Análise Comparativa de Controladores para Redes Definidas por Software de Classe *Carrier Grade*

Caio César Ferreira

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2016

Caio César Ferreira

Análise Comparativa de Controladores para Redes Definidas por Software de Classe *Carrier Grade*

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Prof. Dr. Pedro Frosi Rosa
Coorientador: Prof. Dr. Flávio de Oliveira Silva

Uberlândia
2016

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

F383a 2016

Ferreira, Caio César, 1988-
Análise comparativa de controladores para redes definidas por software de Classe Carrier Grade / Caio César Ferreira. - 2016.
151 f. : il.

Orientador: Pedro Frosi Rosa.
Coorientador: Flávio de Oliveira Silva.
Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.

1. Computação - Teses. 2. Internet - Teses. 3. Redes de computadores - Protocolos - Teses. I. Rosa, Pedro Frosi. II. Silva, Flávio de Oliveira, 1970- III. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

CDU: 681.3

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Análise Comparativa de Controladores para Redes Definidas por Software de Classe *Carrier Grade***" por **Caio César Ferreira** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação.**

Uberlândia, 11 de Março de 2016

Orientador: _____

Prof. Dr. Pedro Frosi Rosa
Universidade Federal de Uberlândia

Co-orientador: _____

Prof. Dr. Flávio de Oliveira Silva
Universidade Federal de Uberlândia

Banca Examinadora:

_____

Prof. Dr. José Gonçalves Pereira Filho
Universidade Federal do Espírito Santo

_____

Prof. Dr. Rodrigo Sanches Miani
Universidade Federal de Uberlândia

Dedico este trabalho aos meus ídolos, minha mãe Waldete Ferreira, à minha namorada Maria Fernanda Ferreira da Silva, à minha tia Dalva Luiza Ferreira e aos meus mestres Pedro Frosi Rosa e Flavio Oliveira, meus grandes incentivadores.

Agradecimentos

Agradeço à minha família, em especial à minha mãe Waldete e minha namorada Maria Fernanda, pelo apoio incondicional em todos os caminhos que resolvi percorrer no âmbito pessoal, profissional e acadêmico. No fim aprendemos que família e conhecimento são as maiores riquezas.

Agradeço aos colegas do projeto EDOBRA, Doughert, Enrique, Luiz Gustavo Borges, Jhon Martinez pela troca de experiências e aprendizado. Também aos colegas do grupo MEHAR, em especial ao meu amigo Flávio Silva, meu coorientador, que mostrou como precisamos de pessoas desbravadoras e de personalidade única.

Gostaria de deixar um agradecimento especial aos grandes amigos Natal Neto e Maurício Gonçalves, companheiros desta jornada e da vida. Este trabalho seria menos produtivo e certamente menos divertido sem vocês.

Agradeço aos docentes e técnicos da Faculdade de Computação pelo ensinamento e suporte prestados nos últimos anos. É uma honra poder dizer que, após esses anos, faço parte de uma comunidade tão importante.

Agradeço à Algar Telecom pelo apoio para a participação em congressos, e aos colegas de trabalho pela motivação que sempre proporcionaram. Neste âmbito, agradeço em especial ao João Henrique de Souza Pereira, amigo pesquisador e grande incentivador, pelos conselhos e diretrizes tão relevantes durante todo este período.

Por fim, gostaria de deixar um agradecimento especial ao meu amigo Pedro Frosi Rosa, orientador deste trabalho e responsável por me ensinar que um bom café entre amigos pode resultar em grandes ideias que não surgiriam nas reuniões em salas fechadas.

"Juntos nós resistimos, divididos nós caímos...", por isso agradeço a todas as pessoas que contribuíram de alguma maneira para a minha conquista, pois sem elas nada disso seria possível.

"L'homme est condamné à être libre." *Jean-Paul Sartre - L'Être et le Néant*

Resumo

O projeto original da Internet foi concebida na década de 1960, em um contexto totalmente diferente do atual. Nesse tempo, a rede ganhou novos propósitos e passou a ser utilizada em áreas e atividades que seriam impensáveis durante a sua concepção. As novas aplicações às quais a rede foi submetida trouxeram consigo diversos novos requisitos que, em sua maioria não foram adequadamente atendidos devido a limitações na arquitetura. A Internet enfrenta desafios importantes para sua evolução, vislumbrados desde a década de 1990, quando começou a enfrentar sérios problemas relativos ao esgotamento de sua capacidade de endereçamento, ou até mesmo antes, quando se percebeu a necessidade de um identificador com maior expressividade semântica. Uma abordagem que vem tomando força nos últimos anos é a SDN (*Software Defined Network*), que cria uma camada de abstração para o controle dos elementos de rede, promovendo uma separação entre os planos de Dados e de Controle. Em todo o mundo, a infraestrutura das redes de dados é provida, em geral, pelas operadoras de telecomunicações, então há que se pensar nas redes do futuro com qualidade *carrier grade*. O termo *carrier grade* é originário da área de telecomunicações, mas hoje em dia em diversas áreas é sinônimo de alta disponibilidade, escalabilidade, qualidade de serviço e segurança. Deste modo, qualquer tecnologia de redes de computadores, e o OpenFlow em particular, deve apresentar uma solução com capacidade *carrier grade* para se tornar uma tecnologia candidata a implementar as redes de dados das operadoras de telecomunicações no futuro. O objetivo deste trabalho é fazer uma análise comparativa experimental dos principais controladores OpenFlow para verificar suas propriedades *carrier grade* que os tornem potenciais soluções a ser utilizadas por operadoras de telecomunicações.

Palavras-chave: SDN; Controladores OpenFlow; ONOS; OpendayLight; Internet do Futuro; *Clean Slate*; Alta disponibilidade; Escalabilidade; *Carrier Grade*; SLEE; *Cluster* de Controladores.

Abstract

The original design of the Internet was conceived during sixties, into a whole different context of the current days. In this meantime new proposals have been added to the network which became used into the unthinkable areas and activities during its conception. New applications for which the network was subjected brought with them many new requirements, most of whom were not adequately met due to architectural limitations. The Internet evolution faces major challenges, foreseen from the nineties, when it began to face serious problems concerning the exhaustion of its addressing capabilities, and even before, when it was realised that an identifier with greater semantic expressiveness was needed. The SDN is an approach that has earned strength in recent years, by creating two abstraction layers for the network elements, which promotes the independence of the Data Plan and Control Plan. Around the world, the data network infrastructure is provided in general by the telecom operators, thus we must think on the future of networks with carrier grade quality. Carrier grade is a term originated in telecom area, but nowadays in many areas is synonymous with high availability, scalability, quality of service and security. Therefore, computer network technology, including OpenFlow, must present a solution with carrier grade capacity to become a candidate technology to implement the data networks of the telecommunication operators in the future. The goal of this work is to make a comparative experimental analysis of the main OpenFlow controllers to check their carrier grade properties that make them potential solutions being used by telecom operators.

Keywords: SDN; OpenFlow Controllers; ONOS; OpendayLight; Future Internet; Clean Slate; High Availability; Scalability; Carrier Grade; SLEE; Cluster of Controllers.

Lista de ilustrações

Lista de tabelas

Lista de siglas

API *Application Programming Interface*

CLI *Command Line Interface*

FACOM Faculdade de Computação da Universidade Federal de Uberlândia

HA *High Availability*

IP *Internet Protocol*

ONF *Open Networking Foundation*

ONOS *Open Network Operating System*

QoS *Quality of Service*

QoE *Quality of Experience*

SAL *Service Abstraction Layer*

SDN *Software Defined Networking*

TCP *Transmission Control Protocol*

Sumário

Capítulo 1

Introdução

A Internet foi concebida na década de sessenta (BARAN, 1964) e, apesar das camadas Física e de Aplicação apresentarem evoluções significativas desde sua criação, as camadas de Rede e de Transporte praticamente não foram alteradas nesse período. Há apenas quatro décadas, as principais proposições basicamente se referiam a aspectos de confiabilidade e de baixo *overhead* na troca de mensagens, enquanto hoje contemplam necessidades bem mais complexas, tais como *Quality of Service* (QoS), *Quality of Experience* (QoE), mobilidade, segurança, *multicast* e *real-time*.

A Internet enfrenta desafios importantes para sua evolução (ZAHARIADIS et al., 2011), vislumbrados desde a década de noventa (CLARK et al., 1991) quando começou a enfrentar sérios problemas relativos ao esgotamento de sua capacidade de endereçamento (EGEVANG; FRANCIS, 1994; GROUP; HINDEN, 1993), ou até mesmo antes, quando se percebeu a necessidade de um identificador com maior expressividade semântica (POSTEL, 1983; MOCKAPETRIS, 1983a; MOCKAPETRIS, 1983b).

A explosão da Internet a partir dos anos 1990 tornou a operação das redes uma atividade onerosa e de alto risco, impondo grande complexidade aos planos de Dados (REKHTER; LI, 1995), à Arquitetura e ao plano de Controle (AGUIAR, 2008).

Nesse contexto, pesquisadores do mundo todo têm trabalhado em duas abordagens (REXFORD; DOVROLIS, 2010): a "revolucionária"ou *clean slate* (ROBERTS, 2009), que não se limita à especificação da arquitetura atual e possibilita a concepção de novas ideias; e a "evolucionária", que visa evoluir a arquitetura atual, mantendo os seus princípios básicos.

É consenso que qualquer que seja o novo modelo de Internet aceito para o futuro, ele deverá ser flexível para suportar as mudanças de requisitos das próximas gerações de aplicações. Uma abordagem que vem tomando força nos últimos anos é a SDN (*Software Defined Network*) (GOTH, 2011; GREENE, 2009), que cria uma camada de abstração para o controle dos elementos de rede (NE – *Network Element*), promovendo uma separação entre os planos de Dados e de Controle. A ortogonalidade desses planos permite que as redes possam ser programadas para atender comportamentos específicos, o que abre uma

ampla margem de possibilidades para as redes atuais e futuras. Essa camada programável está para as redes, assim como o sistema operacional está para os computadores.

O plano de Controle na tecnologia OpenFlow (primeira materialização de uma rede SDN) é implementado por um software denominado controlador, que é um sistema que controla a rede utilizada na abordagem SDN. O controlador OpenFlow é responsável por: gerir o ciclo de vida de *hosts* e *switches* na rede; por armazenar informações e estatísticas da rede; e por estabelecer a comunicação entre aplicações hospedada em *hosts* através de configurações nos elementos de rede.

A relevância deste trabalho está em apresentar comparação entre os controladores SDN que atuam no plano de controle e definir uma plano de testes para avaliar se atendem os requisitos *carrier grade* (JOHN et al., 2013; SHARMA et al., 2013) e se estão aptos serem implantados na indústria.

1.1 Motivação

Em todo o mundo, a infraestrutura das redes de dados é provida, em geral, pelas operadoras de telecomunicações, então há que se pensar nas redes do futuro com qualidade *carrier grade*. O termo *carrier grade* é originário da área de telecomunicações. Hoje em dia em diversas áreas é sinônimo de alta disponibilidade, escalabilidade, qualidade de serviço e segurança.

Deste modo, qualquer tecnologia de redes de computadores, e o OpenFlow em particular, deve apresentar uma solução com capacidade *carrier grade* para se tornar uma tecnologia candidata a implementar as redes de dados de operadoras de telecomunicações no futuro.

Embora as publicações dos principais projetos de controladores OpenFlow apresentem diversas propriedades desejáveis em uma solução *carrier grade*, incluindo aqueles escolhidos para compor a análise de trabalhos correlatos apresentada no Capítulo 2, até onde foi possível pesquisar no intervalo de tempo deste projeto, não se encontrou um trabalho que fizesse uma análise experimental com esses controladores.

1.2 Hipótese

Acredita-se que há controladores OpenFlow estáveis, e com características *carrier grade* para serem utilizados em projetos de redes por operadoras de telecomunicações, capazes de proporcionar maior flexibilidade e garantir um melhor entendimento de requisitos de aplicações atuais e futuras.

1.3 Objetivos Geral e Específicos

O objetivo geral deste trabalho é avaliar controladores OpenFlow disponíveis em versões estáveis, especificamente aqueles que se propõem a oferecer características *carrier grade* tais como alta disponibilidade, tolerância a falha entre outros. Foram escolhidos os dois principais controladores OpenFlow que, no momento, apresentam tais características. Para se alcançar o objetivo geral deste trabalho, são previstos os seguintes objetivos específicos:

- ❏ Apresentar uma análise dos principais controladores OpenFlow da atualidade;
- ❏ Definir os dois controladores OpenFlow mais próximos de características *carrier grade*;
- ❏ Especificar um Plano de Testes para os dois controladores e apresentar os resultados desses testes; e,
- ❏ Apresentar análise comparativa dos resultados obtidos através do plano de testes.

1.4 Organização da Dissertação

Afim de alcançar os objetivos citados, essa dissertação está organizada da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica e trabalhos correlatos que serviram de base para este trabalho; o Capítulo 3 apresenta uma visão geral do processo experimental, define os planos de testes e analisa os resultados obtidos; o Capítulo 4 apresenta as conclusões, contribuições e trabalhos futuros; o Apêndice A oferece o *script* para a criação da topologia utilizada nos experimentos; o Apêndice B oferece os *scripts* para o ambiente de testes TestON visando a verificação de propriedades em controladores OpenFlow em *cluster*; e o Apêndice C apresenta o *script* para a instanciação de controladores OpendayLight em *cluster*.

Capítulo 2

Fundamentação Teórica

Este capítulo tem por objetivo apresentar uma visão geral do paradigma SDN e sua relação com a tecnologia OpenFlow, descrever o ecossistema de controladores e detalhar os dois principais controladores - ONOS e OpenDaylight - que têm sido apoiados pela indústria. Para os controladores escolhidos, são feitas análises sobre suas características *carrier grade*.

2.1 Visão paradigma SDN

As Redes Definidas por Software (SDN – *Software Defined Networking*) foram introduzidas com o propósito de permitir o estudo de novas propostas de arquiteturas de rede e de protocolos da rede Internet atual (FARHADY; LEE; NAKAO, 2015) (KREUTZ et al., 2015). O principio básico da SDN é permitir a programação em regime operacional de elementos de rede (roteadores, comutadores dentre outros). A Fig. 1 representa uma comparação entre o modelo atual e a abordagem SDN(ONF SOLUTION BRIEF, 2013).

A SDN se fundamenta na separação dos planos de Controle e de Dados, visando tornar a rede programável e mais flexível em regime permanente, melhorando, então, a utilização dos recursos. A SDN abre a possibilidade de exploração de novos negócios e pode promover inovações na indústria de serviços de rede, conforme previsão da *Open Networking Foundation* (ONF) (Open Networking Foundation - ONF, 2014).

A ONF (opennetworking, 2015) é uma organização colaborativa que vem fomentando de forma pioneira a adoção de SDN em todo mundo. A arquitetura SDN, proposta pela ONF, define três camadas principais (Open Networking Foundation - ONF, 2014):

- Camada de Aplicacão: consiste em entidades de aplicação que utilizam os serviços SDN, através de *Application Programming Interface* (API), que tornam possível especificar o comportamento desejável da rede, oferecendo uma visão abstrata da rede para fins de tomada de decisão;

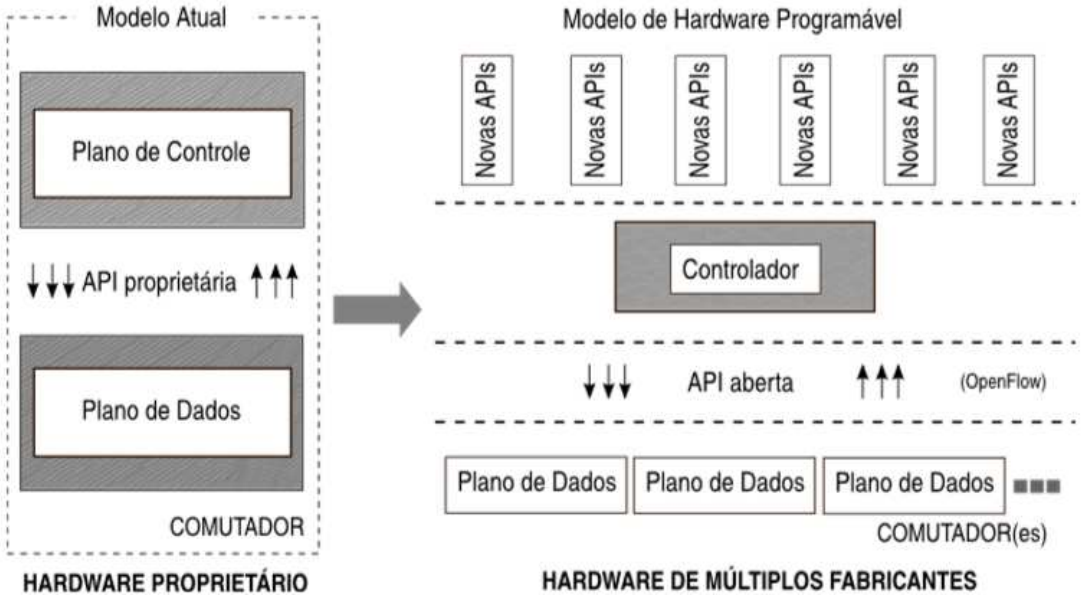

Figura 1 – Modelo programável (SDN) vs. Modelo atual. (SILVA, 2015)

- Camada de Controle: esta camada é implementada como um plano de controle, logicamente centralizado, que traduz requisitos da camada de aplicação para a camada subjacente, e oferece às aplicações uma visão abstrata da rede; e
- Camada de Infraestrutura: composta de elementos da rede, interconectividade (topologia) e dispositivos que realizam a transmissão de dados propriamente dita.

A comunicação entre a camada de controle (plano de controle) e a camada de infraestrutura (plano de dados) é feita através de um protocolo padronizado. Desse modo, tanto o controlador (que implementa o plano de controle) quanto os elementos de rede, como roteadores e comutadores (que implementam o plano de dados), precisam interpretar esse protocolo.

A Universidade de Stanford propôs o protocolo OpenFlow para a comunicação entre essas camadas (ERICKSON, 2014). OpenFlow é uma materialização de SDN e é uma contribuição fundamental para a comunidade de pesquisa no âmbito da Internet do Futuro, permitindo o uso e a avaliação de mecanismos para inovar tanto o controle de rede quanto o transporte de dados.

Um *switch* OpenFlow 1.0 pode ser dividido basicamente em três partes, conforme mostra a Fig. 2:

- A Tabela de Fluxos (*Flow Table*), que indica quais fluxos o *switch* deve processar, sendo que para entrada dessa tabela (*flow*) é especificado um conjunto de ações que deve ser aplicado a cada fluxo;
- Um Canal Seguro, que conecta o *switch* ao Controlador através do protocolo OpenFlow, permitindo o envio de comandos (serviços) e pacotes; e

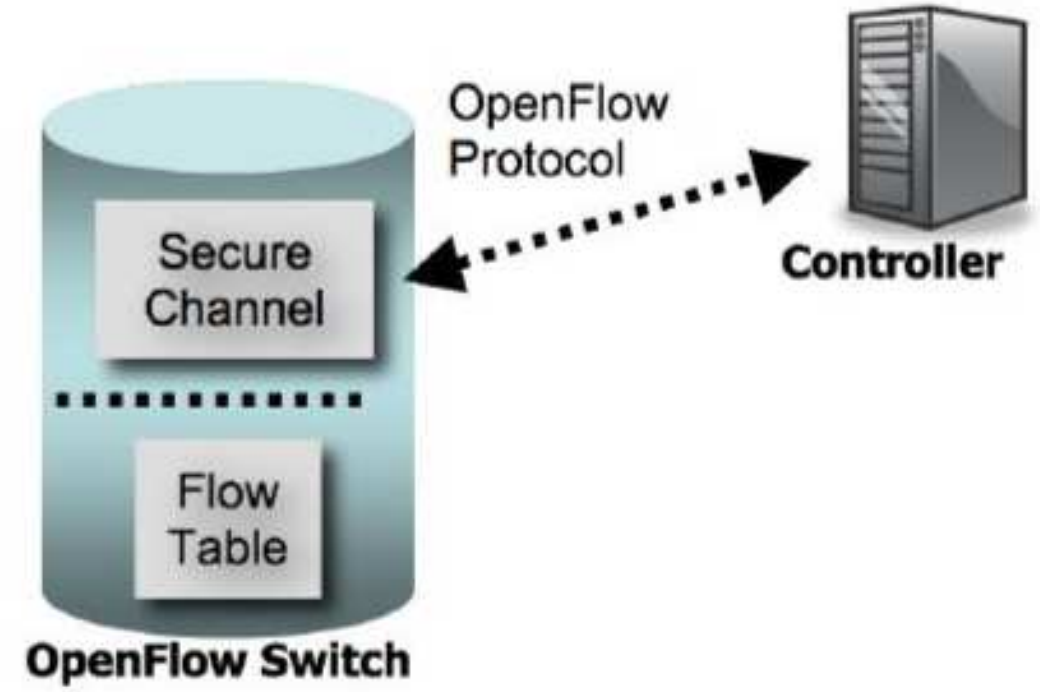

Figura 2 – Representação das partes de um *Switch* OpenFlow (Open Networking Foundation, 2009)

- O Protocolo OpenFlow que especifica os serviços disponíveis através do Canal Seguro.

As principais ações que um *switch* OpenFlow deve suportar são as seguintes:

- Encaminhar fluxo(s) de pacotes para uma determinada porta;
- Encapsular fluxo(s) de pacotes e encaminhar para o Controlador;
- Descartar fluxo(s) de pacotes.

Uma Tabela de Fluxos (*Flow Table*) consiste de entradas (*Flow Entry*), cujo formato com os principais componentes podem ser vistos na Tabela 1:

- *Match Fields*: Conjunto de campos utilizados para selecionar (*Match*) pacotes que chegam ao *switch*, podendo conter campos dos cabeçalhos de datagramas, *frames*, porta de entrada ou metadados previamente especificados;
- *Counters*: Existem contadores específicos para tabelas, fluxos, portas ou filas que fornecem informações estatísticas, como por exemplo, o tempo (duração) que uma regra de fluxo foi instalada no comutador, quantidade de pacotes ou bytes recebidos em uma determinada porta ou por um determinado fluxo;

Match Fields	Counters	Actions

Tabela 1 – Formato de uma entrada (*flow entry*) – Tabela adaptada

- *Actions*: Conjunto de ações que descrevem como um *switch* OpenFlow deve manipular os pacotes recebidos que combinaram (*match*) com a regra de fluxo especificada, sendo que se nenhuma ação for especificada, o pacote é descartado.

Um fluxo possui um conjunto de campos do cabeçalho (tuplas) do pacote recebido pelo dispositivo, como exemplificado pela Fig. 3. As tuplas são campos de protocolos das camadas de enlace, de rede ou de transporte, tendo como base a pilha de protocolos TCP/IP. O controlador OpenFlow gerencia e controla as tabelas de fluxos no *switch* OpenFlow. Contudo, (ROTHENBERG et al., 2010) ressalta-se que muitas aplicações de rede podem ser desenvolvidas e anexadas aos mesmos para uma infinidade de objetivos.

in port	Ethernet			VLAN		IP				TCP/UDP	
	src	dst	type	id	pri	src	dst	proto	ToS	src	dst

Figura 3 – Exemplo de campos do OpenFlow (ROTHENBERG et al., 2010)

2.2 Principais Controladores OpenFlow

Controladores são um elemento fundamental na arquitetura OpenFlow, pois materializam a separação dos planos de Controle e de Dados. Basicamente, o papel do controlador enseja em receber todo o tráfego enviado pelo *switch* e decidir a ação a ser aplicada ao tráfego recebido (McKeown et al., 2008).

Um Controlador OpenFlow se comporta como um servidor, que pode ser implantado em computadores de propósito geral, sendo que todo *switch* OpenFlow em operação na rede pode ou não estar ligado fisicamente a um controlador. Um *switch* OpenFlow, que não esteja ligado a um controlador, funciona como um *switch* comum (pré-OpenFlow), que não faz uso das funcionalidades fornecidas pelo OpenFlow. Há diversos controladores OpenFlow que se diferenciam em essência pelas linguagens de programação ou foco em um tipo de aplicação.

Dentre os primeiros controladores apresentados, destaca-se o NOX (GUDE et al., 2008)(NOXREPO, 2013), que foi o primeiro projeto de um controlador OpenFlow baseado na linguagem C/C++, com características de desempenho e I/O assíncrono – testado nas seguintes distribuições: Ubuntu, Debian e RHEL (*Red Hat Enterprise Linux*). O NOX

disponibiliza uma API que oferece uma visão da topologia de rede dos respectivos *switches* e enlaces.

Dos mesmos criadores do NOX, o controlador POX (NOXREPO, 2013) é desenvolvido em Python e oferece uma interface de programação simplificada, que permite a implementação rápida de novas aplicações de rede, facilitando o uso do controlador. O POX foi testado nos seguintes sistemas operacionais: MacOS, Linux e Microsoft Windows. O POX inclui a descoberta de topologia de rede, além de outras ferramentas do NOX. Uma comparação entre ambos mostra que o POX apresenta maior latência e menor *throughput* que o NOX.

Podem ser mencionados outros controladores tais como o Beacon (ERICKSON, 2014) e o Maestro (CAI, 2014) (desenvolvidos em linguagem Java); e o Ryu (NTT Communications, 2013) (desenvolvido em Python), todos capazes de suportar as versões 1.0, 1.2 e 1.3 do protocolo OpenFlow. Ainda, pode-se citar os controladores Trema (SHIMONISHI et al., 2011) (desenvolvido em Ruby) e FlowER (desenvolvido em Erlang) (TRAVELPING, 2014).

Esses controladores vêm com alguns códigos fontes com exemplos que mostram como criar novas aplicações (serviços) de rede para cada abordagem proposta. Um exemplo clássico desse tipo de aplicação é o *Learning Switch*. Normalmente, essas aplicações oferecem serviços de baixo nível e o desenvolvedor de rede é responsável por construir novos serviços de acordo com requisitos específicos. Esses controladores são adequados como ponto de partida para usar o conceito de SDN, porém não são suficientes para alcançar confiabilidade e desempenho para *carrier grade networks demands* (STAESSENS et al., 2011) (TAM, 2007).

O controlador FloodLight foi criado a partir do código-fonte do Beacon e seu foco é o desenvolvimento de um controlador que possa ser utilizado em ambientes comerciais. A versão de código aberto do FloodLight não oferece resiliência ou escalabilidade como na versão comercial, chamado *Big Network Controller*(Big Network Controller, 2013), que é baseado em servidores *cluster* para *High Availability* (HA) e utiliza o núcleo do FloodLight.

Os criadores do NOX, motivados pela incapacidade de satisfazer os requisitos de confiabilidade e escalabilidade, propuseram o Onix, um sistema distribuído sobre a rede de plano de controle e que oferece uma visão global da rede. O Onix define uma API que pode ser utilizada para construção de novos serviços do plano de controle e endereçamento de requisitos. Esse sistema foi a base para o software oferecido pela Nicira (NICIRA, 2013) e sua abordagem é utilizada pelo projeto *Modern SDN Stack* (RESEARCH, 2013).

O projeto *OpenDaylight* (OpenDaylight, 2014) visa à criação de uma arquitetura comum que possa ser explorada pela indústria para a criação de novos serviços inovadores, de acordo com a abstração SDN. Essa arquitetura é composta por várias camadas, sendo uma a *Service Abstraction Layer* (SAL), que pode interagir com diferentes protocolos

que seriam expostos por *plug-ins*. Outra camada é o *Controller Platform* (OpenDaylight, 2014) que controla os elementos de rede, como roteadores e *switches*, e define uma API comum a ser utilizada pelas aplicações da camada superior.

Outro controlador de código aberto que está atualmente em desenvolvimento é o ONOS *(Open Network Operation System)* (ON.LAB, 2013), que visa fornecer uma arquitetura tolerante a falhas, a distribuição de estado em vários controladores e oferecer uma abstração gráfica de alto nível do estado da rede. De acordo com o ON.LAB, essas funcionalidades tornam o ONOS uma boa alternativa para os provedores de serviços e também para as grande operadoras WAN. Um protótipo foi apresentado pelo OnLab no *Open Networking Summit* (ONS) em 2013 e 2014, indicando que ONOS ainda está no caminho para atingir seus objetivos (BERDE, 2014).

Diversos controladores, entre eles o Onix, não estão disponíveis como código aberto e, portanto, a comunidade SDN não é capaz de executar experimentos utilizando essas soluções privadas. O projeto *OpenDayLight* e também o ONOS possuem um caminho à frente.

A Tabela 2 apresenta um resumo das características dos principais controladores, na qual é possível observar a similaridade entre as funcionalidades desses controladores.

Tabela 2 – Características dos controladores OpenFlow.

Controlador	**API**	***OpenSource***	**Linguagem**	**Controle distribuído**
OpenDaylight	Sim	Sim	Java	Sim
ONOS	Sim	Sim	Java	Sim
Onix	Sim	Sim	C++	Sim
NOX	Sim	Sim	C++	Não
POX	Sim	Sim	Python	Não
Floodlight	Sim	Sim	Java	Não
Big Network Controller	Sim	Não	Java	Não

2.2.1 Controlador ONOS

Os controladores SDN, em geral, são apenas para prova de conceito, mostrando que é possível construir uma rede SDN (FERREIRA et al., 2014a). Nesse cenário, o controlador *Open Network Operating System* (ONOS) surge como uma opção de controlador para tais redes, pois atende a critérios que o caracterizem como *carrier grade*. Entre esses critérios estão alta disponibilidade, confiabilidade, tolerância a falhas e desempenho.

ONOS é um projeto desenvolvido pela equipe do ON.Lab, uma organização sem fins lucrativos especializada em tecnologias SDN e OpenFlow. A organização é apoiada por grandes nomes no mercado de telecomunicações e computação, tais como AT&T, Cisco, Ciena, Fujitsu, Huawei, Intel, NEC e a ONF (ONOS members, 2015). O projeto do controlador é pautado por algumas metas, tais como: liberar o desenvolvedor da aplicação de conhecer detalhes intrínsecos do hardware proprietário; libertar o operador de rede de

interfaces e protocolos proprietários e permitir a evolução de componentes de hardware e software, i.e., cada um poderá seguir a linha do tempo de mercado de forma independente.

A figura 4 ilustra como ONOS é dividido arquiteturalmente, assim como a abstração das interfaces *Southbound* e *Northbound* e demais características enumeradas a seguir:

- *Distributed Core*: é a principal camada da arquitetura responsável pelo gerenciamento dos recursos, do estado da rede como um todo e pela execução distribuída do sistema, garantindo a característica de *carrier grade* ao plano de controle ao ser executado como um *cluster*, permitindo de forma rápida atender às necessidades do plano de controle das operadoras de telecomunicações;

- *Northbound abstraction*/APIs: permite o desenvolvimento de aplicações que usem o ONOS como plataforma, com grau liberdade que exime os desenvolvedores de conhecer detalhes topológicos, como por exemplo obter grafo da topologia da rede – O *Intent FrameWork* permite às aplicações solicitar serviços sem conhecer detalhes subjacentes (ONOS, 2015);

- *Southbound abstraction*/APIs: camada composta de módulos que realizam o controle e acesso diretos a cada dispositivo, por meio de API que abstrai cada dispositivo, permitindo que todos sejam usados da mesma forma nas camadas superiores, possibilitando, assim, a interação com diversos dispositivos, sobretudo com cada qual operando com um protocolo diferente — por exemplo: OpenFlow e NetConf – e permitindo a criação de interfaces para um serviço definido, como integridade arquitetural, coerência entre os módulos, mudanças e customização de componentes (ONOS, 2015);

- *Software Modularity*: torna mais fácil o desenvolvimento, depuração, manutenção e atualização do ONOS, bem como os programas desenvolvidos pela comunidade e fornecedores.

Pode-se destacar o *Distributed Core*, em que o ONOS pode ser implementado como um *cluster* de servidores e cada instância é executada em um servidor. Utilizando a biblioteca *Hazelcast* para gerenciar as instâncias do *cluster*, permite-se *failover* transparentemente para os equipamentos (CHAN, 2015). *Hazelcast* é um sistema *open source* projetado para fornecer computação distribuída e paralela nos acessos aos dados armazenados na memória de várias máquinas (HAZELCAST, 2015).

Os operadores de rede podem agregar novos servidores ao *cluster* sem interrupção dos serviços, conforme necessário à capacidade do plano de controle (ONOS ON.LAB, 2015). As instâncias do ONOS trabalham simultaneamente, criando virtualmente uma plataforma única. O núcleo distribuído faz o trabalho pesado para oferecer essas capacidades. Para as aplicações e os dispositivos de rede é transparente se há uma única ou com várias instâncias do ONOS (ONOS ON.LAB, 2015), sempre parecerá que há um único servidor.

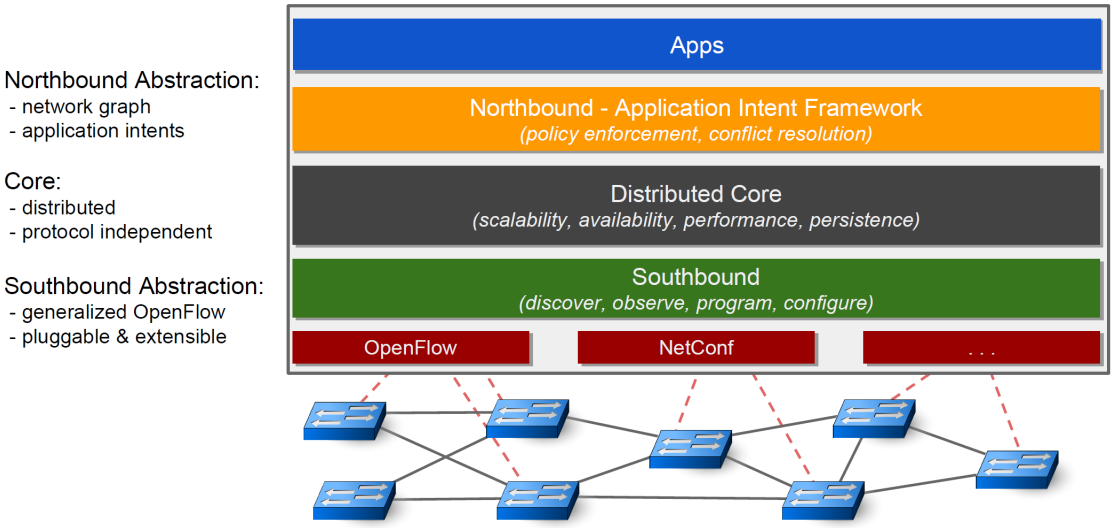

Figura 4 – Camadas da Arquitetura ONOS (ONOS, 2015)

Esta característica torna o ONOS escalável, permitindo se adequar às necessidades de processamento.

A camada *Distributed Core* se baseia em serviços de mensagens para a eleição do líder e para o gerenciamento de estados das instâncias. Desse modo, múltiplas instâncias se comportam como uma única entidade lógica, por meio de mensagens no modelo *publish/-subscribe*. O ONOS implementa um protocolo de recuperação para tratar as mensagens (de atualizações) perdidas devido a eventuais falhas de instâncias.

As mudanças de estados das instâncias são gerenciadas utilizando-se vários serviços, um para cada estado. O serviço de eleição do líder garante que haverá apenas uma instância *Master* entre os *switches* que compõem a HA. Juntos, os serviços de gerenciamento de estados e os mecanismos de eleição do líder garantem alto *throughput*, baixa latência e alta disponibilidade (ONOS ON.LAB, 2015).

2.2.2 Controlador Opendaylight

O controlador OpenDayLight (OPENDAYLIGHT, 2015), também referenciado como ODL, é um projeto colaborativo *open source* realizado pela *Linux Foundation* e seu principal objetivo é acelerar a adoção do SDN. O ODL visa ser um controlador de alta disponibilidade, escalável e extensível de forma que trabalhe com dispositivos de diferentes fabricantes (MAYORAL et al., 2014). Os membros do projeto buscam permitir a interoperabilidade entre produtos, estimular a aplicação e fornecer direção para tecnologias SDN (ORTIZ, 2013).

O controlador em seu núcleo consiste em uma plataforma modular e flexível desenvolvida em Java (MEDVED et al., 2014). Ressalte-se que essa tecnologia depende de um grande número de tecnologias de terceiros. Dessa forma, existe uma grande quantidade de

módulos que podem ser dinamicamente plugados de acordo com as necessidades da rede. Também é possível inserir outros serviços e extensões para melhorar a funcionalidade do SDN (FIGUEROLA, 2015).

A primeira versão, Hydrogen, disponibilizada em fevereiro de 2014, consiste de um controlador, de virtualização de capacidades e de alguns *plugins* (OPENDAYLIGHT, 2015). A versão Helium (OPENDAYLIGHT, 2015), lançada em novembro do mesmo ano, tornou-se o ambiente baseado em *Karaf* e gerenciamento de rede *model-driven*. A última versão, Lithium (OPENDAYLIGHT, 2015), oferece suporte a novos *plugins* e protocolos de rede, melhora o suporte à integração com outras tecnologias, como OpenStack Neutron, e suporta novas versões do protocolo OpenFlow (1.0/1.3).

O OpenDaylight inspirou-se originalmente no controlador Beacon, a partir do qual introduziu os conceitos de Modularidade e Componentes, utilizando *Open Service Gateway Interface (OSGi)* (MEDVED et al., 2014; JARRAYA; MADI; DEBBABI, 2014). O controlador ONOS se divide basicamente nas seguintes camadas:

- *The controller platform*: é a camada que possui interface com a *Northbound* e *Southbound*, conforme ilustra a Fig. 5;
- *Northbound applications and services*: A interface da camada *Northbound* oferece serviços de controle e um conjunto de APIs comuns (REST) para que as aplicações possam gerenciar a configuração da rede (infra-estrutura);
- *Southbound plugins and protocols*: esta camada suporta vários protocolos, para gerenciar e controlar a infra-estrutura de rede, por meio de *plugins*, tais como OpenFlow, NETCONF e SNMP.

The Controller Platform se comunica com a infra-estrutura de rede usando *Southbound plugins* e oferece serviços básicos de rede por meio de um conjunto de gerenciadores apresentados na *Base Network Service Functions* conforme pode ser visto na Fig. 5, tais como gerenciamento da topologia, *Switch* gerenciador, coleta de estatísticas, armazenamento de informações sobre *hosts* (*end-to-end*), informações de menor caminho entre dois nós e configuração da rede. Além dos serviços de *core* da rede mencionados, esta camada oferece outros serviços que interagem com os módulos do núcleo do controlador para aspectos específicos (MIRANTIS, 2015).

Uma das contribuições do OpenDayLight, para o mundo SDN, é a camada *Service Abstraction Layer* (SAL), que conecta *plugins* de protocolos para acesso aos módulos *Service Network Function* (OPENDAYLIGHT, 2015).

O OpenDayLight adota *Model-driven Service Abstraction*, uma camada abstrata de serviços, que permite o desenvolvimento de aplicações por meio de descrições de funcionalidades da rede (dispositivos, serviços, políticas), de forma fácil, através de uma ampla variedade de hardware e protocolos *Southbound*. A plataforma já contém alguns *plugins*,

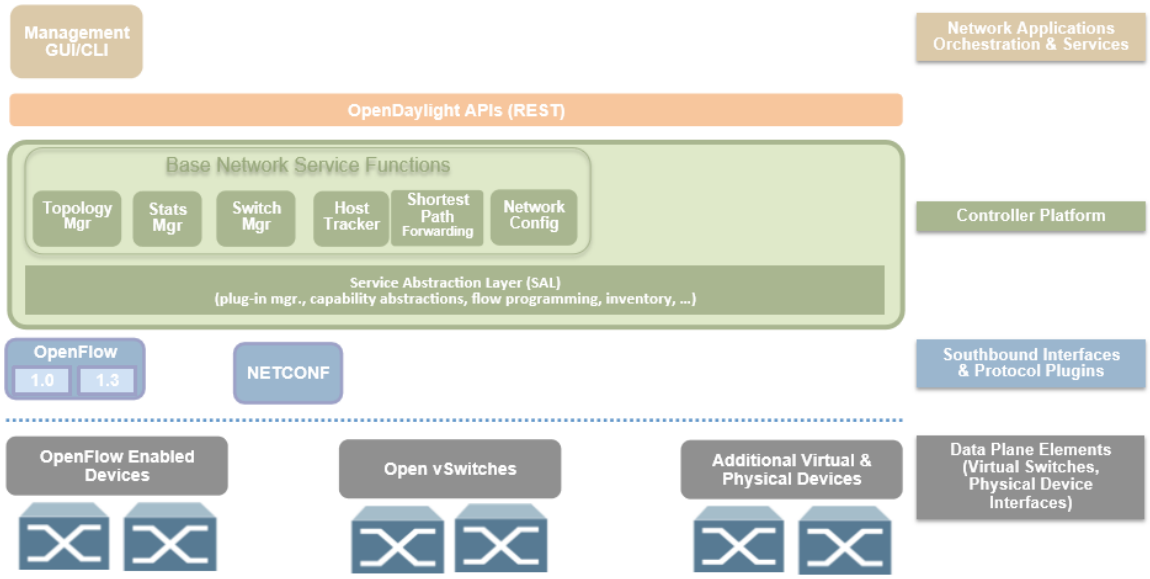

Figura 5 – Camadas OpenDaylight (OPENDAYLIGHT, 2015)

dentre eles, a visualização da topologia da rede e informações de estatísticas (MIRANTIS, 2015).

Destaque-se a facilidade introduzida pela camada SAL, que abstrai o fato de que o OpenDayLight pode ser implementado como um *cluster* de servidores, no qual cada instância é executada em um servidor. É utilizada a biblioteca denominada AKKA, para gerenciar *cluster* (DFARRELL07, 2015), que faz o trabalho, por exemplo, de trocar mensagens e de sincronizar informações (topologia da rede, inventário da elementos, modelos da dados genéricos e *failover*). Akka é uma plataforma que implementa o modelo de atores para criação de sistemas concorrentes, tolerantes a falhas e de alta escalabilidade em Scala ou Java (AKKA, 2015). Akka é uma plataforma que implementa o modelo de atores para criação de sistemas concorrentes, tolerantes a falhas e de alta escalabilidade em Scala ou Java (AKKA, 2015).

Para oferecer a visão de uma plataforma única, um membro do *cluster* é eleito como líder (*Master*), sendo que somente o *Master* poderá responder às requisições. Para a eleição do líder, dentro da camada SAL, foi utilizado o algoritmo de consenso *RAFT* (ONGARO, 2015; DFARRELL07, 2015). Apenas o *Master* possui privilégios de leitura e escrita e os outros membros do *cluster* possuem apenas permissão de leitura (DFARRELL07, 2015). Essas características fazem com que o OpenDayLight seja escalável. Assim como no caso do ONOS, esses mecanismos de distribuição habilitam alto *throughput*, baixa latência e alta disponibilidade.

2.3 Ferramentas para Testes

Serão apresentadas nesta seção as principais ferramentas de análise de desempenho *Benchmarking* de controladores OpenFlow.

2.3.1 TestOn

TestON é um *framework* criado pela Paxterra (PAXTERRA, 2013), em parceria com a ON.LAB (ON.LAB, 2013), e foi projetado para automatizar testes de elementos OpenFlow. Além de fazer testes exaustivos. Essa solução oferece facilidades para o *debugging* de *scripts* criado por desenvolvedores.

O *framework* TestOn foi desenvolvido sobre uma camada de *drivers* que conecta vários componentes, tais como controladores (por exemplo, ONOS e OpenDaylight) e o Mininet (MININET, 2009), e pode referenciar tanto *Internet Protocol* (IP) locais quanto remotos. Ele gerencia esses componentes através de *scripts* desenvolvidos na linguagem Python (ONOS, 2013).

O TestOn disponibiliza um conjunto de diferentes Cenários de Testes. Cada Cenário de Teste pode conter um ou mais Casos de Teste, sendo que para cada caso é oferecido o *status* da execução (*(Pass/Fail)*). Em alguns cenários, os Casos de Teste podem ser executados isoladamente.

Testes mais complexos podem ser feitos com diversos cenários de teste em série e, ao final da execução global, retornar uma porcentagem de *(Pass/Fail)* para todos os testes. Certas combinações de Casos de Teste, que fazem mais sentido em determinados cenários, podem ser especificadas pelo desenvolvedor. Por exemplo, uma sequência lógica de três Casos de testes poderia ser: 1) buscar a topologia da rede no controlador; 2) reiniciar o controlador *Master* do *cluster*; 3) buscar a topologia da rede no controlador e comparar com o estado anterior à reinicialização (ONOS, 2013).

Representada na figura 6, a arquitetura do *framework* TestOn é divida em *core*, *Test Suite*, *drivers*, conforme explanado a seguir:

- TestON/*Core*: onde é feito o *parse*, a execução do testes (incluindo a inicialização dos componentes de testes) e a execução da impressão (*logging*), sendo que o *Core* exceções são lançadas para serem tratadas pela aplicação (Caso de Teste) que a invocou;
- *Drivers*: é um conjunto de *scripts* Python que possui um rol de funções específicas para o componente de teste (eg. Mininet1 ou ONOScli1) e cada função executa a funcionalidade de um único componente (ou alguma variação) – por exemplo, o *driver* Mininet executa o comando pingall na instância e lança todas as possíveis exceções que podem acontecer; e
- *Test Suite*: especifica o conjunto de teste e apresenta a seguinte estruturação:

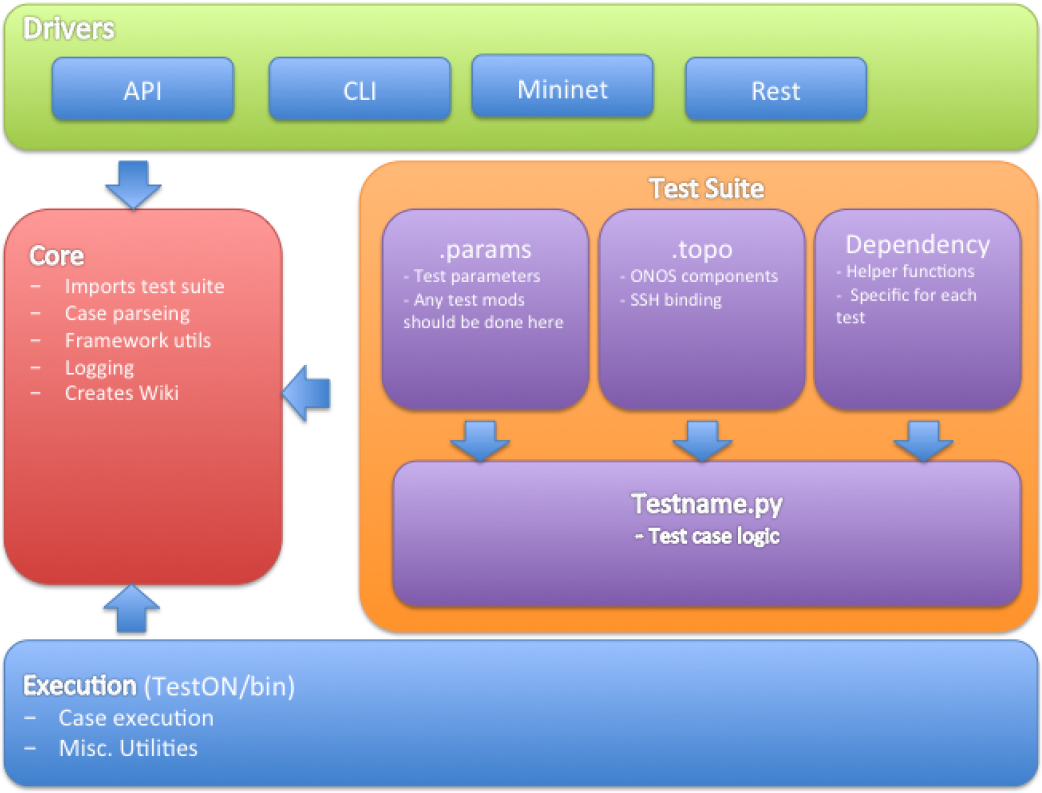

Figura 6 – Arquitetura TestOn (ONOS, 2013)

– Topologia (.topo): define todos os componentes e opções que serão executadas pelo TestOn – por exemplo, o arquivo de topo inclui as informações de login e parâmetros para a máquina onde o Mininet está instalado, sendo que no início do teste, Teston se conectará em todos os componentes especificados;

– Parâmetros (.params): é um arquivo comum personalizado, no qual as variáveis são especificadas de forma a modularizar os casos de teste; e,

– Python (.py): lugar onde todos os casos de testes são especificados, contendo chamadas de função implementadas pelos receptivos *drivers*, no qual a progressão de eventos produz resultados *(Pass/Fail)*.

2.3.2 Cbench

Cbench é uma ferramenta de *benchmarking* utilizada para medir diferentes aspectos de um controlador OpenFlow. Cbench permite definir uma quantidade de pacotes a ser enviados ao controlador e coletar os tempos de respostas apresentando os resultados na forma: tempo mínimo; tempo máximo; e média de tempos de respostas (CBENCH, 2013).

CBench suporta dois modos de operação: modo Latência; e modo *Throughput* (TOOTOONCHIAN; GORBUNOV; GANJALI, 2015; WANG et al., 2014). No modo Latência,

cada *switch* emulado mantém exatamente uma (nova) requisição de *flow*, esperando pela respectiva resposta, antes de solicitar a próxima requisição. O modo Latência mede o tempo de processamento dos controladores OpenFlow em condições de baixa carga. No modo *Throughput*, cada *switch* envia várias requisições, que ficam pendentes no *buffer*, isto é, são enviadas requisições até que ocorra o bloqueio de envio do *buffer* pelo *Transmission Control Protocol* (TCP) local. Assim, o modo *Throughput* mede a quantidade máxima de *flows* que um controlador pode manter. CBench também suporta o modo híbrido, que é uma junção dos dois modos simultaneamente (TOOTOONCHIAN; GORBUNOV; GANJALI, 2015).

Portanto, as métricas de desempenho simplistas do Cbench fornecem uma visão do estado da infraestrutura, sendo que uma análise completa e precisa das capacidades de desempenho e escalabilidade (PERFORMANCE, 2015) requer uma outra ferramenta de teste.

2.4 Trabalhos Correlatos

Lançado inicialmente como uma tecnologia de rede para permitir experimentação em uma rede de campus, o OpenFlow tem grande potencial disruptivo para a concepção de novas aplicações sobre uma rede flexível, para a promoção de inovação e para a redução da complexidade na (re)configuração da infraestrutura de rede em regime operacional. Esse trabalho enfoca o OpenFlow sob a óptica da tolerância à falha, um requisito para redes *carrier-grade*. O termo *Carrier-grade* define o requisito de que a rede deve se recuperar de uma falha dentro de um intervalo especificado. Por exemplo, as redes de telecomunicações trabalham com um intervalo de recuperação de falha de 50 ms. Além disso, redes *carrier-grade* devem apresentar as seguintes propriedades: 1) área de cobertura; 2) tamanho, complexidade, grandes quantidades de informação; 3) soluções proprietárias; 4) altos padrões e requisitos de segurança, estabilidade e eficiência da rede (WANG et al., 2014).

Há muitas pesquisas sobre análise e comparação entre os principais fornecedores da indústria de controladores SDN (JIANG et al., 2014). No entanto, esses controladores não podem gerenciar um grande número de dispositivos e, como resultado, o controlador pode se tornar o gargalo de segurança e desempenho para uma rede baseada em SDN. Por esse motivo, por exemplo, OpenFlow não é largamente utilizados por operadoras de telecomunicações que requerem disponibilidade classe 5 (99,999% de disponibilidade).

O trabalho de (WANG et al., 2014) propôs um sistema de gerenciamento de *cluster* de controladores, a comunicação entre os controladores é feita por uma interface padrão. O gerenciador de *cluster* foi desenvolvido em C, utilizando técnicas de otimização de multi-thread, sendo criado 3 threads para cada conexão com um *switch* (recebimento, processamento e envio) como filas. O controlador foi divido em dois módulos: básicos (topologia, descoberta de conexões, armazenamento, controles da dados e entre outros) e

aplicação (firewall, Web UI, balanceamento de carga e entre outros). Cada controlador, quando iniciado, deve registrar no sistema de gerenciamento de controlador de *cluster*. O sistema de gerenciamento irá atribuir funções (sendo um master e retantes como slave) aos controlador, lendo e analisando suas informações. As informações topológica, *switch* e *hosts* serão sincronizadas entres os controladores. Ambos elementos de gerenciamento do *cluster*, controladores, *switches* e manter uma conexão heartbeat (WANG et al., 2014), indiciando um sinal de conectividade. Por exemplo, caso algumas mensagens enviadas para *switch* não sejam respondidas em um certo tempo, uma nova eleição de controlador master é realizada.

O Controlador NEC ProgrammableFlow (NEC, 2015) oferece alta disponibilidade, colocando os controladores redundantes em *stand-by* ao lado do servidor *Master*. No entanto, a arquitetura ProgrammableFlow não fornece qualquer solução para a escalabilidade do controlador (LEE et al., 2014).

Onix (KOPONEN et al., 2010) criou um *middleware* que trabalha com a conectividade com dispositivos SDN, a distribuição de estados e o balanceamento de carga entre os controladores. No entanto, a arquitetura do Onix não é completamente escalável horizontalmente (LEE et al., 2014).

HyperFlow (TOOTOONCHIAN; GANJALI, 2010) é um controlador OpenFlow que usa o modelo *event-driven* de mensagens *publish/subscribe* para sincronizar o estado da rede entre todos os controladores (compartilhando os eventos de rede). Isso permite que cada controlador tome decisões locais com segurança sem depender de outros controladores.

ElastiCon (DIXIT et al., 2013) pretende ser uma solução eficaz e prática para os requisitos de escalabilidade e disponibilidade, propondo um *pool* de controladores, que aumenta ou diminui de acordo com as condições de tráfego da rede, e fazendo balanceamento de carga de forma dinâmica.

Capítulo 3

Método de Trabalho

Devido ao crescimento das redes de computadores, à evolução para as redes SDN e às novas necessidades não atendidas completamente pelas arquiteturas de redes atuais – em particular a Arquitetura Internet, as operadoras estão em busca de soluções de rede *carrier-grade*, incluindo os controladores da tecnologia OpenFlow. Para alcance do objetivo deste trabalho, serão utilizados dois controladores *open source*, os considerados mais próximos de oferecer tais características à indústria, para se obter uma comparação entre os dados obtidos em laboratório com aqueles disponíveis na Web. Este capítulo apresentará uma visão geral do plano de teste e, em particular, apresentará a aplicação do plano de teste para os controladores ONOS e Opendaylight. Também mostrará a ordem de execução dos cenários de experimentos realizados para cada controlador proposto neste trabalho. Por fim, apresenta os resultados obtidos.

3.1 *Overview* do Processo Experimental

Para a proposta de execução foi definido o seguinte precedimento, que será divido em 4 passos, basicamente:

- Passo 1: serão seguidos os passos definidos pelas equipes de desenvolvimento do ONOS e a do Opendaylight para os cenários de teste;
- Passo 2: serão executados os casos de testes para os controladores ONOS e Opendaylight;
- Passo 3: serão coletados os resultados obtidos por meio dos casos de testes; e,
- Passo 4: será feita análise comparativa dos resultados obtidos experimentalmente com aqueles disponíveis para cada controlador.

3.1.1 Especificação dos Planos de Testes

Para atingir os objetivos do trabalho, planos de testes devem ser especificados de modo a garantir que as análises, sobre os controladores OpenFlow escolhidos, sejam feitas com bases bem definidas. As seções 3.1.2 e 3.1.3 apresentam os Planos de Testes para cada controlador, sendo relativas, respectivamente, aos controladores ONOS e Opendaylight.

O ideal seria que uma mesma sequência de testes fosse aplicada a ambos os controladores. Todavia, os Planos de Testes são desenvolvidos separadamente, pois há funcionalidades cujas implementações requerem passos específicos, como por exemplo o Método Aplicado (*Intents* – para descoberta de melhor caminho disponível) do ONOS, que não existe para o Opendaylight.

Os Planos de Testes foram especificados tendo como base documentos de testes propostos pelos respectivos grupos de desenvolvimento dos controladores, sendo que testes adicionais foram planejados para verificar especificamente as propriedades de um controlador *carrier grade*.

3.1.2 Controlador ONOS

O Plano de Teste apresentado nesta seção se baseia na *wiki* (ON.LAB ONOS CLUSTER, 2015) do projeto ONOS (HaTestMinorityRestart), de onde se extraiu uma lista de 15 dos principais casos de testes, cujo foco recai nos aspectos *carrier grade*. A seguir serão apresentados os casos de testes:

- Caso 1 – Configuração e instalação do controlador ONOS, instanciação da Mininet e sessões *Command Line Interface* (CLI), com os seguintes passos:

  Fazer o Download do pacote de instalação;

  Fazer configurações de *cluster*;

  Iniciar corretamente o controlador ONOS; e,

  Iniciar a Mininet.

- Caso 2: Conectar os *switches* aos respectivos controladores;

- Caso 3: Fazer a descoberta dos hosts via pingall e pré-determinar *Intents* host a host:

  Fazer a instalação da aplicação de encaminhamento pré-ativa;

  Fazer a descoberta do hosts via pingAll;

  Desinstalar a aplicação de encaminhamento pré-ativa; e,

  Adicionar *Intents* para os hosts via CLI.

- Caso 4: Verificar a conectividade entre todos os hosts sobre *Intents*:

  Fazer ping sobre os hosts adicionando *Intents*;

  Checar se todos os *Intents* foram instalados;

  Checar o líder; e,

  Aguardar um minuto e realizar o teste de ping novamente.

- Caso 5: Coletar os dados do estado atual das instâncias de ONOSs e checar a consistência entre os ONOSs e a Mininet:

  Checar se cada *switch* possui um controlador *Master*;

  Buscar o *MasterShip* de cada controlador do *cluster*;

  Checar os papéis (*roles*) de cada controlador;

  Buscar os *Intents* de cada controlador;

  Checar os *Intents* entre os controladores;

  Buscar os *flows* de cada controlador;

  Checar os *flows* entre os controladores;

  Buscar as entradas da OpenFlow *Table*;

  Executar pings contínuos;

  Coletar informações da topologia no ONOS;

  Verificar se um nó ONOS (*ONOS node*) é visto a partir dos outros controladores;

  Verificar se cada host possui um endereço IP;

  Verificar se as visões das informações são consistentes e corretas em todos os controladores;

  Comparar a topologia do ONOS com a da Mininet;

  Verificar se informações dos dispositivos estão corretas;

  Verificar se informações de links estão corretas; e,

  Verificar se os nós ONOSs estão corretos.

- Caso 6: Reiniciar todo o *cluster* do ONOS:

  Matar as instâncias de ONOSs; e,

  Iniciar os ONOSs e verificar se estão funcionando apropriadamente.

- Caso 7: Checar estados das constante do ONOS após falha no plano de controle:

  Checar se cada *switch* possui um controlador *Master*;

  Fazer a leitura dos papéis dos dispositivos do ONOS;

Checar as consistências dos papéis de cada controlador;

Buscar os *Intents* e comparar com todos os nós;

Buscar as OpenFlow *Tables* e comparar com os estados antes de falhas; e,

Verificar se a eleição de líder está funcionando.

- Caso 8: Comparar topologia do ONOS com a da Mininet:

  Coletar informações da topologia no ONOS;

  Verificar se a visão das informações dos hosts é consistente e correta em todos os controladores;

  Verificar se o ponto de conexão do Host na rede está correto;

  Verificar se a visão das informações de *cluster* é consistente e correta em todos os controladores;

  Verificar se as informações dos dispositivos estão corretas;

  Verificar se as informações de links estão corretas;

  Verificas se os nós ONOSs estão corretos.

- Caso 9: Desligar o link entre os *switches* 3 e 28 para checar se a descoberta de topologia está funcionando apropriadamente;

- Caso 10: Religar o link entre os *switches* 3 e 28 para checar se a descoberta de topologia está funcionando apropriadamente;

- Caso 11: Desligar o *switch* 5 para checar se a descoberta de topologia está funcionando apropriadamente;

- Caso 12: Religar o *switch* 5 para checar se a descoberta de topologia está funcionando apropriadamente;

- Caso 13: Fazer a limpeza dos Estados:

  Parar a Mininet; e,

  Checar se o ONOS não possui erros.

- Caso 14: Iniciar o processo de eleição de líder em nós do ONOS; e,

- Caso 15: Verificar se a eleição de líder ainda está funcionando:

  Executar a eleição de líder em cada nó;

  Verificar se cada nó mostra o mesmo líder;

  Encontrar o atual líder e desligá-lo;

  Verificar se há a eleição de um novo líder;

Executar novamente a eleição com o antigo líder funcionando; e,

Verificar se houve alteração do líder (atual), após releição.

3.1.3 Controlador Opendaylight

O Plano de Teste apresentado nesta seção se baseia na *wiki* e em documentos do projeto Opendaylight (Opendaylight Cluster HA, 2014), de onde se extraiu uma lista de 15 dos principais casos de testes, cujo foco recai nos aspectos *carrier grade*. A seguir, são apresentados os casos de testes:

- ❏ Caso 1: Configuração e instalação do controlador Opendaylight, instanciação da Mininet e sessões CLI:

  Fazer o Download do pacote de instalação;

  Fazer configurações de *cluster*;

  Iniciar corretamente o controlador Opendaylight; e,

  Iniciar a Mininet.

- ❏ Caso 2: Conectar os *switches* aos controladores;

- ❏ Caso 3: Fazer a descoberta dos hosts via PingAll;

- ❏ Caso 4: Verificar a conectividade de todos os hosts:

  Fazer ping sobre os hosts;

  Checar se todos os *flows* foram instalados;

  Checar se o líder foi apropriadamente assumido; e,

  Aguardar um minuto e realizar o teste de Ping novamente.

- ❏ Caso 5: Coletar os dados do estado atual das instâncias de Opendaylights e a Mininet:

  Checar se cada *switch* possui um controlador *Master*;

  Buscar o *MasterShip* de cada controlador do *cluster*;

  Checar os papéis (*roles*) de cada controlador;

  Buscar os *flows* de cada controlador;

  Checar os *flows* entre os controladores;

  Buscar as entradas OpenFlow *Table*;

  Executar pings contínuos;

  Coletar informações da topologia no Opendaylight;

Verificar se um nó Opendaylight é visto pelos outros controladores;

Verificar se cada host possui um endereço IP;

Verificar se a visão de informações é consistente e correta em todos os controladores;

Comparar a topologia do Opendaylight com a da Mininet;

Verificar se as informações dos dispositivos estão corretas;

Verificar se informações de links estão corretas; e,

Verificar se os nós Opendaylights estão corretos.

- Caso 6: Reiniciar todo o cluster do Opendaylight:

  Matar as instâncias do Opendaylight; e,

  Iniciar os Opendaylights e verificar se estão funcionando apropriadamente.

- Caso 7: Checar estados das constante do Opendaylight, após falha no plano de controle:

  Checar se cada *switch* possui um controlador *Master*;

  Faz a leitura dos papéis dos dispositivos do Opendaylight;

  Checar as consistências dos papéis de cada controlador;

  Buscar as OpenFlow *Tables* e comparar com os estados antes de falhas; e,

  Verificar se a eleição de Lider esta funcionando.

- Caso 8: Comparar topologia do Opendaylight com a da Mininet;

  Coletar informações da topologia no Opendaylight;

  Verificar se a visão das informações dos hosts é consistente e correta em todos os controladores;

  Verificar o ponto conexão do Host na rede esta correto;

  Verificar se a visão das informações de *cluster* é consistente e correta em todos os controladores;

  Verificar se informações dos dispositivos estão corretas;

  Verificar se informações de links estão corretas; e,

  Verificar se os nós Opendaylight estão corretos.

- Caso 9: Desligar o link entre os *switches* 3 e 28 para checar se a descoberta da topologia está funcionando apropriadamente;

- Caso 10: Religar o link entre os *switches* 3 e 28 para checar se a descoberta de topologia está funcionando apropriadamente;

- ❏ Caso 11: Desligar o *switch* 5 para checar se o descoberta de topologia está funcionando apropriadamente;
- ❏ Caso 12: Religar o *switch* 5 para checar se a descoberta de topologia está funcionando apropriadamente;
- ❏ Caso 13: Fazer a limpeza dos Estados e verificar se Opendaylight finalizou sem erros;

  Parar a Mininet;

  Checar se o Opendaylight não possui erros.
- ❏ Caso 14: Iniciar o processo de eleição de Líder em nós do Opendaylight; e,
- ❏ Caso 15: Verificar se a eleição de líder ainda esta funcionando;

  Executar a eleição de líder em cada nó;

  Verificar se cada nó mostra o mesmo líder;

  Encontrar o atual líder e desligá-lo;

  Verificar se há a eleição de um novo líder;

  Executar novamente a eleição com o antigo líder funcionando; e,

  Verificar se houve alteração do líder (atual), após releição.

Observe-se que os Planos de Testes foram especificados a partir de casos e, eventualmente, ações em cada caso, de tal modo que uma parte significativa deles basta verificar se 'passou' ou 'não passou' naquele(a) caso (ação). Isso é interessante, pois facilita para novos pesquisadores e desenvolvedores saberem se o ambiente está funcionando apropriadamente.

3.2 Ambiente Experimental

É importante lembrar que uma parte dos testes tem o objetivo de comparar os resultados obtidos nos laboratórios da Faculdade de Computação da Universidade Federal de Uberlândia (FACOM) com aqueles apresentados nos sites oficiais dos projetos dos controladores escolhidos para esse trabalho. A comparação é importante, pois permitirá verificar que o ponto de partida para os experimentos são consistentes.

Por este motivo, os ambientes experimentais serão configurados para apresentar a mesma infra-estrutura e topologia de testes, tornando-os (ambientes experimentais) os mais próximos possível daqueles descritos pelos ambientes dos controladores escolhidos. O ambiente experimental consistirá do seguinte:

- Cada máquina será configurada com 5 GB de memória RAM, CPU 2 *cores* de 2 GHz e 10GB de disco;
- Em cada máquina será instalado o Sistema Operacional Ubuntu, versão 14.04, contendo o controlador, a Mininet (MININET, 2009) e as ferramentas para construção de redes virtuais;
- Cada instância da máquina representa papel de nós de um controlador, da mesma forma que cada instância representa conjuntos de *switches* conectados aos controladores, conforme descrito abaixo:
  - Sete instâncias de Controladores, sendo uma por máquina e todas as instâncias em *Cluster Master/Slave*; e,
  - Uma instância de Mininet, que representa a rede de clientes.

A Figura 7 apresenta a topologia de rede do ambiente experimental de testes contendo 28 *switches* e 28 nós conectados aos *switches*.

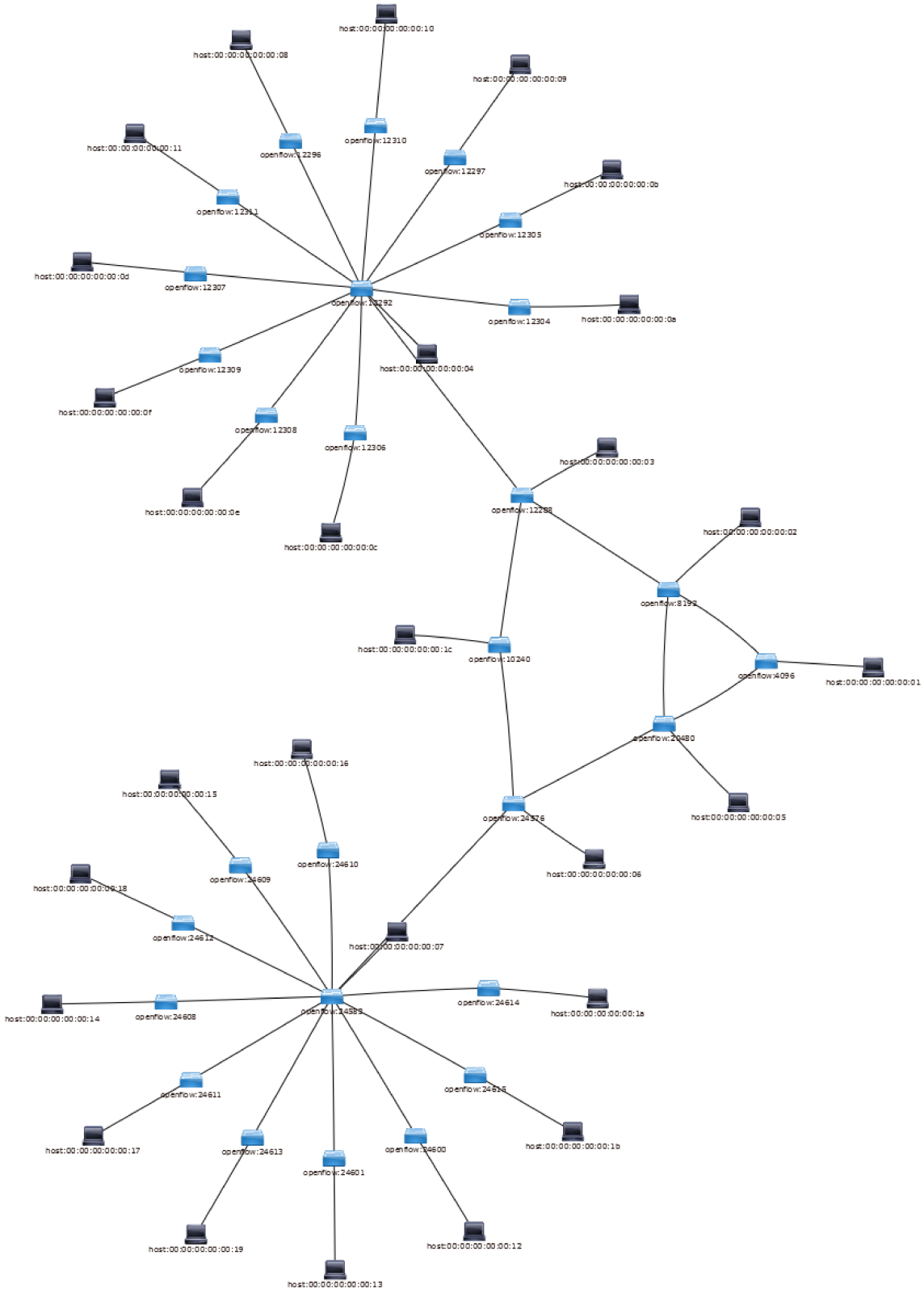

Figura 7 – Topologia da Rede de Testes (*Switches*/Nós)

3.3 Aplicação dos Planos de Testes

Como mencionado na Seção 3.1.1, os planos de testes foram especificados de tal modo que a aplicação do plano é facilmente verificável se 'passou' ou 'não passou' naquele(a) caso (ação). Esta seção apresentará a execução cronológica dos testes e os respectivos resultados dos experimentos para cada um dos controladores ONOS e Opendaylight.

Como pode ser visto nas Tabelas 3 e 4, alguns casos serão aplicados mais de uma vez, como por exemplo o Caso 4 (Teste de conectividade do ambiente) e o Caso 8 (Compara a topologia existente no controlador e no ambiente Mininet), sempre que houver testes de (des)conexão de algum elemento de rede ou de eleição de *Master*.

3.3.1 Testes do Controlador ONOS

Nesta seção, são apresentados os resultados obtidos a partir da aplicação do plano de testes para o Controlador ONOS, contendo o Caso, o *Status* e a Descrição para a situação de cada caso. O plano de teste foi aplicado pela ferramenta TestON e os resultado obtidos são listados em ordem temporal na Tabela 3.

A sequência dos casos de testes da Tabela 3 foi elaborada o mais aproximamente possível da sequência de testes proposta pelos desenvolvedores. Observe-se que esta sequência permite avaliar os cenários de falha e recuperação do controlador. No plano de testes, alguns casos são repetidos a fim de avaliar um cenário de teste, como, por exemplo, os casos 4 e 8, que se repetem diversas vezes na tabela de teste a fim de avaliar a conetividade e a topologia antes e depois de alguns casos específicos.

Houve 5 casos de falhas em um total de 25 casos de testes, sendo que será apresentada uma análise para esses casos, conforme listada a seguir:

- A primeira falha (Caso 2) ocorreu devido ao *cluster* do ONOS não ter conseguido fazer a eleição do líder, antes que os *switches* se conectassem, fazendo com que os nós tomassem decisões variadas (e incorretas) - por exemplo, atribuindo-lhe o papel de *Master* do *switch*;

- A segunda falha (Caso 5) ocorreu devido ao fato de que alguns nós não apresentavam os mesmos *Intents* instalados, apesar do passo anterior (Caso 4) ter executado o pingAll para todos os hosts e feitas as instalações pelo nó *Master*, todavia, provavelmente, o módulo de replicação não conseguiu replicar todas as informações de *Intents* para os nós *Slave*;

- A terceira falha (Caso 7), que ocorre após a simulação de falha em todos os nós do *cluster* no Caso 6, aconteceu por os nós do *cluster* terem reiniciado alguns *switches*, que se conectaram em controladores não *Master* e não conseguiram sincronizar as informações de *intents*, similarmente aos Casos 2 e 5;

Caso	***Status***	**Descrição**
Caso 1	PASSOU	Inicialização do ambiente ONOS
Caso 2	FALHOU	Os *switches* foram conectadas aos nós do ONOS, mas alguns nós não conseguiram atribuir o controlador *Master*
Caso 8	PASSOU	Topologia do ONOS e da Mininet são idênticas
Caso 3	PASSOU	Descoberta de hosts via PingAll e *Intents* pre-determinados
Caso 4	PASSOU	Foram instalados todos os *Intents* e o ping funcionou
Caso 5	FALHOU	Ocorreu falha na consistência de *Intents* de alguns controladores
Caso 14	PASSOU	O processo de releição de líder funcionou
Caso 6	PASSOU	O *restart* das instâncias do ONOS funcionou
Caso 8	PASSOU	Topologia do ONOS e da Mininet são idênticas
Caso 7	FALHOU	Alguns *switches* tiveram o controlador *Master* alterado e, também, os controladores tiveram diferentes visões de *Intents*
Caso 4	PASSOU	Ocorreu com sucesso o teste de conectividade entre os hosts
Caso 15	FALHOU	O processo de reeleição falhou na tentativa de encontrar o líder atual para alguns nós
Caso 9	PASSOU	O módulo de descoberta identificou a queda no link entre os *switches* 3 e 28
Caso 8	PASSOU	Topologia do ONOS e da Mininet são idênticas
Caso 4	PASSOU	Ocorreu com sucesso o teste de conectividade entre os hosts
Caso 10	PASSOU	O módulo de descoberta identificou a conexão entre os *switches* 3 e 28
Caso 8	PASSOU	Topologia do ONOS e da Mininet são idênticas
Caso 4	PASSOU	Ocorreu com sucesso o teste de conectividade entre os hosts
Caso 11	PASSOU	O módulo de descoberta identificou a queda do *switch* 5
Caso 8	PASSOU	Topologia do ONOS e da Mininet são idênticas
Caso 4	PASSOU	Ocorreu com sucesso o teste de conectividade entre os hosts
Caso 12	PASSOU	O módulo de descoberta identificou o *switch* 5 juntamente com seus hosts
Caso 8	FALHOU	Alguns nós ONOS mostraram topologias diferentes da Mininet
Caso 4	PASSOU	Ocorreu com sucesso o teste de conectividade entre os hosts
Caso 13	PASSOU	A Mininet foi finalizada e o ONOS não apresentou falha grave

Tabela 3 – Aplicação do Plano de Testes - Controlador ONOS

- A quarta falha (Caso 15) aconteceu na tentativa da reeleição de líder, uma vez que não conseguiu encontrar o líder atual, e a falha se deve a informações que não foram sincronizadas após reinicialização de todo o *cluster*; e,
- A quinta falha (Caso 8) aconteceu por uma discrepância na topologia da Mininet com aquela dos nós do ONOS, onde alguns nós tiveram diferentes visões de topologia comparado com a da Mininet após o teste de desligar e ligar o *switch* 5.

3.3.2 Testes do controlador OpendayLight

Nesta seção são apresentados os resultados obtidos a partir da aplicação do plano de testes para o Controlador Opendaylight, contendo o Caso, o *Status* e a Descrição para a situação de cada caso.

Para este controlador não foi encontrada nenhuma ferramenta que faça a execução e verificação automática de um cenário multi-*controller*. Deste modo, os resultados foram obtidos manualmente para cada caso listado na Tabela 4.

Caso	***Status***	**Descrição**
Caso 1	PASSOU	Inicialização do ambiente OpendayLight
Caso 2	PASSOU	Os *switches* foram conectados aos nós do OpendayLight e conseguiram atribuir o controlador *Master* a todos
Caso 8	PASSOU	Topologia do OpendayLight e da Mininet são idênticas
Caso 3	PASSOU	Descoberta de todos os hosts via PingAll
Caso 4	PASSOU	Teste de conectividade realizado com sucesso entre os hosts
Caso 5	PASSOU	Estão consistentes os dados entre o OpendayLight e a Mininet
Caso 14	PASSOU	O processo de releição de líder funcionou
Caso 6	PASSOU	O *restart* das instâncias do OpendayLight funcionou
Caso 8	PASSOU	Topologia do OpendayLight e da Mininet são idênticas
Caso 7	FALHOU	Alguns *switches* apresentam controladores *Master* diferentes
Caso 4	PASSOU	Ocorreu com sucesso o teste de conectividade entre os hosts
Caso 15	PASSOU	Processo de reeleição executado
Caso 9	PASSOU	O módulo de descoberta identificou a queda no link entre os *switches* 3 e 28
Caso 8	PASSOU	Topologia do OpendayLight e da Mininet são idênticas
Caso 4	PASSOU	Ocorreu com sucesso o teste de conectividade entre os hosts
Caso 10	PASSOU	O módulo de descoberta identificou a conexão entre os *switches* 3 e 28
Caso 8	PASSOU	Topologia do OpendayLight e da Mininet são idênticas
Caso 4	PASSOU	Ocorreu com sucesso o teste de conectividade entre os hosts
Caso 11	PASSOU	O módulo de descoberta identificou a queda do *switch* 5
Caso 8	PASSOU	Topologia do OpendayLight e da Mininet são idênticas
Caso 4	PASSOU	Ocorreu com sucesso o teste de conectividade entre os hosts
Caso 12	PASSOU	O módulo de descoberta identificou o *switch* 5 juntamente com seus hosts
Caso 8	PASSOU	Topologia do OpendayLight e da Mininet são idênticas
Caso 4	PASSOU	Ocorreu com sucesso o teste de conectividade entre os hosts
Caso 13	FALHOU	A Mininet foi finalizada e o OpendayLight apresentou erros de sincronização entre os nós do *cluster*

Tabela 4 – Aplicação do Plano de Testes - Controlador OpendayLight

Assim como nos casos de testes do controlador ONOS, a sequência dos casos para o controlador OpendayLight foi elaborada o mais proximamente possível da sequência de testes proposta pelos desenvolvedores, com as mesmas finalidades, sendo que, neste caso,

a sequência foi executada manualmente.

É interessante notar que a execução dos testes listados na Tabela 4 apresentou apenas 2 casos de falhas em um total de 25. A seguir, é feita uma análise para os dois casos:

- A primeira falha (Caso 7) acontece pelos mesmos motivos pelos quais ela ocorreu na sequência de testes do controlador ONOS, excetuando-se os aspectos relativos aos *Intents*, que não se aplicam ao controlado OpendayLight, i.e., houve uma simulação de falhas em todos os nós do *cluster* (Caso 6) e, quando os nós do *cluster* reiniciaram, alguns *switches* se conectaram em controladores não *Master*, que somente aconteceu porque o processo de eleição do líder não terminara ainda;
- A segunda falha (Caso 13) ocorreu durante a finalização da Mininet, os logs indicavam a presença de erros dos nós na tentativa de sincronizar as informações da topologia.

Cabe, nesse ponto, uma análise que se aplica comparando-se os resultados obtidos nas duas sequências de testes. Ressalte-se o Caso 2, para cada um dos controladores, no qual o ONOS apresenta falha e o OpenDaylight apresenta sucesso. Como a sequência de testes para o ONOS foi automatizada pela ferramenta TestON, é possível que o TestON tenha se conectado nas instâncias do ONOS antes que terminasse o processo de sincronismo e eleição do *Master* para o *cluster*. Para o OpendayLight isso provavelmente não aconteceu devido ao processo de execução e validação ter sido feito manualmente; então, houve tempo suficiente.

Para o Caso 7, que falhou em ambos os controladores, observa-se que quando ocorre a falha geral dos nós (de forma sequencial ou aleatória), o sincronismo e a eleição de líder começam a não funcionar corretamente, mas quando a falha acontece apenas em uma minoria de nós, o processo funciona normalmente.

3.4 Análise Comparativa Envolvendo Dados Disponíveis

Embora tenham sido escolhidos os controladores ONOS e OpendayLight, cujos testes realizados em laboratórios foram apresentados nas seções 3.1.2 e 3.1.3, respectivamente, o controlador OpendayLight disponibiliza em seu site apenas alguns testes simplistas ((opendaylight Test MD SAL, 2014)), não havendo informações suficientes para uma análise pormenorizada.

Para o controlador ONOS, foi feita uma equivalência dos casos de testes realizados em laboratório, com aqueles disponibilizados em seu site ((ON.LAB ONOS CLUSTER, 2015)). A Tabela 5 apresenta os vinte e cinco casos de testes (caso a caso) e em duas colunas os resultados obtidos em laboratório (coluna Testes em Laboratório) e os resultados

disponibilizados pelo projeto ONOS (coluna Testes Disponíveis). Como já apresentado anteriormente, os testes em laboratório apresentaram falhas em cinco casos de testes. Nessa Tabela 5 é possível perceber que foram oito falhas nos testes desempenhados pelo projeto ONOS.

Caso	***Teste em Laboratório***	**Testes Disponíveis**
Caso 1	PASSOU	PASSOU
Caso 2	FALHOU	PASSOU
Caso 8	PASSOU	PASSOU
Caso 3	PASSOU	FALHOU
Caso 4	PASSOU	PASSOU
Caso 5	FALHOU	FALHOU
Caso 14	PASSOU	PASSOU
Caso 6	PASSOU	PASSOU
Caso 8	PASSOU	PASSOU
Caso 7	FALHOU	FALHOU
Caso 4	PASSOU	FALHOU
Caso 15	FALHOU	PASSOU
Caso 9	PASSOU	PASSOU
Caso 8	PASSOU	PASSOU
Caso 4	PASSOU	FALHOU
Caso 10	PASSOU	PASSOU
Caso 8	PASSOU	PASSOU
Caso 4	PASSOU	FALHOU
Caso 11	PASSOU	PASSOU
Caso 8	PASSOU	PASSOU
Caso 4	PASSOU	FALHOU
Caso 12	PASSOU	PASSOU
Caso 8	FALHOU	PASSOU
Caso 4	PASSOU	FALHOU
Caso 13	PASSOU	PASSOU

Tabela 5 – Comparação de Resultados: Testes em Laboratório *versus* Disponíveis pelo ONOS

Os casos de testes disponíveis pelo ONOS que mais apresentaram falhas se referem a conectividade entre *hosts* relativamente a *Intents*, sendo seis casos de um total de vinte e cinco casos. Infelizmente, o ONOS não informa quais foram as configurações do ambiente utilizadas em seus experimentos, não permitindo uma análise mais detalhada sobre os *Intents* que falharam. Os outros dois casos que falharam, oscasos cinco e sete, são referentes à sincronização de estado entre os nós do *cluster*, conforme ocorrera nos experimentos em laboratório.

3.5 Dificuldades encontradas

Para a avaliação dos controladores houve diversas dificuldades. A primeira consiste na pouca documentação sobre as configurações dos *clusters*, como o OpendayLight (opendaylight Running HA, 2014) e o ONOS (ONOS Running Environment, 2014). Foram necessárias diversas tentativas, procuras por documentações complementares sobre bibliotecas utilizadas pelos controladores, ressaltando-se que houve necessidade de customizações, até que se pudesse chegar a uma configuração adequada para os testes.

Outro ponto que merece destaque é a dificuldade em encontrar uma ferramenta de *benchmark* capaz de fazer avaliação multi-controladores. Assim como foi apresentado na seção 2.3, existe uma gama variada de ferramentas, porém nenhuma delas é capaz de realizar a cobertura completa de testes para os controladores escolhidos.

TestON (PAXTERRA, 2013) foi a ferramenta que mais se mostrou promissora, sendo que atualmente ela possui módulos de testes multi-controladores escritos apenas para o ONOS. Para o Opendaylight, até onde esta pesquisa conseguiu evoluir, não foi encontrada ferramenta capaz de analisar multi-controladores. Há diversos trabalhos que apresentam como executar apenas uma instância (Opendaylight Performance, 2014).

Desse modo, para o controlador ONOS se utilizou a ferramente TestON e para o controlador Opendaylight, os testes foram reproduzidos de forma manual.

Capítulo 4

Conclusão e Trabalhos Futuros

Nos últimos anos, SDN deixou de ser uma tendência e, com sua materialização por meio do OpenFlow, passou a ser uma alternativa para o desenvolvimento de novas arquiteturas. A separação dos Planos de Controle e de Dados permite que requisitos de QoS/QoE se tornem realidade nas aplicações futuras. Atualmente, esses requisitos são oferecidos no melhor esforço e a realidade é que muitas aplicações multimídia não funcionam apropriadamente em todo o tempo. Considerando que a infraestrutura das redes de dados é provida, em geral, pelas operadoras de telecomunicações, há que se pensar, então, nas redes no futuro com qualidade *carrier grade*. O termo *carrier grade* é originário da área de telecomunicações, mas, hoje em dia, em diversas áreas é sinônimo de alta segurança, disponibilidade, escalabilidade e qualidade de serviço. O OpenFlow, em particular, deve apresentar uma solução com capacidade *carrier grade* para se tornar uma tecnologia candidata a implementar as redes de dados no futuro.

Há muitas publicações sobre as propriedades de uma gama de controladores OpenFlow, como se pode observar na seção 2.4. Em particular, separam-se na seção 2.2 dois controladores OpenFlow com premissas *carrier grade*.

Nesse sentido, o objetivo principal deste trabalho é realizar uma análise comparativa dos dois principais controladores OpenFlow candidatos a soluções *carrier grade*, ONOS e OpendayLight, tendo como ponto central verificar suas propriedades experimentalmente.

Esse objetivo não é trivial e requer um alto grau de especialização para que seja conquistado, uma vez que são necessários conhecimentos profundos dos requisitos das redes de telecomunicações - em particular, das propriedades *carrier grade*, das características das redes de dados existentes, do protocolo OpenFlow (em todas suas versões), de uma gama variada de soluções de controladores OpenFlow disponíveis (ver seção 2.4) para poder selecionar os controladores a ser testados, de ferramentas de testes para *cluster* de servidores (controladores) e habilidade para teste manuais diretamente em linha de comando. Ressalte-se que, além dos conhecimentos mencionados, fazer o planejamento das instalações, fazer as configurações e colocar em funcionamento são tarefas árduas que, de certo modo, requerem muita pesquisa, conhecimento e experiência.

Para se alcançar o objetivo geral do trabalho, foi necessário estabelecer um Plano de Testes que verificasse as propriedades para cada controlador, criando, então, as bases para a comparação de resultados. A primeira conclusão que se pode chegar após a realização do trabalho é que há diversos pontos que precisam de evolução para que as operadoras adotem OpenFlow como uma solução. Nenhum dos controladores analisados atendem aos requisitos de uma operadora de telecomunicações.

Em relação ao atendimento dos objetivos, o primeiro objetivo introduzido na seção 1.3 foi resolvido na seção 2.2. Esta seção apresentou os principais controladores OpenFlow com o seguinte critério: o propósito pelo qual foi desenvolvido; plataforma/linguagem utilizada; e capacidade de funcionar de forma distribuída.

O segundo objetivo introduzido na seção 1.3 foi atendido por meio da seleção de dois controladores OpenFlow, os mais próximos de atender os requisitos *carrier grade*, sendo eles detalhados nas seções 2.2.1 (Controlador ONOS) e 2.2.2 (Controlador OpendayLight).

O terceiro objetivo introduzido na seção 1.3 visava a criação de Planos de Testes bem definidos para cada controlador, tal como apresentado nas seções 3.1.2 (Plano de Testes para o Controlador ONOS) e 3.1.3 (Plano de Testes para o Controlador OpendayLight) e as respectivas execuções.

O último objetivo proposto na seção 1.3, que propõe a análise comparativa e a análise dos testes individuais, pode ser observado nas seções 3.3.1 (Execução de Testes para o Controlador ONOS) e 3.3.2 (Execução de Testes para o Controlador OpendayLight).

Uma contribuição - que não foi objetivo específico, mas que merece ser frisada - é o conjunto de *scripts* dos Planos de Testes (ver no Apêndice B.1) para a ferramenta TestON, que pode ser utilizado em trabalhos futuros.

É interessante notar que o controlador ONOS mostrou-se maduro, comparado a controladores mais antigos, contando com o grande número de empresas colaborando no projeto. Contudo, há melhorias a serem feitas, uma vez que os testes (total de 25 casos) mostraram instabilidade em relação ao sincronismo de informações das aplicações entre os nós, apresentando cinco falhas.

O controlador OpendayLight é um pouco mais antigo que o controlador ONOS e, também, conta com diversas empresas apoiando o projeto, sendo que algumas delas colaboram em ambos os projetos. Para o OpendayLight, os resultados foram mais satisfatórios, praticamente não apresentando problemas de sincronismo. Todavia, quando foi forçada uma falha geral envolvendo todos os nós, ambos os controladores não foram capazes de estabelecer comunicação para a eleição do *Master*, causando perdas de informações e impedindo o alinhamento dos nós.

4.1 Trabalhos Futuros

Os controladores analisados apresentaram ainda muitos aspectos a ser evoluídos. Espera-se, então, que ocorra a maturação nos controladores até um estágio que viabilize a sua implantação nas operadoras. Para isso, trabalhos futuros poderão se ater a aspectos chaves para *carrier grade*, tais como alta disponibilidade, escalabilidade, gerenciamento, latência, *throughput*, segurança, *logging*/auditoria etc.

Há espaço para a construção de um controlador OpenFlow com propriedades *carrier grade* que utilize equipamentos, padrões e camadas de softwares já utilizados pelas operadoras, tais como SLEE (FERREIRA et al., 2014b; FERRY, 2014; MOBICENTS, 2015). Acredita-se que a construção de um controlador OpenFlow nessa abordagem facilitará sua adoção pelas operadoras.

4.2 Contribuições em Produção Bibliográfica

Ao longo do trabalho foram explorados temas relacionados a redes de computadores, telecomunicações, SDN e Internet do Futuro, com a proposição de uma nova arquitetura *clean slate*. Concluiu-se o Projeto de Pesquisa Internacional FP7/OFELIA/EDOBRA (*ExtenDing and Deploying Ofelia in BRAzil*), concluindo todos os seus *deliverables*. Diversos artigos foram publicados em congressos Qualis classificados no índice restrito, sendo:

- *Towards a Carrier Grade SDN Controller: Integrating OpenFlow with Telecom Services*. Publicado no The Tenth Advanced International Conference on Telecommunications (AICT), 2014, Paris. Este artigo ganhou *Best Paper Award* (FERREIRA et al., 2014b).
- *Cross Layers Semantic Experimentation for Future Internet* – SBRC/WPEIF 2013 – (DIAS et al., 2012).
- *Semantically Enriched Services to Understand the Need of Entities* – LNCS e FIA 2012 – (SILVA et al., 2012).

Além dos acima citados, os artigos *Enabling a Carrier Grade SDN by Using a Top-Down Approach* e *Deployment and Experimentation of the Entity Title Architecture at OFELIA Testbed: Learned Lessons Revealed* foram apresentados respectivamente em 2014 e 2015 no Workshop de Pesquisa Experimental da Internet do Futuro (WPEIF), Workshop que faz parte do SBRC (Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos).

Referências

AGUIAR, R. L. Some comments on hourglasses. **SIGCOMM Comput. Commun. Rev.**, v. 38, n. 5, p. 69–72, set. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1452335.1452346>.

AKKA. **Build powerful concurrent and distributed applications**. 2015. Disponível em: <http://akka.io/>.

BARAN, P. On distributed communications networks. **IEEE Transactions on Communications Systems**, v. 12, n. 1, p. 1–9, mar. 1964. ISSN 0096-1965.

BERDE, P. **ONOS at ONS 2014**. 2014. Disponível em: <http://www.slideshare.net/ON_LAB/onos-at-ons-2014>.

Big Network Controller. **Big Network Controller**. 2013. Disponível em: <http://www.bigswitch.com/products/SDN-Controller>.

CAI, Z. **Maestro**. 2014. Disponível em: <http://code.google.com/p/maestro-platform/>.

CBENCH. **Cbench - Scalable Cluster Benchmarking**. 2013. Disponível em: <http://sourceforge.net/projects/cbench/>.

CHAN, C. **Distributed ONOS**. 2015. Disponível em: <https://wiki.onosproject.org/display/ONOS/Distributed+ONOS>.

CLARK, D. et al. **Towards the Future Internet Architecture**. IETF, 1991. RFC 1287 (Informational). (Request for Comments, 1287). Disponível em: <http://www.ietf.org/rfc/rfc1287.txt>.

DFARRELL07. **Architecture:Clustering**. 2015. Disponível em: <https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:Architecture:Clustering>.

DIAS, A. et al. Cross layers semantic experimentation for future internet. In: **XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. Anais do III Workshop de Pesquisa Experimental na Internet do Futuro (WPEIF)**. Ouro Preto, Brasil: Sociedade Brasileira de Computação, 2012. p. 16–19. ISBN 2177-496X.

DIXIT, A. et al. Towards an elastic distributed sdn controller. In: ACM. **ACM SIGCOMM Computer Communication Review**. [S.l.], 2013. v. 43, n. 4, p. 7–12.

EGEVANG, K.; FRANCIS, P. **The IP Network Address Translator (NAT)**. IETF, 1994. RFC 1631 (Informational). (Request for Comments, 1631). Obsoleted by RFC 3022. Disponível em: <http://www.ietf.org/rfc/rfc1631.txt>.

ERICKSON, D. **Beacon**. 2014. Disponível em: <https://openflow.stanford.edu/display/Beacon/Home>.

FARHADY, H.; LEE, H.; NAKAO, A. Software-defined networking. **Comput. Netw.**, Elsevier North-Holland, Inc., New York, NY, USA, v. 81, n. C, p. 79–95, abr. 2015. ISSN 1389-1286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2015.02.014>.

FERREIRA, C. et al. Towards a Carrier Grade SDN Controller: Integrating OpenFlow With Telecom Services. In: . [s.n.], 2014. p. 70–75. ISBN 978-1-61208-360-5. Disponível em: <http://www.thinkmind.org/index.php?view=article&articleid=aict_2014_3_40_10162>.

_____. Towards a carrier grade sdn controller: Integrating openflow with telecom services. 2014.

FERRY, D. **JAIN SLEE (JSLEE) 1.1 Specification, Final Release**. 2014. Disponível em: <http://www.jcp.org/en/jsr/detail?id=240>.

FIGUEROLA, A. S. Opendaylight as a controller for software defined networking. Universitat Politècnica de Catalunya, ago. 2015.

GOTH, G. Software-defined networking could shake up more than packets. **IEEE Internet Computing**, v. 15, n. 4, p. 6–9, ago. 2011. ISSN 1089-7801.

GREENE, K. TR10: software-defined networking. **MIT Technology Review**, v. 112, n. 2, abr. 2009. Disponível em: <http://www.technologyreview.com/web/22120/>.

GROUP, I. E. S.; HINDEN, R. **Applicability Statement for the Implementation of Classless Inter-Domain Routing (CIDR)**. IETF, 1993. RFC 1517 (Historic). (Request for Comments, 1517). Disponível em: <http://www.ietf.org/rfc/rfc1517.txt>.

GUDE, N. et al. NOX: towards an operating system for networks. **SIGCOMM Comput. Commun. Rev.**, v. 38, n. 3, p. 105–110, jul. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1384609.1384625>.

HAZELCAST. **hazelcast**. 2015. Disponível em: <https://hazelcast.com/products/>.

JARRAYA, Y.; MADI, T.; DEBBABI, M. A survey and a layered taxonomy of software-defined networking. **Communications Surveys & Tutorials, IEEE**, IEEE, v. 16, n. 4, p. 1955–1980, 2014.

JIANG, G. et al. Survey and quantitative analysis of sdn controllers. **Journal of Frontiers of Computer Science and Technology**, v. 8, n. 6, p. 653–664, 2014.

JOHN, W. et al. Splitarchitecture: SDN for the carrier domain. v. 52, n. 10, p. 146–152, 2013. ISSN 0163-6804.

KOPONEN, T. et al. Onix: A distributed control platform for large-scale production networks. In: **OSDI**. [S.l.: s.n.], 2010. v. 10, p. 1–6.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. **Proceedings of the IEEE**, v. 103, n. 1, p. 14–76, Jan 2015. ISSN 0018-9219.

LEE, B. et al. Iris: The openflow-based recursive sdn controller. In: IEEE. **Advanced Communication Technology (ICACT), 2014 16th International Conference on**. [S.l.], 2014. p. 1227–1231.

MAYORAL, A. et al. Experimental validation of automatic lightpath establishment integrating opendaylight sdn controller and active stateful pce within the adrenaline testbed. In: IEEE. **Transparent Optical Networks (ICTON), 2014 16th International Conference on**. [S.l.], 2014. p. 1–4.

McKeown, N. et al. OpenFlow: Enabling innovation in campus networks. v. 38, n. 2, p. 69–74, 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>.

MEDVED, J. et al. Opendaylight: Towards a model-driven sdn controller architecture. In: IEEE. **2014 IEEE 15th International Symposium on**. [S.l.], 2014. p. 1–6.

MININET. **Emulator for rapid prototyping of Software Defined Networks**. 2009. Disponível em: <http://mininet.org/>.

MIRANTIS. **whats-opendaylight**. 2015. Disponível em: <https://www.mirantis.com/blog/whats-opendaylight/>.

MOBICENTS. **Mobicents JAIN SLEE**. 2015. Http://www.mobicents.org/slee/intro.html. Disponível em: <http://www.mobicents.org/slee/intro.html>.

MOCKAPETRIS, P. **Domain names: Concepts and facilities**. IETF, 1983. RFC 882. (Request for Comments, 882). Obsoleted by RFCs 1034, 1035, updated by RFC 973. Disponível em: <http://www.ietf.org/rfc/rfc882.txt>.

_____. **Domain names: Implementation specification**. IETF, 1983. RFC 883. (Request for Comments, 883). Obsoleted by RFCs 1034, 1035, updated by RFC 973. Disponível em: <http://www.ietf.org/rfc/rfc883.txt>.

NEC. **ProgrammableFlow Controller by NEC**. 2015. Disponível em: <http://www.nec.com/en/global/prod/pflow/controller.html>.

NICIRA. **Nicira**. [s.n.], 2013. Disponível em: <http://nicira.com/>.

NOXREPO. **About NOX**. [s.n.], 2013. Disponível em: <http://www.noxrepo.org/nox/about-nox/>.

NTT Communications. **Ryu SDN Framework**. 2013. Disponível em: <http://osrg.github.io/ryu/>.

ONF SOLUTION BRIEF. **OpenFlow Enabled Mobile and Wireless Networks**. 2013. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-wireless-mobile.pdf>.

ONGARO, J. O. S. U. D. **In Search of an Understandable Consensus Algorith**. 2015. Disponível em: <https://ramcloud.stanford.edu/wiki/download/attachments/11370504/raft.pdf>.

ON.LAB. **ONOS - Open Network Operating System**. 2013. Disponível em: <http://tools.onlab.us/onos.html>.

ON.LAB. **TestON framework**. 2013. Disponível em: <https://wiki.onosproject.org/pages/viewpage.action?pageId=4752804>.

ON.LAB ONOS CLUSTER. **Master - HA Cluster Restart**. 2015. Disponível em: <https://wiki.onosproject.org/display/ONOS12/Master+-+HA+Cluster+Restart>.

ONOS. **Open Network Operating System**. 2015. Disponível em: <http://onosproject.org/>.

ONOS members. **Open Network Operating System member**. 2015. Disponível em: <http://onosproject.org/community/members/>.

ONOS, O. **TestON framework**. 2013. Disponível em: <https://wiki.onosproject.org/pages/viewpage.action?pageId=2133836>.

ONOS ON.LAB. **Introducing ONOS - a SDN network operating system for Service Providers**. 2015. Disponível em: <http://onosproject.org/wp-content/uploads/2014/11/Whitepaper-ONOS-final.pdf>.

ONOS Running Environment. **Test Environment Setup**. 2014. Disponível em: <https://wiki.onosproject.org/display/ONOS/Test+Environment+Setup>.

Open Networking Foundation. **OpenFlow Switch Specification 1.0.0**. 2009. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf>.

Open Networking Foundation - ONF. **Software-Defined Networking: The New Norm for Networks**. 2014. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>.

OpenDaylight. **OpenDaylight Technical Overview**. 2014. Disponível em: <http://www.opendaylight.org/project/technical-overview>.

OPENDAYLIGHT. **OpenDayLightSDNController**. 2015. Disponível em: <https://www.opendaylight.org/>.

Opendaylight Cluster HA. **CSIT Test Plan Cluster HA**. 2014. Disponível em: <https://wiki.opendaylight.org/view/CrossProject:Integration_Group:CSIT_Test_Plan_Cluster_HA>.

Opendaylight Performance. **Integration Group Performance Tests**. 2014. Disponível em: <https://wiki.opendaylight.org/view/CrossProject:Integration_Group:Performance_Tests>.

opendaylight Running HA. **Running and testing an OpenDaylight Cluster**. 2014. Disponível em: <https://wiki.opendaylight.org/view/Running_and_testing_an_OpenDaylight_Cluster>.

opendaylight Test MD SAL. **Clustering Test Plan opendaylight**. 2014. Disponível em: <https://wiki.opendaylight.org/view/MD-SAL_Clustering_Test_Plan>.

opennetworking. **Open Networking Foundation About**. 2015. Disponível em: <https://www.opennetworking.org/about/onf-overview>.

ORTIZ, S. Software-defined networking: On the verge of a breakthrough? **IEEE Computer**, v. 46, n. 7, p. 10–12, 2013.

PAXTERRA. **TestON framework**. 2013. Disponível em: <https://github.com/Paxterra/TestON/wiki>.

PERFORMANCE, O. **Performance Whitepape rBlackbird release technical**. 2015. Disponível em: <http://onosproject.org/wp-content/uploads/2014/11/PerformanceWhitepaperBlackbirdrelease-technical.pdf>.

POSTEL, J. **Domain names plan and schedule**. IETF, 1983. RFC 881. (Request for Comments, 881). Updated by RFC 897. Disponível em: <http://www.ietf.org/rfc/rfc881.txt>.

REKHTER, Y.; LI, T. **A Border Gateway Protocol 4 (BGP-4)**. IETF, 1995. RFC 1771 (Draft Standard). (Request for Comments, 1771). Obsoleted by RFC 4271. Disponível em: <http://www.ietf.org/rfc/rfc1771.txt>.

RESEARCH, O. **Modern SDN Stack Project**. 2013. Disponível em: <http://onrc.stanford.edu/research_modern_sdn_stack.html>.

REXFORD, J.; DOVROLIS, C. Future internet architecture: clean-slate versus evolutionary research. **Communications of the ACM**, v. 53, n. 9, p. 36–40, 2010. ISSN 0001-0782.

ROBERTS, J. The clean-slate approach to future internet design: a survey of research initiatives. **annals of telecommunications - annales des télécommunications**, v. 64, n. 5-6, p. 271–276, 2009. ISSN 0003-4347. Disponível em: <http://dx.doi.org/10.1007/s12243-009-0109-y>.

ROTHENBERG, C. E. et al. Openflow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. **Cad. CPqD Tecnologia, Campinas**, v. 7, n. 1, p. 65–76, 2010.

SHARMA, S. et al. Fast failure recovery for in-band OpenFlow networks. In: **Design of Reliable Communication Networks (DRCN), 2013 9th International Conference on the**. [S.l.: s.n.], 2013. p. 52–59.

SHIMONISHI, H. et al. **Trema Repository**. [s.n.], 2011. Disponível em: <http://trema.github.io/trema/>.

SILVA, B. **Solução para controle de conexão com a Internet em uma rede doméstica com SDN/OpenFlow**. 2015. Disponível em: <http://docplayer.com.br/docview/22/1413706/#file=/storage/22/1413706/1413706.pdf>.

SILVA, F. de O. et al. Semantically enriched services to understand the need of entities. In: ÁLVAREZ, F. et al. (Ed.). **The Future Internet**. Springer Berlin / Heidelberg, 2012, (Lecture Notes in Computer Science, v. 7281). p. 142–153. ISBN 978-3-642-30240-4. Disponível em: <http://www.springerlink.com/content/1222874ul734676k/abstract/>.

STAESSENS, D. et al. Software defined networking: Meeting carrier grade requirements. In: **2011 18th IEEE Workshop on Local Metropolitan Area Networks (LANMAN)**. [S.l.: s.n.], 2011. p. 1–6.

TAM, F. On engineering standards based carrier grade platforms. In: **Proceedings of the 2007 workshop on Engineering fault tolerant systems**. New York, NY, USA: ACM, 2007. (EFTS '07). ISBN 978-1-59593-725-4. Disponível em: <http://doi.acm.org/10.1145/1316550.1316554>.

TOOTOONCHIAN, A.; GANJALI, Y. Hyperflow: A distributed control plane for openflow. In: USENIX ASSOCIATION. **Proceedings of the 2010 internet network management conference on Research on enterprise networking**. [S.l.], 2010. p. 3–3.

TOOTOONCHIAN, A.; GORBUNOV, S.; GANJALI, Y. **On Controller Performance in Software-Defined Networks**. 2015. Disponível em: <https://www.usenix.org/system/files/conference/hot-ice12/hotice12-final33_0.pdf>.

TRAVELPING. **FlowER - Erlang OpenFlow Development Platform**. 2014. Disponível em: <http://travelping.github.io/flower/>.

WANG, F. et al. A research on high-performance sdn controller. In: IEEE. **Cloud Computing and Big Data (CCBD), 2014 International Conference on**. [S.l.], 2014. p. 168–174.

ZAHARIADIS, T. et al. Towards a future internet architecture. In: DOMINGUE, J. et al. (Ed.). **The Future Internet. Future Internet Assembly 2011: Achievements and Technological Promises**. Berlin, Heidelberg: Springer-Verlag, 2011, (LNCS, v. 6656). p. 7–18. ISBN 978-3-642-20897-3. Disponível em: <http://link.springer.com/chapter/10.1007/978-3-642-20898-0_1>.

Apêndices

APÊNDICE **A**

Codificação da topologia utilizada

A topologia utilizada para os experimentos no Capítulo 3 foram criadas através de scripts em linguagem Python. Esse script foi passado para o Mininet através do parâmetro custom. *Dessa forma, o Mininet inicia-se considerando a topologia presente em cada script.*

O script utilizado é apresentado no presente Apêndice.

A.1 Script Python para a montagem da topologia no experimento para ambos os Controladores

```
#!/usr/bin/env python

from mininet.topo import Topo

class ObeliskTopo( Topo ):
    def __init__( self ):
        Topo.__init__( self )
        topSwitch = self.addSwitch('s1',dpid='1000'.zfill(16))
        leftTopSwitch = self.addSwitch('s2',dpid='2000'.zfill(16))
        rightTopSwitch = self.addSwitch('s5',dpid='5000'.zfill(16))
        leftBotSwitch = self.addSwitch('s3',dpid='3000'.zfill(16))
        rightBotSwitch = self.addSwitch('s6',dpid='6000'.zfill(16))
        midBotSwitch = self.addSwitch('s28',dpid='2800'.zfill(16))

        topHost = self.addHost( 'h1' )
        leftTopHost = self.addHost('h2')
        rightTopHost = self.addHost('h5')
        leftBotHost = self.addHost('h3')
        rightBotHost = self.addHost('h6')
        midBotHost = self.addHost('h28')
```

```
        self.addLink(topSwitch,topHost)
        self.addLink(leftTopSwitch,leftTopHost)
        self.addLink(rightTopSwitch,rightTopHost)
        self.addLink(leftBotSwitch,leftBotHost)
        self.addLink(rightBotSwitch,rightBotHost)
        self.addLink(midBotSwitch,midBotHost)
        self.addLink(leftTopSwitch,rightTopSwitch)
        self.addLink(topSwitch,leftTopSwitch)
        self.addLink(topSwitch,rightTopSwitch)
        self.addLink(leftTopSwitch,leftBotSwitch)
        self.addLink(rightTopSwitch,rightBotSwitch)
        self.addLink(leftBotSwitch,midBotSwitch)
        self.addLink(midBotSwitch,rightBotSwitch)

        agg1Switch = self.addSwitch('s4',dpid = '3004'.zfill(16))
        agg2Switch = self.addSwitch('s7',dpid = '6007'.zfill(16))
        agg1Host = self.addHost('h4')
        agg2Host = self.addHost('h7')
        self.addLink(agg1Switch,agg1Host)
        self.addLink(agg2Switch,agg2Host)
        self.addLink(agg1Switch, leftBotSwitch)
        self.addLink(agg2Switch, rightBotSwitch)

        for i in range(10):
            num = str(i+8)
            switch = self.addSwitch('s'+num,dpid = ('30'+num.zfill(2)).zf
            host = self.addHost('h'+num)
            self.addLink(switch, host)
            self.addLink(switch, agg1Switch)

        for i in range(10):
            num = str(i+18)
            switch = self.addSwitch('s'+num,dpid = ('60'+num.zfill(2)).zf
            host = self.addHost('h'+num)
            self.addLink(switch, host)
            self.addLink(switch, agg2Switch)

topos = { 'obelisk': (lambda: ObeliskTopo() ) }
```

```
def run():
    topo = ObeliskTopo()
    net = Mininet( topo=topo, controller=RemoteController, autoSetMac
    net.start()
    CLI( net )
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )
    run()
```

Utiliza-se o seguintes comandos para conectar os *switches* aos Controladores o OpenFlow. Existe outras maneiras de realizar conexão além desta descrita.

```
sh ovs-vsctl set-controller s1 ptcp:6634 tcp:10.0.3.223:6633 tcp:10.0
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s2 ptcp:6635 tcp:10.0.3.223:6633 tcp:10.0
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s3 ptcp:6636 tcp:10.0.3.223:6633 tcp:10.0
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s4 ptcp:6637 tcp:10.0.3.223:6633 tcp:10.0
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s5 ptcp:6638 tcp:10.0.3.223:6633 tcp:10.0
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s6 ptcp:6639 tcp:10.0.3.223:6633 tcp:10.0
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s7 ptcp:6640 tcp:10.0.3.223:6633 tcp:10.0
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s8 ptcp:6641 tcp:10.0.3.223:6633 tcp:10.0
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s9 ptcp:6642 tcp:10.0.3.223:6633 tcp:10.0
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s10 ptcp:6643 tcp:10.0.3.223:6633 tcp:10.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s11 ptcp:6644 tcp:10.0.3.223:6633 tcp:10.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s12 ptcp:6645 tcp:10.0.3.223:6633 tcp:10.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s13 ptcp:6646 tcp:10.0.3.223:6633 tcp:10.
Expected Prompt 'mininet>' Found
```

```
sh ovs-vsctl set-controller s14 ptcp:6647 tcp:10.0.3.223:6633 tcp:10.0.3.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s15 ptcp:6648 tcp:10.0.3.223:6633 tcp:10.0.3.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s16 ptcp:6649 tcp:10.0.3.223:6633 tcp:10.0.3.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s17 ptcp:6650 tcp:10.0.3.223:6633 tcp:10.0.3.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s18 ptcp:6651 tcp:10.0.3.223:6633 tcp:10.0.3.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s19 ptcp:6652 tcp:10.0.3.223:6633 tcp:10.0.3.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s20 ptcp:6653 tcp:10.0.3.223:6633 tcp:10.0.3.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s21 ptcp:6654 tcp:10.0.3.223:6633 tcp:10.0.3.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s22 ptcp:6655 tcp:10.0.3.223:6633 tcp:10.0.3.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s23 ptcp:6656 tcp:10.0.3.223:6633 tcp:10.0.3.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s24 ptcp:6657 tcp:10.0.3.223:6633 tcp:10.0.3.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s25 ptcp:6658 tcp:10.0.3.223:6633 tcp:10.0.3.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s26 ptcp:6659 tcp:10.0.3.223:6633 tcp:10.0.3.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s27 ptcp:6660 tcp:10.0.3.223:6633 tcp:10.0.3.
Expected Prompt 'mininet>' Found
sh ovs-vsctl set-controller s28 ptcp:6661 tcp:10.0.3.223:6633 tcp:10.0.3.
Expected Prompt 'mininet>' Found
```

APÊNDICE **B**

Codificação Casos de Teste do TestON

A topologia utilizada para os experimentos no Capítulo 3 foram criadas através de scripts feito utilizando a ferramenta TestON. Este script realiza do a simulação e execução da Mininet e comandos externos.

O scripts utilizados são apresentado no presente Apêndice.

B.1 Script criação de casos de testes para a execução dos experimentos

Arquivo casos de teste HATestMinorityRestart.py

```
#!/usr/bin/env python
"""
Description: This test is to determine if ONOS can handle
    a minority of it's nodes restarting

List of test cases:
CASE1: Compile ONOS and push it to the test machines
CASE2: Assign mastership to controllers
CASE3: Assign intents
CASE4: Ping across added host intents
CASE5: Reading state of ONOSs
CASE6: The Failure case.
CASE7: Check state after control plane failure
CASE8: Compare topo
CASE9: Link s3-s28 down
CASE10: Link s3-s28 up
CASE11: Switch down
CASE12: Switch up
CASE13: Clean up
CASE14: start election app on all onos nodes
```

```
CASE15: Check that Leadership Election is still functional
"""


class HATestMinorityRestart:

    def __init__( self ):
        self.default = ''

    def CASE1( self, main ):
        """
        CASE1 is to compile ONOS and push it to the test machines

        Startup sequence:
        cell <name>
        onos-verify-cell
        NOTE: temporary - onos-remove-raft-logs
        onos-uninstall
        start mininet
        git pull
        mvn clean install
        onos-package
        onos-install -f
        onos-wait-for-start
        start cli sessions
        start tcpdump
        """
        main.log.report(
            "ONOS HA test: Restart minority of ONOS nodes - initializatio
        main.case( "Setting up test environment" )
        # TODO: save all the timers and output them for plotting

        # load some variables from the params file
        PULLCODE = False
#         if main.params[ 'Git' ] == 'True':
#             PULLCODE = True
#         gitBranch = main.params[ 'branch' ]
        cellName = main.params[ 'ENV' ][ 'cellName' ]
```

```
        # set global variables
        global ONOS1Port
        global ONOS2Port
        global ONOS3Port
        global ONOS4Port
        global ONOS5Port
        global ONOS6Port
        global ONOS7Port
        global numControllers
        numControllers = int( main.params[ 'num_controllers' ] )

        # FIXME: just get controller port from params?
        # TODO: do we really need all these?
        ONOS1Port = main.params[ 'CTRL' ][ 'port1' ]
        ONOS2Port = main.params[ 'CTRL' ][ 'port2' ]
        ONOS3Port = main.params[ 'CTRL' ][ 'port3' ]
        ONOS4Port = main.params[ 'CTRL' ][ 'port4' ]
        ONOS5Port = main.params[ 'CTRL' ][ 'port5' ]
        ONOS6Port = main.params[ 'CTRL' ][ 'port6' ]
        ONOS7Port = main.params[ 'CTRL' ][ 'port7' ]

        global CLIs
        CLIs = []
        global nodes
        nodes = []
        for i in range( 1, numControllers + 1 ):
            CLIs.append( getattr( main, 'ONOScli' + str( i ) ) )
            nodes.append( getattr( main, 'ONOS' + str( i ) ) )

        main.step( "Applying cell variable to environment" )
        cellResult = main.ONOSbench.setCell( cellName )
        verifyResult = main.ONOSbench.verifyCell()

        # FIXME:this is short term fix
#         main.log.report( "Removing raft logs" )
#         main.ONOSbench.onosRemoveRaftLogs()

#         main.log.report( "Uninstalling ONOS" )
#         for node in nodes:
```

```
#             main.ONOSbench.onosUninstall( node.ip_address )

        cleanInstallResult = main.TRUE
        gitPullResult = main.TRUE

        main.step( "Starting Mininet" )
        main.Mininet1.startNet( )

#         main.step( "Compiling the latest version of ONOS" )
#         if PULLCODE:
#             main.step( "Git checkout and pull " + gitBranch )
#             main.ONOSbench.gitCheckout( gitBranch )
#             gitPullResult = main.ONOSbench.gitPull()
#             if gitPullResult == main.ERROR:
#                 main.log.error( "Error pulling git branch" )

#             main.step( "Using mvn clean & install" )
#             cleanInstallResult = main.ONOSbench.cleanInstall()
#         else:
#             main.log.warn( "Did not pull new code so skipping mvn " +
#                            "clean install" )
        main.ONOSbench.getVersion( report=True )

#         main.step( "Creating ONOS package" )
#         packageResult = main.ONOSbench.onosPackage()
        packageResult = main.TRUE

#         main.step( "Installing ONOS package" )
#         onosInstallResult = main.TRUE
#         for node in nodes:
#             tmpResult = main.ONOSbench.onosInstall( options="-f",
#                                                     node=node.ip_address
#             onosInstallResult = onosInstallResult and tmpResult
        onosInstallResult = main.TRUE

        main.step( "Checking if ONOS is up yet" )
        for i in range( 2 ):
            onosIsupResult = main.TRUE
            for node in nodes:
```

```
            started = main.ONOSbench.isup( node.ip_address )
            if not started:
                main.log.report( node.name + " didn't start!" )
                main.ONOSbench.onosStop( node.ip_address )
                main.ONOSbench.onosStart( node.ip_address )
            onosIsupResult = onosIsupResult and started
        if onosIsupResult == main.TRUE:
            break

main.log.step( "Starting ONOS CLI sessions" )
cliResults = main.TRUE
threads = []
for i in range( numControllers ):
    t = main.Thread( target=CLIs[i].startOnosCli,
                     name="startOnosCli-" + str( i ),
                     args=[nodes[i].ip_address] )
    threads.append( t )
    t.start()

for t in threads:
    t.join()
    cliResults = cliResults and t.result

main.step( "Start Packet Capture MN" )
main.Mininet2.startTcpdump(
    str( main.params[ 'MNtcpdump' ][ 'folder' ] ) + str( main
    + "-MN.pcap",
    intf=main.params[ 'MNtcpdump' ][ 'intf' ],
    port=main.params[ 'MNtcpdump' ][ 'port' ] )

appCheck = main.TRUE
threads = []
for i in range( numControllers ):
    t = main.Thread( target=CLIs[i].appToIDCheck,
                     name="appToIDCheck-" + str( i ),
                     args=[] )
    threads.append( t )
    t.start()
```

```
        for t in threads:
            t.join()
            appCheck = appCheck and t.result
        utilities.assert_equals( expect=main.TRUE, actual=appCheck,
                                 onpass="App Ids seem to be correct",
                                 onfail="Something is wrong with app Ids"
        if appCheck != main.TRUE:
            main.log.warn( CLIs[0].apps() )
            main.log.warn( CLIs[0].appIDs() )

        case1Result = ( cleanInstallResult and packageResult and
                        cellResult and verifyResult and onosInstallResult
                        and onosIsupResult and cliResults and appCheck)

        utilities.assert_equals( expect=main.TRUE, actual=case1Result,
                                 onpass="Test startup successful",
                                 onfail="Test startup NOT successful" )

        if case1Result == main.FALSE:
            main.cleanup()
            main.exit()

    def CASE2( self, main ):
        """
        Assign mastership to controllers
        """
        import re
        assert numControllers, "numControllers not defined"
        assert main, "main not defined"
        assert utilities.assert_equals, "utilities.assert_equals not defi
        assert CLIs, "CLIs not defined"
        assert nodes, "nodes not defined"
        assert ONOS1Port, "ONOS1Port not defined"
        assert ONOS2Port, "ONOS2Port not defined"
        assert ONOS3Port, "ONOS3Port not defined"
        assert ONOS4Port, "ONOS4Port not defined"
        assert ONOS5Port, "ONOS5Port not defined"
        assert ONOS6Port, "ONOS6Port not defined"
        assert ONOS7Port, "ONOS7Port not defined"
```

```
main.log.report( "Assigning switches to controllers" )
main.case( "Assigning Controllers" )
main.step( "Assign switches to controllers" )

# TODO: rewrite this function to take lists of ips and ports?
#       or list of tuples?
for i in range( 1, 29 ):
    main.Mininet1.assignSwController(
        sw=str( i ),
        count=numControllers,
        ip1=nodes[ 0 ].ip_address, port1=ONOS1Port,
        ip2=nodes[ 1 ].ip_address, port2=ONOS2Port,
        ip3=nodes[ 2 ].ip_address, port3=ONOS3Port,
        ip4=nodes[ 3 ].ip_address, port4=ONOS4Port,
        ip5=nodes[ 4 ].ip_address, port5=ONOS5Port,
        ip6=nodes[ 5 ].ip_address, port6=ONOS6Port,
        ip7=nodes[ 6 ].ip_address, port7=ONOS7Port )

mastershipCheck = main.TRUE
for i in range( 1, 29 ):
    response = main.Mininet1.getSwController( "s" + str( i )
    try:
        main.log.info( str( response ) )
    except Exception:
        main.log.info( repr( response ) )
    for node in nodes:
        if re.search( "tcp:" + node.ip_address, response ):
            mastershipCheck = mastershipCheck and main.TRUE
        else:
            main.log.error( "Error, node " + node.ip_address
                            "not in the list of controllers s
                            str( i ) + " is connecting to." )
            mastershipCheck = main.FALSE
if mastershipCheck == main.TRUE:
    main.log.report( "Switch mastership assigned correctly" )
utilities.assert_equals(
    expect=main.TRUE,
    actual=mastershipCheck,
```

```
        onpass="Switch mastership assigned correctly",
        onfail="Switches not assigned correctly to controllers" )
    # Manually assign mastership to the controller we want
    roleCall = main.TRUE
    roleCheck = main.TRUE
    try:
        for i in range( 1, 29 ):  # switches 1 through 28
            # set up correct variables:
            if i == 1:
                ip = nodes[ 0 ].ip_address  # ONOS1
                deviceId = main.ONOScli1.getDevice( "1000" ).get( 'id
            elif i == 2:
                ip = nodes[ 1 ].ip_address  # ONOS2
                deviceId = main.ONOScli1.getDevice( "2000" ).get( 'id
            elif i == 3:
                ip = nodes[ 1 ].ip_address  # ONOS2
                deviceId = main.ONOScli1.getDevice( "3000" ).get( 'id
            elif i == 4:
                ip = nodes[ 3 ].ip_address  # ONOS4
                deviceId = main.ONOScli1.getDevice( "3004" ).get( 'id
            elif i == 5:
                ip = nodes[ 2 ].ip_address  # ONOS3
                deviceId = main.ONOScli1.getDevice( "5000" ).get( 'id
            elif i == 6:
                ip = nodes[ 2 ].ip_address  # ONOS3
                deviceId = main.ONOScli1.getDevice( "6000" ).get( 'id
            elif i == 7:
                ip = nodes[ 5 ].ip_address  # ONOS6
                deviceId = main.ONOScli1.getDevice( "6007" ).get( 'id
            elif i >= 8 and i <= 17:
                ip = nodes[ 4 ].ip_address  # ONOS5
                dpid = '3' + str( i ).zfill( 3 )
                deviceId = main.ONOScli1.getDevice( dpid ).get( 'id'
            elif i >= 18 and i <= 27:
                ip = nodes[ 6 ].ip_address  # ONOS7
                dpid = '6' + str( i ).zfill( 3 )
                deviceId = main.ONOScli1.getDevice( dpid ).get( 'id'
            elif i == 28:
                ip = nodes[ 0 ].ip_address  # ONOS1
```

```
                deviceId = main.ONOScli1.getDevice( "2800" ).get(
            else:
                main.log.error( "You didn't write an else stateme
                                "switch s" + str( i ) )
            # Assign switch
            assert deviceId, "No device id for s" + str( i ) + "
            # TODO: make this controller dynamic
            roleCall = roleCall and main.ONOScli1.deviceRole( dev
                                                              ip
            # Check assignment
            master =  main.ONOScli1.getRole( deviceId ).get( 'mas
            if ip in master:
                roleCheck = roleCheck and main.TRUE
            else:
                roleCheck = roleCheck and main.FALSE
                main.log.error( "Error, controller " + ip + " is
                                " master " + "of device " +
                                str( deviceId ) + ". Master is "
                                repr( master ) + "." )
    except ( AttributeError, AssertionError ):
        main.log.exception( "Something is wrong with ONOS device
        main.log.info( main.ONOScli1.devices() )
    utilities.assert_equals(
        expect=main.TRUE,
        actual=roleCall,
        onpass="Re-assigned switch mastership to designated contr
        onfail="Something wrong with deviceRole calls" )

    utilities.assert_equals(
        expect=main.TRUE,
        actual=roleCheck,
        onpass="Switches were successfully reassigned to designat
               "controller",
        onfail="Switches were not successfully reassigned" )
    mastershipCheck = mastershipCheck and roleCall and roleCheck
    utilities.assert_equals( expect=main.TRUE, actual=mastershipCh
                             onpass="Switch mastership correctly
                             onfail="Error in (re)assigning switc
                             " mastership" )
```

```
def CASE3( self, main ):
    """
    Assign intents
    """
    import time
    import json
    assert numControllers, "numControllers not defined"
    assert main, "main not defined"
    assert utilities.assert_equals, "utilities.assert_equals not defi
    assert CLIs, "CLIs not defined"
    assert nodes, "nodes not defined"
    main.log.report( "Adding host intents" )
    main.case( "Adding host Intents" )

    main.step( "Discovering  Hosts( Via pingall for now )" )
    # FIXME: Once we have a host discovery mechanism, use that instea

    # install onos-app-fwd
    main.log.info( "Install reactive forwarding app" )
    appResults = CLIs[0].activateApp( "org.onosproject.fwd" )

    # FIXME: add this to asserts
    appCheck = main.TRUE
    threads = []
    for i in range( numControllers ):
        t = main.Thread( target=CLIs[i].appToIDCheck,
                         name="appToIDCheck-" + str( i ),
                         args=[] )
        threads.append( t )
        t.start()

    for t in threads:
        t.join()
        appCheck = appCheck and t.result
    utilities.assert_equals( expect=main.TRUE, actual=appCheck,
                             onpass="App Ids seem to be correct",
                             onfail="Something is wrong with app Ids"
    if appCheck != main.TRUE:
```

```
        main.log.warn( CLIs[0].apps() )
        main.log.warn( CLIs[0].appIDs() )

    # REACTIVE FWD test
    pingResult = main.FALSE
    for i in range(2):  # Retry if pingall fails first time
        time1 = time.time()
        pingResult = main.Mininet1.pingall()
        utilities.assert_equals(
            expect=main.TRUE,
            actual=pingResult,
            onpass="Reactive Pingall test passed",
            onfail="Reactive Pingall failed, one or more ping pai
        time2 = time.time()
        main.log.info( "Time for pingall: %2f seconds" % ( time2

    # uninstall onos-app-fwd
    main.log.info( "Uninstall reactive forwarding app" )
    appResults = appResults and CLIs[0].deactivateApp( "org.onosp
    threads = []
    for i in range( numControllers ):
        t = main.Thread( target=CLIs[i].appToIDCheck,
                         name="appToIDCheck-" + str( i ),
                         args=[] )
        threads.append( t )
        t.start()

    for t in threads:
        t.join()
        appCheck = appCheck and t.result
    utilities.assert_equals( expect=main.TRUE, actual=appCheck,
                             onpass="App Ids seem to be correct",
                             onfail="Something is wrong with app
    if appCheck != main.TRUE:
        main.log.warn( CLIs[0].apps() )
        main.log.warn( CLIs[0].appIDs() )

    # timeout for fwd flows
    time.sleep( 11 )
```

```
main.step( "Add host intents" )
intentIds = []
# TODO:  move the host numbers to params
#        Maybe look at all the paths we ping?
intentAddResult = True
hostResult = main.TRUE
for i in range( 8, 18 ):
    main.log.info( "Adding host intent between h" + str( i ) +
                   " and h" + str( i + 10 ) )
    host1 = "00:00:00:00:00:" + \
        str( hex( i )[ 2: ] ).zfill( 2 ).upper()
    host2 = "00:00:00:00:00:" + \
        str( hex( i + 10 )[ 2: ] ).zfill( 2 ).upper()
    # NOTE: getHost can return None
    host1Dict = main.ONOScli1.getHost( host1 )
    host2Dict = main.ONOScli1.getHost( host2 )
    host1Id = None
    host2Id = None
    if host1Dict and host2Dict:
        host1Id = host1Dict.get( 'id', None )
        host2Id = host2Dict.get( 'id', None )
    if host1Id and host2Id:
        nodeNum = ( i % 7 )
        tmpId = CLIs[ nodeNum ].addHostIntent( host1Id, host2Id )
        if tmpId:
            main.log.info( "Added intent with id: " + tmpId )
            intentIds.append( tmpId )
        else:
            main.log.error( "addHostIntent returned: " +
                            repr( tmpId ) )
    else:
        main.log.error( "Error, getHost() failed for h" + str( i
                        " and/or h" + str( i + 10 ) )
        hosts = CLIs[ 0 ].hosts()
        main.log.warn( "Hosts output: " )
        try:
            main.log.warn( json.dumps( json.loads( hosts ),
                                       sort_keys=True,
```

```
                                                indent=4,
                                                separators=( ',', ': '
            except ( ValueError, TypeError ):
                main.log.warn( repr( hosts ) )
            hostResult = main.FALSE
    # FIXME: DEBUG
    intentStart = time.time()
    onosIds = main.ONOScli1.getAllIntentsId()
    main.log.info( "Submitted intents: " + str( intentIds ) )
    main.log.info( "Intents in ONOS: " + str( onosIds ) )
    for intent in intentIds:
        if intent in onosIds:
            pass  # intent submitted is in onos
        else:
            intentAddResult = False
    # FIXME: DEBUG
    if intentAddResult:
        intentStop = time.time()
    else:
        intentStop = None
    # Print the intent states
    intents = main.ONOScli1.intents()
    intentStates = []
    installedCheck = True
    main.log.info( "%-6s%-15s%-15s" % ( 'Count', 'ID', 'State' ) )
    count = 0
    try:
        for intent in json.loads( intents ):
            state = intent.get( 'state', None )
            if "INSTALLED" not in state:
                installedCheck = False
            intentId = intent.get( 'id', None )
            intentStates.append( ( intentId, state ) )
    except ( ValueError, TypeError ):
        main.log.exception( "Error parsing intents" )
    # add submitted intents not in the store
    tmplist = [ i for i, s in intentStates ]
    missingIntents = False
    for i in intentIds:
```

```
        if i not in tmplist:
            intentStates.append( ( i, " - " ) )
            missingIntents = True
    intentStates.sort()
    for i, s in intentStates:
        count += 1
        main.log.info( "%-6s%-15s%-15s" %
                       ( str( count ), str( i ), str( s ) ) )
    leaders = main.ONOScli1.leaders()
    try:
        if leaders:
            parsedLeaders = json.loads( leaders )
            main.log.warn( json.dumps( parsedLeaders,
                                       sort_keys=True,
                                       indent=4,
                                       separators=( ',', ': ' ) ) )
            # check for all intent partitions
            topics = []
            for i in range( 14 ):
                topics.append( "intent-partition-" + str( i ) )
            main.log.debug( topics )
            ONOStopics = [ j['topic'] for j in parsedLeaders ]
            for topic in topics:
                if topic not in ONOStopics:
                    main.log.error( "Error: " + topic +
                                    " not in leaders" )
        else:
            main.log.error( "leaders() returned None" )
    except ( ValueError, TypeError ):
        main.log.exception( "Error parsing leaders" )
        main.log.error( repr( leaders ) )
    partitions = main.ONOScli1.partitions()
    try:
        if partitions :
            parsedPartitions = json.loads( partitions )
            main.log.warn( json.dumps( parsedPartitions,
                                       sort_keys=True,
                                       indent=4,
                                       separators=( ',', ': ' ) ) )
```

```
                # TODO check for a leader in all paritions
                # TODO check for consistency among nodes
            else:
                main.log.error( "partitions() returned None" )
        except ( ValueError, TypeError ):
            main.log.exception( "Error parsing partitions" )
            main.log.error( repr( partitions ) )
        pendingMap = main.ONOScli1.pendingMap()
        try:
            if pendingMap :
                parsedPending = json.loads( pendingMap )
                main.log.warn( json.dumps( parsedPending,
                                           sort_keys=True,
                                           indent=4,
                                           separators=( ',', ': ' ) ) )
                # TODO check something here?
            else:
                main.log.error( "pendingMap() returned None" )
        except ( ValueError, TypeError ):
            main.log.exception( "Error parsing pending map" )
            main.log.error( repr( pendingMap ) )

        intentAddResult = bool( pingResult and hostResult and intentA
                                and not missingIntents and installedC
        utilities.assert_equals(
            expect=True,
            actual=intentAddResult,
            onpass="Pushed host intents to ONOS",
            onfail="Error in pushing host intents to ONOS" )
        for i in range(100):
            onosIds = main.ONOScli1.getAllIntentsId()
            main.log.info( "Submitted intents: " + str( sorted( inten
            main.log.info( "Intents in ONOS: " + str( sorted( onosIds
            if sorted(onosIds) == sorted(intentIds):
                break
            else:
                time.sleep(1)
        # FIXME: DEBUG
        if not intentStop:
```

```
        intentStop = time.time()
    gossipTime = intentStop - intentStart
    main.log.info( "It took about " + str( gossipTime ) +
                    " seconds for all intents to appear on ONOS1" )
    # FIXME: make this time configurable/calculate based off of numbe
    #         nodes and gossip rounds
    utilities.assert_greater_equals(
            expect=30, actual=gossipTime,
            onpass="ECM anti-entropy for intents worked within " +
                   "expected time",
            onfail="Intent ECM anti-entropy took too long" )
    if gossipTime <= 30:
        intentAddResult = True

    if not intentAddResult or "key" in pendingMap:
        import time
        installedCheck = True
        main.log.info( "Sleeping 60 seconds to see if intents are fou
        time.sleep( 60 )
        onosIds = main.ONOScli1.getAllIntentsId()
        main.log.info( "Submitted intents: " + str( intentIds ) )
        main.log.info( "Intents in ONOS: " + str( onosIds ) )
        # Print the intent states
        intents = main.ONOScli1.intents()
        intentStates = []
        main.log.info( "%-6s%-15s%-15s" % ( 'Count', 'ID', 'State' )
        count = 0
        try:
            for intent in json.loads( intents ):
                # Iter through intents of a node
                state = intent.get( 'state', None )
                if "INSTALLED" not in state:
                    installedCheck = False
                intentId = intent.get( 'id', None )
                intentStates.append( ( intentId, state ) )
        except ( ValueError, TypeError ):
            main.log.exception( "Error parsing intents" )
        # add submitted intents not in the store
        tmplist = [ i for i, s in intentStates ]
```

```
for i in intentIds:
    if i not in tmplist:
        intentStates.append( ( i, " - " ) )
intentStates.sort()
for i, s in intentStates:
    count += 1
    main.log.info( "%-6s%-15s%-15s" %
                   ( str( count ), str( i ), str( s ) ) )
leaders = main.ONOScli1.leaders()
try:
    if leaders:
        parsedLeaders = json.loads( leaders )
        main.log.warn( json.dumps( parsedLeaders,
                                   sort_keys=True,
                                   indent=4,
                                   separators=( ',', ': '
        # check for all intent partitions
        # check for election
        topics = []
        for i in range( 14 ):
            topics.append( "intent-partition-" + str( i )
        # FIXME: this should only be after we start the a
        topics.append( "org.onosproject.election" )
        main.log.debug( topics )
        ONOStopics = [ j['topic'] for j in parsedLeaders
        for topic in topics:
            if topic not in ONOStopics:
                main.log.error( "Error: " + topic +
                                " not in leaders" )
    else:
        main.log.error( "leaders() returned None" )
except ( ValueError, TypeError ):
    main.log.exception( "Error parsing leaders" )
    main.log.error( repr( leaders ) )
partitions = main.ONOScli1.partitions()
try:
    if partitions :
        parsedPartitions = json.loads( partitions )
        main.log.warn( json.dumps( parsedPartitions,
```

```
                                                sort_keys=True,
                                                indent=4,
                                                separators=( ',', ': ' ) )
                        # TODO check for a leader in all paritions
                        # TODO check for consistency among nodes
                    else:
                        main.log.error( "partitions() returned None" )
                except ( ValueError, TypeError ):
                    main.log.exception( "Error parsing partitions" )
                    main.log.error( repr( partitions ) )
                pendingMap = main.ONOScli1.pendingMap()
                try:
                    if pendingMap :
                        parsedPending = json.loads( pendingMap )
                        main.log.warn( json.dumps( parsedPending,
                                                sort_keys=True,
                                                indent=4,
                                                separators=( ',', ': ' ) )
                        # TODO check something here?
                    else:
                        main.log.error( "pendingMap() returned None" )
                except ( ValueError, TypeError ):
                    main.log.exception( "Error parsing pending map" )
                    main.log.error( repr( pendingMap ) )

    def CASE4( self, main ):
        """
        Ping across added host intents
        """
        import json
        import time
        assert numControllers, "numControllers not defined"
        assert main, "main not defined"
        assert utilities.assert_equals, "utilities.assert_equals not defi
        assert CLIs, "CLIs not defined"
        assert nodes, "nodes not defined"
        description = " Ping across added host intents"
        main.log.report( description )
        main.case( description )
```

```
        PingResult = main.TRUE
        for i in range( 8, 18 ):
            ping = main.Mininet1.pingHost( src="h" + str( i ),
                                           target="h" + str( i + 10 )
            PingResult = PingResult and ping
            if ping == main.FALSE:
                main.log.warn( "Ping failed between h" + str( i ) +
                               " and h" + str( i + 10 ) )
            elif ping == main.TRUE:
                main.log.info( "Ping test passed!" )
                # Don't set PingResult or you'd override failures
        if PingResult == main.FALSE:
            main.log.report(
                "Intents have not been installed correctly, pings fai
            # TODO: pretty print
            main.log.warn( "ONOS1 intents: " )
            try:
                tmpIntents = main.ONOScli1.intents()
                main.log.warn( json.dumps( json.loads( tmpIntents ),
                                           sort_keys=True,
                                           indent=4,
                                           separators=( ',', ': ' ) )
            except ( ValueError, TypeError ):
                main.log.warn( repr( tmpIntents ) )
        if PingResult == main.TRUE:
            main.log.report(
                "Intents have been installed correctly and verified b
        utilities.assert_equals(
            expect=main.TRUE,
            actual=PingResult,
            onpass="Intents have been installed correctly and pings w
            onfail="Intents have not been installed correctly, pings

        installedCheck = True
        if PingResult is not main.TRUE:
            # Print the intent states
            intents = main.ONOScli1.intents()
            intentStates = []
            main.log.info( "%-6s%-15s%-15s" % ( 'Count', 'ID', 'State
```

```
count = 0
# Iter through intents of a node
try:
    for intent in json.loads( intents ):
        state = intent.get( 'state', None )
        if "INSTALLED" not in state:
            installedCheck = False
        intentId = intent.get( 'id', None )
        intentStates.append( ( intentId, state ) )
except ( ValueError, TypeError ):
    main.log.exception( "Error parsing intents." )
intentStates.sort()
for i, s in intentStates:
    count += 1
    main.log.info( "%-6s%-15s%-15s" %
                   ( str( count ), str( i ), str( s ) ) )
leaders = main.ONOScli1.leaders()
try:
    if leaders:
        parsedLeaders = json.loads( leaders )
        main.log.warn( json.dumps( parsedLeaders,
                                   sort_keys=True,
                                   indent=4,
                                   separators=( ',', ': ' ) ) )
        # check for all intent partitions
        # check for election
        topics = []
        for i in range( 14 ):
            topics.append( "intent-partition-" + str( i ) )
        # FIXME: this should only be after we start the app
        topics.append( "org.onosproject.election" )
        main.log.debug( topics )
        ONOStopics = [ j['topic'] for j in parsedLeaders ]
        for topic in topics:
            if topic not in ONOStopics:
                main.log.error( "Error: " + topic +
                                " not in leaders" )
    else:
        main.log.error( "leaders() returned None" )
```

```
            except ( ValueError, TypeError ):
                main.log.exception( "Error parsing leaders" )
                main.log.error( repr( leaders ) )
            partitions = main.ONOScli1.partitions()
            try:
                if partitions :
                    parsedPartitions = json.loads( partitions )
                    main.log.warn( json.dumps( parsedPartitions,
                                               sort_keys=True,
                                               indent=4,
                                               separators=( ',', ': '
                    # TODO check for a leader in all paritions
                    # TODO check for consistency among nodes
                else:
                    main.log.error( "partitions() returned None" )
            except ( ValueError, TypeError ):
                main.log.exception( "Error parsing partitions" )
                main.log.error( repr( partitions ) )
            pendingMap = main.ONOScli1.pendingMap()
            try:
                if pendingMap :
                    parsedPending = json.loads( pendingMap )
                    main.log.warn( json.dumps( parsedPending,
                                               sort_keys=True,
                                               indent=4,
                                               separators=( ',', ': '
                    # TODO check something here?
                else:
                    main.log.error( "pendingMap() returned None" )
            except ( ValueError, TypeError ):
                main.log.exception( "Error parsing pending map" )
                main.log.error( repr( pendingMap ) )

        if not installedCheck:
            main.log.info( "Waiting 60 seconds to see if the state of
                           "intents change" )
            time.sleep( 60 )
            # Print the intent states
            intents = main.ONOScli1.intents()
```

```
intentStates = []
main.log.info( "%-6s%-15s%-15s" % ( 'Count', 'ID', 'State' )
count = 0
# Iter through intents of a node
try:
    for intent in json.loads( intents ):
        state = intent.get( 'state', None )
        if "INSTALLED" not in state:
            installedCheck = False
        intentId = intent.get( 'id', None )
        intentStates.append( ( intentId, state ) )
except ( ValueError, TypeError ):
    main.log.exception( "Error parsing intents." )
intentStates.sort()
for i, s in intentStates:
    count += 1
    main.log.info( "%-6s%-15s%-15s" %
                   ( str( count ), str( i ), str( s ) ) )
leaders = main.ONOScli1.leaders()
try:
    if leaders:
        parsedLeaders = json.loads( leaders )
        main.log.warn( json.dumps( parsedLeaders,
                                   sort_keys=True,
                                   indent=4,
                                   separators=( ',', ': ' ) )
        # check for all intent partitions
        # check for election
        topics = []
        for i in range( 14 ):
            topics.append( "intent-partition-" + str( i ) )
        # FIXME: this should only be after we start the app
        topics.append( "org.onosproject.election" )
        main.log.debug( topics )
        ONOStopics = [ j['topic'] for j in parsedLeaders ]
        for topic in topics:
            if topic not in ONOStopics:
                main.log.error( "Error: " + topic +
                                " not in leaders" )
```

```
                else:
                    main.log.error( "leaders() returned None" )
            except ( ValueError, TypeError ):
                main.log.exception( "Error parsing leaders" )
                main.log.error( repr( leaders ) )
            partitions = main.ONOScli1.partitions()
            try:
                if partitions :
                    parsedPartitions = json.loads( partitions )
                    main.log.warn( json.dumps( parsedPartitions,
                                               sort_keys=True,
                                               indent=4,
                                               separators=( ',', ': '
                    # TODO check for a leader in all paritions
                    # TODO check for consistency among nodes
                else:
                    main.log.error( "partitions() returned None" )
            except ( ValueError, TypeError ):
                main.log.exception( "Error parsing partitions" )
                main.log.error( repr( partitions ) )
            pendingMap = main.ONOScli1.pendingMap()
            try:
                if pendingMap :
                    parsedPending = json.loads( pendingMap )
                    main.log.warn( json.dumps( parsedPending,
                                               sort_keys=True,
                                               indent=4,
                                               separators=( ',', ': '
                    # TODO check something here?
                else:
                    main.log.error( "pendingMap() returned None" )
            except ( ValueError, TypeError ):
                main.log.exception( "Error parsing pending map" )
                main.log.error( repr( pendingMap ) )

    def CASE5( self, main ):
        """
        Reading state of ONOS
        """
```

```
import json
import time
assert numControllers, "numControllers not defined"
assert main, "main not defined"
assert utilities.assert_equals, "utilities.assert_equals not defi
assert CLIs, "CLIs not defined"
assert nodes, "nodes not defined"
# assumes that sts is already in you PYTHONPATH
from sts.topology.teston_topology import TestONTopology

main.log.report( "Setting up and gathering data for current state
main.case( "Setting up and gathering data for current state" )
# The general idea for this test case is to pull the state of
# ( intents,flows, topology,... ) from each ONOS node
# We can then compare them with each other and also with past sta

main.step( "Check that each switch has a master" )
global mastershipState
mastershipState = '[]'

# Assert that each device has a master
rolesNotNull = main.TRUE
threads = []
for i in range( numControllers ):
    t = main.Thread( target=CLIs[i].rolesNotNull,
                     name="rolesNotNull-" + str( i ),
                     args=[] )
    threads.append( t )
    t.start()

for t in threads:
    t.join()
    rolesNotNull = rolesNotNull and t.result
utilities.assert_equals(
    expect=main.TRUE,
    actual=rolesNotNull,
    onpass="Each device has a master",
    onfail="Some devices don't have a master assigned" )
```

```
main.step( "Get the Mastership of each switch from each contr
ONOSMastership = []
mastershipCheck = main.FALSE
consistentMastership = True
rolesResults = True
threads = []
for i in range( numControllers ):
    t = main.Thread( target=CLIs[i].roles,
                     name="roles-" + str( i ),
                     args=[] )
    threads.append( t )
    t.start()

for t in threads:
    t.join()
    ONOSMastership.append( t.result )

for i in range( numControllers ):
    if not ONOSMastership[i] or "Error" in ONOSMastership[i]:
        main.log.report( "Error in getting ONOS" + str( i + 1
                         " roles" )
        main.log.warn(
            "ONOS" + str( i + 1 ) + " mastership response: "
            repr( ONOSMastership[i] ) )
        rolesResults = False
utilities.assert_equals(
    expect=True,
    actual=rolesResults,
    onpass="No error in reading roles output",
    onfail="Error in reading roles from ONOS" )

main.step( "Check for consistency in roles from each controll
if all([ i == ONOSMastership[ 0 ] for i in ONOSMastership ] )
    main.log.report(
        "Switch roles are consistent across all ONOS nodes" )
else:
    consistentMastership = False
utilities.assert_equals(
    expect=True,
```

```
        actual=consistentMastership,
        onpass="Switch roles are consistent across all ONOS nodes",
        onfail="ONOS nodes have different views of switch roles" )

    if rolesResults and not consistentMastership:
        for i in range( numControllers ):
            try:
                main.log.warn(
                    "ONOS" + str( i + 1 ) + " roles: ",
                    json.dumps(
                        json.loads( ONOSMastership[ i ] ),
                        sort_keys=True,
                        indent=4,
                        separators=( ',', ': ' ) ) )
            except ( ValueError, TypeError ):
                main.log.warn( repr( ONOSMastership[ i ] ) )
    elif rolesResults and consistentMastership:
        mastershipCheck = main.TRUE
        mastershipState = ONOSMastership[ 0 ]

    main.step( "Get the intents from each controller" )
    global intentState
    intentState = []
    ONOSIntents = []
    intentCheck = main.FALSE
    consistentIntents = True
    intentsResults = True
    threads = []
    for i in range( numControllers ):
        t = main.Thread( target=CLIs[i].intents,
                         name="intents-" + str( i ),
                         args=[],
                         kwargs={ 'jsonFormat': True } )
        threads.append( t )
        t.start()

    for t in threads:
        t.join()
        ONOSIntents.append( t.result )
```

```
for i in range( numControllers ):
    if not ONOSIntents[ i ] or "Error" in ONOSIntents[ i ]:
        main.log.report( "Error in getting ONOS" + str( i + 1
                         " intents" )
        main.log.warn( "ONOS" + str( i + 1 ) + " intents resp
                       repr( ONOSIntents[ i ] ) )
        intentsResults = False
utilities.assert_equals(
    expect=True,
    actual=intentsResults,
    onpass="No error in reading intents output",
    onfail="Error in reading intents from ONOS" )

main.step( "Check for consistency in Intents from each contro
if all([ sorted( i ) == sorted( ONOSIntents[ 0 ] ) for i in O
    main.log.report( "Intents are consistent across all ONOS
                     "nodes" )
else:
    consistentIntents = False
    main.log.report( "Intents not consistent" )
utilities.assert_equals(
    expect=True,
    actual=consistentIntents,
    onpass="Intents are consistent across all ONOS nodes",
    onfail="ONOS nodes have different views of intents" )

if intentsResults and not consistentIntents:
    n = len(ONOSIntents)
    main.log.warn( "ONOS" + str( n ) + " intents: " )
    main.log.warn( json.dumps( json.loads( ONOSIntents[ -1 ]
                               sort_keys=True,
                               indent=4,
                               separators=( ',', ': ' ) ) )
    for i in range( numControllers ):
        if ONOSIntents[ i ] != ONOSIntents[ -1 ]:
            main.log.warn( "ONOS" + str( i + 1 ) + " intents:
            main.log.warn( json.dumps( json.loads( ONOSIntent
                                       sort_keys=True,
```

```
                                                indent=4,
                                                separators=( ',', ': ' ) )
            else:
                main.log.warn( nodes[ i ].name + " intents match ONOS
                               str( n ) + " intents" )
    elif intentsResults and consistentIntents:
        intentCheck = main.TRUE
        intentState = ONOSIntents[ 0 ]

    main.step( "Get the flows from each controller" )
    global flowState
    flowState = []
    ONOSFlows = []
    ONOSFlowsJson = []
    flowCheck = main.FALSE
    consistentFlows = True
    flowsResults = True
    threads = []
    for i in range( numControllers ):
        t = main.Thread( target=CLIs[i].flows,
                         name="flows-" + str( i ),
                         args=[],
                         kwargs={ 'jsonFormat': True } )
        threads.append( t )
        t.start()

    # NOTE: Flows command can take some time to run
    time.sleep(30)
    for t in threads:
        t.join()
        result = t.result
        ONOSFlows.append( result )

    for i in range( numControllers ):
        num = str( i + 1 )
        if not ONOSFlows[ i ] or "Error" in ONOSFlows[ i ]:
            main.log.report( "Error in getting ONOS" + num + " flows"
            main.log.warn( "ONOS" + num + " flows response: " +
                           repr( ONOSFlows[ i ] ) )
```

```
                flowsResults = False
                ONOSFlowsJson.append( None )
            else:
                try:
                    ONOSFlowsJson.append( json.loads( ONOSFlows[ i ]
                except ( ValueError, TypeError ):
                    # FIXME: change this to log.error?
                    main.log.exception( "Error in parsing ONOS" + num
                                        " response as json." )
                    main.log.error( repr( ONOSFlows[ i ] ) )
                    ONOSFlowsJson.append( None )
                    flowsResults = False
        utilities.assert_equals(
            expect=True,
            actual=flowsResults,
            onpass="No error in reading flows output",
            onfail="Error in reading flows from ONOS" )

        main.step( "Check for consistency in Flows from each controll
        tmp = [ len( i ) == len( ONOSFlowsJson[ 0 ] ) for i in ONOSFl
        if all( tmp ):
            main.log.report( "Flow count is consistent across all ONO
        else:
            consistentFlows = False
        utilities.assert_equals(
            expect=True,
            actual=consistentFlows,
            onpass="The flow count is consistent across all ONOS node
            onfail="ONOS nodes have different flow counts" )

        if flowsResults and not consistentFlows:
            for i in range( numControllers ):
                try:
                    main.log.warn(
                        "ONOS" + str( i + 1 ) + " flows: " +
                        json.dumps( json.loads( ONOSFlows[i] ), sort_
                                    indent=4, separators=( ',', ': '
                except ( ValueError, TypeError ):
                    main.log.warn(
```

```
                    "ONOS" + str( i + 1 ) + " flows: " +
                    repr( ONOSFlows[ i ] ) )
    elif flowsResults and consistentFlows:
        flowCheck = main.TRUE
        flowState = ONOSFlows[ 0 ]

    main.step( "Get the OF Table entries" )
    global flows
    flows = []
    for i in range( 1, 29 ):
        flows.append( main.Mininet2.getFlowTable( 1.0, "s" + str( i )
    if flowCheck == main.FALSE:
        for table in flows:
            main.log.warn( table )
    # TODO: Compare switch flow tables with ONOS flow tables

    main.step( "Start continuous pings" )
    main.Mininet2.pingLong(
        src=main.params[ 'PING' ][ 'source1' ],
        target=main.params[ 'PING' ][ 'target1' ],
        pingTime=500 )
    main.Mininet2.pingLong(
        src=main.params[ 'PING' ][ 'source2' ],
        target=main.params[ 'PING' ][ 'target2' ],
        pingTime=500 )
    main.Mininet2.pingLong(
        src=main.params[ 'PING' ][ 'source3' ],
        target=main.params[ 'PING' ][ 'target3' ],
        pingTime=500 )
    main.Mininet2.pingLong(
        src=main.params[ 'PING' ][ 'source4' ],
        target=main.params[ 'PING' ][ 'target4' ],
        pingTime=500 )
    main.Mininet2.pingLong(
        src=main.params[ 'PING' ][ 'source5' ],
        target=main.params[ 'PING' ][ 'target5' ],
        pingTime=500 )
    main.Mininet2.pingLong(
        src=main.params[ 'PING' ][ 'source6' ],
```

```
        target=main.params[ 'PING' ][ 'target6' ],
        pingTime=500 )
    main.Mininet2.pingLong(
        src=main.params[ 'PING' ][ 'source7' ],
        target=main.params[ 'PING' ][ 'target7' ],
        pingTime=500 )
    main.Mininet2.pingLong(
        src=main.params[ 'PING' ][ 'source8' ],
        target=main.params[ 'PING' ][ 'target8' ],
        pingTime=500 )
    main.Mininet2.pingLong(
        src=main.params[ 'PING' ][ 'source9' ],
        target=main.params[ 'PING' ][ 'target9' ],
        pingTime=500 )
    main.Mininet2.pingLong(
        src=main.params[ 'PING' ][ 'source10' ],
        target=main.params[ 'PING' ][ 'target10' ],
        pingTime=500 )

    main.step( "Create TestONTopology object" )
    ctrls = []
    for node in nodes:
        temp = ( node, node.name, node.ip_address, 6633 )
        ctrls.append( temp )
    MNTopo = TestONTopology( main.Mininet1, ctrls )

    main.step( "Collecting topology information from ONOS" )
    devices = []
    threads = []
    for i in range( numControllers ):
        t = main.Thread( target=CLIs[i].devices,
                         name="devices-" + str( i ),
                         args=[ ] )
        threads.append( t )
        t.start()

    for t in threads:
        t.join()
        devices.append( t.result )
```

```
hosts = []
threads = []
for i in range( numControllers ):
    t = main.Thread( target=CLIs[i].hosts,
                     name="hosts-" + str( i ),
                     args=[ ] )
    threads.append( t )
    t.start()

for t in threads:
    t.join()
    try:
        hosts.append( json.loads( t.result ) )
    except ( ValueError, TypeError ):
        # FIXME: better handling of this, print which node
        #        Maybe use thread name?
        main.log.exception( "Error parsing json output of hosts"
        # FIXME: should this be an empty json object instead?
        hosts.append( None )

ports = []
threads = []
for i in range( numControllers ):
    t = main.Thread( target=CLIs[i].ports,
                     name="ports-" + str( i ),
                     args=[ ] )
    threads.append( t )
    t.start()

for t in threads:
    t.join()
    ports.append( t.result )
links = []
threads = []
for i in range( numControllers ):
    t = main.Thread( target=CLIs[i].links,
                     name="links-" + str( i ),
                     args=[ ] )
    threads.append( t )
```

```
        t.start()

    for t in threads:
        t.join()
        links.append( t.result )
    clusters = []
    threads = []
    for i in range( numControllers ):
        t = main.Thread( target=CLIs[i].clusters,
                         name="clusters-" + str( i ),
                         args=[ ] )
        threads.append( t )
        t.start()

    for t in threads:
        t.join()
        clusters.append( t.result )
    # Compare json objects for hosts and dataplane clusters

    # hosts
    consistentHostsResult = main.TRUE
    for controller in range( len( hosts ) ):
        controllerStr = str( controller + 1 )
        if "Error" not in hosts[ controller ]:
            if hosts[ controller ] == hosts[ 0 ]:
                continue
            else:  # hosts not consistent
                main.log.report( "hosts from ONOS" +
                                 controllerStr +
                                 " is inconsistent with ONOS1" )
                main.log.warn( repr( hosts[ controller ] ) )
                consistentHostsResult = main.FALSE

        else:
            main.log.report( "Error in getting ONOS hosts from ON
                             controllerStr )
            consistentHostsResult = main.FALSE
            main.log.warn( "ONOS" + controllerStr +
                           " hosts response: " +
```

```
                        repr( hosts[ controller ] ) )
    utilities.assert_equals(
        expect=main.TRUE,
        actual=consistentHostsResult,
        onpass="Hosts view is consistent across all ONOS nodes",
        onfail="ONOS nodes have different views of hosts" )

    ipResult = main.TRUE
    for controller in range( 0, len( hosts ) ):
        controllerStr = str( controller + 1 )
        for host in hosts[ controller ]:
            if not host.get( 'ips', [ ] ):
                main.log.error( "DEBUG:Error with host ips on control
                                controllerStr + ": " + str( host ) )
                ipResult = main.FALSE
    utilities.assert_equals(
        expect=main.TRUE,
        actual=ipResult,
        onpass="The ips of the hosts aren't empty",
        onfail="The ip of at least one host is missing" )

    # Strongly connected clusters of devices
    consistentClustersResult = main.TRUE
    for controller in range( len( clusters ) ):
        controllerStr = str( controller + 1 )
        if "Error" not in clusters[ controller ]:
            if clusters[ controller ] == clusters[ 0 ]:
                continue
            else:  # clusters not consistent
                main.log.report( "clusters from ONOS" + controllerStr
                                 " is inconsistent with ONOS1" )
                consistentClustersResult = main.FALSE

        else:
            main.log.report( "Error in getting dataplane clusters " +
                             "from ONOS" + controllerStr )
            consistentClustersResult = main.FALSE
            main.log.warn( "ONOS" + controllerStr +
                           " clusters response: " +
```

```
                            repr( clusters[ controller ] ) )
    utilities.assert_equals(
        expect=main.TRUE,
        actual=consistentClustersResult,
        onpass="Clusters view is consistent across all ONOS nodes
        onfail="ONOS nodes have different views of clusters" )
    # there should always only be one cluster
    try:
        numClusters = len( json.loads( clusters[ 0 ] ) )
    except ( ValueError, TypeError ):
        main.log.exception( "Error parsing clusters[0]: " +
                            repr( clusters[ 0 ] ) )
    clusterResults = main.FALSE
    if numClusters == 1:
        clusterResults = main.TRUE
    utilities.assert_equals(
        expect=1,
        actual=numClusters,
        onpass="ONOS shows 1 SCC",
        onfail="ONOS shows " + str( numClusters ) + " SCCs" )

    main.step( "Comparing ONOS topology to MN" )
    devicesResults = main.TRUE
    portsResults = main.TRUE
    linksResults = main.TRUE
    for controller in range( numControllers ):
        controllerStr = str( controller + 1 )
        if devices[ controller ] or "Error" not in devices[ contr
            currentDevicesResult = main.Mininet1.compareSwitches(
                MNTopo,
                json.loads( devices[ controller ] ) )
        else:
            currentDevicesResult = main.FALSE
        utilities.assert_equals( expect=main.TRUE,
                                 actual=currentDevicesResult,
                                 onpass="ONOS" + controllerStr +
                                 " Switches view is correct",
                                 onfail="ONOS" + controllerStr +
                                 " Switches view is incorrect" )
```

```
        if ports[ controller ] or "Error" not in ports[ controller ]:
            currentPortsResult = main.Mininet1.comparePorts(
                MNTopo,
                json.loads( ports[ controller ] ) )
        else:
            currentPortsResult = main.FALSE
        utilities.assert_equals( expect=main.TRUE,
                                 actual=currentPortsResult,
                                 onpass="ONOS" + controllerStr +
                                 " ports view is correct",
                                 onfail="ONOS" + controllerStr +
                                 " ports view is incorrect" )

        if links[ controller ] or "Error" not in links[ controller ]:
            currentLinksResult = main.Mininet1.compareLinks(
                MNTopo,
                json.loads( links[ controller ] ) )
        else:
            currentLinksResult = main.FALSE
        utilities.assert_equals( expect=main.TRUE,
                                 actual=currentLinksResult,
                                 onpass="ONOS" + controllerStr +
                                 " links view is correct",
                                 onfail="ONOS" + controllerStr +
                                 " links view is incorrect" )

        devicesResults = devicesResults and currentDevicesResult
        portsResults = portsResults and currentPortsResult
        linksResults = linksResults and currentLinksResult

    topoResult = ( devicesResults and portsResults and linksResults
                   and consistentHostsResult and consistentClustersRe
                   and clusterResults and ipResult )
    utilities.assert_equals( expect=main.TRUE, actual=topoResult,
                             onpass="Topology Check Test successful",
                             onfail="Topology Check Test NOT successf

    finalAssert = main.TRUE
```

```
        finalAssert = ( finalAssert and topoResult and flowCheck
                        and intentCheck and consistentMastership
                        and mastershipCheck and rolesNotNull )
        utilities.assert_equals( expect=main.TRUE, actual=finalAssert
                                 onpass="State check successful",
                                 onfail="State check NOT successful"

    def CASE6( self, main ):
        """
        The Failure case.
        """
        import time
        assert numControllers, "numControllers not defined"
        assert main, "main not defined"
        assert utilities.assert_equals, "utilities.assert_equals not
        assert CLIs, "CLIs not defined"
        assert nodes, "nodes not defined"
        main.log.report( "Killing 3 ONOS nodes" )
        main.case( "Restart minority of ONOS nodes" )
        # TODO: Randomize these nodes
        # TODO: use threads in this case
        main.ONOSbench.onosKill( nodes[0].ip_address )
        time.sleep( 10 )
        main.ONOSbench.onosKill( nodes[1].ip_address )
        time.sleep( 10 )
        main.ONOSbench.onosKill( nodes[2].ip_address )

        main.step( "Checking if ONOS is up yet" )
        count = 0
        onosIsupResult = main.FALSE
        while onosIsupResult == main.FALSE and count < 10:
            onos1Isup = main.ONOSbench.isup( nodes[0].ip_address )
            onos2Isup = main.ONOSbench.isup( nodes[1].ip_address )
            onos3Isup = main.ONOSbench.isup( nodes[2].ip_address )
            onosIsupResult = onos1Isup and onos2Isup and onos3Isup
            count = count + 1
        # TODO: if it becomes an issue, we can retry this step
a few times
```

```
        cliResult1 = main.ONOScli1.startOnosCli( nodes[0].ip_address )
        cliResult2 = main.ONOScli2.startOnosCli( nodes[1].ip_address )
        cliResult3 = main.ONOScli3.startOnosCli( nodes[2].ip_address )
        cliResults = cliResult1 and cliResult2 and cliResult3

        # Grab the time of restart so we chan check how long the gossip
        # protocol has had time to work
        main.restartTime = time.time()
        caseResults = main.TRUE and onosIsupResult and cliResults
        utilities.assert_equals( expect=main.TRUE, actual=caseResults,
                                 onpass="ONOS restart successful",
                                 onfail="ONOS restart NOT successful" )

    def CASE7( self, main ):
        """
        Check state after ONOS failure
        """
        import json
        assert numControllers, "numControllers not defined"
        assert main, "main not defined"
        assert utilities.assert_equals, "utilities.assert_equals not defi
        assert CLIs, "CLIs not defined"
        assert nodes, "nodes not defined"
        main.case( "Running ONOS Constant State Tests" )

        main.step( "Check that each switch has a master" )
        # Assert that each device has a master
        rolesNotNull = main.TRUE
        threads = []
        for i in range( numControllers ):
            t = main.Thread( target=CLIs[i].rolesNotNull,
                             name="rolesNotNull-" + str( i ),
                             args=[ ] )
            threads.append( t )
            t.start()

        for t in threads:
            t.join()
            rolesNotNull = rolesNotNull and t.result
```

```
        utilities.assert_equals(
            expect=main.TRUE,
            actual=rolesNotNull,
            onpass="Each device has a master",
            onfail="Some devices don't have a master assigned" )

        ONOSMastership = []
        mastershipCheck = main.FALSE
        consistentMastership = True
        rolesResults = True
        threads = []
        for i in range( numControllers ):
            t = main.Thread( target=CLIs[i].roles,
                             name="roles-" + str( i ),
                             args=[] )
            threads.append( t )
            t.start()

        for t in threads:
            t.join()
            ONOSMastership.append( t.result )

        for i in range( numControllers ):
            if not ONOSMastership[i] or "Error" in ONOSMastership[i]:
                main.log.report( "Error in getting ONOS" + str( i + 1
                                 " roles" )
                main.log.warn(
                    "ONOS" + str( i + 1 ) + " mastership response: "
                    repr( ONOSMastership[i] ) )
                rolesResults = False
        utilities.assert_equals(
            expect=True,
            actual=rolesResults,
            onpass="No error in reading roles output",
            onfail="Error in reading roles from ONOS" )

        main.step( "Check for consistency in roles from each controll
        if all([ i == ONOSMastership[ 0 ] for i in ONOSMastership ] )
            main.log.report(
```

```
            "Switch roles are consistent across all ONOS nodes" )
    else:
        consistentMastership = False
    utilities.assert_equals(
        expect=True,
        actual=consistentMastership,
        onpass="Switch roles are consistent across all ONOS nodes",
        onfail="ONOS nodes have different views of switch roles" )

    if rolesResults and not consistentMastership:
        for i in range( numControllers ):
            main.log.warn(
                "ONOS" + str( i + 1 ) + " roles: ",
                json.dumps(
                    json.loads( ONOSMastership[ i ] ),
                    sort_keys=True,
                    indent=4,
                    separators=( ',', ': ' ) ) )
    elif rolesResults and not consistentMastership:
        mastershipCheck = main.TRUE

    description2 = "Compare switch roles from before failure"
    main.step( description2 )
    try:
        currentJson = json.loads( ONOSMastership[0] )
        oldJson = json.loads( mastershipState )
    except ( ValueError, TypeError ):
        main.log.exception( "Something is wrong with parsing " +
                            "ONOSMastership[0] or mastershipState" )
        main.log.error( "ONOSMastership[0]: " + repr( ONOSMastership[
        main.log.error( "mastershipState" + repr( mastershipState ) )
        main.cleanup()
        main.exit()
    mastershipCheck = main.TRUE
    for i in range( 1, 29 ):
        switchDPID = str(
            main.Mininet1.getSwitchDPID( switch="s" + str( i ) ) )
        current = [ switch[ 'master' ] for switch in currentJson
                    if switchDPID in switch[ 'id' ] ]
```

```
        old = [ switch[ 'master' ] for switch in oldJson
                if switchDPID in switch[ 'id' ] ]
        if current == old:
            mastershipCheck = mastershipCheck and main.TRUE
        else:
            main.log.warn( "Mastership of switch %s changed" % sw
            mastershipCheck = main.FALSE
    if mastershipCheck == main.TRUE:
        main.log.report( "Mastership of Switches was not changed"
    utilities.assert_equals(
        expect=main.TRUE,
        actual=mastershipCheck,
        onpass="Mastership of Switches was not changed",
        onfail="Mastership of some switches changed" )
    # NOTE: we expect mastership to change on controller failure
    mastershipCheck = consistentMastership

    main.step( "Get the intents and compare across all nodes" )
    ONOSIntents = []
    intentCheck = main.FALSE
    consistentIntents = True
    intentsResults = True
    threads = []
    for i in range( numControllers ):
        t = main.Thread( target=CLIs[i].intents,
                         name="intents-" + str( i ),
                         args=[],
                         kwargs={ 'jsonFormat': True } )
        threads.append( t )
        t.start()

    for t in threads:
        t.join()
        ONOSIntents.append( t.result )

    for i in range( numControllers ):
        if not ONOSIntents[ i ] or "Error" in ONOSIntents[ i ]:
            main.log.report( "Error in getting ONOS" + str( i + 1
                             " intents" )
```

```
            main.log.warn( "ONOS" + str( i + 1 ) + " intents response
                           repr( ONOSIntents[ i ] ) )
            intentsResults = False
    utilities.assert_equals(
        expect=True,
        actual=intentsResults,
        onpass="No error in reading intents output",
        onfail="Error in reading intents from ONOS" )

    main.step( "Check for consistency in Intents from each controller
    if all([ sorted( i ) == sorted( ONOSIntents[ 0 ] ) for i in ONOSI
        main.log.report( "Intents are consistent across all ONOS " +
                         "nodes" )
    else:
        consistentIntents = False
    utilities.assert_equals(
        expect=True,
        actual=consistentIntents,
        onpass="Intents are consistent across all ONOS nodes",
        onfail="ONOS nodes have different views of intents" )
    intentStates = []
    for node in ONOSIntents:  # Iter through ONOS nodes
        nodeStates = []
        # Iter through intents of a node
        try:
            for intent in json.loads( node ):
                nodeStates.append( intent[ 'state' ] )
        except ( ValueError, TypeError ):
            main.log.exception( "Error in parsing intents" )
            main.log.error( repr( node ) )
        intentStates.append( nodeStates )
        out = [ (i, nodeStates.count( i ) ) for i in set( nodeStates
        main.log.info( dict( out ) )

    if intentsResults and not consistentIntents:
        for i in range( numControllers ):
            main.log.warn( "ONOS" + str( i + 1 ) + " intents: " )
            main.log.warn( json.dumps(
                json.loads( ONOSIntents[ i ] ),
```

```
                sort_keys=True,
                indent=4,
                separators=( ',', ': ' ) ) )
    elif intentsResults and consistentIntents:
        intentCheck = main.TRUE

    # NOTE: Store has no durability, so intents are lost across s
    #       restarts
    main.step( "Compare current intents with intents before the f
    # NOTE: this requires case 5 to pass for intentState to be se
    #      maybe we should stop the test if that fails?
    sameIntents = main.TRUE
    if intentState and intentState == ONOSIntents[ 0 ]:
        sameIntents = main.TRUE
        main.log.report( "Intents are consistent with before fail
    # TODO: possibly the states have changed? we may need to figu
    #       what the acceptable states are
    else:
        try:
            main.log.warn( "ONOS intents: " )
            main.log.warn( json.dumps( json.loads( ONOSIntents[ 0
                                       sort_keys=True, indent=4,
                                       separators=( ',', ': ' ) )
        except ( ValueError, TypeError ):
            main.log.exception( "Exception printing intents" )
            main.log.warn( repr( ONOSIntents[0] ) )
        sameIntents = main.FALSE
    utilities.assert_equals(
        expect=main.TRUE,
        actual=sameIntents,
        onpass="Intents are consistent with before failure",
        onfail="The Intents changed during failure" )
    intentCheck = intentCheck and sameIntents

    main.step( "Get the OF Table entries and compare to before "
               "component failure" )
    FlowTables = main.TRUE
    flows2 = []
    for i in range( 28 ):
```

```
        main.log.info( "Checking flow table on s" + str( i + 1 ) )
        tmpFlows = main.Mininet2.getFlowTable( 1.0, "s" + str( i + 1
        flows2.append( tmpFlows )
        tempResult = main.Mininet2.flowComp(
            flow1=flows[ i ],
            flow2=tmpFlows )
        FlowTables = FlowTables and tempResult
        if FlowTables == main.FALSE:
            main.log.info( "Differences in flow table for switch: s"
                           str( i + 1 ) )
    if FlowTables == main.TRUE:
        main.log.report( "No changes were found in the flow tables" )
    utilities.assert_equals(
        expect=main.TRUE,
        actual=FlowTables,
        onpass="No changes were found in the flow tables",
        onfail="Changes were found in the flow tables" )

    main.step( "Check the continuous pings to ensure that no packets
               "were dropped during component failure" )
    main.Mininet2.pingKill( main.params[ 'TESTONUSER' ],
                            main.params[ 'TESTONIP' ] )
    LossInPings = main.FALSE
    # NOTE: checkForLoss returns main.FALSE with 0% packet loss
    for i in range( 8, 18 ):
        main.log.info(
            "Checking for a loss in pings along flow from s" +
            str( i ) )
        LossInPings = main.Mininet2.checkForLoss(
            "/tmp/ping.h" +
            str( i ) ) or LossInPings
    if LossInPings == main.TRUE:
        main.log.info( "Loss in ping detected" )
    elif LossInPings == main.ERROR:
        main.log.info( "There are multiple mininet process running" )
    elif LossInPings == main.FALSE:
        main.log.info( "No Loss in the pings" )
        main.log.report( "No loss of dataplane connectivity" )
    utilities.assert_equals(
```

```
        expect=main.FALSE,
        actual=LossInPings,
        onpass="No Loss of connectivity",
        onfail="Loss of dataplane connectivity detected" )

    # Test of LeadershipElection
    leaderList = []
    # FIXME: make sure this matches nodes that were restarted
    restarted = [ nodes[0].ip_address, nodes[1].ip_address,
                  nodes[2].ip_address ]

    leaderResult = main.TRUE
    for cli in CLIs:
        leaderN = cli.electionTestLeader()
        leaderList.append( leaderN )
        if leaderN == main.FALSE:
            # error in  response
            main.log.report( "Something is wrong with " +
                             "electionTestLeader function, check
                             " error logs" )
            leaderResult = main.FALSE
        elif leaderN is None:
            main.log.report( cli.name +
                             " shows no leader for the election-a
                             " elected after the old one died" )
            leaderResult = main.FALSE
        elif leaderN in restarted:
            main.log.report( cli.name + " shows " + str( leaderN
                             " as leader for the election-app, bu
                             "was restarted" )
            leaderResult = main.FALSE
    if len( set( leaderList ) ) != 1:
        leaderResult = main.FALSE
        main.log.error(
            "Inconsistent view of leader for the election test ap
        # TODO: print the list
    if leaderResult:
        main.log.report( "Leadership election tests passed( consi
                         "view of leader across listeners and a n
```

```
                        "leader was re-elected if applicable )" )
        utilities.assert_equals(
            expect=main.TRUE,
            actual=leaderResult,
            onpass="Leadership election passed",
            onfail="Something went wrong with Leadership election" )

        result = ( mastershipCheck and intentCheck and FlowTables and
                   ( not LossInPings ) and rolesNotNull and leaderResult
        result = int( result )
        if result == main.TRUE:
            main.log.report( "Constant State Tests Passed" )
        utilities.assert_equals( expect=main.TRUE, actual=result,
                                 onpass="Constant State Tests Passed",
                                 onfail="Constant state tests failed" )

    def CASE8( self, main ):
        """
        Compare topo
        """
        import sys
        # FIXME add this path to params
        sys.path.append( "/home/admin/sts" )
        # assumes that sts is already in you PYTHONPATH
        from sts.topology.teston_topology import TestONTopology
        import json
        import time
        assert numControllers, "numControllers not defined"
        assert main, "main not defined"
        assert utilities.assert_equals, "utilities.assert_equals not defi
        assert CLIs, "CLIs not defined"
        assert nodes, "nodes not defined"

        description = "Compare ONOS Topology view to Mininet topology"
        main.case( description )
        main.log.report( description )
        main.step( "Create TestONTopology object" )
        ctrls = []
        for node in nodes:
```

```
        temp = ( node, node.name, node.ip_address, 6633 )
        ctrls.append( temp )
    MNTopo = TestONTopology( main.Mininet1, ctrls )

    main.step( "Comparing ONOS topology to MN" )
    devicesResults = main.TRUE
    portsResults = main.TRUE
    linksResults = main.TRUE
    hostsResults = main.TRUE
    topoResult = main.FALSE
    elapsed = 0
    count = 0
    main.step( "Collecting topology information from ONOS" )
    startTime = time.time()
    # Give time for Gossip to work
    while topoResult == main.FALSE and elapsed < 60:
        count += 1
        if count > 1:
            # TODO: Deprecate STS usage
            MNTopo = TestONTopology( main.Mininet1, ctrls )
        cliStart = time.time()
        devices = []
        threads = []
        for i in range( numControllers ):
            t = main.Thread( target=CLIs[i].devices,
                             name="devices-" + str( i ),
                             args=[ ] )
            threads.append( t )
            t.start()

        for t in threads:
            t.join()
            devices.append( t.result )
        hosts = []
        ipResult = main.TRUE
        threads = []
        for i in range( numControllers ):
            t = main.Thread( target=CLIs[i].hosts,
                             name="hosts-" + str( i ),
```

```
                        args=[ ] )
    threads.append( t )
    t.start()

for t in threads:
    t.join()
    try:
        hosts.append( json.loads( t.result ) )
    except ( ValueError, TypeError ):
        main.log.exception( "Error parsing hosts results" )
        main.log.error( repr( t.result ) )
for controller in range( 0, len( hosts ) ):
    controllerStr = str( controller + 1 )
    for host in hosts[ controller ]:
        if host is None or host.get( 'ips', [] ) == []:
            main.log.error(
                "DEBUG:Error with host ips on controller" +
                controllerStr + ": " + str( host ) )
            ipResult = main.FALSE
ports = []
threads = []
for i in range( numControllers ):
    t = main.Thread( target=CLIs[i].ports,
                     name="ports-" + str( i ),
                     args=[ ] )
    threads.append( t )
    t.start()

for t in threads:
    t.join()
    ports.append( t.result )
links = []
threads = []
for i in range( numControllers ):
    t = main.Thread( target=CLIs[i].links,
                     name="links-" + str( i ),
                     args=[ ] )
    threads.append( t )
    t.start()
```

```
for t in threads:
    t.join()
    links.append( t.result )
clusters = []
threads = []
for i in range( numControllers ):
    t = main.Thread( target=CLIs[i].clusters,
                     name="clusters-" + str( i ),
                     args=[ ] )
    threads.append( t )
    t.start()

for t in threads:
    t.join()
    clusters.append( t.result )

elapsed = time.time() - startTime
cliTime = time.time() - cliStart
print "CLI time: " + str( cliTime )

for controller in range( numControllers ):
    controllerStr = str( controller + 1 )
    if devices[ controller ] or "Error" not in devices[
            controller ]:
        currentDevicesResult = main.Mininet1.compareSwitc
            MNTopo,
            json.loads( devices[ controller ] ) )
    else:
        currentDevicesResult = main.FALSE
    utilities.assert_equals( expect=main.TRUE,
                             actual=currentDevicesResult,
                             onpass="ONOS" + controllerSt
                             " Switches view is correct",
                             onfail="ONOS" + controllerSt
                             " Switches view is incorrect

    if ports[ controller ] or "Error" not in ports[ contr
        currentPortsResult = main.Mininet1.comparePorts(
```

```
            MNTopo,
            json.loads( ports[ controller ] ) )
    else:
        currentPortsResult = main.FALSE
    utilities.assert_equals( expect=main.TRUE,
                             actual=currentPortsResult,
                             onpass="ONOS" + controllerStr +
                             " ports view is correct",
                             onfail="ONOS" + controllerStr +
                             " ports view is incorrect" )

    if links[ controller ] or "Error" not in links[ controlle
        currentLinksResult = main.Mininet1.compareLinks(
            MNTopo,
            json.loads( links[ controller ] ) )
    else:
        currentLinksResult = main.FALSE
    utilities.assert_equals( expect=main.TRUE,
                             actual=currentLinksResult,
                             onpass="ONOS" + controllerStr +
                             " links view is correct",
                             onfail="ONOS" + controllerStr +
                             " links view is incorrect" )

    if hosts[ controller ] or "Error" not in hosts[ controlle
        currentHostsResult = main.Mininet1.compareHosts(
            MNTopo, hosts[ controller ] )
    else:
        currentHostsResult = main.FALSE
    utilities.assert_equals( expect=main.TRUE,
                             actual=currentHostsResult,
                             onpass="ONOS" + controllerStr +
                             " hosts exist in Mininet",
                             onfail="ONOS" + controllerStr +
                             " hosts don't match Mininet" )

    devicesResults = devicesResults and currentDevicesResult
    portsResults = portsResults and currentPortsResult
    linksResults = linksResults and currentLinksResult
```

```
        hostsResults = hostsResults and currentHostsResult

    # Compare json objects for hosts and dataplane clusters

    # hosts
    consistentHostsResult = main.TRUE
    for controller in range( len( hosts ) ):
        controllerStr = str( controller + 1 )
        if "Error" not in hosts[ controller ]:
            if hosts[ controller ] == hosts[ 0 ]:
                continue
            else:  # hosts not consistent
                main.log.report( "hosts from ONOS" + controll
                                 " is inconsistent with ONOS1
                main.log.warn( repr( hosts[ controller ] ) )
                consistentHostsResult = main.FALSE

        else:
            main.log.report( "Error in getting ONOS hosts fro
                             controllerStr )
            consistentHostsResult = main.FALSE
            main.log.warn( "ONOS" + controllerStr +
                           " hosts response: " +
                           repr( hosts[ controller ] ) )
    utilities.assert_equals(
        expect=main.TRUE,
        actual=consistentHostsResult,
        onpass="Hosts view is consistent across all ONOS node
        onfail="ONOS nodes have different views of hosts" )

    # Strongly connected clusters of devices
    consistentClustersResult = main.TRUE
    for controller in range( len( clusters ) ):
        controllerStr = str( controller + 1 )
        if "Error" not in clusters[ controller ]:
            if clusters[ controller ] == clusters[ 0 ]:
                continue
            else:  # clusters not consistent
                main.log.report( "clusters from ONOS" +
```

```
                                        controllerStr +
                                        " is inconsistent with ONOS1" )
                        consistentClustersResult = main.FALSE

                else:
                    main.log.report( "Error in getting dataplane clusters
                                     "from ONOS" + controllerStr )
                    consistentClustersResult = main.FALSE
                    main.log.warn( "ONOS" + controllerStr +
                                   " clusters response: " +
                                   repr( clusters[ controller ] ) )
            utilities.assert_equals(
                expect=main.TRUE,
                actual=consistentClustersResult,
                onpass="Clusters view is consistent across all ONOS nodes
                onfail="ONOS nodes have different views of clusters" )
            # there should always only be one cluster
            try:
                numClusters = len( json.loads( clusters[ 0 ] ) )
            except ( ValueError, TypeError ):
                main.log.exception( "Error parsing clusters[0]: " +
                                    repr( clusters[0] ) )
            clusterResults = main.FALSE
            if numClusters == 1:
                clusterResults = main.TRUE
            utilities.assert_equals(
                expect=1,
                actual=numClusters,
                onpass="ONOS shows 1 SCC",
                onfail="ONOS shows " + str( numClusters ) + " SCCs" )

            topoResult = ( devicesResults and portsResults and linksResul
                           and hostsResults and consistentHostsResult
                           and consistentClustersResult and clusterResult
                           and ipResult )

        topoResult = topoResult and int( count <= 2 )
        note = "note it takes about " + str( int( cliTime ) ) + \
            " seconds for the test to make all the cli calls to fetch " +
```

```
        "the topology from each ONOS instance"
    main.log.info(
        "Very crass estimate for topology discovery/convergence(
        str( note ) + " ): " + str( elapsed ) + " seconds, " +
        str( count ) + " tries" )
    utilities.assert_equals( expect=main.TRUE, actual=topoResult,
                             onpass="Topology Check Test successf
                             onfail="Topology Check Test NOT succ
    if topoResult == main.TRUE:
        main.log.report( "ONOS topology view matches Mininet topo

    # FIXME: move this to an ONOS state case
    main.step( "Checking ONOS nodes" )
    nodesOutput = []
    threads = []
    for i in range( numControllers ):
        t = main.Thread( target=CLIs[i].nodes,
                         name="nodes-" + str( i ),
                         args=[ ] )
        threads.append( t )
        t.start()

    for t in threads:
        t.join()
        nodesOutput.append( t.result )
    ips = [ node.ip_address for node in nodes ]
    for i in nodesOutput:
        try:
            current = json.loads( i )
            for node in current:
                if node['ip'] in ips:  # node in nodes() output i
                    if node['state'] == 'ACTIVE':
                        pass  # as it should be
                    else:
                        main.log.error( "Error in ONOS node avail
                        main.log.error(
                                json.dumps( current,
                                            sort_keys=True,
                                            indent=4,
```

```
                                                separators=( ',', ': ' )
                        break
            except ( ValueError, TypeError ):
                main.log.error( "Error parsing nodes output" )
                main.log.warn( repr( i ) )

    def CASE9( self, main ):
        """
        Link s3-s28 down
        """
        import time
        assert numControllers, "numControllers not defined"
        assert main, "main not defined"
        assert utilities.assert_equals, "utilities.assert_equals not defi
        assert CLIs, "CLIs not defined"
        assert nodes, "nodes not defined"
        # NOTE: You should probably run a topology check after this

        linkSleep = float( main.params[ 'timers' ][ 'LinkDiscovery' ] )

        description = "Turn off a link to ensure that Link Discovery " +\
                      "is working properly"
        main.log.report( description )
        main.case( description )

        main.step( "Kill Link between s3 and s28" )
        LinkDown = main.Mininet1.link( END1="s3", END2="s28", OPTION="dow
        main.log.info( "Waiting " + str( linkSleep ) +
                       " seconds for link down to be discovered" )
        time.sleep( linkSleep )
        utilities.assert_equals( expect=main.TRUE, actual=LinkDown,
                                 onpass="Link down successful",
                                 onfail="Failed to bring link down" )
        # TODO do some sort of check here

    def CASE10( self, main ):
        """
        Link s3-s28 up
        """
```

```
        import time
        assert numControllers, "numControllers not defined"
        assert main, "main not defined"
        assert utilities.assert_equals, "utilities.assert_equals not
        assert CLIs, "CLIs not defined"
        assert nodes, "nodes not defined"
        # NOTE: You should probably run a topology check after this

        linkSleep = float( main.params[ 'timers' ][ 'LinkDiscovery' ]

        description = "Restore a link to ensure that Link Discovery i
                      "working properly"
        main.log.report( description )
        main.case( description )

        main.step( "Bring link between s3 and s28 back up" )
        LinkUp = main.Mininet1.link( END1="s3", END2="s28", OPTION="u
        main.log.info( "Waiting " + str( linkSleep ) +
                       " seconds for link up to be discovered" )
        time.sleep( linkSleep )
        utilities.assert_equals( expect=main.TRUE, actual=LinkUp,
                                 onpass="Link up successful",
                                 onfail="Failed to bring link up" )
        # TODO do some sort of check here

    def CASE11( self, main ):
        """
        Switch Down
        """
        # NOTE: You should probably run a topology check after this
        import time
        assert numControllers, "numControllers not defined"
        assert main, "main not defined"
        assert utilities.assert_equals, "utilities.assert_equals not
        assert CLIs, "CLIs not defined"
        assert nodes, "nodes not defined"

        switchSleep = float( main.params[ 'timers' ][ 'SwitchDiscover
```

```
        description = "Killing a switch to ensure it is discovered correc
        main.log.report( description )
        main.case( description )
        switch = main.params[ 'kill' ][ 'switch' ]
        switchDPID = main.params[ 'kill' ][ 'dpid' ]

        # TODO: Make this switch parameterizable
        main.step( "Kill " + switch )
        main.log.report( "Deleting " + switch )
        main.Mininet1.delSwitch( switch )
        main.log.info( "Waiting " + str( switchSleep ) +
                       " seconds for switch down to be discovered" )
        time.sleep( switchSleep )
        device = main.ONOScli1.getDevice( dpid=switchDPID )
        # Peek at the deleted switch
        main.log.warn( str( device ) )
        result = main.FALSE
        if device and device[ 'available' ] is False:
            result = main.TRUE
        utilities.assert_equals( expect=main.TRUE, actual=result,
                                 onpass="Kill switch successful",
                                 onfail="Failed to kill switch?" )

    def CASE12( self, main ):
        """
        Switch Up
        """
        # NOTE: You should probably run a topology check after this
        import time
        assert numControllers, "numControllers not defined"
        assert main, "main not defined"
        assert utilities.assert_equals, "utilities.assert_equals not defi
        assert CLIs, "CLIs not defined"
        assert nodes, "nodes not defined"
        assert ONOS1Port, "ONOS1Port not defined"
        assert ONOS2Port, "ONOS2Port not defined"
        assert ONOS3Port, "ONOS3Port not defined"
        assert ONOS4Port, "ONOS4Port not defined"
        assert ONOS5Port, "ONOS5Port not defined"
```

```
        assert ONOS6Port, "ONOS6Port not defined"
        assert ONOS7Port, "ONOS7Port not defined"

        switchSleep = float( main.params[ 'timers' ][ 'SwitchDiscovery
        switch = main.params[ 'kill' ][ 'switch' ]
        switchDPID = main.params[ 'kill' ][ 'dpid' ]
        links = main.params[ 'kill' ][ 'links' ].split()
        description = "Adding a switch to ensure it is discovered cor
        main.log.report( description )
        main.case( description )

        main.step( "Add back " + switch )
        main.log.report( "Adding back " + switch )
        main.Mininet1.addSwitch( switch, dpid=switchDPID )
        for peer in links:
            main.Mininet1.addLink( switch, peer )
        main.Mininet1.assignSwController( sw=switch.split( 's' )[ 1 ]
                                          count=numControllers,
                                          ip1=nodes[ 0 ].ip_address,
                                          port1=ONOS1Port,
                                          ip2=nodes[ 1 ].ip_address,
                                          port2=ONOS2Port,
                                          ip3=nodes[ 2 ].ip_address,
                                          port3=ONOS3Port,
                                          ip4=nodes[ 3 ].ip_address,
                                          port4=ONOS4Port,
                                          ip5=nodes[ 4 ].ip_address,
                                          port5=ONOS5Port,
                                          ip6=nodes[ 5 ].ip_address,
                                          port6=ONOS6Port,
                                          ip7=nodes[ 6 ].ip_address,
                                          port7=ONOS7Port )
        main.log.info( "Waiting " + str( switchSleep ) +
                       " seconds for switch up to be discovered" )
        time.sleep( switchSleep )
        device = main.ONOScli1.getDevice( dpid=switchDPID )
        # Peek at the deleted switch
        main.log.warn( str( device ) )
        result = main.FALSE
```

```
        if device and device[ 'available' ]:
            result = main.TRUE
        utilities.assert_equals( expect=main.TRUE, actual=result,
                                 onpass="add switch successful",
                                 onfail="Failed to add switch?" )

    def CASE13( self, main ):
        """
        Clean up
        """
        import os
        import time
        assert numControllers, "numControllers not defined"
        assert main, "main not defined"
        assert utilities.assert_equals, "utilities.assert_equals not defi
        assert CLIs, "CLIs not defined"
        assert nodes, "nodes not defined"

        # printing colors to terminal
        colors = { 'cyan': '\033[96m', 'purple': '\033[95m',
                   'blue': '\033[94m', 'green': '\033[92m',
                   'yellow': '\033[93m', 'red': '\033[91m', 'end': '\033[
        description = "Test Cleanup"
        main.log.report( description )
        main.case( description )
        main.step( "Killing tcpdumps" )
        main.Mininet2.stopTcpdump()

        main.step( "Copying MN pcap and ONOS log files to test station" )
        testname = main.TEST
        teststationUser = main.params[ 'TESTONUSER' ]
        teststationIP = main.params[ 'TESTONIP' ]
        nodeUser = 'ubuntu'
        # NOTE: MN Pcap file is being saved to ~/packet_captures
        #       scp this file as MN and TestON aren't necessarily the sam
        # FIXME: scp
        # mn files
        # TODO: Load these from params
        # NOTE: must end in /
```

```
logFolder = "/opt/onos/log/"
logFiles = [ "karaf.log", "karaf.log.1" ]
# NOTE: must end in /
dstDir = "~/packet_captures/"
for f in logFiles:
    for node in nodes:
        main.ONOSbench.handle.sendline( "scp " + nodeUser + "
                                        ":" + logFolder + f +
                                        teststationUser + "@"
                                        teststationIP + ":" +
                                        dstDir + str( testnam
                                        "-" + node.name + "-"
        main.ONOSbench.handle.expect( "\$" )

# std*.log's
# NOTE: must end in /
logFolder = "/opt/onos/var/"
logFiles = [ "stderr.log", "stdout.log" ]
# NOTE: must end in /
dstDir = "~/packet_captures/"
for f in logFiles:
    for node in nodes:
        main.ONOSbench.handle.sendline( "scp " + nodeUser + "
                                        ":" + logFolder + f +
                                        teststationUser + "@"
                                        teststationIP + ":" +
                                        dstDir + str( testnam
                                        "-" + node.name + "-"
        main.ONOSbench.handle.expect( "\$" )
# sleep so scp can finish
time.sleep( 10 )

main.step( "Stopping Mininet" )
main.Mininet1.stopNet()

main.step( "Checking ONOS Logs for errors" )
for node in nodes:
    print colors[ 'purple' ] + "Checking logs for errors on "
        node.name + ":" + colors[ 'end' ]
```

```
            print main.ONOSbench.checkLogs( node.ip_address )

        main.step( "Packing and rotating pcap archives" )
        os.system( "~/TestON/dependencies/rotate.sh " + str( testname ) )

        # TODO: actually check something here
        utilities.assert_equals( expect=main.TRUE, actual=main.TRUE,
                                 onpass="Test cleanup successful",
                                 onfail="Test cleanup NOT successful" )

    def CASE14( self, main ):
        """
        start election app on all onos nodes
        """
        import time
        assert numControllers, "numControllers not defined"
        assert main, "main not defined"
        assert utilities.assert_equals, "utilities.assert_equals not defi
        assert CLIs, "CLIs not defined"
        assert nodes, "nodes not defined"

        leaderResult = main.TRUE
        main.log.info( "Install leadership election app" )
        main.ONOScli1.activateApp( "org.onosproject.election" )
        leaders = []
        for cli in CLIs:
            leader = cli.electionTestLeader()
            if leader is None or leader == main.FALSE:
                main.log.report( cli.name + ": Leader for the election ap
                                 "should be an ONOS node, instead got '"
                                 str( leader ) + "'" )
                leaderResult = main.FALSE
            leaders.append( leader )
        if len( set( leaders ) ) != 1:
            leaderResult = main.FALSE
            main.log.error( "Results of electionTestLeader is order of CL
                            str( leaders ) )
        if leaderResult:
            main.log.report( "Leadership election tests passed( consisten
```

```
                                "view of leader across listeners and a l
                                "was elected )" )
        utilities.assert_equals(
            expect=main.TRUE,
            actual=leaderResult,
            onpass="Leadership election passed",
            onfail="Something went wrong with Leadership election" )

    def CASE15( self, main ):
        """
        Check that Leadership Election is still functional
        """
        assert numControllers, "numControllers not defined"
        assert main, "main not defined"
        assert utilities.assert_equals, "utilities.assert_equals not
        assert CLIs, "CLIs not defined"
        assert nodes, "nodes not defined"

        leaderResult = main.TRUE
        description = "Check that Leadership Election is still functi
        main.log.report( description )
        main.case( description )
        main.step( "Find current leader and withdraw" )
        leader = main.ONOScli1.electionTestLeader()
        # do some sanity checking on leader before using it
        withdrawResult = main.FALSE
        if leader is None or leader == main.FALSE:
            main.log.report(
                "Leader for the election app should be an ONOS node,"
                "instead got '" + str( leader ) + "'" )
            leaderResult = main.FALSE
            oldLeader = None
        for i in range( len( CLIs ) ):
            if leader == nodes[ i ].ip_address:
                oldLeader = CLIs[ i ]
                break
        else:  # FOR/ELSE statement
            main.log.error( "Leader election, could not find current
        if oldLeader:
```

```
        withdrawResult = oldLeader.electionTestWithdraw()
    utilities.assert_equals(
        expect=main.TRUE,
        actual=withdrawResult,
        onpass="App was withdrawn from election",
        onfail="App was not withdrawn from election" )

    main.step( "Make sure new leader is elected" )
    # FIXME: use threads
    leaderList = []
    for cli in CLIs:
        leaderN = cli.electionTestLeader()
        leaderList.append( leaderN )
        if leaderN == leader:
            main.log.report(  cli.name + " still sees " + str( leader
                              " as leader after they withdrew" )
            leaderResult = main.FALSE
        elif leaderN == main.FALSE:
            # error in  response
            # TODO: add check for "Command not found:" in the driver,
            #       means the app isn't loaded
            main.log.report( "Something is wrong with " +
                             "electionTestLeader function, " +
                             "check the error logs" )
            leaderResult = main.FALSE
        elif leaderN is None:
            # node may not have recieved the event yet
            leaderN = cli.electionTestLeader()
            leaderList.pop()
            leaderList.append( leaderN )
    consistentLeader = main.FALSE
    if len( set( leaderList ) ) == 1:
        main.log.info( "Each Election-app sees '" +
                       str( leaderList[ 0 ] ) +
                       "' as the leader" )
        consistentLeader = main.TRUE
    else:
        main.log.report(
            "Inconsistent responses for leader of Election-app:" )
```

```
        for n in range( len( leaderList ) ):
            main.log.report( "ONOS" + str( n + 1 ) + " response:
                             str( leaderList[ n ] ) )
    leaderResult = leaderResult and consistentLeader
    if leaderResult:
        main.log.report( "Leadership election tests passed( consi
                         "view of leader across listeners and a n
                         "leader was elected when the old leader
                         "resigned )" )
    utilities.assert_equals(
        expect=main.TRUE,
        actual=leaderResult,
        onpass="Leadership election passed",
        onfail="Something went wrong with Leadership election" )

    main.step( "Run for election on old leader( just so everyone
               "is in the hat )" )
    if oldLeader:
        runResult = oldLeader.electionTestRun()
    else:
        runResult = main.FALSE
    utilities.assert_equals(
        expect=main.TRUE,
        actual=runResult,
        onpass="App re-ran for election",
        onfail="App failed to run for election" )
    if consistentLeader == main.TRUE:
        afterRun = main.ONOScli1.electionTestLeader()
        # verify leader didn't just change
        if afterRun == leaderList[ 0 ]:
            leaderResult = main.TRUE
        else:
            leaderResult = main.FALSE
    # TODO: assert on  run and withdraw results?

    utilities.assert_equals(
        expect=main.TRUE,
        actual=leaderResult,
        onpass="Leadership election passed",
```

```
            onfail="Something went wrong with Leadership election after "
                   "the old leader re-ran for election" )
```

Arquivo da topologia HATestMinorityRestart.topo

```
<TOPOLOGY>
    <COMPONENT>

        <ONOSbench>
            <host>127.0.0.1</host>
            <user>mininet</user>
            <password>mininet</password>
            <type>OnosDriver</type>
            <connect_order>1</connect_order>
            <COMPONENTS> </COMPONENTS>
        </ONOSbench>

        <ONOScli1>
            <host>127.0.0.1</host>
            <user>mininet</user>
            <password>mininet</password>
            <type>OnosCliDriver</type>
            <connect_order>2</connect_order>
            <COMPONENTS> </COMPONENTS>
        </ONOScli1>

        <ONOScli2>
            <host>127.0.0.1</host>
            <user>mininet</user>
            <password>mininet</password>
            <type>OnosCliDriver</type>
            <connect_order>3</connect_order>
            <COMPONENTS> </COMPONENTS>
        </ONOScli2>

        <ONOScli3>
            <host>127.0.0.1</host>
            <user>mininet</user>
            <password>mininet</password>
            <type>OnosCliDriver</type>
```

```
    <connect_order>4</connect_order>
    <COMPONENTS> </COMPONENTS>
</ONOScli3>


<ONOScli4>
    <host>127.0.0.1</host>
    <user>mininet</user>
    <password>mininet</password>
    <type>OnosCliDriver</type>
    <connect_order>5</connect_order>
    <COMPONENTS> </COMPONENTS>
</ONOScli4>


<ONOScli5>
    <host>127.0.0.1</host>
    <user>mininet</user>
    <password>mininet</password>
    <type>OnosCliDriver</type>
    <connect_order>6</connect_order>
    <COMPONENTS> </COMPONENTS>
</ONOScli5>


<ONOScli6>
    <host>127.0.0.1</host>
    <user>mininet</user>
    <password>mininet</password>
    <type>OnosCliDriver</type>
    <connect_order>7</connect_order>
    <COMPONENTS> </COMPONENTS>
</ONOScli6>


<ONOScli7>
    <host>127.0.0.1</host>
    <user>mininet</user>
    <password>mininet</password>
```

```
    <type>OnosCliDriver</type>
    <connect_order>8</connect_order>
    <COMPONENTS> </COMPONENTS>
</ONOScli7>

<ONOS1>
    <host>10.0.3.223</host>
    <user>sdn</user>
    <password>sdn</password>
    <type>OnosDriver</type>
    <connect_order>9</connect_order>
    <COMPONENTS> </COMPONENTS>
</ONOS1>

<ONOS2>
    <host>10.0.3.169</host>
    <user>sdn</user>
    <password>sdn</password>
    <type>OnosDriver</type>
    <connect_order>10</connect_order>
    <COMPONENTS> </COMPONENTS>
</ONOS2>

<ONOS3>
    <host>10.0.3.56</host>
    <user>sdn</user>
    <password>sdn</password>
    <type>OnosDriver</type>
    <connect_order>11</connect_order>
    <COMPONENTS> </COMPONENTS>
</ONOS3>

<ONOS4>
    <host>10.0.3.181</host>
    <user>sdn</user>
    <password>sdn</password>
    <type>OnosDriver</type>
    <connect_order>12</connect_order>
    <COMPONENTS> </COMPONENTS>
```

```
</ONOS4>

<ONOS5>
    <host>10.0.3.235</host>
    <user>sdn</user>
    <password>sdn</password>
    <type>OnosDriver</type>
    <connect_order>13</connect_order>
    <COMPONENTS> </COMPONENTS>
</ONOS5>

<ONOS6>
    <host>10.0.3.179</host>
    <user>sdn</user>
    <password>sdn</password>
    <type>OnosDriver</type>
    <connect_order>14</connect_order>
    <COMPONENTS> </COMPONENTS>
</ONOS6>

<ONOS7>
    <host>10.0.3.11</host>
    <user>sdn</user>
    <password>sdn</password>
    <type>OnosDriver</type>
    <connect_order>15</connect_order>
    <COMPONENTS> </COMPONENTS>
</ONOS7>

<Mininet1>
    <host>127.0.0.1</host>
    <user>mininet</user>
    <password>mininet</password>
    <type>MininetCliDriver</type>
    <connect_order>16</connect_order>
    <COMPONENTS>
        #Specify the Option for mininet
        <arg1> --custom ~/mininet/custom/topo-HA.py </arg1>
        <arg2> --topo mytopo </arg2>
```

```
                <arg3> </arg3>
                <controller> none </controller>
            </COMPONENTS>
        </Mininet1>

    </COMPONENT>
</TOPOLOGY>
```

Arquivo da topologia HATestMinorityRestart.params

```
<PARAMS>
    <testcases>1,2,8,3,4,5,14,16,17,[6],8,7,4,15,17,9,8,4,10,8,4,11,8,4,1
    <ENV>
        <cellName>HA</cellName>
    </ENV>
    <Git>False</Git>
    <branch> master </branch>
    <num_controllers> 7 </num_controllers>
    <tcpdump> False </tcpdump>

    <CTRL>
        <ip1>10.0.3.223</ip1>
        <port1>6633</port1>

        <ip2>10.0.3.169</ip2>
        <port2>6633</port2>

        <ip3>10.0.3.56</ip3>
        <port3>6633</port3>

        <ip4>10.0.3.181</ip4>
        <port4>6633</port4>

        <ip5>10.0.3.235</ip5>
        <port5>6633</port5>

        <ip6>10.0.3.179</ip6>
        <port6>6633</port6>

        <ip7>10.0.3.11</ip7>
        <port7>6633</port7>
```

```
</CTRL>
<TESTONUSER>admin</TESTONUSER>
<TESTONIP>127.0.0.1</TESTONIP>
<PING>
    <source1>h8</source1>
    <source2>h9</source2>
    <source3>h10</source3>
    <source4>h11</source4>
    <source5>h12</source5>
    <source6>h13</source6>
    <source7>h14</source7>
    <source8>h15</source8>
    <source9>h16</source9>
    <source10>h17</source10>
    <target1>10.0.0.18</target1>
    <target2>10.0.0.19</target2>
    <target3>10.0.0.20</target3>
    <target4>10.0.0.21</target4>
    <target5>10.0.0.22</target5>
    <target6>10.0.0.23</target6>
    <target7>10.0.0.24</target7>
    <target8>10.0.0.25</target8>
    <target9>10.0.0.26</target9>
    <target10>10.0.0.27</target10>
</PING>
<timers>
    <LinkDiscovery>.2</LinkDiscovery>
    <SwitchDiscovery>.2</SwitchDiscovery>
</timers>
<kill>
    <switch> s5 </switch>
    <dpid> 0000000000005000 </dpid>
    <links> h5 s2 s1 s6 </links>
</kill>
<MNtcpdump>
    <intf>eth0</intf>
    <port> </port>
    <folder>~/packet_captures/</folder>
</MNtcpdump>
```

```
</PARAMS>
```

APÊNDICE C

Codificação para Colocar os Controladores do Opendaylight em Cluster

As configurações executadas em cada arquivo das instâncias do controlador para os experimentos no Capítulo 3. Estes arquivo devem ser configurado.

As configurações e comandos utilizadas para cada instância são apresentadas no presente Apêndice.

C.1 Habilitar Módulos de Cluster

Habilitar módulos de cluster para cada uma das 7 instância

```
# cd distribution-karaf-0.3.2-Lithium-SR2
# ./bin/karaf
$ karaf > feature:install odl-mdsal-clustering odl-openflowplugin-flo
$ karaf > feature:install http
$ karaf > bundle:install -s mvn:org.jolokia/jolokia-osgi/1.1.5
$ karaf > logout
```

C.2 Arquivos de Configuração para Iniciar o Cluster do OpenDayligth

Deve-se fazer uma confuguração para os seguinte arquivos "akka.conf"e "module-shards.conf".

Para o arquivo "configuration/initial/akka.conf", deve realizar as seguintos passos para cada uma das 7 instância:

A Onde ocorrer o trecho hostname = "127.0.0.1", deve alterar para o endereço IP da instancia;

B Onde ocorrer o trecho roles = ["member-1"], deve-se o role associado ao IP para cada instancias. Como por exemplo a primeira instância roles = ["member-1"], a segunda instância roles = ["member-2"], a terceira instância roles = ["member-2"] e assim por diante ate a sétima instância;

C Alterar os dois trecho que ocorre "seed-nodes", observe que são diferente os trechos. Sendo o mesmo para todas as instâncias:

seed-nodes = ["akka.tcp://opendaylight-cluster-data@192.168.1.7:2550", "akka.tcp://opendaylight-cluster-data@192.168.1.8:2550", "akka.tcp://opendaylight-cluster-data@192.168.1.9:2550", "akka.tcp://opendaylight-cluster-data@192.168.1.10:2550", "akka.tcp://opendaylight-cluster-data@192.168.1.11:2550", "akka.tcp://opendaylight-cluster-data@192.168.1.12:2550", "akka.tcp://opendaylight-cluster-data@192.168.1.13:2550"]

seed-nodes = ["akka.tcp://odl-cluster-rpc@192.168.1.7:2551", "akka.tcp://odl-cluster-rpc@192.168.1.8:2551", "akka.tcp://odl-cluster-rpc@192.168.1.9:2551", "akka.tcp://odl-cluster-rpc@192.168.1.10:2551", "akka.tcp://odl-cluster-rpc@192.168.1.11:2551", "akka.tcp://odl-cluster-rpc@192.168.1.12:2551", "akka.tcp://odl-cluster-rpc@192.168.1.13:2551"]

Abaixo segue o modelo do arquivo para instância 1, seguindo as instruções acima para construir as outras 6 instâncias:

```
odl-cluster-data {
  bounded-mailbox {
    mailbox-type = "org.opendaylight.controller.cluster.common.actor.Mete
    mailbox-capacity = 1000
    mailbox-push-timeout-time = 100ms
  }

  metric-capture-enabled = true

  akka {
    loglevel = "INFO"
    loggers = ["akka.event.slf4j.Slf4jLogger"]
    logger-startup-timeout = 300s

    actor {
      provider = "akka.cluster.ClusterActorRefProvider"
      serializers {
        java = "akka.serialization.JavaSerializer"
        proto = "akka.remote.serialization.ProtobufSerializer"
```

```
      readylocal = "org.opendaylight.controller.cluster.datastore.m
    }

    serialization-bindings {
      "com.google.protobuf.Message" = proto
      "org.opendaylight.controller.cluster.datastore.messages.Ready
    }

    default-dispatcher {
      # Setting throughput to 1 makes the dispatcher fair. It proce
      # the mailbox before moving on to the next mailbox
      throughput = 1
    }

    default-mailbox {
      # When not using a BalancingDispatcher it is recommended that
      # as it is the most efficient for multiple producer/single co
      mailbox-type="akka.dispatch.SingleConsumerOnlyUnboundedMailbo
    }
  }
  remote {
    log-remote-lifecycle-events = off
    netty.tcp {
      hostname = "192.168.1.7"
      port = 2550
      maximum-frame-size = 419430400
      send-buffer-size = 52428800
      receive-buffer-size = 52428800
    }
  }

  cluster {
    seed-nodes = ["akka.tcp://opendaylight-cluster-data@192.168.1.7
          "akka.tcp://opendaylight-cluster-data@192.168.1.8:2550",
          "akka.tcp://opendaylight-cluster-data@192.168.1.9:2550",
          "akka.tcp://opendaylight-cluster-data@192.168.1.10:2550",
          "akka.tcp://opendaylight-cluster-data@192.168.1.11:2550",
          "akka.tcp://opendaylight-cluster-data@192.168.1.12:2550",
          "akka.tcp://opendaylight-cluster-data@192.168.1.13:2550"]
```

```
      seed-node-timeout = 12s

      auto-down-unreachable-after = 300s

      roles = ["member-1"]

    }

    persistence {
      # By default the snapshots/journal directories live in KARAF_HOME.
      # modifying the following two properties. The directory location sp
      # The relative path is always relative to KARAF_HOME.

      # snapshot-store.local.dir = "target/snapshots"
      # journal.leveldb.dir = "target/journal"

    }
  }
}

odl-cluster-rpc {
  bounded-mailbox {
    mailbox-type = "org.opendaylight.controller.cluster.common.actor.Mete
    mailbox-capacity = 1000
    mailbox-push-timeout-time = 100ms
  }

  metric-capture-enabled = true

  akka {
    loglevel = "INFO"
    loggers = ["akka.event.slf4j.Slf4jLogger"]
    logger-startup-timeout = 300s

    actor {
      provider = "akka.cluster.ClusterActorRefProvider"

    }
```

```
    remote {
      log-remote-lifecycle-events = off
      netty.tcp {
        hostname = "192.168.1.7"
        port = 2551
        maximum-frame-size = 419430400
        send-buffer-size = 52428800
        receive-buffer-size = 52428800
      }
    }

    cluster {
      seed-nodes = ["akka.tcp://odl-cluster-rpc@192.168.1.7:2551",
            "akka.tcp://odl-cluster-rpc@192.168.1.8:2551",
            "akka.tcp://odl-cluster-rpc@192.168.1.9:2551",
            "akka.tcp://odl-cluster-rpc@192.168.1.10:2551",
            "akka.tcp://odl-cluster-rpc@192.168.1.11:2551",
            "akka.tcp://odl-cluster-rpc@192.168.1.12:2551",
            "akka.tcp://odl-cluster-rpc@192.168.1.13:2551"]

      auto-down-unreachable-after = 300s
    }
  }
}
```

Para o arquivo "configuration/initial/module-shards.conf"define-se os membros da replicação, sendo o mesmo para todas as instâncias:

```
# This file describes which shards live on which members
# The format for a module-shards is as follows,
# {
#    name = "<friendly_name_of_the_module>"
#    shards = [
#        {
#            name="<any_name_that_is_unique_for_the_module>"
#            replicas = [
#                "<name_of_member_on_which_to_run>"
#            ]
#    ]
# }
#
```

```
# For Helium we support only one shard per module. Beyond Helium
# we will support more than 1
# The replicas section is a collection of member names. This information
# will be used to decide on which members replicas of a particular shard
# located. Once replication is integrated with the distributed data store
# this section can have multiple entries.
#
#


module-shards = [
    {
        name = "default"
        shards = [
            {
                name="default"
                replicas = ["member-1",
            "member-2",
            "member-3",
            "member-4",
            "member-5",
            "member-6",
            "member-7"]
            }
        ]
    },
    {
        name = "topology"
        shards = [
            {
                name="topology"
                replicas = ["member-1",
            "member-2",
            "member-3",
            "member-4",
            "member-5",
            "member-6",
            "member-7"]
            }
```

```
        ]
    },
    {
        name = "inventory"
        shards = [
            {
                name="inventory"
                replicas = ["member-1",
            "member-2",
            "member-3",
            "member-4",
            "member-5",
            "member-6",
            "member-7"]
            }
        ]
    },
         {
             name = "toaster"
             shards = [
                 {
                     name="toaster"
                     replicas = ["member-1",
            "member-2",
            "member-3",
            "member-4",
            "member-5",
            "member-6",
            "member-7"]
                 }
             ]
         }

]
```