MOZART PISTORI TOMAZETTI

SIMULAÇÃO DO PROTOCOLO DE ROTEAMENTO OSPF COM NS-3

Monografia apresentada como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação. Departamento de Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Elias P. Duarte Jr.

CURITIBA

2013

SUMÁRIO

RESUMO

O protocolo *Open Shortest Path First* (OSPF) é um dos protocolos de roteamento mais importantes da Internet. Os roteadores que utilizam o OSPF mantêm uma representação local da topologia da rede na forma de um grafo. A partir dessa representação, os roteadores utilizam o algoritmo de Djikstra para calcular o caminho mínimo de uma origem para cada destino da rede. A criação e manutenção desta representação da topologia é feita por três subprotocolos que juntos formam o OSPF. Cada um desses protocolos tem uma função, incluindo verificar se a comunicação com os roteadores vizinhos está operacional e informar todos os roteadores que ocorreu uma falha na rede. O *Network Simulator 3* (NS-3) é um simulador de eventos discretos, voltado principalmente para simulação de redes de computadores. O simulador NS-3 é um software livre disponível publicamente. O NS-3 é construído como um sistema de bibliotecas de *software*. Estas bibliotecas são escritas na linguagem C++. Este trabalho descreve como simular o OSPF no NS-3. Inicialmente é avaliada a possibilidade de usar uma implementação do OSPF nativa do NS-3. Entretanto, o NS-3 não possui o OSPF implementado. Sendo assim, um módulo do NS-3 chamado *Direct Code Execution* (DCE) passa a ser uma alternativa para simular o protocolo OSPF. O DCE permite executar aplicações reais dentro do NS-3. Uma destas aplicações é a implementação do protocolo OSPF feita pela *Quagga Routing Suite*. Portanto, o DCE fornece uma implementação completa do OSPF. Entretanto, o DCE não é nativo do NS-3 e isso adiciona complexidade ao procedimento de simular o OSPF. Na verdade, usar o DCE implica em trabalhar com a própria implementação do protocolo no sistema operacional, o que elimina a vantagem da simulação, na medida em que múltiplos detalhes devem ser considerados mesmo para testes simples. Além disso, a documentação do DCE, especialmente a parte relacionada ao OSPF, é insuficiente e incompleta.

CAPÍTULO 1

INTRODUÇÃO

A Internet é dividida em conjuntos de roteadores. Cada conjunto de roteadores está sob o controle de uma única entidade administrativa. Dentro destes conjuntos os roteadores trocam informações através do protocolo de roteamento interno. Além disso, os protocolos de roteamento são tipicamente classificados em dois tipos: vetor de distância e estado de enlace. O *Open Shortest Pah First* (OSPF) [18] é um protocolos de roteamento interno do tipo estado de enlace. O OSPF utiliza descrições do estado dos enlaces da rede para criar um representação da topologia da rede na forma de um grafo. A partir dessa representação, os roteadores utilizam o algoritmo de Dijkstra [12] para calcular o caminho mínimo de uma origem para cada destino da rede. A criação e manutenção desta representação da topologia é feita, principalmente, por três subprotocolos que juntos formam o OSPF. O protocolo *Hello* que tem o objetivo de verificar o estado dos enlaces e eleger um roteador designado e um roteador reserva. Quando dois roteadores estabelecem conectividade bidirecional, o protocolo *Exchange* é reponsável pela sincronização da representação local da topologia da rede. Além disso, o protocolo *Flooding* é reponsável por informar a todos os roteadores quando ocorre uma falha na rede.

O *Network Simulator 3* (NS-3) [4] é um simulador de eventos discretos, voltado principalmente para simulação de redes de computadores. O simulador NS-3 é um software livre disponível publicamente. O NS-3 é construído como um sistema de bibliotecas de *software*. Estas bibliotecas são escritas na linguagem C++.

Este trabalho descreve como simular o OSPF no NS-3. Inicialmente é avaliado o uso do *Global Router Manager* que é resposável por preencher as tabelas de roteamento do roteadores. Entretanto, o *Global Router Manager* não implementa um protocolo de

roteamento. Uma vez que, ele somente calcula caminhos estáticos, pois as tabelas de roteamento são preenchidas antes de iniciar o tráfego de pacotes na rede. Portanto, não são utilizados pacotes de controle e quando ocorre uma falha na rede os rotedores não são informados. Sendo assim, um módulo do NS-3 chamado *Direct Code Execution* (DCE) passa a ser uma alternativa para simular o protocolo OSPF. O DCE permite executar aplicações reais dentro do NS-3 e uma destas aplicações a implementação do protocolo OSPF feita pela *Quagga Routing Suite* [5]. O Quagga fornece implementações de protocolos de roteamento para as plataformas Unix. Ao executar as simulações é possível observar o comportamento do OSPF implementado no módulo DCE. O funcionamento da implementação do DCE é semelhante ao da implementação nativa do NS-3. Entretanto, é possível notar a diferença entre as duas implementações nas simulações em que ocorrem falhas na rede. Na implementação do NS-3 quando ocorre um falha de enlace, todos os pacotes que utilizavam o caminho afetado são perdidos. No OSPF do DCE somente alguns pacotes são perdidos, pois os roteadores percebem a falha do enlace e encontram um novo caminho mínimo.

Entretanto, o DCE não é nativo do NS-3 e isso adiciona complexidade ao procedimento de simular o OSPF. Caso o NS-3 seja instalado de acordo com sua documentação, o módulo DCE não é instalado e é necessário refazer a instalação para permitir o uso do DCE. Além disso, a maneira de configurar a simulação passa por diversas mudanças quando é utilizado o DCE e grande parte do conhecimento adquirido ao utilizar o NS-3 se torna obsoleto. De fato, usar o DCE implica em trabalhar com a própria implementação do protocolo no sistema operacional, o que elimina a vantagem da simulação, na medida em que múltiplos detalhes devem ser considerados mesmo para testes simples. Além disso, a documentação do DCE, especialmente a parte relacionada ao OSPF, é insuficiente e incompleta.

O restante deste trabalho está organizado da seguinte maneira. O capítulo 2 descreve as características do protocolo OSPF. O capítulo 3 descreve os procedimentos necessários para instalar e configurar o NS-3 para simular o OSPF. O capítulo 4 descreve execução de simulações do protocolo OSPF no NS-3 e o uso do DCE. Finalmente, o capítulo 5 traz as conclusões.

CAPÍTULO 2

O PROTOCOLO OSPF

Este capítulo descreve o protocolo de roteamento da Internet *Open Shortest Path First* (OSPF). Antes, uma visão geral do roteamento na Internet apresenta os conceitos de sistemas autônomos e protocolos de roteamento interno e externo. Em seguida, são descritas as tecnologias de roteamento denominadas vetor de distância e estado de enlace. O capítulo termina com uma descrição detalhada do protocolo OSPF.

2.1 Roteamento na Internet

A Internet é organizada em regiões chamadas sistemas autônomos [17]. Cada sistema autônomo consiste em um conjunto de roteadores sob o controle de uma única entidade administrativa. Dentro de um sistema autônomo os roteadores trocam informações através do protocolo de roteamento interno; para trocar informações entre sistemas autônomos é utilizado o protocolo de roteamento externo.

Todos os roteadores que pertencem a um único sistema autônomo devem permanecer conectados entre si [14]. Isto significa que os roteadores devem trocar informações para manter a conectividade do sistema autônomo. Um protocolo de roteamento interno é usado para configurar e manter as tabelas de roteamento dentro dos sistemas autônomos [15]. Segundo Comer [10], os roteadores pertencentes ao mesmo sistema autônomo se comunicam periodicamente entre si a fim de trocar informações de roteamento. Existem diversos protocolos para uso dentro de um sistema autônomo. Parte do motivo para a diversidade vem das várias topologias e tecnologias em uso. Como resultado, vários protocolos se tornaram populares. Como não existe um padrão único, o termo *Interior*

Gateway Protocol (IGP) é utilizado como uma descrição genérica para qualquer protocolo que os roteadores internos usam quando trocam informações de roteamento. Um único roteador pode utilizar dois protocolos de roteamento diferentes simultaneamente, um para comunicação fora de seu sistema autônomo e outro para comunicação dentro de seu sistema autônomo.

De acordo com Huitema [14], dividir a Internet em vários sistemas autônomos visa reduzir a sobrecarga de roteamento e facilitar a gestão da rede. As tabelas de roteamento de um sistema autônomo devem permitir que pacotes sejam enviados para todos os possíveis destinos da Internet. As tabelas de roteamento são mantidas pelos protocolos de roteamento. Porém as mensagens do protocolo de roteamento interno só transitam dentro do sistema autônomo. Os roteadores devem obter informações sobre redes externas. Para obter informações sobre redes externas é necessário realizar a troca de mensagens através de roteadores externos, que são pontos de entrada em sistemas autônomos adjacentes. A função do protocolo de roteamento externo é realizar a troca de informações de acessibilidade. Esta informação consiste de um conjunto de redes alcançáveis. Quando um roteador externo recebe informações de um outro roteador externo, ele utiliza o protocolo de roteamento interno para distribuir estas informações dentro do sistema autônomo.

Kurose e Ross [15] ressaltam algumas das diferenças entre o protocolo de roteamento interno e externo. Os protocolos de roteamento externo têm como prioridade transmitir políticas específicas de roteamento. Como o roteamento externo é orientado a políticas, não há preferência ou custos associados aos caminhos. Contudo, nos protocolos de roteamento interno as preocupações políticas podem ser ignoradas. Deste modo, os algoritmos de roteamento interno priorizam calcular, de maneira rápida, os melhores caminhos.

Segundo Kurose e Ross [15], três protocolos de roteamento têm sido usados para roteamento dentro de sistemas autônomos na Internet: o *Routing Information Protocol* (RIP), o *Open Shortest Pah First* (OSPF) e o *Enhanced Interior Gateway Routing Protocol* (EIGRP), proprietário da Cisco. A versão 2 do protocolo de roteamento interno RIP está definida na RFC 2453 [16]. A versão 2 do protocolo de roteamento interno OSPF está definida na RFC 2328 [18]. O protocolo OSPF possui uma nova versão dedicada ao IPv6

que está definida na RFC 5340 [9]. Atualmente o protocolo utilizado para roteamento entre sistemas autônomos é o *Border Gateway Protocol* (BGP). A versão 4 do protocolo de roteamento externo BGP está definida na RFC 4271 [19].

2.2 Classificação de Protocolos de Roteamento

Os protocolos de roteamento são tipicamente classificados em dois tipos: vetor de distância e estado de enlace, descritos a seguir.

2.2.1 Vetor de Distância

De acordo com Kurose e Ross [15], um protocolo do tipo vetor de distância é iterativo, assíncrono e distribuído. É distribuído porque cada roteador recebe informações de um ou mais vizinhos diretamente ligados a ele. Com as informações recebidas, o roteador realiza cálculos e distribui os resultados para seus vizinhos. É iterativo porque esse processo continua até que mais nenhuma informação seja trocada entre vizinhos. É assíncrono porque não requer que todos os roteadores executem determinadas operações em intervalos regulares. A principal estrutura de dados de um protocolo vetor de distância é a tabela de distâncias. Cada roteador mantém uma tabela de distâncias. A tabela de distâncias contém uma linha para cada destino na rede e uma coluna para cada um de seus vizinhos adjacentes. Cada célula da tabela de distâncias indica o custo do caminho até o destino correspondente (linha) através do vizinho adjacente (coluna). Assim, para identificar o caminho de menor custo para um determinado destino basta procurar o resgistro com o menor valor na linha correspondente. Com as informações presentes na tabela de distâncias é possível montar a tabela de roteamento de cada roteador. A tabela de roteamento mostra qual enlace de saída deve ser utilizado para encaminhar os pacotes até um dado destino.

Periodicamente, cada roteador envia uma cópia da sua tabela de roteamento aos seus roteadores adjacentes. Quando um roteador recebe uma mensagem contendo uma tabela de roteamento, ele examina o conjunto de destinos informados e a distância a cada deles.

Se na mensagem recebida houver uma mudança em um caminho conhecido ou um destino novo, o roteador deve atualizar sua tabela. O roteador também deve atualizar sua tabela de distâncias caso detecte uma falha em algum de seus enlaces.

Na Internet, o protocolo RIP (roteamento interno) é do tipo vetor de distância e executa o algoritmo de Bellman-Ford [8, 13]. De acordo com Kurose e Ross [15], o BGP (roteamento externo) tem quase a mesma característica do protocolo vetor de distância, porém ele é mais apropriadamente caracterizado como um protocolo vetor de caminho. Isso porque o BGP em um roteador não propaga informação de custo.

2.2.2 Estado de Enlace

Comer [10] descreve que um roteador que executa um protocolo de roteamento de estado de enlace desempenha duas funções. Primeiro, ele testa ativamente o estado de todos os roteadores vizinhos. Segundo, um roteador periodicamente propaga as informações de estado do enlace para todos os outros roteadores. Para testar o estado de um vizinho diretamente conectado, os dois vizinhos trocam mensagens curtas. Estas mensagens servem para verificar se o outro vizinho está ativo e acessível. Para informar todos os outros roteadores, cada roteador difunde periodicamente uma mensagem que relata o estado de cada um dos seus enlaces. Sempre que o estado do enlace muda, o roteador utiliza um algoritmo para recalcular os caminhos. O algoritmo calcula os caminhos mais curtos até todos os destinos de uma única origem.

De acordo com Huitema [14], os protocolos de estado de enlace são baseados no conceito de manutenção distribuída da topologia da rede. Nestes protocolos todos os nós mantêm uma representação local da topologia da rede, que é atualizada regularmente. A topologia de rede é representada por um grafo, onde cada aresta representa um enlace na rede. Cada aresta é inserida por um roteador que é responsável pelo enlace respectivo. Os enlaces contêm um identificador de interface, o identificador do enlace e informações que descrevem seu estado. Cada roteador pode calcular o caminho mais curto de si mesmo para todos os outros roteadores. Como todos os roteadores têm a mesma representação local da topologia e executam o mesmo algoritmo, as rotas são coerentes e ciclos não

podem ocorrer. Quando um roteador detecta a mudaça no estado de um enlace, ele deve atualizar a aresta respectiva em seu grafo da topologia da rede. Além disso, ele deve transmitir esta informação para os demais roteadores. É necessário tomar precauções para que mensagens antigas, que ainda circulam na rede, não alterem o grafo com informações incorretas. Portanto, cada mensagem contém um contador. Deste modo, é possível ignorar mensagens antigas. Quando um roteador recebe uma nova mensagem, ele procura pelas arestas, contidas na mensagem, em seu grafo. Se alguma das arestas ainda não está presente, então adiciona a aresta no grafo e retransmite a mensagem para os demais roteadores. Se o contador na aresta do grafo for menor do que o contador da mensagem, então atualiza grafo com o valor da aresta da mensagem e retransmite a mensagem para os demais roteadores. Se o contador na aresta do grafo for maior, então transmite o valor desta aresta em uma nova mensagem.

Na Internet, o protocolo OSPF, tema do trabalho descrito nesta monografia, é do tipo estado de enlace e executa o algoritmo de Dijkstra [12].

2.3 OSPF: O Protocolo

O *Open Shortest Path First* (OSPF) é um protocolo de roteamento interno usado para distribuir informações de roteamento dentro de um sistema autônomo. Além disso, o OSPF pertence à categoria de protocolos de roteamento chamados protocolos de estado de enlace. A seguir são apresentadas algumas vantagens e funcionalidades do protocolo OSPF. Em seguida é descrito o algoritmo de Dijkstra, utilizado pelo OSPF para calcular os caminhos entre os roteadores de um sistema autônomo. A seção termina com a descrição dos subprotocolos que fazem parte do OSPF.

2.3.1 Vantagens do OSPF

De acordo com Moy [17], após a falha de um enlace, os melhores caminhos para determinados destinos mudam. Leva algum tempo para qualquer protocolo de roteamento para encontrar os novos melhores caminhos. Enquanto isso, os caminhos utilizados, às vezes

são de qualidade inferior ou até mesmo inutilizáveis. O processo de encontrar o novo caminho é chamado de convergência. Uma das principais vantagens do protocolo OSPF é fazer a convergência de maneira rápida e sem *loops*. Além disso, o OSPF tem como objetivo diminuir a quantidade da largura de banda utilizada pelas mensagens de controle do protocolo. Outras vantagens do protocolo OSPF são descritas com mais detalhes a seguir.

O protocolo RIP usa a contagem de saltos como métrica de roteamento, e o custo de um caminho pode variar entre 1 e 15. Isso criou dois problemas para os administradores de rede. Primeiro, ele limita os diâmetros das suas redes para 15 saltos. Segundo, os administradores não poderiam considerar fatores como a largura de banda ou atraso dos pacotes. No protocolo OSPF a limitação no diâmetro da rede foi removida. Além disso, é possível utilizar diversas métricas no cálculo do caminho mínimo. Para o protocolo OSPF calcular o caminho mínimo ele precisa conhecer a topologia da rede [14]. Sendo assim, é possível utilizar métricas como caminho de maior vazão ou caminho com maior confiabilidade no cálculo do caminho mínimo.

Sistemas autônomos que executam o protocolo RIP têm dificuldade de distinguir quais informações são confiáveis [17], sendo que o protocolo RIP não faz distinção entre informações de roteamento interno ou externo. No protocolo OSPF as informações de roteamento externo são rotuladas e são substituídas por qualquer informação de roteamento interno. Segundo Huitema [14], quando há apenas um roteador de saída para o mundo externo, os protocolos vetor distância e estado de enlace, se comportam de maneira semelhante. Basta anunciar um caminho padrão para esse roteador. Porém quando é preciso escolher entre vários roteadores ou entre vários prestadores de serviços, a solução de anunciar uma rota padrão não é muito eficiente. O protocolo OSPF consegue calcular caminhos para roteadores externos de maneia mais simples e utilizando métricas mais precisas.

O protocolo OSPF oferece a possibilidade de calcular mais de um caminho de custo mínimo para um dado destino [17]. De acordo com Huitema [14], uma maneira de aumentar a eficiência do protocolo OSPF seria dividir o tráfego entre os diversos caminhos

de mesmo custo. Além disso, dividir o tráfego também oferece outras vantagens. Caso o único caminho sendo utilizado se torne indisponível, o tráfego será redirecionado através do caminho alternativo. Isso, possivelmente levará ao congestionamento do novo caminho. Porém, se o tráfego está espalhado por diversos caminhos, apenas uma parte do tráfego terá de ser redirecionada após a falha de um enlace. Dividir o tráfego entre vários caminhos pode ocasionar efeitos prejudiciais, especialmente no caso de conexões TCP. Pacotes encaminhados por caminhos diferentes podem chegar ao destino fora de ordem. Muitas implementações TCP irão desencadear retransmissões desnecessárias. Uma solução para este problema consiste em encaminhar os pacotes relacionados à uma determinada conexão TCP sempre por um único caminho.

2.3.2 Características do OSPF

O protocolo OSPF foi desenvolvido com suporte para: diferenciar *hosts* e roteadores, redes com capacidade de *broadcast*, redes sem capacidade de *broadcast* e dividir redes muito grandes em áreas [14]. Essas funcionalidades são descritas com mais detalhes a seguir.

Os *hosts* em geral estão conectados à Internet através de um roteador. Se for aplicado o modelo do estado de enlace, as relações entre cada *host* e o roteador devem ser inseridas na representção local da topologia, descrita em um grafo. O protocolo OSPF permite uma simplificação, pois todos os hosts da *Ethernet* pertencem a uma única subrede. Sendo assim, é suficiente anunciar um enlace entre o roteador e a rede (neste caso chamada de subrede). O enlace para um roteador vizinho é identificado pelo endereço IP do vizinho. De maneira semelhante, o enlace para uma subrede é identificado pelo endereço da rede ou subrede.

As redes com capacidade de *broadcast* possuem duas características. A primeira é que qualquer roteador pode enviar um pacote para qualquer outro roteador. A segunda é que o pacote é recebido por todos os roteadores. Porém, estas características causam um problema. Dados N roteadores na rede local, existe um total de $N(N-1)/2$ enlaces entre estes roteadores. Então, cada roteador pode anunciar $N-1$ enlaces para os outros

roteadores, mais um enlace para os *hosts* da rede. Levando em conta todos os roteadores o total é de $O(N^2)$ adjacências. A Figura 2.1 mostra uma rede *broadcast* com cinco roteadores na qual cada roteador estabele adjacência com todos os roteadores.

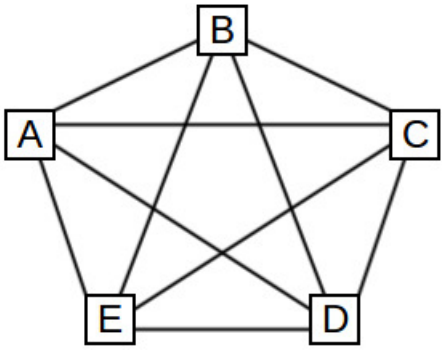

Figura 2.1: Exemplo de $O(N^2)$ adjacências entre roteadores.

O protocolo OSPF reduz este número para somente $O(N)$ adjacências ao selecionar um dos roteadores como roteador *designado*. Os demais roteadores estabelecem adjacências somente com o roteador designado. A Figura 2.2 mostra um rede com cinco roteadores na qual o roteador "A" é selecionado como roteador designado e deve manter adjacência com os outros roteadores. Além disso, um roteador reserva é selecionado juntamente com o roteador designado. Caso o roteador designado falhe, o roteador reserva é utilizado para substitui-lo. Portanto os roteadores também devem estabelecer adjacência com o roteador reserva. Neste caso, cada roteador estabelece no máximo três adjacências, uma com o roteador designado, uma com o roteador reserva e uma com os *hosts* da rede; com excessão dos roteadores designado e reserva que devem estabelecer adjacências com todos os roteadores da rede.

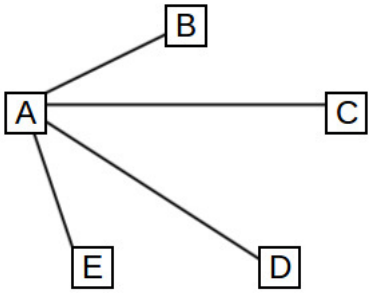

Figura 2.2: Exemplo de $O(N)$ adjacências entre os roteadores, A é o roteador designado.

O procedimento de utilizar um roteador designado reduz a quantidade de tráfego utilizado pelas mensagens de controle do protocolo [18]. Além disso, diminui o tamanho da representação da topologia que guarda os estados dos enlaces da rede.

O protocolo OSPF aplica o mesmo gerenciamento para redes *broadcast* e redes não *broadcast*. Os roteadores selecionam um roteador designado e um roteador reserva. As informações de roteamento são trocadas apenas com estes dois roteadores. O uso de um roteador designado não altera o roteamento das demais informações. Caminhos virtuais são estabelecidos entre qualquer par de roteadores. Os pacotes são transmitidos diretamente sobre estes caminhos virtuais. Mas estes caminhos são estabelecidos apenas sob demanda. Os únicos caminhos que serão usados permanentemente pelas atualizações de roteamento são os que ligam os roteadores com o roteador designado e o roteador reserva.

O protocolo OSPF permite que diversas redes adjacentes sejam agrupadas. Cada grupo, em conjunto com os enlaces entre os roteadores pertencentes ao grupo, é chamado de *área*. Cada área executa uma cópia separada do protocolo de roteamento. Isto significa que cada área tem sua própria representação local da topologia, descrita por um grafo. A topologia de uma área é invisível para as demais áreas. Este isolamento permite reduzir o tráfego das mensagens do protocolo de roteamento. Com a introdução de áreas, as representações da topologia nos roteadores deixam de ser idênticas. Cada roteador tem um grafo separado para cada área em que está conectado. Dois roteadores pertencentes à mesma área têm, para essa área, as representações da topologia idênticas. O roteamento em um sistema autônomo ocorre em dois níveis, dependendo da origem e o destino de um pacote pertencerem a diferentes áreas ou à mesma área. No roteamento dentro de uma área, o pacote é encaminhado utilizando apenas informações obtidas dentro da área. Isso impede que os roteadores de uma determinada área influenciem de maneira nociva o roteamento das demais áreas [14].

2.3.3 Algoritmo de Dijkstra

Como foi visto anteriormente, o protocolo OSPF utiliza um grafo para representar a topologia da rede. A partir deste grafo é possível calcular o caminho mínimo de uma

origem para cada destino. Para encontrar estes caminhos o OSPF usa o algoritmo de Dijkstra [11] também chamado de algoritmo do caminho mínimo, descrito a seguir.

O algoritmo de Dijkstra resolve o problema de encontrar os caminhos mínimos, a partir uma única origem, em um grafo direcionado $G = (V, E)$, com a função peso $w : E \rightarrow R$ que mapeia as arestas para valores de pesos. O peso do caminho $p = v_0, v_1, ..., v_k$ é dado pela soma dos pesos das arestas que constituem o caminho p:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

O peso do caminho mínimo de u para v é definido por

$$\delta(u, v) = \begin{cases} min\{w(p) : u \overset{p}{\rightsquigarrow} v\} & \textit{se existe um caminho de } u \textit{ para } v, \\ \infty & \textit{caso contrário.} \end{cases}$$

Um caminho mínimo do vértice u ao vértice v é então definido como qualquer caminho p com peso $w(p) = \delta(u, v)$.

Os pesos das arestas são muitas vezes utilizados para representar tempo, custo, perda, ou qualquer outra quantidade que se acumula linearmente ao longo de um caminho e que se pretende minimizar. Entretanto as arestas não devem possuir valores negativos.

O algoritmo de Dijkstra utiliza uma técnica chamada relaxamento. Para cada vértice $v \in V$, existe um atributo $d[v]$, que é um limite superior sobre o peso de um caminho mínimo da origem s para v. O atributo $d[v]$ é considerado uma estimativa do caminho mínimo. O atributo $\pi[v]$ é considerado o vértice antecessor ao vértice v. A Figura 2.3 mostra a inicialização da estimativa do caimnho mais curto e seus antecessores.

INICIALIZA(G, s)
1. **para cada** *vértice* $v \in V[G]$
2. **faça** $d[v] \leftarrow \infty$
3. $\pi[v] \leftarrow NIL$
4. $d[s] \leftarrow 0$

Figura 2.3: Pseudo-código do algoritmo responsável por inicializar os atributos d e π.

Após a inicialização, π[v] = NIL para todo $v \in V$, $d[s] = 0$ e $d[v] = \infty$ para todo $v \in V - \{s\}$. A Figura 2.4 mostra que o processo de relaxamento de uma aresta (u, v)

consiste em testar se é possível melhorar o caminho mínimo para v encontrado até agora, passando por u e, nesse caso, atualizar $d[v]$ e $\pi[v]$. Um passo do relaxamento pode diminuir o valor da estimativa do caminho mínimo $d[v]$ e atualizar o atributo do seu antecessor $\pi[v]$. No algoritmo de Dijkstra cada aresta é relaxada exatamente uma vez. O código a seguir executa um passo do relaxamento à aresta (u, v).

```
RELAXAMENTO(u, v, w)
1.   se d[v] > d[u] + w(u, v)
2.     então d[v] ← d[u] + w(u, v)
3.           π[v] ← u
4.
```

Figura 2.4: Pseudo-código do algoritmo responsável pelo relaxamento da aresta (u, v).

A Figura 2.6 mostra que algoritmo de Dijkstra mantém um conjunto de vértices S, no qual o peso de alguns dos caminhos mínimos s já foram determinados. O algoritmo seleciona repetidamente o vértice $u \in V - S$ com a menor estimativa do caminho mínimo, acrescenta u a S, e aplica o relaxamento em todas as arestas que tem origem em u. É utilizada uma fila de vértices Q com prioridade baseada em seus valores da estimativa do caminho mínimo d.

```
DIJKSTRA(G, w, s)
1.   INICIALIZA(G, s)
2.   S ← ∅
3.   Q ← V[G]
4.   enquanto Q ≠ ∅
5.     faça u ← EXTRAI − MINIMO(Q)
6.          S ← S ∪ u
7.          para cada vértice v ∈ Adj[u]
8.                execute RELAXAMENTO(u, v, w)
9.
```

Figura 2.5: Pseudo-código do algoritmo de Dijkstra.

A primeira linha do algoritmo executa a inicialização dos valores de d e π e a segunda linha inicializa o conjunto S como vazio. A terceira linha inicializa a fila de prioridade Q para conter todos os vértices em V, uma vez que $S = \emptyset$. O algoritmo mantém a invariante que $Q = V - S$ no início de cada iteração do laço das linhas 4 até 8. Em cada iteração

deste laço, um vértice u é extraído de $Q = V - S$ e adicionado ao conjunto S, mantendo assim o invariante. Na primeira iteração, $u = s$. Portanto, o vértice u tem a menor estimativa do caminho mínimo de qualquer vértice em $V - S$. Em seguida, nas linhas 7 e 8 acontece o relaxamento de cada aresta (u, v) com origem em u. No relaxamento, a estimativa $d[v]$ e do antecessor $\pi[v]$ são atualizados se o caminho mínimo para v pode ser melhorado, passando por u. É possível observar que os vértices nunca são inseridos em Q após a terceira linha e que cada vértice é extraído de Q e adicionado em S exatamente uma vez, de modo que o laço das linhas 4 a 8 itera exatamente $|V|$ vezes.

A Figura 2.6 mostra uma execução do algoritmo de Dijkstra, na qual a origem é o vértice s, mais à esquerda. Os valores das estimativas de caminho mínimo são mostrados dentro dos vértices. Os vértices pretos estão no conjunto S, e os vértices brancos estão na fila de prioridade $Q = V - S$. O vértice sombreado de cada estado será selecionado na fila de prioridade Q, pois tem o menor valor da estimativa do caminho mínimo. O estado incial (a) mostra o grafo após a inicialização. No estado (b), o vértice s é selecionado e removido da fila Q. Em seguida, é aplicado o relaxamento nas arestas com origem em s e as estimativas de caminho mínimo dos vértices t e y recebem novos valores. No estado (c), o vértice y é selecionado na fila Q. Em seguida, é aplicado o relaxamento. Este processo se repete até que a fila Q fique vazia. O estado (f) mostra os valores finais das estimativas de caminho mínimo e as arestas sombreadas indicam os vértices antecessores.

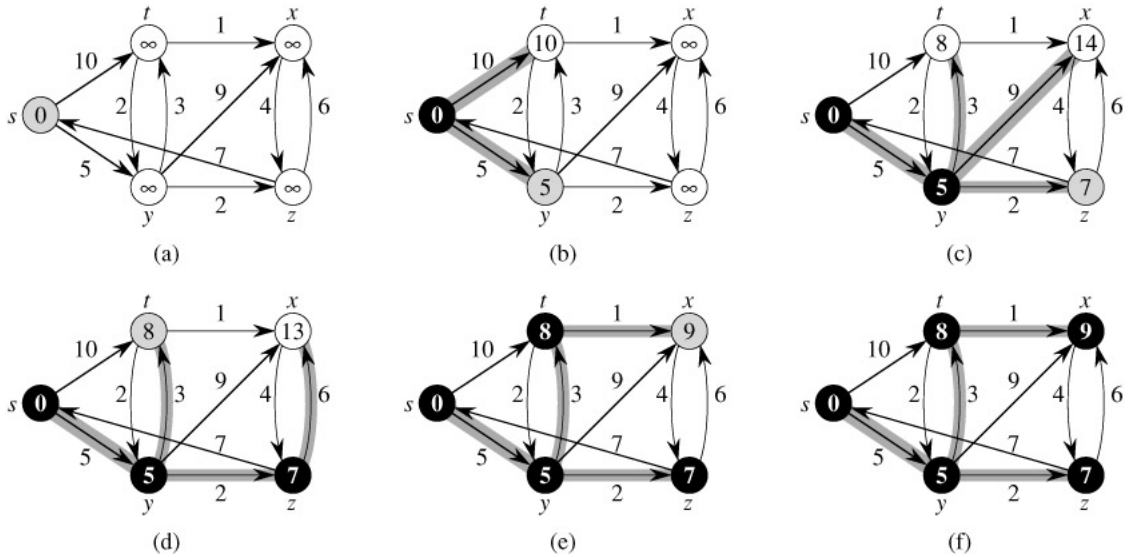

Figura 2.6: Um exemplo de execução do algoritmo de Dijkstra [11].

2.3.4 Os Protocolos Internos ao OSPF

O protocolo OSPF é composto de três subprotocolos: *Hello*, *Exchange* e *Flooding* [14], são descritos a seguir. A descrição é baseada em [17, 14].

O Protocolo Hello

O protocolo *Hello* é utilizado para dois propósitos. Verificar o estado dos enlaces e eleger um roteador designado e um roteador reserva. Para isso, pacotes denominados *hello* são enviados pelos roteadores a cada *hello-interval* segundos. Os pacotes incluem o endereço do roteador designado, ou 0 se ainda não houver roteador designado, e o endereço do roteador reserva ou 0. Além disso, também incluem uma lista de todos os vizinhos dos quais um pacote *hello* foi recebido nos últimos *dead-interval* segundos. Ambos os intervalos *hello-interval* e *dead-interval* são parâmetros dos enlaces que são configurados pelo administrador da rede. A ligação entre dois roteadores é dita *operacional* se os pacotes podem fluir em ambas as direções. A conectividade bidirecional é identificada pela lista de roteadores conhecidos por um roteador vizinho. Se o identificador do roteador local não estiver listado nos pacotes *hello* enviados pelos roteadores vizinhos, isso significa que eles ainda não receberam os pacotes *hello* enviados pelo roteador local. O enlace é então declarado de *sentido único* e não pode ser usado para o roteamento. Se o roteador local está listado nos pacotes *hello* dos roteadores vizinhos, então o enlace é *bidirecional*.

Depois de estabelecer uma conexão bidirecional, os roteadores começam a determinar suas adjacências. Primeiro é preciso selecionar o roteador designado e o roteador reserva. O processo de eleição usa o campo de *prioridade* presente nos pacotes *hello*. Cada roteador é configurado com uma prioridade, que varia entre 0 e 255. Como resultado da eleição é selecionado o roteador com a maior prioridade. Se o roteador de maior prioridade fica inativo, outro será selecionado; esta seleção permanecerá mesmo depois do roteador de maior prioridade se tornar ativo novamente. Roteadores com prioridade zero, nunca são selecionados como roteador designado.

Imediatamente após os enlaces se tornarem ativos, o roteador permanece em um estado de *espera* durante um intervalo igual ao *dead-interval*. Durante este intervalo, o roteador

irá transmitir pacotes de *hello*, mas não vai se candidatar a roteador designado ou reserva. Ele irá receber pacotes *hello* e inicializar os identificadores dos roteadores designado e reserva da seguinte forma. Para cada vizinho, o roteador guarda a prioridade do vizinho e o estado do enlace. Além disso, ele guarda se o vizinho se candidata como roteador designado ou reserva. Se um ou vários vizinhos se candidatam como roteador reserva, aquele com maior prioridade é selecionado. Em caso de empate, o roteador com o maior identificador é selecionado. Se nenhum vizinho se candidata a roteador reserva, o vizinho com a maior prioridade é selecionado ou, em caso de empate, aquele com o maior ID. Se um ou vários vizinhos se candidatam a roteador designado, aquele com a maior prioridade é selecionado como roteador designado. Em caso de empate, o roteador com o maior identificador é selecionado. Se nenhum vizinho se candidata a roteador designado, o reserva é *promovido*. Um roteador não pode se candidatar a ambos roteador designado e reserva. Assim, se for preciso promover o roteador reserva para roteador designado, também será necessário eleger um novo roteador reserva.

O Protocolo Exchange

Quando dois roteadores estabelecem conectividade bidirecional, eles devem sincronizar sua representação local da topologia da rede. Em redes *broadcast*, isso ocorre entre os roteadores e o roteador designado ou o roteador reserva. A sincronização inicial é realizada através do protocolo *Exchange*. O protocolo *Flooding*, em seguida, é responsável por manter as representações da topologia da rede sincronizadas. O protocolo *Exchange* é assimétrico. A primeira etapa do protocolo tem a função de selecionar um mestre e um escravo. Depois de concordar sobre esses papéis, os dois roteadores trocam a descrição de seus grafos, que representam a topologia da rede. Cada um irá listar os enlaces que serão requisitados em seguida.

No pacote utilizado pelo protocolo *Exchange* existem três *bits* utilizados para controle: I (Inicializa), M (Mais) e MS (Mestre-Escravo). O roteador que pretende iniciar o procedimento de sincronização envia um pacote vazio, com os *bits* I, M e MS definidos como 1 e o contador definido para um valor arbitrário, não visto anteriormente pelo outro roteador.

O outro roteador concorda em fazer a função de *escravo* durante a sincronização enviando um pacote igual ao que foi recebido, porém com o *bit* MS definido como 0. Uma vez que os papéis foram distribuídos, o mestre irá enviar a descrição dos enlaces presentes na representação local da topologia em uma sequência de pacotes, nos quais o *bit* I será definido como 0, o *bit* MS definido como 1 e o *bit* M definido como 1, exceto para o último pacote. Os pacotes têm sequência numerada e devem ser enviados um de cada vez. Depois de cada pacote ser enviado, o escravo irá enviar uma confirmação de que recebeu o pacote. Os pacotes de confirmação contêm a descrição dos enlaces presentes na representação local da topologia do roteador escravo. Se a confirmação não for recebida dentro do intervalo de retransmissão, o mestre retransmite o seu pacote. Quando o escravo recebe um pacote com o valor do contador igual ao valor do pacote anterior, deve retransmitir o último pacote de confirmação. Quando o mestre transmite sua última descrição dos enlaces, ele vai definir o *bit* M como 0. Se o escravo ainda tem descrições de enlaces para transmitir, ele irá enviar uma confirmação com o *bit* M definido como 1. O mestre continuará enviando pacotes de descrição vazios com o *bit* M definido como 0. Como consequência, continuará aceitando confirmações, até que eventualmente receba uma confirmação com o *bit* M definido como 0. Nesse ponto a primeira etapa da sincronização está completa.

Durante a sincronização, o mestre e o escravo processam as descrições dos estados de enlace que se encontram nos pacotes e nas confirmações. Primeiro verificam se não há um enlace com a mesma descrição na representção local da topologia. Além disso verificam, através do valor de um contador, se esse enlace não está desatualizado. Se alguma das condições for satisfeita, os roteadores devem colocar a descrição do enlace em uma lista de *enlaces pendentes*. Na segunda etapa da sincronização, os roteadores irão solicitar informações dos enlaces pendentes. Após receber uma solicitação, cada roteador deve enviar um ou mais pacotes contendo um conjunto de atualizações de estado de enlace. A transmisão destes pacotes utiliza exatamente os mesmos procedimentos do protocolo *Flooding* descrito a seguir.

O Protocolo Flooding

Quando um enlace muda de estado, o roteador responsável por esse enlace vai transmitir uma mensagem com a nova versão do estado do enlace. Para cada enlace, existe um contador que é comparado com o valor na representação local da topologia. Se o valor do contador indicar que o enlace presente na mensagem recebida é novo, então a mensagem recebida deve ser retransmitida em todas as interfaces. Em qualquer caso, o roteador que enviou a mensagem deve aguardar uma confirmação de recebimento. Para tornar o protocolo *Flooding* confiável, o roteador deve retransmitir suas atualizações em intervalos regulares até que receba a confirmação. Cada pacote de confirmação pode ser responsável por informar o recebimento de uma ou mais mensagens, com informações de estado de enlace. Portanto é normal atrasar a sua transmissão, a fim de agrupar a confirmação, de recebimento do estado de vários enlaces, em um único pacote. Este atraso deve ser curto, a fim de evitar retransmições desnecessárias.

CAPÍTULO 3

INSTALANDO O SIMULADOR NS-3 PARA EXECUTAR OSPF

Este capítulo descreve os procedimentos necessários para instalar e configurar o simulador de redes *Network Simulator 3* (NS-3) [4]. Inicialmente, são apresentadas as características e as funcionalidades do NS-3. Em seguida, são descritas as instruções utilizadas para instalar e configurar o NS-3.

3.1 Características e Funcionalidades

O NS-3 é um simulador de rede de eventos discretos, voltado principalmente para pesquisa e uso educacional [4]. O simulador NS-3 é um software livre, licenciado sob a licença GNU GPLv2, e está disponível publicamente para a pesquisa e desenvolvimento.

O simulador NS-3 oferece suporte à pesquisa em redes IP e redes não baseadas em IP. No entanto, a grande maioria de seus usuários se concentra em simulações de redes sem fio que envolvem modelos para, por exemplo, Wi-Fi, WiMAX ou LTE.

Outra funcionalidade do simulador está na reutilização de aplicações reais. Atualmente, estão sendo testados módulos do NS-3 para a execução de aplicações reais e o uso dos protocolos da Internet implementados no kernel do Linux.

O NS-3 é construído como um sistema de bibliotecas de *software*. Estas bibliotecas são escritas na linguagem orientada a objetos conhecida como C++. Entretanto os *scripts* de simulação não precisam ser escritos em C++, eles podem ser escritos em Pyhton.

3.2 Instalação e Configuração

Atualmente, existe uma ferramenta para auxiliar na instalação do simulador NS-3 chamada *Bake* [6]. *Bake* é uma ferramenta utilizada na instalação e configuração do NS-3 e seus módulos. A máquina utilizada para simulações do protocolo OSPF usa um sistema operacional baseado no Ubuntu 12.04. Portanto, os pacotes necessários foram instalados através do gerenciador de pacotes do Ubuntu, o *apt-get*. A seguir é apresentada uma série de procedimentos necessários para o uso do simulador NS-3.

Para usar a ferramenta *Bake* é preciso ter a linguagem Python (versão 2.6 ou superior) instalada. Além disso é necessária a ferramenta de controle de versão Mercurial utilizada para fazer o *download* do *Bake*.

```
sudo apt-get install python python-dev mercurial
```

Em seguida é necessário fazer o *download* da ferramenta *Bake* usando o Mercurial.

```
hg clone http://code.nsnam.org/bake
```

Isso cria um repositório da ferramenta *Bake* no diretório. Para facilitar o uso da ferramenta *Bake*, é preciso adicionar ao *PATH* o endereço do diretório onde foi criado o repositório do *Bake*.

```
export BAKE_HOME=‘pwd‘/bake
export PATH=$PATH:$BAKE_HOME
export PYTHONPATH=$PYTHONPATH:$BAKE_HOME
```

Depois disso, é possível utilizar o *Bake* para encontrar os pacotes pendentes. Para descobrir quais pacotes estam faltando e são necessários para a instalação do NS-3 e seus módulos basta executar o segtuinte comando:

```
bake.py check
```

Com isso é possível visualizar a seguinte lista de pacotes a serem utilizados:

Compilador GNU C++.

```
sudo apt-get install gcc g++
```

CVS, GIT e Bazaar.

```
sudo apt-get install cvs git-core bzr
```

Ferramentas Tar, Unzip e Unrar. Além dos compressores de dados 7z e XZ.

```
sudo apt-get install tar unzip unrar p7zip-full xz-utils
```

Make e cMake.

```
sudo apt-get install make cmake
```

Patch tool.

```
sudo apt-get install patch
```

Autoreconf tool.

```
sudo apt-get install autoconf
```

Também foram instalados alguns pacotes adicionais para visualizar os resultados das simulações e fazer depuração de código.

```
sudo apt-get install tcpdump gdb valgrind
```

Na implementação do NS-3 existe um conjunto de classes que tem o objetivo de simular o funcionamento de um protocolo de roteamento baseado no OSPF. Como foi visto anteriormente, o protocolo OSPF é dividido em três subprotocolos: os protocolos *Hello*, *Exchange* e *Flooding*. Entretanto, a implementação do simulador NS-3 não simula o funcionamento dos protocolos *Hello* e *Flooding*. É possível observar que somente o funcionamento do protocolo *Exchange* é simulado no NS-3. Sendo assim, o NS-3 não simula o funcionamento completo do protocolo OSPF. Deste modo, o estudo do protocolo OSPF fica muito limitado. Para aprofundar a análise do OSPF é necessário o uso de uma ferramenta adicional. Existe um módulo para o simulador NS-3 chamado *Direct Code Execution* (DCE) [7]. O módulo DCE permite a execução de algumas aplicações reais dentro do NS-3. Umas das aplicações que o DCE fornece suporte é a implementação do protocolo OSPF presente na *Quagga Routing Suite* [5]. Ao utilizar o módulo DCE é

possível ampliar o estudo do protocolo OSPF e deixá-lo mais abragente. O DCE precisa dos seguintes pacotes.

```
sudo apt-get install automake flex bison wget indent pkg-config gawk
   libssl-dev libsysfs-dev libc6-dbg
```

Neste ponto todos os pacotes necessários para instalar o NS-3 e simular o OSPF estão instalados. Agora é preciso configurar o *Bake* para utilizar os seguintes módulos. O módulo dce-quagga que permite usar o protocolo OSPF, implementado pelo Quagga, nas simulações. Além do módulo dce-linux que permite utilizar a implementação do protocolo IP nativo do Linux.

```
mkdir dce
cd dce
bake.py configure -e dce-linux-1.1 -e dce-quagga-1.1
```

É possível vizualizar quais módulos serão utilizados e os requisitos específicos do sistema para esta configuração, através do seguinte comando.

```
bake.py show
```

Por último, para baixar, configurar e instalar o simulador NS-3 além dos módulos necessários basta executar o segtuinte comando:

```
bake.py deploy
```

Após a execução do comando anterior é possível verificar se a instalação do módulo Quagga foi bum sucedida.

```
cd source/ns-3-dce
./test.py -s dce-quagga
```

Depois da conclusão da instalação, o simulador NS-3 está disponível. Além disso, o protocolo OSPF, implementado na *Quagga Routing Suite*, está acessível para realizar simulações através do NS-3.

CAPÍTULO 4

SIMULAÇÃO DO OSPF NO NS-3

Este capítulo, descreve a execução de simulações com o protocolo OSPF no simulador de redes NS-3. Primeiro, são apresentadas as instruções utilizadas para criar a simulação no NS-3. Em seguida, são descritos os procedimentos necessários para utilizar o módulo DCE nas simulações do OSPF. O capítulo termina com os resultados das simulações que utilizam o módulo DCE.

4.1 OSPF no NS-3

A primeira etapa para realizar a simulação de uma rede é definir a topologia desta rede. A Figura 4.1 mostra a topologia escolhida para as simulações, com cinco roteadores contectados por enlaces pono-a-ponto.

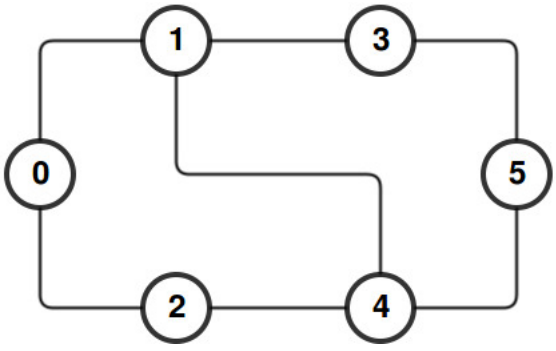

Figura 4.1: Representação da topologia da rede utilizada nas simulações do protocolo OSPF.

Em seguida, é necessário montar a topologia da rede no NS-3. Porém, antes de escrever códigos no NS-3 é importante entender os conceitos e abstrações do sistema.

Na Internet, um dispositivo computacional que conecta-se a uma rede é chamado de *host*. Devido ao fato do NS-3 ser um simulador de rede, e não um simulador da Internet,

o termo *host* é intencionalmente não utilizado, pois está intimamente associado com a Internet e seus protocolos. Ao invés disso, é utilizado o termo *node* (nó) que é um termo mais genérico e também usado por outros simuladores. A abstração de um dispositivo computacional básico é chamado então de nó. Essa abstração é representada em C++ pela classe *Node*. Esta classe fornece métodos para gerenciar as representações de dispositivos computacionais nas simulações. O nó deve ser pensado como um computador no qual se adicionam funcionalidades, tal como aplicativos, pilhas de protocolos e periféricos com seus *drivers* associados que permitem ao computador executar tarefas úteis.

Normalmente, programas de computador são divididos em duas classes. *Programas de sistema* que organizam recursos do computador, tais como memória, processador, disco, rede, entre outros. Tais programas normalmente não são utilizados diretamente pelos usuários. Na maioria das vezes, os usuários fazem uso de *aplicações*, que usam os recursos controlados pelos programas de sistema para atingir seus objetivos. Geralmente, a separação entre programas de sistema e aplicações de usuários é feita pela mudança no nível de privilégios que acontece na troca de contexto feita pelo sistema operacional. No NS-3, não existe um conceito de sistema operacional real, não há o conceito de níveis de privilégios nem chamadas de sistema. Há apenas aplicações que são executadas nos nós para uma determinada simulação. No NS-3, a abstração básica para um programa de usuário que gera alguma atividade a ser simulada é a aplicação. Esta abstração é representada em C++ pela classe *Application*, que fornece métodos para gerenciar a representação de suas versões de aplicações a serem simuladas.

No mundo real, computadores estão conectados em uma rede. Normalmente, o meio sobre o qual os dados trafegam é chamada de canal (*channel*). Quando um cabo Ethernet é ligado ao conector na parede, na verdade está se conectando a um canal de comunicação Ethernet. No mundo simulado do NS-3, um nó é conectado a um objeto que representa um canal de comunicação. A abstração de canal de comunicação é representada em C++ pela classe *Channel*. A classe textitChannel fornece métodos para gerenciar objetos de comunicação de sub-redes e nós conectados a eles.

Para conectar os computadores em uma rede é necessário uma placa de rede. A placa

de rede não funciona sem o driver que a controle. No Unix (ou Linux), um periférico (como a placa de rede) é classificado como um dispositivo (*device*). As placas de rede, também chamadas de dispositivos de rede (*net devices*), são controladas através de *drivers* de dispositivo de rede (*network device drivers*). No NS-3 a abstração do *dispositivo de rede* cobre tanto o *hardware* (placa de rede) quanto o *software* (*driver*). Um dispositivo de rede é "instalado" em um nó para permitir que este se comunique com outros na simulação, usando os canais de comunicação (*channels*). Assim como em um computador real, um nó pode ser conectado a mais que um canal via múltiplos dispositivos de rede. A abstração do dispositivo de rede é representado em C++ pela classe *NetDevice*, que fornece métodos para gerenciar conexões para objetos *Node* e *Channel*.

São necessárias diversas operações distintas para criar um dispositivo, instalar o dispositivo em um nó, configurar a pilha de protocolos no nó em questão e por fim, conectar o dispositivo a um canal. Além disso é necessário conectar múltiplos dispositivos e então fazer a interconexão das várias redes. Para facilitar o trabalho, são disponibilizados objetos que são assistentes de topologia (*topology helpers*), que combinam estas operações distintas de modo conveniente. A seguir são utilizados diversos destes assistentes para montar a topologia da rede utilizada nas simulações.

As próximas duas linhas de código criam os objetos do tipo *Node* que representarão os roteadores na simulação.

```
NodeContainer routers;
routers.Create (6);
```

O assistente *NodeContainer* fornece uma forma conveniente de criar, gerenciar e acessar qualquer objeto *Node* que criamos para executar a simulação. A primeira linha declara um *NodeContainer* chamado *routers*. A segunda linha chama o método *Create* sobre o objeto *routers* e pede para criar seis nós.

Os roteadores, como estão no código, não fazem nada. O próximo passo é conectar os roteadores com enlaces. Uma forma simples de conectar dois roteadores em uma rede é com um enlace ponto-a-ponto. A instrução seguinte, instancia o objeto *PointToPointHelper* que é utilizado para configurar as placas de rede e os enlaces ponto-a-ponto

que conectam os roteadores.

```
PointToPointHelper pointToPoint;
```

A próxima instrução, informa ao objeto *PointToPointHelper* para usar o valor de "5Mbps" (cinco megabits por segundo) como *DataRate* (taxa de transferência) das placas de rede que são utilizadas pelos roteadores.

```
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
```

Semelhante ao *DataRate* presente na placa de rede também existe o atributo *Delay* (atraso) associado ao enlace. A instrução a seguir, informa ao *PointToPointHelper* para usar o valor de "2ms" (dois milissegundos) como valor de atraso de transmissão para os enlaces utilizados para conectar os roteadores.

```
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
```

Agora que o *PointToPointHelper* contém as configurações das placas de rede e dos enlaces é preciso criar, configurar e instalar estes dispositivos. Para isso é utilizado o método *Install* do *PointToPointHelper* que recebe como parâmetro dois roteadores que dividem um enlace na nossa representação da topologia. Internamente, para cada roteador é criada uma placa de rede (*NetDevice*). Além disso, é criado um enlace (*Channel*) e as duas placas de rede são conectadas. Como os objetos são criados pelo *PointToPointHelper*, os atributos, passados anteriormente, são configurados pelo assistente. No código abaixo o *NetDeviceContainer* e_ij representa um enlace no qual i e j são os identificadores dos roteadores que dividem este enlace.

```
NetDeviceContainer e_01 = pointToPoint.Install (routers.Get (0), routers.
   Get (1));
```

Por exemplo, o *NetDeviceContainer* e_01 é o enlace responsável pela conexão entre o roteador 0 e o roteador 1. As instruções a seguir são responsáveis pela criação e configuração dos outros enlaces da rede.

```
NetDeviceContainer e_02 = pointToPoint.Install (routers.Get (0), routers.
   Get (2));
NetDeviceContainer e_13 = pointToPoint.Install (routers.Get (1), routers.
   Get (3));
NetDeviceContainer e_14 = pointToPoint.Install (routers.Get (1), routers.
   Get (4));
NetDeviceContainer e_24 = pointToPoint.Install (routers.Get (2), routers.
   Get (4));
NetDeviceContainer e_35 = pointToPoint.Install (routers.Get (3), routers.
   Get (5));
NetDeviceContainer e_45 = pointToPoint.Install (routers.Get (4), routers.
   Get (5));
```

Após a execução de todas as instruções anteriores, a rede vai conter seis roteadores conectados por canais ponto-a-ponto. Todos os dispositivos são configurados para ter uma taxa de transferência de dados de cinco *megabits* por segundo que, por sua vez, tem um atraso de transmissão de dois milissegundos. Os roteadores e suas placas de rede estão configurados, porém não existe nenhuma pilha de protocolos instalada. As próximas instruções são responsáveis por isso.

```
InternetStackHelper stack;
stack.Install (routers);
```

O *InternetStackHelper* é um assistente de topologia inter-rede. O método *Install* utiliza como parâmetro o *NodeContainer routers* que contém todos os roteadores da rede. Quando o método é executado, ele irá instalar em todos os roteaores a pilha de protocolos da Internet incluindo, por exemplo, os protocolos IP, UDP, TCP, entre outros.

Em seguida é necessário associar endereços IP's aos roteadores. Para isso existe um assistente de topologia (*Ipv4AddressHelper*) que gerencia a alocação de endereços IP's.

```
Ipv4AddressHelper ipv4AddrHelper;
```

Para facilitar na identificação dos roteadores, que pertencem a um determinado enlace,

são utilizados IP's no formato 10.i.j.0 no qual i e j são os identificadores dos roteadores que pertencem ao enlace. No código abaixo o método *SetBase* do *Ipv4AddressHelper* configura alocação de IP's no formato definido. Além disso é utilizada a máscara 255.255.255.0. Por padrão, os endereços alocados iniciam do primeiro endereço IP disponível e são incrementados um a um. Como são utilizados enlaces ponto-a-ponto um dos roteadores recebe o endereço 10.i.j.1 e o outro roteador recebe 10.i.j.2. O endereçamento é feito a partir dos enlaces e_ij criados anteriormente. O método *Assign* recebe como parâmetro um destes enlaces e atribui os endereços IP aos roteadores deste enlace. Cada *Ipv4InterfaceContainer* i_ij contém uma lista das interfaces de rede criadas pelo assistente de topologia na qual i e j são os identificadores dos roteadores que pertem à esta lista. O NS-3 mantém todos os endereços IP's alocados e gera um erro fatal se o mesmo endereço for usado duas vezes.

```
ipv4AddrHelper.SetBase ("10.0.1.0", "255.255.255.0");
Ipv4InterfaceContainer i_01 = ipv4AddrHelper.Assign (e_01);
```

Nas intruções acima os roteadores do enlace e_01 recebem os endereços IP 10.0.1.1 e 10.0.1.2. O *Ipv4InterfaceContainer* i_01 guarda esta informação para consultas futuras. Este procedimento deve ser repetido para os demais enlaces da rede.

```
ipv4AddrHelper.SetBase ("10.0.2.0", "255.255.255.0");
Ipv4InterfaceContainer i_02 = ipv4AddrHelper.Assign (e_02);
ipv4AddrHelper.SetBase ("10.1.3.0", "255.255.255.0");
Ipv4InterfaceContainer i_13 = ipv4AddrHelper.Assign (e_13);
ipv4AddrHelper.SetBase ("10.1.4.0", "255.255.255.0");
Ipv4InterfaceContainer i_14 = ipv4AddrHelper.Assign (e_14);
ipv4AddrHelper.SetBase ("10.2.4.0", "255.255.255.0");
Ipv4InterfaceContainer i_24 = ipv4AddrHelper.Assign (e_24);
ipv4AddrHelper.SetBase ("10.3.5.0", "255.255.255.0");
Ipv4InterfaceContainer i_35 = ipv4AddrHelper.Assign (e_35);
ipv4AddrHelper.SetBase ("10.4.5.0", "255.255.255.0");
Ipv4InterfaceContainer i_45 = ipv4AddrHelper.Assign (e_45);
```

Após a execução do código anterior todos os roteadores possuem um endereço IP para cada enlace ao qual eles fazem parte. Neste ponto, a rede está funcionando, com pilhas de protocolos instaladas e endereços IP's configurados. Agora é neccessário que os roteadores presentes na rede utilizem o protocolo OSPF. Para isso é utilizado o *Global Router Manager* que é responsável por preencher as tabelas de roteamento do roteadores. Entretanto, o *Global Router Manager* não implementa o protocolo OSPF. Uma vez que, ele somente calcula caminhos estáticos, pois as tabelas de roteamento são preenchidas antes de iniciar o tráfego de pacotes na rede. Portanto, não são utilizados pacotes de controle e quando ocorre uma falha na rede os rotedores não são informados. Porém, a criação das tabelas de roteamento funciona de maneira semelhante ao OSPF, pois o *Global Router Manager* é baseado na implementação do OSPF feita pela *Quagga Routing Suite*.

Na instrução a seguir, o método *PopulateRoutingTables* do *Ipv4GlobalRoutingHelper* tem a função de preencher as tabelas de rotemento. Este método cria as representações locais da topologia da rede. Em seguida, o algoritmo de Dijkstra calcula os caminhos mínimos e inicializa as tabelas de roteamento dos roteadores na simulação.

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

A próxima etapa é executar o simulador. Isto é feito usando a instrução a seguir.

```
Simulator::Run ();
```

O código da simulação está completo e pronto para uso. Porém não existe nenhuma aplicação instalada em nenhum dos roteadores. Portanto a simulação não produz nenhum resultado. O NS-3 oferece diversas aplicações com diversas funcionalidades e cada uma delas produz um resultado diferente. Entretanto os resultados das aplicações seguem um padrão. As aplicações são utilizadas para gerar tráfego de rede. Além disso, as aplicações mostram para o usuário informações sobre os pacote enviados ou recebidos. A informações exibidas são, por exemplo, o IP de origem ou destino e o tamanho do pacote.

O NS-3 fornece diversos mecanismos de monitoramento. Um deles é o uso da biblioteca *pcap*, que permite a captura de pacotes [1]. Com o uso da *pcap* é possível visualizar

informações detalhadas de todos os pacotes enviados ou recebidos por qualquer roteador da rede. Para habilitar o rastreamento *pcap* é usado o assistente *PointToPointHelper* que informa ao NS-3 para criar arquivos que guardam informações do pacotes transmitidos em enlaces ponto-a-ponto. Os arquivos são criados no seguinte formato, o prefixo *ospf*, o número do roteador, o número da interface deste roteador e o sufixo *.pcap*. O programa mais conhecido para ler o mostrar este formato é o *Wireshark* [2]. Entretanto, existem muitos analisadores de tráfego que usam este formato como, por exemplo, o *tcpdump* [3].

```
pointToPoint.EnablePcapAll ("ospf");
```

Um dos principais objetivos das simulações é observar o comportamento do protocolo OSPF em situações que ocorrem falhas na rede. Para isso é preciso simular uma falha na rede. Existe mais de uma maneira de simular uma falha. Um método é simular a falha de um enlace. Isto pode ser feito no NS-3 ao destivar a interface de rede de um roteador. O código a seguir é responsável por isso.

```
Ptr<Node> r0 = routers.Get (0);
Ptr<Ipv4> router_0 = r0->GetObject<Ipv4> ();
uint32_t interface_1 = 1;
```

Na primeira instrução *Ptr<Node> r0* representa um apontador para o roteador 0 que contém a interface que irá falhar. Na segunda instrução *Ptr<Ipv4> router_0* representa um apontador para o conjunto de interfaces de redes do roteador 0. Na terceira instrução *uint32_t interface_1* representa qual das interfaces do roteador 0 foi selecionada para falhar.

Além disso, é necessário criar um evento na simulação. A instrução a seguir cria um evento que é executado aos 10 segundos da simulação. O parâmetro *Ipv4::SetDown* determina que o evento deve desativar um das interfaces de rede de um roteador. Os últimos parâmetros informam que a interface 1 do roteador 0 deve ser desativada. A interface 0 dos roteadores é atribuída ao endereço de *loopback*. Portanto a interface 1 é atribuída ao enlace entre o roteador 0 e o roteador 1.

```
Simulator::Schedule (Seconds (10.0), &Ipv4::SetDown, router_0,
   interface_1);
```

É possível perceber nas simulações que quando ocorre uma falha os pacotes que utilizam o caminho falho não chegam aos seus destinos. Ao utilizar um aplicação para gerar tráfego no caminho com falha é posível observar que os pacotes são enviados porém não são recebidos. Esse comportamento se mantém por tempo indefinido. Deste modo, é possível perceber que a implementação do *Global Router Manager* é muito limitada. Uma vez que, quando um enlace falha, os roteadores da rede deveriam calcular novos caminhos. Porém isso não acontece nas simulações. Neste ponto é perceptível que não é simulado o funcionamento dos protocolos *Hello* e *Flooding*. Somente o funcionamento do protocolo *Exchange* é simulado no NS-3. Sendo assim, o módulo DCE passa a ser uma alternativa para simular o protocolo OSPF. Pois o DCE possui a implementação do protocolo OSPF feita pela *Quagga Routing Suite*. No DCE todos os subprotocolos que compõem o OSPF estão implementados. Portanto, é possível executar simulações com falhas na redes.

4.2 OSPF no DCE

Para utilizar o protocolo OSPF disponibilizado pelo DCE são necessárias algumas modificações no código. Primeiramente, são adicionados dois procedimentos para auxiliar na configuração das interfaces dos roteadores. Como foi visto anteriormente, o DCE permite utilizar a implementação do protocolo da Internet presente no sistema operacional Linux. Os procedimentos a seguir fazem uso dessa funcionalidade.

No primeiro procedimento a seguir é utilizado um assistente *DceApplicationHelper* denominado *process* para configurar algumas da aplicações que serão executadas nas simulações. O objeto *ApplicationContainer* chamado de *apps* é utilizado para gerenciar as aplicações. A função *SetBinary* define qual aplicação será usada. Como este procedimento tem o objetivo de auxiliar na configuração das interfaces de rede dos roteadores é usado o executável *ip*, nativo do Linux. Em seguida, a função *SetStackSize* define o tamanho da pilha utilizada pelo executável *ip*, neste caso é usado um valor padrão do DCE. Em

seguida, a função *ResetArguments* retira os parâmetros, do executável *ip*, que estão configurados no sistema operacional. A função *ParseArguments* adiciona uma *string* como parâmetro para o executável *ip*. A função *Install* instala o executável no *Ptr<Node> router* que representa um roteador. Por fim, a função *Start* define em qual momento da simulação o executável deve iniciar sua execução. Para isso é utilizado o parâmetro *at*

```
static void RunIp (Ptr<Node> router, Time at, std::string str) {
    DceApplicationHelper process;
    ApplicationContainer apps;
    process.SetBinary ("ip");
    process.SetStackSize (1 << 16);
    process.ResetArguments ();
    process.ParseArguments (str.c_str ());
    apps = process.Install (router);
    apps.Start (at); }
```

O segundo procedimento é utilizado para adicionar endereços a um determinado roteador *Ptr<Node> router*. Primeiro é criada uma variável *std::ostringstream oss* para guardar as configurações do endereço. Na instrução seguinte, o parâmetro "-f inet" informa que é um endereço do tipo IPv4. o parâmetro "addr add *address*" define qual endereço será utilizado. o parâmetro "dev *name*" informa qual interface de rede receberá o endereço definido. Por fim, é chamado o procedimento *RunIp* para utilizar o executável *ip* com as configurações definidas.

```
static void AddAddress (Ptr<Node> router, Time at, const char *name,
   const char *address)
{
    std::ostringstream oss;
    oss << "-f inet addr add " << address << " dev " << name;
    RunIp (router, at, oss.str ());
}
```

O executável *ip* é utilizado para realizar o endereçamento dos roteadores presentes na rede. Portanto as instruções reponsáveis por instalar a pilha de protocolos e associar os endereços IP são modificadas. Para instalar a pilha de protocolos é utilizado um assistente *DceManagerHelper* denominado *processManager*. O método *SetTaskManagerAttribute* é usado para definir alguns atributos do DCE relacionados a execução de processos. Um desses atributos é o *FiberManagerType* que é usado para alternar o contexto de execução entre *threads* e processos. Por padrão este atributo é configurado para auxiliar na depuração de código, mas neste caso, o parâmetro *EnumValue (0)* altera seu comportamento para funcionar de maneira mais eficiente. O método *SetNetworkStack* é responsável por informar ao assistente qual pilha de protocolos da Internet deve ser usada. Neste caso deve ser usada a pilha de protocolos da Internet nativa do Linux. Por fim, o método *Install* configura todos os roteadores da rede com os atributos definidos.

```
DceManagerHelper processManager;
processManager.SetTaskManagerAttribute ("FiberManagerType", EnumValue (0)
   );
processManager.SetNetworkStack ("ns3::LinuxSocketFdFactory", "Library",
   StringValue("liblinux.so"));
processManager.Install (routers);
```

O próximo passo é definir os endereços IP dos roteadores. Para isso são utilizados os procedimentos *AddAddress* e *RunIp* criados no início desta seção.

```
AddAddress (routers.Get (0), Seconds (0.1), "sim0", "10.0.1.1/24");
AddAddress (routers.Get (0), Seconds (0.1), "sim1", "10.0.2.1/24");
```

O procedimento *AddAddress* define que no tempo 0.1 as interface sim0 e sim1, do roteador 0, devem receber os endereços 10.0.1.1 e 10.0.2.1 respectivamente. Além disso é informado que somente o um *byte* é usado para endereçamento de *hosts*. Os outros três *bytes* são usados para endereçamento de redes. De maneira semelhante a simulação criada sem utilizar o DCE, são utilizados IP's no formato $10.i.j.0$ no qual i e j são os identificadores dos roteadores que pertencem ao enlace. No código a seguir é utilizado o

procedimento *RunIp* para tornar ativas as interfaces do roteador 0 que no tempo 0.11. Os valores de *tempo* utilizados são padrões do DCE.

```
RunIp (routers.Get (0), Seconds (0.11), "link set lo up");
RunIp (routers.Get (0), Seconds (0.11), "link set sim0 up");
RunIp (routers.Get (0), Seconds (0.11), "link set sim1 up");
```

As instruções anteriores configuram somente as interfaces do primeiro roteador. Entretanto, as interfaces de todos os roteadores devem ser configuradas. Para isso basta utilizar as mesmas instruções e torcar o valor roteador escolhido e os endereços IP das interfaces deste roteador.

Neste ponto só resta informar aos roteadores que eles devem utilizar o protocolo de roteamento OSPF. Para isso é utilizado o assitente *QuaggaHelper* denominado *quagga*.

```
QuaggaHelper quagga;
```

É necessário informar aos roteadores quais interfaces devem utilizar o protocolo OSPF. Para isso é usado método *EnableOspf* que informa aos roteadores de um determinado enlace para usarem o OSPF na interface com o endereço IP determinado. A instrução a seguir mostra o método *EnableOspf* configurando os roterdores do enlace e_01 para usarem OSPF na interface com IP 10.0.1.0.

```
quagga.EnableOspf (e_01, "10.0.1.0/24");
```

Como cada enlace é consideredo uma rede ponto-a-ponto com IP's diferentes é preciso repetir o procedimento anterior para todos os enlaces. Cada enlace com seu endereço respectivo.

```
quagga.EnableOspf (e_02, "10.0.2.0/24");
quagga.EnableOspf (e_13, "10.1.3.0/24");
quagga.EnableOspf (e_14, "10.1.4.0/24");
quagga.EnableOspf (e_24, "10.2.4.0/24");
quagga.EnableOspf (e_35, "10.3.5.0/24");
quagga.EnableOspf (e_45, "10.4.5.0/24");
```

O *NodeContainer routers* contém todos os roteadores da rede. A seguir, *routers* é usado pelo método *EnableOspfDebug* para informar aos roteadores que as mensagens de depuração do protocolo OSPF devem ser exibidas no resultado da simulação.

```
quagga.EnableOspfDebug (routers);
```

De maneira semelhante a todos os assistentes do NS-3, o método *Install* aplica as configurações definidas em todos os roteadores.

```
quagga.Install (routers);
```

Com as últimas alterações o NS-3 está configurado para simular o protocolo OSPF do DCE. Portanto, é possível observar o novo comportamento do OSPF nas simulações que ocorrem falhas na rede. Entretanto a instrução para simular uma falha na rede deve ser modificada. A falha na rede é simulada ao destivar uma interface de algum roteador. Para isso é usado o método *RunIp* que utiliza como parâmetro um roteador, o intervalo da simulação que deve ocorrer a falha e qual interface deve ser desativada. Na instrução a seguir foi selecionada a interface sim1 do roteador 0 para ser destivada aos 62 segundos da simulação. Para desativar alguma interface dos outros roteadores basta alterar os parâmetros desta instrução.

```
RunIp (routers.Get (0), Seconds (62), "link set sim1 down");
```

Ao executar as simulações é possível observar o comportamento do OSPF implementado no módulo DCE. O funcionamento da implementação do DCE é semelhante ao da implementação do *Global Router Manager*. Entretanto, é possível notar a diferença entre as duas implementações nas simulações que ocorrem falhas na rede. Ao utilizar o *Global Router Manager*, quando ocorre um falha de enlace todos os pacotes, que utilizavam o caminho afetado, são perdidos. No OSPF do DCE somente alguns pacotes são perdidos, pois através do OSPF os roteadores percebem a falha do enlace e encontram um novo caminho mínimo. Além disso é utilizado o executável *iperf* para medir a diferença na vazão da rede em simulações com falhas. Através do uso desta ferramenta é possível perceber o seguinte. Durante as simulações sem falhas a taxa de transferência é de 4.19 *Mbits* por

segundo. Nas simulações que a falha ocorre em um enlace escolhido pelo algoritmo de Dijkstra esta taxa cai para 3.15 *Mbits* por segundo. Além disso, ao analisar os arquivos gerados pelo *pcap* é possível perceber que os roteadores demoram, em média, 300 milissegundos para encontrar um novo caminho mínimo. Outro aspecto observado foi que o protocolo OSPF não faz balanceamento de tráfego quando existe mais de um caminho mínimo para o roteador destino.

O DCE fornece uma implementação completa do OSPF. Entretanto, o DCE não é nativo do NS-3 e isso adiciona complexidade ao procedimento simular o OSPF. Primeiramente, se o NS-3 foi instalado de acordo com sua documentação, o módulo DCE não é instalado. Além disso, a maneira de configurar a simulação passa por diversas mudanças quando é utilizado o DCE. Deste modo, grande parte do conhecimento adquirido ao utilizar o NS-3, sem o DCE, se torna obsoleto. Sendo assim, é necessário consultar a documentação do DCE para compreender alguns dos procedimentos usados na configuração das novas simulações. Porém a documentação do DCE, especialmente a parte relacionada ao OSPF, é insuficiente e incompleta. Além disso, alguns procedimentos exigem conhecimento de executáveis externos ao NS-3 como, por exemplo para configurar as interfaces de rede dos roteadores é preciso conhecer o executável “ip”. Na verdade, usar o DCE implica em trabalhar com a própria implementação do protocolo no sistema operacional, o que elimina a vantagem da simulação, na medida em que múltiplos detalhes devem ser considerados mesmo para testes simples.

CAPÍTULO 5

CONCLUSÃO

A Internet é organizada em regiões chamadas sistemas autônomos. Os protocolos responsáveis pelo roteamento dentro dos sistemas autônomos são chamados de protocolos de roteamento interno. Um destes protocolos utilizado atualmente é o OSPF. Além disso, este protocolo pertence a uma classe conhecida como estado de enlace. O OSPF utiliza descrições do estado dos enlaces da rede para criar uma representação da topologia da rede na forma de um grafo. A partir dessa representação, os roteadores utilizam o algoritmo de Djikstra para calcular o caminho mínimo de uma origem para cada destino da rede. A criação e manutenção desta representação da topologia é feita por três subprotocolos que juntos formam o OSPF. O protocolo *Hello* tem o objetivo de verificar o estado dos enlaces e eleger um roteador designado e um roteador reserva. Quando dois roteadores estabelecem conectividade bidirecional, o protocolo *Exchange* é reponsável pela sincronização da representação local da topologia da rede. O protocolo *Flooding* é reponsável por informar a todos os roteadores quando ocorre uma falha na rede.

Neste trabalho descrevemos como simular o OSPF no NS-3. Inicialmente é avaliado o uso do *Global Router Manager* que é resposável por preencher as tabelas de roteamento do roteadores. Entretanto, o *Global Router Manager* não implementa um protocolo de roteamento. Uma vez que, ele somente calcula caminhos estáticos, pois as tabelas de roteamento são preenchidas antes de iniciar o tráfego de pacotes na rede. Portanto, não são utilizados pacotes de controle e quando ocorre uma falha na rede os rotedores não são informados. Sendo assim, é possível perceber que a implementação nativa do NS-3 é muito limitada. Uma vez que, quando um enlace falha, os roteadores da rede não encontram novos caminhos. Fica claro que o funcionamento dos subprotocolos *Hello* e *Flooding* não

são simulados. Portanto, um módulo do NS-3 chamado DCE passa a ser uma alternativa para simular o protocolo OSPF. O DCE fornece a implementação do protocolo OSPF feita pela *Quagga Routing Suite*. O Quagga possui a implementação dos três subprotocolos do OSPF. Portanto, os roteadores descobrem quando ocorre uma falha na rede e encontram um novo caminho mínimo.

Realizar simulações do OSPF usando a implementação feita pelo Quagga apresenta uma série de desafios. Como o DCE não é nativo do NS-3 o procedimento de simular o OSPF se torna mais complexo. A maneira de configurar a simulação passa por diversas mudanças quando é utilizado o DCE. Deste modo, grande parte do conhecimento adquirido ao utilizar o NS-3, sem o DCE, se torna obsoleto. Ao consultar a documentação do DCE para compreender alguns dos procedimentos usados na configuração das novas simulações é possível perceber que a documentação do DCE, especialmente a parte relacionada ao OSPF, é insuficiente e incompleta. Na verdade, usar o DCE implica em trabalhar com a própria implementação do protocolo no sistema operacional, o que elimina a vantagem da simulação, na medida em que múltiplos detalhes devem ser considerados mesmo para testes simples. O pesquisador, em geral, recorre a uma simulação para obter resultados de forma simples e direta, sem que seja necessário analisar os detalhes das implementações. Assim, é possível concluir que o trabalho futuro necessário, para permitir a continuidade deste trabalho, é implementar o OSPF no NS-3.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] *pcap*, http://www.tcpdump.org/pcap3_man.html, Acessado em dezembro de 2013.

[2] *Wireshark*, http://www.wireshark.org/, Acessado em dezembro de 2013.

[3] *tcpdump*, http://www.tcpdump.org/, Acessado em dezembro de 2013.

[4] *NS-3*, http://www.nsnam.org/, Acessado em novembro de 2013.

[5] *Qagga Software Routing Suite*, http://www.nongnu.org/quagga/index.html, Acessado em novembro de 2013.

[6] *Bake*, http://www.nsnam.org/docs/bake/tutorial/html/bake-over.html, Acessado em novembro de 2013.

[7] *Direct Code Execution*, http://www.nsnam.org/overview/projects/direct-code-execution/, Acessado em novembro de 2013.

[8] R. E. Bellman. *Dynamic Programming*, Princeton University Press, 1957.

[9] R. Coltun, D. Ferguson, J. T. Moy, e A. Lindem. *OSPF for IPv6*, RFC 5340, 2008.

[10] D. E. Comer. *Interligação de Redes com TCP/IP*, 5ª Edição, Editora Campus, 2006.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, e C. Stein. *Introduction to Algorithms*, 1ª Edição, MIT Press and McGraw-Hill, 1990.

[12] E. W. Dijkstra. *A Note on Two Problems in Connection with Graphs*, Numerische Mathematic 1, 1959.

[13] L. R. Ford e D. R. Fulkerson. *Flows in Network*, Princeton University Press, 1962.

[14] C. Huitema. *Routing in the Internet*, 2ª Edição, Editora Prentice Hall, 1999.

[15] J. F. Kurose e K. W. Ross. *Redes de Computadores e a Internet: Uma Nova Abordagem*, 1ª Edição, Editora Pearson Education, 2003.

[16] G. Malkin. *RIP Version 2*, RFC 2453, 1998.

[17] J. T. Moy. *OSPF: Anatomy of an Internet Routing Protocol*, 1ª Edição, Editora Addison Wesley, 1995.

[18] J. T. Moy. *OSPF Version 2*, RFC 2328, 1998.

[19] Y. Rekhter, T. Li, e S. Hares. *A Border Gateway Protocol 4 (BGP-4)*, RFC 4271, 2006.