Um Protocolo de Eleição Tolerante a Falhas e Instabilidades para Controladores SDN

Nelson G. Prates Jr.[1], Michele Nogueira[2], Ricardo T. Macedo[1]

[1]Universidade Federal de Santa Maria - Campus Frederico Westphalen, RS
Departamento de Tecnologia da Informação- DeTecInf

[2]Universidade Federal do Paraná
Departamento de Ciências Exatas

juniorpratesg@mail.ufsm.br, michele.nogueira@ufpr.br,
rmacedo@inf.ufsm.br

Abstract. *Software Defined Networks provide an overview of the network with greater management flexibility. However, it ends up making the controller a single point of failure and also susceptible to Distributed Denial of Service (DDoS) attacks, which can compromise communication between network devices. In the literature, there are efforts to improve fault tolerance caused by denial-of-service attacks on SDNs and also to improve the election process. However the form they are implemented consumes a large amount of network resource consumption. This work presents a Fault Tolerant and Instability Protocol for SDN Controllers with the objective of maintaining the stability of the network even with faulty controllers. This proposal consumes a low number of messages exchanges both to elect and to reestablish leadership structures. The protocol was developed and evaluated using the POX Controller, OpenFlow Switch, Mininet and the Cbench. The results obtained show the efficacy of the protocol.*

Resumo. *As Redes Definidas por Software propõem um controlador logicamente centralizado, possibilitando uma visão global da rede. Porém, acaba tornando o controlador um ponto único de falhas e também suscetível a sofrer ataques Distribuídos de Negação de Serviço (do inglês, Distributed-Denial of Service - DDoS), podendo comprometer a comunicação entre os dispositivos de rede. Na literatura existem esforços para aprimorar a tolerância a falhas causadas por ataques de negação de serviço em SDNs e também para melhorar o processo de eleição. No entanto a forma que são implementadas consomem uma grande quantidade de recursos da rede. Este trabalho apresenta um protocolo de eleição de líder tolerante a falhas e instabilidades para controladores SDN com o objetivo de manter a estabilidade da rede mesmo com controladores falhos. Esta proposta consome um número baixo de trocas de mensagens tanto para eleger, quanto para reestabelecer as estruturas de lideranças. O protocolo foi desenvolvido e avaliado utilizando POX Controller, OpenFlow Switch, Mininet e o Cbench. Os resultados obtidos mostram a eficácia do protocolo.*

1. Introdução

A SDN oferece uma potencial vantagem de segurança com sua abordagem centralizada que traz uma visão global, facilitando a realização da análise e gerência da rede para a

detecção de defeitos. A principal proposta das SDNs, é o desacoplamento do plano de controle do plano de dados. O plano de controle é direcionado para um controlador logicamente centralizado e o plano de dados fica a cargo dos dispositivos de encaminhamento (*switches*). Assim, os *switches* quando recebem uma nova demanda, solicitam ao controlador novas regras de encaminhamento. Esta estrutura oferece uma camada de abstração que permite a programação de aplicações em alto nível para controlar as regras de tráfego nos dispositivos. Com isso, facilitando a implementação de políticas de segurança, balanceamento de carga, tráfego de filtro, etc [Hu et al. 2015]. Embora a ideia principal das SDNs ser um único controlador logicamente centralizado, este novo paradigma permite também múltiplos controladores em diferentes domínios da rede. Essa abordagem distribuída possibilita melhorias de escalabilidade, tolerância a falhas e balanceamento de carga [Schmid and Suomela 2013].

No entanto, o paradigma SDN com múltiplos controladores apresentam desafios, como a tolerância a controladores falhos ou instáveis. Como os controladores são divididos em diferentes domínios. Então cada controlador fica responsável por todas as instalações de regras de tráfego de cada domínio, habilitando pontos falha na estrutura SDN. Assim, se por algum motivo o controlador falhar ou se caracterizar instável, os *switches* subordinados a ele não vão saber lidar com novas demandas de fluxo. Isto torna os controladores alvos interessantes para ataques Distribuídos de Negação de Serviço (do inglês, *Distributed Denial of Service* - DDoS). Os ataques DDoS são baseados em uma rede de computadores zumbis, que através de comandos de um usuário mal-intencionado, geram carga em um único componente da rede, causando falha e/ou instabilidade, impossibilitando o acesso de usuários/dispositivos legítimos a ele [Dao et al. 2015]. Com isso, manter uma SDN, com o controlador distribuído fisicamente, consistente, tolerante a falhas e instabilidades é a principal motivação que leva a utilização de um protocolo de eleição. Neste contexto, a eleição de um líder, tem como objetivo eleger um controlador capaz de tomar decisões para manter o correto funcionamento da rede, mesmo com um dispositivo em estado de falha ou instabilidade. No entanto, o processo de eleição geralmente demanda uma quantidade significativa de mensagens na rede e também é suscetível a falhas e instabilidades, prejudicando a sua eficiência.

Na literatura existem esforços para aprimorar a tolerância a falhas causadas por ataques de negação de serviço em SDNs e também para melhorar o processo de eleição. Dentre os trabalhos que propõem melhorias no tratamento de falhas causadas por ataques e negação de serviço, [Sattar et al. 2016] propõe a colaboração entre os controladores, [Belyaev and Gaivoronski 2014] distribui os fluxos entre as diversas rotas e [Macedo et al. 2016] utiliza um modelo de detecção de falha baseado em protocolos *gossyp* e realizam uma eleição entre os controladores. Estas abordagens muitas vezes acabam consumindo uma carga considerável de troca de mensagens, podendo comprometer ainda mais a estrutura diante de situação de falha. Para uma melhor eficiência dos protocolos de eleição [Vasudevan et al. 2003], propõe um protocolo de eleição baseado em saltos, [Macedo et al. 2016], elege além do líder, um vice-líder e um grupo de elite ou [Zhang et al. 2009] que estabelece uma organização hierárquica entre os nós e [Lee et al. 2007] que organizou cada nó com uma lista de cinco líderes em ordem crescente de peso. No entanto, além da maioria não considerar uma SDN, [Lee et al. 2007] e [Vasudevan et al. 2003] não oferecem tolerância a falhas e [Zhang et al. 2009] e [Macedo et al. 2016] para realizarem a eleição dependem da can-

didatura dos nós, necessitando de uma grande quantidade de mensagens podendo comprometer a banda disponível na rede.

Este trabalho apresenta um protocolo de eleição que tolera falhas e instabilidades em um ambiente SDN com múltiplos controladores. A principal ideia deste protocolo é manter o correto funcionamento da rede, mesmo com qualquer controlador disposto sob situação de falha ou instabilidade. Para isso definimos um padrão de comunicação através de mensagens, que possibilita eleger um líder, um vice-líder e o grupo de elite. O líder assim que eleito, realiza rotinas de monitoramento para detectar um controlador comprometido. Visto que o líder também é suscetível a falhas e instabilidades, o vice-líder é eleito para agilizar e reduzir quantidades de mensagens na rede ao eleger um novo líder. Então, assim que o líder se caracterizar falho ou instável, o vice-líder envia uma mensagem de verificação para todos os controladores e se mais de um detectou a falha do líder, ele assume a posição de líder e realiza uma nova eleição entre os componentes do grupo de elite. Para o desenvolvimento e teste utilizamos *POX Controller* como sistema para o controlador, *switches Openflow* virtualizados pelo *MiniNet* que é um emulador de rede para experimentos com o protocolo *OpenFlow*.

Uma avaliação foi executada através de diferentes cenários. O primeiro cenário avaliou o protocolo através de instabilidades geradas pela desconexão do líder. O segundo cenário gerou carga através de ataques de negação de serviço em conjunto com uma ferramenta de *benchmarking*. Os resultados obtidos revelam que o protocolo consegue reestabelecer os grupos de liderança, utilizando poucas mensagens mesmo sob situação de falha ou instabilidade.

Este trabalho é organizado da seguinte maneira. A Seção 2 apresenta o referencial teórico. A Seção 3 descreve os trabalhos relacionados. A Seção 4 detalha a solução proposta. A Seção 5 apresenta os resultados. A Seção 6 descreve as conclusões.

2. Referencial Teórico

Esta seção apresenta um breve referencial teórico sobre as áreas envolvidas no desenvolvimento do TGSI. A seção 2.1 apresenta as SDNs e sua importância; a subseção 2.2 introduz os principais conceitos sobre o protocolo *OpenFlow* a seção 2.3 detalha como um Ataque DDoS contra o controlador funciona no ambiente SDN.

2.1. Redes Definidas por Software

As Redes Definidas por Software (do inglês: *Software-Defined Network* - SDN, **Figura 1**) abrem novas perspectivas em termos de abstrações, ambientes de controle e aplicações de rede que podem ser desenvolvidas de forma simples e livre das limitações das tecnologias de rede atuais [Guedes et al. 2012]. A arquitetura de uma SDN é dividida em Plano de Dados, Plano de Controle, Plano de Gerenciamento. O Plano de Dados é composto pelos dispositivos de encaminhamento, eles são conectados ao controlador pelo Canal de Controle e entre si formando o Canal de Dados. O Plano de Controle é composto por um dispositivo chamado controlador SDN, ele executa um *software* que oferece uma camada de abstração entre os Planos de Gerenciamento e Dados. O objetivo do software do Controlador SDN é semelhante à de um sistema operacional, ele oferece uma abstração que relacionam as duas fronteiras da arquitetura (Southbound e Northbound) possibilitando a programação dos dispositivos de encaminhamento. O Plano de Gerenciamento

executa as aplicações, em alto nível, que controlam atividades como roteamento, segurança, *firewalls*, etc. Toda essa estrutura é subdividida em duas fronteiras de abstração, *Southbound* e *Northbound*. O *Southbound* é responsável por definir uma interface de programação (*Application Programming Interface* ou API) para comunicar o plano de dados com o plano de controle e também define o conjunto de instruções internas de baixo nível nos *switches*. O *Northbound* fica a cargo do Controlador SDN, ele define uma API que abstrai as instruções de baixo nível do *Southbound*.

A estrutura SDN é composta por quatro principais pilares [Kreutz et al. 2015]. i) Os planos de controle e de dados são desacoplados. O plano de controle, que no modelo tradicional de redes também está presente nos dispositivos de rede, é desacoplado, tornando os dispositivos como os *switches*, roteadores e *middle boxes* meros encaminhadores de dados. ii) As decisões de encaminhamento são baseadas em fluxo (conjunto de pacotes em transporte) e não no destino. Todos os pacotes de um fluxo recebem a mesma política. Esta abordagem permite a união do comportamento de diferentes dispositivos de encaminhamento (*switches*, roteadores, firewalls, etc). iii) A lógica de controle é movida para uma entidade externa, o chamado controlador SDN. O controlador é uma plataforma de software que é executada em um servidor e fornece os recursos essenciais incluindo abstrações para facilitar a programação de dispositivos de encaminhamento com base em uma visão da rede logicamente centralizada e abstrata. iv) A rede é programável através de aplicações de software em execução no controlador que interage com os dispositivos de plano de dados. Isto permite o desenvolvimento de aplicações em alto nível para controle e segurança da rede.

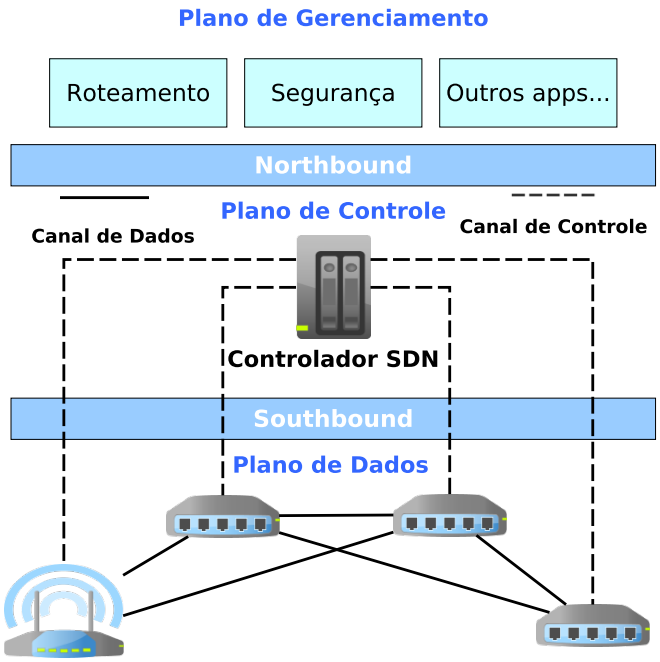

Figura 1. Exemplo de arquitetura de uma SDN.

O controlador fica logicamente posicionado ao centro da estrutura, relacionando todos os componentes da SDN. Então, ele tem um conhecimento global, tanto das aplicações, como dos dispositivos da rede. Um controlador com conhecimento global do estado

da rede simplifica o desenvolvimento de funções, serviços e aplicativos de rede mais sofisticados. Com isso, se torna mais simples e menos propenso a erros modificar políticas de rede através de linguagens de alto nível e componentes de software. Um programa de controle pode reagir automaticamente a mudanças espúrias do estado da rede e assim manter as políticas de alto nível intactas [Kreutz et al. 2015].

2.1.1. Protocolo *openflow*

O protocolo *openflow* é um dos grandes responsáveis pela popularização das Redes Definidas por Software. O protocolo simplifica a abstração do encaminhamento de pacotes da rede através da padronização dos dispositivos de encaminhamento (*switches* OpenFlow) e a definição de fluxo. O OpenFlow possibilita que os pesquisadores testem novos protocolos de roteamento, modelos de segurança, esquemas de endereçamento e até mesmo alternativas ao IP [McKeown et al. 2008]. O protocolo ficou muito popular entre os pesquisadores, pois facilitou a implementação e teste de redes de computadores, sem comprometer a rede que está em uso, também sem demandar de muitos recursos físicos e financeiros [Guedes et al. 2012].

Então, o protocolo *OpenFlow* permite a manipulação das regras de encaminhamento através do controlador. Os *switches OpenFlow* mantém uma ou mais tabelas de regras de fluxo. Estas tabelas definem as regras de como os pacotes de determinado fluxo de dados vão ser encaminhados. O controlador, como citado anteriormente, toma todas as decisões de encaminhamento de tráfego. Para isso, ele traduz as políticas de rede programadas em alto nível para regras de fluxo de baixo nível. Logo, estabelece uma comunicação segura com os *switches OpenFlow*, o que permite o controlador de realizar operações como inserir, deletar e modificar as tabelas de fluxo de cada dispositivo [Kim and Feamster 2013]. Esta arquitetura pode ser observada na Figura 2

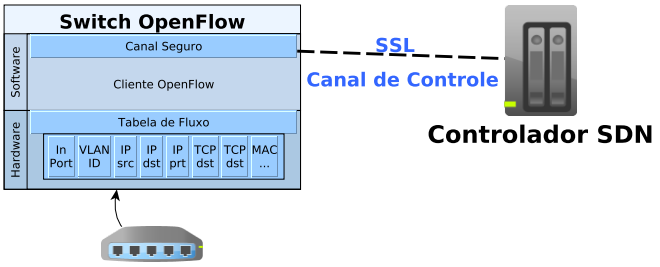

Figura 2. Exemplo de arquitetura do protocolo *OpenFlow*.

Principais Causas que Levam um Controlador SDN ao Estado de Falha ou Instabilidade. Uma SDN depende do bom funcionamento do controlador para o provimento das informações necessárias para o encaminhamento de dados. Para a garantia deste serviço, a estrutura depende de monitoramento, manutenção e segurança. Então, entender as principais motivações que leva um controlador ao estado de falha ou instabilidade é importante para o desenvolvimento de aplicações que realizam este tipo de serviço.

Um dos grandes problemas do modelo SDN, são os Ataques Distribuídos de Negação de Serviço (do inglês, *Distributed-Denial of Service* - DDoS) direcionados ao controlador, pois uma vez que o mesmo estiver inapto toda a rede é comprometida. Um

ataque DDoS é baseado em uma rede de computadores zumbis (*botnet*), onde estes, através de comandos de um usuário mal-intencionado, acessam um componente da rede causando travamento e lentidão impossibilitando o acesso dos usuários/dispositivos legítimos [Dao et al. 2015]. Como o controlador é responsável pelo principal serviço para controle da rede, ele acabe se tornando um alvo interessante. Para atacar um controlador SDN, o usuário mal-intencionado precisa direcionar os acessos ao controlador. Para isso, ele realiza diversas solicitações de novas entradas de fluxo a os dispositivos de encaminhamento. Assim, o controlador vai ficar superlotado de solicitações, esgotando seus recursos de processamento, causando travamento, lentidão e consequentemente negando o serviço de instalações de regras de fluxo aos componentes da rede.

3. Trabalhos Relacionados

Na literatura existem esforços para aprimorar a tolerância a falhas causadas por ataques de negação de serviço em SDNs e também para melhorar o processo de eleição. Dentre os trabalhos que propõem melhorias no tratamento de falhas causadas por ataques e negação de serviço, [Sattar et al. 2016] propõe a colaboração entre os controladores, [Belyaev and Gaivoronski 2014] distribui os fluxos entre as diversas rotas e [Macedo et al. 2016] utiliza um modelo para detecção de falha para disparar uma eleição entre os controladores. Estas abordagens muitas vezes acabam consumindo uma carga considerável de troca de mensagens, podendo comprometer ainda mais a estrutura diante de situação de falha. Para uma melhor eficiência dos protocolos de eleição [Vasudevan et al. 2003], propõe um protocolo de eleição baseado em saltos, [Macedo et al. 2016], elege além do líder, um vice-líder e um grupo de elite ou [Zhang et al. 2009] que estabelece uma organização hierárquica entre os nós e [Lee et al. 2007] que organizou cada nó com uma lista de cinco líderes em ordem crescente de peso. No entanto, além da maioria não considerar uma SDN, [Lee et al. 2007] e [Vasudevan et al. 2003] não oferecem tolerância a falhas e [Zhang et al. 2009] e [Macedo et al. 2016] para realizarem a eleição dependem da candidatura dos nós, necessitando de uma grande quantidade de mensagens podendo comprometer a banda disponível na rede.

Para prover tolerância diante de uma sobrecarga causada por um ataque DDOS é comum encontrar técnicas de mitigação. Estas técnicas, na maioria das vezes trabalham com modelos de distribuição de carga ([Belyaev and Gaivoronski 2014] e [Sattar et al. 2016]) ou com a reorganização dos controladores a partir da falha de um componente da estrutura ([Macedo et al. 2016]). [Belyaev and Gaivoronski 2014] através um modelo de distribuição de carga na camada 4 do modelo OSI. Visto que uma rede pode ter mais de um caminho para um mesmo destino, eles mapearam toda a rede em canais, e conforme esses canais fossem atingindo um limite de tráfego, o controlador ia distribuindo os fluxos pelos demais canais. Esta abordagem, pode solucionar o problema estrangulamento de banda que um ataque pode causar. Porém, esse modelo de distribuição também gera uma grande carga no canal de controle. Através de um módulo de mitigação. [Sattar et al. 2016] calculam o limite de tráfego que cada servidor pode suportar, e o tráfego que todos os servidores podem suportar juntos. Quando um servidor atingir seu limite de acessos (falhar), o controlador envia um sinal de *reset* para os *switches* responsáveis pela rota, que removem todas as tabelas de fluxo. Isto força os *switches* a fazer uma nova solicitação para o controlador, que vai realizar a reinstalação das regras

de fluxo de forma adaptativa. Essa abordagem exige muito do controlador, pois a cada *reset*, se já existir uma quantidade significante de fluxos na rede, o mesmo vai atingir o limite de banda do canal de controle, podendo comprometer a funcionalidade do mesmo. [Macedo et al. 2016] propôs o PATMOS, um protocolo para mitigação de ataques. Este protocolo realiza a detecção de controladores falhas através da técnica *gossip*. Que a partir que um controlador for constatado falho, é dado início a um processo de eleição com o objetivo de selecionar um controlador para compartilhar a carga de trabalho. No entanto, a técnica *gossip* além de consumir uma quantidade significante de mensagens, trabalha de forma aleatória. Uma característica comum de todos os trabalhos citados neste parágrafo é o consumo excessivo de trocas de mensagens.

Os trabalhos que propõe melhorar o processo de eleição, se dividem em eleger somente o líder ([Lee et al. 2007], [Vasudevan et al. 2003]) ou eleger um grupo de apoio ao líder ([Macedo et al. 2016] [Zhang et al. 2009]). No trabalho de [Lee et al. 2007] cada nó organiza uma lista de cinco líderes, em ordem crescente de peso. No entanto, a tolerância a falhas é limitada. [Vasudevan et al. 2003], propôs um protocolo de eleição baseado em saltos. Eles usam sincronização através do *clock*. Porém não oferece tolerância a falhas e o processo demanda de muitas trocas de mensagens entre os nós. Como citado anteriormente, uma característica comum das abordagens que não elegem um grupo de apoio é a falta de tolerância a falhas durante a eleição, o que pode ser prejudicial para o bom funcionamento da eleição e da rede. [Macedo et al. 2016] propõe que, a partir da detecção falha de um controlador, todos os nós precisam se candidatar para serem eleitos. Esta abordagem demanda uma grande quantidade de troca de mensagens. [Zhang et al. 2009] também usa sincronização através do *clock* para eleger o líder. Porém a partir de que o líder falhar e consequentemente, o vice-líder for eleito caso uma falha no novo líder acontecer, o protocolo precisa ser reiniciado. Apesar da tolerância a falhas do líder, estas propostas ainda demandam de uma grande quantidade de troca de mensagens, podendo comprometer a banda disponível na rede. Caso a rede esteja sobre ataque DDoS o alto consumo de banda pode comprometer ainda mais o correto funcionamento da rede.

O protocolo proposto neste trabalho, propõe eleger um líder com disponibilidade para realizar rotinas de manutenção que detectam falhas e instabilidades entre os controladores. Além disso, também tolera falhas e instabilidades do próprio líder. O processo de eleição é semelhante ao de [Macedo et al. 2016], onde são eleitos um vice-líder e um grupo de elite. Porém, no trabalho de [Macedo et al. 2016], o líder é eleito a partir da detecção de falha, neste protocolo nós elegemos um líder que realiza rotinas para detectar a falha. Esta abordagem, reduz significativamente a quantidade de mensagens para realizar a manutenção e eleição dos controladores. dispositivos de plano de dados. Isto permite o desenvolvimento de aplicações em alto nível para controle e segurança da rede.

4. Protocolo de Eleição Tolerante a Falhas e Instabilidades para Controladores SDN

Este trabalho apresenta um protocolo de eleição que tolera falhas e instabilidades em um ambiente SDN com múltiplos controladores. A principal ideia deste protocolo é manter o correto funcionamento da rede, mesmo com qualquer controlador disposto sob situação de falha ou instabilidade. Para isso definimos um padrão de comunicação através de mensagens, que possibilita eleger um controlador líder, um vice-líder e três membros para o grupo de elite. O líder assim que eleito, realiza rotinas de monitoramento para

detectar um controlador comprometido. Visto que o líder também é suscetível a falhas e instabilidades, o vice-líder é eleito para agilizar e reduzir quantidades de mensagens na rede ao eleger um novo líder. Então, assim que o líder se caracterizar falho ou instável, o vice-líder envia uma mensagem de verificação para todos os controladores e se mais de um detectou a falha do líder, ele assume a posição de líder e realiza uma nova eleição entre os controladores pertencentes ao grupo de elite.

O processo de eleição é responsável por eleger um controlador (nó) que tenha disponibilidade de recursos suficientes para realizar a manutenção dos demais controladores. Os principais parâmetros utilizados para classificar os nós são a capacidade de processamento e a ordem de conexão. Então, os dois primeiros nós que se conectam à rede se organizam em uma lista ordenada pela capacidade de processamento. A partir disso, o nó com maior capacidade de processamento é eleito líder, o segundo nó ganha o status de vice-líder e os próximos três nós que se conectarem são eleitos membros do grupo de elite. Caso necessário uma segunda eleição, os nós são organizados somente pela capacidade de processamento. Assim que eleitos, o líder atualiza as informações de todos os nós da rede rotineiramente com o objetivo de detectar controladores falhos ou instáveis. Um controlador é detectado falho no momento que não responder a solicitação de manutenção realizada pelo líder. Uma vez que declarado falho, o nó ganha o status de instável, não podendo ser eleito para nenhum dos cargos de liderança. Caso o nó já faça parte de algum dos cargos de liderança o mesmo é substituído. Esta abordagem evita que controladores instáveis sejam eleitos.

4.1. Modelagem do Sistema

Nós representamos uma SDN com *n* controladores conectados. Onde para identificação e organização dos nós, foram definidos dois objetos padrões. Estes objetos contém as informações que controlam todo o comportamento dos controladores sob o protocolo. Então, cada nó é composto pelos objetos *node* e *network_status*. Para estes objetos são definidos alguns atributos que guardam as informações mais relevantes para a realização das eleições e das manutenções. O objeto *node*(**Tabela 1**) mantém as informações necessárias para identificação e classificação do mesmo. O atributo *cpu_status* é atualizado toda vez que o líder realiza uma solicitação de manutenção e o *status* é atualizado conforme o resultado das eleições. O atributo *status* pode ser definido com único valor dentre três valores padrões 'available', 'leader', 'vice-leader', 'elite' ou 'unstable'. O objeto *network_status* (**Tabela 3**) apesar de estar presente em todos os nós, somente o líder realiza atualizações nele. Isto garante que o líder sempre vai ter as informações de um estado confiável da rede para garantir uma tomada de decisões eficiente. O líder também é responsável por manter os atributos do objeto *network_status* em todos os *n* nós conectados à rede.

node		
Atributo	**Tipo**	**Descrição**
Mac_address	*String*	Endereço de MAC do nó
Total_cpu	*Float*	Capacidade total de processamento
Status_cpu	*Float*	Porcentagem do estado atual do consumo de CPU
Status	*String*	Estado atual do nó

Tabela 1. Tabela de atributos do objeto *node*

Estabelecemos também um padrão de mensagens para a realização das eleições e das manutenções. Todas as mensagens recebidas por um nó, são adicionadas em uma

current(node)		
Atributo	**Tipo**	**Descrição**
Concurrent	*Booleano*	Indica se o nó está realizando alguma atividade como eleição, tratamento e resposta de alguma mensagem ou manutenção em caso do nó atual ser líder.
Last_leader_message	*Float*	Toda vez que o nó recebe uma mensagem do líder, este atributo é atualizado. Uma vez que o tempo limite estabelecido for atingido, o nó declara situação de instabilidade.
Count_messages	*Integer*	Conta todas as mensagens trocadas pelo nó. O valor é somado toda vez que o nó recebe e envia uma mensagem
Waiting	*Booleano*	Indica que o líder está aguardando e priorizando mensagens RMM.
Unstable	*List*	Todos os nós que não responderem as mensagens de manutenção do líder são armazenados aqui com o endereço de MAC e um índice que é acrescentado quando ele não responde as futuras manutenções ou decrementado quando ele responde. Se o índice for menor que -2 ele volta ao estado 'available' podendo ser eleito.
RMM_list	*List*	Fila que espera. Quando o Líder realiza uma manutenção, ele adiciona os nós nesta lista. Assim conforme for recebendo as mensagens de resposta ele elimina os nós.

Tabela 2. Tabela de atributos do objeto *current*

network_status		
Atributo	**Tipo**	**Descrição**
Leader	*String*	Endereço de MAC do líder eleito
Vice-leader	*String*	Endereço de MAC do Vice-líder eleito
Elite-group	*List*	Lista com os endereços dos membros do grupo de elite
Nodes	List	Lista de objetos node
Election	*Booleano*	Indica se a rede está ou não em estado de eleição

Tabela 3. Tabela de atributos do objeto *network_status*

fila. Esta fila pode sofrer alterações na estrutura, exclusão de algumas mensagens, conforme a ocasião. As mensagens são compostas por um cabeçalho e alguns atributos de *network_status* e *node*.

- EM: inicia uma eleição, esta mensagem é disparada na rede, via broadcast, quando o nó se conecta à rede ou perde o contato com o líder, ela contém os atributos *network_status.election*, *network_status.nodes*, e todos os atributos do objeto *node*;
- AIM: mensagem de informação e de reconhecimento, esta mensagem é enviada via unicast e contém os todos os atributos do objeto *network_status*. Ela é enviada quando um nó se conecta à rede e necessita atualizar as informações dos objetos.
- UM: mensagem de atualização, envia via broadcast os atributos atualizados do objeto *network* sempre que uma eleição é realizada e também ao final de toda rodada de manutenção.
- MM: mensagem de manutenção que o líder envia para todos os nós da rede via multicast. Ela serve basicamente como uma requisição de atualização.
- RMM: resposta da mensagem de manutenção via unicast, assim que o nó recebe uma MM ele responde para o líder com esta mensagem. Ela contém todos os atributos do objeto *node*.

Para controle de envio, recebimento e das estruturas de dados, uma herança do objeto *node* foi criada, o objeto *current* (**Tabela 2**). Este objeto mantém atributos essenciais para o controle de tempo individual de cada nó, controle das estruturas de aguardo de mensagens específicas e dos nós instáveis.

4.2. Visão Geral do Protocolo

Para início do protocolo vamos assumir que nenhum nó se conecta ao mesmo tempo do outro e que todos os nós estão conectados por links e estruturas confiáveis. As mensagens são dispostas em uma fila, onde cada uma é tratada e respondida por vez. Outra informação importante é que as mensagens *EM* tem preferência em todas as filas, mesmo quando o líder esteja aguardando mensagens de resposta. Um controlador é declarado instável a partir de que ele não responde as mensagens de manutenção. Quando ele fica por mais de 3 rodadas de manutenção marcado como instável ele é eliminado das listas, ficando apenas na lista de instável.

O protocolo é iniciado a partir da conexão do segundo nó, o primeiro que se conectou realiza o processo de eleição. Caso seja eleito, o mesmo encaminha uma *UM*, caso não, ele encaminha uma *AIM* para o nó eleito. Isto garante que somente o líder atualize as informações do objeto *network*. Quando o segundo nó receber uma mensagem *AIM* informando que ele é o líder, ele realiza uma nova eleição e encaminha uma *UM*. A partir disso, o líder começa os ciclos de monitoramento. Para o monitoramento, basicamente o líder envia uma mensagem de requisição MM para todos os nós conhecidos e aguarda a resposta dos demais. Para isso, o líder utiliza os atributos *current.waiting* e *current.RMM_list*. Enquanto aguarda as mensagens o atributo *current.waiting* é marcado como verdadeiro e a lista *current.RMM_list* é incrementada com o endereço de MAC de cada controlador conhecido pelo líder. Com a opção *current.waiting* marcada como verdadeira o líder dá prioridade na fila de mensagens para as mensagens com cabeçalho *RMM*.

Um momento considerado crítico é quando o líder entra em estado de falha. A **Figura 3** demonstra a reação do protocolo diante desta situação. A) j envia uma *MM* e recebe de todos os nós uma *RMM*. B) j caiu, os nós ficam o tempo de verificação sem se comunicar com o líder e enviam uma mensagem de eleição. Como k era o vice-líder se auto elege líder, declara a falha do líder e elege os nós do grupo de elite. C) O novo vice-líder envia e uma UM para todos os nós, deixando a rede organizada como na fase C.

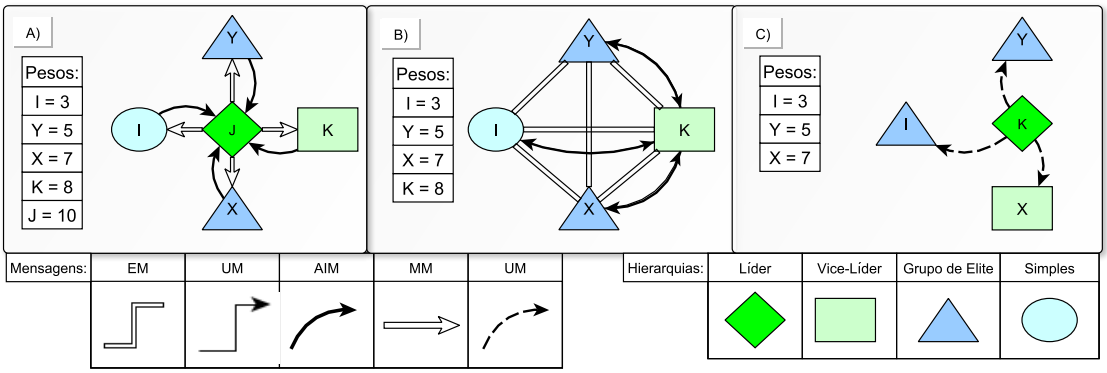

Figura 3. Um exemplo do protocolo reagindo em situação de falha do líder.

5. Avaliação de Desempenho

Nesta seção vamos apresentar os resultados obtidos a partir das avaliações e testes do protocolo sob diferentes cenários. Os objetivos da avaliação consistem em analisar a eficiência do protocolo diante de uma situação de falha do líder ocasionadas por ataques de negação de serviço e sobrecarga legítima. Além disso, avaliar o processo de eleição. Para comprovar a eficiência do protocolo foram elaborados alguns cenários de testes, onde o mesmo é posto em situação de falhas ou instabilidade do líder.

As métricas utilizadas foram, quantidades de mensagens e a taxa de utilização de CPU. A quantidade de mensagens é contabilizada a partir da falha do líder, ela representa o impacto do protocolo na utilização da rede. A quantificação das mensagens foi feita através do sistema de *log* implementado no protocolo. O consumo de CPU foi coletado a partir da biblioteca *PSutil*, para análise do comportamento do protocolo em situação de instabilidade. Esta métrica mostra os indícios das instabilidades no controlador, gerados pelos ataques e sobrecargas.

O ambiente de testes foi composto por quatro controladores, todos com os mesmos componentes de software, a **Figura 4** mostra a estrutura do ambiente. A figura ilustra que os nem todos os dispositivos apresentam a mesma configuração de hardware e o controlador utilizado foi o POX. Os cenários utilizados foram geranos apartir dos defeitos induzidos no controlador. No primeiro cenário, o líder era desconectado periodicamente fazendo uma rotatividade em todos os nós. Para o cenário foram utilizados um *script* que simulava a queda da porta de conexão. A cada rodada de testes, o papel de líder era alternado entre os nós. No segundo cenário foram utilizadas duas ferramentas. Sendo elas uma ferramenta de *Benchmarking*, o *Cbench* [Sherwood and Yap 2011] e um *script* que simula ataques DDoS [Mousavi 2014].

Leader Election			
Instante	**Emissor**	**Receptor**	**Cabeçalho**
1	H1	ALL	EM
2	H1	H2	EM
3	*H2*	H1	AIM
4	H1	ALL	UM

Tabela 4. Exemplo da troca e mensagens em uma eleição

Leader Down			
Instante	**Emissor**	**Receptor**	**Cabeçalho**
0	***Leader Down***		
1	Vice-leader	ALL	AIM
3	*Elite*	Vice-Leader	AIM
4	Leader*	ALL	UM

Tabela 5. Exemplo da troca de mensagens em no cenário de queda do líder

As **Tabela 5** e **Tabela 4** mostram os resultados da métrica quantidade de mensagens nos cenários que simulam a queda do líder e a eleição do líder a partir da conexão do primeiro nó. Estas tabelas apontam que foram trocadas três mensagens para eleger o líder, mostrando que o protocolo consegue estabelecer a estrutura de lideranças utilizando poucas mensagens. A **Figura 5** mostra a métrica utilização de CPU no cenário de geração de carga. Os resultados obtidos mostram uma utilização constante do CPU. Este percentual reflete a quantidade a quantidade de carga gerada através do ataque e da ferramenta de *Benchmark*.

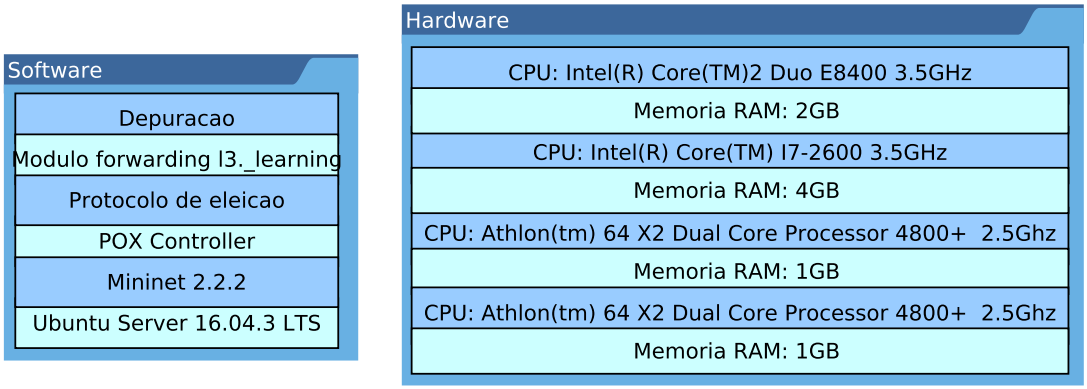

Figura 4. Estrutura do cenário

6. Conclusões

Este trabalho apresentou um protocolo de eleição tolerante a falhas e instabilidades para controladores SDN. O protocolo proposto utiliza uma baixa quantidade de envio de mensagens, diferentemente dos trabalhos existentes. Também diante da falha do líder, o protocolo proposto consegue reestabelecer a estrutura das lideranças utilizando poucas trocas de mensagens. Uma avaliação de desempenho foi conduzida utilizando quatro computadores como controladores, executando o protocolo proposto. Foram avaliados, cenário com instabilidades geradas por desconexões e ataques de negação de serviço. Os resultados mostram que o protocolo consegue reestabelecer os grupos de liderança mesmo diante de sobrecargas. Como trabalhos futuros serão realizadas avaliações de desempenho considerando outros tipos de ataques.

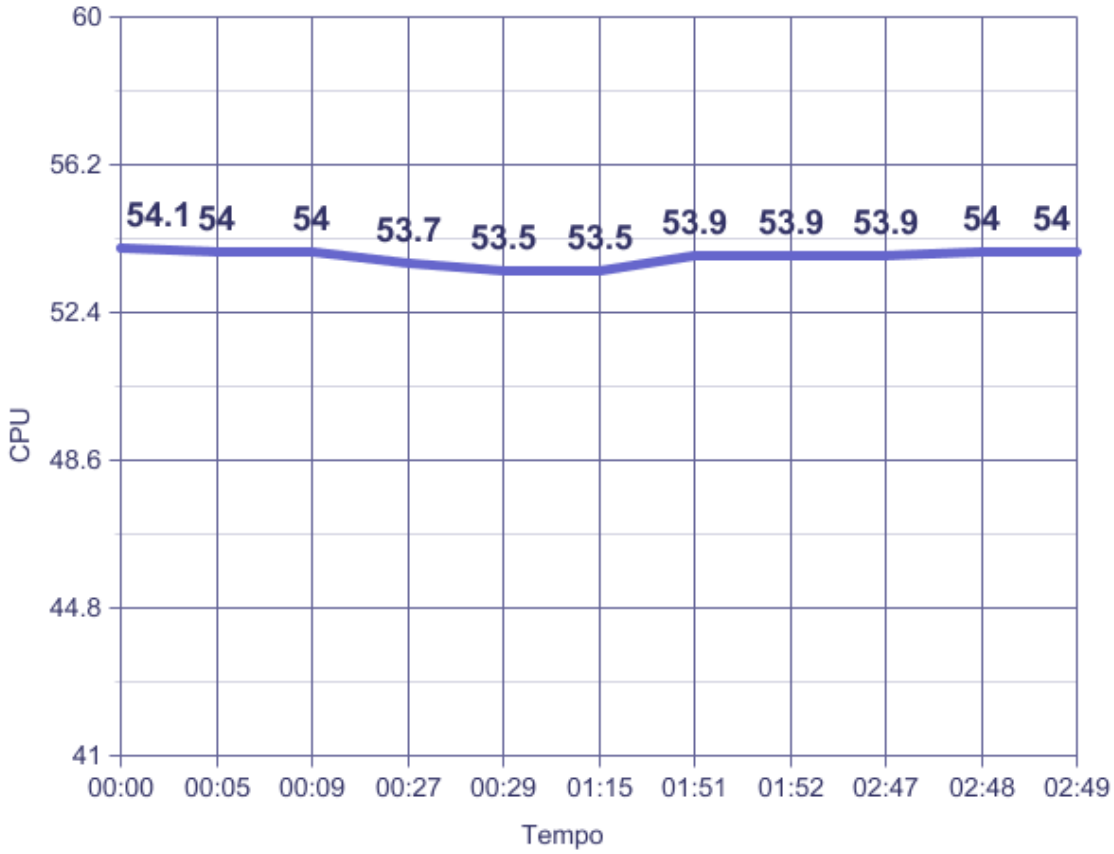

Figura 5. Taxa de utilização de CPU no cenário de sobrecarga

Referências

Belyaev, M. and Gaivoronski, S. (2014). Towards load balancing in sdn-networks during ddos-attacks. In *Science and Technology Conference (Modern Networking Technologies)(MoNeTeC), 2014 First International*, pages 1–6. IEEE.

Dao, N.-N., Park, J., Park, M., and Cho, S. (2015). A feasible method to combat against ddos attack in sdn network. In *Information Networking (ICOIN), 2015 International Conference on*, pages 309–311. IEEE.

Guedes, D., Vieira, L., Vieira, M., Rodrigues, H., and Nunes, R. V. (2012). Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC 2012*, 30(4):160–210.

Hu, Z., Wang, M., Yan, X., Yin, Y., and Luo, Z. (2015). A comprehensive security architecture for sdn. In *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*, pages 30–37. IEEE.

Kim, H. and Feamster, N. (2013). Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119.

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings*

of the IEEE, 103(1):14–76.

Lee, S., Muhammad, R. M., and Kim, C. (2007). A leader election algorithm within candidates on ad hoc mobile networks. In *International Conference on Embedded Software and Systems*, pages 728–738. Springer.

Macedo, R., de Castro, R., Santos, A., Ghamri-Doudane, Y., and Nogueira, M. (2016). Self-organized sdn controller cluster conformations against ddos attacks effects. In *Global Communications Conference (GLOBECOM), 2016 IEEE*, pages 1–6. IEEE.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

Mousavi, S. M. (2014). Early detection of ddos attacks in software defined networks controller. *PhD diss., Carleton University Ottawa*.

Sattar, D., Matrawy, A., and Adeojo, O. (2016). Adaptive bubble burst (abb): Mitigating ddos attacks in software-defined networks. In *Telecommunications Network Strategy and Planning Symposium (Networks), 2016 17th International*, pages 50–55. IEEE.

Schmid, S. and Suomela, J. (2013). Exploiting locality in distributed sdn control. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 121–126. ACM.

Sherwood, R. and Yap, K. (2011). Cbench controller benchmarker. *Last accessed, Nov*.

Vasudevan, S., DeCleene, B., Immerman, N., Kurose, J., and Towsley, D. (2003). Leader election algorithms for wireless ad hoc networks. In *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, volume 1, pages 261–272. IEEE.

Zhang, G., Kuang, X., Chen, J., and Zhang, Y. (2009). Design and implementation of a leader election algorithm in hierarchy mobile ad hoc network. In *Computer Science & Education, 2009. ICCSE'09. 4th International Conference on*, pages 263–268. IEEE.