UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE ESPECIALIZAÇÃO EM REDES DE COMPUTADORES E TELEINFORMÁTICA

FELIPE BOLSI

ESTUDO DE CASO PRÁTICO SOBRE CONTENT-CENTRIC NETWORK

MONOGRAFIA DE ESPECIALIZAÇÃO EM REDES DE COMPUTADORES E TELEINFORMÁTICA

CURITIBA

2018

FELIPE BOLSI

ESTUDO DE CASO PRÁTICO SOBRE CONTENT-CENTRIC NETWORK

Monografia de Especialização, apresentada ao Curso de Especialização em Redes de Computadores e Teleinformática, do Departamento Acadêmico de Eletrônica - DAELN, da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Especialista.

Orientador: Prof. M.Sc. Juliano de Mello Pedroso

CURITIBA

2018

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Curitiba

Diretoria de Pesquisa e Pós-Graduação
Departamento Acadêmico de Eletrônica
Curso de Especialização em Redes de Computadores e Teleinformática

TERMO DE APROVAÇÃO

ESTUDO DE CASO PRÁTICO SOBRE CONTENT-CENTRIC NETWORK

por

FELIPE BOLSI

Esta monografia foi apresentada em 09 de Julho de 2018 como requisito parcial para a obtenção do título de Especialista em Redes de Computadores e Teleinformática. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. M.Sc. Juliano de Mello Pedroso
Orientador

_____
Prof. Dr. Kleber Kendy Horikawa Nabas
Membro titular

_____
Prof. M.Sc. Omero Francisco Bertol
Membro titular

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

Ao meu querido irmão Gabriel Bolsi, que nos deixou há pouco tempo, mas continua sendo minha maior força e inspiração na vida.

RESUMO

BOLSI, Felipe. **Estudo de Caso Prático sobre Content-Centric Network**. 51 f. Monografia de Especialização em Redes de Computadores e Teleinformática – Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2018.

A medida que a Internet se populariza, a natureza dos serviços ofertados e as demandas dos usuários exigem uma mudança na arquitetura da mesma. Para melhor atender as necessidades atuais, um novo modelo para a troca de dados na Internet é proposto, o *Content Centric Network*, onde a requisição é feita pelo nome do conteúdo, e o relevante na comunicação é *o que* está sendo consumido, e não *onde* esta o conteúdo que se deseja. Para verificar os princípios básicos do modelo *CCN*, é descrito em detalhes como proceder para configurar um ambiente de testes que utiliza um *framework* que implementa a pilha de protocolo *CCN*. Em seguida, utilizando o ambiente de testes apresentado, alguns experimentos são realizados para verificar o comportamento dos elementos da rede na arquitetura *CCN* em uma requisição de conteúdo por nome. Também é documentado alguns problemas e dificuldades enfrentados na utilização do *framework* para viabilizar e executar os testes.

Palavras-chave: CCN. Encaminhamento de Pacotes. Teste Prático.

ABSTRACT

BOLSI, Felipe. **Practical Study Case on Content-Centric Network**. 51 f. Monografia de Especialização em Redes de Computadores e Teleinformática – Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2018.

As the Internet becomes popular, the nature of the services offered and the demands of the users call for a change in its architecture. To better meet current needs, a new model for data exchange on the Internet is proposed, the Content Centric Network, where the request is made by the name of the content, and the relevant one in the communication is *what* is being consumed, and not *where* is the desired content. To verify the basic principles of the CCN model, it is described in detail how to configure a test environment that uses a framework that implements the CCN protocol stack. Then, using the test environment presented, some experimentations are performed to verify the behavior of CCN elements in a content named based request. It also documented some problems and difficulties faced in using the framework to enable and execute the tests.

Keywords: CCN. Packet Forwarding. Practical Test.

LISTA DE FIGURAS

LISTA DE SIGLAS

CCN	*Content Centric Network*
FIB	*Forwarding Information Base*
ICN	*Information-Centric Networking*
IP	*Internet Protocol*
LFU	*Least Frequently Used*
LRU	*Least Recently Used*
PIT	*Pending Interest Table*
TCP	*Transmission Control Protocol*
VPP	*Vector Packet Processing*

SUMÁRIO

1 INTRODUÇÃO

A Internet, desde o seu primórdio no final da década de 60, foi pensada e desenvolvida para atender a um problema central: compartilhamento de recursos. Na época, os recursos de *hardware* eram escassos e caros, e compartilhá-los era fundamental.

Porém, com o passar do tempo, o barateamente dos equipamentos e a popularização da Internet fez surgir diversos serviços e a tornou cada vez mais difundida no cotidiano das pessoas, que passaram a utilizar a rede para diversos fins.

1.1 PROBLEMA

A medida que a Internet se populariza, a natureza dos serviços ofertados e as demandas dos usuários não são totalmente compatíveis com o modo de operação da rede. Nos dias de hoje, uma das principais aplicações que a Internet atende são o *streaming* de áudio e vídeo.

Para melhor atender as necessidades atuais da Internet, um novo modelo para a troca de dados é proposto, o CCN (*Content Centric Network*), apresentando uma mundaça de paradigma, com uma arquitetura de comunicação sobre dados nomeados, onde pacotes são endereçados pelo nome do conteúdo, e não há noção de *hosts*.

O modelo CCN pretende ser bastante abrangente em sua solução, passando pelas camadas de rede e transporte, e também já incorporando segurança como premissa.

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

Esse trabalho tem o objetivo de utilizar um *framework* que implementa um protótipo da pilha de protocolo CCN.

1.2.2 OBJETIVOS ESPECÍFICOS

Para atender ao objetivo geral neste trabalho de conclusão de curso de especialização os seguintes objetivos específicos serão abordados: verificar os princípios básicos do modelo, analisando o comportamento de alguns componentes mais em específico, como o mecanismo de encaminhamento de pacotes por nome do roteador CCN.

1.3 ESTRUTURA DO TRABALHO

Esta monografia de especialização está dividida em 6 (seis) seções. Nesta primeira seção foi introduzido o assunto tema do trabalho e também foram abordados o problema, objetivo geral, objetivos específicos da pesquisa e a estrutura geral do trabalho.

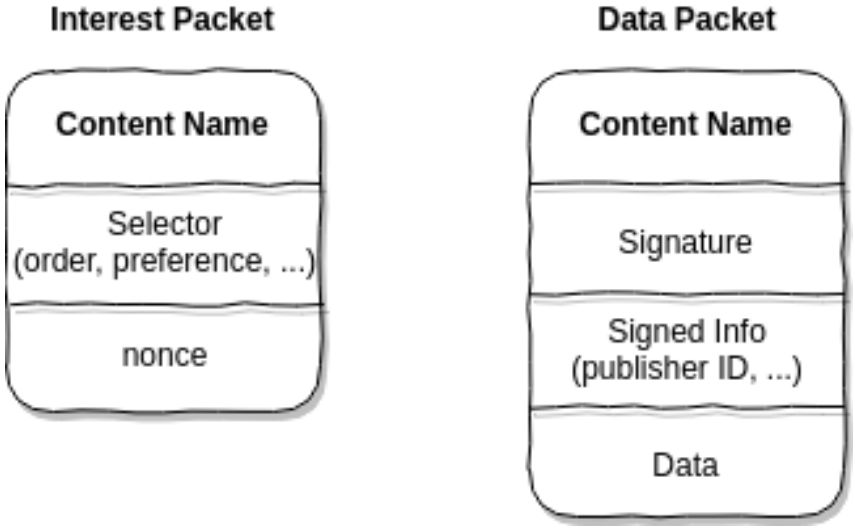
No capítulo 2 é apresentada a arquitetura do modelo CCN. No capítulo 3 é exposto um ambiente de testes para o modelo CCN, e no capítulo 4 são apresentados testes realizados nesse ambiente. O capítulo 5 aponta as dificuldades e erros encontrados. Por fim, o capítulo 6 expõe as conclusões.

2 REFERENCIAL TEÓRICO

CCN (*Content Centric Network*) (JACOBSON VAN, 2009) é uma arquitetura desenvolvida para atender o consumo de conteúdo, valorizando *o que* a Internet disponibiliza, ao contrário do modo de operação de atual, que foca no *onde* esse conteúdo se encontra.

A comunicação CCN é movida pelos consumidores de dados, que manifestam o interesse por determinado conteúdo, e são atendidos com os dados referente a esse conteúdo. Essa manifestação de interesse e dado representam os dois pacotes CCN: *interest packet* e *data packet* como pode ser observado na Figura 1.

Figura 1. Pacotes CCN: *Interest* e *Data*



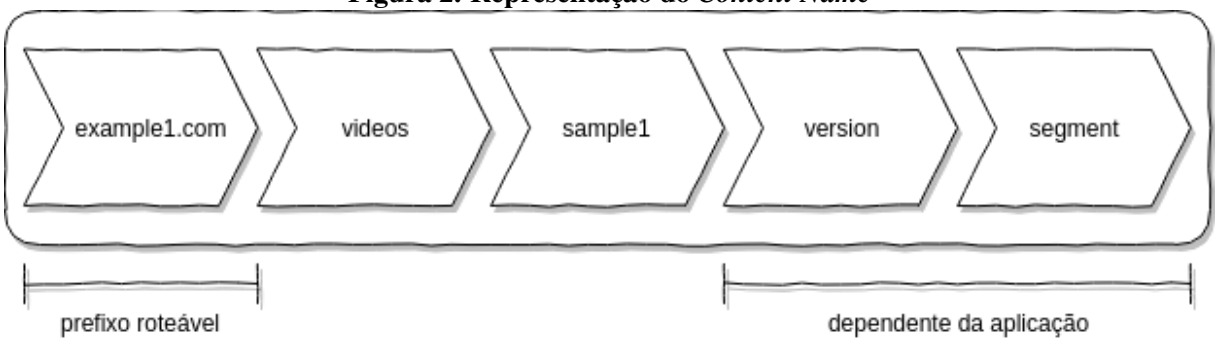
Fonte: Autoria própria.

O modo de operação CCN é bastante direto: um *host* querendo determinado conteúdo manifesta seu interesse enviando o *interest packet* em *broadcast* na rede. Qualquer *host* que receba o pacote e disponha do conteúdo que satisfaça a solicitação responde com o *data packet* correspondente. *Interest* e *data packet* são sempre um para um, não havendo *data packet* enviando sem uma requisição prévia. Diz-se que um dado satisfaz a um interesse se o nome do conteúdo (*Content Name*) no *interest packet* for um prefixo do nome do conteúdo do *data packet*. Os *Content Name* são binários, e com uma representação hierárquica, onde o prefixo é

uma sub-árvore desse nome, e a parte globalmente roteável do nome corresponde a raíz. Essa hierarquia é representada pelos vários componentes que formam o nome, como mostra a Figura 2.

Em uma rede CCN os *data packets* não duplicam, pois o mecanismo da *Content Store* elimina os pacotes duplicados, como será apresentado em detalhes mais adiante. Já os *interest packets* podem sim ser duplicados e ainda causar *loop* na rede. Para evitar esse problema (Figura 2), o *interest packet* tem um campo de *nonce*, que contém um valor aleatório para auxíliar os roteadores CCN a detectar duplicatas e descartar o pacote nesse caso.

Figura 2. Representação do *Content Name*



Fonte: Autoria própria.

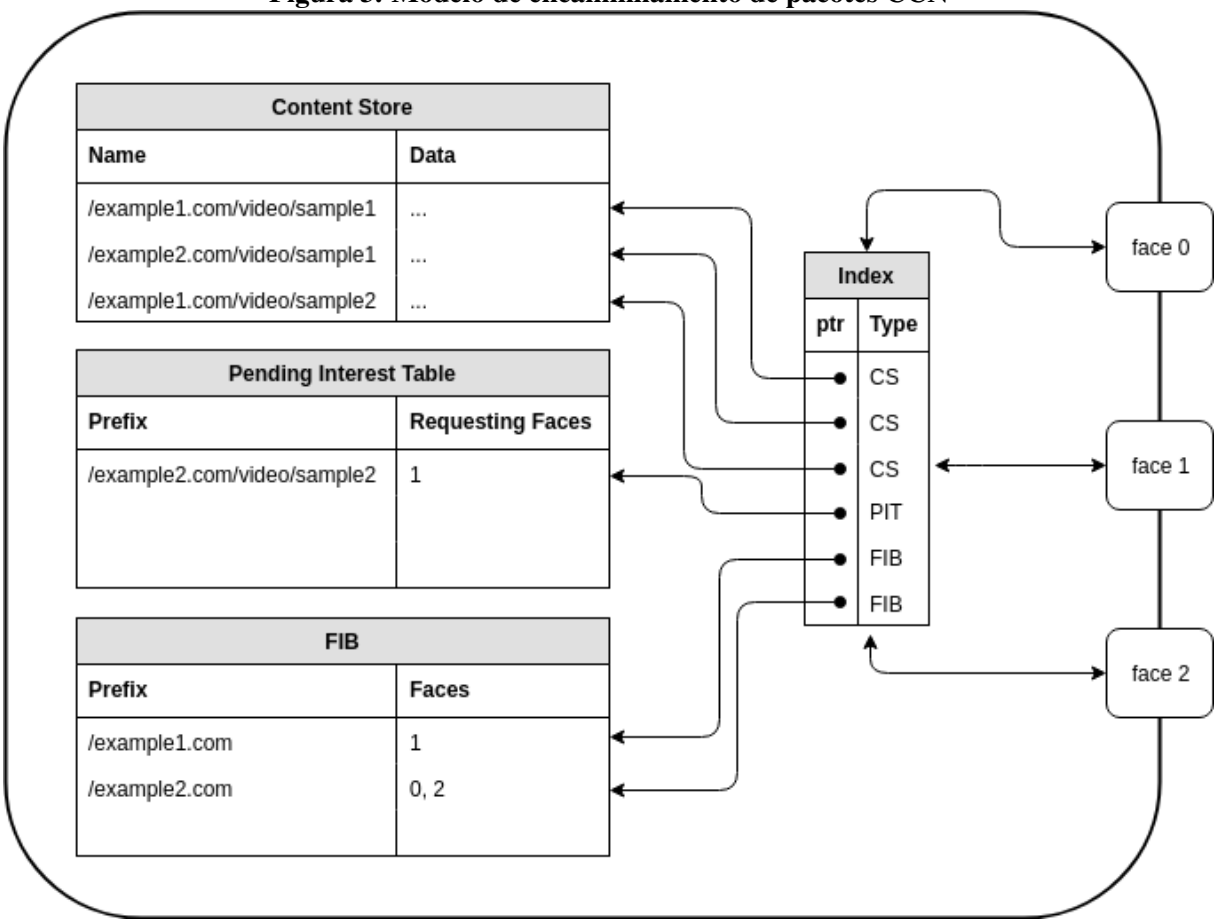
O controle da sequência dos pacotes são definidos e gerenciados pelas aplicações. Um *interest packet* pode especificar precisamente qual versão e porção do segmento deseja do conteúdo. Porém, geralmente devido ao fato de que o consumidor não conhece o nome completo de um conteúdo, o seu nome é especificado como um nome relativo.

Um roteador CCN, responsável por encaminhar os pacotes, é composto por três partes principais: *Content Store*, PIT (*Pending Interest Table*) e FIB (*Forwarding Information Base*), como pode ser observado na Figura 3.

Content Store é basicamente uma *cache*. Nela são armazenados os *data packets* com conteúdo já solicitados por outros consumidores. Esses *data packtes* são associados ao seu *Content Name* para buscas futuras. O propósito de manter essa *cache* é a de responder a solicitações pelo conteúdo com os dados armazenados localmente, reduzindo assim o tráfego *upstream*, e também diminuindo a latência *downstream*. Para armazenar os dados pelo maior tempo possível e potencializar o compartilhamento do conteúdo, utiliza-se as políticas de substituição da *cache* LRU (*Least Recently Used*) ou LFU (*Least Frequently Used*).

A PIT é um registro dos *interest packets* pendentes que ainda não foram atendidos. O objetivo é saber por quais *faces* foram recebidos as requisições por conteúdo, para que, quando os *data packets* estejam sendo processados no volta, se tenha o registro de para quais *faces*

Figura 3. Modelo de encaminhamento de pacotes CCN



Fonte: Autoria própria.

deve ser encaminhado *downstream* os *data packet*. Uma entrada na PIT para um *interest packet* que nunca seja satisfeito eventualmente expira e é removida. Na arquitetura CCN somente os *interest packets* são roteados; os *data packets* apenas seguem o 'rastro' deixado nas PIT até a orgirem da solicitação.

Já a FIB é utilizada efetivamente para rotear os *interest packtes*. Ela contém a associação entre os prefixos contido no *Content Name* e as *faces* de saída para encaminhamento *upstream* dos *interest packets* em direção aos potenciais provedores de conteúdo.

A tabela FIB é populada através de anúncios. Os provedores, também chamdos de *data sources*, enviam mensagens *broadcast* informando quais são os prefixos que suportam. Os roteadores CCN, ao receberem tais anúncios, aprendem por quais *faces* devem ser enviados os *interest packtes* para determinado prefixo.

Como já mencionado, *interest* e *data packets* são processados de maneira distinta, e seguem as seguintes regras:

- Processamento do *Interest packet*: Verificado na *Content Store* se já existe armazenado o conteúdo que satisfaça a busca pelo *Cotent Name* contido no *interest packet*. Se sim, um *data packet* é enviado pela *face* por onde foi recebido o *interest packet* e o mesmo é descartado, pois já foi atendido.

  Não sendo encontrado o conteúdo na *Content Store*, o próximo passo é verificar na PIT e, caso haja uma entrada nela de uma requisição para o mesmo conteúdo, a *face* por onde foi recebido o *interest packet* é adicionada a lista de *requesting faces* e o *interest packet* é descartado, pois uma solicitação já foi enviada upstream;

  No caso de não ter ocorrido um *match* na PIT, é então verificado a FIB, e se for encontrada para o prefixo a *face* de origem do *interest packet* é removido da lista de *faces* e o pacote é enviado *upstream* por todas as *faces* listadas, se ainda houver alguma, e uma entrada na PIT é criada. Se não for encontrado um prefixo na FIB o pacote é descartado.

- Processamento do *Data Packet*: É verificado na *Content Store* se o conteúdo desse *data packet* já se encontra armazenado e, em caso afirmativo, é descartado, pois está duplicado.

  Em seguida é analisado na PIT se houve uma requisição por esse conteúdo feita por esse nó. É então criada uma entrada na *Content Store* para esse *data packet*. Em seguida é gerada uma lista de saída com todas as *requesting interfaces* e então o pacote é enviado *downstream* pos essas *faces* e entrada e removida da PIT, pois foi atendida.

  No caso de não haver uma entrada na PIT para essa requisição, e acabar ocorrendo um *match* na FIB, isso significa que o pacote é *unsolicited*, e ele é então descartado.

No Apêndice A o Algoritmo 1 sumariza e representa a lógica de processamento de ambos pacotes.

A rede, como sempre, é um ambiente não seguro e sem garantia de entrega dos pacotes. Pode aconter de pacotes serem corrompidos ou perdidos. Então, caso um *interest packet* não tenha sido atendido em um determinado intervalo de tempo, é papel de quem fez a requisição reenviar a requisição, caso ainda queira o conteúdo.

Além dos problemas com a entrega dos pacotes, há a segurança da informação, que sempre é uma questão importante e primordial. Na rede CCN, a proteção e confiança do pacote estão junto do próprio conteúdo, ao invés de ser uma propriedade da conexão. Na arquitetuta CCN todo conteúdo é autenticado, mas mais do que isso, é autenticado o vínculo entre *Content Name* e o próprio conteúdo. A autenticação se dá por pacote, e utiliza a infraestrutura padrão de chave pública-privada. Para viabilizar isso, cada *data packet* CCN (Figura 1) contém informação para permitir a busca da chave pública para verificar a integridade e autenticidade

do par nome do conteúdo/conteúdo. Além disso, caso a informação seja confidencial, ela pode ser encriptada.

Toda a infraestrutura de autenticação é fundamental para possibilitar um dos aspectos mais importantes da rede CCN, o *caching* de conteúdo, pois, se o consumidor receberá a cópia do conteúdo da fonte mais próxima, que não a origem do conteúdo, deve ser possível validar os dados que recebe.

3 DESENVOLVIMENTO

Para colocar à prova o modelo *Content Centric Network* e explorar alguns componentes da arquitetura, um teste prático utilizando o *framework* que implementa a pilha de protocolos CCN foi executado. O teste, que consiste na utilização de máquinas virtuais com alguns *softwares* específicos para viabilizá-lo, consegue verificar a efetividade da requisição de conteúdo por nome, o roteamento e resposta do dado solicitado, assim como verificar um dos pontos mais interessantes do modelo, que é o *caching* do conteúdo mais próximo do solicitante.

As seguintes seções desde capítulo detalham os elementos envolvido no teste, suas funções e a topologia, assim como todos os *softwares* necessários para a configuração do ambiente.

3.1 CONSUMIDOR

O consumidor, como o nome já sugere, é o *host* que consumirá o conteúdo. É ele que envia a mensagem de requisição pelo conteúdo desejado, o *interest packet*, e aguarda uma resposta que atenda a solicitação ou uma mensagem de erro. Esse elemento deve estar apto a tratar requisição por nome de conteúdo, identificar o nome e verificar em sua tabela de rotas se possui um *gateway* para encaminhar a mensagem.

3.2 ROTEADOR CCN

O roteador é o elemento que implementa o mecanismo de ecaminhamento de pacotes CCN, como descrito na Figura 3 e Algoritmo 1. Ele é responsável por encaminhar as mensagens de *interest* em direção ao provedor, e repassar os *data packet* com o conteúdo requisitado de volta para quem os solicitou.

No roteador CCN do ambiente de testes será possível analisar com mais detalhes alguns componentes e características da aquitetura apresentada no Capítulo 2.

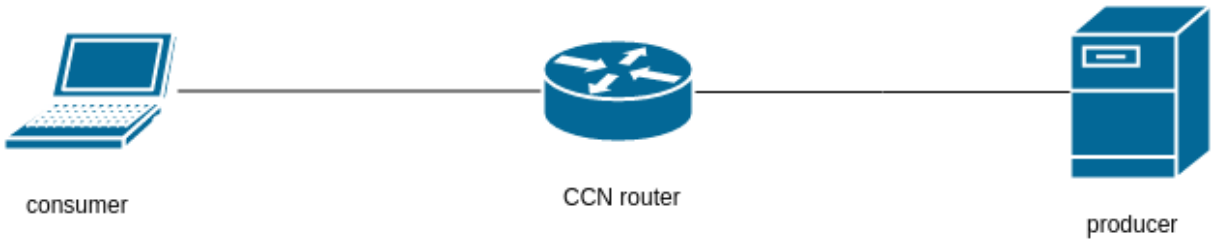
3.3 PROVEDOR

O provedor, também como o nome sugere, é o elemento que proverá o conteúdo. Ele disponibilizará conteúdo através de uma instância de um servidor HTTP capaz de tratar requisição pelo nome do conteúdo. O provedor recebe o *interest packet*, analisa se o prefixo do *Content Name* é atendido pelo servidor HTTP, busca o conteúdo e responde com o *data packet*.

3.4 TOPOLOGIA DO TESTE

A topologia do teste é bastante simples e consiste dos três elementos, consumidor, roteador e provedor em uma topologia em linha, onde o roteador é o elemento central e conecta-se aos demais *hosts*, como mostra a Figura 4.

Figura 4. Topologia do cenário de teste



Fonte: Autoria própria.

3.5 SOFTWARES UTILIZADOS PARA O TESTE

Um conjunto variado de *softwares* é necessário para operacionalizar o cenário de teste CCN. A seguir são relacionados os principais componentes necessários, assim como uma breve descrição sobre sua utilidade e respectiva versão utilizada.

3.5.1 UBUNTU

A versão do sistema operacial utilizada nos testes é Ubuntu 16.04 LTS.

A arquitetura da máquina é de 64 bits, com 8 GB de memória RAM e um processador Intel Core i7.

3.5.2 VAGRANT

Vagrant é um *software* utilizado para configurar e manter ambientes de desenvolvimento (VAGRANT, 2018).

A versão do Vagrant utilizada nos testes é a 2.0.1.

3.5.3 VIRTUALBOX

VirtualBox é um vitualizador de propósito geral para *hardwares* com arquitetura x86, como foco em servidores, *desktop* e embarcados (VIRTUALBOX, 2018).

A versão do VirtualBox utilizada nos testes é a 5.2.6.

3.5.4 METIS

A aplicação Metis (metis_daemon) é uma encaminhadora de pacotes CCN (FD.IO, 2018c). Basicamente tem a função de receber e encaminhar pacotes, mas, dependendo da sua posição na rede, sua responsabilidade pode variar, como descrito abaixo:

- Como elemento da ponta da rede: responsável por receber os pacotes CCN da rede e encaminhar para outras aplicações como, por exemplo, encaminhar para o servidor HTTP;
- Como elemento intermediário da rede: tem responsabilidade de receber e encaminhar os pacotes CCN entre elementos da rede.

A versão do Metis utilizada nos testes é a 1.0.

3.5.4.1 METIS CONTROL

Metis Control é uma aplicação auxiliar utilizada para alterar parâmetros do Metis quando ele já está em execução.

3.5.5 SERVIDOR HTTP

O servidor HTTP publica para seus clientes, via o protocolo HTTP, conteúdo na rede. O servidor é capaz de atender as requisições dos clientes tanto via TCP quanto via ICN (*Information-Centric Networking*) como o protocolo de transporte (FD.IO, 2018d). Para a realização dos testes aqui propostos se está interessados no modo ICN.

No modo ICN, o servidor ainda é bastante limitado e suporta apenas o método de *GET*.

Não foi encontrada a informação sobre qual é a versão do servidor.

3.5.6 LIBICNET

Biblioteca que provê a camada de transporte para aplicações que desejam se comunicar utilizando a pilha de protocolo ICN (FD.IO, 2018b).

Essa biblioteca também disponibiliza a aplicação iget, que será utilizada nos testes para enviar o método *GET*.

Também vale ressaltar que biblioteca é dependência direta do servidor HTTP.

Não foi encontrada a informação sobre qual é a versão da biblioteca.

3.5.7 VPP

VPP (*Vector Packet Processing*) é uma tecnologia Cisco que provê um *framework* para funcionalidades de switch/router (FD.IO, 2018e).

VPP, como o nome sugere, utiliza processamento em vetor, que processa mais de um pacote por vez, em contraste com o modelo escalar tradicional, que processa apenas um pacote por vez.

3.5.8 CICN

Um *plugin* CCNx ICN para o *framework* VPP (FD.IO, 2018a).

O *plugin* CICN fornece as seguintes funcionalidades:

- Rápido processamento de pacotes;
- Agregação dos pacotes de *Interest*;
- *Caching* de conteúdo.

3.6 CONFIGURAÇÃO DO AMBIENTE DE TESTES CCN

Para obter o Vagrantfile que faz o *bootstrap* do ambiente, realizar o *download* do arquivo Vagrantfile:

```
$ wget https://cisco.box.com/shared/static/\
d9y0x4v16w67iacvt0seu0q3nwl425cf –O Vagrantfile
```

Após terminado o *download* do arquivo, é necessário inicializar cada uma das máquinas virtuais da topologia. Para isso, utiliza-se o comando *vagrant up machine_name*. Então, para inicializar o consumidor, que no Vagrantfile recebe o nome de cicn1, faz-se:

```
$ vagrant up cicn1
```

Para inicializar o roteador, que no Vagrantfile recebe o nome de cicn2, faz-se:

```
$ vagrant up cicn2
```

E por fim, para inicializar o provedor, que no Vagrantfile recebe o nome de cicn3, faz-se:

```
$ vagrant up cicn3
```

Após inicializadas todas as máquinas virtuais, é possível verificar o estado de cada uma delas.

```
$ vagrant status
Current machine states:

cicn1                     running (virtualbox)
cicn2                     running (virtualbox)
cicn3                     running (virtualbox)
```

Todas as máquinas virtuais devem ter o seu estado indicado como *running*.

Depois de ter inicializado todas as máquinas virtuais, deve-se realizar algumas configurações específicas em cada uma para que todos os componentes e configurações do ambiente sejam alcançadas.

3.6.1 CONFIGURAÇÃO DO CONSUMIDOR

Entrar na máquina virtual do consumidor:

```
$ vagrant ssh cicn1
```

Instalar a biblioteca Libicnet, para definir a aplicação iget, que será utilizada para enviar a requisição pelo conteúdo. Para isso, deve-se adicionar o repositório que contém a aplicação a lista de confiáveis e prosseguir com a instalação:

```
$ echo "deb [trusted=yes] https://nexus.fd.io/content/
repositories/fd.io.master.ubuntu.$(lsb_release -sc).main/ ./"
| sudo tee -a /etc/apt/sources.list.d/99fd.io.list
$ sudo apt-get update
$ sudo apt-get install libicnet
```

O arquivo de configuração do metis_daemon para o cicn1 deve ter sido criado automaticamente pelo Vagrantfile e estar como abaixo:

```
$ cat metis.cfg
add listener tcp local0 127.0.0.1 9695
add listener udp local1 127.0.0.1 9695
add listener udp remote0 10.0.1.21 33302
add connection udp conn0 10.0.1.22 33302 10.0.1.21 33302
add route conn0 ccnx:/cicn 2
```

Destaca-se no arquivo de configuração mestis.cfg as duas últimas linhas, pois nelas estão a configuração que estabelece a conexão entre o conumidor e o roteador e dá-se o nome de *conn0* a conexão e, em seguida, é criada uma rota que tem *match* pelo prefixo *cicn* e encaminha o pacote pela conexão *conn0*.

Inicializar o metis_daemon:

```
$ metis_daemon --config metis.cfg &
```

São somentes esses os passos necessários para configurar o consumidor. Nesse ponto ele já está apto para enviar requisições.

3.6.2 CONFIGURAÇÃO DO ROTEADOR

Entrar na máquina virtual do roteador:

```
$ vagrant ssh cicn2
```

Instalar a biblioteca Libicnet, que será utilizada para encaminhar as mensagens. Para isso, deve-se adicionar o repositório onde está a biblioteca a lista de confiáveis e depois prosseguir com a instalação:

```
$ echo "deb [trusted=yes] https://nexus.fd.io/content/repositories/
fd.io.master.ubuntu.$(lsb_release -sc).main/ ./"
| sudo tee -a /etc/apt/sources.list.d/99fd.io.list
$ sudo apt-get update
$ sudo apt-get install libicnet
```

O arquivo de *shell script* do o cicn2 deve ter sido criado automaticamente pelo Vagrantfile e estar como abaixo:

```
$ cat start_vppcicn.sh
  ifconfig enp0s8 down
  ifconfig enp0s9 down
  modprobe uio
  modprobe igb_uio
  systemctl stop vpp
  dpdk-devbind -b igb_uio 00:08.0
  dpdk-devbind -b igb_uio 00:09.0
  systemctl start vpp
  vppctl set int ip address GigabitEthernet0/8/0 10.0.1.22/24
  vppctl set int state GigabitEthernet0/8/0 up
  vppctl set int ip address GigabitEthernet0/9/0 10.0.3.22/24
  vppctl set int state GigabitEthernet0/9/0 up
  vppctl cicn enable
  vppctl cicn cfg face add local 10.0.1.22:33302 remote 10.0.1.21:33302
  vppctl cicn cfg face add local 10.0.3.22:33302 remote 10.0.3.23:33302
  vppctl cicn cfg fib add prefix /cicn face 2
```

Destaca-se no arquivo de *shell script* as quatro últimas linhas, onde é habilitado o *pluging* CICN, adicionado as *faces* para conexão com o consumidor e provedor e, por último, adicionada uma entrada na tabela FIB para encaminhar para a face 2, a face do provedor, qualquer mensagem de *interest* que tenha o prefixo *cicn*.

Após a correta execução dos comandos de configuração do *shell script* deve-se ter as interfaces GigabitEthernet no estado *UP*, como na execução do *show* abaixo:

```
# vppctl show interface
              Name                Idx       State
Counter          Count
GigabitEthernet0/8/0                1        up
```

```
GigabitEthernet0/9/0                2          up
local0                              0          down
```

Também é possível verificar a configuração do *plugin* CICN, estatísticas de *interest* e *data packets* enviados e recebidos, e observar a criação de uma entrada na tabela FIB para encaminhar *upstream* pela *face* 2 pacotes com o prefixo *cicn*.

```
$ sudo vppctl cicn show
Forwarder no-name: enabled
  FIB:: max entries:512
  PIT:: max entries:131072, lifetime default: 2.000 sec
(min:0.200, max:2.000)
  CS::  max entries:4096
  PIT entries (now): 0
  CS entries (now): 0
  Forwarding statistics:
    pkts_processed: 0
    pkts_interest_count: 0
    pkts_data_count: 0
    pkts_nak_count: 0
    pkts_from_cache_count: 0
    pkts_nacked_interests_count: 0
    pkts_nak_hoplimit_count: 0
    pkts_nak_no_route_count: 0
    pkts_no_pit_count: 0
    pit_expired_count: 0
    cs_expired_count: 0
    cs_lru_count: 0
    pkts_drop_no_buf: 0
    interests_aggregated: 0
    interests_retransmitted: 0
Faces:
    Face 1: 10.0.1.22:33302 <-> 10.0.1.21:33302 (swif 1)
        Face Type:peer, State:up, FIB_NHs:0, Class:dpdk(clone)
        Hello Protocol: State:disabled
        Originated:     Interests:0, Data:0, Naks:0
        Terminated:     Interests:0, Data:0, Naks:0
        Received:       Interests:0, Data:0, Naks:0
```

```
        Sent:           Interests:0, Data:0, Naks:0
    Face 2: 10.0.3.22:33302 <-> 10.0.3.23:33302 (swif 2)
        Face Type:peer, State:up, FIB_NHs:1, Class:dpdk(clone)
        Hello Protocol: State:disabled
        Originated:     Interests:0, Data:0, Naks:0
        Terminated:     Interests:0, Data:0, Naks:0
        Received:       Interests:0, Data:0, Naks:0
        Sent:           Interests:0, Data:0, Naks:0
cicn FIB:
  /cicn/...       (face:2, weight:16)
```

3.6.3 CONFIGURAÇÃO DO PROVEDOR

Entrar na máquina virtual do provedor:

```
$ vagrant ssh cicn3
```

O arquivo de entrada do metis_daemon para o cicn3 deve ter sido criado automaticamente pelo Vagrantfile e estar como abaixo:

```
$ cat metis.cfg
add listener tcp local0 127.0.0.1 9695
add listener udp local1 127.0.0.1 9695
add listener udp remote0 10.0.3.23 33302
add connection udp conn0 10.0.3.22 33302 10.0.3.23 33302
```

Destaca-se no arquivo de configuração a última linha, onde é criada a conexão entre o provedor e o roteador.

Inicializar o metis_daemon:

```
$ metis_daemon --config metis.cfg &
```

Após, prosseguir e instalar o servidor HTTP. Para isso, deve-se adicionar o repositório onde está o servidor a lista de confiáveis e depois prosseguir com a instalação:

```
$ echo "deb [trusted=yes] https://nexus.fd.io/content/repositories/
fd.io.master.ubuntu.$(lsb_release -sc).main/ ./" \
        | sudo tee -a /etc/apt/sources.list.d/99fd.io.list
$ sudo apt-get update
```

```
$ sudo apt-get install http-server
```

Para colocar o servidor HTTP no ar basta executar:

```
$ http-server
Using web root folder: [/var/www/html]
Using locator: [http://webserver]
Route set correctly!
```

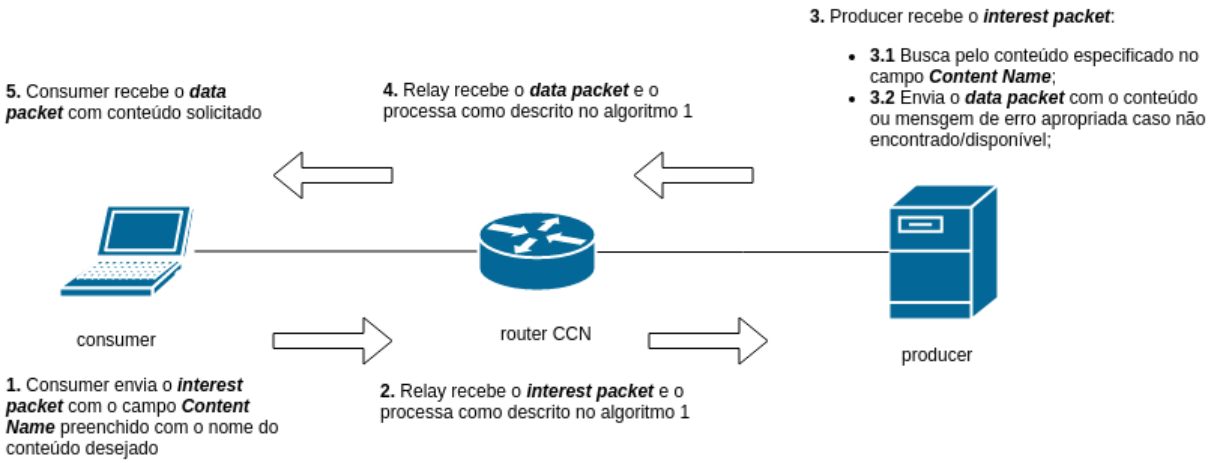
Observar que, com os valores *default* de inicialização do servidor HTTP, fica definido que o diretório raíz para disponibilizar conteúdo é '/var/www/html'. Esse diretório é criado automaticamente pelo servidor na sua na subida se já não existir. Também já fica estabelecido '*webserver*' como o prefixo a ser atendido por esse servidor.

Toda configuração que diz respeito ao cicn3, o provedor, está concluída. Basta apenas publicar conteúdo.

4 RESULTADOS

Para mostrar o mecanismo CCN em execução no ambiente de teste apresentado, foi realizado um teste, com os registros exibidos nesse capítulo, que tem o objetivo de mostrar, principalmente, como o mecanismo CCN é executado pelo roteador, destacando alguns pontos pertinentes ao longo da execução. Mas também ilustrará o consumidor enviando o *interest packet* que, por sua vez, é encaminhado pelo roteador CCN para o provedor, que analiza o prefixo, busca o conteúdo e responde com o *data packet* correspondente. Por fim, o roteador CCN verifica por onde veio a requisição do conteúdo e encaminha pela *face* apropriada para o cosumidor. Todo o fluxo dos pacotes é representado na Figura 5.

Figura 5. Processamento dos *interest* e *data packet*



Fonte: Autoria própria.

O teste pode ser dividido em três partes:

- Configuração do consumidor, roteador e provedor;
- Requisição por conteúdo inexistente e como fica o estado do roteador CCN;
- E por fim, requisições por conteúdo bem sucedidas e também uma análise de como fica o estado do roteador CCN;

O teste assume que as máquinas virtuais já estão em execução e com os *softwares* necessários instalados.

4.1 TESTE PRÁTICO: REQUISIÇÃO DE CONTEÚDO POR NOME E O MECANISMO CCN

Entrar no ambiente do consumidor (Figura 6):

Figura 6. Acesso a máquina virtual do consumidor

```
felipe@felipe-samsung:~/pos/workspace$ vagrant ssh cicn1
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-104-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

14 packages can be updated.
0 updates are security updates.


*** System restart required ***
Last login: Mon Feb 12 15:54:37 2018 from 10.0.2.2
-bash: warning: setlocale: LC_CTYPE: cannot change locale (pt_BR.UTF-8)
ubuntu@cicn1-v-felipe-samsung:~$
```

Fonte: Autoria própria.

Inicializar o metis_daemon no consumidor (Figura 7):

Figura 7. Inicialização do Metis no consumidor

```
ubuntu@cicn1-v-felipe-samsung:~$ metis_daemon --config metis.cfg &
[1] 16228
ubuntu@cicn1-v-felipe-samsung:~$ metis 1.0.20180205 2017-02-17T15:34:37.319950
Copyright (c) 2017 Cisco and/or its affiliates.

            __  __        _    _
           |  \/  |  ___ | |_ (_) ___
           | |\/| | / _ \| __|| |/ __|
           | |  | ||  __/| |_ | |\__ \
           |_|  |_| \___| \__||_||___/

[Metis Forwarding Strategy] --- set "random" for ccnx:/cicn
2018-02-12T16:02:23.152508Z Alert cicn1-v-felipe-samsung Core metis 65307676 [ m
etis running port 9695 configuration-port 2001 ]

ubuntu@cicn1-v-felipe-samsung:~$
```

Fonte: Autoria própria.

Executar o metis_control no consumidor (Figura 8) para adicionar uma rota com o prefixo *utfpr*:

Figura 8. Configuração de rota via Metis Control no consumidor

```
ubuntu@cicn1-v-felipe-samsung:~$ metis_control -p 1234
metis 1.0.20180205 2017-02-17T15:34:37.319950
Copyright (c) 2017 Cisco and/or its affiliates.

            __  __      _   _
           |  \/  | ___| |_(_)___
           | |\/| |/ _ \ __| / __|
           | |  | |  __/ |_| \__ \
           |_|  |_|\___|\__|_|___/

No keystore specified. Will try default.
> list routes
 iface  protocol   route     cost                 next prefix
     4    STATIC LONGEST        1 ---.---.---.---/.... ccnx:/cicn
Done

> add route conn0 ccnx:/utfpr 1
[Metis Forwarding Strategy] --- set "random" for ccnx:/utfpr
> list routes
 iface  protocol   route     cost                 next prefix
     4    STATIC LONGEST        1 ---.---.---.---/.... ccnx:/cicn
     4    STATIC LONGEST        1 ---.---.---.---/.... ccnx:/utfpr
Done

> quit
exiting interactive shell
ubuntu@cicn1-v-felipe-samsung:~$
```

Fonte: Autoria própria.

Na Figura 8 pode-se observar na listagem de rotas a rota para o prefixo *cicn* e a nova rota para o prefixo *utfpr*.

Por fim, no consumidor é criado o diretório de *downloads* para melhor organizar os conteúdos recebidos (Figura 9):

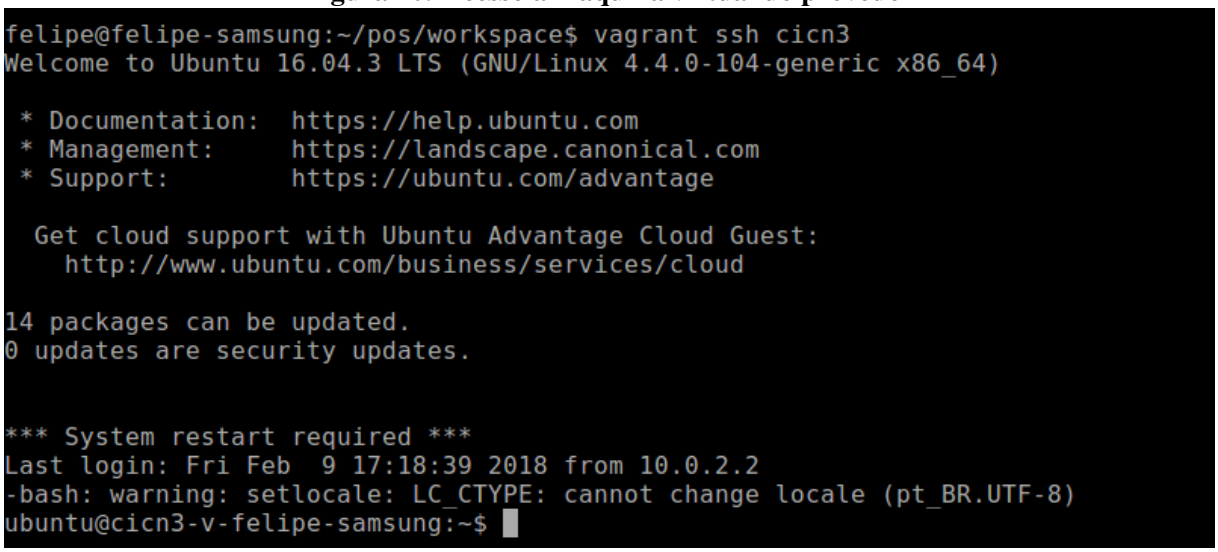
Figura 9. Criação do diretório de *downloads* no consumidor

```
ubuntu@cicn1-v-felipe-samsung:~$ mkdir downloads
ubuntu@cicn1-v-felipe-samsung:~$ cd downloads/
ubuntu@cicn1-v-felipe-samsung:~/downloads$ ls -l
total 0
ubuntu@cicn1-v-felipe-samsung:~/downloads$
```

Fonte: Autoria própria.

Entrar no ambiente do provedor (Figura 10):

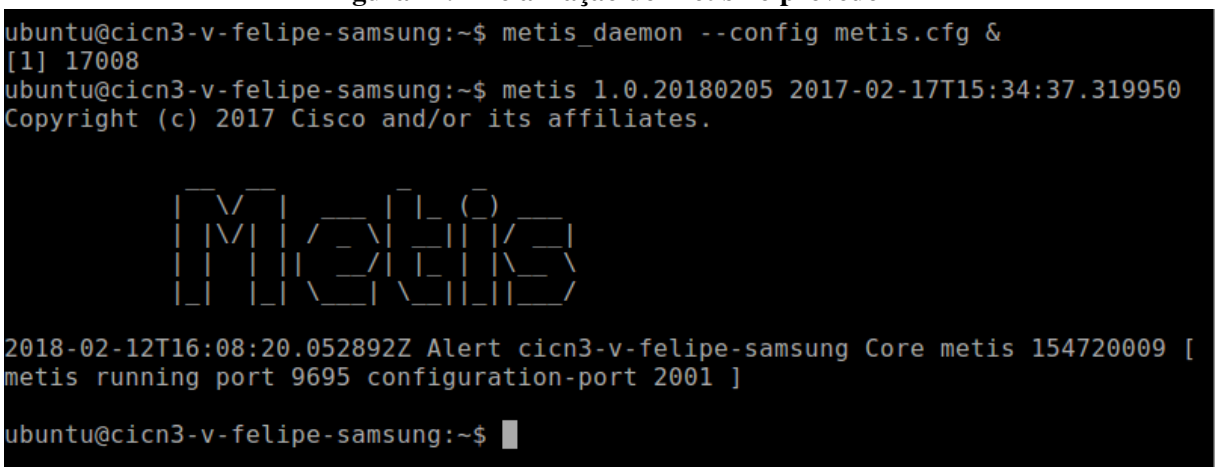
Figura 10. Acesso a máquina virtual do provedor



Fonte: Autoria própria.

Inicializar o metis_daemon no provedor (Figura 11):

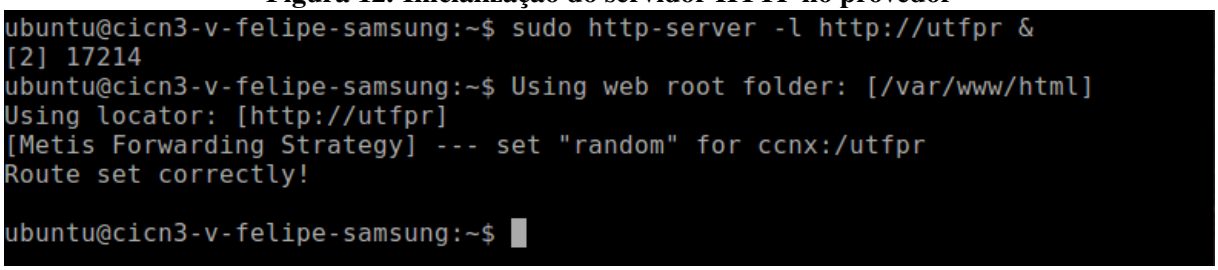
Figura 11. Inicialização do Metis no provedor



Fonte: Autoria própria.

Inicializar o servidor HTTP (Figura 12):

Figura 12. Inicialização do servidor HTTP no provedor

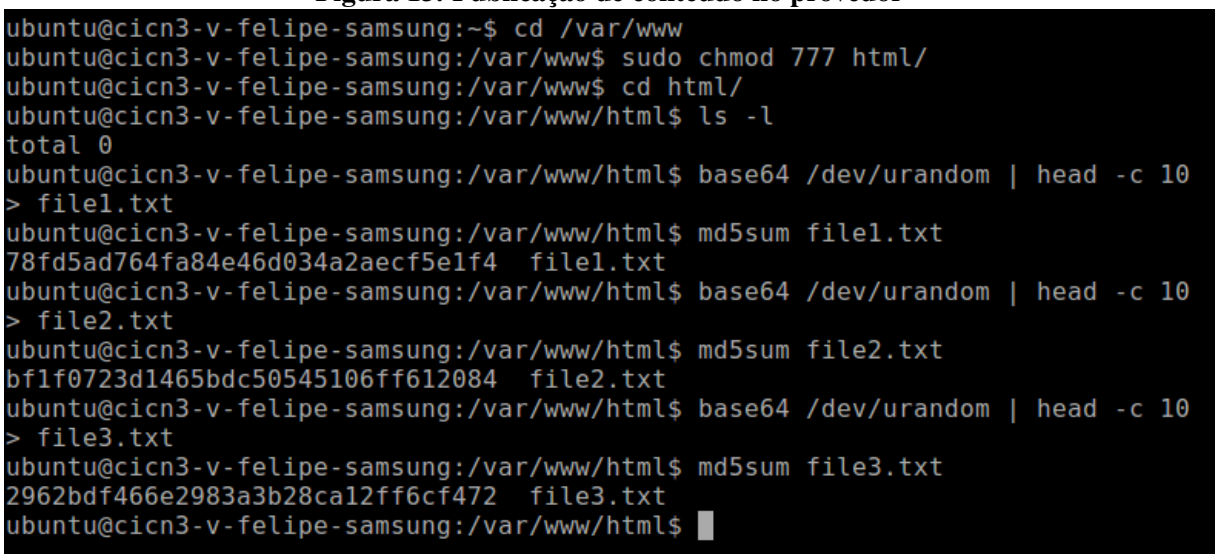


Fonte: Autoria própria.

Foi especificado o prefixo *utfpt* como o atendido pelo servidor HTTP, como pode ser observado na Figura 12.

No provedor é então criado e publicado três arquivos de texto com conteúdo aleatório, como pode ser observado na Figura 13. Também é calculado o md5sum dos arquivos para posterior verificação de integridade.

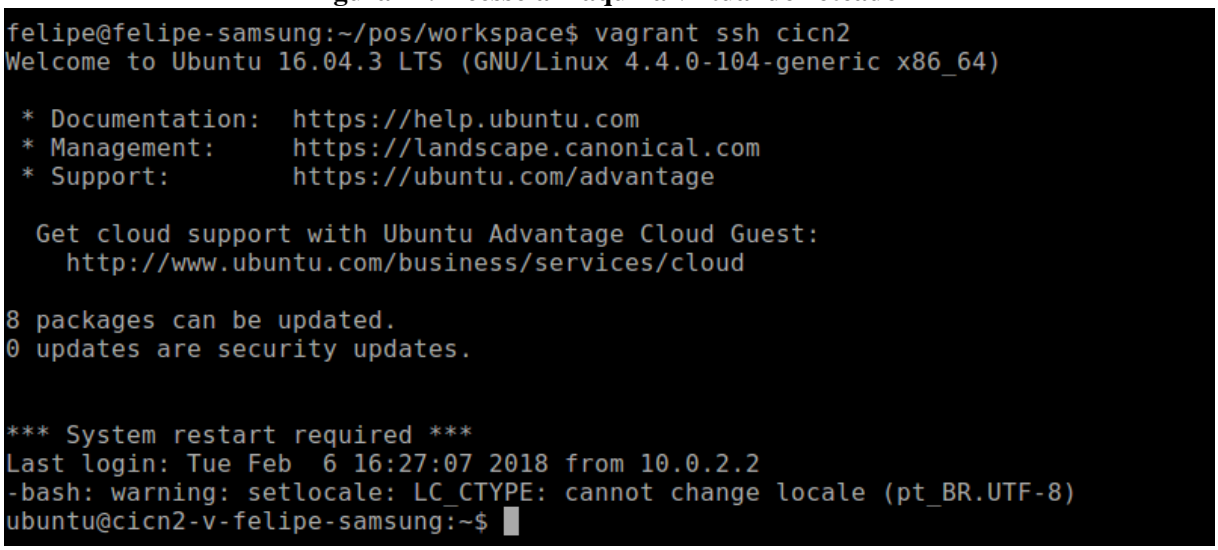
Figura 13. Publicação de conteúdo no provedor



Fonte: Autoria própria.

Entrar no ambiente do roteador (Figura 14):

Figura 14. Acesso a máquina virtual do roteador



Fonte: Autoria própria.

Realizar a configuração necessário no roteador (Figura 15):

Figura 15. Configuração do roteador

```
ubuntu@cicn2-v-felipe-samsung:~$ sudo ifconfig enp0s8 down
enp0s8: ERROR while getting interface flags: No such device
ubuntu@cicn2-v-felipe-samsung:~$ sudo ifconfig enp0s9 down
enp0s9: ERROR while getting interface flags: No such device
ubuntu@cicn2-v-felipe-samsung:~$ sudo modprobe uio
ubuntu@cicn2-v-felipe-samsung:~$ sudo modprobe igb_uio
ubuntu@cicn2-v-felipe-samsung:~$ sudo systemctl stop vpp
ubuntu@cicn2-v-felipe-samsung:~$ sudo dpdk-devbind -b igb_uio 00:08.0
0000:00:08.0 already bound to driver igb_uio, skipping

ubuntu@cicn2-v-felipe-samsung:~$ sudo dpdk-devbind -b igb_uio 00:09.0
0000:00:09.0 already bound to driver igb_uio, skipping

ubuntu@cicn2-v-felipe-samsung:~$ sudo systemctl start vpp
ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl set int ip address GigabitEthernet0
/8/0 10.0.1.22/24

ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl set int state GigabitEthernet0/8/0
up

ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl set int ip address GigabitEthernet0
/9/0 10.0.3.22/24

ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl set int state GigabitEthernet0/9/0
up

ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl cicn control param cs size 2

ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl cicn enable
cicn: worker count 0, first idx 1
cicn: fwdr initialize => 0

ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl cicn cfg name test-utfpr
name test-utfpr: added successfully

ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl cicn cfg face add local 10.0.1.22:3
3302 remote 10.0.1.21:33302
Face ID: 1

ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl cicn cfg face add local 10.0.3.22:3
3302 remote 10.0.3.23:33302
Face ID: 2

ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl cicn cfg fib add prefix /cicn face
2

ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl cicn cfg fib add prefix /utfpr face
 2

ubuntu@cicn2-v-felipe-samsung:~$
```

Fonte: Autoria própria.

Da configuração do rotedor, mostrado na Figura 15, tem-se alguns pontos que podem ser destacados:

- *Content Store* teve seu tamanho alterado para suportar apenas duas entradas;

- Nas últimas duas linhas é realizada a configuração das duas entradas na FIB para os prefixos *cicn* e *utfpr*.

Execução do *show cicn* para exibir estatísticas iniciais do mecanismo CCN, pode ser observada na (Figura 16).

Figura 16. *Show cicn* inicial no roteador

```
ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl cicn show
Forwarder test-utfpr: enabled
  FIB:: max entries:512
  PIT:: max entries:131072, lifetime default: 2.000 sec (min:0.200, max:2.000)
  CS::  max entries:2
  PIT entries (now): 0
  CS entries (now): 0
  Forwarding statistics:
    pkts_processed: 0
    pkts_interest_count: 0
    pkts_data_count: 0
    pkts_nak_count: 0
    pkts_from_cache_count: 0
    pkts_nacked_interests_count: 0
    pkts_nak_hoplimit_count: 0
    pkts_nak_no_route_count: 0
    pkts_no_pit_count: 0
    pit_expired_count: 0
    cs_expired_count: 0
    cs_lru_count: 0
    pkts_drop_no_buf: 0
    interests_aggregated: 0
    interests_retransmitted: 0
Faces:
    Face 1: 10.0.1.22:33302 <-> 10.0.1.21:33302 (swif 1)
        Face Type:peer, State:up, FIB_NHs:0, Class:dpdk(clone)
        Hello Protocol: State:disabled
        Originated:    Interests:0, Data:0, Naks:0
        Terminated:    Interests:0, Data:0, Naks:0
        Received:      Interests:0, Data:0, Naks:0
        Sent:          Interests:0, Data:0, Naks:0
    Face 2: 10.0.3.22:33302 <-> 10.0.3.23:33302 (swif 2)
        Face Type:peer, State:up, FIB_NHs:2, Class:dpdk(clone)
        Hello Protocol: State:disabled
        Originated:    Interests:0, Data:0, Naks:0
        Terminated:    Interests:0, Data:0, Naks:0
        Received:      Interests:0, Data:0, Naks:0
        Sent:          Interests:0, Data:0, Naks:0
cicn FIB:
  /cicn/...      (face:2, weight:16)
  /utfpr/...     (face:2, weight:16)

ubuntu@cicn2-v-felipe-samsung:~$
```

Fonte: Autoria própria.

Na saída do *show cicn* inicial, mostrado na Figura 16, pode-se observar:

- As duas rotas criadas na FIB;

- A PIT não possui nenhuma entrada;
- O indicativo de que a *Content Store* suporta apenas 2 entradas e também como não há nada armazenado nela até o momento;
- Todos os contadores de pacotes estão zerados.

Terminado o provisionamento dos serviços pode-se prosseguir com o teste e realizar a requisição de conteúdo. Inicialmente o consumidor envia uma requisição pelo arquivo file_fail.txt, porém não é atendida e não recebe resposta pois o conteúdo é inexistente. Essa situação pode ser observada na Figura 17.

Figura 17. Requisição não atendida do arquivo file_fail.txt

```
ubuntu@cicn1-v-felipe-samsung:~/downloads$ iget http://cicn/file_fail.txt
Setting RAAQM parameters:
params:  autotune = 0
params:  lifetime = 500
params:  retransmissions = 128
params:  beta = 0.99
params:  drop = 0.003
params:  beta_wifi_ = 0.99
params:  drop_wifi_ = 0.6
params:  beta_lte_ = 0.99
params:  drop_lte_ = 0.003
params:  wifi_delay_ = 200
params:  lte_delay_ = 9000
params:  alpha = 0.95
params:  batching = 200
params: choice = 0
init done
Drop Factor: 0.003
Minimum drop prob: 1e-05
Sample number: 30
lifetime: 500
beta: 0.99
Timeout on ccnx:/cicn/get/file_fail.txt/Chunk=%00
Timeout on ccnx:/cicn/get/file_fail.txt/Chunk=%00
Timeout on ccnx:/cicn/get/file_fail.txt/Chunk=%00
Timeout on ccnx:/cicn/get/file_fail.txt/Chunk=%00
Timeout on ccnx:/cicn/get/file_fail.txt/Chunk=%00
Timeout on ccnx:/cicn/get/file_fail.txt/Chunk=%00
Timeout on ccnx:/cicn/get/file_fail.txt/Chunk=%00
Timeout on ccnx:/cicn/get/file_fail.txt/Chunk=%00
Timeout on ccnx:/cicn/get/file_fail.txt/Chunk=%00
Timeout on ccnx:/cicn/get/file_fail.txt/Chunk=%00
^C
ubuntu@cicn1-v-felipe-samsung:~/downloads$
```

Fonte: Autoria própria.

Na Figura 18, são exibindas as estatísticas do mecanismo CCN após requisição não atendida.

Figura 18. *Show cicn* após processamento dos pacotes referentes ao file_fail.txt

```
ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl cicn show
Forwarder test-utfpr: enabled
  FIB:: max entries:512
  PIT:: max entries:131072, lifetime default: 2.000 sec (min:0.200, max:2.000)
  CS::  max entries:2
  PIT entries (now): 1
  CS entries (now): 0
  Forwarding statistics:
    pkts_processed: 11
    pkts_interest_count: 11
    pkts_data_count: 0
    pkts_nak_count: 0
    pkts_from_cache_count: 0
    pkts_nacked_interests_count: 0
    pkts_nak_hoplimit_count: 0
    pkts_nak_no_route_count: 0
    pkts_no_pit_count: 0
    pit_expired_count: 10
    cs_expired_count: 0
    cs_lru_count: 0
    pkts_drop_no_buf: 0
    interests_aggregated: 0
    interests_retransmitted: 0
Faces:
    Face 1: 10.0.1.22:33302 <-> 10.0.1.21:33302 (swif 1)
        Face Type:peer, State:up, FIB_NHs:0, Class:dpdk(clone)
        Hello Protocol: State:disabled
        Originated:    Interests:0, Data:0, Naks:0
        Terminated:    Interests:0, Data:0, Naks:0
        Received:      Interests:11, Data:0, Naks:0
        Sent:          Interests:0, Data:0, Naks:0
    Face 2: 10.0.3.22:33302 <-> 10.0.3.23:33302 (swif 2)
        Face Type:peer, State:up, FIB_NHs:2, Class:dpdk(clone)
        Hello Protocol: State:disabled
        Originated:    Interests:0, Data:0, Naks:0
        Terminated:    Interests:0, Data:0, Naks:0
        Received:      Interests:0, Data:0, Naks:0
        Sent:          Interests:11, Data:0, Naks:0
cicn FIB:
  /cicn/...      (face:2, weight:16)
  /utfpr/...     (face:2, weight:16)

ubuntu@cicn2-v-felipe-samsung:~$
```

Fonte: Autoria própria.

Na saída do *show cicn* após requisição não atendida, mostrado da Figura 18, pode-se observar:

- PIT possui uma entrada para um *interest packet* que não foi atendido;
- Foram processados 11 pacotes, e todos eles são do tipo *interest packet* pois, como não houve resposta, não foi enviado nenhum *data packet*;
- Todos os 11 *interest packets* foram enviados pelo consumidor para o roteador, e encaminhados para o provedor;

Consumidor envia o *interest packet* como requisição pelo arquivo file1.txt, como pode ser observado na Figura 19

Figura 19. Requisição e recepção do arquivo file1.txt

```
ubuntu@cicn1-v-felipe-samsung:~/downloads$ iget http://utfpr/file1.txt
Setting RAAQM parameters:
params:  autotune = 0
params:  lifetime = 500
params:  retransmissions = 128
params:  beta = 0.99
params:  drop = 0.003
params:  beta_wifi_ = 0.99
params:  drop_wifi_ = 0.6
params:  beta_lte_ = 0.99
params:  drop_lte_ = 0.003
params:  wifi_delay_ = 200
params:  lte_delay_ = 9000
params:  alpha = 0.95
params:  batching = 200
params: choice = 0
init done
Drop Factor: 0.003
Minimum drop prob: 1e-05
Sample number: 30
lifetime: 500
beta: 0.99
Saving to: file1.txt 0kB
Elapsed Time: 0.012 seconds -- 0.006[Mbps] -- 0.006[Mbps]
ubuntu@cicn1-v-felipe-samsung:~/downloads$ md5sum file1.txt
78fd5ad764fa84e46d034a2aecf5e1f4  file1.txt
ubuntu@cicn1-v-felipe-samsung:~/downloads$
```

Fonte: Autoria própria.

Na Figura 19 vê-se que a requisição pelo arquivo file1.txt é atendida com sucesso e o arquivo é recebido. No fim é calculado o md5sum do arquivo para verificar que está íntegro.

O provedor recebendo a requisição pelo file1.txt e enviando o *data packet* como resposta, pode ser observado na Figura 20.

Figura 20. Envio do arquivo file1.txt

```
ubuntu@cicn3-v-felipe-samsung:~$ Request ID-1650140887
Received request for: /file1.txt
Starting new thread
Rrsponse Id 1174940172
Replying to ccnx:/utfpr/get/file1.txt
ccnx:/utfpr/get/file1.txt
[Metis Forwarding Strategy] --- set "random" for ccnx:/utfpr/get/file1.txt
Route set correctly!
```

Fonte: Autoria própria.

Agora, com uma requisição bem sucedida, ao exibir as estatísticas do mecanismo CCN,

pode ser observada na Figura 21.

Figura 21. *Show cicn* após processamento dos pacotes referentes ao file1.txt

```
ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl cicn show
Forwarder test-utfpr: enabled
  FIB:: max entries:512
  PIT:: max entries:131072, lifetime default: 2.000 sec (min:0.200, max:2.000)
  CS::  max entries:2
  PIT entries (now): 1
  CS entries (now): 1
  Forwarding statistics:
    pkts_processed: 13
    pkts_interest_count: 12
    pkts_data_count: 1
    pkts_nak_count: 0
    pkts_from_cache_count: 1
    pkts_nacked_interests_count: 0
    pkts_nak_hoplimit_count: 0
    pkts_nak_no_route_count: 0
    pkts_no_pit_count: 0
    pit_expired_count: 10
    cs_expired_count: 0
    cs_lru_count: 0
    pkts_drop_no_buf: 0
    interests_aggregated: 0
    interests_retransmitted: 0
Faces:
    Face 1: 10.0.1.22:33302 <-> 10.0.1.21:33302 (swif 1)
        Face Type:peer, State:up, FIB_NHs:0, Class:dpdk(clone)
        Hello Protocol: State:disabled
        Originated:    Interests:0, Data:0, Naks:0
        Terminated:    Interests:0, Data:0, Naks:0
        Received:      Interests:12, Data:0, Naks:0
        Sent:          Interests:0, Data:2, Naks:0
    Face 2: 10.0.3.22:33302 <-> 10.0.3.23:33302 (swif 2)
        Face Type:peer, State:up, FIB_NHs:2, Class:dpdk(clone)
        Hello Protocol: State:disabled
        Originated:    Interests:0, Data:0, Naks:0
        Terminated:    Interests:0, Data:0, Naks:0
        Received:      Interests:0, Data:1, Naks:0
        Sent:          Interests:12, Data:0, Naks:0
cicn FIB:
  /cicn/...      (face:2, weight:16)
  /utfpr/...     (face:2, weight:16)

ubuntu@cicn2-v-felipe-samsung:~$
```

Fonte: Autoria própria.

Pode-se observar na Figura 21 que mostra o *show cicn* após processamento das mensagens referentes ao arquivo file1.txt:

- *Content Store* agora possui uma entrada, que corresponde ao dado associado ao nome do arquivo file1.txt;
- Possível verificar que foram processados o *interest packet* enviado na requisição e também *data packet*, enviado como resposta a requisição pelo file1.txt.

Na Figura 22, o consumidor envia o *interest packet* como requisição pelo arquivo file2.txt.

Figura 22. Requisição e recepção do arquivo file2.txt

```
ubuntu@cicn1-v-felipe-samsung:~/downloads$ iget http://utfpr/file2.txt
Setting RAAQM parameters:
params:  autotune = 0
params:  lifetime = 500
params:  retransmissions = 128
params:  beta = 0.99
params:  drop = 0.003
params:  beta_wifi_ = 0.99
params:  drop_wifi_ = 0.6
params:  beta_lte_ = 0.99
params:  drop_lte_ = 0.003
params:  wifi_delay_ = 200
params:  lte_delay_ = 9000
params:  alpha = 0.95
params:  batching = 200
params: choice = 0
init done
Drop Factor: 0.003
Minimum drop prob: 1e-05
Sample number: 30
lifetime: 500
beta: 0.99
Saving to: file2.txt 0kB
Elapsed Time: 0.077 seconds -- 0.001[Mbps] -- 0.001[Mbps]
ubuntu@cicn1-v-felipe-samsung:~/downloads$ md5sum file2.txt
bf1f0723d1465bdc50545106ff612084  file2.txt
ubuntu@cicn1-v-felipe-samsung:~/downloads$
```

Fonte: Autoria própria.

Na Figura 22, vê-se que a requisição também é atendida com sucesso e o arquivo é corretamente recebido.

Provedor recebendo a requisição pelo file2.txt e enviando o *data packet* como resposta é apresentada na Figura 23.

Figura 23. Envio do arquivo file2.txt

```
ubuntu@cicn3-v-felipe-samsung:~$ Request ID-2019368426
Received request for: /file2.txt
Starting new thread
Rrsponse Id 1376196800
Replying to ccnx:/utfpr/get/file2.txt
ccnx:/utfpr/get/file2.txt
[Metis Forwarding Strategy] --- set "random" for ccnx:/utfpr/get/file2.txt
Route set correctly!
```

Fonte: Autoria própria.

Na Figura 24, são exibidas as estatísticas do mecanismo CCN após processamento dos

pacotes referentes ao file2.txt.

Figura 24. *Show cicn* após processamento dos pacotes referentes ao file2.txt

```
ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl cicn show
Forwarder test-utfpr: enabled
  FIB:: max entries:512
  PIT:: max entries:131072, lifetime default: 2.000 sec (min:0.200, max:2.000)
  CS::  max entries:2
  PIT entries (now): 1
  CS entries (now): 2
  Forwarding statistics:
    pkts_processed: 15
    pkts_interest_count: 13
    pkts_data_count: 2
    pkts_nak_count: 0
    pkts_from_cache_count: 2
    pkts_nacked_interests_count: 0
    pkts_nak_hoplimit_count: 0
    pkts_nak_no_route_count: 0
    pkts_no_pit_count: 0
    pit_expired_count: 10
    cs_expired_count: 0
    cs_lru_count: 0
    pkts_drop_no_buf: 0
    interests_aggregated: 0
    interests_retransmitted: 0
Faces:
    Face 1: 10.0.1.22:33302 <-> 10.0.1.21:33302 (swif 1)
        Face Type:peer, State:up, FIB_NHs:0, Class:dpdk(clone)
        Hello Protocol: State:disabled
        Originated:    Interests:0, Data:0, Naks:0
        Terminated:    Interests:0, Data:0, Naks:0
        Received:      Interests:13, Data:0, Naks:0
        Sent:          Interests:0, Data:4, Naks:0
    Face 2: 10.0.3.22:33302 <-> 10.0.3.23:33302 (swif 2)
        Face Type:peer, State:up, FIB_NHs:2, Class:dpdk(clone)
        Hello Protocol: State:disabled
        Originated:    Interests:0, Data:0, Naks:0
        Terminated:    Interests:0, Data:0, Naks:0
        Received:      Interests:0, Data:2, Naks:0
        Sent:          Interests:13, Data:0, Naks:0
cicn FIB:
  /cicn/...      (face:2, weight:16)
  /utfpr/...     (face:2, weight:16)

ubuntu@cicn2-v-felipe-samsung:~$
```

Fonte: Autoria própria.

Na saída do *show cicn*, mostrado na Figura 24, pode-se verificar:

- *Content Store* agora possui duas entradas. A nova entrada corresponde ao dado associado ao nome do arquivo file2.txt;
- Possível verificar que foram processados o *interest packet* enviado na requisição e também o *data packet* enviado como resposta a requisição pelo file2.txt.

Por fim na Figura 25, o consumidor envia o *interest packet* como a requisição pelo arquivo file3.txt.

Figura 25. Requisição e recepção do arquivo file3.txt

```
ubuntu@cicn1-v-felipe-samsung:~/downloads$ iget http://utfpr/file3.txt
Setting RAAQM parameters:
params:  autotune = 0
params:  lifetime = 500
params:  retransmissions = 128
params:  beta = 0.99
params:  drop = 0.003
params:  beta_wifi_ = 0.99
params:  drop_wifi_ = 0.6
params:  beta_lte_ = 0.99
params:  drop_lte_ = 0.003
params:  wifi_delay_ = 200
params:  lte_delay_ = 9000
params:  alpha = 0.95
params:  batching = 200
params: choice = 0
init done
Drop Factor: 0.003
Minimum drop prob: 1e-05
Sample number: 30
lifetime: 500
beta: 0.99
Saving to: file3.txt 0kB
Elapsed Time: 0.01 seconds -- 0.008[Mbps] -- 0.008[Mbps]
ubuntu@cicn1-v-felipe-samsung:~/downloads$ md5sum file3.txt
2962bdf466e2983a3b28ca12ff6cf472  file3.txt
ubuntu@cicn1-v-felipe-samsung:~/downloads$
```

Fonte: Autoria própria.

Na Figura 25 vê-se que a requisição também é atendida com sucesso e o arquivo é corretamente recebido.

Provedor recebendo a requisição pelo file3.txt e enviando o *data packet* como resposta, pode ser observado na Figura 26.

Figura 26. Envio do arquivo file3.txt

```
ubuntu@cicn3-v-felipe-samsung:~$ Request ID-370134369
Received request for: /file3.txt
Starting new thread
Rrsponse Id 1716520806
Replying to ccnx:/utfpr/get/file3.txt
ccnx:/utfpr/get/file3.txt
[Metis Forwarding Strategy] --- set "random" for ccnx:/utfpr/get/file3.txt
Route set correctly!
```

Fonte: Autoria própria.

E, finalmente na Figura 27, exibindo as estatísticas do mecanismo CCN após

processamento dos pacotes referentes ao file3.txt:

Figura 27. *Show cicn* após processamento dos pacotes referentes ao file3.txt

```
ubuntu@cicn2-v-felipe-samsung:~$ sudo vppctl cicn show
Forwarder test-utfpr: enabled
  FIB:: max entries:512
  PIT:: max entries:131072, lifetime default: 2.000 sec (min:0.200, max:2.000)
  CS::  max entries:2
  PIT entries (now): 1
  CS entries (now): 0
  Forwarding statistics:
    pkts_processed: 17
    pkts_interest_count: 14
    pkts_data_count: 3
    pkts_nak_count: 0
    pkts_from_cache_count: 3
    pkts_nacked_interests_count: 0
    pkts_nak_hoplimit_count: 0
    pkts_nak_no_route_count: 0
    pkts_no_pit_count: 0
    pit_expired_count: 10
    cs_expired_count: 0
    cs_lru_count: 0
    pkts_drop_no_buf: 0
    interests_aggregated: 0
    interests_retransmitted: 0
Faces:
    Face 1: 10.0.1.22:33302 <-> 10.0.1.21:33302 (swif 1)
        Face Type:peer, State:up, FIB_NHs:0, Class:dpdk(clone)
        Hello Protocol: State:disabled
        Originated:    Interests:0, Data:0, Naks:0
        Terminated:    Interests:0, Data:0, Naks:0
        Received:      Interests:14, Data:0, Naks:0
        Sent:          Interests:0, Data:6, Naks:0
    Face 2: 10.0.3.22:33302 <-> 10.0.3.23:33302 (swif 2)
        Face Type:peer, State:up, FIB_NHs:2, Class:dpdk(clone)
        Hello Protocol: State:disabled
        Originated:    Interests:0, Data:0, Naks:0
        Terminated:    Interests:0, Data:0, Naks:0
        Received:      Interests:0, Data:3, Naks:0
        Sent:          Interests:14, Data:0, Naks:0
cicn FIB:
  /cicn/...      (face:2, weight:16)
  /utfpr/...     (face:2, weight:16)

ubuntu@cicn2-v-felipe-samsung:~$
```

Fonte: Autoria própria.

Agora, na saída do *show cicn*, mostrado na Figura 27, pode-se verificar algumas diferenças de comportamente:

- *Content Store* foi limpa e não possui nenhum dado armazenado, pois se atingiu o limite de duas entradas armazenadas;
- Possível verificar que foram processados o *interest packet* enviado na requisição e também o *data packet* enviado como resposta a requisição pelo file3.txt;

Além das observações acima das estatísticas após a última requisição, pode-se constatar na execução geral dos testes:

- Ao longo de todo o teste a entrada na PIT, referente ao arquivo file_fail.txt, nunca expirou, mesmo sendo definido o valor de 2 segundos para o *lifetime*.
- A estatística pkts_from_cache_count = 3 não parece ser coerente, pois todas as requisições bem sucedidas foram atendidas através do provedor;
- Consumidor, através do Metis, possui uma *cache* local de conteúdo. O *interest packet* nunca é enviado para o roteador CCN caso o conteúdo já esteja na *cache* local;
- Com a *cache* local no consumidor desabilitada, a *Content Store* é de fato consultada no caso de uma requisição por um conteúdo que está armazenado nela e utilizado para como desposta, sem necessidade de buscar novamente no provedor;
- Caso o consumidor realize uma requisição da qual o roteador CCN não saiba por qual *face* enviar o *interest packet upstream*, i.e. sem rota, nunhuma mensagem de erro é enviada para o consumidor, que fica retransmitindo a requisição indefinidamente;

5 DIFICULDADES E ERROS

Desde problemas no provisionamento do ambiente de testes até comportamentos não esperados de componentes do *framework*, percalços foram encontrados ao longo da execução do teste para o desenvolvimento do trabalho. As próximas sessões descrevem os principais problemas e dificuldades enfrentados.

5.1 INCOMPATIBILIDADE DE VERSÕES DE SOFTWARES UTILIZADOS

Foi encontrado um erro de utilização do VirtualBox versão 5.2.6, com a versão do Vagrant 1.8.1. Com as duas versões citadas, ao executar o comando *vagrant resume* das máquinas virtuais, ocorria o seguinte erro:

```
Vagrant assumes that this means the command failed!
/sbin/ifdown eth1
```

e abortava-se a configuração das máquinas.

A atualização do Vagrant para a versão 2.0.1 solucionou o problema de compatibilidade.

5.2 PROBLEMAS NA CONFIGURAÇÃO DO AMBIENTE

Ao tentar realizar o *bootstrap* de todo o cenário de teste utilizando o Vagrantfile, que instalaria, configuraria e deixaria todos os serviços necessários em execução, como alegavam as páginas de tutoriais consultadas, porém foram encontrados diversos erros. A seguir são descritos os principais problemas encontrados.

5.2.1 MEMÓRIA RAM DA MÁQUINA VIRTUAL CICN2

A quantidade de memória RAM alocada para a máquina virtual cicn2 é de 8 GB por padrão no Vagrantfile. Como a máquina hospedeira das máquinas virtuais dos testes possui essa

mesma quantidade de memória RAM, ao iniciar a máquina virtual cicn2, que aparenta ser a que mais consome recurso, toda a memória RAM da máquina hospedeira parecia ser utilizada, o que torna a máquina inoperacional.

Foi necessário alterar aos parâmetros de configuração da máquina virtual cicn2 para utilizar apenas 4 GB de memória RAM, e assim contornar o problema.

5.2.2 DIRETÓRIO DE PUBLICAÇÃO DE CONTEÚDO

Inicializar o servidor HTTP pela primeira vez sem ser *root* falha, pois é procurado pelo diretório de publicação de conteúdo */var/www/html* mas, como não existe, ocorre a tentativa de criá-lo. Porém, como a aplicação não tem a permissão necessária, a criação do repositório falha e a inicialização do servidor é interrompida. Abaixo exemplo da mensagem de erro recebida para o problema na inicialização do servidor HTTP:

```
$ http-server
The web root folder /var/www/html does not exist and its
creation failed. Exiting..
```

Foi necessário inicializar o serviço como *root* para que o diretório fosse corretamente criado.

5.2.3 PREFIXO DEFAULT DO SERVIDOR HTTP VERSUS ROTAS CRIADAS

As configurações para as rotas criadas automaticamente no consumidor e roteador tratam do prefixo *cicn*. Porém, o prefixo *default* que o servidor HTTP atende é *webserver*, como já explicado no capítulo 3. Desse modo, para que a requisição, roteamento e resposta estejam coerentes e funcionam como esperado, é necessário atentar para essa diferença e alterar a configuração *default* das rotas para *webserver*, ou definir *cicn* como o prefixo atendido pelo servidor HTTP, pois, caso contrário, nunca haverá resposta para as requisições.

5.2.4 VAGRANTFILE

O arquivo Vagrantfile, que deveria configurar todo o ambiente, apresenta erros e não é possível finalizar todos os passos necessários, e diversas etapas precisaram ser executadas manualmente, desde a instalação de pacotes, configuração de interfaces, até a inicialização das aplicações.

Seguem os pricipais problemas encontrados:

- Não instalar todos os pacotes necessários, como o Libicnet e todos os pacotes relacionados ao VPP;
- Não inicialização de serviços: devido a falta de pacotes, alguns serviços não foram inicializados, como o VPP, mas mesmo o metis_daemon, que é instalado e com o arquivo de configuração corretamente definido não é inicializado;
- Configuração do cicn2 apresenta erro: necessário executar manualmente os passos para configurações das interfaces e do *framework* VPP, que deveriam ser executados pelo *shell script*;

Além disso, o cenário básico não prevê a instalação do servidor HTTP, o que faz com que não seja possível testar se o ambiente CCN está funcional.

5.3 COMPORTAMENTOS

5.3.1 CONTENT STORE DO ROTEADOR ARMAZENA MENSAGEM DE ERRO

Devido ao modo de operação do servidor HTTP, que envia uma mensagem de *data* mesmo quando o conteúdo não é encontrado. Mensagem como essa:

```
HTTP/1.1 404 Not found
Content-Length: 29
```

Isso faz com o que o roteador CCN armazene a mensagem de erro como o conteúdo associado ao prefixo na sua *Content Store*. Dessa forma, mesmo quando o conteúdo é disponibilizado corretamente posteriormente, novas mensagens de *interest* para o tal conteúdo nunca são encaminhadas para o servidor HTTP, pois o roteador CCN responderá com o conteúdo armazenado na *Content Store* como descrito no Algoritmo 1, e somente a mensagem de erro é retornada.

5.3.2 ARQUIVO VAZIO PUBLICADO

Ao realizar o *iget* de um arquivo de texto vazio publicado pelo provedor, nunca há resposta para a requisição. Ocorre algum erro, provavelmente na recepção do arquivo no roteador que fez a requisição, pois é possível constatar que o envio da mensagem de *interest data* pelo provedor é realizada. Como o provedor indica que enviou a resposta para a requisição, mas o roteador não recebe nada e não há nenhuma mensagem de erro, fica-se sem informações sobre o motivo da falha.

5.3.3 CACHING LOCAL POR PADRÃO

O Metis está com a opção de *caching* local habilitado por padrão, mas não é claro que esse é o comportamento padrão. Isso pode levar a uma interpretação equivocada de que o conteúdo está sendo buscado da *Content Store* do router CCN quando, na verdade, o dado armazenado localmente é utilizado como resposta. Essa opção parece ser interessante quando o elemento executando o Metis não está localizado na extremidade da topologia.

6 CONCLUSÃO

Content Centric Network apresenta um modelo de comunicação novo, baseado nos nomes de conteúdos, eliminando a necessidade de conhecimento dos *hosts*. A tecnologia é bastante vasta, cobrindo e propondo novas soluções para aspectos da infraestrutura atual da rede IP. Com essa arquitetura baseada em nomes de conteúdo, o modelo espera ter uma melhor abstração para as características e demandas da comunicação de dados atual.

O fato de a arquitetura CCN permitir que seja desenvolvida de forma incremental e utilizada de maneira concomitantemente com o IP aumenta muito as chances de sua adoção.

A execução dos testes apresentou algumas dificuldades de configuração do ambiente mas, uma vez vencidos, foi possível constatar que o cerne da tecnologia de encaminhamento de pacotes por nome é funcional. Pode-se observar que a utilização da FIB com prefixos é bastante simples e similiar as FIBs implementadas hoje. Porém, devido ao fato da pilha do protocolo não estar completamente implementada no *framework*, diversos pontos da arquitetura CCN não puderam ser verificados, como a parte segurança, PIT expirar suas entradas e o provedor de dados divulgar prefixos para os roteadores. Como o *framework* de prototipação ainda é bastante limitado, deixa bastante espaço para a evolução do mesmo, seja para contemplar outros aspectos da rede CCN, ou mesmo comandos para melhor gerenciar suas partes.

Além disso, aspectos técnicos da arquitetura CCN ainda precisam ser evoluídos ou endereçados, como questões de otimização da utilização da *cache*, escalabilidade, propagação de conteúdo, balanceamento de carga, direito de acesso ao conteúdo da *cache*, etc.

REFERÊNCIAS

FD.IO. **Cicn-pluging**. 2018. Disponível em: <https://wiki.fd.io/view/Cicn-plugin>. Acesso em: 09 fev. 2018.

FD.IO. **ConsumerSocket/ProducerTransport API: data transport library for ICN**. 2018. Disponível em: <https://wiki.fd.io/view/Libicnet>. Acesso em: 09 fev. 2018.

FD.IO. **FD.IO CICN project: Metis**. 2018. Disponível em: <https://wiki.fd.io/view/Sb-forwarder>. Acesso em: 09 fev. 2018.

FD.IO. **HTTP Server over TCP/ICN**. 2018. Disponível em: <https://wiki.fd.io/view/Http-server>. Acesso em: 05 fev. 2018.

FD.IO. **VPP/What is VPP?** 2018. Disponível em: <https://wiki.fd.io/view/VPP/What_is_VPP%3F>. Acesso em: 09 fev. 2018.

JACOBSON VAN, e. a. **Network Named Content**. Rome, Italy: ACM, 2009.

VAGRANT. **Introduction to Vagrant**. 2018. Disponível em: <https://www.vagrantup.com/intro/>. Acesso em: 13 fev. 2018.

VIRTUALBOX. **About VirtualBox**. 2018. Disponível em: <https://www.virtualbox.org/wiki/VirtualBox>. Acesso em: 13 fev. 2018.

APÊNDICE A – PSEUDO-CÓDIGO PARA ENCAMINHAMENTO DE PACOTES CCN

```
if interest packet then
    if Content Name in CS then
        data = GetData(table = CS, Content Name);
        data packet = BuildDataPacket(Content Name, data);
        Send(data packet, origin face);
    else if Content Name in PIT then
        AddFace(table = PIT, Content Name, origin face);
        Drop(interest packet) ; // interest already sent upstream
    else if Content Name Prefix in FIB then
        DelFace(table = FIB, Content Name, origin face);
        output faces = GetFace(table = FIB, Content Name);
        if output faces > 0 then
            Send(interest packet, output faces);
            AddEntry(table = PIT, Content Name, origin face);
        else
            Drop(interest packet) ;  // does not know about content
        end
    else
        Drop(interest packet) ;      // does not know about content
    end
else                                                  // data packet
    if Content Name in CS then
        Drop(data packet) ;                      // duplicated packet
    else if Content Name in PIT then
        AddEntry(table = CS, Content Name, data);
        output faces = GetFace(table = PIT, Content Name);
        Send(data packet, output faces);
        DelEntry(table = PIT, Content Name) ;    // interest satisfied
    else
        Drop(data packet) ;                      // unsolicited packet
    end
end
```

Algorithm 1: Pseudo-código para encaminhamento de pacotes CCN.

APÊNDICE B – COMANDOS DO PLUGIN CICN

Segue abaixo lista de alguns comandos úteis suportados pelo *plugin* CICN:

- vppctl cicn help
- vppctl cicn control start
- vppctl cicn control stop
- vppctl cicn enable-disable [disable]
- vppctl cicn cfg name <*name*>
- vppctl cicn cfg name <*name*> delete
- vppctl cicn cfg fib add prefix <*prefix*> face <*face #*>
- vppctl cicn cfg fib delete prefix <*prefix*> face <*face #*>
- vppctl cicn control param cs size <*# of entries*>
- vppctl cicn control param pit size <*# of entries*>
- vppctl cicn control param pit {dfltlife | minlife | maxlife} <*seconds*>
- vppctl cicn control param fib size <*# of entries*>
- vppctl cicn show

Todos os comandos devem ser executados como *root*.