UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE APOIO À PESQUISA
PROGRAMA INSTITUCIONAL DE BOLSA DE INICIAÇÃO CIENTÍFICA

AUTO-RECUPERAÇÃO DE FALHAS EM REDES COM AGENTNOX

Bolsista: Paulo César da Rocha Fonseca, CNPq

MANAUS
2011

UNIVERSIDADE FEDERAL DO AMAZONAS
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE APOIO À PESQUISA
PROGRAMA INSTITUCIONAL DE BOLSA DE INICIAÇÃO CIENTÍFICA

RELATÓRIO FINAL
PIB-E/0006/2010
AUTO-RECUPERAÇÃO DE FALHAS EM REDES COM AGENTNOX

Bolsista: Paulo César da Rocha Fonseca, CNPq
Orientador: Profº Drº Edjard Souza Mota

MANAUS
2011

Esta pesquisa, financiada pelo Conselho Nacional de Pesquisa – CNPq, através do Programa Institucional de Bolsas de Iniciação Científica da Universidade Federal do Amazonas.

Propõe um mecanismo de replicação usado para garantir o funcionamento da rede em caso de falhas salvando o estado do componente responsável pela conectividade entre os elementos da rede. O sistema operacional para redes a ser usado como plataforma é o NOX. O NOX é um controlador de rede do protocolo OpenFlow, o qual permite a manipulação do tráfego de pacotes para fins experimentais. A literatura apresenta várias abordagens para o problema da replicação e de manter as réplicas consistentes. Entretanto, várias dessas abordagens pressupõem controle distribuído, enquanto o NOX apenas dispões de controle centralizado. Uma das abordagens que satisfaz estas características é a de *primary-backup replication* Neste método toda vez que o servidor primário recebe uma requisição de um dos clientes ele completa a operação solicitada, muda o seu estado e envia uma mensagem de atualização de estado ao servidor secundário (ou, caso haja mais de um secundário, aos servidores). O servidor secundário, então, atualiza o seu estado e envia uma mensagem de confirmação para o primário, este, por sua vez, envia uma confirmação ao cliente que a operação foi completada. Este mecanismo foi implementado em C++ e testado em ambientes virtuais VMWare e Mininet, sendo esta última uma ferramenta desenvolvida pelo grupo criador do NOX. A abordagem *primary-backup* demonstrou-se uma boa alternativa pois foi possível implementá-la sem que fosse preciso mudar a estrutura da rede. Também não foi observada nenhum impacto negativo na performance do NOX. Outros aspectos do protocolo, como limites superiores e inferiores do mecanismo serão observados em trabalhos futuros, assim como testes em ambientes mais robustos. Essa é uma contribuição importante para comunidade uma vez que essa é uma área em franco crescimento com a adesão de cada vez mais empresas a essa tecnologia e, por ser uma área recente, com pouca pesquisa disponível e muitas questões ainda em aberto, não havendo nenhum artigo que aborde cenários de falhas em uma rede Openflow a nível de controlador.

LISTA DE FIGURAS E TABELA

SUMÁRIO

1. INTRODUÇÃO

Sistemas operacionais de redes permitem que aplicações de gerência de redes sejam implementadas sobre uma interface programática centralizada e uniforme que oferece a possibilidade de observação e controle dos dispositivos da rede em alto nível. As aplicações têm acesso há uma visão geral de rede, incluindo topologia, conexões e localização de usuários, enquanto o sistema operacional se encarrega de obter informações de mais baixo nível e aplicar regras nos dispositivos através de um padrão de acesso.

A versão mais completa de um sistema operacional de redes é o NOX [1]. O NOX oferece um núcleo que provê a interface programática para uma camada onde as aplicações de gerência são executadas. Esse núcleo, dentre outras coisas, utiliza o protocolo de acesso OpenFlow [2] para obtenção das informações dos switches e roteadores, bem como a implantação das ações solicitadas pelas aplicações de gerência.

A arquitetura AgentNOX é uma extensão do NOX que prevê que a camada de aplicação seja composta de agentes autônomos proativos, reativos e com habilidades sociais de comunicação, cooperação e negociação. As propriedades desses agentes fazem com que os sistemas operacionais de redes possam auto-gerenciar o comportamento da rede através de características como auto-configuração, auto-recuperação, auto-otimização, auto-proteção [3].

Redes com AgentNOX herdam a propriedade de centralização do NOX, ou seja, um único controlador na rede é responsável pelo registro, autenticação e controle de roteadores, switches e usuários. Por causa disso, em caso de falha do sistema operacional, os agentes de auto-gerenciamento da rede podem ficar impossibilitados de atuar e a operação da rede pode ficar seriamente comprometida. Mecanismos de resiliência devem ser implementados para garantir que em caso de falhas no controlador central, outro controlador possa assumir a partir do último estado válido [1], [4].

O problema é que em um AgentNOX, não apenas o último estado válido do núcleo do sistema operacional de redes deve ser preservado por outro controlador em caso de falhas, mas também os estados de cada agente da camada de aplicação. A manutenção dos estados dos agentes também pode envolver a necessidade de renegociação com outros agentes em outros domínios.

O objetivo desse trabalho é implementar um mecanismo de auto-recuperação que mantenha o estado dos componentes responsáveis pelo funcionamento da rede de maneira que os hospedeiros da rede não percam a conectividade entre si.

2. REVISÃO BIBLIOGRÁFICA

Em março de 2008 o OpenFlow foi apresentado no artigo “OpenFlow: Enabling Innovation in Campus Networks” [2]. OpenFlow é uma maneira de testar protocolos experimentais em redes reais sem interferir com o tráfego de produção, ele é uma característica que pode ser adicionada a switches ethernet que disponham de tal funcionalidade e trabalha à nível de tabela de fluxo, removendo e adicionando novos registros de fluxos nela. Os registros contém o padrão do cabeçalho do pacote pertencente ao fluxo, contadores (para fins estatísticos) e a ação associada ao fluxo. A tabela de fluxo é administrada por um controlador remoto que decide qual ação associar a cada novo fluxo e está conectado ao switch através de um canal seguro (criptografado por SSL). Atualmente um dos principais objetivos do grupo responsável pelo OpenFlow é aumentar a quantidade de empresas que comercializam switches OpenFlow e universidades que o implementam em seu campus. Recentemente foi lançada a versão 1.0 do OpenFlow, sendo a primeira que está oficialmente pronta para ser implementada em switches reais.

Paralelo ao desenvolvimento do OpenFlow ocorre o do NOX, um sistema operacional para redes centralizado baseado no OpenFlow e introduzido pelo artigo “NOX: Towards an Operating System for Networks” [1]. O NOX age como um controlador do OpenFlow, adicionando e removendo registros de fluxo da tabela de fluxo, para exercer controle sobre a rede e seus recursos. Ele provê um plataforma simplificada para escrever componentes que irão gerenciar os recursos da rede. O NOX pode ser dividido em dois aspectos: Os componentes externos de alto-nível que lidam com a API e são feitos majoritariamente em Python, também podem ser escritos em C++ ou ambos, e o núcleo do sistema operacional que lida com tarefas de baixo nível e a sua maior parte é escrita em C++, para conservar a performance, com alguns trechos de código em Python.

Outra área relacionada a este trabalho é a computação autonômica, a qual foi apresentada pelo artigo “The Vision of Autonomic Computing” [5]. que expôs a computação autonômica, seus conceitos e objetivos, introduzidos pela primeira vez em 2001 por Paul Horn, na época vice-presidente da IBM. No artigo eles definem as características necessárias para um sistema ser autonômico: auto-configuração, auto-otimização, auto-recuperação e auto-proteção.

O artigo [1] propõe que para aumentar a confiabilidade da rede seja mantida uma conexão com um controlador secundário que irá tomar controle da rede caso o primário venha a falhar. A literatura apresenta várias abordagens para o problema da replicação e de manter as réplicas consistentes (como pode ser visto em [9]). Entretanto, várias dessas abordagens pressupõem controle distribuído (por exemplo, [11]), enquanto o NOX apenas dispões de controle centralizado. Uma das abordagens que satisfaz estas características é

de *primary-backup replication* (apresentada em [10]). Neste método toda vez que o servidor primário recebe uma requisição de um dos clientes ele completa a operação solicitada, muda o seu estado e envia uma mensagem de atualização de estado ao servidor secundário (ou, caso haja mais de um secundário, aos servidores). O servidor secundário, então, atualiza o seu estado e envia uma mensagem de confirmação para o primário, este, por sua vez, envia uma confirmação ao cliente que a operação foi completada. Paralelamente a isso o secundário envia constantemente mensagens de controle para saber se o primário continua ativo, caso o primário entre em estado de falha e não envie a confirmação (mensagem ACK), o secundário assume o controle da rede.

A abordagem possui outras propriedades e métricas, que serão tratadas mais detalhadamente nas próximas seções.

3. MÉTODOS UTILIZADOS

Inicialmente, foi realizado um estudo sobre o estado da arte do problema e uma pesquisa para verificar se existe alguma pesquisa em andamento ou algum material disponível sobre o auto-recuperação de sistemas operacionais de redes.

Foram realizados estudos da linguagem C++(com foco para parte de *template*, uma vez que se constatou que este conceito é utilizado em pontos cruciais da implementação do núcleo do NOX) e da API em C++ do NOX. A mudança de Python(usado no último ano do projeto) para C++ ocorreu devido a necessidade crescente de eficiência, uma vez que a complexidade do componentes(e futuramente, agentes) e das comunicações entre eles tende a aumentar. A linguagem Python apesar de possuir uma API do NOX mais bem documentada e uma facilidade maior de escrita possui problemas de lentidão e eficiência,ao contrário do C++, porém dispomos de menos documentação.

Também utiliza-se uma ferramenta chamada Mininet, ela provê um ambiente virtual para testes e incluem quase todas as funcionalidades e características presentes em uma rede OpenFlow real, ideal uma vez que o grupo de pesquisa ao qual este projeto está associado ainda não possui laboratório e o Mininet pode ser usado em qualquer computador pessoal. O Mininet foi desenvolvido pelo próprio grupo responsável pela criação e manutenção dos projetos NOX e OpenFlow. Já foi atestado que apesar de ser uma máquina virtual o Mininet é capaz de emular ambientes reais com fidelidade e robustez, chegando a emular ambientes com milhares de hosts.

Dentro deste ambiente foi desenvolvido um componente de recuperação baseado no método de replicação *primary-backup.* Este componente utilizava o componente *messenger* do NOX para realizar a comunicação entre o servidor primário e o secundário. Através dele são enviadas as mensagens de atualização e requisição. Primeiramente, o NOX primário recebe um evento de chegada de pacote, se o pacote for de um endereço MAC não pertencente a sua tabela, ele o insere na sua tabela e envia uma mensagem de atualização de estado para o NOX secundário com as informações necessárias para que ele insira o MAC em sua tabela, mantendo-se assim consistente com a tabela do NOX primário.

Para realizar os testes utilizou-se duas máquinas virtuais VMWare, em uma executava-se o Mininet, supracitado, controlado pelo NOX primário e na outra executava-se o NOX secundário, ambas as máquinas executando o componente de recuperação que as mantêm consistentes entre si.

4. RESULTADOS E DISCUSSÕES

A primeira atividade na qual estivemos envolvidos foi a tradução de um componente de detecção de ataques distribuídos de negação de serviço em uma rede com o NOX, componente apresentado em [7]. Este componente que foi escrito na linguagem Python foi traduzido para C++ (além motivos supracitados, o NOX também possui a característica de que todo o seu núcleo é implementado em C++, enquanto as suas aplicações, que são executadas sobre esse núcleo, são escritas em Python. Logo, como desejamos fazer uma extensão do núcleo do NOX, escolhemos implementar em C++). Na Figura 1 é apresentada uma captura de tela do componente o qual foi feita a tradução em funcionamento, no terminal à esquerda está o Mininet em execução, responsável por virtualizar a rede, e à direita está o componente de detecção indicando que está ocorrendo um ataque.

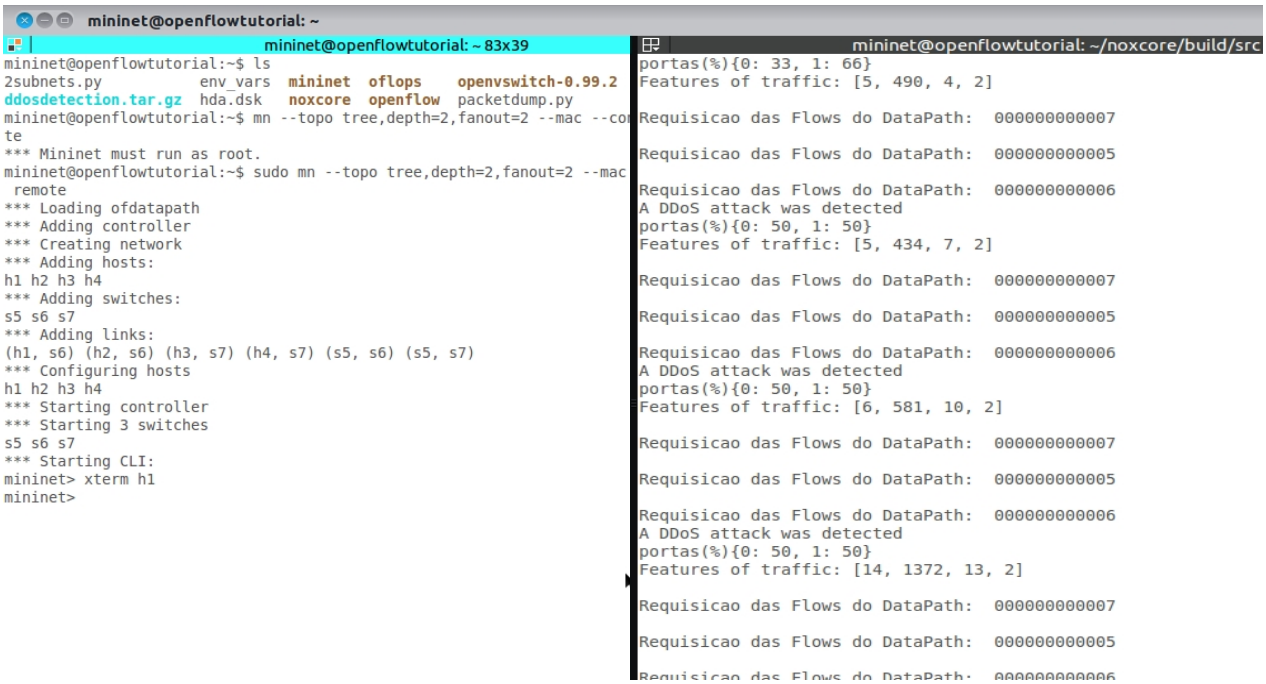

Figura 1 – Componente de detecção de ataque DDoS

Portar este componente consistiu em duas etapas: Implementar em C++ o código do SOM (método de classificação através de um mapa auto-organizável usado pelo componente para identificar um ataque) e traduzir todos os métodos do componente para a API do C++.

A implementação do SOM consistiu, basicamente, na tradução das funções matemáticas em Python para o seu equivalente em C++. A parte da tradução do componente NOX em si e seus devidos métodos pertencentes à API apresentou um grau de dificuldade mais elevado e levou mais tempo para ser concluída uma vez que as modificações também foram feitas no núcleo do NOX, uma vez que, como foi esclarecido

pelo grupo que mantém o código do NOX atualmente, o código original não permitia um uso direto por um componente escrito em C++, apenas pelos implementados em Python. Este componente serviu como base para o estudo dos possíveis cenários de falha (a figura 2 representa a arquitetura de uma rede NOX), mais especificamente no cenário de um ataque distribuído de negação de serviço.

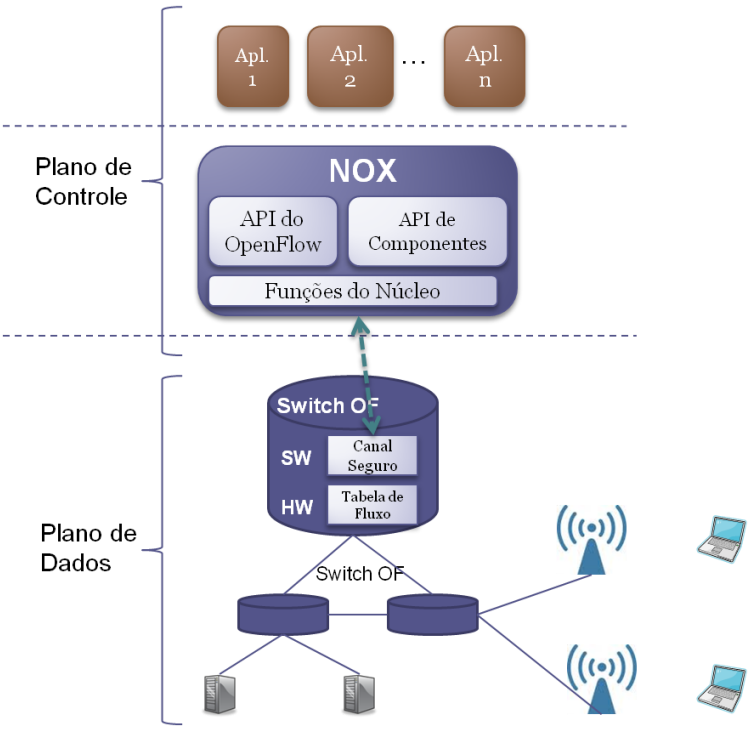

Figura 2 – Arquitetura de uma rede Openflow

Outro cenário de falha, é o de má configuração da rede. Esse caso ocorrerá quando a implementação do protocolo não está de acordo com a suas respectiva especificação [8] ou a rede não está configurada adequadamente. Outro cenário de erro conhecido é o acesso a região crítica. Quando dois processos diferentes tentam acessar o mesmo recurso (área da memória, registrador, etc.) simultaneamente o NOX entrará em estado de erro, uma vez que não há tratamento explícito para tal. O processo do NOX será simplesmente abortado e nenhum contexto será salvo, portanto, não se armazena nenhuma informação para uma possível recuperação do erro.

No artigo de apresentação do NOX [1] sugere-se que para aumentar a confiabilidade de uma solução centralizada devem haver um ou mais servidores secundários que assumam o controle da rede caso o principal entre em estado de falha. Como foi tratado anteriormente, a abordagem que mais adequada a esse cenário é a de replicação *primary-backup.* Esta abordagem define que qualquer protocolo desenvolvido baseado nela deve manter as seguintes propriedades:

i. Existe um predicado local $Primary_s$ no estado de cada servidor s. Em qualquer momento, existe no máximo um servidor s que satisfaz $Primary_s$

ii. Cada cliente i mantém um $dest_i$. Se o cliente desejar fazer uma requisição, ele envia uma mensagem para $dest_i$

iii. Se o servidor s receber um requisição e ele não for o primário ($Primary_s$ não satisfeito) ele não enfileira a requisição

iv. Existem valores k e Δ tais que o serviço comporte-se como um único servidor (k, Δ)-bofo (*bounded outages*, *finitely often*)

A propriedade iv garante que o comportamento de uma rede que utilize o protocolo *primary-backup* seja indistinguível de uma rede com um servidor sujeito a períodos de falhas, onde esses períodos possam ser agrupados em k intervalos de tempo, onde cada intervalo tem duração máxima Δ.

Também se deve observar as métricas de custo associadas a um protocolo *primary-backup*, sendo estas:

i. Grau de replicação: O número de servidores secundários utilizados.

ii. Tempo de bloqueio: O maior tempo possível de espera entre uma requisição e a sua resposta.

Em relação ao grau de replicação foi definido que haverá um servidor secundário, executando uma instância do NOX. O tempo de bloqueio está na casa dos milissegundos como foi definido em [1].

O NOX possui 3 tipos de classes de componentes: *coreapps,* responsáveis pelas funções mais básicas e internas do NOX; *netapps,* gerenciam os aspectos internos da rede; e os *webapps,* realizam os serviços relacionados ao acesso a internet. Para o escopo definido deste projeto a classe de componentes a ser observada foi a *coreapps.* Dentro desta classe o componente *switch* foi identificado como responsável pela comunicação entre os *hosts* e tratamento de fluxos e pacotes.

Para implementar o protocolo foi criado um componente *primarybackup* e adicionado na aos *coreapps.* Este componente possui os métodos responsáveis pela comunicação inter-NOX, processando as mensagens dos componentes e as enviando através da rede. Este componente é chamado dentro do *switch*, tendo sido declarado como um atributo deste.

Dentro do componente *switch* foi feita uma identificação de quais partes do código era necessário isolar e quais estruturas de dados precisavam ser mantidas atualizadas. Foi identificado que o *switch* precisa do *datapath id* (identificador do *switch* na rede), do endereço MAC e do endereço IP para manter uma tabela que relaciona essas 3 informações para identificar cada *host* da rede.

O *switch* passa essas informações para o *primary-backup*, este último, por sua vez, usa o componente *messenger* para enviar a mensagem para o NOX secundário, antes de a

mensagem ser enviada ela é formatada e as informações são nela encapsuladas de maneira que obedeça o padrão de mensagem do NOX, fazendo as conversões necessárias para *network order* e as atribuições de tipo e tamanho da mensagem.

O NOX secundário irá receber esta mensagem na porta 2603 (porta reservada pelo NOX para receber mensagens de componentes externos), o componente *switch* irá reconhecer o evento gerado pela mensagem e realizar o *parse* dela, identificando qual parte da *string* recebida é o MAC e qual parte é o *datapath id.* Uma vez feito o parse, ele converterá cada um para o seu tipo específico e os passará como parâmetro para o construtor instanciar seus objetos e inseri-los na sua tabela.

Durante a implementação também foram observadas as propriedades do protocolo. A propriedades i e ii são garantidas pelo NOX, uma vez que apenas um servidor NOX é conhecido pelos *switchs* e *hosts*, e as requisições são enviadas direta e unicamente a ele. Para garantir a terceira propriedade isolou-se no código do componente *switch* as partes que processam as requisições do cliente, criou-se um atributo booleano *isPrimary*(análogo ao predicado local $Primary_s$) e o switch apenas processará as requisições se este atributo for *TRUE*. A quarta propriedade também será satisfeita, pois essa transição entre o servidor principal e o backup será transparente para os switchs e hosts.

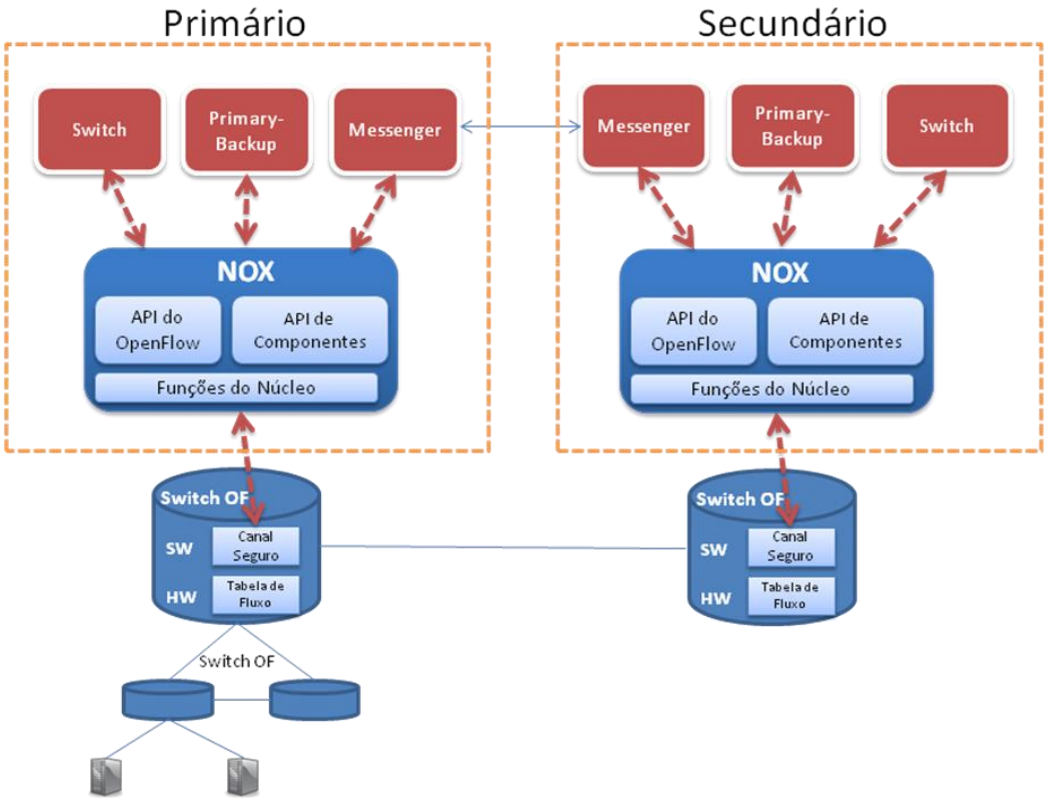

Figura 3 – Arquitetura da rede com *Primary-Backup*

A figura 3 representa a arquitetura e topologia da rede, com os componentes sendo executados sobre o NOX realizando comunicação bidirecional com o mesmo e os

componentes *messenger* de cada NOX responsável pela troca de mensagens. Essas mensagens trafegam pela rede, passando pelos *switchs*, mas em termos de alto-nível ela ocorre entre os *messengers.*

Na figura 4 temos uma captura de tela do componente em funcionamento:

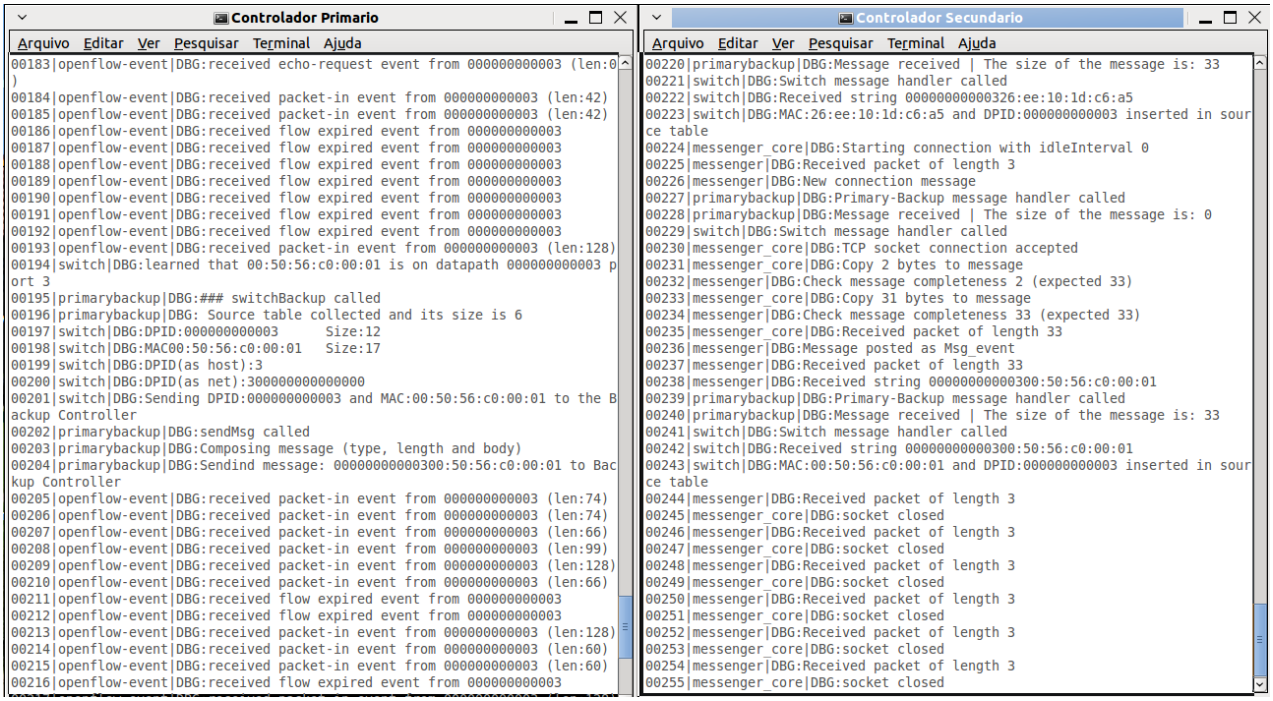

Figura 4 – Captura de tela do componente em funcionamento

O terminal esquerdo representa o controlador primário responsável pela conectividade dos *hosts* e envio de mensagens de atualização para o controlador secundário, pelo *log* é possível observar o encapsulamento do MAC e do *datapath id*. O terminal à direita é o controlador *backup* que recebe as mensagens e atualiza o seu estado para manter a consistência com o primário, é possível o observar o recebimento da mensagem e o *parse* pelo seu *log*.

5. CONCLUSÃO

Este trabalho resultou na implementação de um mecanismo de replicação usado para garantir o funcionamento da rede em caso de falhas salvando o estado do componente responsável pela conectividade entre os elementos da rede. A abordagem escolhida foi a do *Primary-Backup* uma vez que esta demonstrou ser a mais adequada para o problema de falhas em redes NOX. Foram mantidas as principais propriedades da abordagem e também foram observadas suas métricas.

Verificou-se que o protocolo *Primary-Backup* é uma boa abordagem para aumentar a confiabilidade do NOX, uma vez que a arquitetura de uma rede Openflow controlada pelo NOX nas suas especificações sem mudanças drásticas na sua estrutura (outros protocolos, definidos em [11] seria necessário fazer com que os *switchs* fossem configurados para enviar requisições para mais de um controlador simultaneamente e deveria ser implementada um política de negociação entre os controladores para ser decidido qual responderá).

Apesar da constante troca de mensagem entre os dois controladores não foi observada nenhuma perda de performance em nenhum dos controladores. No futuro, serão feitos testes mais robustos para medir a performance em situações mais críticas.

A abordagem também define os limites inferiores e superiores de desempenho em diferentes cenários, este estudo será feito em trabalhos futuros uma vez que demandam de uma análise mais demorada e complexa do problema, não pertencendo ao escopo de um trabalho de PIBIC.

Essa é uma contribuição importante para comunidade uma vez que essa é uma área em expansão, com cada vez mais empresas aderindo a essa tecnologia e com pouca pesquisa disponível e muitas questões em aberto, não havendo nenhum artigo que aborde cenários de falhas em uma rede Openflow a nível de controlador.

6. REFERÊNCIAS BIBLIOGRÁFICAS

[1] N.Gude, T. Koponen, J. Pettit, B. Ptaff, M. Casado, N. McKeown, and S. Shenker, “Nox: Toward an operating system for networks”. ACM SIGCOMM Computer Communications Review: 2008.

[2] N. Mckeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling Innovation in Campus Network”. ACM SIGCOM Computer Communications: 2008.

[3] B. Jennings, S. van Der Merr, S. Balasubramaniam, D. Botvich, M. Ó Foghlú, W. Donnelly, and J. Strassner, "Towards Autonomic Management of Communications Networks," IEEE Communications Magazine, 2007, pp. 112-121.

[4] M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," ACM SIGCOMM, ACM, 2007.

[5] J. Kephart, D. M. Chess. 2003. The vision of autonomic computing. IEEE Comput. 36, 1, (Jan.), 41–50

[6] P. Cong-Vinh, Autonomic Computing and Networking, Boston, MA: Springer US, 2009.

[7] R. S. Braga; E. Mota; A. Passito. Lightweight Ddos Flooding Attack Detection Using Nox/Openflow, In: 35th Annual Ieee Conference On Local Computer Networks, 2010, Denver, Colorado - Usa.

[8] Especificação técnica do protocolo OpenFlow. Disponível em: http://www.openflowswitch.org/documents/openflow-spec-v1.0.0.pdf

[9] R. Guerraoui, A. Schiper. Software-Based Replication for Fault Tolerance, Computer, v.30 n.4, p.68-74, April 1997

[10] N. Budhiraja et al., “The Primary-Backup Approach,” in Distributed Systems, S. Mullender, ed., ACM Press, New York, 1993, pp. 199-216

[11] F.B. Schneider, “Replication Management Using the State-Machine Approach,” in Distributed Systems, S. Mullender, ed., ACM Press, New York, 1993, pp. 169-197

Nº	Descrição	Ago 2010	Set	Out	Nov	Dez	Jan 2011	Fev	Mar	Abr	Mai	Jun	Jul
1	Realizar um estudo das classes de problemas de redes que podem comprometer o desempenho em arquiteturas centralizadas de sistemas operacionais de redes	X	X										
2	Realizar um estudo das classes de problemas que podem comprometer a camada de aplicação formada por agentes		X	X									
3	Investigar mecanismos de auto-recuperação que podem ser executados em caso de ocorrência das falhas estudadas			X	X	X	X						
4	Projetar e programar um mecanismo para a arquitetura AgentNOX					X	X	X	X				
5	Testar o funcionamento e o desempenho do componente.									X	X	X	
	- - Elaboração do Resumo e Relatório Final (atividade obrigatória) - Preparação da Apresentação Final para o Congresso (atividade obrigatória)												X