Carlos Manuel Silvestre Cabral

Mini-CCNx: uma plataforma de prototipagem rápida para Redes Orientadas a Conteúdo

Campinas
2013

Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

Carlos Manuel Silvestre Cabral

Mini-CCNx: uma plataforma de prototipagem rápida para Redes Orientadas a Conteúdo

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas para obtenção do título de Mestre em Engenharia Elétrica. Área de Concentração: Engenharia de Computação

Orientador: Christian Rodolfo Esteve Rothenberg
Co-orientador: Maurício Ferreira Magalhães

Este exemplar corresponde à versão final da dissertação defendida pelo aluno Carlos Manuel Silvestre Cabral e orientada pelo Prof. Dr. Christian Rodolfo Esteve Rothenberg

---

Campinas
2013

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

C112m

Cabral, Carlos Manuel Silvestre, 1988-
Mini-CCNx: uma plataforma de prototipagem rápida para redes orientadas à conteúdo / Carlos Manuel Silvestre Cabral. – Campinas, SP : [s.n.], 2013.

Orientador: Christian Rodolfo Esteve Rothenberg.
Coorientador: Maurício Ferreira Magalhães.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes de computadores. 2. Redes de computadores - Arquitetura. 3. Internet (Redes de computação). I. Esteve Rothenberg, Christian Rodolfo,1982-. II. Magalhães, Maurício Ferreira,1951-. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Mini-CCNx: fast prototyping tool for information-centric networks
Palavras-chave em inglês:
Computer networks
Computer networks - Architecture
Internet (Computer networks)
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora:
Christian Rodolfo Esteve Rothenberg [Orientador]
Antonio Marinho Pilla Barcellos
Eleri Cardozo
Data de defesa: 16-07-2013
Programa de Pós-Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Carlos Manuel Silvestre Cabral

Data da Defesa: 16 de julho de 2013

Título da Tese: "Mini-CCNx: Uma Plataforma de Prototipagem Rápida para Redes Orientadas a Conteúdo"

Prof. Dr. Christian Rodolfo Esteve Rothenberg (Presidente): _____

Prof. Dr. Antonio Marinho Pilla Barcellos: _____

Prof. Dr. Eleri Cardozo: _____

Agradecimentos

Agradeço,

a Deus pelos diversos dons com os quais fui agraciado e busco sempre cultivar.

aos Professores Maurício e Christian pela amizade, dedicação e confiança depositada em mim e em meu trabalho.

à Tatiana pelo apoio e pela compreensão de que eu nem sempre podia estar com ela, pois "esse tal de Mestrado consome muito tempo".

a minha família, pelo suporte emocional (e financeiro...) durante todo o período de Mestrado.

aos professores da FEEC e da UNICAMP em geral, pois a maioria deles acabam se tornando mais que professores - tornam-se verdadeiros amigos.

aos colegas de LCA, pelas piadas, cafés, almoços, reclamações e pelo ótimo tempo que passei no laboratório.

a todos que, de alguma forma, contribuíram para minha formação como pessoa.

He who has a why to live for can bear with almost any how.

Viktor Frankl

Resumo

A pesquisa experimental em Redes Orientadas a Conteúdo (ROCs) é crucial para a validação de novas propostas arquiteturais que trazem o conteúdo como o elemento central das redes. Essa dissertação apresenta uma nova ferramenta de prototipagem rápida para o modelo CCN (*Content-Centric Networking*), o Mini-CCNx, que visa preencher uma lacuna existente entre as plataformas experimentais atualmente disponíveis. Usando emulação baseada em contêineres e técnicas de isolamento de recursos, o Mini-CCNx aparece como uma ferramenta flexível, escalável, com baixo custo e alta fidelidade de desempenho possibilitando experimentos em redes emuladas com centenas de nós CCN em um simples *laptop*. Assim, esse trabalho de Mestrado traz como principal contribuição a disponibilização do Mini-CCNx, o primeiro emulador genérico especificamente focado para o desenvolvimento e testes de propostas para o modelo CCN. O Mini-CCNx é publicado com seu código aberto e documentação *online* e, portanto, pode ser melhorado e estendido por qualquer pesquisador da área. Essa dissertação apresenta 18 experimentos utilizando o Mini-CCNx, abrangendo desde análises de escalabilidade e coerência até a distribuição de vídeo e protocolos de roteamento. A reprodução de comportamentos observados em redes reais utilizando o emulador também é uma contribuição importante pois mostra que se pode utilizar o Mini-CCNx para detectar eventuais falhas em aplicações e algoritmos antes dos testes em *testbeds* reais, que certamente são mais complexos e custosos quando comparados ao ambiente emulado. Por fim, as características de baixo custo e flexibilidade indicam que o Mini-CCNx também pode ser utilizado em atividades de ensino como uma ferramenta prática de aprendizado e introdução às ROCs e ao modelo CCN.

Palavras-chave: Redes Orientadas a Conteúdo (ROCs). *Content-Centric Networking* (CCN). Emulação. Emulação baseada em contêineres (EBC). CCNx.

Abstract

Experimental research in Information-Centric Networking (ICN) is crucial to the evaluation of new architectural proposals that bring named pieces of content as the main element of networks. This thesis presents a new fast prototyping tool for the CCN (Content-Centric Networking) model, Mini-CCNx, that aims at filling an existing gap in generally available experimental platforms. Using container-based emulation and resource isolation techniques, Mini-CCNx appears as a flexible, scalable, high-fidelity, and low-cost tool that enables experiments on emulated networks with hundreds of CCN nodes in a commodity laptop. Therefore, this Master's project's main contribution is making Mini-CCNx, the first generic emulator focused on the development and evaluation of new proposals for the CCN model, available. Mini-CCNx is opensource and its documentation is publicly available online so it can be extended by any researcher in the area. This thesis presents 18 experiments using Mini-CCNx, ranging from scalability and coherence analysis to video distribution and routing protocols. The reproduction of real networks behavior using the emulator is also an important contribution because it shows one can use Mini-CCNx to detect flaws in applications or algorithms prior to real testbeds tests, which are certainly more complex and costly when compared to the emulated environment. Finally, Mini-CCNx's low-cost and flexibility indicate it can also be used in teaching activities as a practical ICN and CCN learning tool.

Keywords: Information Centric Networks (ICN). Content-Centric Networking (CCN). Emulation. Container-based emulation (CBE). CCNx.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos e Notação

API	*Application Programming Interface*
CBE	*Container-based Emulation*
CCN	*Content-Centric Networking*
CDN	*Content Delivery Network*
cgroups	*Linux Control Groups*
CS	*Content Store*
DNS	*Domain Name System*
DONA	*Data-Oriented Network Architecture*
EBC	Emulação Baseada em Contêineres
FIB	*Forwarding Information Base*
ICN	*Information Centric Networks*
IP	*Internet Protocol*
NDN	*Named Data Networking*
ROCs	Redes Orientadas a Conteúdo
OSPF	*Open Shortest Path First*
OSPFN	*OSPF for Named-data*
PIT	*Pending Interest Table*
PSIRP	*Publish-Subscribe Internet Routing Paradigm*
P2P	*Peer-to-Peer*
RFC	*Request for Comments*
TCP	*Transmission Control Protocol*
URI	*Uniform Resource Identifier*

Sumário

Capítulo 1

Introdução

Este capítulo apresenta uma breve contextualização da área na qual se insere o trabalho de pesquisa descrito nessa dissertação, uma visão geral das motivações para tal trabalho, os objetivos, o escopo e a organização subsequente do texto.

1.1 Contextualização

Os princípios que guiaram o desenvolvimento da Internet nas décadas de 1960 e 1970 não são mais tão relevantes como eram na época. A arquitetura original foi concebida com base no modelo de comunicação ponto-a-ponto entre *hosts*. Esse modelo funciona perfeitamente para as necessidades originais da época como a necessidade de transferência de arquivos e *login* remoto, aplicações essas focadas na comunicação majoritariamente textual entre pares de *hosts* conhecidos e fixos.

Porém, o uso atual está cada vez mais associado à obtenção de conteúdos e acesso a serviços multimídia (Plagemann, Goebel, Mauthe, Mathy, Turletti & Urvoy-Keller 2006) o que pode ser comprovado pela proliferação das arquiteturas par-a-par (*Peer-to-Peer* ou P2P)[1] e de Redes de Distribuição de Conteúdos (*Content Delivery Networks* ou CDNs)[2] nos últimos anos. O usuário atual se importa com o conteúdo em si e geralmente pouco se importa com a sua localização. Por exemplo, o usuário sabe que quer uma notícia específica, um vídeo do Youtube ou sabe que quer acessar sua conta de banco *online* mas não sabe (ou não deseja saber) em qual máquina o dado ou serviço desejado está hospedado (Koponen, Chawla, Chun, Ermolinskiy, Kim, Shenker & Stoica 2007).

Essa nova dinâmica de utilização da rede impõe uma série de desafios que não foram previstos originalmente pelos desenvolvedores da Internet. Com a tecnologia atual, quando um usuário deseja obter um determinado conteúdo, é necessário fazer um mapeamento entre **o quê** um usuário deseja e **onde** esse conteúdo está (o que envolve uma série de indireções como resoluções de nomes DNS e obtenção do endereço IP para finalmente requisitar o conteúdo). Isso implica em questões sobre mobilidade (já que esse mapeamento precisa ser refeito caso o cliente ou o

[1] Como, por exemplo, as baseadas no famoso protocolo Bittorrent (Pouwelse, Garbacki, Epema & Sips 2005).
[2] Sendo um exemplo notável a CDN Akamai (Akamai 2013).

proprietário do conteúdo mudem de rede)[3] e segurança (pois em geral a segurança do conteúdo é tratada em termos da localização do seu publicador e da conexão sobre a qual ele trafega). Além disso, questões sobre a escalabilidade do modelo tradicional podem surgir dado o aumento explosivo do tráfego global devido ao alto crescimento no número de dispositivos móveis e de conexões máquina-a-máquina (*machine-to-machine* ou M2M), ao aumento da velocidade média das redes de acesso, ao maior número de usuários da Internet (estima-se cerca de 3.4 bilhões de usuários em 2016) e a uma crescente demanda por vídeo. Todos esses fatores irão resultar num tráfego global anual de 1.3 zettabytes em 2016 (Cisco 2012).

Visando endereçar essas questões, as chamadas Redes Orientadas a Conteúdo (ROCs) propõem que o conteúdo seja o elemento central das redes, independentemente de sua localização. A principal abstração na qual as redes se baseiam mudaria de **onde** para **o quê**. Nas ROCs, a infraestrutura da rede participa ativamente no armazenamento (*caching*) e na distribuição dos conteúdos visando atingir um aumento na eficiência e na disponibilidade global (de Brito, Velloso & Moraes 2012). Além disso, as diversas propostas de arquitetura para as ROCs como CCN (*Content-Centric Networking*) (Jacobson, Smetters, Thornton, Plass, Briggs & Braynard 2009), DONA (*Data-Oriented Network Architecture*) (Koponen et al. 2007), PSIRP (*Publish-Subscribe Internet Routing Paradigm*) (Jokela, Zahemszky, Esteve Rothenberg, Arianfar & Nikander 2009) e NetInf (*Network of Information*) (Dannewitz, Kutscher, Ohlman, Farrell, Ahlgren & Karl 2013)), dentre outras, trazem outros conceitos inovadores tais quais conteúdo nomeado, roteamento baseado nesses nomes e segurança aplicada diretamente ao conteúdo (Smetters & Jacobson 2009).

O novo enfoque trazido pelas ROCs, porém, traz também novos desafios como, por exemplo, de que forma nomear os conteúdos de maneira persistente, segura e única; como autenticar os dados de forma a trazer confiabilidade aos mesmos; como rotear utilizando esses nomes; como organizar o *caching* interno à rede para aumentar a eficiência da distribuição de conteúdos, dentre outros. As diversas propostas citadas anteriormente diferem, basicamente, na maneira como respondem a essas questões.

1.2 Motivação

Uma nova proposta de arquitetura de ROC, para tornar-se uma opção concreta para a arquitetura da Internet no futuro, precisa passar por uma intensa e extensa validação. As modelagens matemática e estatística, como feitas por exemplo em (Carofiglio, Gallo, Muscariello & Perino 2011)[4], apresentam de fato novas estratégias e algoritmos a serem adotados. Porém, para que haja efetivamente uma posterior adoção de tal arquitetura pelo mercado, a experiência passada com a ARPANET (Barbaroux 2012) e a NSFNET (Mills & Braun 1988) mostra que a intensa pesquisa experimental e prática, envolvendo o maior número possível de organizações e entidades, tende a ser o verdadeiro catalisador para fazer com que as propostas de arquiteturas possam atingir, por fim, uma aceitação global.

[3] O endereço IP funciona, ao mesmo tempo, como um identificador do *host* e um localizador do mesmo (Gurtov 2008).

[4] Nesse trabalho é feita uma análise estatística do modelo de *caching* e de entrega de conteúdos da arquitetura CCN.

Felizmente, os pesquisadores contam, atualmente, com algumas vantagens no contexto da pesquisa experimental com as quais os desenvolvedores da Internet nas décadas de 1960 e 1970 não puderam contar. Os preços dos recursos computacionais (sem contar a variedade e a disponibilidade) são muito mais baixos que no passado. A popularização e a evolução da virtualização e das arquiteturas de *Cloud Computing* permitem que os experimentos atinjam uma grande escala de maneira muito mais flexível, ágil e com custos cada vez menores. Simuladores e emuladores de rede são cada vez mais abrangentes e flexíveis.

A pesquisa experimental é um fator crucial para avaliar as diferentes soluções apresentadas para os novos desafios trazidos pelas ROCs. O trabalho proposto por esse projeto de pesquisa se insere nesse contexto: identificar uma possível lacuna existente entre as plataformas experimentais atualmente disponíveis para o teste e desenvolvimento das ROCs e propor uma ferramenta que a preencha. A área de Redes de Computadores apresenta um conjunto amplo de plataformas de testes. Contudo, por ser um tema relativamente novo, a pesquisa específica em ROCs conta com um número reduzido (e ainda com um baixo nível de maturidade e documentação) de ferramentas experimentais e, assim, se faz necessário o desenvolvimento de mais opções que possam de fato auxiliar na validação das novas arquiteturas propostas.

1.3 Objetivos

O trabalho tem como objetivo desenvolver uma nova ferramenta experimental especificamente focada nas ROCs. Ela deverá apresentar um conjunto de características que facilitem e agilizem a validação de novas propostas dentro dessa área de pesquisa como: protocolos de roteamento, estratégias de encaminhamento, técnicas de *caching*, desenvolvimento de aplicações orientadas à conteúdo, dentre outras.

Para que tal objetivo seja alcançado, serão definidos cinco requisitos aos quais a ferramenta proposta deverá atender. São eles:

Flexibilidade. Deverá ser possível criar, com agilidade, diversas topologias e cenários de ROCs. O utilizador da ferramenta deverá ter a possibilidade de testar suas idéias em topologias cujos nós e *links* possam ser livremente escolhidos. O utilizador também deverá ter a possibilidade de especificar diferentes parâmetros de configuração de *links* em seus cenários de teste, sendo eles: *(i)* atraso de propagação, *(ii)* banda nominal e *(iii)* porcentagem de perda de pacotes.

Escalabilidade. Deverá ser possível criar topologias com um número suficientemente grande de nós (algumas dezenas). A validação de uma nova proposta na área de Redes de Computadores tipicamente começa com uma topologia simples, com cerca de dois ou três nós, apenas para uma validação inicial do conceito. Uma vez concluída essa etapa, é comum partir para um cenário mais complexo, com um número maior de nós. A ferramenta proposta deverá acomodar tal dinâmica típica de pesquisa, desde a etapa inicial até o cenário mais complexo.

Baixo Custo. A ferramenta poderá ser executada em um *hardware* convencional, ou seja, não necessitará de uma infraestrutura completa de rede com *switches*, roteadores, enlaces e servidores para sua execução. É claro que a ferramenta se beneficiará de um *hardware* com mais memória e maior poder de processamento, porém o uso de uma configuração básica não deverá

ser impeditivo para a avaliação de cenários mais complexos, como por exemplo uma topologia com dezenas de nós.

Realismo. Tal qual descrito em (Handigol, Heller, Jeyakumar, Lantz & McKeown 2012), pode-se definir duas dimensões de realismo para uma plataforma de testes:

- *Realismo Funcional:* O sistema de testes deve apresentar o mesmo comportamento de um ambiente real e o código executado nele deve ser o mesmo código executado em um *hardware* real.
- *Realismo Temporal:* O desempenho do sistema de testes deve ser próximo, ou seja, da ordem de grandeza do desempenho do sistema real correspondente.
- *Realismo de Tráfego:* O sistema de testes deve ser capaz de gerar e receber tráfego real, vindo da Internet ou de uma rede local.

A ferramenta proposta neste trabalho deverá atender a tais requisitos de realismo.

Facilidade de Uso. Apesar de ser um requisito relativamente subjetivo, considerar-se-á que a ferramenta apresenta facilidade de uso se não exigir especificamente a utilização de codificação para a definição dos cenários de teste e, também, se apresentar algumas facilidades como configuração via interface gráfica e definição automática de topologias pré-definidas. Também é desejável que o tempo despendido na configuração e definição do cenário de testes não seja demasiadamente elevado - idealmente, ele deve ser menor que o tempo gasto no teste em si.

Uma ferramenta que atenda a todo esse conjunto de requisitos certamente será de grande utilidade para a pesquisa experimental na área de ROCs, o que é justamente o principal objetivo do trabalho proposto. Além disso, o atendimento aos requisitos de facilidade de uso, baixo custo e realismo sugerem fortemente que tal ferramenta possa também ser utilizada na atividade de ensino dos novos conceitos propostos por essa área de pesquisa, trazendo para a sala de aula uma experiência prática com aplicações orientadas à conteúdo, novos protocolos e novas arquiteturas.

1.4 Escopo

Definidos os requisitos e os objetivos gerais, é necessário delimitar o escopo da plataforma de testes proposta nesse texto. Para tal, algumas escolhas (seguidas de suas respectivas justificativas) devem ser feitas para estreitar o foco desse trabalho. São elas:

Modelo de ROC. É necessário escolher um dos atuais modelos propostos para as ROCs para ser endereçado pela plataforma de testes. Seria impraticável fazer uma plataforma que endereçasse diversas propostas simultaneamente pois cada uma delas apresenta um nível diferente de especificação, maturidade, implementação, etc. Optou-se pelo modelo CCN (*Content-Centric Networking*) (Jacobson, Smetters, Thornton, Plass, Briggs & Braynard 2009). Proposto pelo pioneiro da Internet Van Jacobson (reconhecido por seu trabalho em algoritmos de controle de fluxo para o modelo TCP/IP (Jacobson 1988)(ACM Awards 2001)), o CCN é desenvolvido por

um grupo de universidades americanas no contexto do Projeto NDN (*Named Data Networking Project*) (NDN Project 2012a), contando com uma especificação madura e desenvolvimento ativo (CCNx 2013). Além disso, tal modelo tem sido escolhido como base por um grande número de publicações na literatura recente da área de ROCs em todo o mundo.

Sistema Operacional. Optou-se por utilizar o Linux como o Sistema Operacional alvo da ferramenta proposta. O Linux é utilizado amplamente pela comunidade científica por ser um sistema de código aberto, de reconhecida eficiência de desempenho, por contar com uma série de boas ferramentas de apoio focadas em redes, dentre outros motivos.

Plataforma de Divulgação. Para que a proposta atinja um grande número de pesquisadores da área de ROCs, seria interessante utilizar uma plataforma aberta e pública de divulgação da documentação, tutoriais e do próprio código da ferramenta proposta nesse texto. Para isso, optou-se pelo GitHub (GitHub 2013), uma plataforma de versionamento de *software online*, gratuita e de fácil utilização.

1.5 Desambigüação: CCN x NDN

Em trabalhos recentes na área de ROCs, tem havido uma certa ambigüidade com relação ao nome da arquitetura que essa dissertação aborda. Originalmente (Jacobson, Smetters, Thornton, Plass, Briggs & Braynard 2009), o modelo foi denominado *Content-Centric Networking* ou CCN. Porém, alguns artigos atuais se referem a essa mesma arquitetura como *Named Data Networking* ou NDN, ou seja, como o nome do projeto que está implementando a proposta original. Esse texto adota a seguinte nomenclatura: quando se estiver falando sobre a arquitetura, utilizar-se-á o nome CCN; o termo NDN será utilizado exclusivamente para se referir ao projeto que especifica e implementa o modelo.

O restante desse texto está organizado da seguinte forma: o capítulo 2 apresenta uma revisão bibliográfica consistindo de uma análise das características do modelo CCN e de uma análise das ferramentas de teste atualmente disponíveis para a avaliação de propostas baseadas nesse modelo. O capítulo 3 apresenta os detalhes da arquitetura da ferramenta proposta nesse trabalho. O capítulo 4 apresenta vários experimentos realizados para a validação da ferramenta. O capítulo 5 finaliza o texto com as conclusões e trabalhos futuros. O apêndice A apresenta as publicações geradas durante esse trabalho de pesquisa e o apêndice B traz uma figura com a topologia do *testbed* NDN utilizado nos experimentos do capítulo 4.

Capítulo 2

Revisão Bibliográfica

Este capítulo apresenta a revisão bibliográfica feita no contexto desse trabalho de pesquisa e consiste de duas partes. Na primeira, é feita uma descrição do modelo CCN. Na segunda, faz-se uma análise comparativa das diversas plataformas atualmente disponíveis para a avaliação e testes de propostas para o modelo. Essa análise pode ser considerada uma contribuição deste trabalho para pesquisadores que estejam na etapa de decisão de qual ferramenta utilizar para validar sua proposta para o modelo CCN.

2.1 Caraterísticas e Propostas do Modelo CCN

Como explicado no primeiro capítulo, optou-se por adotar o modelo CCN (*Content-Centric Networking*) (Jacobson, Smetters, Thornton, Plass, Briggs & Braynard 2009) como base para a plataforma de testes proposta neste texto. Portanto, serão apresentadas mais detalhadamente as principais características e propostas trazidas por tal modelo. Isso também é importante para introduzir conceitos e nomenclaturas que serão utilizadas nos próximos capítulos.

Não faz parte do escopo desse texto apresentar e analisar as diversas características dos outros modelos de ROCs. Nessa seção, serão feitas breves referências aos outros modelos apenas para efeito de contraste com as características do modelo CCN. Uma análise comparativa mais detalhada pode ser encontrada em (de Brito et al. 2012) e (Ahlgren, Dannewitz, Imbrenda, Kutscher & Ohlman 2012).

2.1.1 Nomeação

O modelo CCN utiliza nomes hierárquicos e legíveis (formados por sequências de caracteres e números) para identificar os conteúdos, de forma semelhante a identificadores uniformes de recursos (*Uniform Resource Identifiers* - URIs) (Mealling & Denenberg 2012). Tais nomes possuem característica semântica, ou seja, os componentes hierárquicos que os formam trazem algum tipo de informação sobre o conteúdo como, por exemplo, versão, formato ou propriedade. A Figura 2.1 mostra um exemplo de formação do nome de um vídeo. Essa característica semântica permite, por exemplo, que o consumidor derive qual o nome da próxima versão (por exemplo, v1, v2, v3, etc.) ou qual o tempo de vídeo desejado através do *timestamp* (para saltar

para frente ou para trás na execução do vídeo). Cada sequência de caracteres entre as barras no nome (como, por exemplo, `br.unicamp`, `feec` e `institucional.avi` na Figura 2.1) é chamada de componente do nome.

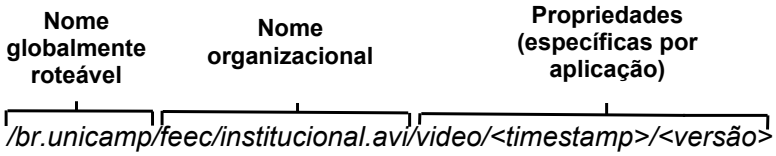


Figura 2.1: Exemplo de um nome de conteúdo no modelo CCN

Porém, a principal motivação por trás de tal esquema de nomeação é justamente a possibilidade de agregar prefixos, de forma análoga ao que é feito com os endereços IP. O intuito é buscar a escalabilidade comprovada trazida pela agregação e também utilizar algoritmos e técnicas já amplamente testados. Por exemplo, com o nome da Figura 2.1, um possível esquema de roteamento poderia divulgar apenas o prefixo `/br.unicamp/` para domínios externos e o prefixo `/br.unicamp/feec/` para o domínio interno da Unicamp. Propostas de ROCs como DONA (Koponen et al. 2007) propõem a utilização de nomes planos (*flat*) visando atribuir uma relação mais direta entre o nome e conteúdo - o nome seria o *hash* criptográfico do conteúdo ou, também, o *hash* criptográfico de uma chave pública associada ao proprietário do conteúdo. Porém esse esquema de nomeação, além de não apresentar semântica, impõe sérias restrições à agregação de nomes, o que certamente leva a questionamentos sobre a escalabilidade de um modelo de nomeação desse tipo[1].

2.1.2 Modelo Básico de Comunicação

O CCN é um modelo baseado no consumidor e conta com dois tipos de pacotes: *Interest* e *Data*. Um consumidor expressa seu interesse inserindo o nome do conteúdo desejado em uma mensagem *Interest* e a envia. Eventualmente, um produtor que possui tal conteúdo receberá essa mensagem e o enviará ao consumidor através de uma mensagem *Data*. A mensagem *Data* é enviada somente em resposta a uma mensagem *Interest* e consome esse *Interest*. Ou seja, essas mensagens possuem uma correspondência um-a-um e estabelecem um controle de fluxo similar ao feito pelo TCP (Jacobson 1988). Diz-se que um pacote *Data* satisfaz um *Interest* se o nome de conteúdo presente no *Interest* é o nome no pacote *Data*. A literatura tem também utilizado o termo *Content Object* para se referir aos pacotes *Data* em trabalhos mais recentes. O capítulo 4 desse texto utiliza essa segunda nomenclatura justamente por ela ser a mais utilizada em trabalhos recentes ao quais o projeto proposto nesse texto será comparado. Porém, para todos os efeitos, ambas as nomenclaturas, *Data* e *Content Object*, possuem o mesmo significado: a mensagem contendo o conteúdo que é enviada ao consumidor em resposta a um *Interest* recebido.

[1]Em (Ghodsi, Shenker & Berkeley 2011), os autores fazem um esboço de um esquema de agregação de nomes planos. Porém, não houve um maior desenvolvimento desse conceito tanto nesse próprio artigo ou em outros trabalhos na literatura.

A Figura 2.2 mostra uma representação gráfica dos pacotes do modelo CCN. O *Interest* pode possuir alguns parâmetros opcionais como seletores de escopo, preferência de ordem e filtro de exclusão. *Interests* também possuem um *nounce* aleatório e único para que duplicatas recebidas por diferentes interfaces do nó possam ser descartadas, eliminando assim a possibilidade de *loops*. O *Data*, além do nome e do conteúdo em si, também carrega a assinatura e algumas informações opcionais, como identificador do publicador e localização da chave, para auxiliar na verificação da assinatura.

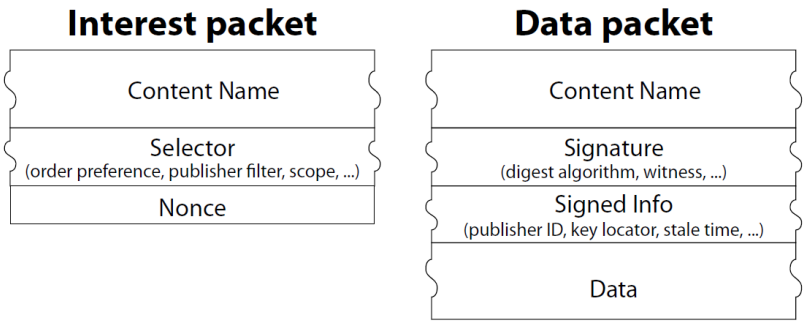


Figura 2.2: Mensagens *Interest* e *Data* do modelo CCN (reproduzido de (Jacobson, Smetters, Thornton, Plass, Briggs & Braynard 2009))

2.1.3 Encaminhamento e *Caching*

Os nós CCN são capazes de armazenar temporariamente os pacotes *Data* em uma estrutura denominada CS (*Content Store*). Essa estrutura é análoga à memória *buffer* de um roteador IP mas com a diferença de que os nós CCN buscam armazenar os pacotes *Data* pelo maior tempo possível, pois um mesmo conteúdo pode ser compartilhado por muitos consumidores. Essa funcionalidade de maximizar a probabilidade de compartilhamento de conteúdos, e portanto minimizar o consumo de banda, é chamada de *caching* interno à rede (*in-network caching*).

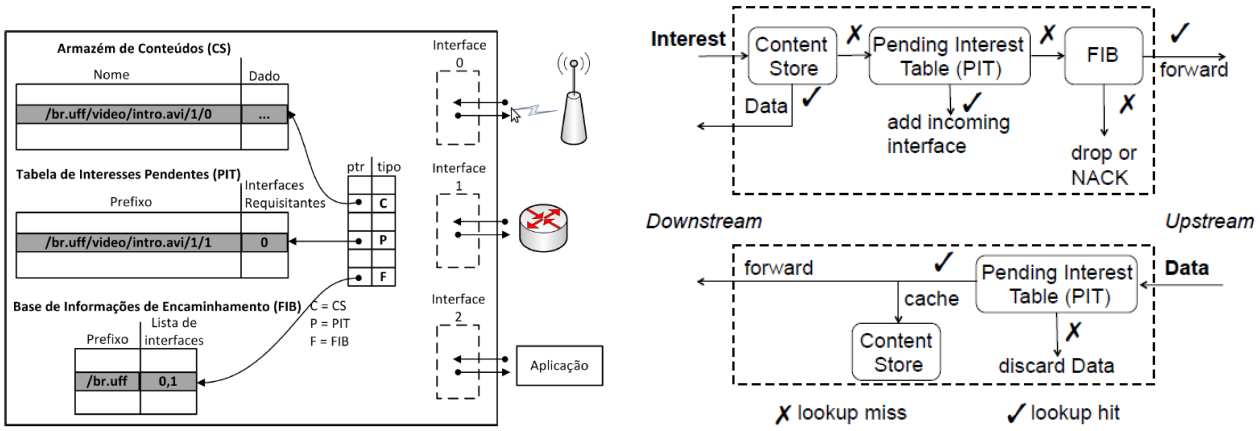
Quando um pacote *Interest* chega ao nó, se o conteúdo requisitado estiver armazenado no CS, o pacote *Data* é imediatamente retornado. Caso contrário, o nó insere o nome do conteúdo desejado e a interface pela qual o pacote foi recebido na PIT (*Pending Interest Table*). A PIT registra, portanto, todos os *Interests* que passaram pelo nó em direção ao produtor do conteúdo (o chamado *upstream*) para que, quando o pacote *Data* voltar (*downstream*), ele possa ser encaminhado corretamente em direção ao(s) consumidor(es). É importante frisar que apenas *Interests* são roteados no CCN; os pacotes *Data* simplesmente seguem as "migalhas"[2] (entradas na PIT) deixadas no caminho de volta ao consumidor. Entradas na PIT são apagadas assim que o pacote *Data* correspondente é encaminhado ao consumidor ou por temporização (no caso em que o *Interest* não encontra um pacote *Data* correspondente).

Após a marcação na PIT, o pacote *Interest* é encaminhado para a FIB (*Forwarding Information Base*) do nó na qual é feita uma procura de prefixo-mais-longo (*longest-prefix match*) e

[2]O termo "migalhas" é utilizado em referência ao conto de fadas "Hänsel und Gretel" dos irmãos Grimm no qual dois irmãos deixavam migalhas de pão no chão da floresta para marcar o caminho de volta para casa.

decide-se, por fim, por qual interface enviar o *Interest*. Caso não haja uma entrada correspondente, o *Interest* é descartado. A FIB é análoga à tabela de encaminhamento dos dispositivos IP (que também é chamada de FIB) e armazena prefixos de nome e as interfaces de saída correspondentes. Como no modelo IP, os protocolos de roteamento são responsáveis por popular a FIB.

A Figura 2.3 sumariza as estruturas de cada nó e a dinâmica de encaminhamento.



(a) Estruturas de um nó CCN (reproduzido de (de Brito et al. 2012))

(b) Dinâmica do encaminhamento CCN (reproduzido de (Yi et al. 2012))

Figura 2.3: Estruturas de um nó CCN e dinâmica de encaminhamento

2.1.4 Camada de Estratégia e Mobilidade

No modelo CCN é possível que, para um dado prefixo na FIB, existam múltiplas interfaces de saída possíveis. A camada de estratégia é responsável por determinar dinamicamente qual dessas interfaces é a melhor escolha no momento do encaminhamento. Isso é feito considerando as condições da própria rede com a camada de estratégia podendo optar pela interface que possui uma menor média de RTT (*round-trip time*) na entrega do conteúdo ou pela interface que apresenta a menor taxa de *timeouts* observados na PIT.

É importante notar que no modelo IP o encaminhamento fica restrito às *spanning trees* e assim é difícil se beneficiar da existência de mais de uma interface de saída ou se adaptar rapidamente a situações onde há mobilidade. Como o CCN é focado no conteúdo e não em nós (*hosts*), não é necessário obter um identificador de camada 3 (endereço IP) ou fazer um mapeamento desse identificador com um de camada 2 (como por exemplo um endereço MAC). Assim, mesmo que a conectividade da rede mude rapidamente, o CCN pode sempre recomeçar a troca de dados assim que uma conexão física for estabelecida.

2.1.5 Roteamento

Em geral, qualquer modelo de roteamento utilizado nas redes IP pode ser aplicado no modelo CCN já que ambos os modelos apresentam encaminhamento e esquema de nomeação similares.

Os nomes CCN apresentam a mesma característica no que é relativo ao roteamento (agregação hierárquica com *longest-prefix match*). Além disso, o CCN apresenta menos restrições que o IP na questão do uso de múltiplas fontes ou destinos por evitar laços através da utilização de *nounces* para cada pacote (Jacobson, Smetters, Thornton, Plass, Briggs & Braynard 2009). Essa semelhança entre os modelos também pode ser útil na implementação gradativa do modelo CCN, o qual poderá conviver com o IP sem grandes dificuldades. O OSPFN (OSPFN 2012) é um exemplo de protocolo de roteamento para CCN e será detalhado nos capítulos 3 e 4 desse texto.

O roteamento utilizado pelo CCN pode ser chamado de não-hierárquico (ou não-estruturado), ou seja, não exige a presença de sistemas dedicados ao armazenamento de informações de roteamento ou estruturas hierárquicas pré-definidas para a organização dos nós. Isso se assemelha à maneira como é feito o roteamento na Internet atual. Diferentemente do CCN, algumas propostas de ROC, como DONA (Koponen et al. 2007), adotam uma estratégia de roteamento hierárquico, ou seja, assumem que os roteadores estão dispostos em uma estrutura em árvore e exploram esse fato para obter e disseminar informações associadas ao roteamento na rede. Essa organização hierárquica dos nós de fato facilita a operação dos algoritmos de roteamento, porém insere uma certa rigidez quanto à disposição dos roteadores na rede e também quanto à inserção e remoção dos mesmos ao longo do tempo, o que certamente pode reduzir a flexibilidade na definição de topologias.

2.1.6 Segurança

O CCN utiliza o conceito de segurança baseada no conteúdo: todos os pacotes *Data* (incluindo o nome, o conteúdo e outras informações) são autenticados pela assinatura do publicador e conteúdos privados são protegidos com criptografia. As redes IP atuais focam em autenticar a conexão sobre a qual o conteúdo trafega. O CCN, porém, insere as informações de segurança (assinatura e informações para obtenção da chave) nos próprios pacotes de dados. O publicador pode atribuir com flexibilidade qualquer nome que desejar para o conteúdo e assinar com sua chave, em oposição ao que é feito em propostas de ROCs que utilizam nomes auto-certificáveis (nas quais um *digest* criptográfico do conteúdo e possivelmente da chave pública do publicador são utilizado para a formação do nome, como proposto em (Ghodsi et al. 2011)).

Os conteúdos CCN são publicamente autenticáveis - as assinaturas nos pacotes seguem o padrão desse tipo de assinatura e qualquer nó no caminho, não apenas a fonte ou o destino, pode verificar o vínculo estabelecido entre o nome e o respectivo conteúdo utilizando a assinatura disponível no pacote e a chave pública. Aplicações CCN devem resolver os problemas tradicionais de gerenciamento de chaves, como associar chaves públicas a indivíduos e organizações (certificação).

Os próprios princípios de arquitetura do CCN ajudam a proteger o modelo de típicos ataques de redes. Autenticar todo o conteúdo (incluíndo o roteamento) previne a modificação de dados ou o uso de *spoofing*. O próprio fato das mensagens CCN focarem apenas no conteúdo e não em *hosts* também dificulta o envio de pacotes maliciosos para alvos específicos. A correspondência um-a-um existente especificamente entre um *Interest* e um *Data* previne o envio de tráfego excessivo. Ataques de negação de serviço (*Denial of Service* - DoS) são possíveis, como um

ataque de *Interest flooding*. Porém, os roteadores CCN podem utilizar técnicas como limitação do número de *Interests* que eles encaminham com um dado prefixo para mitigar tais ataques.

2.1.7 Implementação

A proposta do modelo CCN vem sendo utilizada como base para o projeto NDN (*Named Data Networking*) (Zhang, Estrin, Bruke, Jacobson, Thornton, Smetters, Zhang, Tsudik, Krioukov, Massey, Papadopoulos, Abdelzaher, Wang, Crowley & Yeh 2010), o qual visa desenvolver o modelo original, trazendo novas propostas, fazendo provas de conceito e implementando novos protocolos, estratégias e aplicações. Várias Universidades e institutos americanos de renome são membros do projeto.

Dentre os vários trabalhos desenvolvidos, o PARC (*Palo Alto Research Center - Xerox*) vem trabalhando ativamente desde 2010 em uma implementação de referência do modelo CCN denominada CCNx (CCNx 2013). O CCNx é composto de uma série de elementos dos quais o principal é o *daemon* `ccnd`, responsável por implementar todo o plano de encaminhamento CCN (com as estruturas PIT, CS e FIB detalhadas anteriormente). Outro componente importante é o `ccnr`, um repositório de longo prazo para conteúdos. O CCNx é majoritariamente implementado em C e reside no espaço de usuário em oposição às implementações otimizadas e amplamente testadas do modelo TCP/IP que atualmente executam no espaço do *kernel* dos Sistemas Operacionais. O CCNx possui também API's para as linguagens C e Java, além de uma versão focada no Sistema Android para *smartphones* e *tablets*. Por fim, também foram desenvolvidas uma série de aplicações auxiliares para o envio personalizado de pacotes (`ccnpeek`/`ccnpoke`), para publicação e requisição de arquivos (`ccngetfile`/`ccnputfile`), inspeção da tabela de encaminhamento (`ccndstatus`), dentre outras.

As conexões CCNx atualmente operam sobre IP, ou seja, é necessária a criação de uma conectividade básica entre os nós (túneis UDP ou TCP) para que as mensagens CCN trafeguem entre os nós da rede. Porém, é importante frisar que isso não é uma premissa ou uma necessidade básica do modelo. O CCNx apenas utiliza tais túneis para obter a conectividade necessária, semelhante ao serviço fornecido pela camada 2 (enlace) do modelo OSI (*Open Systems Interconnection*) para o IP (camada 3). Em momento algum é feito um mapeamento entre conteúdo e localização. Na realidade, seria totalmente possível utilizar o protocolo IP sobre CCN inserindo, por exemplo, o endereço IP no campo de nome dos pacotes CCN (Jacobson, Smetters, Thornton, Plass, Briggs & Braynard 2009). Essa possibilidade é bem interessante em um cenário de transição entre os dois modelos pois permite que domínios baseados em modelos diferentes possam se comunicar utilizando essa característica.

2.1.8 Desafios

Apesar de ser altamente promissor, o modelo CCN possui diversos desafios em aberto. Essa subseção apresentará alguns deles e também mostrará algumas propostas de solução a tais desafios trazidas por trabalhos recentes da literatura.

Escalabilidade

Talvez a escalabilidade seja o maior desafio que o modelo necessita endereçar. Como explicado anteriormente, para cada *Interest* recebido por um nó CCN, uma entrada na PIT é criada contendo o nome do conteúdo requisitado e a interface pela qual tal mensagem entrou no nó. Essa entrada é removida somente quando o *Data* correspondente for recebido no nó e encaminhado na direção do consumidor. Portanto, a PIT precisa ser suficientemente grande para armazenar todos os *Interests* pendentes. Por exemplo, em (You et al. 2012), faz-se a seguinte estimativa: se a taxa média de chegada de *Interests* for da ordem de 125 milhões de pacotes por segundo e o RTT (*round-trip time*) médio for de 80ms, a PIT precisaria armazenar um número da ordem de 10^7 entradas. Além disso, utilizar alguma técnica de agregação para as entradas da PIT (o que reduziria seu tamanho) é um desafio pois é preciso fazer um *matching* exato entre o nome armazenado na PIT e o nome dentro do pacote *Data* recebido.

Visando melhorar a escalabilidade global do CCN, em (You et al. 2012) é proposta a adoção de uma nova implementação para a PIT denominada DiPIT (*Distributed* PIT). A idéia é criar uma memória de menor tamanho, chamada PITi, para cada interface i do nó, a qual é responsável por armazenar entradas referentes somente a essa interface. Cada PITi é implementada na forma de um *Bloom-filter* contador (Ahlgren et al. 2012), o que resulta em um espaço menor de armazenamento e buscas mais rápidas. A Figura 2.4 apresenta uma visão geral das estruturas em um nó CCN com as várias PITi associadas a cada interface.

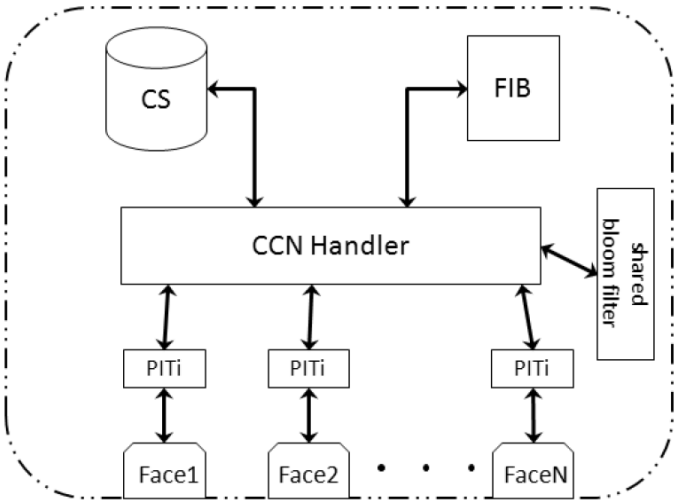


Figura 2.4: Arquitetura geral de um nó CCN segundo a proposta DiPIT (reproduzido de (You et al. 2012))

Em (Yuan, Song & Crowley 2012), é feita uma análise detalhada sobre as estruturas de dados utilizadas atualmente pelo CCNx para implementar os principais componentes lógicos do modelo CCN (PIT, FIB e CS). Além disso, também é feita uma análise de desempenho do processamento de mensagens em um nó CCN. Várias propostas são feitas buscando melhorar a escalabilidade como: simplificação do fluxo de processamento das mensagens através da remoção de uma tabela *hash* compartilhada entre as implementações internas da PIT e da FIB; buscar mecanismos de *lookup* com custo constante mapeando internamente nomes mais longos em nomes curtos; utilizar

os componentes dos nomes como unidades de *matching* ao invés dos caracteres individuais; utilização de estruturas como *d-left hash tables* (Broder & Mitzenmacher 2001) para implementar componentes que requerem atualizações rápidas e constantes, dentre outras.

Nomeação

Outro desafio a ser enfrentado pelo modelo é o fato dos nomes dos conteúdos serem mais complexos quando comparados aos endereços IP (Ghodsi et al. 2011). Apesar de poderem utilizar agregação como o IP, os nomes CCN (compostos por sequências de caracteres de tamanho arbitrário) tendem a ser mais longos, dinâmicos e numerosos, em oposição às estruturas fixas e bem definidas do IPv4 ou IPv6. Dessa forma, espera-se que as tabelas de roteamento sejam muito maiores e complexas, exigindo novas implementações e algoritmos para o processamento dos nomes.

Em (Dai, Liu, Chen & Wang 2012), os autores exploram a estrutura hierárquica do nome CCN para propor um novo método de representação interna chamado *Name Component Encoding* (NCE) no qual cada componente do nome recebe um código (número inteiro). Esse códigos são utilizados para construir uma estrutura chamada *Encode Name Prefix Trie* (ENPT) a qual permite que os códigos sejam utilizados em buscas, atualizações e apagamentos na PIT resultando, assim, em uma operação mais rápida durante essas operações.

Caching

O *caching* é um elemento central do modelo CCN (e provavelmente de qualquer arquitetura de ROC) devido à necessidade de se tratar, de maneira eficiente, a crescente demanda pela distribuição de conteúdos entre os usuários da Internet e reduzir, assim, a utilização geral de banda no núcleo da rede. Apesar da extensa literatura existente sobre estratégias de *caching*, o modelo CCN traz novos desafios como o fato dos nós CCN poderem ser organizados em topologias arbitrárias (em oposição às estratégias atuais que exploram uma topologia pré-definida), a necessidade de armazenar *chunks* de dados de pequena granularidade (em oposição às arquiteturas originais que armazenam objetos maiores) e ao grande tamanho dos *caches* dos nós necessários para o armazenamento do crescente tráfego da Internet atual (Rossini & Rossi 2012).

Em (Rossini & Rossi 2012), é feito um estudo detalhado sobre o desempenho geral do *caching* considerando *(i)* a influência da localização dos usuários que fazem as requisições e *(ii)* a eficiência na utilização de múltiplos repositórios para armazenamento de um mesmo conteúdo. Os autores mostram que explorar a localização dos usuários pode simplificar a criação de estratégias de *caching* mas não é suficiente para trazer grandes ganhos na escalabilidade na distribuição de conteúdos. Também é mostrado que a existência de múltiplos repositórios para um mesmo conteúdo pode prejudicar o desempenho geral do sistema devido ao surgimento de uma situação de competição pelos *caches* dos roteadores resultando na remoção de conteúdo previamente armazenado em múltiplos caminhos e na redução da probabilidade geral de *cache hit*. Por fim, os autores mostram que a utilização de uma estratégia aleatória de substituição dos conteúdos em *caching* pode ser utilizada sem grandes perdas de desempenho, mas com um custo de implementação bem menor, se comparada à estratégia padrão de "menos usado recentemente" (LRU - *Least Recent Used*).

Aplicações

É necessário explorar como a nova arquitetura de rede proposta pelo CCN afeta o desenvolvimento de aplicações no nível de usuário pois, no fim, suportar as aplicações e os usuários finais é o objetivo principal de qualquer arquitetura de rede.

Em (Kulinsk & Burke 2012), um trabalho no qual é desenvolvida uma aplicação de *streaming* de vídeo, os autores já notam uma diferença com relação às aplicações similares utilizadas no contexto do modelo TCP/IP: a aplicação cliente (no caso específico, aquela que faz as requisições pelos vídeos) é mais complexa do que a aplicação servidora. Isso ocorre devido ao fato de o modelo CCN ser baseado no cliente (*receiver-driven*), ou seja, todo conteúdo na arquitetura é enviado somente em resposta a uma requisição feita por um cliente - não é possível haver um envio "espontâneo" de conteúdo. Dessa forma, no caso da aplicação de vídeo, o cliente é totalmente responsável pelos dados que está recebendo, ou seja, é responsável por estimar a taxa de envio de *Interests*, detectar *timeouts*, requisitar conteúdos anteriores ou posteriores ao tempo atual, gerenciar o *buffer*, dentre outras tarefas. A aplicação servidora deve, simplesmente, enviar eficientemente os pacotes *Data* requisitados[3].

Em (Burke, Horn & Marianantoni 2012), é apresentada outra interessante categoria de aplicações: a utilização do modelo CCN no contexto de sistemas de automação de construção (*Building Automation Systems* - BAS). Mais especificamente, foi desenvolvida uma aplicação para controlar as luzes de um estúdio de televisão na Universidade da Califórnia em Los Angeles (UCLA). Esse é um exemplo interessante de como explorar a semântica dos nomes CCN. Por exemplo, utilizou-se nomes do tipo "/<raiz_topológica>/lighting/<prédio>/<sala>/<luz>" para acessar uma luz específica do estúdio. Segundo os autores, essa aproximação semântica entre a aplicação e a rede facilitou o desenvolvimento do projeto. Além disso, a segurança intrínseca ao modelo CCN trouxe grandes ganhos ao sistema de autenticação permitindo que cada componente do sistema de iluminação tivesse seu par de chaves pública e privada garantindo assim uma facilidade maior na configuração de segurança dos componentes. Anteriormente, a segurança era garantida por separação física ou virtual (VLANs) das redes dos diversos componentes do sistema de iluminação (atuadores, luzes, controladores de energia e outros dispositivos).

Outros Desafios

Os desafios do modelo certamente não se encerram por aqui. Com relação ao roteamento, por exemplo, a literatura apresenta um número bem menor de trabalhos se comparado à quantidade de artigos e projetos que analisam a escalabilidade e aplicações para o modelo. É preciso definir com mais detalhes quais as diferenças entre os contextos de roteamento intra e inter domínios e também analisar as implicações de escala global para o roteamento, já que o número de conteúdos a ser endereçado é maior que o número de endereços IP tratados pelo roteamento global atual em várias ordens de grandeza - estima-se que a Internet atual trate cerca de 1 bilhão de endereços IP mas a quantidade de conteúdos seria, pelos menos, da ordem de 1 trilhão (Perino & Varvello 2011).

[3]Obviamente, outra parte essencial à aplicação é a questão da captura e renderização dos vídeos em si. Porém, a discussão aqui foca especificamente na parte de interação da aplicação com a rede.

A interação entre os planos de encaminhamento e roteamento também pode ser melhor explorada visando atingir uma maior eficiência geral em ambos os planos. Em (Yi et al. 2012) é mostrado como o CCN pode prover um plano de encaminhamento rico em informações da condição da rede em tempo real. Seria interessante analisar como utilizar tais informações em benefício do roteamento, que poderia tomar as decisões sobre as melhores rotas não somente com base em métricas tradicionais (como, por exemplo, número de *hops*) mas também em condições de tráfego e congestionamento de tempo real fornecidas pelo plano de encaminhamento.

O modelo CCN pode certamente ser considerado um modelo promissor para as ROCs. Apesar dos desafios citados aqui (e outros como mobilidade, modelo de segurança ou comportamento em ambientes sem-fio), o CCN certamente é o modelo mais maduro encontrado na atualidade, tanto com relação à implementação (CCNx) quanto à especificação e modelagem. Dessa forma, o CCN se apresenta como um forte candidato à adoção do conceito de ROCs em redes de produção (Perino & Varvello 2011). Resta agora saber se, futuramente, as questões econômicas e técnicas convergirão no sentido de uma adoção efetiva das propostas trazidas pelas ROCs nas redes de produção.

2.2 Plataformas Experimentais para o modelo CCN

Serão analisadas agora as diversas opções que um pesquisador tem, atualmente, para validar suas propostas como, por exemplo, uma nova aplicação ou um novo protocolo para o modelo CCN. Será apresentada uma taxonomia com a categorização das ferramentas, uma análise das plataformas atualmente disponíveis e, por fim, um resumo comparativo das características dessas plataformas. Ao longo dessa seção serão referenciados os requisitos definidos na seção "1.3 Objetivos" do primeiro capítulo desse texto no intuito de avaliar o atendimento desses requisitos pelas ferramentas atualmente disponíveis.

2.2.1 Simuladores

Simuladores de redes são, tipicamente, simuladores de eventos discretos, ou seja, modelam a operação do sistema como uma sequência discreta de eventos ao longo de um tempo virtual (Chong 1994). Cada evento ocorre em um certo instante desse tempo e resulta numa mudança do estado do sistema (envio de um pacote, por exemplo). Entre dois eventos consecutivos assume-se não haver mudanças no sistema e, portanto, a simulação pode saltar diretamente de um evento para o outro. Esse fato confere grande escalabilidade aos simuladores, que podem chegar a simular milhares de nós em uma máquina de configuração modesta. Simuladores de rede tipicamente modelam os dispositivos (*hosts*, roteadores, etc), enlaces e o tráfego.

O ndnSIM (ndnSIM 2013) é um módulo do ns-3 (ns-3 2013) que implementa o modelo CCN trazendo novas classes que representam as diferentes entidades do modelo como PIT, FIB, CS, estratégias de encaminhamento, dentre outras. Ele foi desenvolvido no contexto da validação de um novo plano de encaminhamento para o CCN no qual a interface de saída para um *Interest* é escolhida não somente com base na escolha feita pelo roteamento, mas também por estatísticas de tempo real sobre o estado da rede (Yi et al. 2012). A Figura 2.5 mostra os principais componentes introduzidos ao ns-3 pelo ndnSIM.

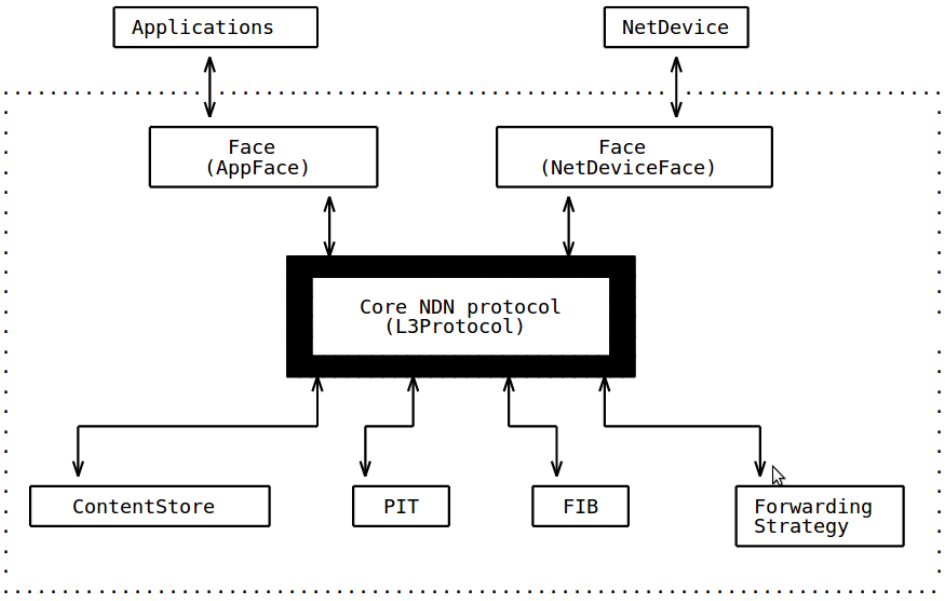


Figura 2.5: Principais componentes implementados no ndnSIM (reproduzido de (ns-3 2013))

Um módulo interessante é o *Forwarding Strategy* (estratégia de encaminhamento). Através da extensão desse módulo é possível avaliar uma estratégia de encaminhamento completamente nova para o modelo CCN sem a necessidade de alterar os outros componentes.

Outro simulador utilizado na área é o ccnSim (Rossi & Rossini 2011). O ccnSim é escrito em C++ sobre o *framework* OMNeT++ (OMNeT++ 2013) e foca principalmente na análise de diferentes cenários de *caching*, permitindo trabalhar com um número da ordem de 10^6 *chunks* (partes de um conteúdo CCN). Vários trabalhos relacionados a diferentes estratégias de *caching*, como (Rossi & Rossini 2012) e (Chiocchetti, Rossi, Carofiglio, Lucent & Labs 2012), utilizaram o ccnSim como plataforma de testes.

Ambos, ndnSim e ccnSim, apresentam as vantagens e desvantagens de qualquer simulador. São certamente flexíveis e escaláveis pois permitem a criação de diversos cenários de maneira rápida e com um número elevado de nós. Em geral, também não requerem um *hardware* potente e, portanto, pode-se considerar que são ferramentas de baixo custo.

Porém, a grande desvantagem dos simuladores é a falta de realismo. O código (ou *script*) utilizado no simulador não é o mesmo código que rodaria nativa e diretamente em um sistema Linux. Por exemplo, o código de um consumidor CCN implementado no ndnSIM precisaria ser posteriormente reescrito, utilizando a API do CCNx, caso o usuário deseje avaliar o comportamento dessa aplicação em um ambiente real. Isso fere o requisito de realismo funcional. Outra questão também reside na fidelidade dos modelos de *hardware*, pilha de protocolos e tráfego dos simuladores. Por maior que seja a evolução recente na modelagem de todos esses componentes, seu realismo pode ser questionado devido ao fato de que modelos sempre representam abstrações simplificadas dos componentes reais e podem não capturar todas as características ou sofrer as mesmas influências externas do ambiente real. Evoluções recentes do ns-3 (Alvarez, Orea, Cabrero, Pañeda, García & Melendi 2010) permitem a interação entre ferramenta e o tráfego real externo. Considerando essa abordagem (que possui desvantagens como a sincronização

de eventos do simulador com o tempo real e a adaptação de tráfego), porém, o ns-3 pode ser classificado mais como um emulador e, portanto, essa funcionalidade foi desconsiderada nessa seção.

Finalmente, uma desvantagem secundária do uso de simuladores está relacionada ao aprendizado e à facilidade de uso dos mesmos. Apesar de utilizarem linguagens de amplo uso na configuração de seus cenários (C++ no caso de ns-3 e OMNeT++), em geral é necessário despender um certo tempo para analisar e aprender sobre as estruturas, classes e abstrações fornecidas pelo simulador.

2.2.2 *Testbeds*

Testbeds são infraestruturas experimentais que fornecem recursos (servidores, enlaces, *switches* e roteadores) reais e/ou virtualizados para testes e avaliação de novos cenários. Em geral, diferem de testes de campo por permitirem a execução dos testes em ambientes isolados. Eles podem ser compartilhados entre vários pesquisadores, como no caso dos grandes *testbeds* internacionais (GENI (GENI 2013), PlanetLab (PlanetLab 2013) e Emulab (Emulab 2013)) ou criados especificamente para experimentos locais (como em (Ambiel, Rothenberg & Magalhães 2013)). No contexto de ROCs, pode-se citar o *testbed* do projeto NDN (NDN Testbed 2013) mas, em geral, os *testbeds* específicos para a área ainda não são públicos ou de grande abrangência.

Testbeds possuem a grande vantagem de apresentar realismo inerente pois executam código real em ambientes com pilhas de protocolos e tráfego real. Com isso, os resultados são de grande relevância prática. Porém, em geral, apresentam alto custo de criação e manutenção relacionados ao custo dos dispositivos e à administração da infraestrutura. Os grandes *testbeds* precisam até mesmo de administradores de sistema para essa atividade. Esse alto custo impacta a flexibilidade e a escalabilidade que os experimentos podem apresentar. A configuração de diferentes cenários pode ser onerosa podendo até chegar ao caso no qual cada *host* precisa ser reconfigurado individualmente entre diferentes testes. A flexibilidade com relação à definição de enlaces (conectividade, atraso e banda) também é, em geral, reduzida.

O projeto FITS (*Future Internet Testbed with Security*) (FITS 2013) apresenta um *testbed* baseado em virtualização e que visa endereçar a questão de reduzida flexibilidade e configurabilidade através de uma arquitetura orientada a serviços para monitoramento e configuração da infraestrutura. Porém, a escalabilidade e flexibilidade atingidas ainda não podem ser comparadas com as de um simulador.

2.2.3 Emuladores

Primeiramente, é necessário definir o termo **emulação**, já que o termo é muitas vezes usado de forma ambígua, tanto na academia quanto na indústria. Uma das primeiras definições na literatura científica data de 1969 é encontrada em (Lichstein 1969): "(emulação) é um processo através do qual um computador é configurado de forma a permitir a execução de programas escritos para outro computador". Em geral, essa definição se aplica bem para o caso do que hoje se chama emulação de *hardware* - a técnica de emular CPUs para permitir que aplicações compiladas em uma arquitetura possam ser utilizadas em outra. Por exemplo, um emulador de

hardware[4] pode permitir que um programa compilado para a plataforma ARM seja executado em um computador com a arquitetura x86. Um emulador de *hardware* é diferente de um virtualizador pois virtualizadores executam a maior parte das instruções diretamente na CPU real enquanto emuladores simulam todas as instruções em *software*. Dessa forma, virtualizadores não podem executar programas compilados para diferentes arquiteturas tal qual pode ser feito por emuladores.

De qualquer forma, essa definição não é a ideal para a emulação de redes, que é o foco desse texto. Assim, esse texto adota a seguinte definição (ligeiramente modificada de (Carson & Santay 2003)):

Emulação é um ambiente semi-sintético para a execução de código real. O ambiente combina a execução de implementações reais de rede (como a do kernel Linux) com a definição de falhas, configurações e atrasos de rede através de meios suplementares.

Portanto, um emulador de rede é um ambiente que executa aplicações reais, sobre pilhas de protocolos reais e com tráfego real. Vários elementos da rede (como *hosts* e roteadores) podem ser executados em uma única máquina e deve haver meios de emular a conexão entre eles, incluindo a configuração de parâmetros de *link* como atrasos, banda ou mesmo perdas. Nota-se que, diferentemente de um simulador que foca em implementar um modelo abstrato de um sistema, um emulador foca na reprodução exata do comportamento real da rede e das aplicações.

Pode-se considerar que os emuladores se situam numa posição intermediária entre os *testbeds* e os simuladores com relação aos requisitos e características analisados nessa seção. Os emuladores apresentam realismo pois, como os *testbeds*, executam código real em pilhas de protocolos reais e lidam com tráfego real. O mesmo código executado num emulador pode ser movido, posteriormente, para testes em um *testbed* sem alteração. O tráfego utilizado é o tráfego real, podendo ser analisado com *packet sniffers* (como o Wireshark (Wireshark 2013) ou `tcpdump` (tcpdump 2013)) e se comunicar com tráfego externo, gerado em outras máquinas. Emuladores podem não apresentar a mesma escalabilidade dos simuladores, mas podem ser igualmente flexíveis e também possuem baixo custo para a sua execução.

O CCN-JOKER(*Java Opensource Kit EmulatoR*) (Cianci, Grieco & Boggia 2012) é um dos primeiros emuladores desenvolvidos tendo o modelo CCN como base. Porém, ele não é genérico pois é focado especificamente na avaliação de cenários *ad hoc* sem-fio. Além disso, ele não utiliza a implementação oficial do modelo (CCNx) como base (uma implementação específica é feita nesse caso).

2.2.4 Análise Comparativa das Plataformas

A Tabela 2.1 sumariza as características das diversas plataformas tal qual analisadas até aqui nessa seção e adiciona mais uma característica: a possibilidade de configuração de parâmetros de *link* (como banda, atraso e taxa de perda de pacotes)[5].

[4] Como o conhecido QEMU (QEMU 2013) no modo usuário de emulação.

[5] Os modelos dos simuladores naturalmente incorporam a configuração de tais parâmetros. Emuladores podem utilizar ferramentas como `tc` (tc 2013) e `netem` (Linux Foundation 2013) para isso. *Testbeds* também podem

Tabela 2.1: Resumo das características das plataformas de teste para o modelo CCN.

	Simuladores	**Emuladores**	***Testbeds***
Ex:	*ndnSIM/ccnSim*	-	*Testbed NDN*
Flexibilidade	Alta	Alta	Baixa
Escalabilidade	Alta	Média	Baixa
Custo	Baixa	Baixa	Alta
Realismo	Baixo	Médio/Alto	Alto
Facilidade de Config.	Alta	Média	Baixa
Config. de *links*	Sim	Sim	Com restrições

Primeiramente, vale notar que é comum a utilização de duas ou mais plataformas para a avaliação de uma proposta. Por exemplo, num estágio inicial pode-se utilizar uma simulação para avaliar novo algoritmo de roteamento proposto para as ROCs. Em seguida, ele pode ser implementado e testado em um emulador. Por fim, o teste final pode ser feito em um *testbed*.

Com base na Tabela 2.1, pode-se notar que quando a escalabilidade é um requisito chave para avaliação de um cenário ou protocolo (por exemplo em um cenário com centenas de nós), a opção por um simulador parece ser a mais atraente num primeiro momento (no qual não se deseja ou não se tem recursos para criar uma infraestrutura real semelhante). Os *testbeds*, por sua vez, são interessantes quando se dispõe de recursos e se deseja o máximo de realismo na avaliação da proposta. Um possível exemplo pode ser uma situação na qual um novo protocolo em avaliação encontra-se muito próximo de ser implantado por soluções de mercado ou em casos onde não é possível simular ou emular todas as características relevantes para o cenário. Já os emuladores apresentam características intermediárias entre as dos simuladores e *testbeds*: apresentam baixo custo e realismo que podem ser alcançados sem sacrificar a flexibilidade ou a escalabilidade.

Quando a proposta analisada não exige uma grande escalabilidade, emuladores podem acelerar o processo de avaliação. Utilizando o exemplo anterior do desenvolvimento do protocolo de roteamento, o pesquisador pode partir diretamente para o desenvolvimento da proposta validando-a em um emulador e, posteriormente, em um ambiente real. Emuladores também podem reduzir o custo total de avaliação da proposta em comparação ao custo de *testbeds*. Como visto na tabela, o custo da avaliação em ambientes reais geralmente é alto principalmente em relação a configuração dos diversos dispositivos e posterior coleta e análise de resultados (além do custo dos dispositivos em si). Erros de implementação cujos resultados foram observados em *testbeds* poderiam ter sido previamente detectados em um emulador. O custos (computacional e temporal) em um emulador são muito menores do que os do ambiente real e, portanto, o pesquisador poderia se beneficiar do uso anterior de um emulador[6].

Especificamente para as ROCs baseadas no modelo CCN, não foi encontrado na literatura nenhum emulador genérico para avaliação de propostas na área. A existência dessa lacuna, sob

utilizar as mesmas ferramentas, porém, pode haver restrições quanto a disponibilidade de tais configurações, principalmente quando o *testbed* é compartilhado entre vários pesquisadores.

[6]Um exemplo específico de uma situação como essa será exemplificada no capítulo 4 desse texto.

o ponto de vista das características analisadas nessa seção, sugere fortemente que o desenvolvimento de um emulador especificamente focado no modelo CCN poderia trazer grandes benefícios para a pesquisa experimental na área visando validar as diversas propostas que visam endereçar os desafios desse novo paradigma para a Internet do futuro.

O próximo capítulo apresentará o Mini-CCNx, um novo emulador para o modelo CCN desenvolvido justamente para preencher a lacuna identificada entre as ferramentas atualmente disponíveis.

Capítulo 3

Arquitetura e Implementação

Este capítulo apresenta a arquitetura geral da ferramenta proposta, descrevendo a abordagem utilizada em seu desenvolvimento, as principais classes que a compõem, suas funcionalidades e, por fim, um exemplo típico do fluxo de trabalho de um experimentador utilizando a ferramenta na avaliação de um cenário de ROC. Ao longo desse capítulo, tomou-se o cuidado de justificar as várias opções de implementação e abordagens escolhidas no intuito de atender aos requisitos, objetivos e escopo delineados no primeiro capítulo.

3.1 Abordagem

Com base nos objetivos propostos e nas revisões técnica e bibliográfica realizadas optou-se por desenvolver um emulador focado especificamente no modelo CCN, denominado Mini-CCNx. O Mini-CCNx utiliza emulação baseada em contêineres (EBC) (LXC 2012), uma técnica de virtualização leve no nível de Sistema Operacional, e também técnicas de isolamento de recursos utilizando os *cgroups* do *kernel* Linux (cgroups 2012). O restante dessa seção descreve cada uma dessas técnicas e explica como elas auxiliam a ferramenta proposta a alcançar os objetivos para ela delineados.

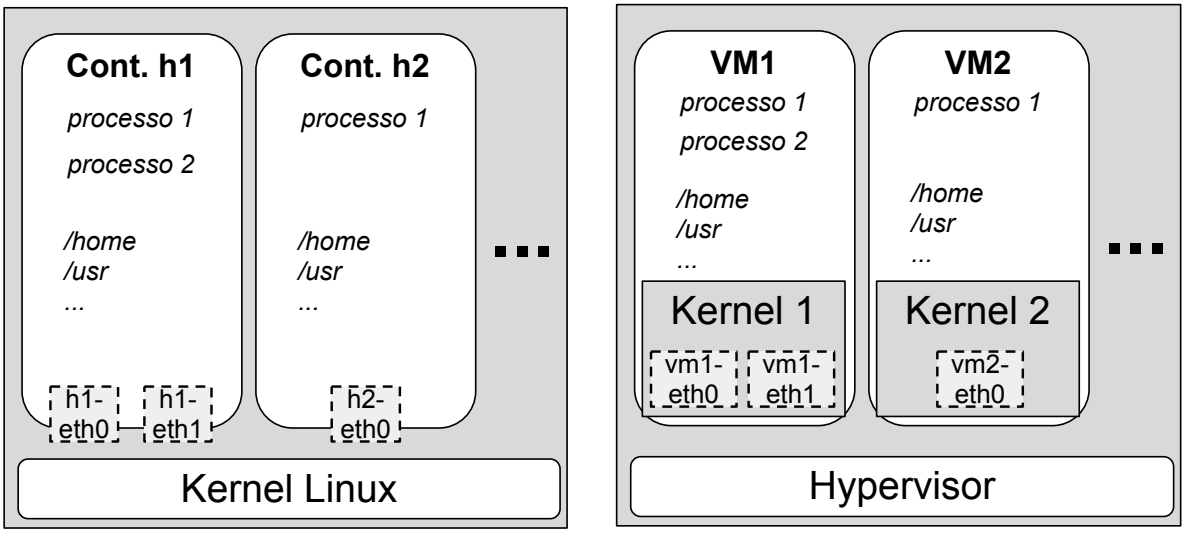
3.1.1 Emulação Baseada em Contêineres (EBC)

Primeiramente, é importante notar que a opção pelo desenvolvimento da plataforma de testes na forma de um emulador está fortemente associada ao atendimento dos requisitos propostos. Somente com um emulador seria possível a realização de testes com código real (realismo funcional e de tráfego), em um *hardware* de baixo custo, com flexibilidade e agilidade de criação de cenários e, ainda assim, mantendo níveis aceitáveis de escalabilidade.

O Mini-CCNx adota a mesma abordagem de virtualização leve utilizada em sistemas como vEmulab (Hibler, Ricci, Stoller, Duerig, Guruprasad, Stack, Webb & Lepreau 2008) e Mininet (Lantz, Heller & McKeown 2010) para emular *hosts* e *links* na rede. A EBC permite que diferentes grupos de processos, cada qual em seu contêiner Linux, tenham visões independentes (diferentes espaços de nome) de recursos do sistema como IDs de processos, nome de usuários, sistemas de arquivo e interfaces de rede mas, ainda assim, sejam executados no mesmo *kernel*.

A Figura 3.1(a) mostra esse conceito em um *host* Linux com dois contêineres (denominados **h1** e **h2**).

A EBC fornece melhor desempenho se comparada a sistemas de virtualização completa baseada em *hypervisors*, como VMware (VMware 2013a) ou Xen (Barham, Dragovic, Fraser, Hand, Harris, Ho, Neugebauer, Pratt & Warfield 2003), pois troca a habilidade de poder executar múltiplos *kernels* por um menor *overhead* de execução e, com isso, pode atingir uma maior escalabilidade (Soltesz, Pötzl, Fiuczynski, Bavier & Peterson 2007) em um mesmo *hardware* (ou seja, com um menor custo), o que é justamente um dos objetivos propostos para o Mini-CCNx. Além disso, a utilização da EBC também favorece a flexibilidade pois a criação de uma rede emulada utilizando essa técnica é feita de maneira extremamente ágil (uma análise detalhada será feita no capítulo 4). A Figura 3.1(b) ilustra a virtualização baseada em *hypervisor* com duas máquinas virtuais (**VM1** e **VM2**).



(a) Contêineres Linux

(b) Virtualização baseada em *hypervisor*

Figura 3.1: Comparação entre EBC e virtualização baseada em *hypervisor*

Porém, a maior escalabilidade atingida com a utilização da EBC tem seu preço: o isolamento de recursos pode ficar comprometido. Como todos os contêineres compartilham os mesmos recursos do *kernel* como escalonador de processos, escalonador de *links* e a mesma memória virtual, pode haver interferência entre os contêineres. Por exemplo, se uma aplicação em um contêiner começa a alocar memória indefinidamente, eventualmente faltará memória para as outras aplicações nos outros contêineres o que provavelmente afetará o comportamento e o desempenho das mesmas. Essa situação fere diretamente o requisito de realismo (funcional e temporal) estipulado para o Mini-CCNx pois essa interferência certamente prejudicará o comportamento esperado para a rede emulada. Um sistema de virtualização baseado em *hypervisor* provê um alto nível de isolamento pois cada máquina virtual possui seu próprio *kernel* que gerencia individualmente os recursos providos pelo *hypervisor*. Portanto, é necessário o uso de técnicas de isolamento de recursos conjuntamente ao modelo de EBC para garantir que essa interferência seja minimizada ao máximo.

3.1.2 Isolamento de Recursos

Os *Control Groups* (ou *cgroups*) (cgroups 2012), adicionados oficialmente ao *kernel* Linux em sua versão 2.6.24, permitem alocar recursos como tempo de CPU, memória de sistema ou banda de rede especificamente para um grupo de processos em execução no sistema. Com isso, é possível especificar, por exemplo, que um contêiner (grupo de processos) poderá usar, no máximo, 50% da CPU e 20% da memória disponível no sistema. Dessa forma, a aplicação que aloca memória indefinidamente no exemplo da subseção anterior, poderia fazê-lo até atingir o limite de 20%; as aplicações nos outros contêineres não seriam afetadas. A Figura 3.2 (reproduzida e adaptada de (Prpic et al. 2013)) mostra dois grupos de processos, **cg1** e **cg2**, que estão com os recursos CPU e memória limitados.

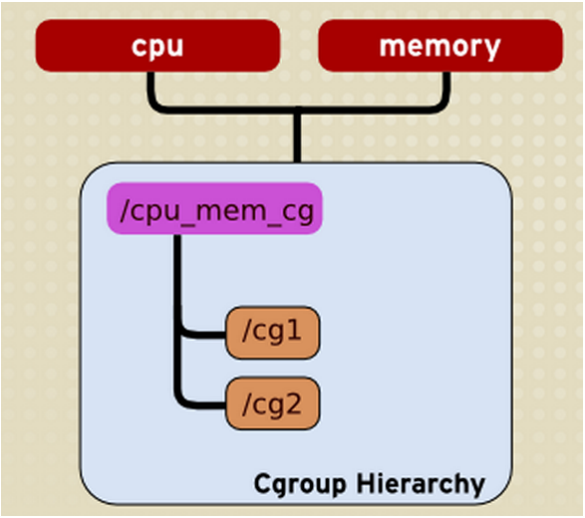


Figura 3.2: Conceito de *cgroups* - Alocação de CPU e memória (reproduzido de (Prpic et al. 2013))

Os *cgroups* permitem também a alocação de recursos como limitação de E/S para dispositivos de bloco (disco rígido, dispositivos USB), acesso limitado a dispositivos do sistema expostos no diretório /dev, dentre outros. Porém, para o desenvolvimento do Mini-CCNx, apenas a alocação dos recursos memória e CPU foi utilizada pois ambos se mostraram os mais necessários para a garantia de isolamento entre os contêineres.

Ainda no contexto de isolamento de recursos, mas agora focando nos recursos associados à rede, outra ferramenta utilizada foi o `tc` (tc 2013). Ela permite um controle detalhado da geração e recebimento de tráfego pela pilha TCP/IP do *kernel* Linux. Com `tc` é possível configurar as políticas de enfileiramento nos *buffers*, configurar filtros de pacotes e limitar a banda disponível para cada interface. O Mini-CCNx utilizou essa última funcionalidade. Para a emulação de *links*, adotou-se a ferramenta `netem` (Linux Foundation 2013). Utilizou-se especificamente a emulação de atrasos e taxas de perda de pacotes a serem utilizadas nos *links* entre os contêineres.

Utilizando as ferramentas `tc` e `netem`, buscou-se atender ao requisito de flexibilidade na definição de parâmetros de experimentos (no caso, banda, atraso e taxa de perda). A utilização dos *cgroups*, por sua vez, visa endereçar o desafio de isolamento de recursos trazidos pela EBC com o intuito de restaurar as características de realismo prejudicadas pela interferência existente

entre os contêineres.

3.2 Visão Geral

Introduzidos os conceitos de EBC e isolamento de recursos, será apresentada agora uma visão geral do Mini-CCNx.

No Mini-CCNx, cada contêiner é um nó CCN com o seu próprio espaço de nomes de rede privado. Isso significa que cada contêiner possui suas próprias interfaces de rede e todas as estruturas associadas, como *cache* ARP e tabela de roteamento. Além disso, cada nó também possui as estruturas específicas do modelo CCN, como CS, PIT e FIB implementadas pelo *daemon* `ccnd` e, opcionalmente, repositórios de dados implementados pelo *daemon* `ccnr`. Nota-se que, por ser um emulador, o Mini-CCNx pode se beneficiar do uso do código base oficial do modelo CCN que implementa esses *daemons*. Os nós são conectados entre si utilizando *links* Ethernet virtuais (`veth`) ponto-a-ponto no espaço de *kernel*. Utilizando os *cgroups*, pode-se especificar limites de CPU e memória para cada contêiner (nó) da topologia. A Figura 3.3 exemplifica esses conceitos com três nós (**n1**, **n2** e **n3**), cada qual com sua(s) interface(s) e estruturas CCN.

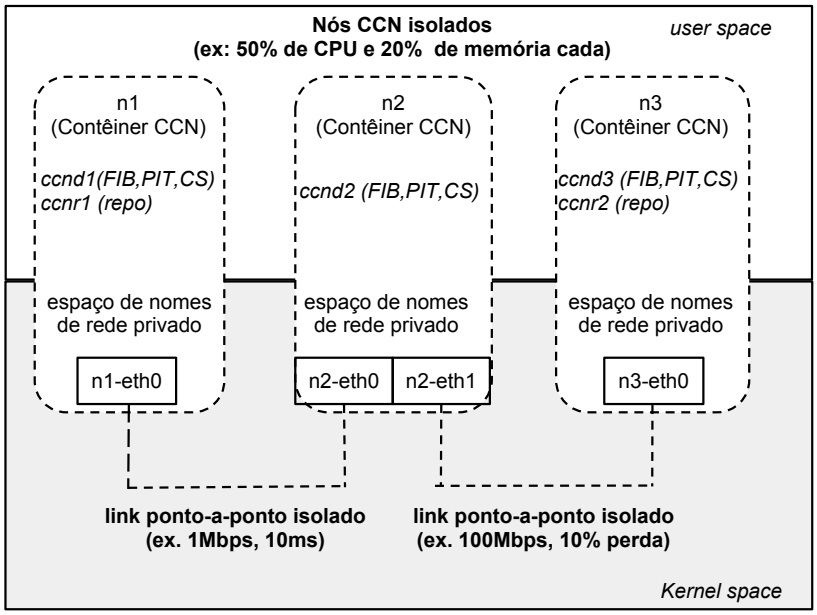


Figura 3.3: Três contêineres CCN ligados com *links* ponto-a-ponto

Dessa forma, os nós trocam tráfego CCN real, com o encaminhamento e *caching* sendo feitos pelo *daemon* `ccnd`, tendo como base o próprio ambiente Linux mas com a diferença de que cada nó tem o seu próprio espaço de rede privado e isolado. Isso permite que uma aplicação orientada a conteúdo desenvolvida e testada utilizando o Mini-CCNx possa ser executada posteriormente em um ambiente real (com diversos servidores Linux, *switches* e enlaces reais) sem alteração, atendendo aos requisitos de realismo funcional e de tráfego.

3.3 Implementação

O Mini-CCNx é um *fork* do Mininet-HiFi (Handigol et al. 2012). O Mininet é um sistema de EBC focado em Redes Definidas por Software baseadas em OpenFlow (McKeown, Anderson, Balakrishnan, Parulkar, Peterson, Rexford, Shenker & Turner 2008). É importante ressaltar que o Mini-CCNx não utiliza nenhuma funcionalidade relacionada ao OpenFlow, como por exemplo controladores ou *switches* programáveis. O intuito de estender o Mininet é aproveitar a infraestrutura já existente de criação dos contêineres. O Mini-CCNx traz uma série de extensões para permitir a emulação de ROCs baseadas no modelo CCN, incluindo a implementação de novas classes, novos mecanismos de construção de topologia, implementação de toda a interface com os *daemons* CCNx, um novo sistema de configuração e definição de cenários de ROCs, inserção das tabelas de encaminhamento de nós, dentre outros. A maior parte do código é feita em Python, com exceção de algumas tarefas que são feitas em C (criação e configuração de *cgroups*, término de processos, dentre outros). Além dessa parte principal, também foram implementadas ferramentas auxiliares para agilizar a criação dos cenários, como geradores automáticos de topologia e uma ferramenta gráfica (`miniccnxedit`) para geração de *templates* para o Mini-CCNx, tendo como intuito atender ao requisito proposto de facilidade de uso.

A Figura 3.4 mostra uma visão geral dos componentes do Mini-CCNx. A figura não pretende ser uma representação formal ou completa da implementação do sistema (como, por exemplo, um diagrama de componentes ou de classes UML). O intuito é apenas deixar claro quais partes foram introduzidas pelo Mini-CCNx (blocos com bordas em negrito), quais partes foram aproveitadas do Mininet mas com alterações significativas (blocos com bordas pontilhadas) e finalmente quais partes do Mininet foram aproveitadas apenas com alterações pontuais (bordas com linhas simples). Os *daemons* com os quais o Mini-CCNx faz interface (`ccnd`, `ospfn`, etc.) foram representados através de formas ovais. A seguir, a implementação dos diversos grupos de componentes será detalhada.

3.3.1 Nós Mini-CCNx

As principais funcionalidades do Mini-CCNx estão nas classes que implementam os nós CCN (**CCNHost** e **LimitedCCNHost**). A Figura 3.5 mostra um diagrama simplificado de tais classes, destacadas com bordas em negrito. As classes **Node**, **Host** e **CPULimitedHost** foram aproveitadas do código original, sendo necessário fazer apenas algumas alterações pontuais nas mesmas. É importante também ressaltar que o modelo CCN não estabelece uma diferenciação explícita entre *hosts* e roteadores - para todos os efeitos, tais componentes são chamados genericamente de nós CCN. Refletindo isso, todos os nós (contêineres) do Mini-CCNx serão uma instância da classe **CCNHost** ou de sua subclasse **LimitedCCNHost**.

Classe CCNHost

A classe **CCNHost** estende a classe base **Host**. Ela é responsável por dois grandes grupos de funções:

1. Implementação das estruturas do modelo CCN. Três novas estruturas e métodos associados

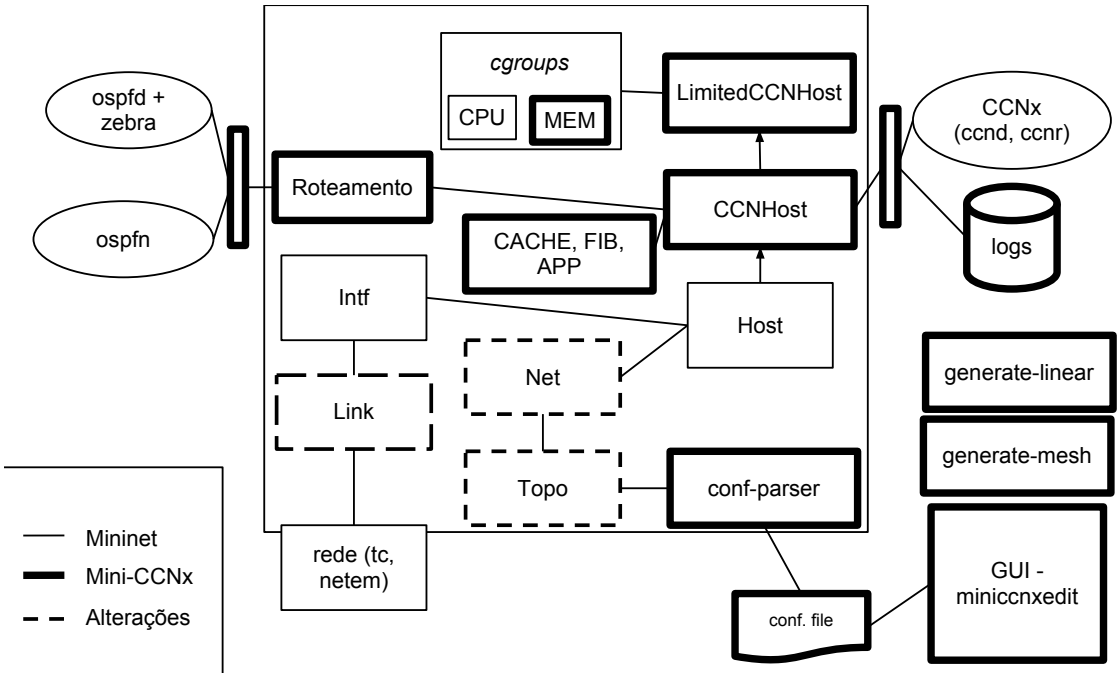


Figura 3.4: Blocos funcionais do Mini-CCNx

foram implementados: FIB, APP e CACHE. Cada nó mantém uma FIB (tabela de encaminhamento) que armazena os prefixos de nomes e os respectivos próximos *hops*. A FIB foi implementada como uma simples lista de tuplas na forma (*prefixo,próximoHop*), onde *prefixo* e *próximoHop* são *strings* Python. A estrutura APP representa a aplicação padrão (opcional) que será executada automaticamente no nó assim que o ambiente Mini-CCNx iniciar como, por exemplo, instanciar um repositório de conteúdos CCN no nó. APP precisa apontar para o caminho completo (*full path*) e válido de uma aplicação no sistema de arquivos pois o Mini-CCNx utilizará essa informação para executá-la através do módulo Python **subprocess**. Caso o usuário deseje executar não somente uma mas várias aplicações padrão no nó, o parâmetro APP pode apontar para um *shell script* contendo as chamadas para as diversas aplicações necessárias. Vale ressaltar que APP é um parâmetro opcional, ou seja, o usuário pode optar por não especificar uma aplicação padrão para o nó. É possível iniciar uma aplicação de maneira manual quando o ambiente Mini-CCNx estiver em execução, no momento que for conveniente para o usuário - o mecanismo de utilização da estrutura APP visa apenas trazer uma maior automação para a configuração e execução dos nós. Finalmente, cada nó contém sua estrutura CACHE, que representa a sua *Content Store* (CS). Informações como o tamanho da memória *cache* e o tempo de expiração dos conteúdos nela são armazenados e permitem que cada nó possua um perfil individual de *cache*. Isso possibilita a realização de experimentos nos quais os nós possuam, por exemplo, *caches* de diferentes tamanhos como em um caso com sensores (pequena memória) ligados a um servidor centralizado (memória maior).

2. Interface com os daemons CCNx. Como explicado anteriormente, em tempo de execução, o encaminhamento, armazenamento, assinatura de conteúdos e outras tarefas são realizados pelos *daemons* do CCNx. O Mini-CCNx é responsável por criar e configurar todo o ambiente emulado

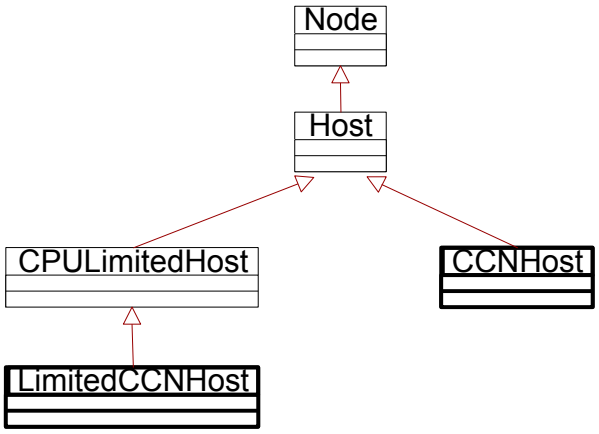


Figura 3.5: Classes de nós CCN no Mini-CCNx (destacadas com bordas em negrito)

e isolado dentro do qual tais *daemons* serão executados. Portanto, é necessário implementar toda a interface entre o emulador e o código do CCNx. Primeiramente, o Mini-CCNx é responsável por iniciar o *daemon* `ccnd` em cada nó através do utilitário `ccndstart`. Configura-se também o seu *socket* (parâmetro CCN_LOCAL_SOCKNAME) e sua porta padrão (CCN_LOCAL_PORT). A partir das informações contidas na estrutura CACHE, as variáveis de ambiente (utilizadas pelo `ccnd`) CCND_CAP (capacidade de *cache*) e CCND_DEFAULT_TIME_TO_STALE (tempo de expiração dos conteúdos armazenados) são configuradas. Os métodos `insert_fib` e `remove_fib` são responsáveis por fazer a inserção e a remoção de entradas na estrutura FIB do nó. O estado de tal estrutura é passado dinamicamente para o `ccnd` através do utilitário `ccndc`, utilizando também informações de conectividade fornecidas pela classe Net (explicada em detalhes na próxima seção). A terminação do `ccnd` também é realizada pela classe CCNHost, através do comando `kill` (sinal SIGINT seguido de SIGKILL para garantia de término dos *daemons*). Outra função importante realizada por essa classe é o direcionamento dos *logs* gerados pelas instâncias do `ccnd` de cada nó para um diretório em comum (subdiretório `log/` no diretório atual de execução do Mini-CCNx) visando centralizar uma posterior análise e *parsing* das informações geradas por todos os nós durante o experimento.

Classe LimitedCCNHost

A classe **LimitedCCNHost** estende as classes **CPULimitedHost** e **CCNHost**. Essa herança múltipla significa que tal classe é responsável por fazer todas as funções descritas na subseção anterior somadas às funções de criação, configuração e anexação de *cgroups* reservando o recurso CPU para o nó (por estender a classe **CPULimitedHost**) além de fazer uma nova configuração referente a reserva do recurso memória, trazida pelo Mini-CCNx. Assim, um **LimitedCCNHost** nada mais é que um **CCNHost** com reserva de recursos.

A alocação do recursos é descrita na sequência. Cada nó será um *cgroup* Linux cujo nome será o mesmo nome do nó. A primeira etapa é montar a partição de *cgroups* no diretório "/sys" com a opção "-t cgroups" do comando `mount` do Linux e especificar os tipos de recursos a serem alocados (*cpu*, *cpuacct*, *cpuset* e *memory*, sendo o último uma adição do Mini-CCNx ao Mininet,

tal qual explicado anteriormente). Essa montagem é feita somente uma vez, pelo primeiro nó da topologia. Essa operação resulta na criação de subdiretórios em "/sys/fs/cgroup/" com os nomes dos recursos citados anteriormente. Em seguida, o nó cria de fato o *cgroup* com o comando `cgcreate`. Por exemplo, caso o nó tenha o nome **h1** e foi configurado para possuir limitação de memória, a operação de criação cria o diretório "/sys/fs/cgroup/memory/h1", no qual haverá uma série de arquivos nos quais é possível configurar diversos aspectos de utilização da memória. O único parâmetro utilizado pelo Mini-CCNx é o *memory.limit_in_bytes* que limita o uso total de memória por um nó. Essa configuração é feita através do comando `cgset`. Finalmente, ao fim da execução do emulador, cada nó também é responsável por apagar o *cgroup* por ele criado com o comando `cgdelete`. Mais detalhes e opções referentes à configuração de *cgroups* podem ser encontrados em (cgroups 2012).

3.3.2 Instanciação e Execução Cenários

As funções de criar a topologia com tais nós e colocar o ambiente de fato em execução são responsabilidade das classes no diagrama simplificado da Figura 3.6. Nessa figura foi mantida a legenda utilizada na Figura 3.4. Portanto, classes com bordas pontilhadas (**Net**, **Link** e **Topo**) foram aproveitadas do Mininet mas sofreram alterações significativas e classes com bordas simples (**Intf** e **Node**) também foram aproveitadas mas sofreram somente alterações pontuais.

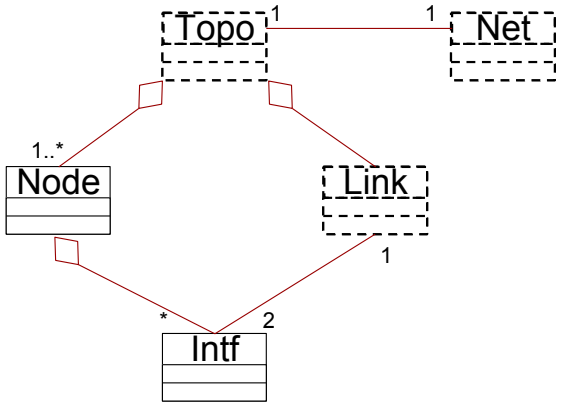


Figura 3.6: Classes para a instanciação e execução de cenários

Um nó pode ter diversas interfaces (classe **Intf**). Por padronização, o nome Linux para tais interfaces será sempre *nomedohost-ethX*. Por exemplo, o *host* **h1** terá interfaces de nome *h1-eth0*, *h1-eth1* e assim por diante. Esses são exatamente os nomes de interface observados na saída do comando `ifconfig` executado individualmente em cada nó. Um **Link** é a associação de duas interfaces e onde as ferramentas `tc` e `netem` configuram a banda nominal disponível, o atraso de propagação e a taxa de perda de pacotes (se especificados). A classe **Topo** é responsável por guardar toda a topologia do cenário de testes na forma de um grafo com os nós sendo os vértices e os *links* as arestas. A classe **Net** é responsável por instanciar todos os nós e *links* descritos por **Topo** e dar início de fato a execução do ambiente de testes especificado.

Como explicado no capítulo 2, a implementação atual do CCNx opera sobre conexões TCP ou UDP. A implementação original do Mininet exige que o usuário especifique o(s) endereço(s)

IP de cada nó da topologia, insira as rotas desejadas e detalhe conectividade com *software switches*. Em contrapartida, o intuito do Mini-CCNx é remover qualquer referência ao IP por parte do usuário - a idéia é que ele precise se preocupar somente com prefixos e os respectivos próximos *hops*. Por esse motivo, o Mini-CCNx introduz uma mudança substancial na maneira como é criada a conetividade entre nós e altera o comportamento principal das classes **Topo**, **Link** e **Net**. O Mini-CCNx cria, de maneira totalmente transparente ao usuário, *links* ponto-a-ponto entre os nós da topologia. Para cada *link*, configura-se uma subrede /30, iniciando com o endereço de rede 1.0.0.0/30 e incrementando esse valor conforme haja mais enlaces na topologia. Isso implica em um limite de implementação de no máximo 1061208000 (255*255*255*64) enlaces para os cenários - um valor que certamente é muito elevado. Será mostrado no próximo capítulo que a quantidade de recursos computacionais (memória e CPU) utilizados pelos nós e enlaces do Mini-CCNx em um *hardware* típico se esgotam muito antes de se atingir tal quantidade de *links* e, portanto, isso não será um limite prático para a execução do emulador. Finalmente, cria-se a conectividade TCP entre os *daemons* `ccnd` dos nós em cada ponta do enlace.

3.3.3 Configuração de Cenários e Topologias

Visando atingir uma maior facilidade de uso, a classe **ConfParser** (Figura 3.7) é responsável por ler o arquivo de configuração do Mini-CCNx, fazer o *parse* das informações descritas (nós, suas configurações e os *links* entre eles) e criar uma representação correspondente de topologia para ser armazenada por **Topo**.

Figura 3.7: Classes para a configuração de cenários

O arquivo de configuração do Mini-CCNx é um simples arquivo texto composto de duas seções: `nodes` e `links`. Ambas são descritas a seguir:

Seção `nodes`: Cada linha de tal seção representa um nó CCN da topologia. Para cada nó, pode-se configurar seu nome, sua aplicação padrão (caminho completo para mesma), o tamanho máximo de seu *cache* (em KBytes), sua reserva de uso de CPU (em porcentagem), sua reserva do uso de memória (em porcentagem) e entradas padrão na FIB inseridas na forma de tuplas (*prefixo,próximoHop*). É interessante notar que com essa última configuração é possível popular previamente a tabela de encaminhamento de cada nó da topologia sem a necessidade específica de ativar algum algoritmo de roteamento durante o experimento. Isso pode ser útil no caso de cenários nos quais o foco não está especificamente no roteamento, como por exemplo em uma análise do comportamento do plano de encaminhamento ou para testes simples de aplicações sendo desenvolvidas.

Seção `links`: Cada linha dessa seção representa um *link* entre nós descritos previamente na seção anterior. É possível configurar o atraso de propagação (de 0 a 1000ms), a banda nominal

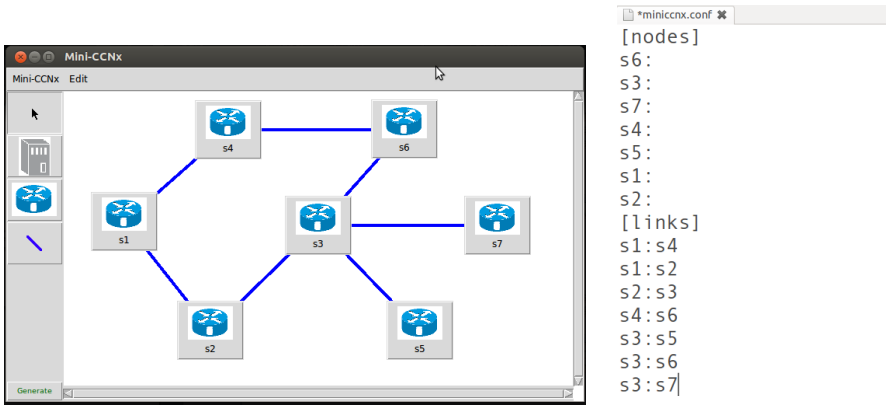
disponível (de 1 a 1000 Mbps) e a taxa de perda de pacotes (em porcentagem).

Nota-se que, diferentemente da implementação original do Mininet na qual é necessária codificar em Python o cenário OpenFlow desejado, os cenários do Mini-CCNx são descritos em um simples arquivo texto. Isso pode facilitar o uso da ferramenta por pesquisadores e estudantes que não sejam propriamente proficientes em tal linguagem.

Ferramentas para a criação de topologias

Ainda com o intuito de aumentar a facilidade de uso, algumas aplicações auxiliares foram desenvolvidas para gerar *templates* do arquivo de configuração do Mini-CCNx de forma ainda mais ágil.

O `miniccnxedit` (Figura 3.8) é uma aplicação gráfica baseada no *framework* TkInter (TkInter 2013) com a qual é possível gerar topologias Mini-CCNx. Seu uso também pode ser interessante quando é necessário se ter uma noção visual de um cenário sendo testado. Com o *template* gerado, o usuário pode completar o arquivo de configuração com as configurações desejadas.



(a) Aplicação `miniccnxedit` sendo executada (b) *Template* gerado pela aplicação

Figura 3.8: Geração de *templates* de arquivo de configuração com o `minccnxedit`

Além do `miniccnxedit`, dois geradores de topologias padrão foram desenvolvidos. O `generate-linear` gera um *template* com n nós ligados linearmente entre si e com entradas na tabela de encaminhamento apontando sempre para o próximo *hop*. Tal configuração de topologia pode ser interessante em experimentos nos quais deseja-se analisar a influência do número de *hops* no cenário sendo testado. O `generate-fullmesh` gera uma topologia com n nós nas quais todos os nós possuem *links* entre si. Tal configuração pode ser interessante para testar cenários com alto nível de conectividade. Ambos geradores foram implementados na forma de *scripts* Python.

3.3.4 Roteamento CCN

Como citado na subseção anterior, com o Mini-CCNx é possível criar cenários nos quais o usuário especifica previamente todas as entradas das FIBs de todos os nós antes de executar o emulador. Isso lembra uma abordagem de configuração de roteadores e *hosts* IP na qual todas

as rotas são inseridas de maneira estática pelo administrador da rede. Porém, o Mini-CCNx implementa também um novo módulo de roteamento, com foco no modelo CCN. Essa pode ser considerada uma contribuição já que o Mininet não inclui qualquer implementação referente a protocolos de roteamento (como RIP, OSPF ou BGP) nos nós IP originais.

O Mini-CCNx inicia o roteamento baseado em nomes denominado OSPFN (*OSPF for Named-data*) (OSPFN 2012) em toda a topologia especificada. O OSPFN é baseado no OSPF (Open Shortest Path First) (Moy 1998) e utiliza OLSAs (*Opaque Link State Advertisements*) (Berger & Bryskin 2008) para anunciar prefixos de nomes de conteúdo. O protocolo de roteamento proposto é dinâmico e multicaminhos. Em cada nó da topologia, os *daemons* necessários ao roteamento são iniciados: `ospfd` e `zebra` (vindos da implementação de código aberto Quagga (Quagga 2013)) e o próprio OSPFN. Tais *daemons* periodicamente trocam OLSAs em busca dos melhores caminhos para cada prefixo anunciado alimentando dinamicamente as tabelas de encaminhamento implementadas pelo *daemon* `ccnd`.

Interface com os *daemons* de roteamento

O Mini-CCNx infere que o usuário deseja ativar o roteamento em sua topologia caso detecte, no diretório de execução atual, a existência de subdiretórios cujos nomes sejam os nomes dos nós CCN do cenário, tal qual especificados no arquivo de configuração do Mini-CCNx (seção anterior). Esses subdiretórios serão utilizados para armazenar três arquivos de configuração referentes ao roteamento: dois relativos aos *daemons* `ospfd` e `zebra` (Quagga) e um correspondente ao `ospfn` (para o protocolo de roteamento orientado a conteúdo OSPFN). Os arquivos de configuração para o Quagga são gerados dinamicamente pelo Mini-CCNx. Já o arquivo de configuração do OSPFN deverá ser criado pelo usuário, no qual ele deverá informar os prefixos de conteúdos que serão anunciados pelo nó.

O arquivo *zebra.conf* contém basicamente o *hostname* do nó (no contexto de DNS) e uma senha de acesso à interface de configuração do roteamento Quagga, porém, ambas as funcionalidades não serão utilizadas de forma alguma - tal arquivo é criado apenas por ser um requisito à execução dos *daemons* `zebra` e `ospfd`. O arquivo *ospfd.conf*, por sua vez, contém uma lista com os nomes das interfaces de rede do nó e uma lista das subredes IP (em notação CIDR) que serão anunciadas pelo protocolo OSPF no nó. Uma descrição detalhada sobre a sintaxe de tais arquivos e outras opções pode ser encontrada em (Quagga 2013) - vale ressaltar que o Mini-CCNx utilizou apenas as configurações básicas citadas anteriormente, essenciais à execução do OSPFN. Finalmente, o arquivo de configuração *ospfn.conf*, a ser criado pelo usuário, pode conter dois tipos de diretivas. Linhas iniciadas com a diretiva *ccnname* devem conter o prefixo a ser anunciado pelo nó, um prefixo por linha. Já uma linha iniciada com a diretiva *logdir* deve conter o diretório no qual o *log* do *daemon* `ospfn` será armazenado (recomenda-se utilizar "."), ou seja, armazenar o *log* no diretório atual do nó, juntamente com os *logs* do Quagga).

Ao detectar que a topologia usará o roteamento OSPFN, o Mini-CCNx automaticamente cria os arquivos *zebra.conf* e *ospfd.conf*, utilizando informações vindas da classe *Net* sobre a conectividade base IP do cenário. Em seguida, os dois *daemons* Quagga são iniciados e configurados (basicamente o caminho do sistema de arquivos para a trava de PID e execução no modo *background*) por um simples *shell script* denominado `routing.sh`. Em seguida, o Mini-CCNx

inicia o *daemon* `osfpn` utilizando o arquivo previamente criado pelo usuário. A partir desse momento, o ambiente já está pronto para utilização, na qual será possível observar a operação do roteamento, alterar *links* dinamicamente e observar como se dá a eventual convergência do algoritmo, etc.

Por fim, ao término da execução do emulador, os nós CCN são responsáveis por fazer a terminação de todos os *daemons* iniciados utilizando o comando `kill`. Note que, também ao fim da execução, os subdiretórios conterão os diversos *logs* gerados durante o experimento para uma posterior análise de sua execução.

Emulação um *Testbed* Real

O *testbed* oficial do Projeto NDN (NDN Testbed 2013) conta com 16 nós CCN espalhados por toda a extensão Estados Unidos e mais um nó em Pequim, na China[1]. Ele é utilizado pelos membros do projeto para fazer testes de maior escala e também na validação de técnicas e protocolos de roteamento para o modelo CCN. O Mini-CCNx conta com uma funcionalidade pré-configurada na qual é possível fazer a emulação de todo esse *testbed* através de um simples comando "`miniccnx -testbed`". Os atrasos para os *links* foram calculados com base na distância geográfica entre os institutos/Universidades que hospedam tais nós[2] utilizando a ferramenta Google Earth (Google 2013).

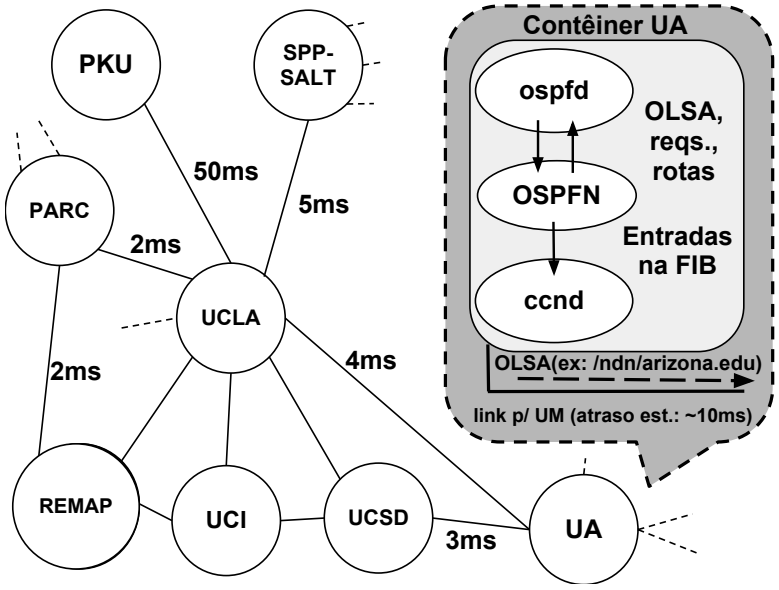


Figura 3.9: Parte do *testbed* NDN com destaque para o roteamento OSPFN

Além de instanciar os nós e trazer a estimativa de parâmetros dos *links*, com essa opção o Mini-CCNx também inicia o roteamento baseado em nomes OSPFN para essa topologia. Os prefixos de nomes anunciados são exatamente os mesmos divulgados no *testbed* oficial. A Figura 3.9 mostra uma parte do *testbed* com os atrasos de propagação estimados nos *links*, alguns

[1]Tal qual visto em Março/2013.

[2]Atrasos de processamento e fila são gerados naturalmente pelo emulador. Atrasos de transmissão podem ser desprezados ante o valor dos atrasos de propagação.

nós[3] e com um destaque sobre o nó **UA** (Universidade do Arizona), mostrando os *daemons* utilizados e o anúncio do prefixo `/ndn/arizona.edu` para um de seus vizinhos.

O objetivo de trazer essa funcionalidade é justamente trazer uma experiência prática com roteamento em ROCs baseado em uma topologia real, analisando o comportamento dinâmico e o seu tempo de convergência. Tem-se, também, o intuito de mostrar que o Mini-CCNx pode de fato auxiliar no desenvolvimento e validação de novos protocolos de roteamento para ROCs.

3.4 Fluxo de Trabalho Utilizando o Mini-CCNx

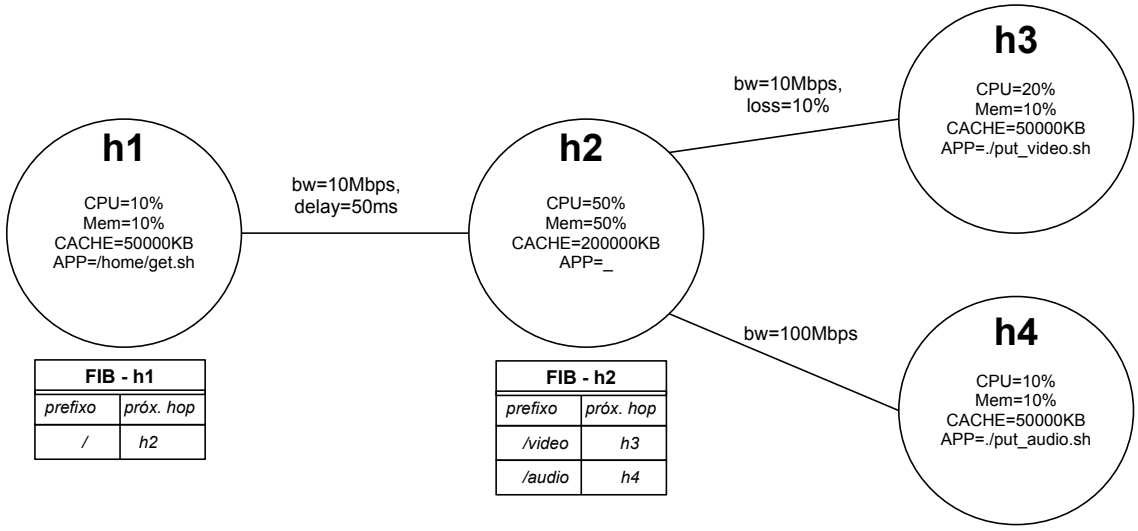


Figura 3.10: Exemplo de cenário a ser emulado no Mini-CCNx

De forma a sintetizar os conceitos apresentados nas seções anteriores, será apresentado agora um exemplo completo de como o usuário pode utilizar o Mini-CCNx para emular um cenário específico de uma ROC. A Figura 3.10 mostra os quatro nós da topologia e as configurações associadas. O nó **h1** faz o papel de um cliente que possui reserva de 10% de CPU e de memória na máquina na qual executa. Ele possui 50000KB de *cache* e executa uma aplicação padrão denominada `/home/get.sh`. Sua tabela de encaminhamento (FIB) é apresentada abaixo do nó e indica que pacotes *Interest* para qualquer prefixo de nome (representado por "/") deverá ser encaminhado para o nó **h2**. O nó **h2**, por sua vez, faz o papel de um roteador central e possui uma reserva maior de recursos (50% de CPU e memória e *cache* de 200000KB). Ele não executa nenhuma aplicação padrão, o que é representado por "_" no parâmetro APP. Sua tabela de encaminhamento indica que pacotes *Interest* buscando conteúdos cujos nomes iniciam com o prefixo "`/video`" devem ser encaminhados para **h3**; prefixos iniciados por "`/audio`", por sua vez, devem ser encaminhados para **h4**. Os nós **h3** e **h4** representam servidores de conteúdo (vídeo e áudio, respectivamente). A explicação da configuração de cada nó é análoga à feita anteriormente com a exceção de que tais nós possuem tabelas de encaminhamento vazias. Agora com relação à ligação entre os nós, nota-se que o *link* entre **h1** e **h2** apresenta um atraso grande

[3]Os nomes dos nós são as siglas dos nomes dos institutos/Universidades que os hospedam.

(50ms) representando um atraso típico em uma longa distância na Internet. O *link* entre **h2** e **h3** possui uma taxa de 10% de perda de pacotes emulando uma ligação sem-fio. Por fim, o **link** entre **h2** e **h4** possui banda de 100Mbps com atraso e perda desprezíveis podendo representar assim o *link* de uma rede local.

Tendo feito previamente essa definição de cenário, o usuário precisa agora basicamente escrever o arquivo de configuração do Mini-CCNx com a seção `nodes` contendo quatro linhas cada qual com a configuração do respectivo nó (de **h1** a **h4**) e com a seção `links` definindo cada *link* e suas configurações. O usuário pode, opcionalmente, utilizar a ferramenta gráfica `miniccnxedit` para gerar um *template* de tal arquivo de configuração. O arquivo resultante é mostrado na Figura 3.11.

example-conf

```
[nodes]
h1: /home/get.sh cache=50000 CPU=0.1 MEM=0.1 /,h2
h2: _ cache=200000 CPU=0.5 MEM=0.5 /video,h3 /audio,h4
h3: ./put_video.sh cache=50000 CPU=0.2 MEM=0.1
h4: ./put_audio.sh cache=50000 CPU=0.1 MEM=0.1
[links]
h1:h2 bw=10 delay=50ms
h2:h3 bw=10 loss=10
h2:h4 bw=100
```

Figura 3.11: Arquivo de configuração para o cenário descrito na Figura 3.10

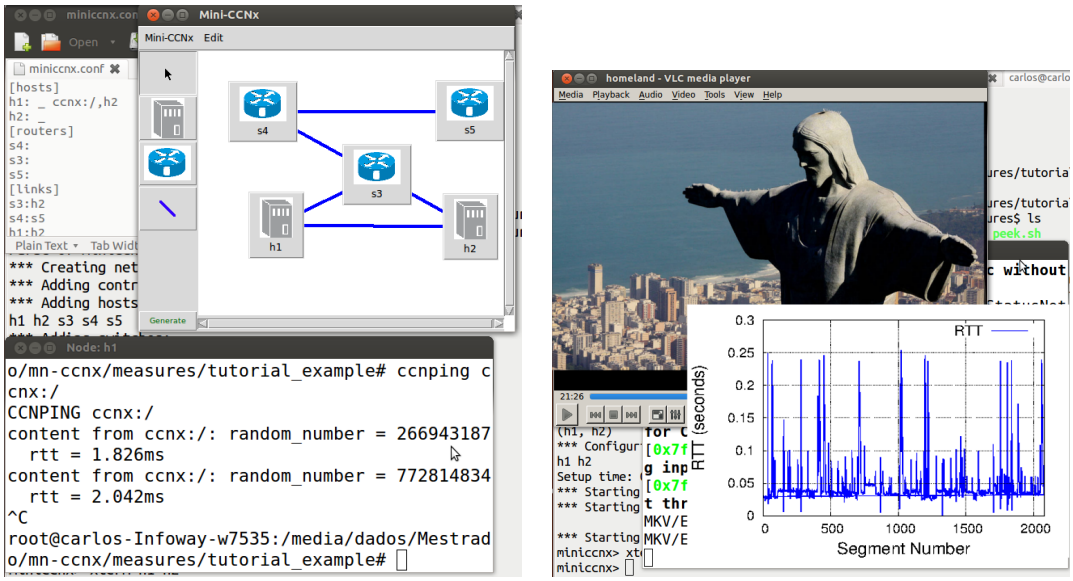


Figura 3.12: Captura de tela da execução do Mini-CCNx

Supondo que o usuário salvou o arquivo de configuração com o nome `example-conf`, o Mini-CCNx pode então ser iniciado com o comando "`miniccnx -f example-conf`". A partir desse momento, o *parse* do arquivo de configuração é feito, os contêineres são criados para cada nó, os *cgroups* são criados e configurados, os *links* entre os nós são criados e os *daemons* e estruturas

CCNx são iniciados. Ao fim desse processo, o Mini-CCNx entrega ao usuário um *prompt* de comando específico. A partir daí é possível inspecionar a topologia e as conexões, abrir terminais isolados (`xterm`) para cada nó, iniciar novas aplicações, inspecionar arquivos e *logs*, analisar a tabela de encaminhamento dos *hosts*, inspecionar os pacotes trocados, dentre outras atividades. O usuário pode interagir com cada nó através do respectivo terminal da mesma com a qual ele interage com máquinas Linux reais - a diferença é que cada contêiner está num espaço de rede privado e com recursos reservados. Por fim, através do comando `quit` no *prompt* do Mini-CCNx, o usuário termina a execução da ferramenta provocando a terminação dos processos e *cgroups*. A Figura 3.12 mostra uma captura de tela do Mini-CCNx em execução.

Num geral, pode-se esquematizar o fluxo de trabalho do uso da ferramenta, tal qual feito no exemplo anterior, através das seguintes etapas (mostradas também no fluxograma da Figura 3.13):

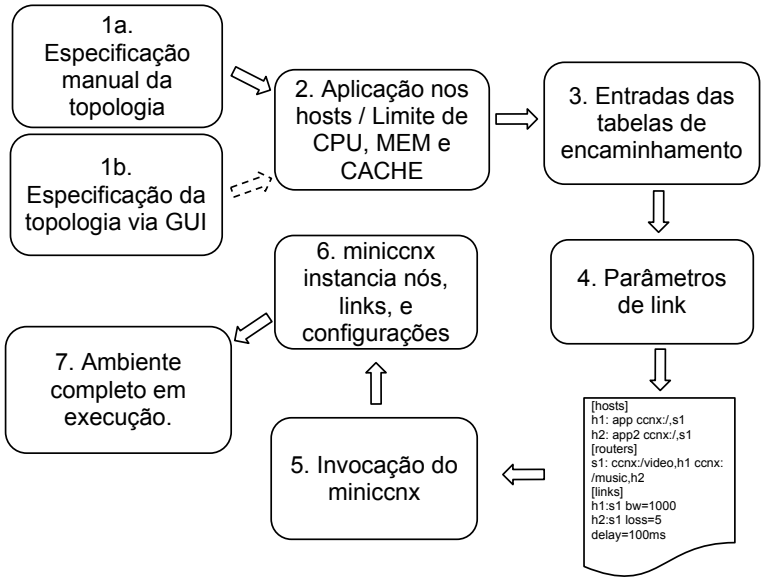


Figura 3.13: Fluxo de Trabalho do Mini-CCNx

1. A especificação da topologia é feita editando o arquivo de configuração do Mini-CCNx especificando os nós e os *links* entre eles (etapa *1a* na Figura 3.13). Opcionalmente, pode-se utilizar a ferramenta gráfica `miniccnxedit` fornecida para gerar um template desse arquivo (etapa *1b* na Figura 3.13).
2. No próprio arquivo de configuração, é possível especificar qual a aplicação será executada automaticamente em cada *host*. Opcionalmente, pode-se definir o limite de uso de CPU e Memória (MEM) em porcentagem, para cada nó. Também é possível especificar o tamanho do *cache* em KB.
3. Ainda no arquivo de configuração, pode-se inserir entradas nas tabelas de encaminhamento dos nós da topologia.
4. Pode-se especificar ainda parâmetros para cada *link* (banda, perda ou atraso). Ao fim dessa etapa, o arquivo de configuração estará concluído.

5. A ferramenta `miniccnx` deve ser invocada, tendo como argumento o arquivo de configuração criado.
6. A ferramenta faz o *parsing* das configurações, instancia os nós, configura as entradas nas tabelas de encaminhamento, aplica os parâmetros especificados para cada *link* e executa as aplicações nos *hosts*.
7. Agora o ambiente está criado e executando, permitindo ao usuário a interação dinâmica com as aplicações baseadas em conteúdo. Opcionalmente, ele pode coletar quaisquer métricas e *logs* que sejam necessários para a avaliação de seus cenários.

3.5 Divulgação e Documentação

Como descrito no primeiro capítulo, o Mini-CCNx pretende, desde o início de seu desenvolvimento, ser uma plataforma de código livre para que possa ser estendida e melhorada ao longo do tempo por diversos grupos interessados em seu uso. Para isso, optou-se pela adoção da plataforma GitHub (GitHub 2013) para versionamento de código *online*. Além disso, o GitHub também se mostrou muito útil para a construção de uma *wiki* na qual foram inseridas a documentação do código, guia de instalação, tutoriais e uma máquina virtual totalmente pré-configurada pronta para uso (focada nas plataformas de virtualização VirtualBox (Oracle 2013) e VMware Workstation (VMware 2013b)). A Figura 3.14 mostra uma captura de tela de uma das páginas da documentação.

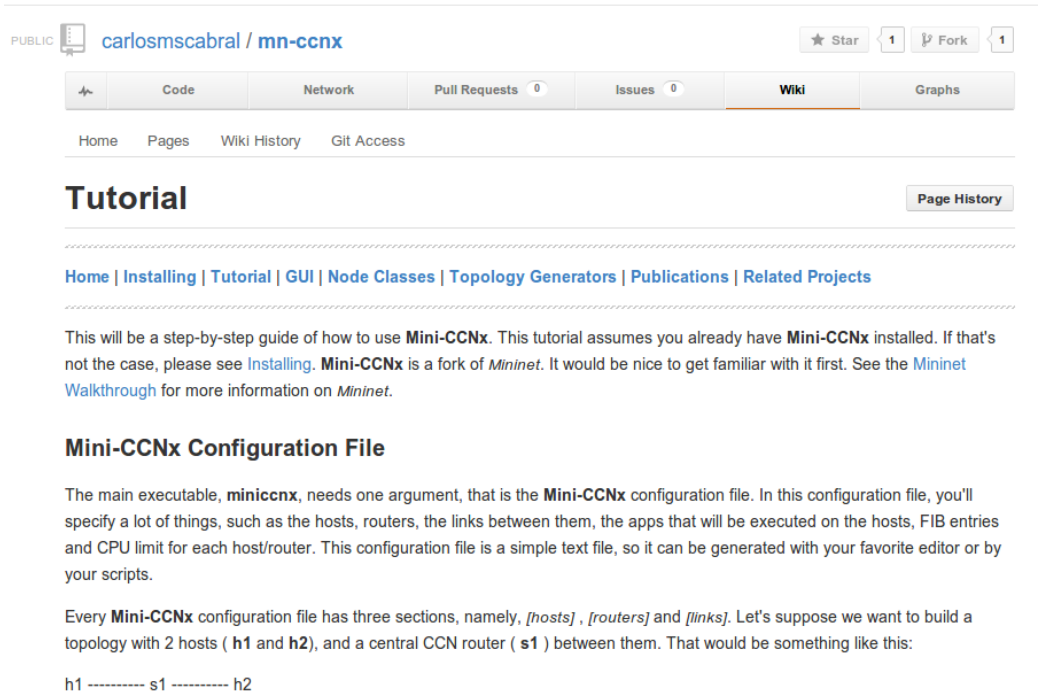


Figura 3.14: Captura de tela da *wiki* do Mini-CCNx

Nesse capítulo, foram apresentadas a arquitetura e a implementação do Mini-CCNx. O próximo capítulo trará uma série de experimentos utilizando o emulador a fim de avaliar diversos aspectos de seu funcionamento e de suas aplicações.

Capítulo 4

Avaliação e Resultados

Esse capítulo apresentará a metodologia de avaliação da ferramenta Mini-CCNx e diversos experimentos realizados no contexto desse projeto de pesquisa.

4.1 Metodologia

A metodologia de validação consiste em realizar diversos experimentos utilizando o Mini-CCNx e avaliá-lo quanto a: *(i)* escalabilidade, *(ii)* coerência, *(iii)* capacidade de isolamento e *(iv)* fidelidade ante a experimentos reais.

Os experimentos devem abranger diversos aspectos das ROCs baseadas no modelo CCN como: *caching*, roteamento, distribuição de conteúdo, comportamento das aplicações orientadas a conteúdo ante a variação de parâmetros de *link* (banda, atraso e perda), execução de aplicações multimídia (como vídeo e áudio) e plano de encaminhamento adaptativo reforçando assim o objetivo da ferramenta ser a mais genérica possível com relação aos experimentos.

4.1.1 Recursos

A maior parte dos experimentos serão realizados utilizando um *hardware* convencional, representante típico de uma máquina de uso pessoal (processador Intel Core I5 2410M e com 4GB de memória RAM) com o intuito de reforçar a possibilidade de realização dos mais diversos experimentos em um equipamento de baixo custo. Eventualmente, *desktops* do laboratório (com processador Intel I3 e também 4GB de memória) serão utilizados para comparar os resultados do emulador ante ao comportamento de ambientes reais. A versão do *software* CCNx utilizada é a `ccnx-0.7` em todos os experimentos. A versão do protocolo de roteamento OSPFN utilizada é a 2.0.

4.1.2 Medidas

Pelo fato de ser um emulador, o Mini-CCNx permite utilizar conhecidas ferramentas de medição para ambientes de rede como `bwm-ng` (medição de banda), `ping` (atraso e perda) e `tcpdump`. Além disso, também é possível utilizar diversas ferramentas focadas especificamente para o modelo CCN como `ccnping`, `ccngetfile`/`ccnputfile` (envio e requisição de arquivos),

`ndndump` (inspeção de pacotes), dentre outras. A inspeção dos *logs* do *daemon* `ccnd` também foi utilizada para a análise de parâmetros como atraso e para auxiliar na situação de *debug* pois esse *log* contém, dentre outras coisas, o registro de cada pacote que foi processado pelo nó (incluindo o tempo, interface de entrada e interface de saída). A ferramenta `ccndstatus`, por sua vez, foi utilizada para a inspeção (de forma dinâmica) das FIBs dos nós, em experimentos envolvendo encaminhamento e roteamento.

Na maioria dos casos, os resultados foram obtidos de *logs* e *traces* gerados durante o experimento, seguidos do cálculo da média e do respectivo desvio padrão. O Mini-CCNx mostrou ser flexível, o que ajudou na repetição de diversos experimentos em um tempo relativamente curto. Aproveitando-se desse fato, onde aplicável, os resultados serão demonstrados dentro do intervalo de confiança de aproximadamente 95%[1]. Os resultados foram tabelados diretamente ou foram plotados utilizando a ferramenta `GnuPlot`.

4.1.3 Reprodução de Resultados da Literatura

Outra categoria interessante de experimentos feitos foi a de reprodução de resultados conhecidos da literatura sobre CCN, os quais foram originalmente obtidos com *testbeds* reais. Será mostrado que, mesmo que eventualmente em uma escala menor, é possível obter o mesmo resultado utilizando o Mini-CCNx. O objetivo aqui é demonstrar o realismo e a fidelidade da ferramenta.

4.1.4 Testes Abertos

Espera-se que o Mini-CCNx evolua e seja utilizado por outros pesquisadores mesmo após a conclusão desse trabalho de pesquisa. Por estar divulgado publicamente, espera-se que diversos interessados ao redor do mundo testem e contribuam com melhorias para a evolução da ferramenta[2].

4.2 Experimentos e Resultados

Os experimentos realizados nessa seção visam avaliar a ferramenta em relação a quatro fatores. O primeiro, escalabilidade, visa investigar o consumo de recursos do Mini-CCNx no *hardware* de referência conforme aumentam o número de nós, enlaces e prefixos de nome CCN. Os testes de coerência, por sua vez, buscam avaliar se características básicas da implementação, como por exemplo os atrasos dos *links* e o *caching* nos nós, estão sendo de fato verificadas. Já os testes de fidelidade comparam o comportamento do emulador ante um cenário real. Por fim, a análise de isolamento investiga se de fato a utilização de *cgroups* fornece um melhor resultado geral dos experimentos com uma menor interferência entre os contêineres.

[1] Utilizando a chamada "Regra dos 3-sigma" a qual diz que, para uma distribuição normal, praticamente todos os valores se encontram dentro de 3 desvios padrões da média. Ou, equivalentemente, cerca de 95% dos valores estão dentro de 2 desvios padrões.

[2] O que já tem acontecido apesar da pequena divulgação feita até o momento da escrita desse texto (restrita a contatos pessoais ou a listas de *email*).

4.2.1 Escalabilidade

A escalabilidade da ferramenta será analisada, primeiramente, em termos de quantos nós e enlaces ela é capaz de instanciar simultaneamente. Para isso, foram escolhidas duas topologias representativas, *full mesh* e linear. Com a topologia *full mesh* (Tabela 4.1), todos os nós possuem conexões para todos os outros nós. Já na topologia linear (Tabela 4.2), os nós são ligados linearmente entre si. Tais configurações foram escolhidas justamente por representarem os dois extremos de conectividade entre os nós - qualquer outra topologia apresentará um nível de conectividade que estará entre esses dois extremos.

Tabela 4.1: Escalabilidade. Topologia *full mesh*

Nº de Nós	**Mem. Mini-CCNx (MB)**	**Mem. *ccnd* (MB)**	**Mem. Total (MB)**	**Nº de *Links***	**Tempo de inic. (s)**
4	15.1	7.2	22.3	6	<1
8	15.3	13.7	29.0	28	3
16	15.7	28.9	44.6	120	11
32	18.2	57.1	75.3	496	48
64	26.0	114.5	140.6	2016	118
128	62.0	229.8	291.8	8128	753

Tabela 4.2: Escalabilidade. Topologia linear

Nº de Nós	**Mem. Mini-CCNx (MB)**	**Mem. *ccnd* (MB)**	**Mem. Total (MB)**	**Nº de *Links***	**Tempo de inic. (s)**
4	15.0	7.2	22.2	3	<1
16	15.1	28.9	44.0	15	1
64	15.7	113.8	129.5	63	6
256	18.1	458.6	476.7	255	36
512	21.6	918.5	940.1	511	95
1024	27.8	1835.4	1863.2	1023	228
1536	35.3	2754.2	2789.6	1535	320

As Tabelas 4.1 e 4.2 mostram o número de nós instanciados, a memória utilizada especificamente pelo Mini-CCNx, a memória utilizada especificamente pelas instâncias do *daemon ccnd* que executam em cada nó, a memória total utilizada (Mini-CCNx + *daemons ccnd*), o número de *links* instanciados e o tempo de iniciação de cada cenário. O critério de parada em ambos os cenários é ou um tempo de iniciação maior que 10 minutos ou um consumo total de memória maior que 75% do que se tem disponível na máquina. Pode-se notar que a principal parcela do uso de memória se refere ao uso de memória pelo *daemon ccnd*, memória essa que cresce linearmente com o número de nós. O tempo de iniciação está fortemente relacionado ao número de *links* instanciados. Por isso, nota-se um tempo maior em uma topologia *full mesh* quando comparada com a topologia linear. A Tabela 4.3 mostra um exemplo de consumo de recursos para a emulação da topologia do *testbed* NDN (apresentada no apêndice B dessa dissertação).

Uma segunda análise referente à escalabilidade é feita com relação à quantidade e tamanho dos prefixos que o Mini-CCNx consegue instanciar nos nós de uma topologia. Na análise

Tabela 4.3: Escalabilidade. Topologia emulada do *testbed* NDN

Nº de Nós	**Mem. Mini-CCNx (MB)**	**Mem. *ccnd* (MB)**	**Mem. Total (MB)**	**Nº de *Links***	**Tempo de inic. (s)**
17	16.1	37.6	53.7	38	6

anterior, os nós CCN estavam com as tabelas de encaminhamento praticamente vazias, com a exceção de algumas entradas padrão. O intuito agora é analisar como a inserção de mais entradas (prefixos de nomes CCN) afetam o consumo de recursos da máquina na qual o emulador está sendo executado. Para isso, duas topologias representativas serão utilizadas: uma com 4 nós linearmente ligados (Tabela 4.4) e outra com 16 nós também linearmente ligados (Tabela 4.5). Em ambos os casos, variou-se tanto o número de prefixos inseridos em cada nó quanto o comprimento de cada um. Os nomes dos conteúdos foram criados de maneira aleatória, formados por caracteres, números e o símbolo "/" (que faz a divisão dos nomes CCN em componentes). Cada célula das Tabelas 4.4 e 4.5 contém a soma da memória total consumida (Mini-CCNx + *daemons* `ccnd`) por todos os nós do cenário e também o tempo de iniciação.

Tabela 4.4: Memória total e tempo de iniciação para topologia com 4 nós em função do tamanho e número de prefixos por nó

		Caracteres/prefixo		
		10	**100**	**1000**
Prefixos/nó	**10**	24.0 MB - 2 seg	24.5 MB - 2 seg	24.8 MB - 2 seg
	100	25.8 MB - 9 seg	26.0 MB - 10 seg	27.3 MB - 10 seg
	1000	35.8 MB - 84 seg	37.3 MB - 85 seg	52.3 MB - 91 seg
	10000	118.7 MB - 830 seg	149.0 MB - 840 seg	802.3 MB - 875 seg

Tabela 4.5: Memória total e tempo de iniciação para topologia com 16 nós em função do tamanho e número de prefixos por nó

		Caracteres/prefixo		
		10	**100**	**1000**
Prefixos/nó	**10**	46.7 MB - 6 seg	49.5 MB - 6 seg	49.7 MB - 7 seg
	100	55.5 MB - 47 seg	55.9 MB - 46 seg	63.8 MB - 49 seg
	1000	105.7 MB - 412 seg	119.7 MB - 414 seg	184.5 MB - 448 seg

Primeiramente, é importante notar que, em ambos os casos, o número de *links* não é mais o principal fator para o aumento do tempo de iniciação já que o número de enlaces é baixo para ambas as topologias. O principal fator influenciando no tempo de iniciação agora é o número de entradas nos nós, independentemente do tamanho delas (nota-se um tempo relativamente próximo para os diferentes tamanhos de prefixo). Esse tempo mais elevado é resultado da inserção individual de cada entrada no *daemon* `ccnd` através da ferramenta `ccndc` (como explicado na seção 3.3.1 do capítulo anterior). Em geral essa operação é custosa, principalmente para um maior número de entradas, porém é a maneira mais simples de se inserir estaticamente entradas nas FIBs dos nós com a implementação atual do CCNx.

Já o consumo total de memória também é influenciado pelo número de prefixos em cada nó. Além disso, o tamanho dos prefixos parece fazer aumentar o consumo de memória principalmente nos casos com mais entradas por nó. Isso pode ser explicado pela estrutura interna do `ccnd` que armazena tais entradas (NPHT - *Name Prefix Hash Table*). Essa tabela *hash* é indexada por componentes do nome (cada sequência de caracteres e/ou números incluídos entre duas barras) e não pelo nome completo em si. Conforme o tamanho do prefixo aumenta, aumenta a possibilidade de o nome ter mais componentes e, assim, aumenta também a memória ocupada pelo *daemon* `ccnd`. Em (Yuan et al. 2012) pode-se encontrar uma análise mais completa das estruturas internas do `ccnd`, além de um estudo sobre os efeitos do números de componentes do nome sobre a escalabilidade geral do modelo CCN. Em geral, os autores também notaram um maior consumo de recursos e maior degradação de desempenho na situação em que os prefixos possuíam um maior número de componentes.

Escalabilidade e Alocação de Recursos

O Mini-CCNx permite, com os *cgroups*, a alocação eficiente de recursos da máquina de forma a prover isolamento de desempenho aos contêineres. Porém, como o usuário pode saber se a quantidade de CPU e memória foi alocada adequadamente aos nós de seu cenário Mini-CCNx? Como saber se há folga nessa alocação ou se eventualmente algum nó está com falta de recursos? A resposta para essas perguntas vai depender exatamente do tipo de aplicação e cenário que o usuário estiver executando no emulador.

A solução mais completa para levantar esse perfil de uso de recursos pelas aplicações pode ser obtido pelo uso de ferramentas especializadas em *performance tracing*. Uma ferramenta de amplo uso por usuários Linux é o LTTng (LTTng 2013), com a qual é possível obter estatísticas detalhadas de desempenho não somente relacionadas à CPU ou memória, como também ao uso do sistema de E/S, chamadas de sistema (*system calls*), chaveamento entre *threads*, dentre várias outras. Com uma ferramenta como essa, é possível obter, por exemplo, informações sobre a utilização de CPU por uma aplicação orientada a conteúdo sendo desenvolvida e, assim, pode-se usar tal informação para alocar CPU de maneira eficiente no Mini-CCNx. A grande desvantagem de se utilizar uma ferramenta desse tipo é o custo e a complexidade de tal análise, que requer fazer o *parsing* de vários arquivos de *trace* e uso de outras ferramentas complementares. Além disso, também é necessário dedicar um tempo para o aprendizado de tal ferramenta - o que certamente é algo interessante, porém não trivial.

Uma abordagem alternativa (mas que leva a resultados satisfatórios) seria fazer uso de ferramentas Linux como `top` ou `htop` para observar, dinamicamente, o uso de recursos pelos nós Mini-CCNx. Assim, pode-se iniciar o cenário com uma alocação arbitrária e observar se alguns nós estão frequentemente chegando a 100% de sua alocação de CPU, por exemplo. Caso isso ocorra, basta alterar o arquivo de configuração (alocando mais CPU para tais nós e reduzindo para outros) e reiniciar o cenário. Em geral, essa abordagem também pode ser utilizada para determinar se o *hardware* utilizado pelo usuário é suficiente para a execução de seu cenário CCN utilizando o Mini-CCNx. Caso não seja, o usuário precisará de um *hardware* com mais recursos ou, eventualmente, utilizar outro tipo de ferramenta. Porém, em geral, será mostrado nos próximos experimentos que o Mini-CCNx permite a execução eficiente de diversos cenários

com aplicações bastante complexas como, por exemplo, distribuição de vídeo e geração exaustiva de tráfego CCN.

4.2.2 Coerência

Outra importante análise a ser feita é a de coerência ante a variação de parâmetros de experimento como atraso e banda dos *links* e número de *hops*. A ferramenta pode ser considerada coerente se, por exemplo, o atraso total aumentar linearmente com o número de *hops*.

Para essa análise, dois cenários simples utilizando nós CCN foram analisados. No primeiro (Figura 4.1), o RTT (*round-trip time*) foi medido com o utilitário `ccnping` (CCN Ping 2013) para diferentes números de *hops* e diferentes valores de atrasos configurados nos *links* da topologia. O número de *hops* variou entre 1 e 10 e os valores utilizados para o atraso foram 0 (desprezível), 5ms, 10ms e 50ms. Os nós são ligados linearmente. Os valores são apresentados com intervalo de confiança de 95% (apesar de nesse caso específico a escala não permitir a visualização das barras de erro).

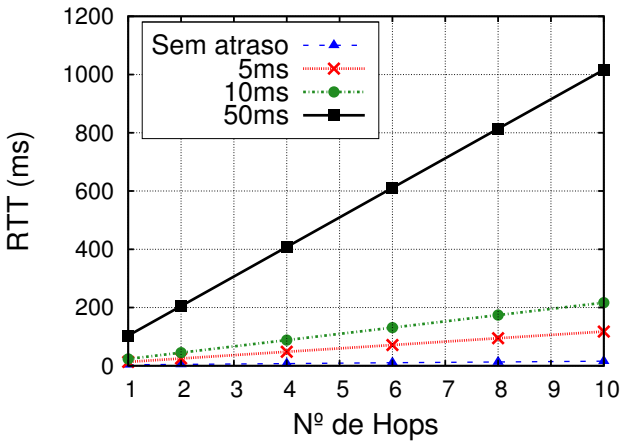


Figura 4.1: RTT vs nº de *hops* para diferentes valores de atraso de *link*

Pode-se notar o crescimento linear do RTT com o aumento do número de *hops*, como esperado. Além disso, para os diferentes valores de atraso nos *links*, o comportamento do RTT é consistente. Por exemplo, se todos os *links* têm atraso de 10ms e o número de *hops* é 2, o *ping* deve levar 20ms na ida, mais 20ms na volta, mais o tempo de processamento em cada nó, o que é justamente o comportamento observado no experimento[3].

No segundo cenário (Figura 4.2), foi medido o tempo médio de *download* de um arquivo de 100 MB para vários números de *hops*, ou seja, fazendo com que o produtor do conteúdo ficasse cada vez mais distante do cliente. Para cada valor de número de *hops*, o cliente faz duas requisições pelo mesmo arquivo. Na primeira, os *caches* dos diversos nós do caminho estão inicialmente limpos. Ao fim dela, os *Content Objects* são armazenados nos nós do caminho e assim que essa primeira requisição é concluída, o cliente pede novamente pelo mesmo arquivo.

[3]O CCNx é atualmente implementado no espaço de usuário e não é tão otimizado como a pilha TCP/IP do *kernel* Linux. Além disso, o `ccnping` funciona como uma aplicação externa ao *daemon* principal `ccnd`. Isso pode justificar o tempo de processamento nos nós relativamente alto de aproximadamente 3ms.

Ou seja, nessa segunda requisição, é provável que grande parte dos *Interests* enviados pelo cliente obtenham *Content Objects* dos *caches* dos nós mais próximos. Novamente, os nós são ligados linearmente e os dados apresentados dentro do intervalo de confiança de 95%.

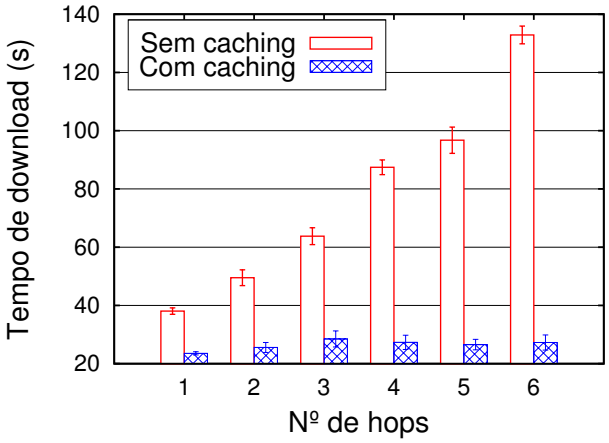


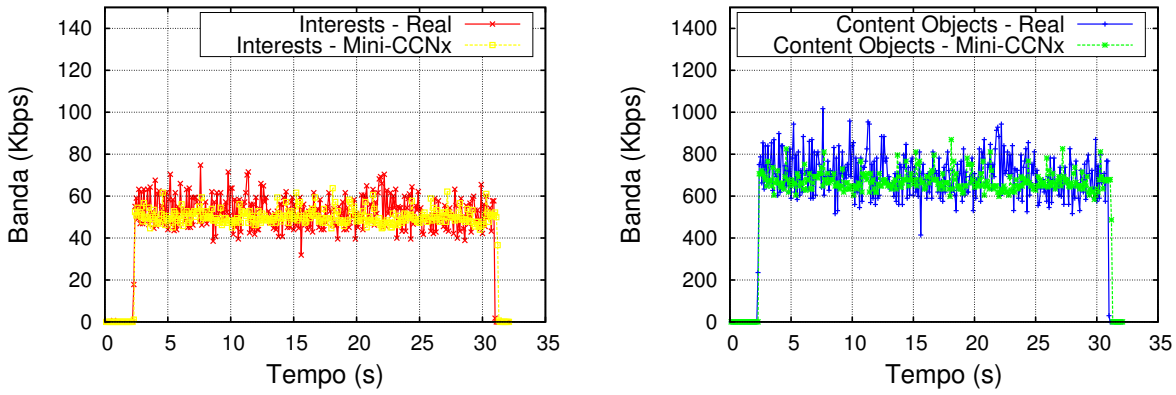
Figura 4.2: Tempo de *download* vs nº de *hops* com e sem *caching*

Na Figura 4.2 nota-se o aumento no tempo de *download* conforme aumenta a distância entre produtor e cliente do conteúdo para o caso sem *caching*, como esperado. Já no caso com *caching*, percebe-se que o tempo médio de *download* é menor do que no caso anterior e pouco relacionado ao número de *hops*. Nesse caso, o tempo de *download* é mais influenciado pela quantidade de conteúdo em *cache* nos nós mais próximos ao requisitante.

4.2.3 Fidelidade ante experimentos reais

O Mini-CCNx deve ser capaz de reproduzir experimentos reais com fidelidade. Para mostrar isso, foi criada uma simples topologia com dois *desktops* reais com placas de 100Mbps ligados diretamente entre si, ambos com instalações nativas do CCNx. Em seguida, utilizando o gerador de tráfego `ccntraffic`(CCNx Traffic 2013), o primeiro *desktop* gera constantemente mensagens *Interest* requisitando diferentes conteúdos enquanto o segundo responde com *Content Objects* de 1024 *bytes*. O mesmo cenário foi reproduzido utilizando o Mini-CCNx, com 100Mbps e 200 μs de atraso como parâmetros de *link*. A Figura 4.3 mostra o tráfego de *Interests* e *Content Objects* para ambos os cenários.

Pode-se notar que o comportamento da banda utilizando o Mini-CCNx é muito próximo do comportamento do ambiente real. Apesar de simples, os resultados desse experimento mostram uma característica típica de resultados que podem ser obtidos com um emulador. A média dos valores obtidos certamente é muito próxima para os dois casos, porém pode-se notar uma maior variância nos resultados do ambiente real se comparados aos do ambiente emulado. Essa diferença provavelmente ocorre pelo fato do emulador ser um ambiente mais isolado se comparado ao ambiente real, no qual pode haver algum tipo de interferência de sinal no cabo, por exemplo. Além disso, as ferramentas que emulam a banda e atraso (`tc` e `netem`) são implementadas em *software* e certamente não conseguem representar com completa fidelidade todas as características físicas relacionadas à transmissão de dados. De qualquer forma, os resultados obtidos com o



(a) Comportamento dos *Interests*

(b) Comportamento dos *Content Objects*

Figura 4.3: Comparação do comportamento da banda entre ambiente real e Mini-CCNx

Mini-CCNx são compatíveis com os estipulados nos objetivos do primeiro capítulo: reproduzir tendências e comportamentos na mesma ordem de grandeza do ambiente real.

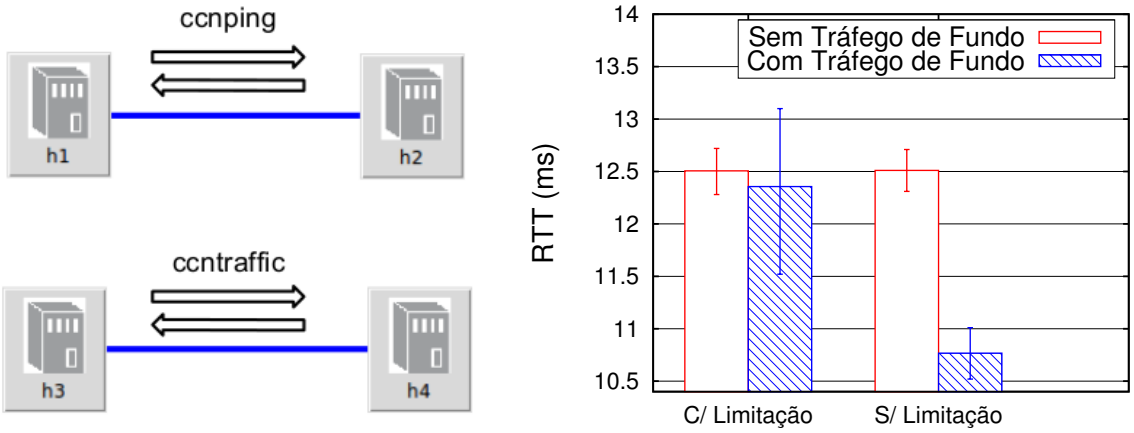
4.2.4 Isolamento

Tipicamente, um cenário de testes de ROCs contém vários fluxos de conteúdo e nós operando em partes distintas da rede. É de se esperar que o tráfego em uma parte da rede não afete os resultados de outra parte que não esteja conectada à primeira. Por mais simples que isso possa parecer, o isolamento é um desafio para o Mini-CCNx, já que diferentes nós e fluxos executando na ferramenta estarão utilizando basicamente o mesmo *kernel*, escalonador de processos e escalonador de *links*.

Porém, utilizando a limitação de CPU para cada nó se certificando de que todos os processos tenham tempo suficiente para executar suas instruções (o que é aferido, empiricamente, pela presença de tempo ocioso de CPU para cada *cgroup*), além de alocar corretamente a banda e a configuração de atrasos nos *links*, é possível obter resultados muito melhores do que os obtidos sem essa limitação.

Para destacar essa questão, uma topologia com 4 nós CCN foi construída. Os nós **h1** e **h2** são conectados diretamente e trocam `ccnping`s. Os nós **h3** e **h4** também são ligados diretamente entre si e trocam tráfego utilizando `ccntraffic`, tráfego esse que chamaremos de tráfego de fundo. Não há nehuma ligação entre as duas redes. Serão observados os valores de RTT entre **h1** e **h2** na ausência e na presença do tráfego de fundo entre **h3** e **h4**. Como estão em redes isoladas, o esperado em uma rede real é que o RTT entre **h1** e **h2** não seja influenciado pela ausência ou presença de tráfego entre **h3** e **h4**. A Figura 4.4 mostra a topologia e o comportamento do RTT para o caso sem e com limitação de recursos. Os valores do gráfico apresentam intervalo de confiança de 95%.

Pode-se notar que, na presença do tráfego de fundo, há uma queda no valor do RTT para o caso onde não se limita os recursos, ou seja, não há isolamento. É possível ver que, mesmo considerando as barras de erro, não há sobreposição dos valores, ou seja, podemos considerar que os valores RTT são estatisticamente diferentes. Por outro lado, quando há limitação de recursos,



(a) Topologia

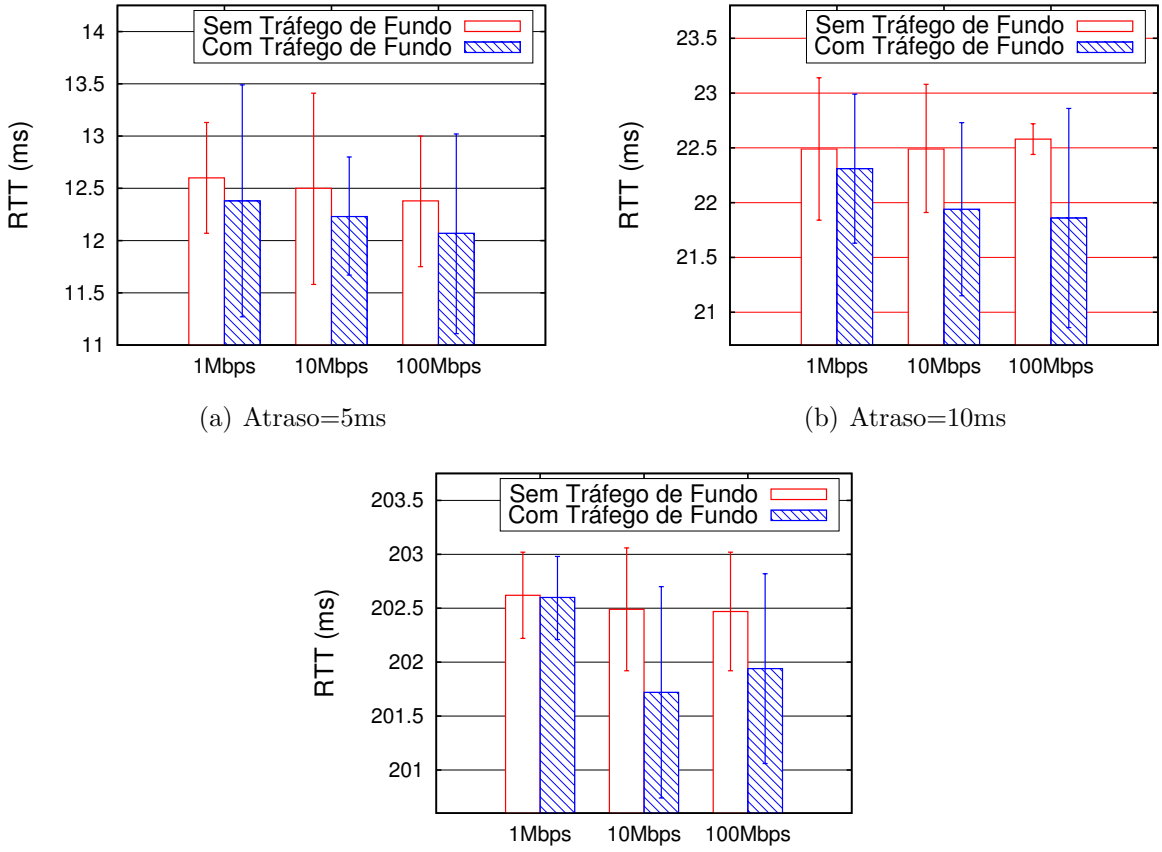
(b) RTT com e sem limitação de recursos

Figura 4.4: Análise de isolamento

os valores médios dos RTTs são muito próximos, o que pode ser confirmado pela sobreposição das barras de erro.

Para analisar ainda mais o caso de isolamento no caso de limitação de recursos, mais alguns cenários foram testados. A topologia utilizada é a mesma do caso anterior. Cada *host* teve reservado para si 50% de CPU (do total de 400% disponíveis na máquina de teste segundo a definição do Linux, com 2 *cores* de duas *threads* físicas.). Para diferentes valores de atraso e a banda, o RTT foi medido sempre comparando o comportamento com e sem o tráfego de fundo, que sempre ocupava cerca de 60% da banda total disponível no *link*. A Figura 4.5 mostra os gráficos com as diversas combinações de parâmetros, nos quais todos os valores possuem o intervalo de confiança de 95%.

Para todos os casos, nota-se a sobreposição entre as barras de erro. Portanto, pode-se concluir que o Mini-CCNx, se corretamente configurado, pode prover um ambiente de testes com isolamento. É importante notar que, em uma primeira análise, o comportamento do `ccnping` quando o tráfego de fundo é ativado pode parecer incoerente: o natural seria que o RTT aumentasse na presença de tal tráfego. Porém, em todos os gráficos, o inverso acontece - pode-se notar que o RTT diminui. Esse efeito ocorre devido a uma característica experimental bem peculiar, relacionada às funcionalidades de economia de energia do processador (Intel 2007) do *hardware* no qual o emulador está sendo executado. O programa `ping` tem um comportamento intermitente, ou seja, um pacote *ping* é enviado e o processo bloqueia até o momento de envio do próximo *ping*, no próximo segundo. Durante esse tempo de espera, para economizar energia, o processador entra em um estado de *sleep*. Um certo tempo é necessário para que o processador "desperte" desse estado e, portanto, o RTT final é mais longo se comparado a uma situação na qual o processador nunca entra nesse estado de economia de energia. O tráfego de fundo desse experimento faz exatamente isso - a geração constante impede que o sistema entre no estado de economia de energia e assim o RTT é menor nessa situação.



(a) Atraso=5ms

(b) Atraso=10ms

(c) Atraso=100ms

Figura 4.5: Análise de isolamento para diferentes valores de atraso e banda

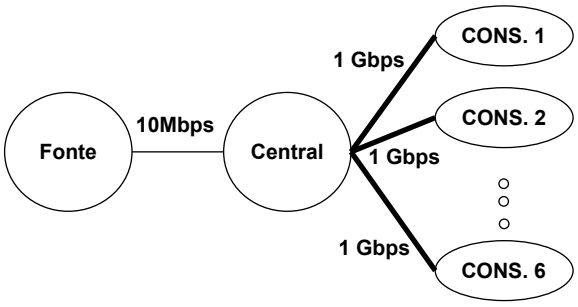
4.2.5 Reprodução de Resultados da Literatura

Eficiência na Distribuição de Conteúdos

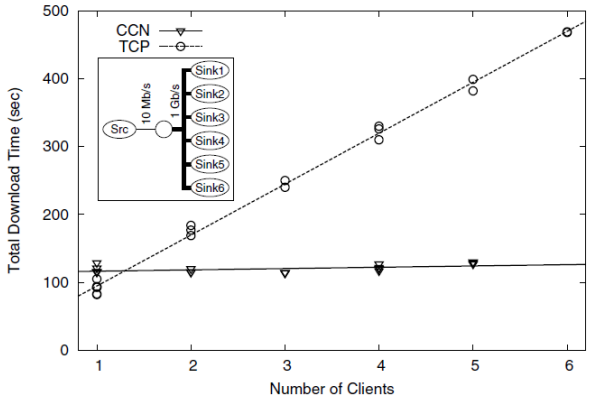
O artigo original do modelo (Jacobson, Smetters, Thornton, Plass, Briggs & Braynard 2009) mostra o bom desempenho do CCN para a distribuição de conteúdos (resultado mostrado na Figura 4.6(b)). Uma comparação entre CCN e TCP é feita medindo o tempo total de *download* simultâneo de um arquivo de 6MB através de um gargalo de rede. O teste utilizou um nó fonte ligado a um nó central através de um enlace de 10Mbps (o gargalo). Esse nó central, por sua vez, é ligado a um *cluster* de consumidores interconectados por enlaces de 1Gbps (Figura 4.6(a)). Os consumidores faziam o *download* do arquivo armazenado no nó fonte simultaneamente. Os resultados originais mostraram que o CCN tem melhor desempenho que o TCP com dois consumidores ou mais devido à característica de *caching* que evita o uso constante do enlace de menor capacidade.

O Mini-CCNx foi utilizado para reproduzir esse comportamento. Para o caso do TCP, utilizou-se a ferramenta Mininet original e o utilitário `wget` para fazer a requisição pelo arquivo armazenado em um servidor HTTP[4].Para o CCN, a requisição foi feita utilizando a ferramenta

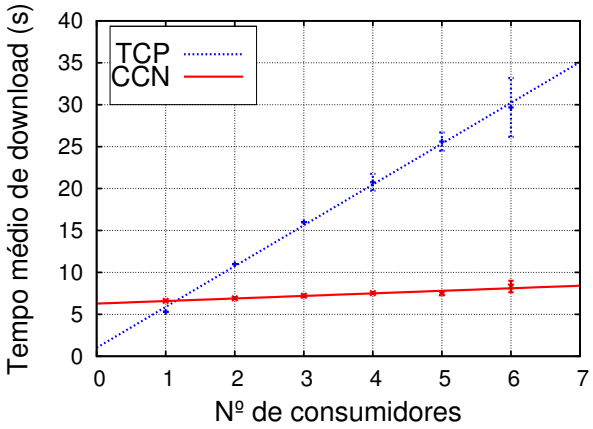
[4]No caso, utilizou-se o servidor `lighttpd` (lighttpd 2013) por ser um servidor mais leve que a opção padrão pelo `Apache` (Apache Software Foundation, 2013). Em geral, essa é uma boa prática ao se utilizar um emulador



(a) Topologia do teste



(b) Resultado original (reproduzido de (Jacobson, Smetters, Thornton, Plass, Briggs & Braynard 2009))



(c) Tempo médio de *download* vs nº de consumidores

Figura 4.6: Comparação de eficiência de distribuição de conteúdos entre CCN e TCP

`ccngetfile` (incluída com o código do CCNx). A Figura 4.6(c) mostra os resultados obtidos.

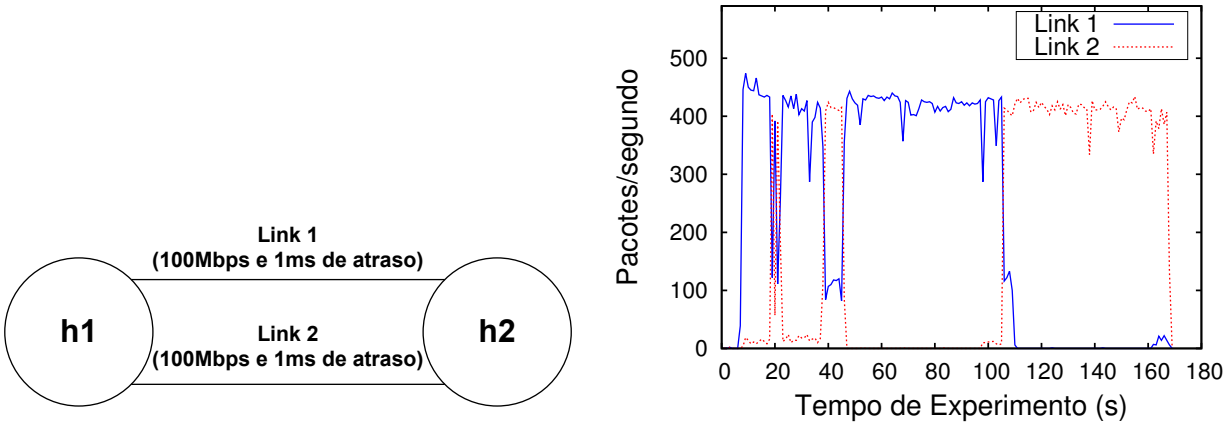
O CCN, de fato, supera o TCP com dois consumidores ou mais. Além disso, pode-se notar a maior dependência do TCP com relação ao número de consumidores (maior coeficiente angular), justamente pela superutilização do enlance de menor capacidade. Para o CCN, teoricamente não deveria haver dependência com relação ao número de consumidores (coeficiente angular igual a zero) já que os mesmos buscam o conteúdo diretamente do *cache* do nó central (o arquivo percorre o enlace de menor capacidade apenas uma vez até popular tal *cache*) e não concorrem pela banda limitada do enlace de 10Mbps. Porém, a pequena inclinação da reta referente ao CCN ocorre devido ao processamento paralelo efetuado pelo nó central e que, conforme aumenta o número de consumidores, afeta a taxa de transmissão do mesmo.

Camada de Estratégia/Encaminhamento

O artigo original também traz um experimento mostrando como um nó CCN se comporta quando ele possui múltiplas entradas na FIB para um mesmo prefixo. Basicamente, é mostrado que a camada de estratégia escolhe dinamicamente a melhor interface no momento e se, por exemplo, o *link* utilizado falhar, a pilha CCN automaticamente escolhe a próxima interface para enviar os pacotes. A esse comportamento dá-se o nome de recuperação automática (*automatic*

para economizar recursos da máquina.

failover). Ao longo do tempo, a pilha também envia *Interests* de prospecção através de outras interfaces além da atual "melhor", medindo o RTT e analisando assim a possibilidade de se escolher uma nova melhor interface para o envio dos pacotes.



(a) Topologia do teste

(b) Consumo de banda em cada *link*

Figura 4.7: Recuperação automática efetuada pela camada de encaminhamento

Esse comportamento será reproduzido com o Mini-CCNx. Dois *hosts* (**h1** e **h2**) são conectados diretamente por dois *links*, *link* 1 e *link* 2 (Figura 4.7(a)), de mesma configuração (100Mbps e 1ms de atraso). O primeiro *host* envia pacotes *Interest* constantemente requisitando diferentes conteúdos enquanto o segundo cria esse conteúdo dinamicamente em resposta e os envia em pacotes *Data*[5]. Na Figura 4.7(b), vemos que a camada de estratégia escolhe o *link* 1 na maior parte do tempo mas envia constantemente pacotes de prospecção através do *link* 2. Aproximadamente em 40 segundos a estratégia parece começar a escolher o *link* 2 na maior parte do tempo mas, aos 45 segundos, o *link* 2 é desconectado até aproximadamente o tempo de 70 segundos. O Mini-CCNx pode fácil e dinamicamente desconectar e conectar os *links* no momento que o usuário desejar. Vê-se que a camada de estratégia automaticamente escolhe o *link* 1 novamente. Aproximadamente aos 110 segundos, o *link* 1 é desconectado até o tempo de 125 segundos. Novamente, vê-se que a camada de estratégia seleciona automaticamente o *link* 2. Portanto, com o Mini-CCNx foi possível reproduzir o comportamento de recuperação automática realizado pela camada de estratégia.

Aplicação de *streaming* de vídeo em tempo real

Os Relatórios Técnicos do Projeto NDN (NDN Project 2012b) apresentam diversos experimentos que visam mostrar o comportamento de aplicações CCN em condições reais. Nessa subseção, tentar-se-á reproduzir alguns resultados de tais experimentos utilizando o Mini-CCNx. Se for possível obter resultados próximos, então talvez os experimentos reais pudessem ter se beneficiado de uso anterior do Mini-CCNx. Não se alega aqui que experimentos em *testbeds* reais

[5]Diferentemente do artigo original que utiliza o tráfego de voz (Jacobson, Smetters, Briggs, Plass, Stewart, Thornton & Braynard 2009), aqui foi utilizado o gerador de tráfego `ccndelphi`. O intuito aqui não é reproduzir exatamente o gráfico obtido originalmente mas sim reproduzir o comportamento de recuperação automática e escolha dinâmica de *links*.

não devam ser feitos - alega-se apenas que alguns resultados e o comportamento das aplicações podem ser previamente detectados usando o Mini-CCNx reduzindo assim o custo e o tempo gasto nos ambientes reais.

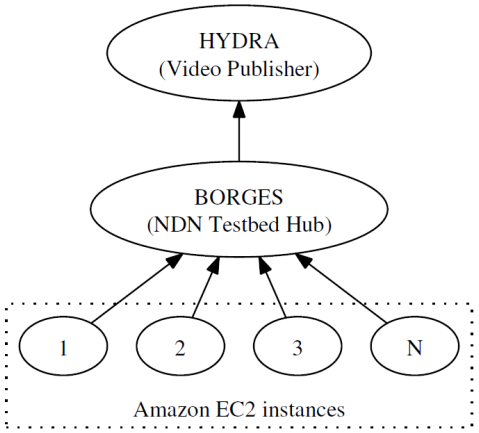


Figura 4.8: Topologia utilizada no teste (Kulinsk & Burke 2012)

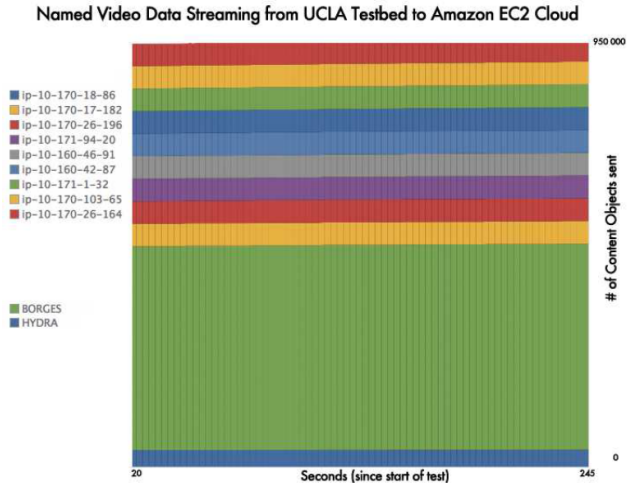


Figura 4.9: Número de pacotes enviados pelo produtor, roteador central e recebidos pelos consumidores. No caso, foram utilizados 9 consumidores (Kulinsk & Burke 2012)

Um dos relatórios apresenta a aplicação NDNVideo, uma aplicação de streaming de vídeo em tempo real que é capaz de reproduzir vídeos gravados ou transmitidos ao vivo sem negociação de sessão e com a possibilidade de acesso aleatório (Kulinsk & Burke 2012). Durante os testes, os desenvolvedores detectaram um comportamento interessante em um ambiente (Figura 4.8) com múltiplos consumidores, um roteador central (denominado *Borges*) e um produtor de vídeo (denominado *Hydra*): cada consumidor recebeu uma quantidade ligeiramente maior do que o produtor enviava. A Figura 4.9 mostra o comportamento observado utilizando o gráfico do próprio relatório. Nota-se que as barras que marcam a quantidade de conteúdo recebido pelos consumidores são ligeiramente maiores do que a do produtor.

O Mini-CCNx foi utilizado para tentar reproduzir tal comportamento (Figura 4.10). Com a mesma topologia mas com um número menor de consumidores (cinco) e com um tempo menor

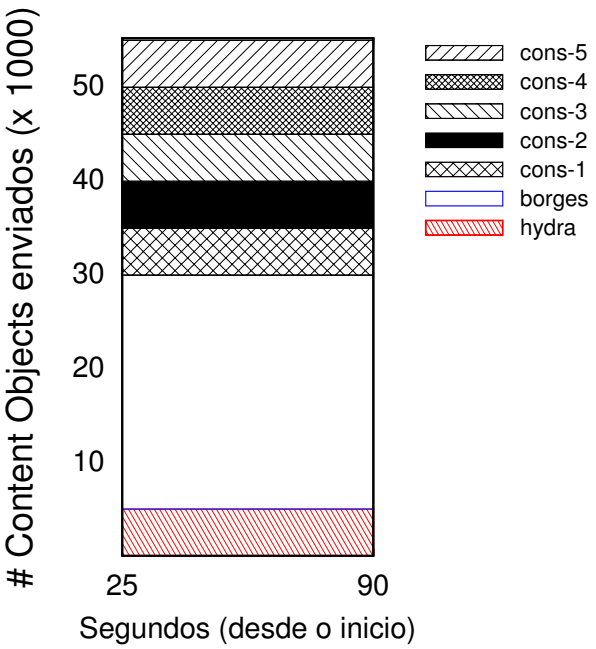
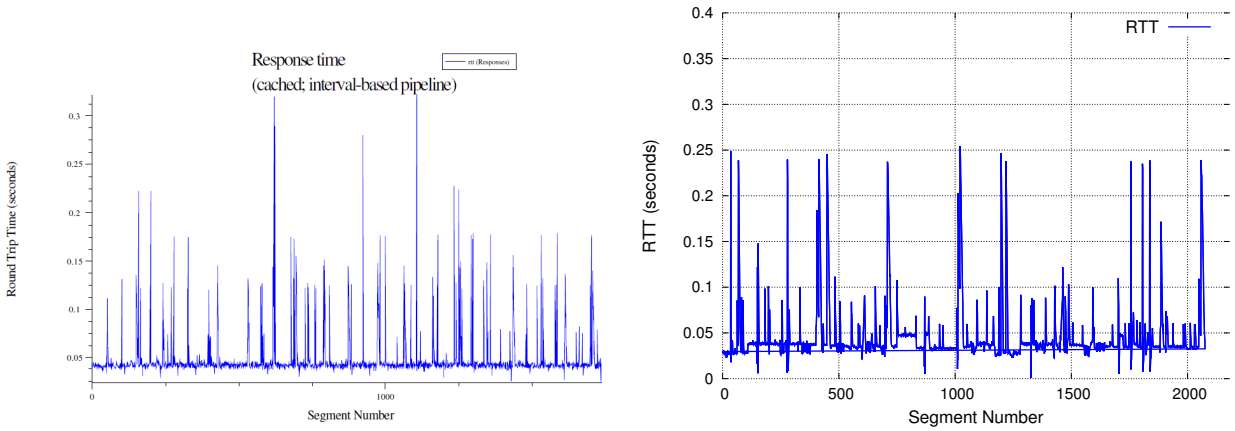


Figura 4.10: Comportamento observado utilizando o Mini-CCNx



(a) Resultado apresentado no relatório (Kulinsk & Burke 2012)

(b) Resultado obtido com o Mini-CCNx

Figura 4.11: Comportamento do RTT após melhorias na aplicação cliente do NDNVideo

de teste, o mesmo resultado foi observado. Estimou-se o atraso de propagação dos *links* como 10ms e uma taxa de perda de 0.1%.

Os desenvolvedores do NDNVideo concluíram que uma característica específica da aplicação (a reordenação de *Interests* feitas pelo CCNx para suportar a funcionalidade de *multicasting* e a maneira como era feito o *pipeline* de *Interests*) causava uma situação na qual os consumidores re-enviavam *Interests* para o mesmo conteúdo e portanto recebiam, de fato, uma maior quantidade de *Content Objects*. Eventualmente ocorriam atrasos muito altos e até perda de pacotes. Para resolver esse problema, foram feitas uma série de melhorias no lado do cliente, como estimação de *timeouts* e o uso de um *pipeline* baseado em intervalos de tempo e assim o comportamento do RTT de fato melhorou apresentando uma menor média geral e também menores picos. A

Figura 4.11(a) mostra o resultado obtido no relatório. A Figura 4.11(b) mostra, por sua vez, o comportamento obtido com o Mini-CCNx utilizando essa versão melhorada da aplicação.

Portanto, nota-se que o Mini-CCNx conseguiu reproduzir o comportamento obtido no relatório com uma boa qualidade. Dessa forma, os desenvolvedores do NDNVideo poderiam ter utilizado o emulador, tanto para detectar o problema de sua aplicação quanto para analisar a solução proposta, antes de partir para o *testbed* real reduzindo assim o custo total associado aos testes em ambientes reais.

4.2.6 Experimentos com Roteamento

Nessa subseção serão apresentados alguns experimentos relacionados ao roteamento OSPFN (descrito no capítulo 3). Será utilizada a topologia do *testbed* NDN apresentada na Figura B.1 do apêndice B dessa dissertação. O intuito é mostrar o funcionamento básico desse roteamento e também alguns tipos de análises que podem ser feitas utilizando o Mini-CCNx. Vale ressaltar que tais análises podem ser feitas tanto para o protocolo OSPFN (atualmente integrado ao Mini-CCNx) como para outras propostas de roteamento. Neste último caso, o usuário precisará instalar e configurar adequadamente as novas implementações nos nós Mini-CCNx. Também é importante deixar claro que em momento algum durante essa subseção foram inseridas entradas de roteamento de maneira estática nos nós - essa função foi exercida somente pelo protocolo de roteamento.

Operação Básica

Cada nó da topologia anuncia prefixos pelos quais eles serão responsáveis. A Figura 4.12(a) mostra os prefixos anunciados por alguns dos nós da topologia. Como explicado no capítulo 3, o OSPFN divulga tais prefixos através de OLSAs OSPF. Portanto, cada nó da topologia cria um OLSA para cada um de seus prefixos (um exemplo pode ser visto na Figura 4.12(b)) que será disseminado pelo protocolo OSPF por toda a topologia.

O OSPFN trata todas as mensagens recebidas (atualizações, inserções ou apagamentos) e dinamicamente atualiza uma tabela interna que mapeia prefixos e os roteadores de origem correspondentes. Por exemplo, o roteador de origem do prefixo "/ndn.ucla.edu/" é UCLA e o prefixo "/ndn/opschat/" tem dois roteadores de origem: CSU e NEU. Assim, se um prefixo for divulgado por um único roteador, o OSPFN cria uma entrada na FIB do nó com tal prefixo e próximo *hop* (obtido através de uma consulta à tabela de roteamento OSPF) para o roteador de origem. Por outro lado, se um prefixo for divulgado por múltiplos roteadores, serão criadas entradas na FIB com o próximo *hop* para cada um dos roteadores de origem, porém a rota com menor custo será inserida por último. Isso é feito pois, segundo a implementação do *ccnd*, a entrada inserida por último será a primeira a ser utilizada pelo encaminhamento, ou seja, terá maior prioridade. Para exemplificar tais situações, a Figura 4.13 mostra uma visão parcial da FIB do nó UCLA após a estabilização do roteamento na topologia, segundo a saída do comando `ccndstatus`.

Os prefixos "/ndn/colostate.edu/netsec/" e "/ndn/pku.edu.cn/" são anunciados por roteadores de origem únicos (CSU e PKU, respectivamente) e estão diretamente ligados ao nó UCLA

PKU
/ndn/pku.edu.cn/
UCLA
/ndn/ucla.edu/
NEU
/ndn/neu.edu/northpole/
/ndn/opschat/
CSU
/ndn/colostate.edu/netsec/
/ndn/opschat/

(a) Prefixos anunciados por alguns dos nós

UCLA OLSA	
Tipo LS	10
Tipo Opaco	236
Roteador Anunciante	0.0.0.14 (UCLA – Valor Arbitrário)
Informação Opaca	
14 (tamanho em *bytes* do prefixo)	
/ndn/ucla.edu/	

(b) Formato de um OLSA

Figura 4.12: Prefixos anunciados e formato dos OLSAs

```
...
ccnx:/ndn/colostate.edu/netsec face: 8 flags: 0x3 expires: 214748301   (face 8 = CSU)
ccnx:/ndn/pku.edu.cn face: 10 flags: 0x3 expires: 2147483012   (face 10 = PKU)
ccnx:/ndn/neu.edu/northpole face: 13 flags: 0x3 expires: 2147483012   (face 13 = UA)
ccnx:/ndn/neu.edu/northpole face: 9 flags: 0x3 expires: 2147483012   (face 9 = PARC)
ccnx:/ndn/neu.edu/northpole face: 8 flags: 0x3 expires: 2147483012   (face 8 = CSU)
ccnx:/ndn/opschat face: 13 flags: 0x3 expires: 2147483167   (face 13 = UA)
ccnx:/ndn/opschat face: 9 flags: 0x3 expires: 2147483167   (face 9 = PARC)
ccnx:/ndn/opschat face: 8 flags: 0x3 expires: 2147483167   (face 8 = CSU)
...
```

Figura 4.13: Visão parcial da FIB do nó UCLA

(custo de 1 *hop*). O prefixo "/ndn/neu.edu/northpole/" também é anunciado por único roteador (NEU) porém há três entradas na FIB para tal prefixo. Isso ocorre pois há três caminhos de igual custo (3 *hops*) entre UCLA e NEU (ver Figura B.1). A ordem de inserção de entradas na FIB, nesse caso, é feita de maneira arbitrária. Por fim, o prefixo "/ndn/opschat/" é divulgado por dois roteadores. Portanto, são criadas as entradas referentes a tais roteadores de origem, CSU (rota (UCLA)-(CSU) com custo de 1 *hop*) e NEU (rotas (UCLA)-(PARC)-(UIUC)-(NEU) e (UCLA)-(UA)-(UM)-(NEU), ambas de custo 3 *hops*). A ordem de inserção na FIB agora é importante: a rota em direção a CSU (de menor custo) deve ser inserida por último de forma a dar maior prioridade a tal caminho. Isso pode ser verificado na Figura 4.13.

Conectividade

O primeiro experimento sobre a topologia da Figura B.1 será um teste básico de conectividade consistindo na troca de `ccnping`s entre alguns nós da rede. A Tabela 4.6 mostra o nó que originou o `ccnping`, o prefixo requisitado, o nó destino que respondeu por esse prefixo, o número de *hops* no caminho, a somatória dos atrasos configurados nos enlaces de tal caminho (RTT em ms), o RTT médio observado pela saída do comando *ccnping* (média de 100 envios realizados, em ms) e, finalmente, o tempo de processamento, em ms, gasto em cada nó. Esse tempo de processamento

é obtido pela diferença entre o RTT observado pelo `ccnping` e o RTT configurado nos enlaces do caminho, dividida pelo número de nós envolvidos. Por exemplo, a segunda linha da tabela mostra um `ccnping` que percorreu 4 *hops* (envolvendo, portanto, 5 nós) em 159,68 ms, em um caminho cujo atraso total configurado nos enlaces era de 146 ms. Assim, (159,68 - 146)/5 = 2,74 ms por nó. O intuito de se obter tal valor é apenas ter uma noção do tempo gasto por nó exclusivamente com o processamento do `ccnping` e não com o atraso de propagação dos *links* da topologia.

Tabela 4.6: Dados de teste básico de conectividade

Origem	**Prefixo**	**Destino**	**Nº de *hops***	**RTT enlaces (ms)**	**RTT médio (ms)**	**Tempo de processamento por nó (ms)**
UCLA	/ndn/colostate.edu/netsec/	CSU	1	14	19,57	2,79
PKU	/ndn/neu.edu/northpole/	NEU	4	146	159,68	2,74
UM	/ndn/opschat/	CSU	1	14	19,8	2,9
UCLA	/ndn/opschat/	CSU	1	14	19,75	2,87

A primeira linha mostra o caso mais simples, no qual os nós UCLA e CSU estão diretamente conectados (1 *hop*) e o prefixo requisitado é divulgado apenas pelo nó CSU em toda a topologia. A segunda linha mostra o `ccnping` entre dois nós distantes (4 *hops*), PKU e NEU, mas cujo prefixo requisitado também é divulgado somente por um nó (NEU). Nota-se que há três caminhos de custo 4 *hops* entre PKU e NEU, sendo que o caminho escolhido pelo plano de encaminhamento foi (PKU)-(UCLA)-(PARC)-(UIUC)-(NEU). Na terceira linha, temos o caso em que um prefixo ("/ndn/opschat/") é divulgado por dois nós, CSU e NEU. Nessa situação, o nó de origem (UM) possui duas entradas em sua FIB para tal prefixo, já que o custo é igual para ambos os caminhos (1 *hop*). O plano de encaminhamento, nesse caso, escolheu obter o conteúdo pelo nó CSU. Por fim, a quarta linha mostra o mesmo prefixo anterior sendo requisitado, porém, agora, o nó de origem é diferente (UCLA). Nesse caso, o custo menor de fato está no caminho até CSU (1 *hop*) e, assim, essa é a única entrada na FIB de UCLA para o prefixo "/ndn/opschat").

Em uma análise mais geral, pode-se notar um comportamento consistente dos atrasos na topologia. Além disso, o tempo de processamento por nó, em todos os casos, foi muito próximo e assim aponta para um comportamento consistente dos atrasos no *testbed* emulado.

Relação entre Roteamento e Plano de Encaminhamento

O próximo experimento mostra como o modelo CCN apresenta uma relação mais estreita entre os planos de roteamento e encaminhamento do que o modelo TCP/IP tradicional e como o plano de encaminhamento é capaz de tomar decisões de maneira independente, apenas com base em informações de alcançabilidade previamente disponibilizadas pelo roteamento. Suponha que o nó CSU queira se comunicar com o nó SPP-HOUS (que divulga o prefixo "/ndn/spphous1/"). Existem duas rotas de mesmo custo (2 *hops*) entre esses nós: (CSU)-(UA)-(SPP-HOUS) e (CSU)-(SPP-SALT)-(SPP-HOUS). Isso pode ser comprovado pelas duas entradas na FIB do nó CSU (Figura 4.14), cuja ordem de inserção foi arbitrária.

Apesar do custo dos dois caminhos, em número de *hops*, ser o mesmo, o atraso de propagação

```
...
ccnx:/ndn/spphous1 face: 7 flags: 0x3 expires: 2147483612   (face 7 = UA)
ccnx:/ndn/spphous1 face: 11 flags: 0x3 expires: 2147483612   (face 11 = SPP-SALT)
...
```

Figura 4.14: Visão parcial da FIB do nó CSU

total do primeiro caminho (12 ms) é ligeiramente menor do que o do segundo (13 ms). Ao se ativar o `ccnping` em CSU com destino a SPP-HOUS, é possível comprovar (com o *packet sniffer* `ndndump`) que o primeiro *Interest* utiliza do segundo caminho, através de SPP-SALT (respeitando a convenção de implementação do `ccnd`, pois tal entrada foi inserida por último na FIB de CSU). Porém, o nó também envia, logo em seguida, um *Interest* exatamente igual ao anterior mas agora o faz pelo primeiro caminho, através de UA. Essa prospecção continua por cerca de dez segundos, como é possível notar na Figura 4.15.

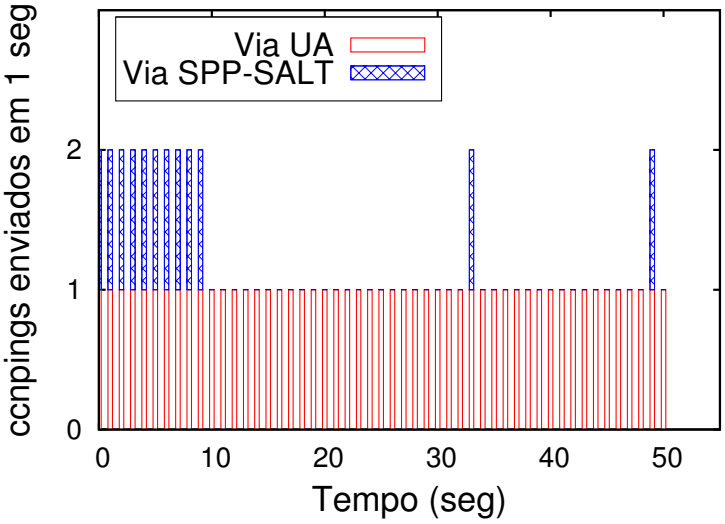


Figura 4.15: *ccnpings* saindo de CSU em direção a SPP-HOUS através de UA e de SPP-SALT

Para esses dez segundos de prospecção, foi medido o RTT dos pacotes por ambos os caminhos. Por UA (primeiro caminho), observou-se um RTT de médio de 29,5 ms com 0,41 ms de desvio padrão. Já pelo caminho que passa por SPP-SALT (segundo caminho), o RTT médio observado foi de 30 ms com 0,46 ms de desvio padrão. Portanto, lembrando que o primeiro caminho possuía um atraso de propagação menor do que o segundo, o plano de encaminhamento do nó acabou observando uma melhor condição de rede pelo primeiro caminhou e optou por utilizá-lo. Nota-se que o plano de encaminhamento enviou alguns pacotes de prospecção pelo caminho via SPP-SALT após essa decisão, mas isso não alterou a opção original pelo caminho via UA.

Portanto, esse experimento mostra que o roteamento atua obtendo informações de conectividade e disponibilidade dos diversos prefixos da rede. Porém, o modelo CCN não se limita a utilizar sempre o melhor caminho escolhido pelo roteamento, tal qual é o comportamento padrão do modelo TCP/IP. O plano de encaminhamento CCN combina informações de roteamento e de condições de tempo real da rede para escolher o melhor caminho para envio dos pacotes.

Alterações na Topologia e Convergência do Roteamento

Será avaliado agora como o OSPFN reage à alterações de topologia. O experimento se inicia com o roteamento já estabelecido, com a topologia da Figura B.1. Para uma melhor visualização, a Figura 4.16 mostra uma visão parcial do *testbed* apenas com os nós e enlaces relevantes ao experimento apresentado nessa subseção. Na Figura 4.13, vemos que o nó UCLA possui uma entrada na FIB para o prefixo "/ndn/colostate.edu/netsec/" apontando diretamente para o nó CSU, que é o roteador de origem para tal prefixo e está diretamente conectado a UCLA. Com o comando "`link ucla csu down`", o Mini-CCNx remove esse enlace de ligação direto entre ambos os nós no tempo 13:23:20. A Figura 4.17 mostra o *log* do *daemon* `ospfd` e também a FIB de UCLA logo após a queda desse enlace.

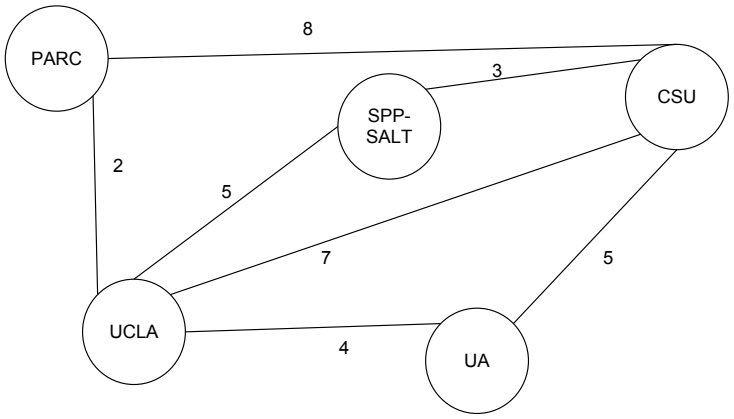


Figura 4.16: Nós e enlaces relevantes ao experimento

```
2013/03/26 13:23:20 OSPF: nsm_change_state(0.0.0.2, Full -> Deleted): scheduling new router-LSA
origination
2013/03/26 13:23:20 OSPF: ospf_apiserver_nsm_change
2013/03/26 13:23:20 OSPF: DR-Election[1st]: Backup 0.0.0.0
2013/03/26 13:23:20 OSPF: DR-Election[1st]: DR     1.0.0.26
2013/03/26 13:23:20 OSPF: interface 1.0.0.26 [298] leave AllSPFRouters Multicast group.
2013/03/26 13:23:20 OSPF: interface 1.0.0.26 [298] leave AllDRouters Multicast group.
2013/03/26 13:23:20 OSPF: ospf_apiserver_ism_change
2013/03/26 13:23:20 OSPF: oi->ifp->name=ucla-eth6
2013/03/26 13:23:20 OSPF: old_state=7
2013/03/26 13:23:20 OSPF: oi->state=1
2013/03/26 13:23:20 OSPF: Router routing table for          0.0.0.2 updated  (0.0.0.2 = CSU)
2013/03/26 13:23:20 OSPF: Router routing table for          0.0.0.3 updated  (0.0.0.3 = NEU)
2013/03/26 13:23:20 OSPF: Router routing table for          0.0.0.7 updated  (0.0.0.7 = SPP-ATLA)
2013/03/26 13:23:20 OSPF: Router routing table for          0.0.0.8 updated  (0.0.0.8 = SPP-HOUS)
2013/03/26 13:23:20 OSPF: Router routing table for         0.0.0.16 updated (0.0.0.16 = UM)
2013/03/26 13:23:20 OSPF: Router routing table for         0.0.0.17 updated (0.0.0.17 = WASHU)
```

(a) *Log* do `ospfd` no nó UCLA

```
...
ccnx:/ndn/colostate.edu/netsec face: 13 flags: 0x3 expires: 2147483207   (face 13 = UA)
ccnx:/ndn/colostate.edu/netsec face: 12 flags: 0x3 expires: 2147483207   (face 12 = SPP-SALT)
ccnx:/ndn/colostate.edu/netsec face: 9 flags: 0x3 expires: 2147483207   (face 9 = PARC)
...
```

(b) FIB de UCLA

Figura 4.17: Comportamento do `ospfd` e da FIB de UCLA após queda do *link*

A primeira mensagem observada no *log* do `ospfd` indica a detecção da remoção de CSU e o respectivo envio de um LSA de atualização para a rede. Em seguida, a interface **ucla-eth6** (com endereço IP 1.0.0.26 no túnel criado pelo Mini-CCNx) é removida dos grupos de *multicast* de cálculo de menor caminho para que as novas rotas possam ser agora definidas. Finalmente, todas as entradas afetadas pela alteração de topologia são atualizadas na tabela de roteamento OSPF (os identificadores 0.0.0.2, 0.0.0.3, ... são escolhidos arbitrariamente para a identificação dos roteadores da rede - essa é uma exigência do *daemon* `ospfd`). Nota-se que toda essa operação foi feita dentro de um segundo (13:23:20)[6]. Ao se observar os *logs* (não apresentados aqui) dos outros roteadores da topologia, nota-se que todas as atualizações também foram realizadas no mesmo segundo e, assim, pode-se concluir que a resposta do roteamento à alteração na rede foi muito eficiente.

O OSPFN, por sua vez, detecta imediatamente tais atualizações na conectividade e reinsere entradas na FIB para os prefixos afetados utilizando as novas informações fornecidas pelo `ospfd`. A Figura 4.17(b) mostra a visão parcial da FIB (tal qual obtida pelo comando `ccndstatus`), que agora apresenta três novos caminhos de 2 *hops* entre UCLA e CSU - nominalmente (UCLA)-(PARC)-(CSU), (UCLA)-(SPP-SALT)-(CSU) e (UCLA)-(UA)-(CSU). Infelizmente, a implementação atual do OSPFN não faz o *log* de tais atualizações, porém a inspeção rápida com o comando interativo `ccndstatus` permite ter uma noção de resposta ágil de atualização da FIB ante as alterações sinalizadas pelo `ospfd`.

Agora, com o comando "`link ucla csu up`", vamos observar o comportamento do roteamento quando o *link* original entre UCLA e CSU é reconectado no tempo 13:24:00. A Figura 4.18 mostra o *log* do `ospfd` e a visão parcial da FIB de UCLA após a reconexão do *link*.

Nota-se que o `ospfd` percebe a reconexão pela primeira entrada do *log* (novamente, pela identificação 1.0.0.26 da interface **ucla-eth6**) logo no tempo 13:24:00. Em seguida, a operação do *daemon* segue até o tempo 13:24:45, no qual finalmente o LSA de atualização é enviado para a rede e a tabela do nó é atualizada. Nesse mesmo momento, observa-se que a FIB de UCLA é imediatamente atualizada, na qual pode-se notar o retorno da entrada "/ndn/colostate.edu/netsec/" apontando diretamente para CSU (em destaque na Figura 4.18(b)), tal qual era o estado original. Portanto, a recuperação total levou agora 45 segundos, tempo esse utilizado pela convergência do roteamento OSPF, segundo a implementação do `ospfd`[7] - o OSPFN insere imediatamente as entradas nas FIBs dos nós uma vez que a tabela de roteamento OSPF é atualizada.

Durante todo esse experimento, um gerador de tráfego (`ccntraffic`) em UCLA gerou um tráfego leve e constante com direção a CSU, que respondia com conteúdos também arbitrários de 1024 bytes (utilizando a ferramenta `ccndelphi`). A banda ao longo do tempo foi medida em 4 pontos ligados diretamente a CSU: *(i)* PARC-CSU, *(ii)* UCLA-CSU, *(iii)* SPP-SALT-CSU e *(iv)* UA-CSU. A Figura 4.19 mostra o comportamento da banda com marcas de tempo no eixo x referentes aos eventos citados anteriormente nesse experimento.

Nota-se que em 13:22:56 o gerador de tráfego foi ligado e atingiu um valor médio de aproximadamente 80 Kbps utilizando o *link* direto entre CSU e UCLA. Em 13:23:20, quando o *link* é removido, nota-se a resposta imediata do plano de encaminhamento, que opta por utilizar

[6] O *log* do `ospfd` apresenta granularidade máxima de 1 segundo e, portanto, não é possível analisar com mais detalhes o tempo de cada uma das operações.

[7] Configurações padrão do Quagga foram utilizadas.

```
2013/03/26 13:24:00 OSPF: interface 1.0.0.26 [298] join AllSPFRouters Multicast group.
2013/03/26 13:24:00 OSPF: ospf_apiserver_ism_change
2013/03/26 13:24:00 OSPF: oi->ifp->name=ucla-eth6
2013/03/26 13:24:00 OSPF: old_state=1
2013/03/26 13:24:00 OSPF: oi->state=3
2013/03/26 13:24:00 OSPF: ospf_apiserver_nsm_change
2013/03/26 13:24:10 OSPF: ospf_apiserver_nsm_change
2013/03/26 13:24:40 OSPF: DR-Election[1st]: Backup 1.0.0.26
2013/03/26 13:24:40 OSPF: DR-Election[1st]: DR    1.0.0.26
2013/03/26 13:24:40 OSPF: DR-Election[2nd]: Backup 1.0.0.25
2013/03/26 13:24:40 OSPF: DR-Election[2nd]: DR    1.0.0.26
2013/03/26 13:24:40 OSPF: interface 1.0.0.26 [298] join AllDRouters Multicast group.
2013/03/26 13:24:40 OSPF: ospf_apiserver_ism_change
2013/03/26 13:24:40 OSPF: oi->ifp->name=ucla-eth6
2013/03/26 13:24:40 OSPF: old_state=3
2013/03/26 13:24:40 OSPF: oi->state=7
2013/03/26 13:24:40 OSPF: ospf_apiserver_nsm_change
2013/03/26 13:24:40 OSPF: Packet[DD]: Neighbor 0.0.0.2: Initial DBD from Slave, ignoring.
2013/03/26 13:24:40 OSPF: DR-Election[1st]: Backup 1.0.0.25
2013/03/26 13:24:40 OSPF: DR-Election[1st]: DR    1.0.0.26
2013/03/26 13:24:45 OSPF: Packet[DD]: Neighbor 0.0.0.2 Negotiation done (Master).
2013/03/26 13:24:45 OSPF: ospf_apiserver_nsm_change
2013/03/26 13:24:45 OSPF: nsm_change_state(0.0.0.2, Exchange -> Full): scheduling new router-LSA
origination
2013/03/26 13:24:45 OSPF: ospf_apiserver_nsm_change
2013/03/26 13:24:45 OSPF: Router routing table for          0.0.0.2 updated
2013/03/26 13:24:45 OSPF: Router routing table for          0.0.0.3 updated
2013/03/26 13:24:45 OSPF: Router routing table for          0.0.0.7 updated
2013/03/26 13:24:45 OSPF: Router routing table for          0.0.0.8 updated
2013/03/26 13:24:45 OSPF: Router routing table for          0.0.0.16 updated
2013/03/26 13:24:45 OSPF: Router routing table for          0.0.0.17 updated
```

(a) *Log* do `ospfd` no nó UCLA

```
...
ccnx:/ndn/colostate.edu/netsec face: 8 flags: 0x3 expires: 214748301   (face 8 = CSU)
ccnx:/ndn/pku.edu.cn face: 10 flags: 0x3 expires: 2147483012   (face 10 = PKU)
...
```

(b) FIB de UCLA

Figura 4.18: Comportamento do `ospfd` e da FIB de UCLA após reconexão do *link*

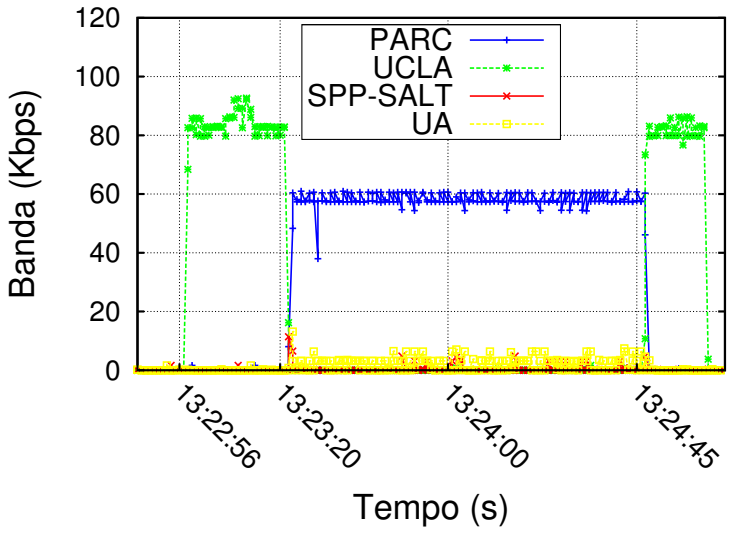


Figura 4.19: Comportamento da banda nos diferentes *links* relevantes ao experimento

o caminho que passa pelo roteador PARC, o que resulta numa degradação do valor médio de banda observado, com uma queda para cerca de 60 Kbps. Nota-se que esse é o pior dos três *links* disponíveis no momento caso o critério for o atraso de propagação total. Porém, mesmo assim, o plano de encaminhamento toma a opção por esse caminho, provavelmente segundo outros critérios adotados na implementação do ccnd. Essa resposta imediata do plano de caminhamento é, porém, compatível com a observação anterior de que quando o *link* é removido, as FIBs dos nós são atualizadas no mesmo segundo. Durante esse tempo, nota-se algumas tentativas de prospecção dos outros *links* (pequenos picos próximos a 5 Kbps), mas o plano de encaminhamento claramente escolheu o caminho via PARC por todo o tempo. Em 13:24:00, o *link* entre CSU e UCLA é reconectado mas agora esse *link* não é imediatamente utilizado. Isso também é compatível com a observação de que a FIB de UCLA só é de fato atualizada com a informação da reconexão do *link* em 13:24:45 - tempo no qual pode-se notar claramente que a opção pelo *link* direto entre CSU e UCLA é retomada.

4.3 Atendimento aos Objetivos

Os experimentos realizados nesse capítulo ajudam a avaliar quão bem atendidos foram os objetivos delineados no primeiro capítulo dessa dissertação. Como previsto, o emulador não consegue atingir a escalabilidade de um simulador, porém experimentos interessantes com algumas dezenas de nós foram realizados e, portanto, pode-se considerar que o Mini-CCNx atinge um bom nível de escalabilidade. Os mais diversos experimentos (*caching*, vídeo, roteamento, distribuição de conteúdos, etc) comprovam que o Mini-CCNx atingiu o seu objetivo de flexibilidade. O objetivo de baixo custo também pode ser considerado atingido, já que todos os experimentos realizados nesse capítulo utilizaram um *hardware* convencional para sua execução. O realismo, tal qual definido no primeiro capítulo, também foi atendido. Todos os códigos executados são códigos reais e a reprodução de experimentos da literatura também comprovam esse atendimento. Por fim, a facilidade de uso é o objetivo mais difícil de ser analisado devido a sua subjetividade. Porém, pode-se considerar que ele também foi atingido devido à criação da ferramenta gráfica `miniccnxedit`, dos criadores automáticos de topologia e também pelo fato de que a definição de cenários não é tão onerosa.

O próximo capítulo encerra essa dissertação com a conclusão e trabalhos futuros.

Capítulo 5

Conclusões e Trabalhos Futuros

As Redes Orientadas a Conteúdo (ROCs) apresentam um paradigma novo e promissor para a Internet do futuro pois se baseiam na nova tendência de utilização da rede (foco na obtenção de conteúdos e acesso a serviços) e também tentam atacar alguns dos desafios que o modelo TCP/IP enfrenta diante da dimensão que a Internet tomou ao longo das quatro ou cinco décadas desde a criação de tal modelo. Atualmente, o modelo *Content Centric Networking* (CCN) se apresenta como a proposta mais madura de ROC, com a especificação e implementação mais madura dentre todas as propostas de ROC; também pode-se considerar tal proposta como a mais rica quanto à quantidade de análises feitas, protocolos propostos e aplicações implementadas pela literatura recente.

Apesar de promissor, o modelo CCN (e as ROCs num geral) ainda precisa endereçar vários desafios, como escalabilidade, roteamento baseado em nomes, mobilidade, dentre vários outros. Também por ser uma proposta complexa e bem recente, não se encontra uma grande quantidade de plataformas experimentais na área que possam auxiliar os pesquisadores a avaliar seus protocolos e aplicações para o modelo.

A partir dessa necessidade, identificou-se uma lacuna entre as plataformas experimentais disponíveis atualmente na área e, assim, essa dissertação apresenta o Mini-CCNx, um novo emulador desenvolvido especificamente para a avaliação de cenários de ROCs baseadas no modelo CCN. O Mini-CCNx pode ser utilizado como plataforma de desenvolvimento e testes por toda a comunidade científica interessada no desenvolvimento desse modelo. O Mini-CCNx é publicado com seu código aberto, documentação e tutoriais em `https://github.com/carlosmscabral/mn-ccnx/wiki` e pode, portanto, ser melhorado e estendido por qualquer pesquisador da área. Esse emulador apresenta uma combinação única de características como flexibilidade, realismo, facilidade de uso, baixo custo e com média escalabilidade, características essas que tornam o Mini-CCNx o primeiro emulador genérico publicamente disponível para a avaliação de novas propostas relacionadas ao modelo CCN.

Além de seu principal objetivo, de suportar o desenvolvimento e testes de novas propostas para o modelo CCN, identificou-se também outro importante uso para o Mini-CCNx. Principalmente por possuir baixo custo, o emulador pode ser também utilizado em atividades de ensino como uma ferramenta prática de aprendizado e introdução às ROCs e ao modelo CCN.

5.1 Trabalhos Futuros

Essa seção apresentará alguns trabalhos futuros que podem ser realizados não somente para melhorar o Mini-CCNx em si, mas também algumas idéias de projetos de pesquisa que podem utilizar o emulador como base para a sua fase de desenvolvimento e avaliação.

5.1.1 Melhorias para a Ferramenta

Essa subseção apresenta propostas para trabalhos que possam melhorar e estender a ferramenta desenvolvida.

Execução Completa através da Interface Gráfica

Uma possível extensão para o Mini-CCNx seria incrementar a ferramenta gráfica `miniccnxedit`, que atualmente é utilizada somente para a criação de *templates* de arquivo de configuração, para suportar o ambiente completo de execução do Mini-CCNx. Atualmente o ambiente de execução é acessado principalmente por linha de comando e por terminais remotos abertos para cada nó, como num acesso comum a *hosts* Linux. Eventualmente, a partir de tais terminais, pode-se iniciar a execução de vídeos ou outras aplicações gráficas, conforme o desejado. A idéia seria possuir um ambiente gráfico exibindo a topologia e que, dinamicamente, permitisse que o usuário acessasse informações individuais de cada nó, como inspecionar sua tabela de roteamento, sua PIT, o *log* de execução, etc.

Após criar a topologia, o usuário poderia, por exemplo, clicar em um nó CCN, o que abriria uma janela gráfica para a inserção de entradas na FIB e configuração dos limites de CPU. Concluída essa pré-configuração, o ambiente entraria em execução. O usuário poderia então iniciar algum algoritmo de roteamento, inspecionar as atualizações nas tabelas (provavelmente com algum botão ou menu ao lado de cada nó), iniciar alguma aplicação ou mesmo inspecionar os pacotes com *ndndump*.

Num geral, essa seria uma grande evolução na usabilidade da ferramenta. Obviamente, seria necessário fazer uma análise para escolher um *framework* gráfico (e idealmente baseado em Python, para uma melhor interação com o Mini-CCNx) leve o suficiente para não trazer grandes impactos ao consumo de recursos da máquina executando o Mini-CCNx.

Métricas CCN

Outra possível melhoria para o Mini-CCNx seria implementar um sistema integrado de métricas CCN. Por exemplo, ao fim da execução de um cenário de testes, o emulador poderia calcular (provavelmente via análise dos *logs* `ccnd`) o número de *Interests* e *Content Objects* enviados (globalmente e por nó), RTT médio observado em cada *link*, taxa média de ocupação dos *caches* (globalmente e por nó), dentre outras. O intuito seria identificar quais métricas são importantes para uma grande parte de experimentos CCN e fazer esse cálculo ao fim de cada execução, facilitando a análise final do usuário do emulador.

Análise de *Stress*

O Mini-CCNx poderia ser também avaliado em termos de quanto recursos (CPU, memória) são utilizados conforme a carga de utilização de banda (utilizando, por exemplo, o `ccntraffic`) varia. Pode-se analisar também a forma com que o isolamento fica comprometido durante essa variação. Uma análise desse tipo ajudaria a entender um pouco melhor outras métricas de escalabilidade do Mini-CCNx não analisadas até então nessa dissertação.

5.1.2 Linhas de Pesquisas Utilizando a Ferramenta

Essa subseção apresenta alguns esboços trabalhos de pesquisa que podem utilizar o Mini-CCNx como plataforma de testes e desenvolvimento.

Redes de Sensores Sem-Fio e o Modelo CCN

Para avaliar o comportamento do modelo CCN em uma rede de sensores de sem-fio seria necessário estender o Mini-CCNx para emular um meio de *broadcast*. Uma maneira de se fazer isso seria utilizar o *software switch* Open vSwitch (Open vSwitch 2013) em modo *hub* fazendo com que todo tráfego recebido em uma interface do *switch* seja reproduzido nas demais interfaces. O Mininet já possui uma integração com o Open vSwitch. Assim, seria necessário apenas adaptar as classes que criam a conectividade CCN para a utilização do *software switch*. Cada sensor poderia ser representado por um nó Mini-CCNx com recursos bem limitados e algum servidor central da rede poderia ser representado por um nó com mais recursos. Alguma implementação adicional seria necessária para introduzir o conceito de distância entre os nós, já que a potência do sinal cai conforme a distância aumenta resultando em uma maior taxa de perda de pacotes nesses casos.

Avaliação de Protocolos de Roteamento

Protocolos de roteamento, em geral, não consomem grande quantidade de banda nem são intensivos em processamento. Devido a esse fato, o Mini-CCNx pode ser uma boa plataforma para avaliação de protocolos de roteamento pois essa menor demanda de processamento pode favorecer a escalabilidade dos cenários a serem utilizados. Uma maior quantidade de nós, que geralmente é necessária para a avaliação de roteamento, poderá ser utilizada combinada ao realismo trazido pela emulação do Mini-CCNx.

Encaminhamento e *Caching*

Como citado anteriormente, a parte principal do modelo CCN (encaminhamento e *caching*) é implementada pelo *daemon* `ccnd`. Praticamente todo o código desse *daemon* está implementado no arquivo **ccnd.c** e em seu respectivo arquivo de cabeçalhos, **ccnd.h**. Portanto, em tais arquivos, pode-se implementar mudanças em alguns pontos centrais do modelo CCN. Por exemplo, alterando somente a função `process_incoming_interest` no arquivo **ccnd.c**, é possível propor uma nova maneira (talvez mais eficiente) de se tratar o recebimento de pacotes *Interest* que entram no nó.

Como explicado no capítulo 3 desse texto, o Mini-CCNx faz interface com os *daemons* CCNx. Se tais pontos de interface não forem alterados, o Mini-CCNx também não precisa ser alterado caso o pesquisador utilize uma nova versão do CCNx. Assim, voltando ao exemplo anterior, um pesquisador pode utilizar o Mini-CCNx, sem alterações, para comparar o desempenho de sua nova proposta de tratamento de *Interests* com relação à implementação original. Nesse caso, será necessário recompilar somente o CCNx (devido à alteração no arquivo **ccnd.c**) mas não o Mini-CCNx, já que a interface com o CCNx não foi alterada. Utilizando exatamente a mesma topologia e cenário Mini-CCNx, o pesquisador poderá assim avaliar qual das propostas apresenta o melhor desempenho segundo as métricas desejadas.

Bibliografia

ACM Awards (2001). SIGCOMM Award Recipients.
URL: *http://www.sigcomm.org/awards/sigcomm-awards*

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

Akamai (2013). CDN Provider. Acesso: Abr/2013.
URL: *http://http://www.akamai.com/*

Alvarez, A., Orea, R., Cabrero, S., Pañeda, X. G., García, R. & Melendi, D. (2010). Limitations of network emulation with single-machine and distributed ns-3, *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, ICST, p. 67.
URL: *http://dl.acm.org/citation.cfm?id=1808143.1808228*

Ambiel, L., Rothenberg, C. E. & Magalhães, M. (2013). Redes orientadas a conteúdo: Abordagem no nível de enlace, *SBRC 2013*.

Apache Software Foundation, (2013). Apache. Acesso: Abr/2013.
URL: *http://httpd.apache.org/*

Barbaroux, P. (2012). Identifying collaborative innovation capabilities within knowledge-intensive environments: Insights from the ARPANET project, *European Journal of Innovation Management* **15**(2): 232–258.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. & Warfield, A. (2003). Xen and the art of virtualization, *Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP '03*, Vol. 37, ACM Press, New York, New York, USA, p. 164.
URL: *http://dl.acm.org/citation.cfm?id=945445.945462*

Berger, L. & Bryskin, I. (2008). RFC 5250 - The OSPF Opaque LSA Option.
URL: *http://www.ietf.org/rfc/rfc5250.txt*

Broder, A. & Mitzenmacher, M. (2001). Using multiple hash functions to improve ip lookups, *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, Vol. 3, IEEE, pp. 1454–1463.

Burke, J., Horn, A. & Marianantoni, A. (2012). Authenticated Lighting Control Using Named Data Networking.
URL: *http://www.named-data.net/techreport/TR011-lighting.pdf*

Cabral, C., Esteve Rothenberg, C. & Magalhaes, M. F. (2013a). Mini-CCNx: fast prototyping for named data networking, *The 3rd ACM SIGCOMM Workshop on Information-Centric Networking (ICN 2013) (ICN 2013)*, Hong Kong, Hong Kong.

Cabral, C., Esteve Rothenberg, C. & Magalhaes, M. F. (2013b). Reproducing real NDN experiments using Mini-CCNx, *The 3rd ACM SIGCOMM Workshop on Information-Centric Networking (ICN 2013) (ICN 2013)*, Hong Kong, Hong Kong.

Cabral, C., Rothenberg, C. E. & Magalhães, M. (2013). Mini-CCNx: prototipagem rápida para Redes Orientadas a Conteúdo baseadas em CCN, *SBRC 2013 - Salão de Ferramentas*, Brasília-DF.

Carofiglio, G., Gallo, M., Muscariello, L. & Perino, D. (2011). Modeling data transfer in content-centric networking, pp. 111–118.
URL: *http://dl.acm.org/citation.cfm?id=2043468.2043487*

Carson, M. & Santay, D. (2003). NIST Net, *ACM SIGCOMM Computer Communication Review* **33**(3): 111.
URL: *http://dl.acm.org/citation.cfm?id=956993.957007*

CCN Ping (2013). NDN-Routing/ccnping - GitHub. Acesso: Jan/2013.
URL: *https://github.com/NDN-Routing/ccnping*

CCNx (2013). Official implementation of the CCN model. Acesso: Fev/2013.
URL: *https://www.ccnx.org/*

CCNx Traffic (2013). CCNx: traffic generation - ARL ONL Wiki. Acesso: Jan/2013.
URL: *http://wiki.arl.wustl.edu/onl/index.php/CCNx:_traffic_generation*

cgroups (2012). Linux Control Groups. Acesso: Out/2012.
URL: *https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt*

Chiocchetti, R., Rossi, D., Carofiglio, G., Lucent, A. & Labs, B. (2012). Exploit the Known or Explore the Unknown ? Hamlet-Like Doubts in ICN, *ACM SIGCOMM, ICN Workshop* pp. 7–12.

Chong, E. K. P. (1994). Discrete event systems: Modeling and performance analysis, *Discrete Event Dynamic Systems: Theory and Applications* **4**(1): 113–116.
URL: *http://link.springer.com/10.1007/BF01516012*

Cianci, I., Grieco, L. A. & Boggia, G. (2012). CCN - Java opensource kit EmulatoR for wireless ad hoc networks, *Proceedings of the 7th International Conference on Future Internet Technologies - CFI '12*, ACM Press, New York, New York, USA, p. 7.
URL: *http://dl.acm.org/citation.cfm?id=2377310.2377313*

Cisco (2012). Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012-2017.
URL: *http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html*

Dai, H., Liu, B., Chen, Y. & Wang, Y. (2012). On pending interest table in named data networking, *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*, ANCS '12, ACM, New York, NY, USA, pp. 211–222.
URL: *http://doi.acm.org/10.1145/2396556.2396600*

Dannewitz, C., Kutscher, D., Ohlman, B., Farrell, S., Ahlgren, B. & Karl, H. (2013). Network of Information (NetInf) - An Information-Centric Networking Architecture, *Computer Communications* .
URL: *http://dx.doi.org/10.1016/j.comcom.2013.01.009*

de Brito, G. M., Velloso, P. B. & Moraes, I. M. (2012). Redes Orientadas a Conteúdo: Um Novo Paradigma para a Internet, *SBRC 2012* .
URL: *http://sbrc2012.dcc.ufmg.br/app/p-04-g.html*

Emulab (2013). Network Emulation Testbed. Acesso: Fev/2013.
URL: *http://emulab.net/*

FITS (2013). Future Internet Testbed with Security. Acesso: Abr/2013.
URL: *http://www.gta.ufrj.br/fits*

GENI (2013). Global Environment for Network Innovations. Acesso: Fev/2013.
URL: *http://www.geni.net/*

Ghodsi, A., Shenker, S. & Berkeley, U. C. (2011). Naming in Content-Oriented Architectures, pp. 1–6.

GitHub (2013). GitHub. Acesso: Abr/2013.
URL: *https://github.com*

Google (2013). Google Earth. Acesso: Fev/2013.
URL: *http://www.google.com/earth/index.html*

Gurtov, A. (2008). *Host Identity Protocol (HIP): Towards the Secure Mobile Internet*, Wiley Publishing.

Handigol, N., Heller, B., Jeyakumar, V., Lantz, B. & McKeown, N. (2012). Reproducible network experiments using container-based emulation, *Proceedings of the 8th international conference on Emerging networking experiments and technologies - CoNEXT '12* p. 253.
URL: *http://dl.acm.org/citation.cfm?doid=2413176.2413206*

Hibler, M., Ricci, R., Stoller, L., Duerig, J., Guruprasad, S., Stack, T., Webb, K. & Lepreau, J. (2008). Large-scale virtualization in the Emulab network testbed, pp. 113–128.
URL: *http://dl.acm.org/citation.cfm?id=1404014.1404023*

Intel (2007). All about System Power States (S0-S5). Acesso: Mai/2013.
URL: *http://software.intel.com/en-us/blogs/2007/01/10/all-about-system-power-states-s0-s5*

Jacobson, V. (1988). Congestion avoidance and control, *ACM SIGCOMM Computer Communication Review* **18**(4): 314–329.
URL: *http://dl.acm.org/citation.cfm?id=52325.52356*

Jacobson, V., Smetters, D. K., Briggs, N. H., Plass, M. F., Stewart, P., Thornton, J. D. & Braynard, R. L. (2009). Voccn: voice-over content-centric networks, *Proceedings of the 2009 workshop on Re-architecting the internet*, ReArch '09, ACM, New York, NY, USA, pp. 1–6.
URL: *http://doi.acm.org/10.1145/1658978.1658980*

Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H. & Braynard, R. L. (2009). Networking named content, *Proceedings of the 5th international conference on Emerging networking experiments and technologies - CoNEXT '09*, ACM Press, New York, New York, USA, p. 1.
URL: *http://dl.acm.org/citation.cfm?id=1658939.1658941*

Jokela, P., Zahemszky, A., Esteve Rothenberg, C., Arianfar, S. & Nikander, P. (2009). Lipsin: line speed publish/subscribe inter-networking, *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, ACM, New York, NY, USA, pp. 195–206.
URL: *http://doi.acm.org/10.1145/1592568.1592592*

Koponen, T., Chawla, M., Chun, B.-G., Ermolinskiy, A., Kim, K. H., Shenker, S. & Stoica, I. (2007). A data-oriented (and beyond) network architecture, *ACM SIGCOMM Computer Communication Review* **37**(4): 181.
URL: *http://dl.acm.org/citation.cfm?id=1282427.1282402*

Kulinsk, D. & Burke, J. (2012). NDNVideo: Random-access Live and Pre-recorded Streaming using NDN.
URL: *http://www.named-data.net/techreport/TR007-streaming.pdf*

Lantz, B., Heller, B. & McKeown, N. (2010). A network in a laptop, *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10*, ACM Press, New York, New York, USA, pp. 1–6.
URL: *http://dl.acm.org/citation.cfm?id=1868447.1868466*

Lichstein, H. A. (1969). When should you emulate, *Datamation* **15**(11): 205–210.

lighttpd (2013). Lighttpd. Acesso: Abr/2013.
URL: *http://www.lighttpd.net/*

Linux Foundation (2013). Network Emulation. Acesso: Fev/2013.
URL: *http://www.linuxfoundation.org/collaborate/workgroups/networking/netem*

LTTng (2013). Linux Trace Tool Project- Next Generation. Acesso: Fev/2013.
URL: *https://lttng.org/*

LXC (2012). Linux Containers. Acesso: Dez/2012.
URL: *http://lxc.sourceforge.net/*

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. & Turner, J. (2008). Openflow: enabling innovation in campus networks, *SIGCOMM Comput. Commun. Rev.* **38**(2): 69–74.
URL: *http://doi.acm.org/10.1145/1355734.1355746*

Mealling, M. & Denenberg, R. (2012). RFC 3305 - Report from the Joint W3C/IETF URI Planning Interest Group: Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations.
URL: *http://www.ietf.org/rfc/rfc3305.txt*

Mills, D. L. & Braun, H. (1988). The NSFNET backbone network, *Proceedings of the ACM workshop on Frontiers in computer communications technology - SIGCOMM '87*, Vol. 17, ACM Press, New York, New York, USA, pp. 191–196.
URL: *http://dl.acm.org/citation.cfm?id=55482.55502*

Moy, J. (1998). RFC 2328 - OSPF Version 2.
URL: *http://www.ietf.org/rfc/rfc2328.txt*

NDN Project (2012a). Named Data Networking. Acesso: Dez/2012.
URL: *http://www.named-data.net/*

NDN Project (2012b). NDN Technical Reports. Acesso: Dez/2012.
URL: *http://www.named-data.net/techreports.html*

NDN Testbed (2013). NDN Routing Topology. Acesso: Mar/2013.
URL: *http://netlab.cs.memphis.edu/script/htm/topology.html*

ndnSIM (2013). NS-3 based NDN simulator. Acesso: Mar/2013.
URL: *http://ndnsim.net/*

ns-3 (2013). Network Simulator. Acesso: Fev/2013.
URL: *http://www.nsnam.org/*

OMNeT++ (2013). Network Simulation Framework. Acesso: Fev/2013.
URL: *http://www.omnetpp.org/*

Open vSwitch (2013). Production Quality, Multilayer Open Virtual Switch. Acesso: Mai/2013.
URL: *http://openvswitch.org/*

Oracle (2013). Oracle VirtualBox. Acesso: Abr/2013.
URL: *https://www.virtualbox.org/*

OSPFN (2012). OSPF for Named-data.
URL: *http://www.named-data.net/techreport/TR003-OSPFN.pdf*

Perino, D. & Varvello, M. (2011). A reality check for content centric networking, *Proceedings of the ACM SIGCOMM workshop on ICN*, ACM Press, pp. 44 –49.

Plagemann, T., Goebel, V., Mauthe, A., Mathy, L., Turletti, T. & Urvoy-Keller, G. (2006). From content distribution networks to content networks - issues and challenges, *Computer Communications* **29**(5): 551–562.
URL: *http://dx.doi.org/10.1016/j.comcom.2005.06.006*

PlanetLab (2013). An open platform for developing, deploying, and accessing planetary-scale services. Acesso: Fev/2013.
URL: *http://www.planet-lab.org/*

Pouwelse, J., Garbacki, P., Epema, D. & Sips, H. (2005). The bittorrent p2p file-sharing system: Measurements and analysis, *in* M. Castro & R. Renesse (eds), *Peer-to-Peer Systems IV*, Vol. 3640 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 205–216.

Prpic, M., Landmann, R. & Silas, D. (2013). Red Hat Enterprise Linux 6 - Resource Management Guide - Edition 4. Acesso: Abr/2013.
URL: *https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide*

QEMU (2013). Open source machine emulator and virtualizer. Acesso: Jan/2013.
URL: *http://wiki.qemu.org/Main_Page*

Quagga (2013). Quagga Routing Suite. Acesso: Mar/2013.
URL: *http://www.nongnu.org/quagga/*

Rossi, D. & Rossini, G. (2011). Caching performance of content centric networks under multi-path routing (and more), *Telecom ParisTech* .
URL: *http://perso.telecom-paristech.fr/ drossi/index.php?n=Software.ccnSim*

Rossi, D. & Rossini, G. (2012). On sizing CCN content stores by exploiting topological information, *2012 Proceedings IEEE INFOCOM Workshops* pp. 280–285.
URL: *http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6193506*

Rossini, G. & Rossi, D. (2012). A dive into the caching performance of content centric networking, *Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2012 IEEE 17th International Workshop on*, pp. 105–109.

Smetters, D. & Jacobson, V. (2009). Securing Network Content, *PARC Technical Report* pp. 1–7.

Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A. & Peterson, L. (2007). Container-based operating system virtualization, *ACM SIGOPS Operating Systems Review* **41**(3): 275.
URL: *http://dl.acm.org/citation.cfm?id=1272998.1273025*

tc (2013). Linux Advanced Routing and Traffic Control. Acesso: Fev/2013.
URL: *http://lartc.org/*

tcpdump (2013). Command-line packet analyzer. Acesso: Jan/2013.
URL: *http://www.tcpdump.org/*

TkInter (2013). Python's de-facto standard GUI. Acesso: Fev/2013.
URL: *http://wiki.python.org/moin/TkInter*

VMware (2013a). Virtualization and Cloud Solutions.
URL: *http://www.vmware.com/*

VMware (2013b). VMware Workstation. Acesso: Abr/2013.
URL: *http://www.vmware.com/br/products/desktop_virtualization/workstation/overview.html*

Wireshark (2013). Packet Sniffer. Acesso: Jan/2013.
URL: *http://www.wireshark.org/*

Yi, C., Afanasyev, A., Wang, L., Zhang, B. & Zhang, L. (2012). Adaptive forwarding in named data networking, *SIGCOMM Comput. Commun. Rev.* **42**(3): 62–67.
URL: *http://doi.acm.org/10.1145/2317307.2317319*

You, W., Mathieu, B., Truong, P., Peltier, J. & Simon, G. (2012). Dipit: A distributed bloom-filter based pit table for ccn nodes, *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, pp. 1–7.

Yuan, H., Song, T. & Crowley, P. (2012). Scalable ndn forwarding: Concepts, issues and principles, *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, pp. 1–9.

Zhang, L., Estrin, D., Bruke, J., Jacobson, V., Thornton, J., Smetters, D., Zhang, B., Tsudik, G., Krioukov, D., Massey, D., Papadopoulos, C., Abdelzaher, T., Wang, L., Crowley, P. & Yeh, E. (2010). NDN Project.
URL: *http://www.named-data.net/techreport/TR001ndn-proj.pdf*

Apêndice A

Publicações

Esse projeto de pesquisa gerou, até o momento, três publicações. Duas publicações, com diferentes focos, foram aceitas no *The Third ACM SIGCOMM Workshop on Information-Centric Networking* (ICN 2013) a ser realizado em agosto de 2013 em Hong Kong. A primeira, sob o título "*Mini-CCNx: Fast Prototyping for Named Data Networking*" (Cabral, Esteve Rothenberg & Magalhaes 2013a), apresenta o estudo feito sobre as ferramentas atualmente existentes na área e explica como o Mini-CCNx preenche uma lacuna existente entre elas. Uma análise das características do Mini-CCNx também é apresentada. A segunda publicação, sob o título "*Reproducing Real NDN Experiments using Mini-CCNx*" (Cabral, Esteve Rothenberg & Magalhaes 2013b), tem um enfoque mais prático e a demonstração mostra como foi possível reproduzir os resultados dos *testbeds* reais utilizando o Mini-CCNx.

Sob o título de "Mini-CCNx: prototipagem rápida para Redes Orientadas a Conteúdo baseadas em CCN" (Cabral, Rothenberg & Magalhães 2013) o trabalho foi apresentado no Salão de Ferramentas do XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC) em maio de 2013 em Brasília. O foco é a demonstração prática da utilização do Mini-CCNx e como ele auxilia no desenvolvimento de aplicações para o modelo CCN em uma abordagem mais introdutória.

Apêndice B

Topologia do *Testbed* NDN

A Figura B.1 apresenta a topologia do *testbed* NDN (NDN Testbed 2013), utilizada nos experimentos sobre roteamento do capítulo 4 desse texto.

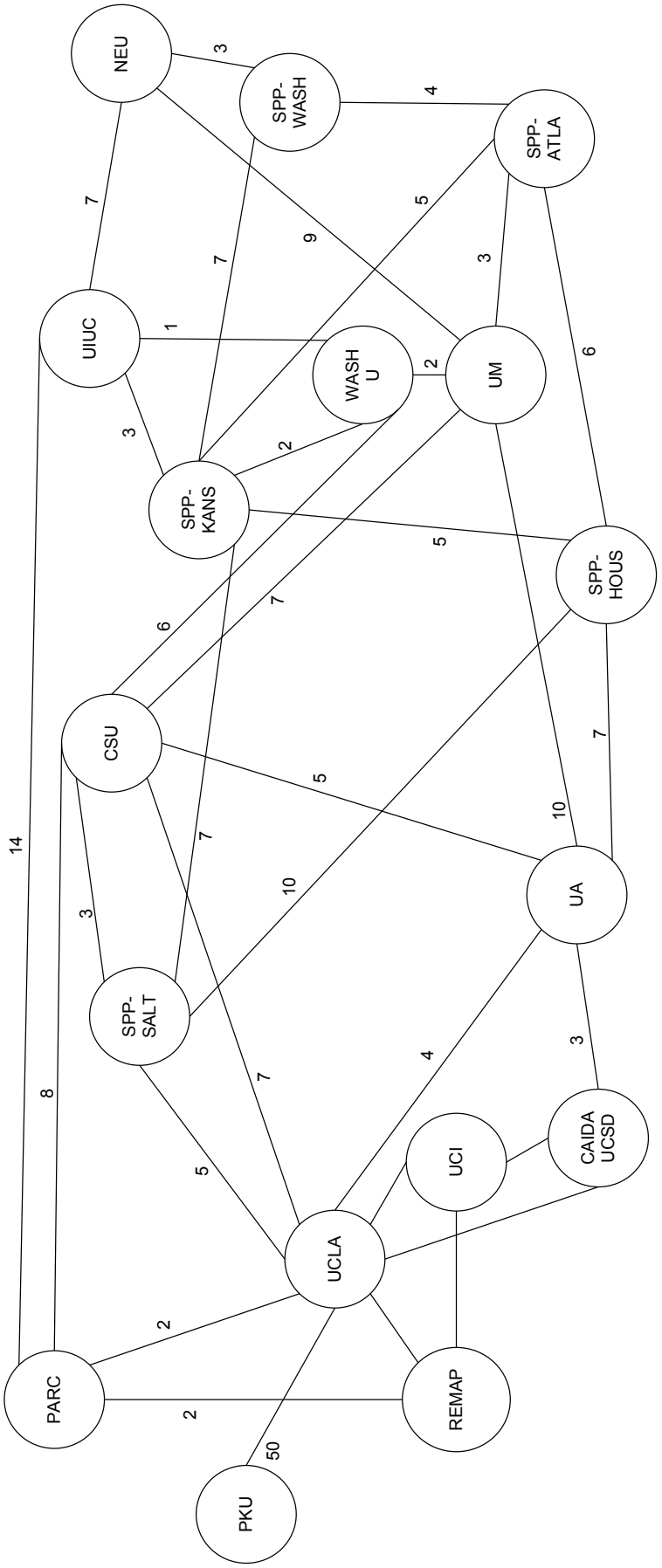


Figura B.1: *Testbed* NDN e atrasos de propagação (em ms). A disposição dos nós está fora de escala geográfica.