diminuir o efeito da agregação no segundo domínio.

Por fim, obtivemos dois resultados interessantes durante a validação do simulador. O primeiro é uma expressão para o cálculo da taxa total de chegada de requisições nos roteadores provenientes do domínio inferior (taxa exógena) e geradas pelo passeio aleatório (taxa endógena). O segundo é que como a taxa total é calculada em função da probabilidade p de encontrar o conteúdo e do TTL, podemos obter valores para esses parâmetros de forma a que seja mantido o equilíbrio nas filas dos roteadores ($\lambda_{total} < \mu$).

Como trabalhos futuros pretendemos avaliar outras topologias assim como incluir outras cidades na topologia da RNP. Pretendemos também comparar os resultados do modelo analítico da arquitetura proposta com os resultados do simulador.

Referências Bibliográficas

[1] KLEINROCK, L., *Queueing Systems, Volume I*. Wiley-Interscience, 1975.

[2] "Cisco, Cisco Visual Networking Index, 2012", `www.cisco.com/web/solutions/sp/vni/vni_forecast_highlights/index.html`.

[3] "Akamai", 2011, `http://www.akamai.com`.

[4] BALACHANDRAN, A., SEKAR, V., AKELLA, A., et al., "Analyzing the Potential Benefits of CDN Augmentation Strategies for Internet Video Workloads". In: *Proceedings of the 2013 ACM Internet Measurement Conference, IMC '13*, pp. 43–56, ACM, 2013.

[5] G. CAROFIGLIO, G. MORABITO, L. M. I. S. M. V., "From content delivery today to information centric networking", *Computer Networks*, v. 57, pp. 3116–3127, 2013.

[6] HAJEK, B., ZHU, J., "The missing piece syndrome in peer-to-peer communication". In: *Proceedings of the 2010 IEEE International Symposium on Information Theory (ISIT)*, pp. 1748–1752, 2010.

[7] ZHU, J., HAJEK, B., "Stability of a peer-to-peer communication system", *IEEE Transactions on Information Theory*, v. 58, n. 7, pp. 4693–4713, 2012.

[8] MENASCHÉ, D., DE A ROCHA, A., DE SOUZA E SILVA, E., et al., "Implications of Peer Selection Strategies by Publishers on the Performance of P2P Swarming Systems", *Perform. Eval. Rev.*, v. 39, n. 3, pp. 55–57, 2011.

[9] DE SOUZA E SILVA, E., LEÃO, R., MENASCHÉ, D., et al., "Sobre a Capacidade de Serviço de Sistemas P2P". In: *Anais do 32º Simpósio Brasileiro de*

Redes de Computadores e Sistemas Distribuídos, SBRC'2014 , pp. 61–74, SBC, 2014.

[10] XYLOMENOS, G., VERVERIDIS, C. N., SIRIS, V. A., et al., "A Survey of Information-Centric Networking Research", *Communications Surveys and Tutorials*, v. 16, n. 2, pp. 1024–1049, 2013.

[11] M. DIALLO, V. SOURLAS, P. F. S. F. L. T., "A content based publish/subscribe framework for large-scale content delivery", *Computer Network*, v. 57, pp. 924–943, 2013.

[12] KUROSE, J., "Information-centric networking: The evolution from circuits to packets to content", *Computer Networks*, v. 66, pp. 112–120, 2014.

[13] DOMINGUES, G., DE SOUZA E SILVA, E., LEÃO, R., et al., "Enabling Information Centric Networks through Opportunistic Search, Routing and Caching". In: *XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pp. 135–148, 2013.

[14] DOMINGUES, G., DE SOUZA E SILVA, E., LEÃO, R., et al., "A name resolution assisted ICN design, supported by opportunistic search, routing and caching policies". In: *XIII Workshop em Desempenho de Sistemas Computacionais e de Comunicação (WPerf'14)*, aceito para publicação, 2014.

[15] ET AL., T. K., "A Data-Oriented Network Architecture", *SIGCOMM, Kyoto - Japan*, pp. 27–31, 2007.

[16] ET AL., M. A., "Architecture Definition, Component descriptions and Requirements", *Deliverable PSIRP*, 2009.

[17] ET AL., V. J., "Networking Named Content", *CoNext, Rome - Italy*, pp. 1–12, 2009.

[18] ET AL., B. A., "Second NetInf Architecture Description", *4Ward*, 2010.

[19] "Distribuição Exponencial", `http://en.wikipedia.org/wiki/Exponential_distribution`.

[20] "Distribuição Normal", http://en.wikipedia.org/wiki/Box-Muller_transform.

[21] L. BRESLAU, P. CAO, L. F. G. P., SHENKER, S., "Web Caching and Zipf-like Distributions: Evidence and Implications", *IEEE Infocom*, 1999.

[22] "Topologia RNP", http://www.rnp.br/.

[23] B. AHLGREN, C. DANNEWITZ, C. I. D. K. B. O., "A Survey of Information-Centric Networking", *IEEE*, v. 12, pp. 26–36, 2012.

[24] "Distribuição Uniforme", http://www.cplusplus.com/reference/cstdlib/rand.

[25] "Distribuição ZPIF", http://en.wikipedia.org/wiki/Zipf's_law.

[26] "Lei de Little", http://en.wikipedia.org/wiki/Little's_law.

[27] "Leonard Kleinrock Web page", 2014, http://www.lk.cs.ucla.edu/personal_history.html.

[28] "Rede Metropolitana", http://www.redecomep.rnp.br.

[29] NYGREN, E., SITARAMAN, R., SUN, J., "The Akamai Network: A Platform for High-performance Internet Applications", *SIGOPS Oper. Syst. Rev.*, v. 44, n. 3, pp. 2–19, 2010.

[30] DE SOUZA E SILVA, E., LEÃO, R. M. M., SANTOS, A. D., et al., "Multimedia Supporting Tools for the CEDERJ Distance Learning Initiative applied to the Computer Systems Course". In: *22th ICDE World Conference on Distance Education*, pp. 1–11, 2006.

[31] DE SOUZA E SILVA, E., LEÃO, R., MENASCHÉ, D., et al., "On the Interplay between Content Popularity and Performance in P2P Systems", In: *Quantitative Evaluation of Systems*, v. 8054, pp. 3–21, *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2013.

[32] ROSENSWEIG, E., KUROSE, J., TOWSLEY, D., "Approximate Models for General Cache Networks". In: *Proceedings IEEE Infocom 2010*, pp. 1–9, 2010.

[33] DE SOUZA E SILVA, E., MUNTZ, R. R., *Métodos Computacionais de Solução de Cadeias de Markov: Aplicações a Sistemas de Computação e Comunicação. VIII Escola de Computação*, Sociedade Brasileira de Computação, 1992.

[34] DE SOUZA E SILVA, E., GAIL, H., "Transient Solutions for Markov Chains", In: GRASSMANN, W. (ed), *Computational Probability*, pp. 43–79, Kluwer, 2000.

[35] GHODSI, A., SHENKER, S., KOPONEN, T., et al., "Information-centric Networking: Seeing the Forest for the Trees". In: *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, *HotNets-X*, pp. 1:1–1:6, ACM, 2011.

[36] DE SOUZA E SILVA, E. A., RATTON, D., LEÃO, R. M., "The TANGRAMII Integrated Modeling Environment for Computer Systems and Networks", *Performance Evaluation Review*, v. 36, pp. 64–69, 2009.

[37] ACM, "1st ACM Conference on Information-Centric Networking (ICN-2014)", 2014.

[38] AMAZON, "Amazon ElastiCache", 2014.

[39] TOWSLEY, D., KUROSE, J., "UMass Seminar", 2014.

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

[41] ROSENSWEIG, E. J., KUROSE, J., "Breadcrumbs: Efficient, best-effort content location in cache networks". In: *INFOCOM 2009, IEEE*, pp. 2631–2635, 2009.

[42] ROSENSWEIG, E. J., MENASCHE, D. S., KUROSE, J., "On the steady-state of cache networks". In: *INFOCOM, 2013 Proceedings IEEE*, pp. 863–871, 2013.

[43] STOICA, I., ADKINS, D., ZHUANG, S., et al., "Internet indirection infrastructure". In: *ACM SIGCOMM Computer Communication Review*, v. 32, n. 4, pp. 73–86, 2002.

[44] LAGUTIN, D., VISALA, K., TARKOMA, S., "Publish/Subscribe for Internet: PSIRP Perspective." *Future Internet Assembly*, v. 84, 2010.

[45] C.DANNEWITZ, D'AMBROSIO, M., KARL, H., et al., "MDHT: A Hierarchical Name Resolution Service for Information-centric Networks". In: *ACM ICN*, 2011.

[46] K. KATSAROS, G. X., POLYZOS, G. C., "MultiCache: An overlay architecture for information-centric networking". In: *Computer Networks*, 2011.

[47] FAYAZBAKHSH, S. K., LIN, Y., TOOTOONCHIAN, A., et al., "Less pain, most of the gain: incrementally deployable ICN". In: *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pp. 147–158, 2013.

[48] PERINO, D., VARVELLO, M., "A reality check for content centric networking". In: *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, pp. 44–49, 2011.

[49] ROSENSWEIG, E. J., KUROSE, J., TOWSLEY, D., "Approximate models for general cache networks". In: *INFOCOM, 2010 Proceedings IEEE*, pp. 1–9, 2010.

[50] MARTINA, V., GARETTO, M., LEONARDI, E., "A unified approach to the performance analysis of caching systems", *arXiv preprint arXiv:1307.6702*, 2013.

[51] CHE, H., TUNG, Y., WANG, Z., "Hierarchical web caching systems: Modeling, design and experimental results", *Selected Areas in Communications, IEEE Journal on*, v. 20, n. 7, pp. 1305–1314, 2002.

[52] FOFACK, N. C., NAIN, P., NEGLIA, G., et al., "Analysis of TTL-based cache networks". In: *Performance Evaluation Methodologies and Tools (VALUETOOLS), 2012 6th International Conference on*, pp. 1–10, 2012.

[53] BERGER, D. S., GLAND, P., SINGLA, S., et al., "Exact analysis of TTL cache networks: The case of caching policies driven by stopping times", *arXiv preprint arXiv:1402.5987*, 2014.

[54] TRAVERSO, S., HUGUENIN, K., TRESTIAN, I., et al., "Tailgate: handling long-tail content with a little help from friends". In: *Proceedings of the 21st international conference on World Wide Web*, pp. 151–160, 2012.

[55] ZHANG, H., LIU, B., WENG, X., et al., "Can online social friends help to improve data swarming performance?" In: *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, pp. 1–7, 2012.

[56] GITZENIS, S., PASCHOS, G. S., TASSIULAS, L., "Asymptotic laws for joint content replication and delivery in wireless networks", *Information Theory, IEEE Transactions on*, v. 59, n. 5, pp. 2760–2776, 2013.

[57] SHARIATPANAHI, S. P., SHAH-MANSOURI, H., KHALAJ, B. H., "Caching Gain in Wireless Networks with Fading: A Multi-User Diversity Perspective", *arXiv preprint arXiv:1309.0088*, 2013.

[58] MALANDRINO, F., KURANT, M., MARKOPOULOU, A., et al., "Proactive seeding for information cascades in cellular networks". In: *INFOCOM, 2012 Proceedings IEEE*, pp. 1719–1727, 2012.

[59] MUNARO, D., DELGADO, C., MENASCHÉ, D. S., "Sistemas de Recomendação de Conteúdo e Seus Impactos no Custo e Desempenho de Redes Par-a-Par", *SBRC*, 2014.

Apêndice A

Configuração do Simulador ICN

O simulador é configurado utilizando com os parâmetros apresentados nas tabelas A e A.

Tabela A.1: Descrição dos parâmetros de configuração do simulador.

Parâmetro	Descrição
SimTime	Tempo de simulação
TTLLimits	Valor máximo do TTL no passeio aleatório
DebugOper	Indica se a geração dos logs de operação estão ativos
DebugRouter	Indica se a geração dos logs dos roteadores estão ativos
DebugClient	Indica se a geração dos logs dos clientes estão ativos
[Trace]	Seção para configuração a geração de um arquivo de trace
IsEnable	Indica se a geração do arquivo de trace está ativa
MinTime	Tempo de início para a geração do trace
MaxTime	Tempo de término para a geração do trace
[RCLimits]	Seção para a configuração dos limites do RC
Upper	Limite superior do RC
Lower	Limite inferior do RC
[RequestDistParams]	Seção para a configuração da distribuição da geração de requisições
Type	Indica o tipo de distribuição. Pode ser 0 = Exponencial, 1 = Normal, 2 = Uniforme e 3 = Zpif
Params	Define os parâmetros da distribuição. Os parâmetros devem ser separados por ';'
[TransmissionDistParams]	Seção para a configuração da distribuição da transmissão dos pacotes
Domain	Indica o domínio que a disitribuição será utilizada
[RandomDistParams]	Seção para a configuração da distribuição da escolha aleatória
[ContentDistParams]	Seção para a configuração da disitribuição da escolha dos conteúdos na geração das requisições
[RCDistParams]	Seção para a configuração da distribuição do decréscimo do RC

Tabela A.2: Descrição dos parâmetros de configuração do simulador (cont.).

Parâmetro	Descrição
[Topology]	Seção para a configuração da topologia simulada
NodesNumber	Número de nós na rede
DomainsNumber	Número de domínios na rede
[NodesParams]	Seção para a configuração de cada nó na rede
Type	Indica o tipo de nó. Pode ser 0 = cliente, 1 = roteador e 2 = área de publicação
Name	Nome do nó
Ident	Identificação do nó na rede. Esta identificação deve ser única
DomainIdent	Domínio que o nó pertence
CacheSize	Esse parâmetro é representado por dois valores separados por ';'. O primeiro é o tamanho da cache local e o segundo é a política de substituição (0 = aleatório e 1 = menor RC)
NodesAboveConnected	Sequência dos nós do domínio acima que estão conectados a ele. A identificação dos nós deve ser separada por ';'
DomainNodesConnected	Sequência dos nós do mesmo domínio que estão conectados a ele. A identificação dos nós deve ser separada por ';'
[ContentItems]	Seção para a configuração dos conteúdos simulados
ContentsNumber	Número total de conteúdos
[ContentParams]	Seção para a configuração de cada conteúdos
Name	Nome do conteúdo
Size	Tamanho máximo. Não utilizado