Armazenamento em Redes Orientadas a Conteúdo baseado em Ranqueamento de Caches

Edson Adriano Maravalho Avelar, Kelvin Lopes Dias.

Centro de Informática – Universidade Federal de Pernambuco (UFPE)
CEP: 50740-540 – Recife – PE – Brasil

{eama,kld}@cin.ufpe.br

Abstract. *In-network caching is one of the main features of the Information Centric Network (ICN). Thus, the adopted caching strategy is crucial for the efficient delivery of content and cache storage. In this paper, we proposed a caching algorithm (RankCache) by exploring the concept of cache ranking. The ranking of each cache is calculated using interval estimation of binomial proportion, where the success parameter is related to requested content popularity. Through simulations, our algorithm is compared with several caching strategies, such as LCE, Fix Probability, Betweenness Centrality, LCD and ProbCache. A performance evaluation was conducted which confirmed the superiority of RankCache in terms of hit rate, hit hop, server hit and overall delay, with gains ranging from 10% to over 30%.*

Resumo. *O armazenamento de dados nos nós da rede é uma das principais características para manter e entregar conteúdos de forma eficiente nas ICN (Information Centric Networks) ou Redes Centradas na Informação. Neste artigo, propomos um algoritmo de armazenamento, chamado RankCache, baseado no conceito de ranqueamento de cache. O ranking de cada cache é calculado usando estimação intervalar da proporção binomial, onde o parâmetro de sucesso está relacionado à popularidade dos conteúdos requisitados. Através de simulações, o algoritmo foi comparado com várias estratégias da literatura, como LCE, Fix, Betweenness Centrality, LCD e ProbCache. A análise de desempenho mostrou a superioridade da proposta em termos de taxa de acerto, número de saltos, acertos no servidor e atraso total, com ganhos que variam de 10% a 30%.*

1. Introdução

Quando a Internet foi idealizada, o principal problema que precisava que ser resolvido era a entrega de dados entre computadores ou entre computadores e terminais. O modelo básico era centrado na comunicação *host-to-host. Esse* modelo reduziu a eficiência dos serviços de distribuição de conteúdo. Como resultado, a rede de comunicação era, e continua a ser, baseada na localização física de hospedeiros [Jacobson 2009, Jacobson 2012].

No entanto, a Internet de hoje está mais preocupado com a conexão entre pessoas, conteúdos e informações. Esses conteúdos incluem *streaming* de áudio e vídeo, redes sociais, jogos online, entre outros. Segundo o documento de previsão da Cisco® [Cisco 2014] para 2013-2018, o tráfego global de vídeo será responsável entre 80% a 90% de todo o tráfego da internet até 2018. Segundo o mesmo relatório, o tráfego gerado por redes de entrega de conteúdo ocupará metade de todo tráfego de vídeo pela

Internet até 2018. Esses números mostram um grande crescimento na busca por conteúdo. Neste novo cenário, a localização física é a que menos importa para os usuários, que querem conteúdo sempre disponível, com maior velocidade possível e melhor qualidade de serviço.

Seguindo essa tendência, as Redes Centradas na Informação [Brito 2012, Torres 2013], tradução adaptada de ICN (*Information Centric Network*), oferecem um novo paradigma onde as operações da rede são focadas no alvo de interesse dos usuários, a informação. Nas ICN, um conteúdo é consumido diretamente através de seu nome e não de sua localização. Diferentemente da abordagem tradicional das redes IP, centrada na identificação e localização de nós, as ICNs usam conceitos mais inovadores como identificação de conteúdo por nome, roteamento baseado em nomes, segurança aplicada diretamente ao conteúdo e armazenamento de dados nos elementos da rede [Ghodsi 2011].

Dentre as várias propostas para ICNs [Ahlgren 2012], a arquitetura CCN (*Content Centric Network*), é uma das que mais se destacou, pois, além da farta documentação, ela possui uma implementação oficial de código fonte aberto. Essa implementação, nomeada CCNx, foi criada e é mantida pela PARC (*Palo Alto Reasearch Center/Xerox*) [CCNx 2014]. Na arquitetura CCN, o nome do conteúdo é desacoplado do endereço físico de seu provedor. O benefício dessa abordagem é a clara separação entre a identificação e a localização do conteúdo. Como nas redes IP não há essa separação, problemas como mobilidade e segurança nunca tiveram soluções amplamente aceitas, devidos os remendos propostos na pilha TCP/IP, como Mobile IP e IPSec.

O CCN utiliza dois tipos de pacotes: *Interest* e *Data*. O consumidor expressa seu interesse inserindo o nome do conteúdo desejado em uma mensagem do tipo *Interest* e a envia para rede. O produtor, ou algum cache no interior da rede, que possui o conteúdo, receberá essa mensagem e enviará de volta ao consumidor uma mensagem do tipo *Data* como resposta. Portanto, essas mensagens possuem uma relação um para um, onde um pacote de interesse satisfaz um de dados se o nome do conteúdo em ambos os pacotes forem equivalentes.

No CCN, os conteúdos são quebrados em pequenos pedaços chamados de *chunks*[1]. Esses pedaços podem ser auto-identificados (*self-indentified*) e auto-autenticados (*self-authenticated*), sem precisar estar associados a um *host* em particular. Além disso, essa arquitetura permite armazenamento dentro dos elementos da rede (*in-network* ou *in-path caching*). Essa característica melhora significativamente a entrega de conteúdos populares (constantemente requisitados), pois os usuários podem receber o conteúdo diretamente de elementos do núcleo da rede, sem que a requisição precise chegar ao provedor do conteúdo. Se um *chunk* do conteúdo requisitado é encontrado no cache de um nó da rede, que não seja o provedor, diz-se que ocorreu um "*acerto*" (*hit*, em inglês) naquele cache. Se um *chunk* não for encontrado em um cache, diz-se que ocorreu um "erro" (*miss*, em inglês) naquele cache.

O armazenamento na rede possui vários benefícios incluindo melhora no desempenho da entrega de conteúdo, diminuição do tráfego na rede, diminuição do atraso dos *chunks* entre outros. No entanto, existem vários desafios que precisam ser

[1] No restante do artigo, para manter o termo consolidado, será usado o termo "*chunk"* para pedaço de conteúdo.

investigados. Um dos mais notáveis é decidir em que caches guardar um *chunk*, de modo a maximizar a probabilidade de ocorrer um *acerto*, diminuir a redundância de *chunks* nos caches e reduzir o atraso de entrega dos conteúdos.

Nesse contexto, este artigo propõe um esquema de armazenamento de *chunks* chamado *RankCache*. O objetivo dessa proposta é prover entrega eficiente de conteúdo e armazenamento otimizado com redução de sobrecarga de gerenciamento e cooperação implícita entre os nós da rede. O algoritmo do RankCache distribui os *chunks* considerando o *ranking* de cada cache, que no CCN pode ser chamado de *Content Store* (CS), *Content Router* (CR) ou apenas cache. As principais características do *RankCache* são:

a) **Ranking baseado na Popularidade**: O algoritmo ajusta o valor dos ranking de cada cache em um dado caminho. O cache com maior valor de ranking possuirá a maior probabilidade de armazenar um *chunk* do que um cache com menor ranking. Esse valor é calculado usando o WSI (*Wilson Score Interval*) [Wilson 1927], que é uma fórmula para calcular o intervalo de confiança para proporção binomial (sucesso versus fracasso). Na proposta, a variável de sucesso é baseada no número de *acertos* de um cache. Quanto mais *chunks* populares passar pelo cache, maior o número de *acertos* dentro dele, e consequentemente, maior sua probabilidade de armazenar outros *chunks*.

a) **Cache Decentralizado**: No RankCache, a decisão de guardar ou não um *chunk* é feita de forma independente. Cada cache calcula sua probabilidade de armazenamento sem a interferência de outros caches. Assim, não há nenhum gerenciamento centralizado. Dessa forma, a proposta é escalável, pois não requer conhecimento a priori da topologia da rede.

O restante do artigo é organizado da seguinte forma. A Seção 2 apresenta os trabalhos que serão utilizados como comparação com o RankCache. A Seção 3 detalha o funcionamento do esquema proposto. Na Seção 4, a avaliação de desempenho, via simulação, mostra a superioridade da proposta e a Seção 5 conclui o artigo.

2. Trabalhos Relacionados: Políticas de Decisão de Cache

As estratégias de armazenamento na rede podem ser divididas em duas políticas: de substituição (*replacement*) e de decisão (*decision*). A primeira define qual *chunk* armazenado será removido quando o cache estiver cheio. A segunda está relacionada à decisão de guardar ou não um *chunk* que atravessa o nó. Existem várias políticas de substituição de *chunks*, porém a mais utilizada em trabalho de cache é a LRU (*Least Recently Used*)[Chai 2013, Psaras 2011], que remove o item menos usado do cache. Essa política também foi recomendada pelos autores do CCN [Jacobson 2009].

Existem alguns esforços para propor soluções no contexto de políticas de decisão de cache. A seguir, serão detalhadas as propostas mais citadas e que farão parte da avaliação comparativa deste trabalho.

A. LCE (*Leave Copy Everywhere*)

Esta é a proposta mais usada em caches multi-níveis. Após detectar uma cópia do *chunk* requisitado, uma cópia é guardada em todos os caches intermediários no caminho de volta para o usuário requisitante. Apesar de simples, esse esquema

apresenta grandes desvantagens, como o alto nível de redundância de *chunks* na rede. Porém, é bastante usada como base para comparação entre propostas [Laoutaris 2004, Chai 2013].

FixProb (ou apenas Fix) é uma versão randômica do LCE. Cada nó intermediário ao longo do caminho entre o consumidor e o provedor é um candidato possível para guardar uma cópia do objeto requisitado. Os caches intermediários podem guardar os objetos com uma probabilidade *p*. Nota-se que o FixProb com *p* igual a 1 (100%) é idêntico ao LCE. Para fins de comparação usou-se *p* igual a 0.3 e 0.7, mesmos valores usados em alguns trabalhos [Psaras 2012].

B. LCD (*Leave Copy Down*)

É um algoritmo, proposto por [Laoutaris 2004], onde uma cópia do conteúdo requisitado é armazenada no cache após um *acerto*. Sempre que é encontrado um objeto dentro do cache, ele é armazenado apenas um cache após o *acerto*. O algoritmo verifica o número de saltos *H* que o *chunk* percorreu; apenas se $H = 1$ o cache armazena o *chunk*. O LCD é mais "conservador" que o LCE, pois ele precisa de várias requisições para levar o conteúdo para perto do consumidor. Por outro lado, esse algoritmo reduz o número de cópias redundantes de *chunks* na rede.

C. Betweenness Centrality

Uma importante classe de políticas de decisão são as baseadas no conceito de centralidade dos nós [Izquierdo 2006]. O princípio fundamental desta estratégia é que, se um nó se encontra ao longo de um grande número de caminhos de entrega de conteúdo, é mais provável que consiga um *acerto*. Uma vez que existem várias estratégias de centralidade, a nossa comparação considerará o algoritmo *Betweenness Centrality* [Chai 2011, Chai 2013], por ser a que apresenta melhor desempenho e maior facilidade de implementação.

D. ProbCache

Apresentado por [Psaras 2012], o ProbCache é uma algoritmo que explora o conceito de capacidade de armazenamento do caminho (*path-caching capacity*). Este valor é calculado através da medição da capacidade de armazenamento médio dos caches no trajeto do pacote. A proposta introduz dois novos cabeçalhos nas mensagens CCN, o TSI (*Time Since Inception*) e o TSB (*Time Since Birth*). Cada roteador aumenta o valor TSI dos pacotes de solicitação em um. O provedor do conteúdo atribui o valor TSI que vê na mensagem de requisição para a mensagem de resposta. Cada roteador aumenta o valor TSB da mensagem de resposta também em um. Assim, quando o conteúdo viaja de volta para o cliente, o TSI tem um valor fixo e indica o comprimento do caminho desse fluxo específico, enquanto o valor de TSB indica o número de saltos que o conteúdo da mensagem de respostas percorreu desde o *acerto*.

3. Algoritmo de Armazenamento em redes CCN baseado no Ranking do Cache.

As melhores propostas de políticas para a tomada de decisão levam em conta a popularidade de pedaços de conteúdo (*chunk*). Um *acerto* no cache envolve nada mais do que achar conteúdos requisitados anteriormente por outros usuários e que estejam armazenados. A probabilidade de um sucesso é diretamente proporcional à popularidade

de um *chunk*. Uma política de decisão de cache eficiente deve ser capaz de prever quando essas requisições repetidas ocorrerão. No RankCache, proposto neste trabalho, o armazenamento dos *chunks* é proporcional ao ranking de cada cache. O ranking de um dado cache é calculado com base no número de requisições de *chunks* repetidos que passam através do cache.

O problema de calcular o ranking de vários caches independentes, usando a proporção de número de *acertos* versus número de *erros*, não é simples como usar um sistema de classificação médio (número de *acertos* dividido pelo número total de pacotes). Por exemplo, com classificação média, um cache com $3x10^3$ *acertos* e $6x10^3$ pacotes no total (*acertos* + *erros*) terá o mesmo valor de ranking que um cache com 6 *acertos* e 12 *chunks*; e isso não é uma representação justa. Neste trabalho, usou-se um cálculo mais sofisticado para criar o ranking de caches. O cálculo leva em consideração dois fatores: a proporção binomial e o valor de proximidade.

Existem algumas fórmulas para cálculo de proporção binomial com Intervalo de Confiança [Edwardes 1994]. Neste artigo, utilizou-se a fórmula chamada *Wilson Score Interval*, proposta por Edwin B. Wilson [Wilson 1927], por sua simplicidade e velocidade de processamento. A fórmula de Wilson foi adaptada neste artigo para calcular o ranking do cache usando o número de *acertos* em um dado cache. Essa fórmula, denominada Proporção Binomial do *i*-ésimo cache (PBc_i), é dada por:

$$PBc_i = \frac{\rho i + \frac{1}{2ni} z^2_{1-\alpha/2} \pm z_{1-\alpha/2} \sqrt{\frac{\rho i(1-\rho i)}{ni} + \frac{z^2_{1-\alpha/2}}{4ni^2}}}{1 + \frac{1}{ni} z^2_{1-\alpha/2}}$$

Onde:

- $\boldsymbol{\rho i}$ é o número de *acertos* observados no cache *i*.
- $\boldsymbol{ni}$ é o número total de *chunks* que passam pelo cache *i*.
- $\boldsymbol{z^2_{1-\alpha/2}}$ é o valor padronizado da distribuição normal para um dado Intervalo de Confiança.

O cálculo do ranking considera a contagem de ocorrências de sucesso como uma amostra estatística a ser estimada. O algoritmo infere, em um intervalo de confiança, um valor de cache baseado na estimativa da ocorrência de *acertos* naquele cache. Usou-se como Intervalo de Confiança o valor 0,95 (95%), o que dá um valor padronizado ($\boldsymbol{z^2_{1-\alpha/2}}$) igual a 1,96.

Além da inferência de proporção binomial, elaboramos neste artigo um cálculo do ranking usando o que denominamos de Fator de Proximidade (*FP*). Esse fator aumenta a probabilidade de um *chunk* ser armazenado perto do consumidor. Estudos mostraram que os conteúdos tendem a ser armazenados por mais tempo nas bordas da rede [Psaras 2011]. O FP ajuda a fórmula do ranking a modelar esse comportamento. O Fator de Proximidade do i-ésimo cache (FPc_i) é calculado por:

$$FPc_i = \frac{\sum_{j=1}^{Hc} H_j}{\sum_{i=1}^{Ht} H_i}$$

Onde:

- $\boldsymbol{H_c}$ é o número atual de saltos após um *acerto*
- $\boldsymbol{H_t}$ é o número de saltos totais do pacote até a ocorrência do *acerto*.
- $\boldsymbol{H_j}$ e $\boldsymbol{H_i}$ são valores somados a cada salto dos pacotes. Alguns saltos podem ter pesos por algum razão, porém, neste trabalho, foram usados saltos unitários.

Para exemplificar o funcionamento do FPc_i, supõe-se que um *chunk* é encontrado no sétimo cache de um dado caminho. No primeiro salto do pacote de dados, FPc_7 = 1/7, no segundo salto, FPc_6 =2/7, no terceiro, FPc_5 = 3/7 e assim por diante, até chegar ao nó requisitante, que não entra no cálculo, ou seja, nunca FPc_i será igual a 7/7. Percebe-se que o FPc_i é incrementado com a proximidade do nó requisitante, isso ajuda a aumentar a probabilidade de armazenamento na borda da rede.

O valor de H_c pode ser obtido através do campo TTL (*Time To Live*) da mensagem CCN. Para possibilitar todos os cálculos necessários, o RankCache requer a adição apenas do H_t no cabeçalho da mensagem CCN. Depois de um *acerto*, o valor de H_t pode ser calculado através do valor de H_c e será gravado na mensagem de resposta.

A fórmula final para o cálculo da probabilidade de guardar o *chunk* no i-ésimo cache é dada pela seguinte equação:

$$PRc_i = PBc_i * FPc_i$$

O Algoritmo 1 descreve como a função de decisão utiliza a fórmula PRc_i para o armazenamento de um *chunk*. Essa função recebe como parâmetro a mensagem CCN de chegada e retorna sua decisão como um valor booleano.

Algoritmo 1: Função de Decisão do RankCache

```
1.  Entrada: Mensagem CCN //recebe como parâmetro a mensagem CCN :
2.  Saída: booleano decisao // Retorna um valor lógico indicando a decisão.

3.  decisao ← falso.
4.
5.  //Define acertos como o número total de acertos deste cache. Ele precisa ser um
    contador com escopo global.
6.  acertos ← Acertos Totais
7.
8.  n ← n + 1 //Incrementa o número total de chunks n.
9.  obtém o valor de Hc da mensagem CCN;
10. obtém o valor de Ht da mensagem CCN;
11.
12. Rp ← Hc/Ht
13. //PBc também é global para este cache, só é atualizado em um intervalo de tempo
    pré-definido para evitar overhead de atualização.(Linhas 27-30)
14. PRc ← PBc * Rp
15.
16. //Gera um número randômico x
17. x ← Número Randômico entre 0 e 1.
18. se x é menor que Prc então:
19.     decisao ← verdadeiro;
20. fim se
21.
```

```
22. se decisao é verdadeiro então:
23.     Incrementa o valor do número de acertos
24. fim se
25.
26. //atualiza o rank do cache.
27. se o intervalo de atualização do Rc foi alcançado então:
28.     phat ← 1.0*acertos/n;
29.     PBc ← (phat + z*z/(2*n) - z * sqrt((phat*(1-phat)+z*z/(4*n))/n))/(1+z*z/n);
30. fim se
31.
32. retorna decisao
```

Primeiramente, são obtidos os valores H_c e H_t através da mensagem CCN (linhas 9-10). Posteriormente, é calculada a probabilidade de armazenamento do *chunk*. Nota-se que, no algoritmo 1, "PB_c" é uma variável global e é inicializada com 1 fora da função. Se a decisão de guardar o *chunk* for positiva, a variável "*acertos*", que guarda o número de *acertos*, é incrementada (linha 23). O ranking do cache é atualizado, através do calculo do *"PBc"*, em intervalos previamente definidos, para evitar o overhead de atualizações desnecessárias (linha 29). Ao final das operações, a função retorna a decisão de armazenamento.

Esse procedimento, detalhado pelo algoritmo 1, é invocado por uma entidade que gerencia as políticas de cache. Caso a decisão de armazenamento seja positiva e não haja mais espaço no cache, o algoritmo de reposição entra em ação para trocar um *chunk* armazenado pelo que acabou de ser escolhido. A Seção 4 apresenta a avaliação do algoritmo proposto.

4. Avaliação

Para avaliar o RankCache, foi desenvolvido um simulador de rede CCN. Ele foi implementado no framework OMNET++[Varga 2001] e foi baseado em algumas ideias de dois outros simuladores; o ccnSim [Chiocchetti 2013] e CCN-Lite [CCN-lite 2013], ambos implementados no OMNET++. O primeiro é um simulador em nível de *chunk* e possui um alto grau de abstração, por isso, torna-se difícil a avaliação de políticas de roteamento e de distribuição de prefixos; e métricas próximas da camada de rede como atrasos, perdas de pacotes e vazão.

Por outro lado, o CCN-Lite não possui a implementação da camada de núcleo CCN, onde residem a PIT (*Pendent Interest Table*), a FIB (*Forwarding Information Base*) e o CS (*Content Store*). No CCN-Lite, o núcleo CCN é implementado fora do OMNET++, de modo que possa ser aproveitado por outras plataformas como Linux e Raspberry Pi. Apesar de ser uma vantagem em alguns casos, isso dificulta bastante a implementação de novas propostas que envolvam o núcleo CCN, como roteamento, políticas de decisão e de armazenamento, por exemplo. No nosso simulador, foi implemento todo o núcleo CCN, incluindo as tabelas PIT, FIB e CS. Além disso, ele implementa o protocolo OSPFN [Wang 2012] para roteamento e distribuição de prefixos. Além do mais, permite extração de métricas de rede como atraso, perdas de pacote e vazão.

O experimento foi conduzido de acordo com a Tabela 1. O LRU foi usado como política de substituição. A distribuição Zipf foi empregada como modelo de

popularidade de conteúdo. O Zipf é uma das distribuições mais usadas para modelar esse comportamento e é, de acordo com alguns estudos, a que melhor explica as estatísticas de download de conteúdo de serviços de vídeo sob demanda, como o Youtube [Cha 2009].

Tabela 1. Configuração dos parâmetros dos experimentos.

Networks	27 nodes (Rede IPÊ)[RNP 2014]
Atrasos na rede.	10-100ms (uniforme)
Intervalo de Atualização (PB*c*)	15s
Packet Capacity	50 pacotes
Transmission rate	exponencial, média = 1/80
Tamanho do catálogo[3]	10^5 files
Tamanho médio dos conteúdos	100 *chunks*
Tempo de simulação	10^3s

Foi utilizada, como cenário de simulação, a topologia da rede IPÊ [RNP 2014]. Ela possui 27 nós distribuídos, através de PoPs (*Points of Presence*/Pontos de Presença), por todos os estados brasileiros. A Figura 1 mostra a topologia implementada no simulador e usada nas simulações.

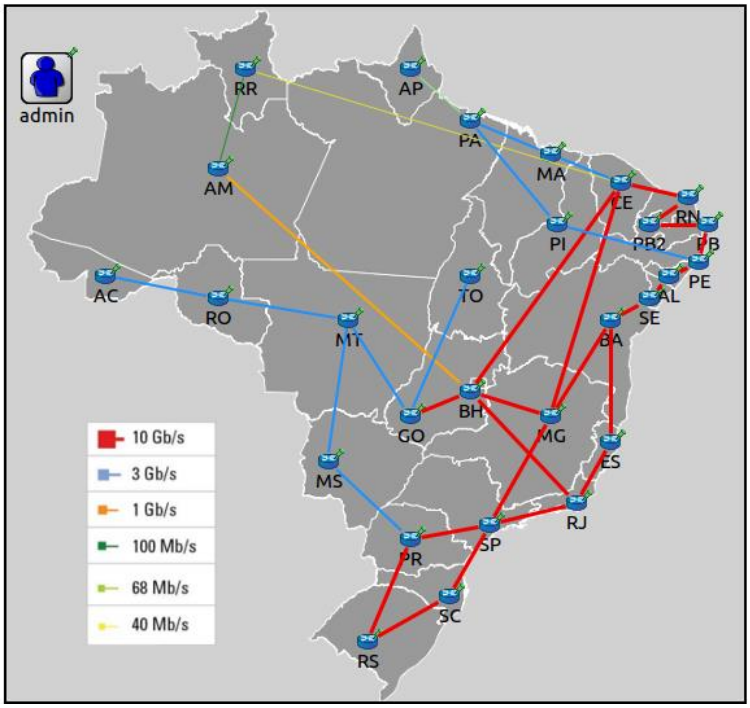

Figura 1. Topologia da Rede IPÊ implementada no simulador

Para todos os experimentos foram feitas 30 simulações. Os valores apresentados nas Figuras 2, 3, 4 e 5, foram obtidos com o expoente da distribuição Zipf iguais a 0.5.

[3] Faz referência ao número de conteúdos únicos.

Quanto menor o Zipf, menor o número de conteúdos populares no catálogo [Psaras 2012].

A Figura 2 apresenta a taxa média de *acertos* em relação à variação da capacidade dos caches. Como pode ser visto na figura, o RankCache possui uma taxa de *acertos* maior que todas as outras propostas. É superior[4] em aproximadamente 15% em relação à média do Fix0.3 e mais de 17% em relação à média do ProbCache.

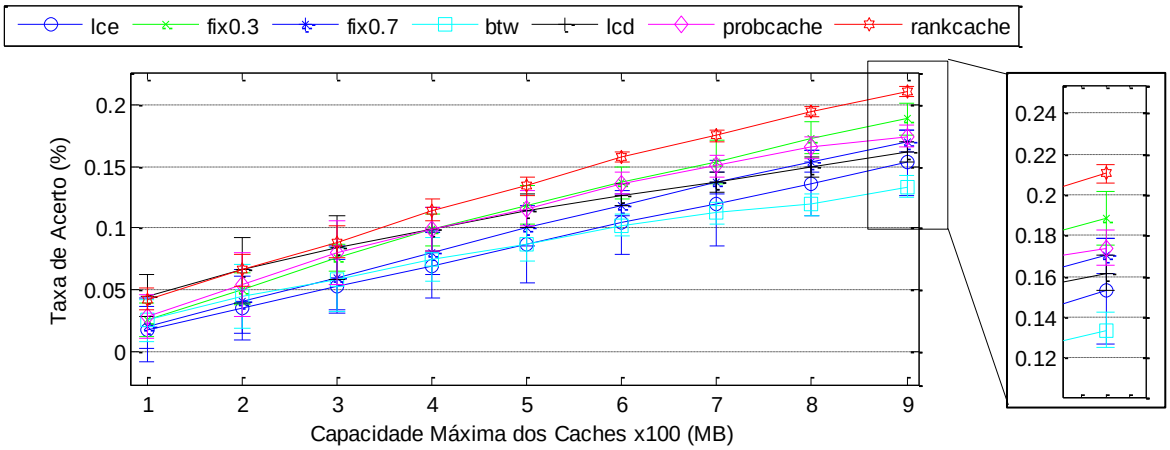

Figura 2. Taxa de *Acertos* comparada com a capacidade dos caches variando de 100 MBytes a 900 MBytes.

A Figura 3 mostra o número de saltos médios necessários para ocorrer um *acerto*. Essa métrica está relaciona à distância média entre o usuário que requisita o conteúdo e onde o conteúdo é encontrado. Quanto menor é o valor dessa métrica, menor é o caminho percorrido pela mensagem de requisição e de resposta. Além disso, mais rápido o usuário consegue o conteúdo requisitado. O número de saltos médio do RankCache, com 900Mb de cache, foi de aproximadamente 2,8 saltos, contra 3,2 saltos da segunda melhor proposta (Probcache). Nessa Figura, o LCE possui o pior desempenho, isso porque o LCE armazena todos os pacotes de forma indiscriminada e sem distinção de nível de popularidade, proximidade ou outro tipo de característica.

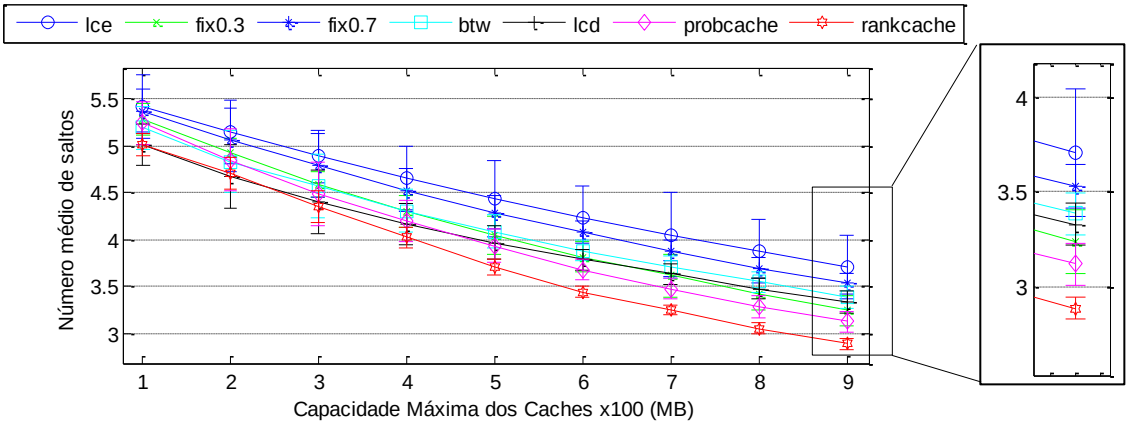

Figura 3. Número de saltos até um *Acertos* em relação à capacidade dos caches variando de 100 MBytes a 900 MBytes.

[4] No artigo, toda a comparação entre propostas é feita em relação à média dos dados.

Outra importante métrica para avaliar o desempenho de algoritmos de cache é a quantidade de *acertos* no servidor, ou seja, no provedor do conteúdo. Quando maior o número de *acertos* no servidor menor é a eficiente do algoritmo em distribuir os *chunks* pela rede. A redução do tráfego total da rede é um dos benefícios do baixo número de *acertos* no servidor, uma vez que, menos pacotes precisaram percorrer todo o caminho entre o consumidor e o provedor. Como apresentado na Figura 3, o RankCache também se mostrou superior nesse experimento. Ele foi o algoritmo que menos atingiu o servidor nas simulações, menor até que o ProbCache, o qual foi desenhado para otimizar essa métrica [Psaras 2012]. Com 900Mbytes de cache, o RankCache foi 17% superior em relação ao ProbCache, segundo mais eficiente.

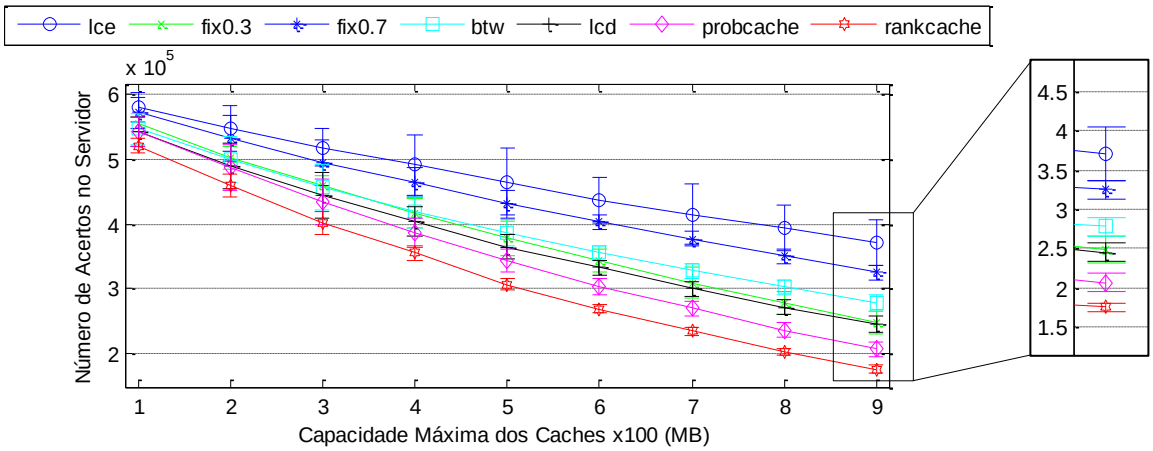

Figura 4. Número de *Acertos* no servidor em relação à capacidade dos caches variando de 100 MBytes a 900 MBytes.

O atraso dos pacotes é outra métrica importante que reflete diretamente no QoS (*Quality of Service*) e o QoE (*Quality of Experience*) das aplicações dos usuários finais. Um baixo atraso é requisito mínimo de serviços de tempo real como streaming de áudio e vídeo. A avaliação mostrou que o RankCache possui o menor atraso médio total. A Figura 5 mostra o tempo de ida e volta (RTT-*Round Trip Time*) das mensagem CCN na rede. Como o RankCache provê uma alta taxa de *acerto* e um baixo número de *acertos* no servidor, o conteúdo é entregue o mais rápido possível, o que reduz o atraso total dos pacotes. O ganho, neste caso, é de 10% em relação ao segundo melhor, ProbCache.

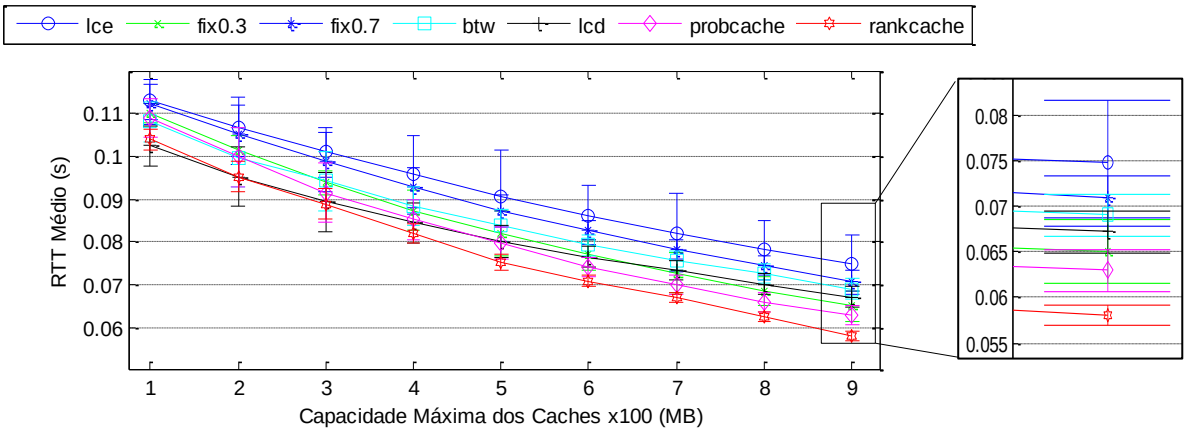

Figura 5. RTT médio em relação à capacidade dos caches variando de 100 MBytes a 900 MBytes.

Na distribuição Zipf, um alto valor do expoente alfa (α) representa um tamanho pequeno de conteúdos com uma popularidade similar. Por outro lado, um valor baixo de α representa um grande número de conteúdos com a mesma popularidade. A Figura 6 mostra o número de *acertos* no servidor em relação à variação do expoente Zipf entre 0.5 e 1.0. O RankCache tem vantagem quando α é pequeno, uma vez que suas características levam em conta a popularidade dos *chunks*. São nos piores casos, onde o número de chunks populares é menor, que o algoritmo se destaca mais. O RankCache tem o melhor desempenho, em torno de 17% com α = 0.5 e 9% com α = 1.0, em relação ao segundo melhor, Probcache.

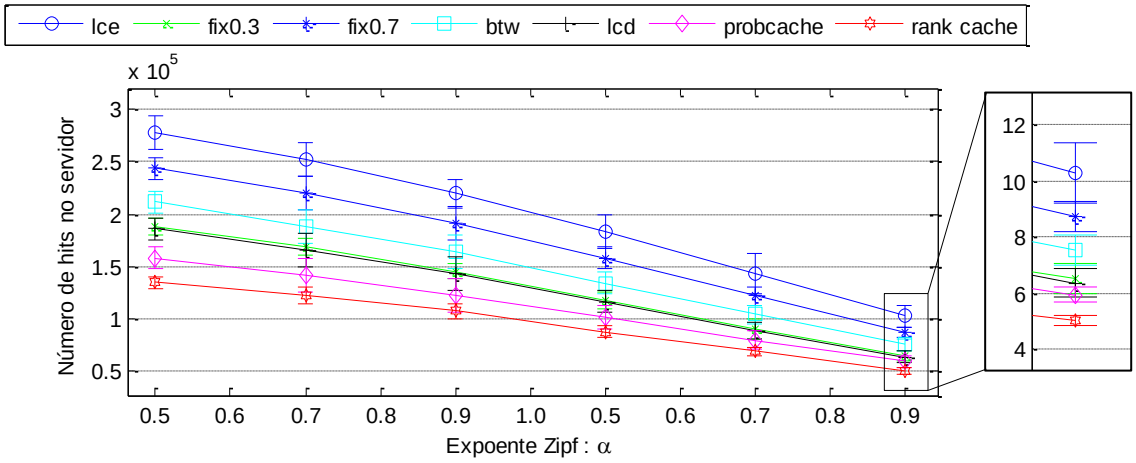

Figura 6. Número de *acertos* no servidor em relação ao expoente Zipf α, onde α varia de $5x10^{-1}$ a $10x10^{-1}$. Tamanho do Cache igual a 900MB

Os resultados que mostram o RTT médio de todos os pacotes em relação ao expoente do Zipf são apresentados na Figura 7. O ganho do RankCache foi de 10% comparado com o Fix03, segundo menor RTT médio.

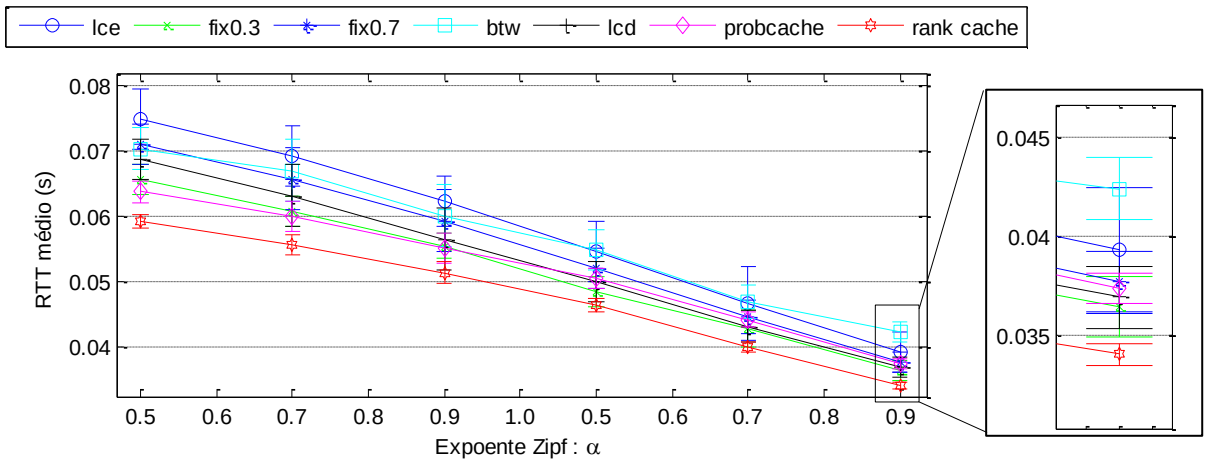

Figura 7. RTT médio em relação ao expoente Zipf α, onde α varia de $5x10^{-1}$ a $10x10^{-1}$. Tamanho do Cache igual a 900MB

Para verificar a natureza distribuída dos algoritmos testados, foi introduzido um modelo de erro de bits igual a $1x10^{-6,}$ uniformemente distribuído. Para essa simulação

foram feitos 15 repetições. A Figura 8 mostra os resultados do número médio de pacotes perdidos na rede. Com um modelo de erro aplicado na rede, perdas de pacotes são inevitáveis. Quanto maior o caminho percorrido, maior a probabilidade de o conteúdo ser perdido por erros na transmissão. O RankCache apresenta menor número de pacotes perdidos, isso se deve ao fato dele distribuir os *chunks* de forma mais eficiente na rede, de modo que as requisições não precisam percorrer longos caminhos para obter o conteúdo. Neste caso, o ganho do RankCache foi de 33% se comparado com o LCD, segundo melhor.

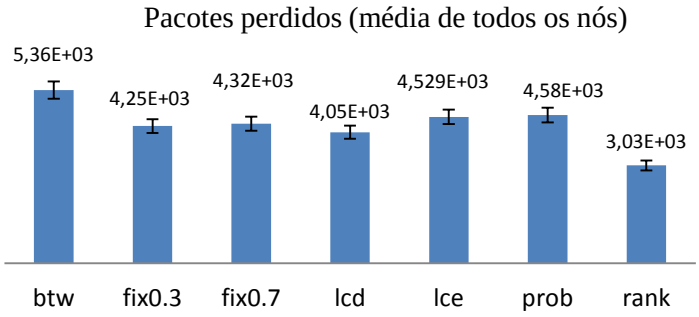

Figura 8. Média dos pacotes perdidos em 15 simulações com modelo de erro de bit igual a 1x10^{-6}

5. Considerações finais

Neste artigo, foi apresentado um novo esquema de armazenamento em nós da rede chamado RankCache. A proposta cria um ranking de caches no caminho do fluxo de pacotes. Os caches com maior ranking possuem a maior probabilidade de armazenar um determinado pacote. Os rankings são calculados através do intervalo de confiança da proporção binomial, usando a equação de Wilson, onde o valor de interesse é o número de vezes que um conteúdo foi encontrado nos caches intermediários.

A proposta foi avaliada e comparada com várias outras da literatura como *Leave Copy Everywheare*, Fix, *Betweenness Centrality*, *Leave Copy Down* e ProbCache. Em todos os casos, o RankCache se mostrou superior nas métricas avaliadas, com ganhos que variam de 10% a 30%, em relação ao segundo melhor de cada métrica.

Redes Orientadas a Conteúdo é uma área ativa de pesquisa e ainda existem várias lacunas que precisam ser preenchidas como mobilidade, esquemas de nomeação, roteamento, armazenamento entre outras [Kurose 2014].

O artigo deixou algumas questões em aberto, como o desempenho do algoritmo em vários outros cenários; uso de caches heterogêneos, ou seja, com diferença expressiva de tamanho; estudo sobre o intervalo de atualização do PBc_i e a avaliação do custo computacional do cálculo do rank. Todas essas questões serão abordadas em trabalhos futuros.

O próximo passo do trabalho é aprofundar a análise do RankCache implementando-o em redes reais ou emuladas [Cabral 2013]. Será investigado também, o impacto do RankCache em redes móveis através de avaliações com métricas de QoE (*Qualidade de Experiência*).

Agradecimentos

Este trabalho foi parcialmente financiado pela Fundação de Amparo à Ciência e Tecnologia do Estado de Pernambuco (FACEPE), através do processo IBPG-0829-1.03/12.

Referências

Ahlgren, B., Dannewitz, C., Imbrenda, C., Kutscher, D., & Ohlman, B. (2012). A survey of information-centric networking. Communications Magazine, IEEE, 50(7), 26-36.

Brito, G. M., Velloso, P. B., & Moraes, I. M. (2012). Redes orientadas a conteúdo: Um novo paradigma para a Internet. Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC, 2012, 211-264.

Cabral, C., Rothenberg, C., Magalhães, M.: (2013) "Mini-CCNx: prototipagem rápida para Redes Orientadas a Conteúdo baseadas em CCN", In Salão de Ferramentas do SBRC 2013.

Chai, W. K., Wang, N., Psaras, I., Pavlou, G., Wang, C., de Blas, G. G., ... & Hadjioannou, E. (2011). Curling: Content-ubiquitous resolution and delivery infrastructure for next-generation services. Communications Magazine, IEEE, 49(3), 112-120.

Chai, W. K., He, D., Psaras, I., & Pavlou, G. (2013). Cache "less for more" in information-centric networks (extended version). Computer Communications, 36(7), 758-770.

Cha, M., Kwak, H., Rodriguez, P., Ahn, Y. Y., & Moon, S. (2009). Analyzing the video popularity characteristics of large-scale user generated content systems. IEEE/ACM Transactions on Networking (TON), 17(5), 1357-1370.

Chiocchetti, Raffaele, Rossi, Dario and Rossini, Giuseppe, ccnSim: an Highly Scalable CCN Simulator (2013). In IEEE International Conference on Communications (ICC).

CCN-lite, "Lightweight CCN Simulator" (2013). Disponível em: http://www.ccn-lite.net/. Acessado em: nov/2014

CCNx (2014). "A CCN Implementation by Palo Alto Research Center (PARC)". Disponível em: http://www.ccnx.org/. Acessado em: nov/2014.

Cisco Visual Networking Index: "Forecast and Methodology, 2013–2018".(2014) Disponível em: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf. Acessado em: nov/2014

Edwardes, M., & Vollset, S. E. (1994). Confidence intervals for a binomial proportion. Statistics in medicine, 13(16), 1693-1698.

Ghodsi, A., Koponen, T., Rajahalme, J., Sarolahti, P. e Shenker, S. (2011). Naming in content-oriented architectures. Em ACM SIGCOMM Workshop on Information-Centric Networking - ICN, páginas 1–6.

Izquierdo, L. R., & Hanneman, R. A. (2006). Introduction to the formal analysis of social networks using mathematica. University of California, Riverside.

RNP (2014). “Rede Nacional de Pesquisa/Arquitetura da Rede IPÊ”. Disónível em: http://www.rnp.br/servicos/conectividade/rede-ipe. Acessado em: nov/2014

Jacobson, V., Smetters, D., Thornton, J., Plass, M., Briggs, N. e Braynard, R. (2009). “Networking named content”. International Conference on emerging Networking EXperiments and Technologies - CoNEXT.

Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M., Briggs, N. e Braynard, R. (2012). “Networking Named Content”. Communications of the ACM, 55(1):117–124.

Kurose, Jim. (2014) "Information-centric networking: The evolution from circuits to packets to content." Computer Networks. p. 112-120.

Laoutaris, N., Syntila, S., & Stavrakakis, I. (2004). Meta algorithms for hierarchical web caches. In Performance, Computing, and Communications, 2004 IEEE International Conference on (pp. 445-452). IEEE.

Psaras, I., Clegg, R. G., Landa, R., Chai, W. K., & Pavlou, G. (2011). Modelling and evaluation of CCN-caching trees. In NETWORKING 2011 (pp. 78-91). Springer Berlin Heidelberg.

Psaras, I., Chai, W. K., & Pavlou, G. (2012). Probabilistic in-network caching for information-centric networks. In Proceedings of the second edition of the ICN workshop on Information-centric networking (pp. 55-60). ACM.

Torres, J. V., Ferraz, L. H. G., Duarte, O. C. M. B. (2013). Redes orientadas a conteúdo baseadas em controladores hierárquicos. XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos-SBRC.

Varga, A. (2001). The OMNeT++ discrete event simulation system. In Proceedings of the European Simulation Multiconference (ESM’2001) (Vol. 9, p. 185). sn.

Wang, L., Hoque, A. K. M. M., Yi, C., Alyyan, A., & Zhang, B. (2012). OSPFN: An OSPF based routing protocol for Named Data Networking. University of Memphis and University of Arizona, Tech. Rep.

Wilson, E. B. (1927). Probable inference, the law of succession, and statistical inference. Journal of the American Statistical Association, 22(158), 209-212.