UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

EDUARDO TIAGO PETRY

Automatização de Testes de Conformidade da Implementação do OSPFv3 do *Quagga* para IPv6

Trabalho de Graduação

Prof. Dr. Sérgio Luis Cechin
Orientador

Porto Alegre, Dezembro de 2011

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Petry, Eduardo Tiago

Automatização de Testes de Conformidade da Implementação do OSPFv3 do *Quagga* para IPv6 / Eduardo Tiago Petry. – Porto Alegre: Instituto de Informática da UFRGS, 2011.

71 f.: il.

Trabalho de Graduação – Universidade Federal do Rio Grande do Sul. Curso de Engenharia de Computação, Porto Alegre, BR–RS, 2011. Orientador: Sérgio Luis Cechin.

1. OSPF. 2. IPv6. 3. Quagga. 4. Ruby. 5. Automatização de testes de conformidade. I. Cechin, Sérgio Luis. II. Automatização de Testes de Conformidade da Implementação do OSPFv3 do *Quagga* para IPv6.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ABR	Area Border Router
AS	Autonomous System
ASBR	Autonomous System Border Router
BDR	Backup Designated Router
BGP	Border Gateway Protocol
CLI	Command Line Interface
DR	Designated Router
DUT	Device Under Test
IEEE	Institute of Electrical and Electronics Engineers
FIB	Forwarding Information Base
IGP	Interior Gateway Protocol
IP	Internet Protocol
LSA	Link State Adverstisement
LSDB	Link State Database
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PC	Personal Computer
RFC	Request For Comments
RIP	Routing Information Protocol
SPF	Shortest Path First
TCP	Transmission Control Protocol
VLAN	Virtual Local Area Network
VTY	Virtual Teletype

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A automatização de testes de conformidade de protocolos presentes em *switches* e roteadores tem crescido entre desenvolvedores de equipamentos para telecomunicações. Usualmente, emprega-se o protocolo OSPF (*Open Shortest Path First*) para o roteamento interno de *backbones* sobre redes Metro Ethernet, as quais operam atualmente, em sua maioria, com endereços IPv4 (*Internet Protocol version 4*).

Tendo em vista a breve migração operacional dessas redes para o uso de endereços IPv6, pretende-se apresentar neste trabalho testes de conformidade da implementação da versão 3 do OSPF, a qual fornece suporte a esse endereçamento.

Portanto, o objetivo é verificar se a implementação deste protocolo de roteamento, apresentada pelo *Quagga*, está em conformidade com a norma RFC 5340 (COLTUN, et al., 2008), tanto em situações normais de operação quanto em momentos de falha.

Palavras-chave: OSPF, IPv6, Quagga, Ruby, Automatização de testes de conformidade.

Automatização de Testes de Conformidade da Implementação do OSPFv3 do *Quagga* para IPv6

ABSTRACT

Automated compliance tests of embedded protocols in switches and routers has increased by developers of telecommunications equipments. Usually, OSPF (Open Shortest Path First) is the internal routing protocol of Internet backbones, deployed over Metro Ethernet networks, which currently operate with IPv4 (Internet Protocol version 4).

Since the operational migration of these networks to use IPv6 addresses is soon, this course conclusion paper intends to introduce compliance tests for OSPF version 3 implementation, which provides support for that addressing.

Hence, this paper aims to check whether Quagga's implementation of this routing protocol is in compliance with standard RFC 5340 (COLTUN, et al., 2008), both in normal operation as in failure.

Keywords: OSPF, IPv6, Quagga, Ruby, Automated compliance tests.

1 INTRODUÇÃO

Diversos protocolos das camadas 2 e 3 do modelo OSI (TANENBAUM, 2003), assim como funcionalidades específicas de *switches* e roteadores, tais como VLAN (IEEE802.1Q, 2006) e Port-Channel (IEEE802.1AX, 2008), são exaustivamente testados em laboratório antes de serem comercializados. Tradicionalmente, testes de conformidade desses equipamentos são compostos por um roteiro que descreve cada etapa a ser executada, a fim de verificar se cada funcionalidade efetua suas ações adequadamente. Caso uma falha seja evidenciada, muitas vezes o *bug* encontrado não é facilmente reproduzível (GROTTKE; TRIVEDI, 2007): seja por concorrência entre os processos internos do equipamento (o que torna o resultado não-determinístico), seja por problemas de temporização das ações do hardware ou do software.

A fim de minimizar esses problemas e, além disso, estabelecer uma forma sistemática de aplicar esses testes, é possível traduzi-los, em uma primeira análise, para *scripts* automatizados. Estes, por sua vez, podem tornar mais ágil a verificação de certas funcionalidades em um equipamento qualquer; obtendo, assim, uma garantia de que a execução dos passos do roteiro, mencionado anteriormente, serão sempre realizados em uma mesma ordem e com os mesmos intervalos de tempo.

Caso um protocolo qualquer (contido no software embarcado de um roteador) possua erros em sua implementação, estes podem ser evidenciados através de testes de conformidade do equipamento, focada nas propriedades desse protocolo. Automatizando os testes, essa execução pode ser feita com uma periodicidade muito menor. Além disso, a sequência de passos executados estarão padronizados, sem influência humana ou outra variável que possa interfir no resultado e na análise de possíveis falhas.

1.1 Contextualização

Neste trabalho, pretende-se analisar o protocolo de rede OSPF (*Open Shortest Path First*), classificado como um IGP (*Interior Gateway Protocol*) (TANENBAUM, 2003), em sua versão 3 (OSPFv3) (COLTUN, et al., 2008), na qual é utilizado o IPv6 (*Internet Protocol version 6*) (HINDEN; DEERING, 1998). Sua versão tradicional (MOY, 1998), implementada para IPv4, é amplamente utilizada em roteamento dentro de um mesmo *SA (Sistema Autônomo)*.

Para que seja possível um computador pessoal operar como um roteador, é utilizado o *Quagga*, o qual implementa os principais protocolos de roteamento, inclusive suportando IPv6. O sistema operacional e o hardware de um computador não apresentam arquitetura dedicada ao roteamento, porém o *Quagga* apresenta bom desempenho em comparação com roteadores Cisco (V. ERAMO M. LISTANTI, 2005). Ele pode ser instalado facilmente em um computador com sistema operacional baseado em *Unix* (ISHIGURO,

et al., 2006). Detalhes da sua arquitetura serão apresentados posteriormente na seção 2.3.

1.2 Motivação

Apesar da capacidade de endereçamento do IPv4 estar próxima do limite (NICBR, 2011), ainda é majoritário seu uso na rede mundial, em especial nas grandes operadoras de telecomunicações no Brasil (por exemplo, *Oi-RJ*, *CEMIG-MG*, *Embratel-RJ* e *Telefonica-SP*). Elas estão preparando seu *backbone* para o uso do IPv6, através de testes internos e adiantando preparação de suas equipes de operação e suporte.

Dessa maneira, uma das motivações para este trabalho é o fato da migração completa para operações reais utilizando IPv6 ainda não ter sido atingida. Por isso, protocolos de roteamento completamente preparados ao novo modo de endereçamento, tal como o OSPFv3, não são, por enquanto, utilizados em larga escala. Portanto, ainda não há testes automáticos plenamente desenvolvidos para validação de protocolos e funcionalidades relacionados ao IPv6.

1.3 Objetivos

Após os objetivos alcançados no Trabalho de Graduação 1 (TG1), disposto no anexo A, o prosseguimento deste trabalho tem as seguintes tarefas para atingir o objetivo de automatizar os testes de conformidade da implementação do OSPFv3 do *Quagga* para IPv6:

- Analisar a especificação dos testes descritos no TG1, melhorando sua descrição e, se necessário, refatorá-los a fim de implementar o código de forma modular.
- Descrever a infraestrutura necessária para implementar e executar os testes de conformidade.
- Implementar um *framework* que suporte a comunicação com o *Quagga*, assim como efetue o tratamento das informações dele recebidas. Deve também conter ferramentas de captura de tráfego e geração de pacotes OSPFv3.
- Implementar os casos de teste utilizando o *framework*.
- Executar os testes de conformidade propostos, os quais devem informar cada etapa que está sendo verificada e, ao final, relatar se houve erros.

O protocolo OSPF possui uma grande quantidade de informações e diferentes pacotes a serem testados. Portanto, a fim de limitar o escopo deste trabalho, serão apresentados em capítulos posteriores os principais pacotes do OSPFv3 (dentro de uma mesma área) e possíveis falhas em uma rede, sobre os quais incidirão testes de conformidade. Dessa forma, pode-se verificar, mesmo que parcialmente, se a implementação do OSPFv3 contida no *Quagga* está de acordo com sua norma específica (COLTUN, et al., 2008).

2 CONCEITOS INICIAIS

2.1 IPv6

A RFC 2460 (HINDEN; DEERING, 1998) especifica um novo formato de endereçamento IP, o qual é composto por 128 bits, ao invés de apenas 32 bits do IPv4. A motivação principal para o incremento na quantidade de bits é exatamente a limitação na quantidade de possibilidades de endereços do IPv4, o qual está chegando ao fim de sua vida útil, com capacidade de pouco mais de 4 bilhões de endereços (STALLINGS, 1997). Com o espaço de endereçamento do IPv6, podem ser gerados, aproximadamente, $3.4 * 10^{38}$ endereços.

2.1.1 Tipos de Endereços Utilizados

Conforme a RFC 4291 (HINDEN; DEERING, 2006), quanto à sua identificação, os endereços IPv6 podem ser classificados de acordo com a tabela 2.1. A exceção é o endereço com todos os bits em 0, o qual é dito "não especificado" e, por isso, não deve ser utilizado em quaisquer situações.

Tabela 2.1: Tipos de endereços IPv6

Tipo de endereço	**Notação**
Loopback	::1/128
Multicast	FF00::/8
Link-Local unicast	FE80::/10
Global unicast	Endereços restantes

Segundo essa norma, é necessário atribuir um endereço *Link-Local Unicast* a cada interface de rede.

A notação "::" indica que há uma sequência de bits com o valor ‘0’ antes ou após esse sinal. Por exemplo, para o endereço do tipo *loopback*, existem 127 bits ‘0’ antes de "::". No entanto, os prefixos de rede dos endereços *Link-Local unicast* contém uma sequência de zeros após a notação "::".

Neste trabalho, além de utilizar-se o formato de endereçamento com prefixo *FE80::/10*, também foram empregados nas interfaces de rede (conforme será visto no capítulo 4) o tipo de endereço IPv6 especificado pela RFC 4193 (HINDEN; HABERMAN, 2005): *Unique Local IPv6 Unicast Addresses*. Estes endereços são identificados pelo prefixo *FC00::/7* e são definidos pelo seguinte formato:

Tabela 2.2: Formato de *Unique Local IPv6 Unicast Addresses*

7 bits	*1 bit*	*40 bits*	*16 bits*	*64 bits*
Prefixo	1, se atribuído localmente	ID global	ID da sub-rede	ID da interface

Chamados simplesmente de "*local IPv6 unicast*", esse endereço teve seu uso planejado apenas para comunicações locais, em roteamento limitado a um escopo menor, o qual é delimitado pela própria configuração aplicada no roteador, ao invés de roteado na Internet.

2.2 OSPFv3

O protocolo de roteamento OSPF (*Open Shortest Path First*), dito como um IGP (*Interior Gateway Protocol*), é utilizado dentro de um mesmo AS (*Autonomous System*) (HAGEN, 2002). Ele é um protocolo de "estado de enlace" (*link-state*). Pode-se considerar um enlace como sendo a interface de um roteador. A descrição dessa interface (seu endereço IP, máscara de rede e tipo de rede em que está conectada, por exemplo) e o relacionamento do enlace com seus roteadores vizinhos representam o "estado de enlace" (HALABI, 1996).

O protocolo OSPF define um conceito de "área", a qual pode conter uma ou mais redes. Caso existam mais áreas, uma delas deve ser identificada como "área 0", que é denominada *backbone* (identificada pelo *ID* '0'). Todas as áreas devem estar diretamente conectadas a ela (ou indiretamente, através de túneis, chamados *virtual links*) (PAYNE; MANWEILER, 2003). Caso um roteador pertença a mais de uma área, ficando na divisão entre elas, chamamos ele de *ABR - Area Border Router*. Outro roteador pode ainda estar na fronteira do seu AS, denominado, então *ASBR - Autonomous System Border Router*.

As principais mensagens OSPF que um roteador envia são:

- *Hello*: primeira mensagem enviada, antes mesmo de estabelecer adjacência com seu nó vizinho. Após isso, o intuito é saber se o vizinho está ativo (usualmente enviada a cada 10 segundos) (CISCO, 2008).

- *Link-State Advertisements - LSAs*: distribui informações de estado do *link* para seus vizinhos.

A fim de manterem-se atualizados, periodicamente (ou quando há alteração na topologia), os roteadores trocam entre si *LSAs*, nas quais constam informações que eles conhecem sobre cada enlace. Caso a topologia esteja estável, para não consumirem tráfego desnecessário na rede onde está presente o protocolo OSPF, as LSAs são reenviadas pelo roteador que a originou após o tempo recomendado de 1800 segundos desde o primeiro envio. O valor é a metade do tempo máximo que uma LSA permanece na base de dados do roteador receptor dela. Esse período de 3600 segundos é chamado de "*MaxAge*" e, após esse tempo, a LSA não é mais utilizada no cálculo da árvore *Shortest Path First* (SPF).

Essa "inundação" (ou "*flooding*"), como é chamada, pode ocorrer do roteador que a originou até um vizinho específico (escopo do *link*), para todos os vizinhos de uma mesma área ou de um mesmo AS. Dessa forma, cada nó monta uma base da dados, formada pelas

LSAs recebidas e pelos dados locais: compondo a LSDB (*Link-State Database*). Sobre esta, é executado localmente o algoritmo, o qual calcula o caminho mais curto para atingir uma determinada rede, levando em consideração o custo associado para chegar ao destino.

Através das informações contidas nos pacotes *Hello*, são eleitos dois roteadores denominados "*Designated Router*" (DR) e "*Backup Designated Router*" (BDR). Com o objetivo de reduzir o número de adjacências, os roteadores pertencentes ao mesmo domínio *broadcast* formam-as somente com o DR, minimizando a quantidade de tráfego específico do OSPF. O BDR assume a função em de caso falha no DR. O mecanismo de eleição segue um algoritmo que está fora do escopo deste trabalho.

2.2.1 Cabeçalho OSPFv3

O cabeçalho do pacote OSPFv3 está destacado na figura 2.1:

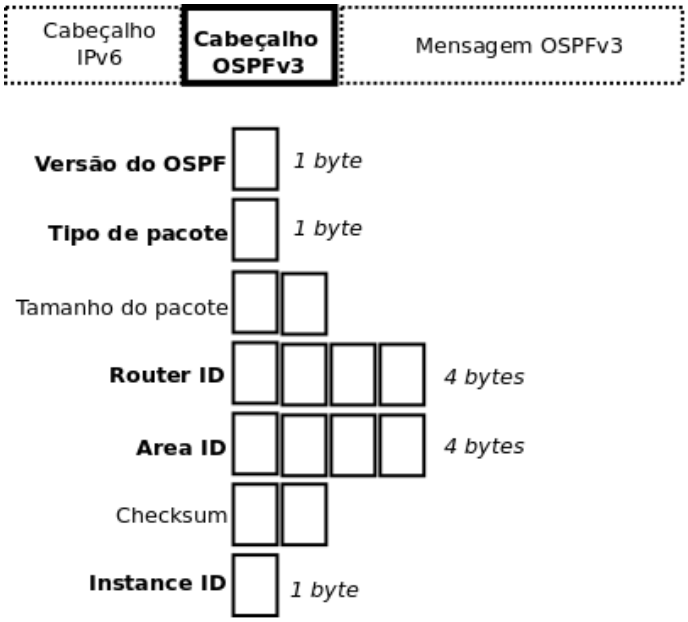

Figura 2.1: Cabeçalho do pacote OSPFv3

Os campos indicados em negrito são aqueles que serão objeto de verificação durante o capítulo 3. Abaixo, segue um resumo sobre cada um desses parâmetros:

- Versão do OSPF: no caso do OSPFv3, sempre contém o valor "3".
- Tipo de pacote: dentro do escopo deste trabalho serão analisados os pacotes com os valores "1" (*Hello*) e "4" (*Link State Update*, referente às LSAs).
- *Router ID*: identificador do roteador que originou o pacote.
- *Area ID*: área à qual a mensagem pertence.
- *Instance ID*: instância do OSPFv3. Permite que diferentes instâncias do OSPF trafeguem no mesmo *link*, apenas diferenciando por esse campo.

2.2.2 Pacote *Hello*

Assim como já comentado anteriormente, o pacote *Hello* do OSPF é responsável por iniciar e manter a formação de adjacência com um roteador vizinho, além de ser a ferramenta de eleição do DR e BDR. Os campos a seguir (destacados na figura 2.2) serão verificados no capítulo 3:

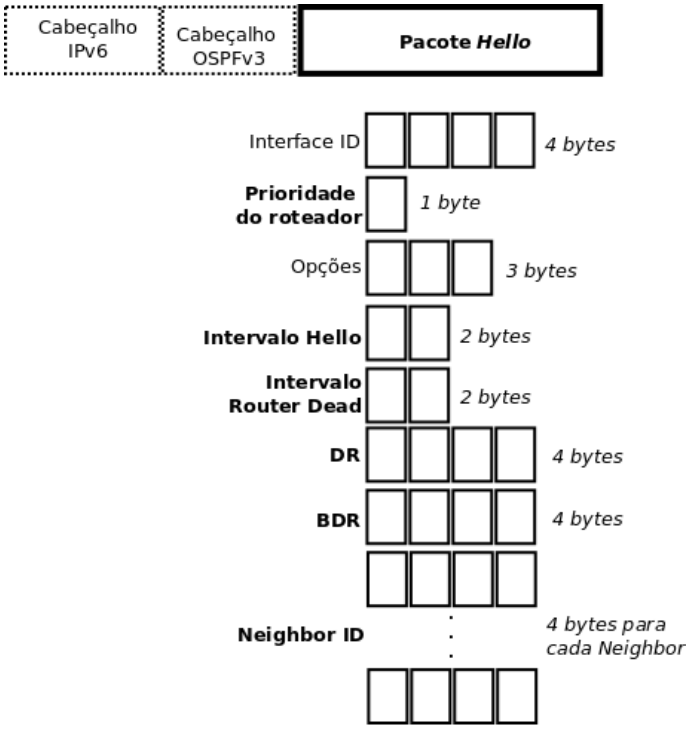

Figura 2.2: Pacote *Hello* do OSPFv3

- Prioridade do roteador: configurado no roteador, cujo valor mais alto é mais propício a torna-se DR.
- Intervalo *Hello*: tempo entre cada pacote.
- Intervalo *Router Dead*: tempo para considerar um roteador vizinho como inativo no *link* em que se comunicam.
- *DR*: *Designated Router* do roteador no *link*.
- *BDR*: *Backup Designated Router* do roteador no *link*.
- *Neighbor ID*: *Router ID* de cada vizinho que o roteador recebe pacotes *Hello*.

2.2.3 Tipos de LSAs

Conforme será apresentado posteriormente na tabela 2.3, quatro (de um total de nove) tipos de LSAs sofrerão teste de conformidade. A quantidade de LSAs escolhida levando em consideração a simplicidade e a limitação do escopo deste trabalho. Segue uma breve descrição de cada uma delas:

- *Router-LSA*: divulgada por cada roteador; lista todos os *links* que possuem, pelo menos, um vizinho adjacente.

- *Network-LSA*: o DR de um *link* faz o anúncio de todos os roteadores agregados a este *link*, através de uma lista contendo o *Router IDs* de cada um.

- *Link-LSA*: originado por cada *link* de cada roteador. Esta LSA fornece, principalmente: o endereço IPv6 *link-local* do roteador e uma lista de prefixos IPv6 associados a este *link*.

- *Intra-Area-Prefix-LSA*: um roteador utiliza essa LSA para divulgar prefixos IPv6 relacionados tanto com o próprio roteador quanto a uma *Network-LSA*.

As LSAs compartilham o mesmo formato de cabeçalho, porém cada uma delas tem sua área de dados específica. No cabeçalho, os últimos 13 bits do campo "*LS type*" informam o tipo da LSA. Na tabela 2.3, baseada em (HAGEN, 2002), está um resumo de cada tipo de pacote *Link State* e, na tabela 2.4, estão os campos que serão verificados nos capítulos posteriores:

Tabela 2.3: Tipos de LSAs

Nome	**Escopo do *flooding***	**Quem envia**
Router-LSA	Área	Cada roteador
Network-LSA	Área	*Designated Router - DR*
Link-LSA	*Link*	Cada roteador para cada *link*
Intra-Area-Prefix-LSA	Área	Cada roteador

Tabela 2.4: Campos das LSAs a serem testados

Nome	***LS type***	**Campo(s)**
Router-LSA	0x2001	*Neighbor Router ID*
Network-LSA	0x2002	*Attached Router*
Link-LSA	0x0008	*Address Prefix*/*PrefixLength* e *Link-local Interface Address*
Intra-Area-Prefix-LSA	0x2009	*Address Prefix*

2.3 Quagga

O *Quagga* é um software que implementa protocolos de roteamento, tais como RIP(*Routing Information Protocol*), OSPF e BGP(*Border Gateway Protocol*). Um PC, emulando o hardware de um roteador e rodando um sistema operacional *Unix*, pode operar como um roteador ao utilizar o *Quagga.*

A comunicação entre a tabela de roteamento do *kernel* e os protocolos ocorre através de um processo gerenciador, denominado *zebra*. Ele é uma abstração entre o *kernel* Unix e a API *Zserv*, pela qual foram implementados os protocolos de roteamento disponíveis no pacote do *Quagga.*

Em uma descrição simples a respeito da sua arquitetura, pode-se dizer que o *zebra* administra a tabela de roteamento através da troca de informações com o *kernel* do

sistema operacional e obtendo dados através dos protocolos de roteamento que estão implementados no *Quagga.*

Existe uma interface de comando (*CLI - Command Line Interface*) chamada de *Virtual teletype* (VTY), típica em roteadores, pela qual é possível conectar-se com o *daemon* utilizando o *Telnet* através do endereço e a porta *TCP (Transmission Control Protocol)* pela qual tal acesso é permitido. Assim, configuram-se as interfaces e parâmetros dos protocolos de roteamento, tal como o OSPFv3 do *Quagga*. Além disso, obtém-se informações relativas ao estado de cada protocolo que deseja-se executar. No caso do interesse deste trabalho, as portas TCP a serem utilizadas serão as de número 2601 e 2606; o endereço (IPv6) será aquele habilitado a receber conexões TCP ao executar os *daemons zebra* e *ospf6d*.

Tabela 2.5: Serviços do *Quagga*

Nome do *daemon*	**Porta TCP**	**Descrição**
zebra	2601	Administração do Zebra
ripd	2602	RIPv1 e RIPv2
ripngd	2603	RIPng - IPv6
ospfd	2604	OSPFv2
bgpd	2605	BGP - IPv4 e IPv6
ospf6d	2606	OSPFv3 - IPv6

Mais detalhes sobre a forma de executar o serviço desejado e, em seguida, acessá-lo serão tratados, posteriormente, na seção 4.1.4.

3 ESPECIFICAÇÃO DOS TESTES

A divisão dos casos de teste que serão utilizados foram inspirados na metodologia proposta em (BHOSALE; JOSHI, 2008). Segue a relação dos itens e uma breve descrição sobre o que cada um deles têm a responsabilidade de testar:

- **Paramétricos**
  Verificar se a configuração de itens tais como *Area ID*, *Router ID*, diferentes parâmetros de temporização e de métrica são aceitos no roteador.

- **Informações no roteador**
  Analisar nos *daemons zebra* e *ospf6d*, dependendo dos dados desejados, se eles apresentam as informações, configuradas anteriormente, de forma correta.

- **Conformidade dos pacotes**
  Assim como já introduzido na seção 2.2, as mensagens do tipo *Hello* e algumas LSAs serão verificadas. Em uma primeira análise, os testes estarão restritos a somente uma área do OSPF.

A cada teste descrito posteriormente, os itens acima descritos serão integralmente verificados, à medida que estiverem dentro do escopo de determinado teste.

3.1 Elementos dos Testes

Os elementos que compõem a especificação dos testes são apresentados na figura 3.1. Eles são de dois tipos:

- **Gerenciador**: responsável por executar os testes, além de gerar, enviar e receber os pacotes trocados com os roteadores.

- **DUT** - *Device Under Test*: roteador que sofrerá o teste. Posteriormente, no capítulo 4, será apresentado o DUT como sendo um PC operando como um roteador quando utilizado o *Quagga*. Poderá variar a quantidade de DUTs utilizados.

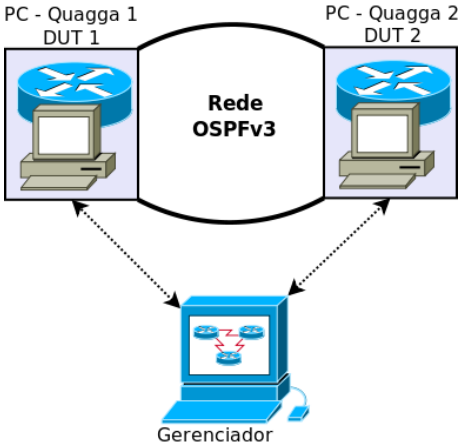

Figura 3.1: Componentes dos testes

3.2 Topologias de Rede

Grande parte dos casos que serão vistos podem ser verificados em diferentes topologias utilizadas em cenários reais. As mais comuns são roteadores em **linha** (figura 3.2) e **anel** (figura 3.3). Portanto, as topologias com dois roteadores, aqui apresentadas, podem ser modificadas mantendo válidos os casos de teste, através do aumento na quantidade de roteadores ou utilizando outro formato de topologia.

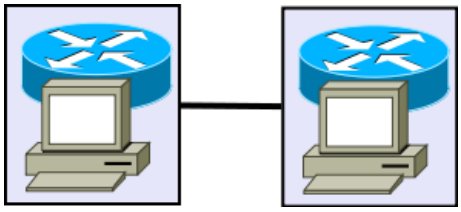

Figura 3.2: Topologia em linha

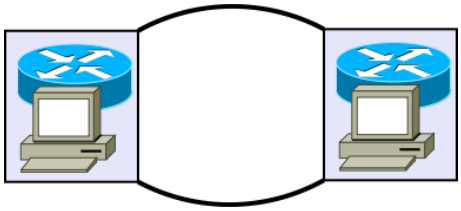

Figura 3.3: Topologia em anel

3.3 Configuração dos Endereços IPv6

Durante os testes serão utilizadas quatro interfaces de rede, nas quais configura-se um par de endereços IPv6 para cada uma delas, totalizando oito endereços. Uma alternativa, caso não esteja disponível tal quantidade de interfaces, é criá-las através de "VLANs",

as quais foram utilizadas para este trabalho e cujo conceito será abordado na seção 4.1.3.1. Os tipos de endereços IPv6 utilizados estão descritos na seção 2.1.1. A tabela 3.1 apresenta os endereços configurados em cada interface.

Tabela 3.1: Endereços IPv6 nas interfaces dos roteadores

Roteador	**Interface (mnemônico)**	**Endereço *Link-local***	**Endereço *Unique Local***
ROUTER_1	A10	fc00::10:1/120	fe80::10:1/100
	A20	fc00::20:1/120	fe80::20:1/100
	A30	fc00::30:1/120	fe80::30:1/100
	A40	fc00::40:1/120	fe80::40:1/100
ROUTER_2	B10	fc00::10:2/120	fe80::10:2/100
	B20	fc00::20:2/120	fe80::2000:2/100
	B30	fc00::30:2/120	fe80::3000:2/100
	B40	fc00::40:2/120	fe80::4000:2/100

3.4 Testes Operacionais

3.4.1 Testes com 1 Roteador - Pacotes *Hello*

O objetivo é utilizar uma topologia simples, ainda sem formar adjacência do OSPF com nenhum roteador vizinho, conforme apresentado na figura 3.4. Através de alguns testes básicos, pretende-se verificar se os pacotes *Hello* estão adequados às configurações que serão aplicadas no roteador alvo do teste.

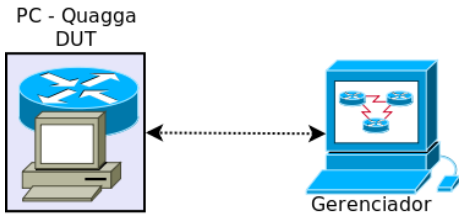

Figura 3.4: Teste inicial

Setup:

1. Utilizando-se somente o roteador (DUT) *ROUTER_1*, habilitar o roteamento IPv6.

2. Habilitar o OSPF na interface *A10* através da área *0.0.0.0*, configurando-se o *Router ID* com o valor *10.1.1.101*, conforme demonstrado na figura 3.5. A opção "*redistribute connected*" faz com que os prefixos localmente configurados no roteador sejam futuramente divulgados a algum roteador vizinho.

```
ROUTER_1# configure terminal
ROUTER_1(config)# router ospf6
ROUTER_1(config-ospf6)# interface A10 area 0.0.0.0
ROUTER_1(config-ospf6)# router-id 10.1.1.101
ROUTER_1(config-ospf6)# redistribute connected
```

Figura 3.5: Configuração inicial do teste com 1 roteador

3. Dentro da configuração da interface *A10*, alterar o valor de cada parâmetro para o apresentado na figura 3.6.

```
cost 100
hello-interval 5
dead-interval 20
retransmit-interval 3
priority 251
transmit-delay 1
instance-id 100
```

Figura 3.6: Parâmetros customizáveis na interface

4. Gerenciador captura pacotes *Hello*.

Resultados esperados:

1. Os parâmetros são aceitos na configuração, verificando que não foram retornados erros.

2. Informações são exibidas corretamente no CLI do roteador, obtendo tais dados a partir do resultado do comando "*show running-config*", executado no roteador.

3. Valores contidos dentro do pacote *Hello* devem ser exatamente aqueles configurados anteriormente.

3.4.2 Testes com 2 Roteadores - Pacotes *Link-State Advertisements*

Aplicando a topologia em linha (figura 3.2), será configurado no *ROUTER_2* parâmetros análogos aos utilizados para o *ROUTER_1* na seção 3.4.1. Dessa forma, haverá formação de adjacência entre os dois roteadores e, posteriormente, intercâmbio de LSAs.

Ambos os roteadores deverão ter endereços IPv6 configurados, tanto na interface pela qual estão conectados, quanto em outra interface secundária, pela qual não há tráfego de pacotes OSPFv3. Maiores detalhes desses endereços serão tratados na implementação, na seção 4.1.3.2.

Conforme apresentado na tabela 2.3, neste teste serão verificadas quatro tipos de LSAs, as quais foram brevemente descritas na seção 2.2.3.

Setup:

1. Dois DUTs.

2. *ROUTER_1* mantém as configurações do teste da seção 3.4.1.
3. NO *ROUTER_2* são aplicadas as configurações da figura 3.6.
4. O *Router ID* do *ROUTER_2* é *10.1.1.102* e a interface é *B10*, conforme a figura 3.7:

```
ROUTER_2# configure terminal
ROUTER_2(config)# router ospf6
ROUTER_2(config-ospf6)# interface B10 area 0.0.0.0
ROUTER_2(config-ospf6)# router-id 10.1.1.102
ROUTER_2(config-ospf6)# redistribute connected
```

Figura 3.7: Configuração para o segundo roteador

5. Gerenciador aguarda 30 segundos para capturar alguns pacotes *Hello* e 1900 segundos para obter pelo menos um pacote de cada tipo de *LSA* especificada na seção 2.2.3.

Resultados esperados:

1. Os parâmetros são aceitos na configuração, verificando que não foram retornados erros.
2. Os prefixos aprendidos pelo *ROUTER_1* estão corretos, verificados a partir das informações exibidas no CLI do roteador através do comando "*show ipv6 route*" (figura 3.8). O sufixo "O" indica que o prefixo foi aprendido através do OSPF:

```
ROUTER_1# show ipv6 route
Codes: K - kernel route, C - connected, S - static, R - RIPng, O - OSPFv3,
I - ISIS, B - BGP, * - FIB route.

C>* ::1/128 is directly connected, lo
O fc00::10:0/120 [110/1] is directly connected, wlan0.10, 00:32:23
C>* fc00::10:0/120 is directly connected, wlan0.10
C>* fc00::20:0/120 is directly connected, wlan0.20
C>* fc00::30:0/120 is directly connected, wlan0.30
C>* fc00::40:0/120 is directly connected, wlan0.40
O>* fc00::2000:0/120 [110/1] via fe80::10:2, wlan0.10, 00:32:23
O>* fc00::3000:0/120 [110/1] via fe80::10:2, wlan0.10, 00:32:23
O>* fc00::4000:0/120 [110/1] via fe80::10:2, wlan0.10, 00:32:23
C * fe80::/100 is directly connected, wlan0.40
C * fe80::/100 is directly connected, wlan0.30
C * fe80::/100 is directly connected, wlan0.20
C>* fe80::/100 is directly connected, wlan0.10
ROUTER_1#
```

Figura 3.8: Saída de "*show ipv6 route*" no roteador *ROUTER_1*

3. Valores dos campos das LSAs, apresentados na seção 2.2.3, estão corretos, verificados através da captura dos pacotes, cujos detalhes são descritos posteriormente na seção 4.2.1.6.

3.4.3 Testes com 2 Roteadores - Aprendizado de Rotas Estáticas

Uma maneira de um roteador aprender rotas através do OSPF, é seu vizinho adjacente redistribuir rotas estáticas configuradas diretamente no *Kernel* do *Linux* ou, alternativamente, no *daemon* do *zebra*. Optou-se por efetuar através desta última opção.

Condição inicial:

A partir de uma rede na topologia em linha (conforme a figura 3.2), estabelecer a adjacência entre os vizinhos, de modo que todos estejam no estado *Full*.

Setup:

1. Configurar 5 rotas estáticas no *daemon zebra* através do comando "*ipv6 route* <*destination_i*> <*gateway*>" no roteador *ROUTER_2*, cujo <*gateway*> é um endereço IPv6 na mesma sub-rede de uma interface (por exemplo, *B20*) onde não há anúncios do OSPF:

```
ROUTER_2(config)# ipv6 route 1001::/96 fc00::2000:10
(...)
ROUTER_2(config)# ipv6 route 1005::/96 fc00::2000:10
```

Figura 3.9: Configuração de rotas estáticas

2. Configurar no mesmo roteador, no *daemon ospf6*, o parâmetro *redistribute static*.

3. Gerenciador captura pacotes trocados entre os DUTs.

Resultados esperados:

1. Os parâmetros são aceitos na configuração, verificando que não foram retornados erros.

2. Os prefixos aprendidos pelo *ROUTER_1* estão corretos, verificados a partir das informações exibidas no CLI do roteador através do comando "*show ipv6 route*". Além das linhas apresentadas no teste da seção 3.4.2, os seguintes prefixos são aprendidos pelo *ROUTER_1*:

```
ROUTER_1# show ipv6 route
(...)
O>* 1001::/96 [110/1] via fe80::10:2, wlan0.10, 00:01:07
O>* 1002::/96 [110/1] via fe80::10:2, wlan0.10, 00:01:07
O>* 1003::/96 [110/1] via fe80::10:2, wlan0.10, 00:01:07
O>* 1004::/96 [110/1] via fe80::10:2, wlan0.10, 00:01:07
O>* 1005::/96 [110/1] via fe80::10:2, wlan0.10, 00:01:07
(...)
ROUTER_1#
```

Figura 3.10: Saída de "*show ipv6 route*" apresentando rotas estáticas

3.5 Testes com Injeção de Falhas

Através da simulação de certas ocorrências de falhas em uma rede Metro Ethernet, são apresentados casos possíveis em que estas podem acontecer:

- Recebimento de pacotes OSPF:
  - Pacote OSPFv3 malformado;
  - Campo com valor diferente do esperado;
  - Pacote OSPFv2;
- Erros de configuração:
  - Parâmetros de temporização;
  - Área;
  - *Router ID*;
- Variação no link:
  - Queda no *link*;

3.5.1 Recebimento de pacotes OSPF

O teste do tratamento na recepção dos pacotes OSPF é um item importante, visto que, ao cobrir a maior parte dos casos possíveis, podem ser evitados ataques maliciosos a roteadores que possuem OSPF habilitado.

Os itens a seguir são alguns dos tipos de pacotes OSPF que há possibilidade de estarem trafegando em uma rede real. Para estes casos, será utilizada a topologia da figura 3.4, ou seja, considerando somente um DUT: *ROUTER_2*.

O modo de geração desses pacotes OSPF é explicado no capítulo 4.

Caso A - Pacote OSPFv3 malformado

Setup:

1. Um DUT configurado com *Router ID* próprio, área 0 e interligado por um único *link* com o Gerenciador.
2. Enviar pacotes OSPFv3 *Hello* cujo cabeçalho esteja malformado. A falha escolhida se refere à ausência do campo "*reserved*", cujo tamanho é de 8 bits.
3. Gerenciador captura pacotes enviados pelo DUT.

Resultados esperados:

1. DUT continua a enviar pacotes, permanecendo estável.
2. É possível acessar via *telnet* o terminal do roteador.

Caso B - Campo com valor diferente do esperado

Setup:

1. Um DUT configurado com *Router ID* próprio, área 0 e interligado por um único *link* com o Gerenciador.
2. Enviar pacotes OSPFv3 *Hello* no qual algum campo da área de dados esteja com valor diferente do esperado. Para esse caso, optou-se por enviar o valor "0x10000000" (notação IPv4 "16.0.0.0") no campo *Area ID*.
3. Gerenciador captura pacotes enviados pelo DUT.

Resultados esperados:

1. DUT continua a enviar pacotes, permanecendo estável.
2. É possível acessar via *telnet* o terminal do roteador.

Caso C - Pacote OSPFv2

Setup:

1. Um DUT configurado com *Router ID* próprio, área 0 e interligado por um único *link* com o Gerenciador.
2. Enviar pacotes OSPFv3 *Hello* com o conteúdo do campo "versão" do cabeçalho IPv6 com o valor igual a "2".
3. Gerenciador captura pacotes enviados pelo DUT.

Resultados esperados:

1. DUT continua a enviar pacotes, permanecendo estável.
2. É possível acessar via *telnet* o terminal do roteador.

3.5.2 Erros de configuração no OSPF

Dois roteadores vizinhos não irão formar adjacência caso alguns parâmetros do pacote *Hello* discordarem; não formando, assim, adjacência entre eles. Por exemplo, poderia ocorrer a configuração errônea, em um dos roteadores, destas variáveis: Área *ID*, *Hello Interval* e *Dead Interval*. Poderia ainda haver discordância, entre os roteadores, quanto aos bits ‘E’ e ‘N’, do campo "opções" do pacote *Hello*, porém estes não serão considerados no teste.

Caso A - Parâmetros de temporização

Setup:

1. Dois DUTs configurados com *Router ID* próprio, na mesma área (0) e interligados por um único *link*.
2. Configurar valores diferentes para cada DUT (roteador) nos parâmetos *Hello Interval* (5 e 6 segundos) e *Dead Interval* (20 e 24 segundos, respectivamente para o *ROUTER_1* e *ROUTER_2*).
3. Gerenciador permanece durante 1900 segundos capturando pacotes trafegados entre os roteadores.

Resultados esperados:

1. Os parâmetros são aceitos na configuração, verificando que não foram retornados erros.

2. Não deve haver nenhum outro pacote OSPF gerado pelos DUTs (exceto *Hello*).

Caso B - Área

Setup:

1. Dois DUTs configurados com *Router ID* próprio, mesmos valores de temporização e interligados por um único *link*.

2. Configurar valores diferentes de Área *ID* diferente de "0.0.0.0" para cada DUT (roteador).

3. Gerenciador captura durante 1900 segundos pacotes trafegados entre os roteadores.

Resultados esperados:

1. Os parâmetros são aceitos na configuração, verificando que não foram retornados erros.

Caso C - *Router ID*

Setup:

1. Dois DUTs configurados com os mesmos valores de temporização, mesma área (0) e interligados por um único *link*.

2. Configurar o mesmo *Router ID* em ambos os roteadores.

3. Gerenciador captura durante 1900 segundos pacotes trafegados entre os roteadores.

Resultados esperados:

1. Os parâmetros são aceitos na configuração, verificando que não foram retornados erros.

2. Não deve haver nenhum outro pacote OSPF gerado pelos DUTs (exceto *Hello*).

3.5.3 Variações no enlace

Nestes testes é acrescentando um *link* entre os roteadores, verificando o comportamento do OSPFv3 diante do rompimento de algum enlace.

Setup inicial:

1. Retornar à configuração estabelecida para os equipamentos *ROUTER_1* e *ROUTER_2* na seção 3.4.2.

2. A partir de uma rede na topologia em anel (conforme a figura 3.3), estabelecer a segunda conexão entre os roteadores *ROUTER_1* e *ROUTER_2*.

Caso A - Criação de novo enlace

Setup:

1. Configurar o OSPF nesse segundo enlace em ambos os roteadores:

```
ROUTER_1# configure terminal
ROUTER_1(config)# router ospf6
ROUTER_1(config-ospf6)# interface A20 area 0.0.0.0
```

Figura 3.11: Configuração de um segundo enlace

Resultados esperados:

1. Os roteadores vizinhos devem formar duas adjacências (uma para cada enlace), permanecendo ambas no estado *Full*.

Caso B - Queda no *link*

Setup:

1. Gerenciador captura pacotes trocados entre os DUTs.
2. A partir do teste anterior (item A), configurar *shutdown* na interface *B20*, através do *daemon zebra*.

```
ROUTER_2# configure terminal
ROUTER_2(config)# interface B20
ROUTER_2(config-if)# shutdown
```

Figura 3.12: Configuração para o rompimento do enlace

Resultados esperados:

1. Verifica-se no *ROUTER_1*, através da CLI, que a segunda adjacência, que estava estabelecida com a interface *B20* do *ROUTER_2*, é desfeita.

4 IMPLEMENTAÇÃO

4.1 Infraestrutura

4.1.1 Mapa Conceitual

O mapa conceitual contendo os elementos que compõem a implementação deste trabalho está na figura 4.1.

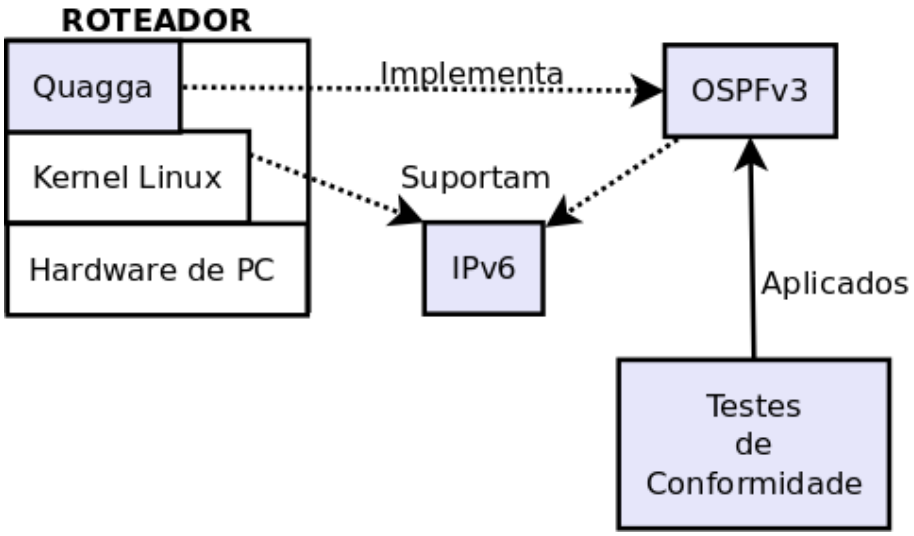

Figura 4.1: Mapa Conceitual

O **roteador** é o elemento composto pelo hardware de um computador pessoal, o ***Kernel Linux*** e pelo software de roteamento ***Quagga***, o qual é detalhado na seção 4.1.4. Dentre as suas implementações, está o **OSPFv3**, que contém totalmente suporte ao **IPv6**.

Os casos dos **testes de conformidade** estão descritos na seção 2.1.1.

4.1.2 Máquinas Utilizadas

A máquina que representa o "Gerenciador" é caracterizada pela seguinte descrição:

- Sistema Operacional: GNU/Linux 2.6.32-35-generic - Ubuntu 10.04
- CPU: Intel Core i3-2310M CPU - 2.10GHz - 4 cores
- Memória: 4GB

O outro computador é uma máquina virtual hospedada no PC mencionado anteriormente, virtualizada com a aplicação *VirtualBox* (ORACLE, 2011a):

- Versão do *VirtualBox*: 4.1.6 r74727
- Sistema Operacional: GNU/Linux 2.6.32-35-generic - Ubuntu 10.04
- Memória: 512MB

Ambas as máquinas tiveram o papel de DUT. O "Gerenciador" teve seu uso estendido para também operar como um roteador. O computador denominado "PC Virtual" será tratado aqui como ***ROUTER_2***. Quando o "Gerenciador" tem o papel de roteador, é chamado de ***ROUTER_1***.

4.1.3 Interfaces de Rede

O computador "Gerenciador" utilizou a interface de rede denominada *wlan0*, enquanto que o "PC Virtual" criou a interface de rede *eth1*; esta, por sua vez, é uma interface do tipo "*bridge*".

No sistema operacional hospedeiro, o *VirtualBox* utiliza o *driver* "*net filter*", o qual filtra os dados da interface de rede física escolhida, *wlan0*. Assim, permitindo ao *VirtualBox* receber e enviar dados filtrados na interface *wlan0*; criando, de fato, uma interface de rede virtual (ORACLE, 2011b). Portanto, as interfaces de rede do "PC Virtual" e do "Gerenciador" trocam dados como se estivessem fisicamente interconectadas.

4.1.3.1 Virtual Local Area Network - VLAN

A fim de permitir um isolamento na comunicação entre as interfaces de rede de máquina, não havendo interferência de outros tipos de tráfego, optou-se pelo uso da funcionalidade do Unix *vconfig*, a qual permite configurar uma VLAN (802.1Q) (IEEE802.1Q, 2006) através da criação de um dispositivo de rede virtual.

Tal configuração é feita seguindo as etapas abaixo:

- Carregar o módulo do *Kernel* referente a VLAN: **modprobe 8021q**
- Atribuir um identificador para a nova interface virtual:
  sudo vconfig add *<interface_real> <ID>*
- Alterar o endereço MAC (*Medium Access Control*) para um valor distinto:
  sudo ip link set *<interface_real>.<ID>* address *<endereço_MAC>*

A tabela 4.1 apresenta as interfaces reais utilizadas em cada equipamento, assim como o ID da VLAN e o endereço MAC configurado em cada interface virtual:

Tabela 4.1: Interfaces no nível de enlace

Equipamento	**Interface (mnemônico)**	***<interface_real>.<ID>***	***<endereço_MAC>***
"Gerenciador"	A10	wlan0.10	00:00:00:00:10:01
	A20	wlan0.20	00:00:00:00:20:01
	A30	wlan0.30	00:00:00:00:30:01
	A40	wlan0.40	00:00:00:00:40:01
"PC Virtual"	B10	eth1.10	00:00:00:00:10:02
	B20	eth1.20	00:00:00:00:20:02
	B30	eth1.30	00:00:00:00:30:02
	B40	eth1.40	00:00:00:00:40:02

Por exemplo, os parâmetros apresentados na figura 4.2 são utilizadas para configurar a VLAN 10 no "PC Virtual":

```
modprobe 8021q
sudo vconfig add eth1 10
sudo ip link set eth1.10 address 00:00:00:00:10:02
```

Figura 4.2: Configuração de VLAN

4.1.3.2 Endereços IPv6

Conforme introduzido na seção 2.1.1 e apresentado na seção 3.3, referente à especificação dos testes, os endereços IPv6 utilizados neste trabalho seguem dois formatos: *Unique Local IPv6 Unicast Addresses* (rede FC00::/7) e *Link-Local Unicast* (rede FE80::/10).

A tabela 3.1 apresentou os endereços IPv6 que deveriam ser configurados em cada interface do roteador. Considerando o nome das interfaces mencionados na seção anterior, a configuração real dos endereços IPv6 foi executada através dos seguintes comandos no *Linux*:

- Tornar ativa a interface virtual: **sudo ifconfig *<interface_real>.<ID>* up**
- Configurar o endereço *Unique Local IPv6 Unicast Addresses*:
  sudo ifconfig *<interface_real>.<ID>* add fc00::*<ID>*:1/120
- Configurar o endereço *Link-Local Unicast*:
  sudo ifconfig *<interface_real>.<ID>* add fe80::*<ID>*:1/100
- Remover o endereço IPv6 *Link-Local Unicast* que é configurado automaticamente ao tornar ativa a interface virtual: **sudo ifconfig wlan0.10 del fe80::[...]/64**

Exemplificando os itens acima, os comandos da figura 4.3 são executados para a VLAN 10 no "Gerenciador":

```
sudo ifconfig wlan0.10 up
sudo ifconfig wlan0.10 add fc00::10:1/120
sudo ifconfig wlan0.10 add fe80::10:1/100
sudo ifconfig wlan0.10 del fe80::6aa3:c4ff:fec1:5ed2/64
```

Figura 4.3: Configuração de endereços IPv6 em uma VLAN

4.1.4 Quagga

Os arquivos fontes do *Quagga* podem ser obtidos através do *site* oficial do projeto (QUAGGA, 2011). Neste trabalho, foi utilizada a última versão estável: "0.99.20". Após compilá-lo no "Gerenciador"e no "PC Virtual", já é possível executar os arquivos binários *zebra* e *ospf6d*, conforme será explicado posteriormente.

4.1.4.1 Configuração

Para habilitar o roteamento IPv6, é necessário configurar o parâmetro "*ipv6 forwarding*" no *vty* do *zebra*.

No *ospf6d*, a fim de indicar a este *daemon* a existência de uma interface, configura-se "*interface dummy*". Os argumentos do OSPF parametrizáveis em uma interface possuem os valores apresentados na figura 4.4 quando eles estão com a configuração padrão:

```
ROUTER_1# show running-config
(...)
!
interface dummy
ipv6 ospf6 cost 1
ipv6 ospf6 hello-interval 10
ipv6 ospf6 dead-interval 40
ipv6 ospf6 retransmit-interval 5
ipv6 ospf6 priority 1
ipv6 ospf6 transmit-delay 1
ipv6 ospf6 instance-id 0
!
ROUTER_1#
```

Figura 4.4: Valor padrão de alguns parâmetros do OSPF

4.1.4.2 Habilitando os Serviços

O *zebra* é habilitado da seguinte forma:

sudo zebra -u root -g root -f <caminho_zebra.conf> -A <IPv6_Link-local>

Habilita-se o *ospf6d* de maneira similar:

sudo ospf6d -u root -g root -A -f <caminho_ospf6d.conf> <IPv6_Link-local>

O usuário e grupo *root* são indicados a fim de permitir acesso em nível "superusuário" do *Linux*. O parâmetro "A" é utilizado para identificar o endereço pelo

qual a interface do serviço estará disponível, enquanto que a opção "f" indica o caminho do arquivo de configuração.

A interface de linha de comando para cada *daemon* é acessada através de *telnet* para o endereço <*IPv6_Link-local*> em uma das portas especificadas na tabela 2.5, de acordo com o serviço.

4.1.5 Ferramentas do Sistema

O *setup* do sistema é composto das seguintes ferramentas e bibliotecas, as quais podem ser obtidas através da funcionalidade do Ubuntu "*apt-get*":

- *ruby*: interpretador *Ruby* versão 1.8.7;
- *rubygems*: gerenciador de pacotes do Ruby;
- *modprobe 8021q*: módulo do *Kernel* referente a VLAN;
- *tcpdump*: ferramenta de captura de tráfego de rede;
- *tshark*: analisador de tráfego de rede;
- *racket*: biblioteca *Ruby* para geração de pacotes.

4.2 Arquitetura de Desenvolvimento

Ruby foi a linguagem escolhida para o desenvolvimento dos testes de conformidade do OSPFv3. Ela é orientada a objeto e interpretada, sendo executada similarmente a um *script*. A versão utilizada é a 1.8.7. Possui um alto poder de abstração e contém diversas bibliotecas, entre elas, as que auxiliam na implementação da geração e captura dos pacotes, a fim de validá-los.

A arquitetura de desenvolvimento para a implementação é composta por distintas classes, as quais foram divididas em dois módulos principais:

- *Framework*: comunicação e interação com o *Quagga*, verificação dos pacotes do OSPF e utilização de ferramentas *Unix*.
- *Tests*: contém todos os testes de conformidade propriamente ditos, os quais herdam propriedades da classe "*Test::Unit::TestCase*", a qual será explicada posteriormente.

Alguns módulos e sua inter-relação estão apresentados na figura 4.5.

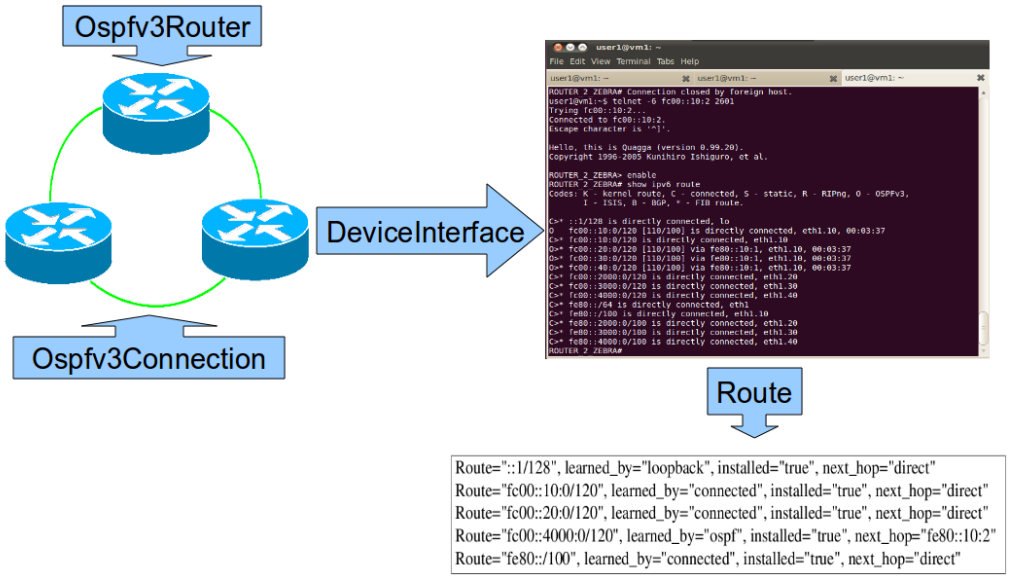

Figura 4.5: Módulos do sistema

4.2.1 *Framework*

As classes do *Framework* relativas ao roteador e ao PC, respectivamente, estão definidas como *DeviceInterface* e *PcInterface*.

Foram implementadas algumas classes para controle das rotas e do processo de roteamento do OSPFv3, denominadas: *Ospfv3Router*, *Ospfv3Connection* e *Route*.

A verificação dos pacotes OSPF é feita através dos módulos "*Captura de Pacotes*" e "*Geração de Pacotes*".

Alguns tipos de erros são tratados pela classe *FrameworkError*.

4.2.1.1 DeviceInterface

Ao instanciar um novo objeto dessa classe, é estabelecida uma sessão *Telnet* com o *daemon zebra* ou *ospf6d* através de endereço IPv6 e porta TCP específicos. Assim, os métodos implementados permitem executar qualquer comando dentro do *vty* desses serviços do *Quagga*.

Dependendo do comando desejado, é encaminhado ao *daemon zebra* ou *ospf6d*.

4.2.1.2 PcInterface

O "Gerenciador" tem um objeto instanciado dessa classe, pois ela tem métodos típicos de um computador *Unix*: captura de pacotes em uma interface de rede, cópia de arquivos, configuração de interface de rede e comandos de teste de conectividade (por exemplo, *ping6*).

4.2.1.3 Ospfv3Router

Define um roteador OSPFv3, permitindo configurar todos os parâmetros internos à árvore de comandos "*router ospf6*" do *vty* do *daemon ospf6d*.

4.2.1.4 Ospfv3Connection

Cria uma conexão lógica entre dois *Ospfv3Routers*, pela qual é possível romper ou restabelecer logicamente um *link* entre dois roteadores.

4.2.1.5 Route

Simboliza uma rota em um roteador. A partir da saída do comando "*show ipv6 route*", cria um objeto para cada prefixo e indica se está diretamente conectada ou foi aprendida através do OSPF; nesse caso, apresenta também o *next hop* da rota.

A tabela de roteamento, na figura 4.6, é um resumo daquela apresentada no teste da seção 3.4.2:

```
ROUTER_1# show ipv6 route
Codes: K - kernel route, C - connected, S - static, R - RIPng, O - OSPFv3,
I - ISIS, B - BGP, * - FIB route.

C>* ::1/128 is directly connected, lo
C>* fc00::10:0/120 is directly connected, wlan0.10
C>* fc00::20:0/120 is directly connected, wlan0.20
O>* fc00::4000:0/120 [110/1] via fe80::10:2, wlan0.10, 00:32:23
C>* fe80::/100 is directly connected, wlan0.10
ROUTER_1#
```

Figura 4.6: Tabela de roteamento do *Quagga*

O mecanismo de *parsing* espera pelo tipo de rota (se foi aprendida pelo OSPF ou se é configurada localmente, por exemplo), se ela está instalada na tabela de roteamento do *Quagga* (indicada pelo sinal '>') e/ou na tabela de encaminhamento (*Forwarding Information Base* - FIB - indicado por '*'). Além disso, obtém o prefixo e seu *gateway* (também chamado "*next hop*"); este é armazenado como "*direct*" caso o prefixo esteja configurado em uma interface de rede do próprio roteador.

Dessa forma, os prefixos listados na figura 4.6 criam objetos do tipo *Route*, mostrados na figura 4.7:

```
Route="::1/128", learned_by="loopback", installed="true", next_hop="direct"
Route="fc00::10:0/120", learned_by="connected", installed="true", next_hop="direct"
Route="fc00::20:0/120", learned_by="connected", installed="true", next_hop="direct"
Route="fc00::4000:0/120", learned_by="ospf",installed="true",next_hop="fe80::10:2"
Route="fe80::/100", learned_by="connected", installed="true", next_hop="direct"
```

Figura 4.7: Objetos do tipo *Route*

4.2.1.6 Captura de Pacotes

A captura dos pacotes é feita pelo "Gerenciador". Além de executar o código *Ruby*, ele também verifica os pacotes trafegados entre os DUTs ou gerado por somente um DUT. A verificação é feita através do analisador de tráfego de rede *tcpdump* (JACOBSON; LERES; MCCANNE, 2011). A fim de filtrar isoladamente cada campo de um pacote,

é utilizada outra ferramenta: *tshark* (COMBS, 2011). Assim, é possível validar cada parâmetro de qualquer pacote OSPF.

O programa *tcpdump* é invocado por um método específico interno à classe *PcInterface*. A linha de execução é:

sudo /usr/sbin/tcpdump -s 1500 -U -i <interface> -w <arquivo_saída> ,

onde "-s 1500" é o tamanho máximo de cada *frame* capturado, "-U" e "-w" são utilizados para salvar a captura em "<arquivo_saída>" à medida que são obtidos dados.

A partir do arquivo "<arquivo_saída>", pode-se utilizar um filtro e aplicá-lo nesse arquivo, obtendo somente os pacotes desejados. Para isso, utiliza-se o programa *tshark*, o qual também é executado a partir da classe *PcInterface*, com os seguintes parâmetros:

sudo /usr/bin/tshark -r <arquivo_saída> -q -z 'io,phs,<filtro>' ,

no qual a opção "-r" é utilizada para ler o arquivo capturado anteriormente e "-q" faz exibir somente no final o total de pacotes. O parâmetro "-z 'io,phs,<filtro>'" cria uma estatística de hierarquia de protoco dos pacotes filtrados com o argumento "<filtro>".

Dessa forma, é possível selecionar, por exemplo, somente os pacotes *Hello* através do filtro "*ospf.msg == 1*", onde o campo "*Tipo de pacote*" (identificado hierarquicamente como "*ospf.msg*") do cabeçalho do pacote OSPFv3 contém o valor "1"; conforme será exposto, em seguida, na tabela 4.2.

Abaixo, são apresentados os valores utilizados na verificação de cada campo dos pacotes *Hello* e LSA indicado na seção 2.2.

- **Cabeçalho OSPFv3 - *Hello***:

Tabela 4.2: Campos do cabeçalho OSPFv3 e filtros

Campo	**Valor**	**Filtro utilizado**
Versão do OSPF	0x3	*frame[54:1] == 03*
Tipo de pacote	0x1	*ospf.msg == 1*
Router ID	10.1.1.101	*ospf.srcrouter == 10.1.1.101*
Area ID	0	*frame[62:4] == 00:00:00:00*
Instance ID	0x64 (100)	*frame[68:1] == 64*

- ***Hello***:

Tabela 4.3: Campos do pacote *Hello* e filtros

Campo	Valor	Filtro utilizado
Prioridade do roteador	0xfb (251)	*frame[74:1] == fb*
Intervalo *Hello*	0x0a (10)	*frame[78:2] == 00:0a*
Intervalo *Router Dead*	0x14 (20)	*frame[80:2] == 00:14*
DR	10.1.1.101	*frame[82:4] == 0a:01:01:65*
BDR	10.1.1.102	*frame[86:4] == 0a:01:01:66*
Neighbor ID	10.1.1.102	*frame[90:4] == 0a:01:01:66*

- **Cabeçalho OSPFv3 - *LSA***:
  Analisando a tabela 4.2, apenas o campo "Tipo de pacote" possui um valor diferente quando a mensagem é do tipo "LSA". Nesse caso, o valor é igual a "4".

- **LSA**:
  A seleção do campo "Tipo de LSA", que se encontra no início da área de dados do pacote OSPFv3 quando este é do tipo "LSA", é obtida através do filtro "*frame[76:2] == **<LS_type>***". Os valores do campo que identifica cada tipo de LSA (*LS_type*) foram vistos na seção 2.2.3. Além disso, também foi abordado que, para cada LSA, alguns campos seriam analisados. Assim, na tabela 4.4 seguem os valores desses campos.

Tabela 4.4: Campos dos pacotes *LSA* e filtros

Nome	**Campo**	**Valor**	**Filtro utilizado**
Router-LSA	*Neighbor Router ID*	10.1.1.101	*frame[110:4] == 0a:01:01:65*
Network-LSA	*Attached Router*	10.1.1.101	*frame[98:4] == 0a:01:01:65*
Network-LSA	*Attached Router*	10.1.1.102	*frame[102:4] == 0a:01:01:66*
Link-LSA	*Address Prefix*	fc::10::	*frame[122:16] == fc::10::*
Link-LSA	*PrefixLength*	120	*frame[118:1] == 78*
Link-LSA	*Link-local Interface Address*	fe:80::10:1	*frame[98:16] == fe:80::10:00:01*
Intra-Area-Prefix-LSA	*Address Prefix*	fc::10::	*frame[110:16] == fc::10::*
Intra-Area-Prefix-LSA	*PrefixLength*	120	*frame[106:1] == 78*

4.2.1.7 Geração de Pacotes

A geração de pacotes que são utilizados nos testes com injeção de falhas (conforme relatado na seção 3.5.1) é feita pela biblioteca *Racket* (HART, 2011), disponível para a linguagem *Ruby*. Com ela, é possível "fabricar" dados desde a camada 2 até a camada 5 e enviá-los pela interface de rede.

Os pacotes produzidos com ou sem falhas são sempre do tipo "*Hello*". Cada camada do pacote é especificada abaixo:

- Endereços MAC de origem e destino são, respectivamente, "00:00:00:00:10:01" e "33:33:00:00:00:05".
- Campos do IPv6:
  - IP de origem: "0xFE80000000000000000000000100001" (endereço IPv6 do tipo *Link-Local unicast*
  - IP de destino: "0xFF020000000000000000000000000005" (endereço IP multicast *AllSPFRouters*)
  - *Flow Label*: 0
  - *Trafic Class*: 0

- TTL: 0x40
- *Next Header Type*: 0x59 (OSPF)
- *Payload length*: 0x24 (36 bytes)

Quando são enviados sem falhas injetadas nos pacotes OSPFv3, as mensagens *Hello* contêm os valores apresentados na figura 4.8, os quais estão em formato do código *Ruby*. O valor de cada campo é indicado após o termo "*default*".

```
1  class OSPFv3HelloPacket < OSPFv3
2  # Cabecalho
3  unsigned :version, 8, {:default => 0x03}
4  unsigned :type, 8, {:default => 0x01}
5  unsigned :length, 16, {:default => 0x0024}
6  unsigned :srvrouter, 32, {:default => 0x0a010165}
7  unsigned :areaid, 32, {:default => 0x00000000}
8  unsigned :checksum, 16, {:default => 0xe6af}
9  unsigned :instanceid, 8, {:default => 0x00}
10 unsigned :reserved, 8, {:default => 0x00}
11 # Dados (Hello)
12 unsigned :interfaceid, 32, {:default => 0x00000003}
13 unsigned :routerprio, 8, {:default => 0x01}
14 unsigned :options, 24, {:default => 0x000013}
15 unsigned :hellointerval, 16, {:default => 0x000a}
16 unsigned :routerdeadinterval, 16, {:default => 0x0028}
17 unsigned :designatedrouter, 32, {:default => 0x0a010165}
18 unsigned :bkpdesignatedrouter, 32, {:default => 0x00000000}
19 # Payload
20 rest :payload
21 end
```

Figura 4.8: Classe que representa um pacote OSPFv3 *Hello*

As mensagens com injeção de falhas possuem diferentes formatos, a fim de explorar, de diversos modos, um possível defeito na implementação do OSPFv3 do *Quagga*.

Este primeiro caso representa a situação "**Pacote OSPFv3 malformado**", no qual o campo reservado de 8 bits do pacote *Hello* foi removido propositalmente, apresentado na figura 4.9.

```
1  class OSPFv3HelloPacketMalformed < OSPFv3
2  unsigned :version, 8, {:default => 0x03}
3  unsigned :type, 8, {:default => 0x01}
4  unsigned :length, 16, {:default => 0x0024}
5  unsigned :srvrouter, 32, {:default => 0x0a010165}
6  unsigned :areaid, 32, {:default => 0x00000000}
7  unsigned :checksum, 16, {:default => 0xe6af}
8  unsigned :instanceid, 8, {:default => 0x00}
9  unsigned :interfaceid, 32, {:default => 0x00000003}
10 unsigned :routerprio, 8, {:default => 0x01}
11 unsigned :options, 24, {:default => 0x000013}
12 unsigned :hellointerval, 16, {:default => 0x000a}
13 unsigned :routerdeadinterval, 16, {:default => 0x0028}
14 unsigned :designatedrouter, 32, {:default => 0x0a010165}
15 unsigned :bkpdesignatedrouter, 32, {:default => 0x00000000}
16 # Payload
17 rest :payload
18 end
```

Figura 4.9: Classe que representa um pacote OSPFv3 *Hello* sem o campo reservado

Neste outro caso, exposto na figura 4.10, o pacote estará somente com os seguintes campos modificados em relação ao apresentado na classe *OSPFv3HelloPacket*: área com valor hexadecimal igual a "0x10000000" (notação IPv4 "16.0.0.0") e, consequentemente, o *checksum* precisou ser alterado para "0xd6af".

```
1  class OSPFv3HelloPacketMalformed < OSPFv3
2  (...)
3  unsigned :areaid, 32, {:default => 0x10000000}
4  unsigned :checksum, 16, {:default => 0xd6af}
5  (...)
6  end
```

Figura 4.10: Classe que representa um pacote OSPFv3 *Hello* com outra *Area ID*

Similarmente ao caso anterior, para a situação em que é enviado o valor "2" na versão do OSPF, os campos "*version*" e "*checksum*" precisaram ser modificados.

4.2.1.8 FrameworkError

Filha da classe *StandardError*, é subdividida em outras subclasses customizadas para alguns tipos de erros mais comuns, adicionando informações relevantes para a inspeção deles.

Por exemplo, a subclasse *ArgumentError* verifica se o argumento passado a um método é algum valor esperado e, caso contrário, informa quais seriam as opções possíveis. Outra situação pode ser um erro ocasionado por um determinado pacote não ter chegado ao seu destino e, dessa forma, a exceção *VerificationError* é sinalizada, podendo ser agregada uma mensagem extra. Quando um comando inválido é executado no *Quagga*, é levantado o erro *CommandError*, indicando o comando que gerou tal exceção.

Os casos mencionados acima são classes filhas de *FrameworkError*.

4.2.2 *Tests*

Os testes de conformidade do OSPFv3 foram implementados de acordo com a estrutura cujo esboço está na seção 4.2.2.2. Eles utilizam ferramentas do módulo *Test::Unit*, que é descrito na seção seguinte.

4.2.2.1 Test::Unit

O *Test::Unit* (TALBOTT, 2011) é um *framework Ruby* para testes unitários, o qual auxilia no desenvolvimento, depuração e avaliação de funcionalidades. É útil, tanto para verificar códigos programados na própria linguagem *Ruby*, quanto para avaliar ferramentas e produtos escritas em outras linguagens de programação. Um exemplo é deste trabalho, o qual executa testes de conformidade em um software (*Quagga*) desenvolvido na linguagem C.

A ideia de um teste unitário é escrever métodos que façam algumas asserções (*Assertions*) contra funcionalidades, muitas vezes, previamente desenvolvidas. Tais métodos podem ser agrupados em uma chamada "suíte de testes" (*Test Suite*) e, então, serem executados. Os resultados são reunidos em uma estrutura de "resultados de testes" (*Test Result*") e exibidos ao usuário.

"Asserção" ou "afirmação" é a declaração de um resultado esperado. Por exemplo, se *a* é 5 e *b* é 6, uma *assertion* "*b* é maior do que *a*" é verdadeira. Logo após, porém, se essas variáveis permutam entre si os seus valores, a mesma asserção será falsa, propagando um erro com alguma informação pertinente.

Os resultados de "passou" ou "falhou" das *Assertions* são utilizados dentro do contexto de um teste, o qual é implementado por um método sempre denominado com o prefixo "*test*" (por exemplo, "*test_01_ospfv3_check_configuration*"). Reunindo diversos testes que possuem verificações em comum, é possível criar, então, uma classe, a qual herdará propriedades da classe *Test::Unit::TestCase*.

A classe *TestCase* engloba uma coleção de métodos de teste, tais como:

- *setup()*: chamada antes de todo método "*test...*".
- *run()*: executa o método teste, coletando estatísticas (por exemplo, tempo total de execução e contabilidade de acertos e falhas) e erros.
- *teardown()*: chamada após todo método "*test...*".

Os métodos *setup()* e *teardown()* são definidos, inicialmente, sem conteúdo; assim, podem-se adequar suas implementações conforme o objetivo de um método de teste.

As falhas, erros e estatísticas coletadas de uma suíte de testes são informadas ao usuário através de um objeto da classe *TestResult*.

4.2.2.2 Testes do OSPFv3

A classe *Ospfv3Sample*, na figura 4.11, exemplifica um teste de conformidade que possui herança da classe *Test::Unit::TestCase*. Ela é implementada pelos métodos: *setup*, *n* métodos "*test_...*" e *teardown*.

O método *setup* obtém a conexão com o *vty* de cada roteador e configura o parâmetro "*ipv6 forwarding*", que é comum a todos os roteadores.

Cada método "*test_...*" é executado em ordem alfabética e numérica, o que justifica a nomenclatura escolhida conter um número logo após o prefixo "*test_*", facilitando o ordenamento desejado.

```
1  # Carrega o modulo Test::Unit
2  require 'test/unit'
3
4  module Test
5    class Ospfv3Sample < Test::Unit::TestCase
6      def setup
7        # Estabelece conexoes
8        # Configura servicos
9      end
10
11     def test_01_ospfv3_one_router
12       configure_ospf_parameters
13       verify_hello_packets
14     end
15
16     def test_02_ospfv3_two_routers
17       configure_ospf_parameters
18       verify_lsa_packets
19     end
20
21     def teardown
22       # Remove configuracoes
23     end
24   end
25 end
```

Figura 4.11: Classe com o esboço de um teste

4.2.3 Execução de um Teste

Um exemplo de como executar um teste é apresentado na figura 4.12. Nesse caso, o comando "*ipv6 ospf6 priority 250*" foi executado de modo inapropriado, ocasionando um erro no resultado.

```
petry@ubuntu: /tg2impl$ ruby lib/test/ospfv3_sample.rb
Loaded suite lib/test/ospfv3_sample
Started
E
Finished in 0.298869 seconds.

1) Error:
test_01(Test::Ospfv3):
Framework::CommandInvalid: ROUTER_1: ‘ipv6 ospf6 priority 250’
(...)
lib/test/ospfv3_sample.rb:42:in ‘verify_01_hello_parameters_cli’
lib/test/ospfv3_sample.rb:104:in ‘test_01’

1 tests, 0 assertions, 0 failures, 1 errors
```

Figura 4.12: Exemplo de execução de um teste retornando erro

Novamente, é executado o mesmo código com a devida correção; agora, executando com sucesso, conforme a figura 4.13 exibe.

```
petry@ubuntu: /tg2impl$ ruby lib/test/ospfv3_sample.rb
Loaded suite lib/test/ospfv3_sample
Started

Finished in 13.192919 seconds.

1 tests, 11 assertions, 0 failures, 0 errors
```

Figura 4.13: Exemplo de execução de um teste retornando sucesso

A explicação das linhas executadas nos dois exemplos anteriores estão descritas abaixo:

- "**ruby lib/test/ospfv3_sample.rb**":
  Ruby executa o arquivo *ospfv3_sample.rb*.
- "**Loaded suite lib/test/ospfv3_sample**":
  Apesar de poder ser agrupado em um conjunto de testes a serem executados, formando uma *Test Suite*, nesse caso é carregado apenas um único teste.
- "**E**":
  Indica que ocorreu um erro durante o teste
- " **1) Error: [...]**":
  No caso descrito no item acima, quando ocorre erro, são impressos a pilha na ordem (de baixo para cima) da execução do arquivo *ospfv3_sample.rb* e a especificação do erro.
- "**Finished in S seconds**":
  Informação padrão originada do módulo *Test::Unit*, informando o tempo total de execução do código.

- "**1 tests, X assertions, Y failures, Z errors**":
  Estatísticas exibidas pela classe *TestResult*.

A execução dos testes, especificados no capítulo 3 e implementados conforme descrito, está relatada no próximo capítulo.

5 EXECUÇÕES E RESULTADOS DOS TESTES

Neste capítulo, serão apresentados os resultados da execução dos casos de testes especificados no capítulo 3.

5.1 Testes com 1 Roteador - Pacotes *Hello*

O primeiro teste executado teve sucesso em seu objetivo: sendo possível configurar o *ROUTER_1*, verificar os dados através de comandos no *vty* e capturar pacotes *Hello* conforme o esperado.

Aproximadamente 32 segundos foram necessários para o código ser executado, considerando que 30 segundos foi o tempo estipulado para capturar alguns pacotes através do *tcpdump*.

Na figura 5.1 é apresentada a saída completa da execução.

```
petry@ubuntu: /tg2impl$ ruby lib/test/ospfv3.rb
Loaded suite lib/test/ospfv3
Started
**************************************************
1) Configuring OSPFv3 parameters
--------------------------------------
Checking configured OSPFv3 parameters
**************************************************
2) Verifying OSPFv3 Hello packets
--------------------------------------
Waiting 30 seconds in order to capture Hello packets...
Total: 13 OSPFv3 Hello packets.
**************************************************
.
Finished in 32.461969 seconds.

1 tests, 8 assertions, 0 failures, 0 errors
petry@ubuntu: /tg2impl$
```

Figura 5.1: Teste com 1 roteador - pacotes *Hello*

5.2 Pacotes *Link-State Advertisements* - 2 Roteadores

Devido à espera de 1900 segundos para garantir a troca de pelo menos um conjunto de LSAs entre os dois roteadores, o tempo total do teste foi de 1966 segundos. A esse tempo, estão incluídos os 30 segundos referentes à captura dos pacotes *Hello* (comentados na seção anterior - 5.1) e do período para a topologia convergir, estipulado também em meio minuto.

Na implementação deste caso, em relação ao teste anterior, a novidade foi a obtenção da tabela de roteamento do roteador e a sua posterior comparação com os valores esperados. Tal verificação está nos itens enumerados com prefixo o "2", na figura 5.2 Observa-se que os prefixos "fc00::2000:0/120", "fc00::3000:0/120" e "fc00::4000:0/120", configurados no *ROUTER_2*, foram aprendidos pelo *ROUTER_1* através do OSPF.

```
petry@ubuntu: /tg2impl$ sudo ruby lib/test/ospfv3_two_routers.rb
Loaded suite lib/test/ospfv3_two_routers
Started
************************************************
1.A) Configuring OSPFv3 parameters in ROUTER_1
------------------------------------
Checking configured OSPFv3 parameters
************************************************
1.B) Configuring OSPFv3 parameters in ROUTER_2
------------------------------------
Checking configured OSPFv3 parameters
************************************************
Waiting 30 seconds for topology convergence...
************************************************
2.A) Getting routing table in ZEBRA_1
------------------------------------
Route ::1/128 learned_by=loopback installed=true next_hop=direct
Route fe80::/100 learned_by=connected installed=true next_hop=direct
Route fe80::/100 learned_by=connected installed=true next_hop=direct
Route fe80::/100 learned_by=connected installed=true next_hop=direct
Route fe80::/100 learned_by=connected installed=true next_hop=direct
Route fc00::4000:0/120 learned_by=ospf installed=true next_hop=fe80::10:2
Route fc00::10:0/120 learned_by=connected installed=true next_hop=direct
Route fc00::20:0/120 learned_by=connected installed=true next_hop=direct
Route fc00::3000:0/120 learned_by=ospf installed=true next_hop=fe80::10:2
Route fc00::40:0/120 learned_by=connected installed=true next_hop=direct
Route fc00::2000:0/120 learned_by=ospf installed=true next_hop=fe80::10:2
Route fc00::30:0/120 learned_by=connected installed=true next_hop=direct
Route 1001::/96 learned_by=ospf installed=true next_hop=fe80::10:2
Route 1002::/96 learned_by=ospf installed=true next_hop=fe80::10:2
Route 1003::/96 learned_by=ospf installed=true next_hop=fe80::10:2
Route 1004::/96 learned_by=ospf installed=true next_hop=fe80::10:2
Route 1005::/96 learned_by=ospf installed=true next_hop=fe80::10:2
------------------------------------
```

```
2.B.1) Route fc00::2000:0/120 learned_by=ospf installed=true next_hop=fe80::10:2
IS present in ROUTER_1
2.B.2) Route fc00::3000:0/120 learned_by=ospf installed=true next_hop=fe80::10:2
IS present in ROUTER_1
2.B.3) Route fc00::4000:0/120 learned_by=ospf installed=true next_hop=fe80::10:2
IS present in ROUTER_1
_____________________________________
**********************************************
3) Verifying OSPFv3 Hello packets
_____________________________________
Waiting 30 seconds in order to capture Hello packets...
Total: 13 OSPFv3 Hello packets.
**********************************************
4) Verifying OSPFv3 LSA packets
_____________________________________
Waiting 1900 seconds in order to capture LS Update packets...
Remaining 950 seconds...
Total: - OSPFv3 LS Update packets.
**********************************************
.
Finished in 1966.32478 seconds.

1 tests, 21 assertions, 0 failures, 0 errors
petry@ubuntu: /tg2impl$
```

Figura 5.2: Teste com 2 roteadores - pacotes LSAs

5.3 Aprendizado de Rotas Estáticas

Este teste verifica, similarmente ao caso anterior (seção 5.2), se prefixos divulgados pelo roteador vizinho são corretamente aprendidos.

A diferença, contudo, está tanto em configurar a redistribuição de rotas estáticas no roteador "anunciante" das informações quanto em não conferir se os pacotes OSPF estão com os dados corretos. Esse último item é justificado por já ter sido feita essa verificação em outros casos de teste. Assim, tendo o propósito de apenas conferir a tabela de roteamento do roteador receptor das LSAs com as rotas estáticas configuradas no roteador vizinho.

A execução leva praticamente o tempo de espera da convergência; estimada, novamente, em 30 segundos. Os cinco prefixos divulgados pelo *ROUTER_2* são corretamente aprendidos pelo *ROUTER_1*, conforme está na figura 5.3.

```
petry@ubuntu: /tg2impl$ ruby lib/test/ospfv3_static.rb
Loaded suite lib/test/ospfv3_static
Started
************************************************
1.A) Configuring OSPFv3 'redistribute connected/static' parameters in ROUTER_1
************************************************
1.B) Configuring OSPFv3 'redistribute connected/static' parameters in ROUTER_2
************************************************
2) Configuring static routes in ZEBRA_2
************************************************
Waiting 30 seconds for topology convergence...
************************************************
3.A) Getting routing table in ZEBRA_1
------------------------------------
Route ::1/128 learned_by=loopback installed=true next_hop=direct
Route fe80::/100 learned_by=connected installed=true next_hop=direct
Route fe80::/100 learned_by=connected installed=true next_hop=direct
Route fe80::/100 learned_by=connected installed=true next_hop=direct
Route fe80::/100 learned_by=connected installed=true next_hop=direct
Route fc00::4000:0/120 learned_by=ospf installed=true next_hop=fe80::10:2
Route fc00::10:0/120 learned_by=connected installed=true next_hop=direct
Route fc00::20:0/120 learned_by=connected installed=true next_hop=direct
Route fc00::3000:0/120 learned_by=ospf installed=true next_hop=fe80::10:2
Route fc00::40:0/120 learned_by=connected installed=true next_hop=direct
Route fc00::2000:0/120 learned_by=ospf installed=true next_hop=fe80::10:2
Route fc00::30:0/120 learned_by=connected installed=true next_hop=direct
Route 1001::/96 learned_by=ospf installed=true next_hop=fe80::10:2
Route 1002::/96 learned_by=ospf installed=true next_hop=fe80::10:2
Route 1003::/96 learned_by=ospf installed=true next_hop=fe80::10:2
Route 1004::/96 learned_by=ospf installed=true next_hop=fe80::10:2
Route 1005::/96 learned_by=ospf installed=true next_hop=fe80::10:2
------------------------------------
```

```
3.B.1) Route 1001::/96 learned_by=ospf installed=true next_hop=fc00::2000:10
IS present in ROUTER_1
3.B.2) Route 1002::/96 learned_by=ospf installed=true next_hop=fc00::2000:10
IS present in ROUTER_1
3.B.3) Route 1003::/96 learned_by=ospf installed=true next_hop=fc00::2000:10
IS present in ROUTER_1
3.B.4) Route 1004::/96 learned_by=ospf installed=true next_hop=fc00::2000:10
IS present in ROUTER_1
3.B.5) Route 1005::/96 learned_by=ospf installed=true next_hop=fc00::2000:10
IS present in ROUTER_1
**********************************************
.
Finished in 30.471621 seconds.

1 tests, 0 assertions, 0 failures, 0 errors
petry@ubuntu: /tg2impl$
```

Figura 5.3: Teste com 2 roteadores - rotas estáticas

5.4 Simulações de Falha

5.4.1 OSPF - Recebimento de pacotes

A execução, apresentada na figura 5.4, contempla as seguintes especificações:

- Caso A - Pacote OSPFv3 malformado
- Caso B - Campo com valor diferente do esperado
- Caso C - Pacote OSPFv2

A fim de haver permissão para o envio de pacotes pela interface de rede, é necessário executar o código *Ruby* precedido por "*sudo*", obtendo privilégio de usuário *root*.

O período de 15 segundos para aguardar pelo menos um pacote *Hello* correto mostrou-se suficiente. Assim, além da confirmação de envio pacotes originados pelo *ROUTER_2*, a sua verificação de integridade foi comprovada através da execução do comando "*show running-config*", como previsto.

Os pacotes malformados não foram contabilizados na linha que informa o total de pacotes *Hello* capturados, visto que atuaram os filtros apresentados nas tabelas 4.2 e 4.3.

```
petry@ubuntu: /tg2impl$ sudo ruby lib/test/ospfv3_failure_1.rb
Loaded suite lib/test/ospfv3_failure_1
Started
**********************************************
1) Sending a correct OSPFv3 Hello packet
Checking integrity of ROUTER_2
Waiting 15 seconds in order to capture Hello packets...
Total: 1 OSPFv3 Hello packets.
**********************************************
2) Sending OSPFv3 Hello packet with different area
Checking integrity of ROUTER_2
Waiting 15 seconds in order to capture Hello packets...
Total: 1 OSPFv3 Hello packets.
**********************************************
3) Sending a malformed OSPFv3 Hello packet
Checking integrity of ROUTER_2
Waiting 15 seconds in order to capture Hello packets...
Total: 1 OSPFv3 Hello packets.
**********************************************
4) Sending a OSPFv3 Hello packet with version 2
Checking integrity of ROUTER_2
Waiting 15 seconds in order to capture Hello packets...
Total: 2 OSPFv3 Hello packets.
.
Finished in 87.061204 seconds.

1 tests, 32 assertions, 0 failures, 0 errors
petry@ubuntu: /tg2impl$
```

Figura 5.4: Simulações de falha - recebimento de pacotes OSPF *Hello*

5.4.2 Erros de configuração no OSPF

5.4.2.1 Caso A - Parâmetros de temporização

A configuração de tempo dos intervalos de *Hello* e *Router Dead* com valores diferentes para cada um dos roteadores foi bem sucedida, cuja execução é exibida na figura 5.5. Assim, comprovando que não foi estabelecida adjacência entre eles, pois não trafegaram pacotes LSA durante 1900 segundos, apenas pacotes *Hello*.

```
petry@ubuntu: /tg2impl$ sudo ruby lib/test/ospfv3_failure_2.rb
Loaded suite lib/test/ospfv3_failure_2
Started
**************************************************
1.A) Configuring Router ID and interface wlan0.10 in ROUTER_1
**************************************************
2.A) Configuring Hello Interval 5s and Router Dead Interval 20s in ROUTER_1
**************************************************
1.B) Configuring Router ID and interface eth1.10 in ROUTER_2
**************************************************
2.B) Configuring Hello Interval 6s and Router Dead Interval 24s in ROUTER_2
**************************************************
3) Verifying that only exists OSPFv3 Hello packets
Waiting 1900 seconds in order to capture Hello packets...
Remaining 950 seconds...
Total: 695 OSPFv3 Hello packets.
Total: 0 LS Updated packets.
.
Finished in 1902.451949 seconds.

1 tests, 2 assertions, 0 failures, 0 errors
petry@ubuntu: /tg2impl$
```

Figura 5.5: Simulações de falha - parâmetros de temporização divergentes

5.4.2.2 Caso B - Área

Dois roteadores que possuem um par de enlaces interligados devem ter suas respectivas interfaces configuradas na mesma área.

Assim como esperado, não houve formação de adjacência entre os dois roteadores, pois a interface de cada um deles estava configurada em áreas distintas. A figura 5.6 apresenta a execução do teste.

```
petry@ubuntu: /tg2impl$ ruby lib/test/ospfv3_failure_3.rb
Loaded suite lib/test/ospfv3_failure_3
Started
************************************************
1.A) Configuring 'interface wlan0.10 area 1.0.0.0' in ROUTER_1
************************************************
1.B) Configuring 'interface eth1.10 area 2.0.0.0' in ROUTER_2
************************************************
2) Verifying that only exists OSPFv3 Hello packets
Waiting 1900 seconds in order to capture Hello packets...
Remaining 950 seconds...
Total: 381 OSPFv3 Hello packets.
Total: 0 LS Updated packets.
.
Finished in 1903.136849 seconds.

1 tests, 2 assertions, 0 failures, 0 errors
petry@ubuntu: /tg2impl$
```

Figura 5.6: Simulações de falha - diferentes valores de área

5.4.2.3 Caso C - Router ID

Concluindo os testes relativos a erros de configuração, foi executado o caso em que é atribuído o mesmo *Router ID* a dois roteadores vizinhos (apresentado na figura 5.7). De modo semelhante aos dois testes anteriores, não formou-se adjacência entre eles.

```
petry@ubuntu: /tg2impl$ ruby lib/test/ospfv3_failure_4.rb
Loaded suite lib/test/ospfv3_failure_4
Started
**********************************************
1) Configuring the same Router ID 10.1.1.101 in both routers
**********************************************
2) Verifying that only exists OSPFv3 Hello packets
Waiting 1900 seconds in order to capture Hello packets...
Remaining 950 seconds...
Total: 380 OSPFv3 Hello packets.
Total: 0 LS Updated packets.
.
Finished in 1901.485915 seconds.

1 tests, 2 assertions, 0 failures, 0 errors
petry@ubuntu: /tg2impl$
```

Figura 5.7: Simulações de falha - mesmo *Router ID*

5.4.3 Variações no enlace

Na execução do caso "A", é estabelecida a adjacência entre um par de roteadores interligados através de dois enlaces. O rompimento de um dos *links* é apresentada no caso "B". Os testes estão na figura 5.8. Os dois itens foram implementados em forma sequencial:

- Caso A - Criação de novo enlace
- Caso B - Queda no *link*

O funcionamento ocorreu conforme esperado. Primeiramente, convergindo a topologia em um tempo estimado de 60 segundos. Após configurar em *shutdown* a interface "*eth1.20*" (chamada de "*B20*" na especificação do teste na seção 3.5.3), o intervalo *Router Dead*, configurado em 40 segundos, precisou ser aguardado para verificar o desaparecimento do roteador vizinho somente no enlace que teve a interface derrubada.

```
petry@ubuntu: /tg2impl$ ruby lib/test/ospfv3_two_routers_ring.rb
Loaded suite lib/test/ospfv3_two_routers_ring
Started
************************************************
1.A) Configuring OSPFv3 interfaces ["wlan0.10", "wlan0.20"] in ROUTER_1
************************************************
1.B) Configuring OSPFv3 interfaces ["eth1.10", "eth1.20"] in ROUTER_2
************************************************
Waiting 60 seconds for topology convergence...
************************************************
2) Neighbors:
Router ID 10.1.1.102: Link-local address fe80::10:2, State is Full
Router ID 10.1.1.102: Link-local address fe80::2000:2, State is Full
************************************************
Must have 2 neighbors: OK!
************************************************
3) Shutting down interface eth1.20 in ROUTER_2
************************************************
Waiting 60 seconds for removing neighbor (Router Dead Interval is 40 seconds)...
************************************************
3) Neighbors:
Router ID 10.1.1.102: Link-local address fe80::10:2, State is Full
************************************************
Must have only 1 neighbor: OK!
************************************************
.
Finished in 120.269288 seconds.

1 tests, 0 assertions, 0 failures, 0 errors
petry@ubuntu: /tg2impl$
```

Figura 5.8: Simulações de falha - variações no enlace

6 CONCLUSÕES

Os testes de conformidade da implementação do OSPFv3 do *Quagga* para IPv6 tiveram êxito em sua execução. Dessa forma, limitando-se ao escopo dos casos de testes aplicados neste trabalho, é possível recomendar a implementação do protocolo OSPFv3 pelo *Quagga*, assim como seu suporte ao endereçamento IPv6.

A maneira abrangente com que foram implementados os testes, desde a conferência de comandos configurados no *vty* dos *daemons zebra* e *ospf6d* até a verificação de pacotes através de captura na rede, contribuiram para certificar, com maior precisão, os resultados das execuções.

O fato dos resultados mostrarem-se eficazes, permite concluir que mais casos de testes de conformidade podem ser especificados em futuros trabalhos, a fim de contemplar mais itens da RFC 5340 (COLTUN, et al., 2008).

Além disso, pode-se afirmar que a implementação foi bastante eficiente em comparação a uma execução manual, pois, neste caso, os testes apresentados neste trabalho seriam uma tarefa que demandaria um tempo muito mais elevado, visto que todas as configurações e comandos que verificam tabelas de roteamento, por exemplo, precisariam ser digitados e conferidos, respectivamente, em cada roteador da topologia. Tal verificação seria inviável ao tentar proceder os mesmos testes para dezenas de roteadores. No entanto, a automatização apresentada é escalável, apenas necessitando adaptar alguns pontos do código para a quantidade desejada de equipamentos.

Uma diferença da implementação apresentada neste trabalho em relação a um *shell script* em *Linux*, por exemplo, é, aquela, poder abstrair elementos tais como os roteadores e as conexões; enquanto, esta, não permitir tal nível de abstração.

Um projeto que pode estender as implementações apresentadas neste trabalho é a criação de uma ferramenta *web* pela qual seja possível executar os testes de conformidade. Através dessa interface, também poderá ser feito o acompanhamento da *suite* de teste, assim como obter seus resultados parcial e final. A possibilidade de executar um teste utilizando uma interface gráfica simples e intuitiva seria muito útil para usuários que não estão familizarizados com o ambiente *Unix*, por exemplo.

REFERÊNCIAS

BHOSALE, S.; JOSHI, R. Conformance Testing of OSPF Protocol. **IET International Conference on Wireless, Mobile and Multimedia Networks**, [S.l.], 2008.

CISCO. **Cisco IOS IPv6 Configuration Guide**. http://www.cisco.com/en/US/docs/ios/ipv6/configuration/guide/12-4/ipv6-12-4-book.pdf. Acessado em 26 de julho de 2011., Release 12.4.

COLTUN, et al. **OSPF for IPv6**: RFC 5340. [S.l.]: Network Working Group, 2008.

COMBS, G. **tshark**. http://www.wireshark.org. Acessado em 05 de agosto de 2011.

GROTTKE, M.; TRIVEDI, K. S. Fighting Bugs: remove, retry, replicate, and rejuvenate. **Computer**, [S.l.], v.40, 2007.

HAGEN, S. **IPv6 Essentials**: Integrating IPv6 into Your IPv4 Network. 1st.ed. [S.l.]: O'Reilly Media Inc., 2002.

HALABI, S. **OSPF Design Guide**. [S.l.]: Cisco Systems - Network Supported Accounts, 1996. http://www.cs.fsu.edu/courses/netdesign/halabi/halabi-ospf-design-guide.pdf. Acessado em 20 de julho de 2011.

HART, J. **Racket - Ruby Raw Packet library**. http://spoofed.org/files/racket/src/. Acessado em 02 de agosto de 2011.

HINDEN, R.; DEERING, S. **Internet Protocol, Version 6 (IPv6) Specification**: RFC 2460. [S.l.]: Network Working Group, 1998.

HINDEN, R.; DEERING, S. **IP Version 6 Addressing Architecture**: RFC 4291. [S.l.]: Network Working Group, 2006.

HINDEN, R.; HABERMAN, B. **Unique Local IPv6 Unicast Addresses**: RFC 4193. [S.l.]: Network Working Group, 2005.

IEEE802.1AX. **Link Aggregation**: IEEE 802.1AX. [S.l.: s.n.], 2008.

IEEE802.1Q. **Virtual LANs**: IEEE 802.1Q. [S.l.: s.n.], 2006.

ISHIGURO, et al. **Quagga**: A routing software package for TCP/IP networks. http://www.quagga.net/docs/quagga.pdf. Acessado em 10 de outubro de 2011.

JACOBSON, V.; LERES, C.; MCCANNE, S. **tcpdump**. http://www.tcpdump.org. Acessado em 05 de agosto de 2011.

MOY, J. **OSPF Version 2**: RFC 2328. [S.l.]: Network Working Group, 1998.

NICBR. **Estoque mundial de endereços IP está esgotado, informa Icann (Internet Corporation for Assigned Names and Numbers)**. http://www.nic.br/imprensa/clipping/2011/midia121.htm. Acessado em 28 de julho de 2011.

ORACLE. **VirtualBox**. https://www.virtualbox.org/. Acessado em 31 de outubro de 2011.

ORACLE. **VirtualBox - Bridge Networking**. http://www.virtualbox.org/manual/ch06.html. Acessado em 05 de novembro de 2011.

PAYNE, R.; MANWEILER, K. **CCIE**: Cisco(TM) Certified Internetwork Expert: Study Guide. 2nd.ed. [S.l.]: Cisco Press, 2003.

QUAGGA. **Quagga Source Code**. http://www.quagga.net/download. Obtido em 10 de outubro de 2011., Versão 0.99.20.

STALLINGS, W. **Data and Computer Communications**. 5th.ed. [S.l.]: Prentice Hall, 1997.

TALBOTT, N. **Test::Unit**. http://www.ruby-doc.org/stdlib-1.8.7/libdoc/test/unit/rdoc/Test/Unit.html. Acessado em 14 de novembro de 2011.

TANENBAUM, A. S. **Redes de Computadores**. 4a..ed. [S.l.]: Campus (Elsevier), 2003.

V. ERAMO M. LISTANTI, A. C. Switching Time Measurement And Optimization Issues In GNU Quagga Routing Software. **IEEE Globecom 2005**, [S.l.], 2005.

ANEXO A - TRABALHO DE GRADUAÇÃO I

Automatização de Testes de Conformidade da Implementação do OSPFv3 do *Quagga* para IPv6

Eduardo Tiago Petry[1], **Prof. Dr. Sérgio Luis Cechin (orientador)**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{etpetry,cechin}@inf.ufrgs.br

Abstract. *The compliance testing of embedded protocols in switches and routers by automated tests has increased by developers of telecommunications equipments. Usually, OSPF (Open Shortest Path First) protocol is the internal routing of Internet backbones, deployed over Metro Ethernet networks, which currently operate with IPv4 (Internet Protocol version 4). Since the operational migration of these networks to use IPv6 addresses is soon, this paper intends to present the compliance testing of OSPF version 3 implementation, which provides support for that addressing. Hence, this paper aims to check reactions of the implementation of this routing protocol when it suffers simulated failures.*

Resumo. *A utilização de testes automáticos para conformidade de protocolos presentes em* switches *e roteadores tem crescido entre desenvolvedores de equipamentos para telecomunicações. Usualmente, emprega-se o protocolo OSPF (*Open Shortest Path First*) para o roteamento interno de* backbones *sobre redes Metro Ethernet, as quais operam atualmente com endereços IPv4 (*Internet Protocol version 4*). Tendo em vista a breve migração operacional dessas redes para o uso de endereços IPv6, pretende-se apresentar neste trabalho testes de conformidade da implementação do* Quagga *para a versão 3 do OSPF, a qual fornece suporte a esse endereçamento. O objetivo deste trabalho, então, é verificar o comportamento da implementação deste protocolo de roteamento diante de situações de falha, as quais serão descritas posteriormente.*

1. INTRODUÇÃO

As redes SONET/SDH estão sendo substituídas gradativamente por topologias Metro Ethernet. Um dos motivos dessa substituição é o fato daquela tecnologia ser inadequada para a demanda de banda requerida pelas aplicações emergentes da Internet [Halabi 2003], além de ter um custo inferior de operação. O avanço dessas novas redes tem sido acompanhado também por maneiras mais ágeis de validar os *switches* e roteadores que as compõem.

Diversos protocolos das camadas 2 e 3 do modelo OSI [Tanenbaum 2003], assim como funcionalidades específicas desses equipamentos, tais como VLAN [IEEE802.1Q 2006] e Port-Channel [IEEE802.1AX 2008], são exaustivamente testados em laboratório antes de serem comercializados. Tradicionalmente, a validação de *switches* e roteadores é composta por um roteiro de testes, o qual descreve cada etapa a ser executada, a fim de verificar se cada funcionalidade do equipamento efetua suas ações adequadamente. Caso uma falha seja evidenciada, muitas vezes o *bug* encontrado

não é facilmente reproduzível [Grottke and Trivedi 2007]: seja por concorrência entre os processos internos do equipamento (o que torna o resultado não-determinístico), seja por problemas de temporização das ações do hardware ou do software.

A fim de minimizar esses problemas e, além disso, estabelecer uma forma sistemática de aplicar essas validações, é possível traduzi-las, em uma primeira análise, para *scripts* automatizados. Estes, por sua vez, podem tornar mais ágil a verificação de certas funcionalidades em um equipamento Metro Ethernet. Obtendo, assim, uma garantia de que a execução dos passos do roteiro da validação serão sempre realizados em uma mesma ordem e com os mesmos intervalos de tempo.

Caso um protocolo qualquer (contido no software embarcado de um roteador) possua erros em sua implementação, estes podem ser evidenciados através de testes de conformidade do equipamento, focada nas propriedades desse protocolo. Automatizando os testes, essa execução pode ser feita com uma periodicidade muito menor. Além disso, a sequência de passos a ser executados será de uma forma padrão, sem influência humana ou outra variável que possa interfir no resultado e na análise de possíveis falhas.

Para escrever esses *scripts*, escolheu-se a linguagem *Ruby*, motivada por sua simplicidade, robustez e flexibilidade [Ruby 2011]. Pelo fato de ser interpretada, intuitiva e possuir uma ampla variedade de bibliotecas que facilitam a implementação de diversas aplicações [Flanagan and Matsumoto 2008], ela traz grande poder ao desenvolvimento de testes de conformidade automáticos.

A DATACOM [DTC 2011], empresa brasileira fabricante de equipamentos de telecomunicações, possui uma linha Metro Ethernet, para a qual foram desenvolvidos testes automáticos para avaliar os protocolos de rede que utilizam IPv4. A implementação dessa ferramenta foi feita com a linguagem *Ruby*, a fim de produzir diversos tipos de testes para diferentes funcionalidades desses equipamentos. Dentre as verificações existentes, um exemplo é o teste da implementação do protocolo OSPF(*Open Shortest Path First*) utilizando redes IPv4, o qual testa, por exemplo, seu comportamento diante do rompimento do *link* entre dois roteadores.

1.1. Contextualização

Neste trabalho, pretende-se analisar o protocolo de rede OSPF (*Open Shortest Path First*), classificado como um IGP (*Interior Gateway Protocol*) [Tanenbaum 2003], em sua versão 3 (OSPFv3) [Coltun 2008], na qual é utilizado o IPv6 (*Internet Protocol version 6*) [Hinden and Deering 1998]. Sua versão tradicional [Moy 1998], implementada para IPv4, é amplamente utilizada em roteamento dentro de um mesmo *SA (Sistema Autônomo)*.

Assim que foi definido o protocolo IPv6 [Hinden and Deering 1998], uma nova versão do OSPF foi desenvolvida. Surgiu, então, a versão 3 [Coltun 2008] do OSPF, sofrendo sua última atualização em 2008. A comparação entre as duas versões é apresentada na seção 2.2.

Para que seja possível a simulação de roteadores reais em um computador pessoal, é utilizado o *Quagga*, o qual implementa os principais protocolos de roteamento, inclusive suportando IPv6. O sistema operacional e o hardware de um computador não apresentam arquitetura dedicada ao roteamento, porém o *Quagga* apresenta razoável performance em

comparação com roteadores Cisco [V. Eramo 2005]. Ele pode ser instalado facilmente em um computador com sistema operacional baseado em Unix [Ishiguro et al. 2006]. Detalhes da sua arquitetura serão apresentados posteriormente na seção 2.3.

1.2. Motivação

Apesar da capacidade de endereçamento do IPv4 estar próxima do limite [NICBR 2011], ainda é majoritário seu uso na rede mundial, em especial nas grandes operadoras de telecomunicações no Brasil (por exemplo, *Oi-RJ*, *CEMIG-MG*, *Embratel-RJ* e *Telefonica-SP*). Elas estão preparando seu *backbone* para o uso do IPv6, através de testes internos e adiantando preparação de suas equipes de operação e suporte.

Dessa maneira, uma das motivações para este trabalho é o fato da migração completa para operações reais utilizando IPv6 ainda não foi atingida. Por isso, protocolos de roteamento completamente preparados ao novo modo de endereçamento, tal como o OSPFv3, não são, por enquanto, utilizados em larga escala. Portanto, ainda não há testes automáticos plenamente desenvolvidos para validação de protocolos e funcionalidades relacionados ao IPv6.

1.3. Objetivos

Os objetivos iniciais deste trabalho são:

- o estudo do protocolo IPv6 e suas principais diferenças para seu antecessor, o IPv4;
- compreensão do protocolo OSPFv3, aplicado ao uso do endereçamento IPv6;
- entendimento sobre o fucionamento e configuração do *Quagga*;

Utilizando a linguagem *Ruby*, a continuação deste trabalho será dedicada, principalmente, à implementação das classes e os métodos que:

- representam o *Quagga* como roteador;
- efetuam a comunicação com o *Quagga*, enviando a ele comandos relativos à configuração do OSPFv3;
- simulam falhas que reflitam em alterações de estado do protocolo OSPF;
- verificam o estado do OSPFv3 através das informações fornecidas pelo *Quagga*;
- informem ao usuário situações de erros ou acertos sobre o estado do OSPF.

Também é verificada se a implementação do OSPFv3, contida no *Quagga*, está de acordo com sua RFC (*Request For Comments*) específica [Coltun 2008].

O protocolo OSPF possui grande quantidade de informações e diferentes pacotes a serem testados. Assim, a fim de limitar o escopo deste trabalho, serão apresentados nas seções seguintes os principais pacotes do OSPFv3 (dentro de uma mesma área) e possíveis falhas em uma rede, sobre os quais incidirão testes de conformidade.

2. ESTADO DA ARTE

2.1. IPv6

A RFC2460 [Hinden and Deering 1998] especifica um novo formato de endereçamento IP, o qual é composto por 128 bits, ao invés de apenas 32 bits do IPv4. A motivação principal para o incremento na quantidade de bits é exatamente a limitação na quantidade

de possibilidades de endereços do IPv4, o qual está chegando ao fim de sua vida útil, com capacidade de pouco mais de 4 bilhões de endereços [Stallings 1997]. Com o espaço de endereçamento do IPv6, pode ser gerados, aproximadamente, $3.4 * 10^{38}$ endereços (cerca de $6 * 10^{23}$ endereços por metro quadrado da superfície terrestre [Hinden 1995]). Muitos destes serão utilizados por dispositivos móveis ou residenciais com endereço IP (desde *smartphones* até eletrodomésticos), ou seja, a demanda já é grande.

2.1.1. Notação dos Endereços

Sua notação é dada por 8 blocos de 16 bits cada um, por exemplo, *2580:DB8:0000:0000:0000:0AC3:1122:0001*.
A fim de simplificar, os zeros consecutivos podem ser removidos. Portanto, estas duas notações também são válidas para o mesmo endereço apresentado anteriormente: *2580:DB8:0:0:0:AC3:1122:0001* e *2580:DB8::AC3:1122:0001* [Kawamura and Kawashima 2010].

A representação da máscara de rede (ou prefixo) de um endereço IPv6 é semelhante àquela já utilizada para os endereços IPv4, utilizando-se a notação *Classless Interdomain Routing (CIDR)*: endereço-IPv6prefixo [Hinden and Deering 2006]. Dessa forma, o endereço mencionado no parágrafo anterior poderia pertencer, por exemplo, à rede *2580:DB8::/64*.

2.1.2. Tipos de Endereços

Há, basicamente, três formas de endereços no IPv6. Quanto a isso, a principal diferença para o IPv4 é a eliminação do endereço *broadcast*. Seguem os tipos de endereços [Hagen 2002]:

- *Unicast*: Endereço IP de uma única interface. Pacote é recebido diretamente pela interface que tem esse endereço de destino;
- *Anycast*: Identificador de um conjunto de interfaces, normalmente pertecente a diferentes hosts e/ou roteadores. O pacote direcionado a um endereço desse tipo é entregue, tipicamente, à localização (para a qual contém tal endereço *anycast*) que o protocolo de roteamento informa ser a melhor.
- *Multicast*: Assim como já era existente no IPv4, define um endereço para um grupo de interfaces. A diferença para o *anycast* é dada pelo fato envio do pacote ter a intenção de atingir todas as interfaces que pertençam a tal grupo. Substitui o *broadcast* do IPv4.

2.1.3. Diferenças para o IPv4

Além da maior quantidade de bits para representar o endereço, utilizaram-se também da experiência de uso do IPv4 para implementar melhorias que julgaram-se necessárias. Outras modificações expressivas podem ser percebidas no formato do pacote [Kurose and Ross 2006]. Entre elas, o cabeçalho possui tamanho fixo de 40 bytes, contendo 7 campos (contra 13 campos do cabeçalho do IPv4). Ele deixou de ser

variável em função do campo "opções" ter sido eliminado. O campo do IPv4 *Protocol Type* foi substituído pelo campo "próximo cabeçalho" (*next header*), o qual indica se há "cabeçalhos de extensão" (informando o tipo) ou, na ausência destes, o tipo de protocolo (mesmos valores do usados no IPv4) [Hagen 2002]. Não há mecanismo de *checksum* no IPv6, deixando essa verificação para as camadas de transporte e de enlace de dados.

A implantação de "cabeçalhos de extensão", localizados antes da área de dados do pacote, fez com que o cabeçalho padrão do IPv6, contendo somente campos obrigatórios, se tornasse mais enxuto e com tamanho pré-determinado. Esse novo padrão possibilita algumas vantagens, tal como o processamento mais rápido no tratamento da recepção do pacote no roteador, melhorando a vazão [Tanenbaum 2003] dos dados, obtendo menos *overhead*. Isso é justificado, principalmente, por ser desnecessário a análise dos campos opcionais do cabeçalho IPv6 [Stallings 1997].

O cabeçalho do IPv6 é ilustrado na figura 1, através da qual é possível identificar visualmente os campos que foram removidos, mantidos e criados em relação ao IPv4.

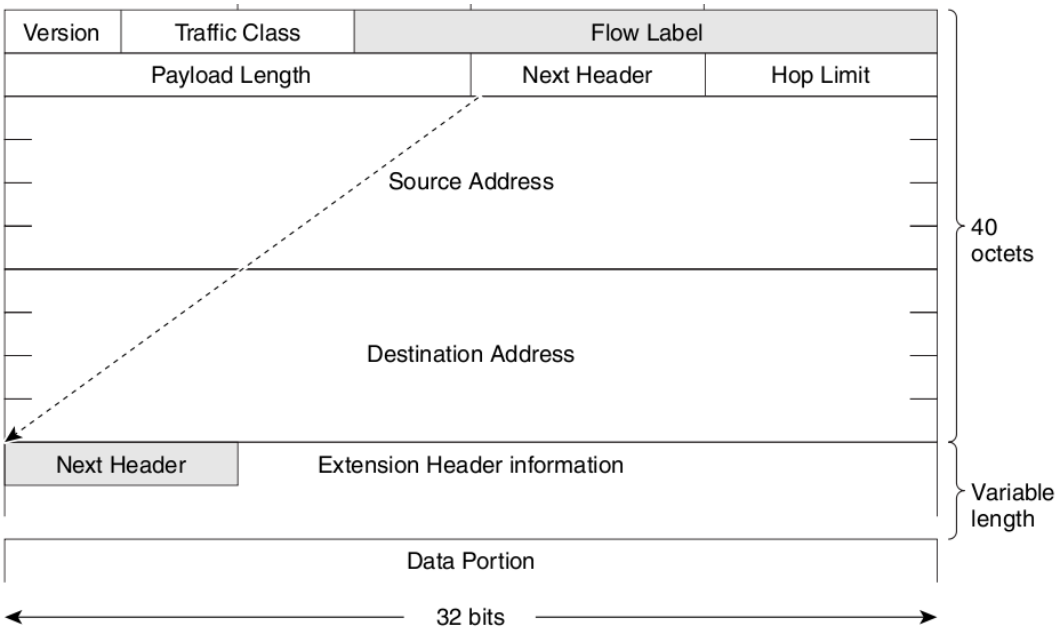

Figura 1. Cabeçalho IPv6 [Payne and Manweiler 2003]

2.2. OSPFv3

O protocolo de roteamento OSPF (*Open Shortest Path First*), dito como um IGP (*Interior Gateway Protocol*), é utilizado dentro de um mesmo AS (*Autonomous System*) [Hagen 2002]. Ele é um protocolo de "estado de enlace" (*link-state*). Pode-se considerar um enlace como sendo a interface de um roteador. A descrição dessa interface (seu endereço IP, máscara de rede e tipo de rede em que está conectada, por exemplo) e o relacionamento do enlace com seus roteadores vizinhos representam o "estado de enlace" [Halabi 1996].

O protocolo OSPF define um conceito de "área", a qual pode conter uma ou mais redes. Caso existam mais áreas, uma delas deve ser identificada como "área 0", que é denominada *backbone* (identificada pelo *ID* '0'). Todas as áreas devem estar

diretamente conectadas a ela (ou indiretamente, através de túneis, chamados *virtual links*) [Payne and Manweiler 2003]. Caso um roteador pertença a mais de uma área, ficando na divisão entre elas, chamamos ele de *ABR - Area Border Router*. Outro roteador pode ainda estar na fronteira do seu AS, denominado, então *ASBR - Autonomous System Border Router*.

As principais mensagens OSPF que um roteador envia são:

- *Hello*: primeira mensagem enviada, antes mesmo de estabelecer adjacência com seu nó vizinho. Após isso, o intuito é saber se o vizinho está ativo (usualmente enviada a cada 10 segundos) [Cisco 2008].
- *Link-State Advertisements - LSAs*: distribui informações de estado do *link* para seus vizinhos.

A fim de manterem-se atualizados, periodicamente (ou quando há alteração na topologia), os roteadores trocam entre si *LSAs*, nas quais constam informações que eles conhecem sobre cada *link*. Dessa forma, cada nó monta uma base da dados, formada pelas LSAs recebidas e pelos dados locais: compondo a LSDB (*Link-State Database*). Sobre esta, é executado localmente o algoritmo *Shortest Path First (SPF)*, o qual calcula o caminho mais curto para atingir uma determinada rede, levando em consideração o custo associado para chegar ao destino.

2.2.1. Diferenças para a Versão 2

Grande parte das características da versão 3 do OSPF, a qual possui suporte ao IPv6, foi mantida em comparação à sua versão anterior (OSPFv2) [Moy 1998]. Por exemplo, o algoritmo de inundação (*flooding*), a eleição do *Designated Router (DR)*/*Backup Designated Router (BDR)* e o cálculo do caminho mais curto [Coltun 2008].

No entanto, devido a alterações do protocolo IPv6 em relação ao IPv4, já comentadas anteriormente, e também para lidar com o aumento da quantidade de bits do IPv6, alguns algoritmos internos do OSPF precisaram sofrer modificações para se adaptarem a esse novo formato de endereçamento [Coltun 2008].

O IPv6 utiliza o conceito de *link*, substituindo as ideias de "rede" e "sub-rede", as quais são utilizados no IPv4 [Coltun 2008]. Pelo fato de poder configurar-se diferentes sub-redes em uma mesma interface de um roteador, é possível que em um *link* único, entre dois deles, haja comunicação mesmo que os roteadores pertençam a sub-redes distintas [Hagen 2002]. Assim, pode-se dizer que uma interface OSPF faz uma conexão através de um *link* ao invés de uma estabelecê-la por uma sub-rede [Coltun 2008].

No cabeçalho dos pacotes do OSPF não estão mais presentes endereços IPv6; estes, porém, apenas podem estar na área de dados do pacote OSPF. Na versão 3 do protocolo OSPF, os campos "*Router ID*", que identifica um roteador, e "área" são os que mantiveram o formato de endereço do IPv4.

2.3. Quagga

O *Quagga* é um software que implementa protocolos de roteamento, tais como RIP(*Routing Information Protocol*), OSPF e BGP(*Border Gateway Protocol*). Ele

permite que um computador com sistema Unix possa transformar-se em um roteador. Ele também possui total suporte ao IPv6.

A comunicação entre a tabela de roteamento do *kernel* e os protocolos ocorre através de um processo gerenciador, denominado *zebra*. Ele é uma abstração entre o *kernel* Unix e a API *Zserv*, pela qual foram implementados os protocolos de roteamento disponíveis no pacote do *Quagga.*

Em uma descrição simples a respeito da sua arquitetura, pode-se dizer que o *zebra* administra a tabela de roteamento através da troca de informações com o *kernel* do sistema operacional e obtendo dados através dos protocolos de roteamento que estão implementados no *Quagga.*

Existe uma interface de comando (*CLI - Command Line Interface*), típica em roteadores, pela qual configuram-se as interfaces e parâmetros dos protocolos de roteamento. Além disso, obtém-se informações relativas ao estado de cada protocolo que deseja-se executar. O modo de acesso ao CLI, correspondente à cada processo, é feito através do *telnet* na máquina onde ele está rodando, utilizando a porta *TCP (Transmission Control Protocol)* pela qual tal acesso é permitido. No caso do interesse deste trabalho, as portas TCP a serem utilizadas serão as de número 2601 e 2606.

2.4. Testes Automáticos

No artigo [Li et al. 2003], é afirmado que a simulação de topologias estuda como gerar informações de roteamento associadas a uma certa topologia de rede e para um dado protocolo de roteamento, além de verificar como introduzir essas informações no *Router Under Test (RUT)*. Ou seja, todo o processamento das informações de roteamento (desde sua origem, propagação e posterior cálculo da tabela) é o conteúdo principal dos testes de conformidade de um protocolo de roteamento.

Simuladores para teste de conformidade de diferentes protocolos de rede estão comercialmente disponíveis, pelo qual as funcionalidades de tais protocolos podem ser verificadas automaticamente [Bhosale and Joshi 2008](tradução minha). Tipicamente, essas ferramentas possuem um alto custo, pela necessidade de serem adquiridas juntamente com equipamentos cujo *hardware* é projetado para dedicação exclusiva às tarefas de verificação dos protocolos. Além disso são capazes, normalmente, de gerar alta taxa de tráfego (na ordem de 1Gbps e 10Gbps) para complementar os testes.

2.4.1. *OSPF Emulation Software* Ixia

A *Ixia* é uma empresa americana fornecedora de soluções em software e hardware para testes de desempenho e validações para equipamentos de rede. Desenvolveu a biblioteca *IxANVL (Ixia Automated Network Validation Library)*, considerada o padrão da indústria dos testes de conformidade de protocolos [IxiaLib 2011].

Dentre as diversas soluções [Ixia 2011] que ela fabrica, uma delas é a *IxNetwork*: uma aplicação gráfica na qual é possível emular diversos roteadores e seus protocolos de roteamento. Integrado a ela, pode-se utilizar o *Ixia's OSPF Emulation Software* [IxiaOSPF 2011], a fim de testar a escalabilidade, funcionalidade e desempenho do OSPF em roteadores reais, suportando tanto OSPFv2 quanto OSPFv3.

2.4.2. *OSPF Conformance Test Suite*: Spirent

O *Spirent* é um software utilizado juntamente ao equipamento de sistema de teste AX/4000 (do mesmo fabricante). O aplicativo contém vários casos de teste a fim de verificar a conformidade do OSPFv2 e OSPFv3 em um roteador. As validações podem ser feitas seletivamente, as quais apresentam resultados do tipo "passou", "falhou" ou "resultado inconclusivo".

Dentre os diversos parâmetros de testes suportados pela ferramenta, estes são os principais:

- Configuração de endereços IP nas interfaces;
- Estabelecimento, persistência e remoção da adjacência com os roteadores vizinhos;
- Permutação da base de dados e LSAs com os vizinhos
- Conferência do formato dos cabeçalhos do OSPFv3 e IPv6, assim como sua área de dados
- Suporte a múltiplas instâncias do OSPF

2.4.3. *InterEmulator OSPF Routing Conformance Test Suite: NetTest*

O *NetTest*, apenas disponível para a versão 2 do OSPF, permite testar praticamente todos os pacotes envolvidos no OSPF, como o *Hello* e as LSAs. Também valida a adjacência entre os roteadores vizinhos, além de simular o cálculo das rotas

2.4.4. Implementação para OSPFv2 e IPv4: DATACOM

Seguindo a apresentação feita na seção 1, a DATACOM desenvolve validações automatizadas focadas nos equipamentos Metro Ethernet fabricados por ela. Diferentemente das ferramentas apresentadas anteriormente, o **DmTest**, denominação do conjunto de software e hardware desenvolvido pela empresa para executar os testes automáticos, é projetado para verificar as funcionalidades e conformidade da operação dos protocolos que estão presentes nos *switch-routers* da própria DATACOM.

Tratando-se de protocolos de roteamento, esses testes possuem ampla cobertura para redes IPv4. No entanto, pelo motivos comentados nas seções introdutórias, ainda é necessário ampliar os testes de conformidade utilizando IPv6.

3. PROJETO

3.1. Visão Geral

A emulação de roteadores reais será feita através do uso do software *Quagga*, o qual foi apresentado na seção 2.3. O mapa conceitual contendo os elementos que irão compor a implementação deste trabalho está na figura 2.

Os componentes que integram os testes executados são apresentados na figura 3. Eles são de dois tipos:

- **Gerenciador**: responsável por gerar, enviar e receber os pacotes trocados com os PCs que emulam roteadores. É, propriamente dito, o executor do teste.

- **DUT** - *Device Under Test**: PC que emula um roteador utilizando o *Quagga.*

(*Pode variar a quantidade de DUTs utilizados)

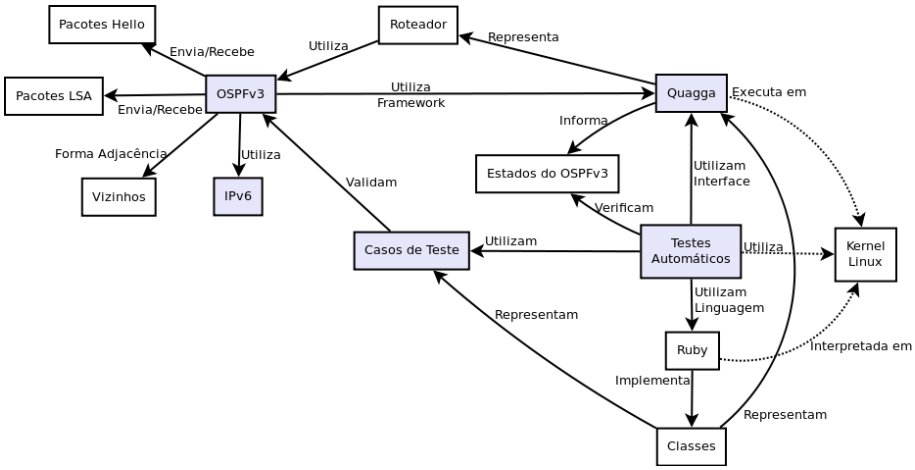

Figura 2. Mapa Conceitual

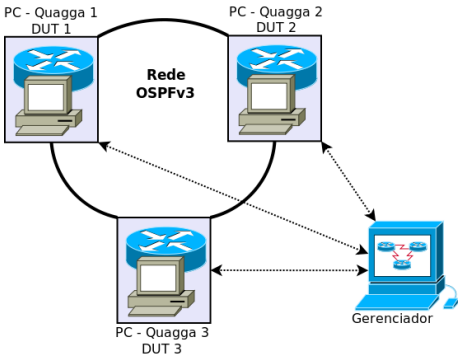

Figura 3. Componentes dos testes

3.2. Quagga

3.2.1. Configuração

Os arquivos fontes do *Quagga* podem ser obtidos através do *site* oficial do projeto [Quagga 2011].

O *daemon* que possui o OSPFv3 implementado no *Quagga* chama-se *ospf6d.* Os arquivos de configuração básica do *zebra* e do *ospf6d* são similares. Os parâmetros mínimos para sua habilitação são:

Listing 1. "zebra.conf" e "ospf6d.conf"

```
1 hostname Router
2 password zebra
3 !
4 line vty
5  no login
```

3.2.2. Habilitando os Serviços

O *zebra* é habilitado da seguinte forma:

sudo zebra -u root -g root -A ::1

Habilita-se o *ospf6d* de maneira similar:

sudo ospf6d -u root -g root -A ::1

A interface de linha de comando para cada *daemon* é acessada através de *telnet* para o endereço *localhost* em uma das portas específicas para cada serviço: 2601 para o *zebra* e 2606 para o *ospf6*.

3.3. Casos de Teste

A divisão dos casos de teste que serão utilizados foram inspirados na metodologia proposta em [Bhosale and Joshi 2008]. Segue a relação dos itens e uma breve descrição sobre o que cada um deles tem a responsabilidade de testar:

- **Paramétricos**: valores configuráveis, tais como *Area ID*, *Router ID*, parâmetros de temporização e de métrica.
- **Conformidade dos pacotes**: analisar campos dos pacotes *Hello* e dos diferentes tipos de LSAs.
- **Informações no roteador**: verificar no *Quagga* se as informações relatadas anteriormente são exibidas de forma coerente.

A cada teste descrito nas próximas subseções, os itens acima descritos serão integralmente verificados, à medida que estiverem dentro do escopo de determinado teste.

3.3.1. Conformidade dos Pacotes

As mensagens de *Hello* e alguns tipos de *Link State (LS)* serão verificadas. Em uma primeira análise, os testes estarão restritos a somente uma área do OSPF. Em função disso, segue uma lista (baseada em [Hagen 2002]) dos pacotes *Link State* que serão validados na tabela 1.

Tipo do LS	Escopo do *flooding*	Quem envia
Router-LSA	Área	Cada roteador
Network-LSA	Área	*Designated Router - DR*
Link-LSA	*Link*	Cada roteador para cada link
Intra-Area-Prefix-LSA	Área	Cada roteador

Tabela 1. Tipos de LSAs

3.4. Verificações Iniciais do OSPF

Utilizando-se somente um roteador (figura 4), a validação do protocolo OSPF para IPv6 começa pela habilitação do *daemon ospf6d*, utilizando-se a área 0:

Listing 2. “Habilitar OSPFv3”

```
1 ospfv3# configure terminal
2 ospfv3(config)# router ospf6
3 ospfv3(config-ospf6)#
4 ospfv3(config-ospf6)# interface eth0 area 0.0.0.0
```

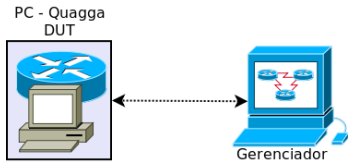

Figura 4. Teste inicial

3.4.1. Pacotes *Hello*

Em uma topologia simples, como indicada na figura 4, serão efetuadas algumas validações iniciais, a fim de verificar se os pacotes *Hello* estão adequados às configurações efetuadas.

Setup:

1. Um DUT.
2. Configurar valores (dentro da interface) para *Hello Interval*, *Dead Interval*, *Retransmit Interval*, *Transmit Delay*, instância, custo e prioridade.
3. Configurar valores (dentro do *router ospf*) para Área *ID* (especificando a interface) e *router ID*.
4. **Gerenciador** captura pacotes *Hello*.

Resultados esperados:

1. Os parâmetros são aceitos na configuração.
2. Valores contidos dentro do pacote *Hello* devem ser exatamente aqueles configurados.
3. Informações são exibidas corretamente no CLI do roteador.

3.5. Simulações de Falha

Expandindo a quantidade de roteadores, a fim de formarem adjacência entre seus vizinhos, serão apresentados os casos em que há falha diante de certas ocorrências em uma rede Metro Ethernet.

- Recebimento de pacotes OSPF:
  - Pacote OSPFv3 malformado;
  - Campo com valor diferente do esperado;
  - Pacote OSPFv2;
- Erros de configuração:
  - Parâmetros de temporização;
  - Área;
  - *Router ID*;
- Variação no link:
  - Queda no *link*;
  - Remoção do IP *link-local*;
  - Alteração do IP *link-local*;

3.6. Implementação: Linguagem Ruby

Ruby é uma linguagem orientada a objeto e interpretada, sendo executada similarmente a um *script*. Possui um alto poder de abstração e contém diversas bibliotecas, entre elas, as que auxiliam na implementação da geração e captura dos pacotes, a fim de validá-los.

As classes da validação automatizada foram divididas da seguinte maneira:

- ***InterfaceQuagga***
  Comunicação através de *telnet* para o envio e recebimento das informações presente na CLI dos roteadores. Um método obrigatório é o que pode-se chamar de "*parsing*", que é responsável por tratar o recebimento das informações requeridas dos roteadores.
- ***Router***
  Simboliza o roteador a ser testado(DUT). Contém os métodos que são operações reais possíveis pelo roteador. Mantém o estado do DUT.
- ***OSPFv3***
  Contém os estados durante o processo de estabelecimento de vizinhança.
- ***ConnectionRouters***
  Estabelece a conexão entre dois roteadores. Cada objeto dessa classe representa uma conexão física.
- ***IPv6***
  Trata os endereços IPv6 e valida as redes através do prefixo de rede.
- **Captura de Pacotes**
  Conforme introduzido na seção 3.3.1, a captura dos pacotes é feita pelo "Gerenciador". Além de executar o código *Ruby*, ele também verifica os pacotes trafegados entre os DUTs ou gerado por somente um DUT (para o caso inicial, *standalone*). A verificação é feita através do analisador de tráfego de rede *tcpdump* [Jacobson et al. 2011]. A fim de filtrar isoladamente cada campo de um pacote, é utilizada outra ferramenta: *tshark* [Combs 2011]. Assim, é possível validar cada parâmetro de qualquer pacote OSPF.
- **Geração de Pacotes**
  A geração de pacotes que são utilizados para simular falhas é feita pela biblioteca *Racket* [Hart 2011], disponível para a linguagem *Ruby*. Com ela, é possível "fabricar" dados desde a camada 2 até a camada 5 e enviá-los pela interface de rede.

4. CONCLUSÕES

Embasando-se no projeto apresentado, juntamente aos estudos do IPv6 e do OSPFv3, é possível implementar os testes de conformidade do protocolo OSPF, implementado no *Quagga*, utilizando-se a linguagem *Ruby*.

Pelo fato do autor deste trabalho já ter se envolvido em melhorias de uma ferramenta empresarial de testes automáticos, utilizando *Ruby*, haverá facilidade quanto ao uso dessa linguagem.

A interação dos testes com a interface oferecida pelo *Quagga* precisa apresentar um bom resultado, pois os dados que irão compor a saída da validação são obtidos, justamente, através da leitura daquilo que é fornecido pelo *Quagga*. É esperado que a implementação do OSPFv3 pelo *Quagga* atenda a sua RFC específica [Coltun 2008].

5. CRONOGRAMA

O estudo, a implementação e a escrita da monografia do trabalho de graduação estão planejados conforme descrito na tabela 2.

Atividade	**Agosto**	**Setembro**	**Outubro**	**Novembro**
Estudo: IPv6/OSPFv3	X	X		
Expansão dos Casos de Teste	X	X		
Testes Básicos: IPv6/OSPFv3/Quagga		X		
Implementação Ruby: *Framework*		X	X	
Implementação Ruby: *Testes*		X	X	
Testes da Implementação			X	
Escrita da Monografia			X	X
Revisão da Implementação e da Escrita				X

Tabela 2. Cronograma 2011

Referências

Bhosale, S. and Joshi, R. (2008). Conformance Testing of OSPF Protocol. *IET International Conference on Wireless, Mobile and Multimedia Networks*.

Cisco (2008). Cisco IOS IPv6 Configuration Guide. Release 12.4. http://www.cisco.com/en/US/docs/ios/ipv6/configuration/guide/12-4/ipv6-12-4-book.pdf. Acessado em 26 de julho de 2011.

Coltun, e. a. (2008). OSPF for IPv6: RFC 5340. Technical report, Network Working Group.

Combs, G. (2011). tshark. http://www.wireshark.org. Acessado em 05 de agosto de 2011.

DTC (2011). DATACOM Telemática. http://www.datacom.ind.br. Acessado em 27 de julho de 2011.

Flanagan, D. and Matsumoto, Y. (2008). *The Ruby Programming Language*. O'Reilly Media Inc., 1st edition.

Grottke, M. and Trivedi, K. S. (2007). Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate. *Computer*, 40.

Hagen, S. (2002). *IPv6 Essentials: Integrating IPv6 into Your IPv4 Network.* O'Reilly Media Inc., 1st edition.

Halabi, S. (1996). OSPF Design Guide. Technical report, Cisco Systems - Network Supported Accounts. http://www.cs.fsu.edu/courses/netdesign/halabi/halabi-ospf-design-guide.pdf. Acessado em 20 de julho de 2011.

Halabi, S. (2003). *Metro Ethernet.* Cisco Press, 1st edition.

Hart, J. (2011). Racket - Ruby Raw Packet library. http://spoofed.org/files/racket/src/. Acessado em 02 de agosto de 2011.

Hinden, R. (1995). IP Next Generation Overview. *COMMUNICATIONS OF THE ACM*, 39 - No. 6.

Hinden, R. and Deering, S. (1998). Internet Protocol, Version 6 (IPv6) Specification: RFC 2460. Technical report, Network Working Group.

Hinden, R. and Deering, S. (2006). IP Version 6 Addressing Architecture: RFC 4291. Technical report, Network Working Group.

IEEE802.1AX (2008). Link Aggregation: IEEE 802.1AX. Technical report.

IEEE802.1Q (2006). Virtual LANs: IEEE 802.1Q. Technical report.

Ishiguro, K., Others, and Anothers (2006). Quagga: A routing software package for TCP/IP networks. http://www.quagga.net/docs/quagga.pdf. Acessado em 20 de julho de 2011.

Ixia (2011). Ixia Solutions. http://www.ixiacom.com/solutions/index.php. Acessado em 1º de agosto de 2011.

IxiaLib (2011). IxANVL - Ixia Automated Network Validation Library. http://www.ixiacom.com/library/white_papers/display?skey=ospf#A25. Acessado em 1º de agosto de 2011.

IxiaOSPF (2011). Ixia's OSPF Emulation Software Datasheet. http://www.ixiacom.com/pdfs/datasheets/ixnetwork_ospf_emulation.pdf. Acessado em 1º de agosto de 2011.

Jacobson, V., Leres, C., and McCanne, S. (2011). tcpdump. ttp://www.tcpdump.org. Acessado em 05 de agosto de 2011.

Kawamura, S. and Kawashima, M. (2010). A Recommendation for IPv6 Address Text Representation: RFC 5952. Technical report.

Kurose, J. F. and Ross, K. W. (2006). *Redes de Computadores e a Internet: Uma abordagem top-down*. 3rd edition.

Li, Z., Wang, Z., and Wu, J. (2003). Automatic conformance testing of OSPF routers. *International Conference on Communication Technology Proceedings*.

Moy, J. (1998). OSPF Version 2: RFC 2328. Technical report, Network Working Group.

NICBR (2011). Estoque mundial de endereços IP está esgotado, informa Icann(Internet Corporation for Assigned Names and Numbers). http://www.nic.br/imprensa/clipping/2011/midia121.htm. Acessado em 28 de julho de 2011.

Payne, R. and Manweiler, K. (2003). *CCIE: Cisco(TM) Certified Internetwork Expert: Study Guide*. Cisco Press, 2nd edition.

Quagga (2011). Quagga Source Code. Versão 0.99.18. http://www.quagga.net/download. Obtido em 15 de julho de 2011.

Ruby (2011). Ruby Language Official Web Site. http://www.ruby-lang.org. Acessado em 25 de julho de 2011.

Stallings, W. (1997). *Data and Computer Communications*. Prentice Hall, 5th edition.

Tanenbaum, A. S. (2003). *Redes de Computadores*. Campus (Elsevier), 4a. edition.

V. Eramo, M. Listanti, A. C. (2005). Switching time measurement and optimization issues in gnu quagga routing software. *IEEE Globecom 2005*.