UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DAVI DANIEL GEMMER

MODELO DE TRÁFEGO EM UM CLUSTER SDN ONOS/ATOMIX

DISSERTAÇÃO

PONTA GROSSA
2021

DAVI DANIEL GEMMER

MODELO DE TRÁFEGO EM UM CLUSTER SDN ONOS/ATOMIX

Traffic Model in an ONOS/Atomix SDN Cluster

Dissertação apresentada como requisito para obtenção do título de Mestre em Ciência da Computação, do Programa de Pós-Graduação em Ciência da Computação, da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Prof. Dr. Augusto Foronda

PONTA GROSSA

2021

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Ponta Grossa

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DAVI DANIEL GEMMER

MODELO DE TRÁFEGO EM UM CLUSTER SDN ONOS/ATOMIX

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre Em Ciência Da Computação da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Sistemas E Métodos De Computação.

Data de aprovação: 25 de Outubro de 2021

Prof Augusto Foronda, Doutorado - Universidade Tecnológica Federal do Paraná

Prof Lourival Aparecido De Gois, Doutorado - Universidade Tecnológica Federal do Paraná

Prof Mauricio Zadra Pacheco, Doutorado - Universidade Estadual de Ponta Grossa (Uepg)

Dedico este trabalho a minha família e aos meus amigos, pelos momentos de ausência.

AGRADECIMENTOS

Este trabalho não poderia ser terminado sem a ajuda de diversas pessoas e instituições às quais presto minha homenagem. Certamente esses parágrafos não irão atender a todas as pessoas que fizeram parte dessa importante fase de minha vida. Portanto, desde já peço desculpas àquelas que não estão presentes entre estas palavras, mas elas podem estar certas que fazem parte do meu pensamento e de minha gratidão.

A minha família, pelo carinho, incentivo e total apoio em todos os momentos da minha vida.

Ao meu orientador, que me mostrou os caminhos a serem seguidos e pela confiança depositada.

A todos os professores e colegas do departamento, que ajudaram de forma direta e indireta na conclusão deste trabalho.

Enfim, a todos os que de alguma forma contribuíram para a realização deste trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

RESUMO

A rede definida por *software* (SDN) é um paradigma de rede emergente que supera as limitações de uma infraestrutura de rede tradicional. A SDN pode melhorar o monitoramento, gerenciamento, segurança e engenharia do tráfego de rede através de um controlador central. A arquitetura SDN com um único controlador apresenta algumas desvantagens, tais como baixa confiabilidade e escalabilidade. Para resolver estes problemas, surgiu a arquitetura SDN distribuída, onde a rede é controlada por um *cluster* com vários controladores. Mas para garantir a consistência das informações da rede, é necessário um protocolo de consenso para atualizar as informações e a sincronização dos controladores. Existem dois modelos de consistência para redes SDN: consistência eventual e consistência forte. Na consistência forte as leituras dos dados em diferentes controladores produzem sempre o mesmo resultado, ao passo que na consistência eventual há períodos transientes em que as leituras em diferentes controladores podem produzir diferentes valores. Um exemplo de protocolo de consistência eventual é o *Anti-Entropy* e de consistência forte é o Raft. E o protocolo de consistência escolhido por cada controlador influencia na quantidade de tráfego entre os controladores, que é usado no projeto de dimensionamento dos enlaces entre os controladores. Neste trabalho é desenvolvido um modelo analítico para determinar o tráfego entre controladores em função do número de *switches* e de enlaces para um *cluster* de 3 controladores. É feita a medição e análise do tráfego entre controladores ONOS em duas arquiteturas: uma só com controladores ONOS, que será chamada de arquitetura ONOS e outra com controladores ONOS mais o *framework* Atomix, que será chamado de arquitetura ONOS/Atomix. A análise é feita em duas situações: com um número diferente de *switches* em cada controlador e com um número diferente de controladores. Os resultados mostram que a arquitetura ONOS e ONOS/Atomix tem aproximadamente o mesmo tráfego entre controladores em diversos cenários. O tráfego em cada enlace cresce diretamente proporcional com o aumento do número de *switches* e enlaces da rede e diminui com o aumento do número de controladores.

Palavras-chave: SDN. Modelo de Tráfego. *Cluster*. ONOS. Atomix.

ABSTRACT

The software-defined network (SDN) is an emerging network paradigm that overcomes the limitations of a traditional network infrastructure. SDN can improve the monitoring, management, security and engineering of network traffic through a central controller. The SDN architecture with a single controller has disadvantages, such as low reliability and scalability. To solve these problems, the distributed SDN architecture emerged, where the network is controlled by a cluster with several controllers. To ensure the consistency of the network information, a consensus protocol is required to update the information and the synchronization of the controllers. There are two consistency models for SDN networks: eventual consistency and strong consistency. Strong consistency means that data readings on different controllers always produce the same result, while eventual consistency means that there are transient periods when readings on different controllers can produce different values. An example of an eventual consistency protocol is Anti-Entropy and a strong consistency is Raft. The consistency protocol chosen by each controller influences the amount of traffic between the controllers, which is used in the design of dimensioning the links between the controllers. In this work, an analytical model is developed in this work to determine the traffic between controllers as a function of the number of switches and links for a cluster of 3 controllers. Traffic analysis between ONOS controllers is made in two architectures: one with ONOS controllers, which will be called ONOS architecture and the other with ONOS controllers with the Atomix framework, which will be called ONOS/Atomix architecture. The analysis is done in two situations: with a different number of switches on each controller and with a different number of controllers. And the influence of traffic on the control plane on data plan traffic is also analyzed. The results show that the ONOS and ONOS/Atomix architecture have approximately the same traffic between controllers in different scenarios. Traffic grows directly proportional to the increase in the number of switches on the network and to the increase in the number of controllers.

Keywords: SDN. Traffic Model. Cluster. ONOS. Atomix.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SIGLAS

API	Application Programming Interface
CPU	Central Processing Unit
ICN	Information-Centric Networking
IP	Internet Protocol
IoT	Internet of Things
KBPS	Kilobits per second
LLDP	Link Layer Discovery Protocol
NOS	Network Operating System
ONOS	Open Network Operating System
OVS	Open vSwitch
RAM	Random Acess Memory
REST	Representational State Transfer
RPC	Remote Procedure Call
SAS	Serial Attached SCSI
SAL	Service Abstration Layer
SDN	Software Network Defined
SWIM	Scalable Weakly-consistent Infection-style Process Group Membership
UTFPR	Universidade Tecnológica Federal do Paraná

LISTA DE SÍMBOLOS

$\neq$	Diferente de
GB	Gigabyte
GHz	Gigahertz
λ	Taxa de Sincronização
$\in$	Pertence

SUMÁRIO

1 INTRODUÇÃO

A indústria da tecnologia de informação sofre atualizações constantes tornando sua evolução necessária para acompanhar a demanda crescente por disponibilidade e velocidade das redes de comunicação. Até alguns anos atrás, os recursos de computação, rede e armazenamento eram mantidos separados física e operacionalmente, porém a demanda por esses recursos forçou as organizações a unir esses diferentes elementos tornando necessária uma mudança nos paradigmas que gerenciavam e operavam esses recursos (COKER; AZODOLMOLKY, 2017).

Um dos paradigma foi a virtualização do sistema operacional, onde servidores que normalmente executavam um único sistema operacional dedicado passaram a executar uma variedade de sistemas operacionais. Mas ao mesmo tempo em que os *data centers* estavam evoluindo, os equipamentos de rede basicamente melhoraram apenas em termos de velocidade. O paradigma atual dos dispositivos de rede tem o plano de dados e de controle no mesmo elemento. Isso geralmente não é um problema até que exista a necessidade de programar as funcionalidades de um dispositivo, onde um equipamento não poderia ser utilizado para um propósito diferente daquele para o qual tenha sido construído. Então o conceito de plano de controle separado do plano de dados voltou a ser discutido como um novo paradigma (QI; LI, 2016).

O desafio de poder programar qualquer solução em um equipamento de rede trouxe grande motivação evoluindo para o que hoje é chamado de rede definida por *software* (SDN). Os fornecedores de dispositivos de rede não estavam mais atendendo as necessidades principalmente no desenvolvimento de recursos e inovação e com o barateamento e aumento do poder de computação tornou-se realidade executar um plano de controle logicamente centralizado até mesmo com *hardware* de baixo custo. SDN é uma arquitetura de rede que otimiza e simplifica as operações, vinculando de forma mais estreita a interação entre aplicativos, serviços e dispositivos de rede reais ou virtualizados. Isso geralmente é obtido empregando um ponto de controle de rede logicamente centralizado realizado por um controlador SDN que gerencia e facilita a comunicação entre aplicativos que desejam interagir com elementos de rede (NADEAU; GRAY, 2013).

Em redes de grande escala um único controlador pode encontrar grandes dificuldades e desafios, principalmente a falta de confiabilidade e escalabilidade. O controlador SDN desempenha um papel importante no processo de controle de transmissão de tráfego, porém a medida que o tráfego de rede aumenta, um único controlador não consegue lidar com um grande

número de solicitações de fluxos enviadas pelos *switches* devido a sua capacidade limitada. E se o controlador falhar, os *switches* não conseguem planejar o encaminhamento dos pacotes recém chegados afetando a comunicação e os aplicativos de rede. Isto levou ao surgimento de uma arquitetura de rede SDN com multi-controladores (EL-GEDER, 2017).

Um multi-controlador SDN consiste em uma arquitetura com o objetivo de resolver os problemas causados pela arquitetura com um único controlador. Através da utilização de mais controladores na topologia de rede, cada um dos controladores gerencia parte da rede compartilhando a mesma lógica de maneira centralizada, de modo que quando novos pacotes chegam no *switch*, todos os controladores podem instalar diretamente o encaminhamento em todos os *switches* correspondentes. Isto alivia a pressão de processamento em um único controlador e também traz a possibilidade de *backup* entre os controladores o que poderia resolver o ponto único de falha trazendo benefícios em comparação com um único controlador melhorando o desempenho da rede SDN (HU *et al.*, 2018). A literatura sobre SDN usa vários termos para descrever uma rede SDN com vários controladores: SDN distribuída, SDN multi-controlador e *cluster* SDN. A partir daqui, neste trabalho será usado o termo *cluster* SDN.

Um *cluster* SDN resolve alguns problemas encontrados na arquitetura com um único controlador, mas apresenta alguns desafios. A consistência dos controladores é um fator importante para garantir a confiabilidade das informações e manter a topologia de rede atualizada. Protocolos como *Anti-Entropy* e Raft são exemplos de soluções para manter a consistência. O protocolo *Anti-Entropy* é um dos protocolos usado para garantir a sincronização das réplicas em uma rede SDN. Seu algoritmo é simples e cada controlador escolhe outro controlador aleatoriamente para as atualizações periodicamente. É classificado como um protocolo de consistência eventual, onde não há uma garantia de que os controladores terão as mesmas informações sobre a rede. Porém pode ser utilizado por aplicações em que o estado dos controladores pode variar ligeiramente (BANNOUR *et al.*, 2018a). Raft é um protocolo que possui uma maior confiabilidade do que o *Anti-Entropy*, sendo considerado de consistência forte. Raft consiste em um algoritmo que gerencia um *log* replicado implementando a consistência elegendo primeiro um líder e dando a ele responsabilidade total pelo gerenciamento do *log* replicado. O líder aceita entradas de *log* de um controlador e as replica em outros controladores informando quando é seguro aplicar entradas de *log* a suas máquinas de estado. Ter um líder simplifica o gerenciamento do *log* replicado e em caso de falhas, um novo líder é eleito (ONGARO; OUSTERHOUT, 2014).

Muitos controladores estão surgindo, dentre eles o ONOS vem ganhando notoriedade. O

controlador ONOS apresenta duas arquiteturas. Uma arquitetura apresenta vários controladores formando um *cluster* SDN. Outra arquitetura apresenta, além do *cluster* SDN, um outro *cluster* formado pelo *framework* Atomix. Para diferenciar as duas arquiteturas, a primeira será chamada de ONOS e a segunda arquitetura será chamada de ONOS/Atomix. A arquitetura ONOS/Atomix surgiu para resolver o problema da arquitetura ONOS quando alguns controladores falhavam e a rede ficava indisponível. Na arquitetura ONOS/Atomix, o *cluster* Atomix é responsável pelo gerenciamento dos controladores ONOS permitindo que estes possam falhar e a rede continuará disponível. Cada arquitetura usa determinado protocolo para manter a consistência, o que influencia no tráfego entre os controladores.

Neste trabalho, será utilizado como base o modelo de tráfego entre controladores para um *cluster* ONOS desenvolvido por Muqaddas *et al.* (2017), e é proposta uma extensão a este modelo para a arquitetura ONOS/Atomix com o objetivo de auxiliar no projeto de dimensionamento de enlaces e distribuição de *switches* e controladores na rede. É analisado o tráfego com um número diferente de *switches* em cada controlador e com um número diferente de controladores (MUQADDAS *et al.*, 2017).

1.1 OBJETIVO GERAL

O objetivo geral deste trabalho é desenvolver um modelo de tráfego em um *cluster* SDN ONOS/Atomix.

1.1.1 Objetivos Específicos

1. Definir o melhor ambiente SDN para possibilitar a análise do tráfego em um *cluster* ONOS e em um *cluster* ONOS/Atomix;
2. Medir o tráfego entre os controladores em um *cluster* ONOS e em um *cluster* ONOS/Atomix em função do número de enlaces e *switches*;
3. Usar os valores medidos para desenvolver equações que representem a vazão nos enlaces para qualquer distribuição de *switches* em um *cluster* ONOS/Atomix com 3 controladores;
4. Comparar os valores de vazão do modelo analítico com valores medidos para diferentes distribuições de *switches* em um *cluster* ONOS/Atomix com 3 controladores;

5. Propor a melhor distribuição de *switches* e controladores para minimizar o tráfego entre controladores.

1.2 ESTRUTURA DA DISSERTAÇÃO

O presente trabalho está organizado em cinco capítulos. O capítulo 1 é referente a contextualização do tema, assim como os objetivos e organização do trabalho. O capítulo 2 apresenta a fundamentação teórica do trabalho de modo a explicar conceitos de SDN juntamente com os trabalhos relacionados. No capítulo 3 é apresentado o modelo de tráfego entre controladores. O capítulo 4 descreve os resultados das simulações para avaliar o modelo. Por fim o capítulo 5 apresenta as conclusões e trabalhos futuros.

2 REFERENCIAL TEÓRICO

Este capítulo aborda os conceitos fundamentais para a compreensão do tema do presente trabalho. A seção 2.1 apresenta os conceitos de SDN, sua arquitetura, camadas e também é abordada a padronização da rede SDN através do modelo OpenFlow juntamente com o uso de *cluster* SDN e a sua consistência. A seção 2.2 detalha a estrutura do controlador SDN ONOS com seus principais componentes. A seção 2.3 contempla as características e arquitetura do *framework* Atomix. Por fim a seção 2.4 aborda os trabalhos relacionados ao tema.

2.1 REDES DEFINIDAS POR *SOFTWARE* (SDN)

As redes definidas por *software* (SDN) são um novo paradigma de rede proposto como uma maneira de facilitar a evolução das redes de comunicação. O conceito de SDN foi originalmente proposto pela Nicira Network na universidade de Stanford (ORTIZ, 2013). Sua principal característica consiste na dissociação entre o *hardware* de encaminhamento e o *hardware* de decisões de controle com o intuito de simplificar o gerenciamento da rede e a sua evolução. SDN fornece uma visão global de toda a rede sem a necessidade da configuração em cada unidade de *hardware* de encaminhamento de pacotes (HU *et al.*, 2014).

A inteligência da rede SDN é logicamente centralizada nos controladores baseados em *software* (Plano de Controle) e os equipamentos de rede tornam-se dispositivos simples de encaminhamento de pacotes (Plano de Dados) (NUNES *et al.*, 2014). Isso permite que os administradores de rede programem todo o plano de controle por meio de uma *Application Programming Interface* (API) comum facilitando o processo e oferecendo controle e flexibilidade sobre os fluxos de tráfego (ORTIZ, 2013).

2.1.1 Arquitetura SDN

A arquitetura SDN pode ser definida em quatro pilares (KREUTZ *et al.*, 2014) :

1. O plano de dados é dissociado do plano de controle. A funcionalidade de controle é removida dos dispositivos de rede que se tornarão simples elementos de encaminhamento de pacotes;

2. As decisões de encaminhamento são baseadas no fluxo ao invés do destino. Um fluxo

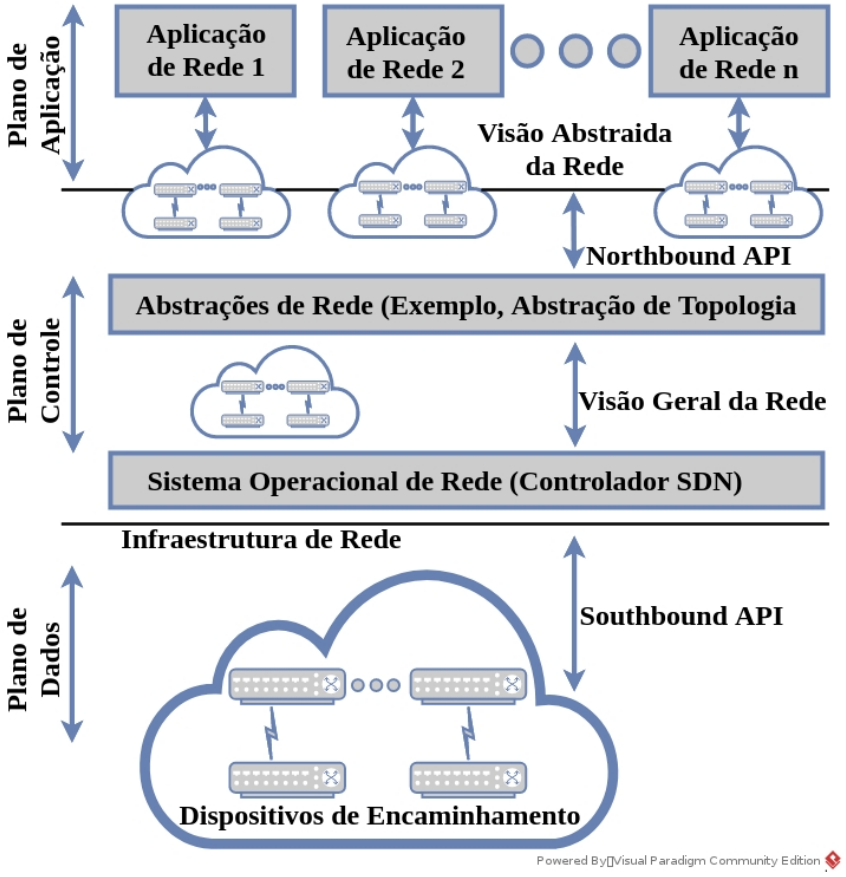
consiste em uma sequência de pacotes entre uma origem e um destino. Todos os pacotes de um fluxo recebem políticas de serviço idênticas nos dispositivos do plano de dados;

3. A lógica de controle é movida para o controlador SDN, que é uma plataforma de *software* que fornece recursos e abstrações para facilitar a programação de dispositivos do plano de dados;

4. A rede é programável por meio de aplicativos de *software* em execução no controlador SDN que interage com os dispositivos do plano de dados.

A Figura 1 mostra a arquitetura SDN, que é dividida em três camadas: plano de controle, plano de dados e plano de aplicação. O plano de controle é representado por um controlador SDN. O plano de dados é onde ficam os equipamentos da rede, como *switches* e roteadores. E o plano de aplicação tem os aplicativos para o controle da rede. A próxima seção explica cada camada da arquitetura SDN em mais detalhes.

Figura 1 – Arquitetura SDN



Fonte: Adaptado de (KREUTZ *et al.*, 2014).

2.1.2 Camadas do SDN

As camadas SDN são compostas por

- Plano de Aplicação: compreende aplicativos que são programas projetados para implementar a lógica e as estratégias de controle da rede. A sua interação com o plano de controle é através da API *northbound* para que aplicativos SDN comuniquem seus requisitos de rede ao controlador SDN, que converte em comandos específicos e regras de encaminhamento na API *southbound* determinando o comportamento dos dispositivos do plano de dados. Os aplicativos expressam diretamente os objetivos e políticas desejados de maneira centralizada e de alto nível sem estar vinculados a implementação e detalhes da distribuição do estado na infraestrutura de rede (BANNOUR *et al.*, 2018b);

- Plano de Controle: compreende um conjunto de controladores SDN baseados em *software*. Fornece funcionalidades de controle supervisionando o comportamento de encaminhamento de rede no plano de dados. Possui interfaces para permitir a comunicação entre controladores, possibilita a comunicação entre aplicativos e serviços, controle para a segurança da rede e gerenciamento. Existem dois principais componentes. O primeiro consiste na lógica de controle SDN sobre o controlador para mapear os requisitos de aplicativos em instruções para recursos de elementos de rede. O segundo componente representa o coordenador e virtualizador que gerenciam o comportamento do controlador (KARAKUS; DURRESI, 2017). O plano de controle em uma rede SDN geralmente é chamado de *Network Operating System* (NOS) devido a suportar a lógica de controle da rede e fornecer a camada de aplicativo uma visão abstrata da topologia da rede que contém informações suficientes para especificar políticas. Por razões de escalabilidade e confiabilidade, normalmente o plano de controle é logicamente centralizado e implementado como um sistema fisicamente distribuído (BANNOUR *et al.*, 2018b);

- Plano de Dados: também conhecido como plano de encaminhamento. Consiste em um conjunto distribuído de elementos de rede como *switches* responsáveis por encaminhar pacotes. O plano de dados de uma rede SDN deve ser remotamente acessível pelo controlador por meio da interface *southbound* independente do fornecedor porque o plano de dados é separado do plano de controle (BANNOUR *et al.*, 2018b). Algumas ações típicas do plano de dados são: encaminhamento, descarte, remarcação e contagem de pacotes,

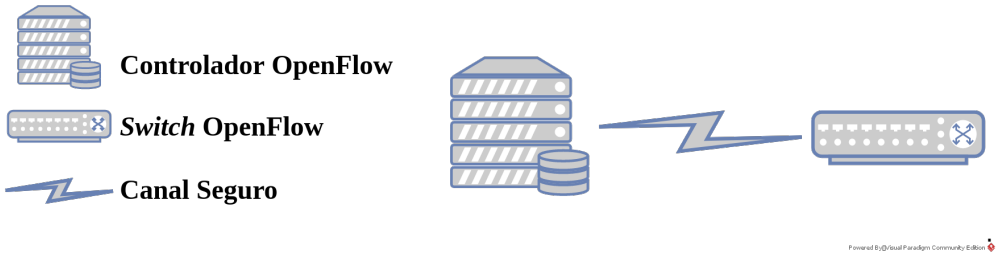
onde algumas dessas ações podem ser combinadas ou encadeadas. O plano de dados pode também implementar alguns serviços como lista de controle de acesso (NADEAU; GRAY, 2013).

A comunicação entre plano de controle e plano de dados é realizada por um protocolo, como por exemplo, o OpenFlow, que será descrito a seguir.

2.1.3 OPENFLOW

O OpenFlow padroniza a forma como o controlador se comunica com dispositivos de rede em uma arquitetura SDN e fornece uma especificação para migrar a lógica de controle de um *switch* para o controlador definindo um protocolo para essa comunicação (MARSCHKE *et al.*, 2015). Os componentes OpenFlow podem ser observados na Figura 2.

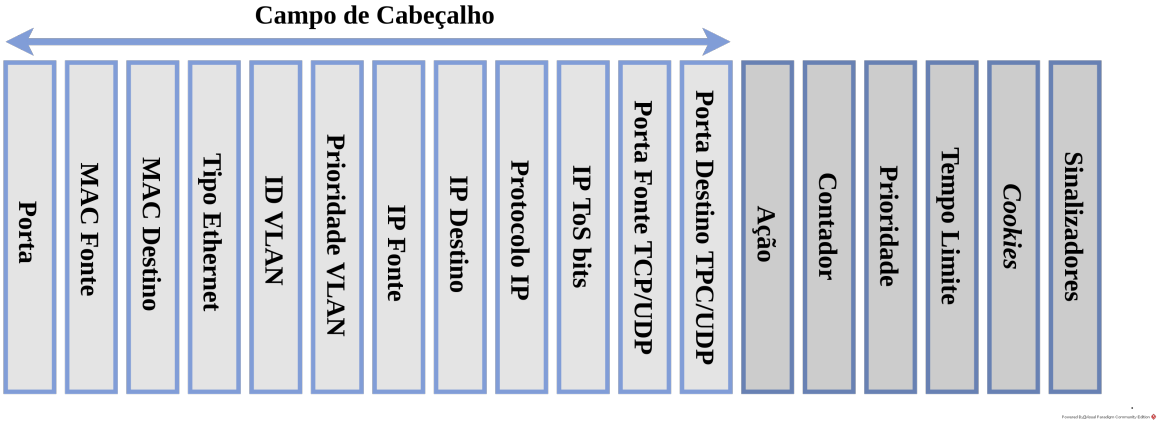
Figura 2 – Componentes do OpenFlow



Fonte: Adaptado de (LARA *et al.*, 2013).

Como pode ser observado na Figura 2, a arquitetura Openflow consiste em três principais componentes: um *switch* compatível com o OpenFlow, um controlador e um canal seguro (LARA *et al.*, 2013). Os *switches* utilizam uma tabela de fluxos que consiste em uma lista de entradas com campos de correspondência, contadores e instruções para encaminhar pacotes, essa tabela pode ser observada na Figura 3. O controlador é responsável por manipular essa tabela de fluxos utilizando o protocolo OpenFlow. O canal seguro representa a interface que conecta o controlador a todos os *switches* (HU, 2014).

Figura 3 – Tabela de Fluxo do Switch OpenFlow



Fonte: Adaptado de (COKER; AZODOLMOLKY, 2017).

Um *switch* OpenFlow utiliza uma tabela de fluxo similar a Figura 3, que consiste em (COKER; AZODOLMOLKY, 2017):

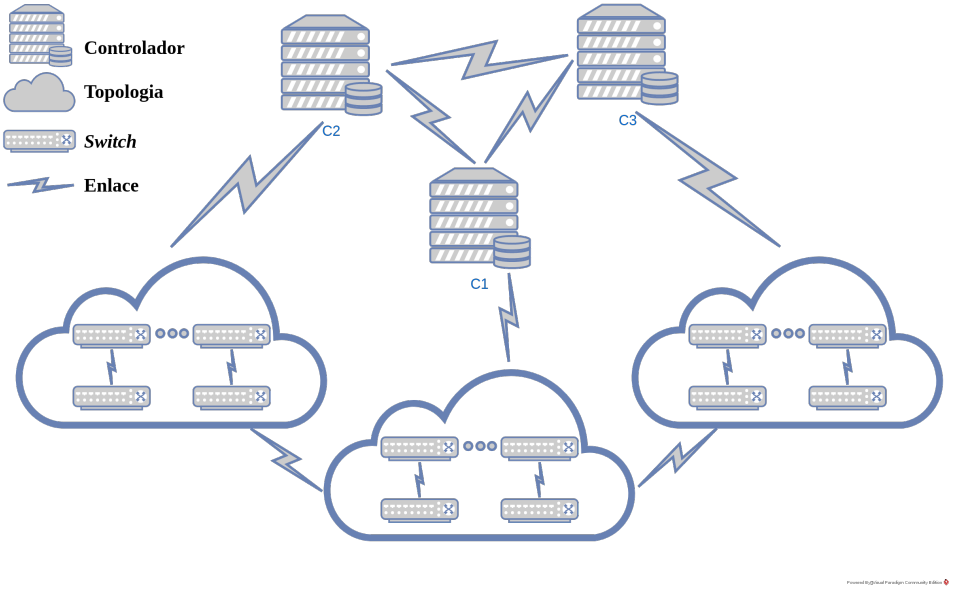
- Campo de cabeçalho: contém informações encontradas no cabeçalho dos pacotes;
- Ação: consiste em um conjunto de instruções a serem aplicadas, como a ação de encaminhar um pacote para uma porta especifica;
- Contador: utilizado para coletar estatísticas de fluxo como por exemplo o número de pacotes recebidos;
- Prioridade: contem informações de fluxo como tempo limite utilizado pelo *switch* como por exemplo o descarte de um fluxo após determinado tempo em que ele permanece ocioso;
- Tempo Limite: é utilizado pelo *switch* para determinar a quantidade máxima de tempo ocioso em que o fluxo deve ser descartado;
- *Cookies*: utilizado pelo controlador para filtrar entradas como solicitações e exclusões de fluxo;
- Sinalizadores: usados na maneira como os fluxos são gerenciados, como para informar o *switch* na verificação de fluxo conflitante com a mesma prioridade.

2.1.4 *Cluster* SDN

O controlador SDN desempenha um papel fundamental na transmissão de dados, porém com o aumento do tráfego no plano de dados, um único controlador pode falhar afetando a

comunicação da rede. Uma solução promissora para redes de grande escala é o uso de um *cluster* SDN. Um *cluster* SDN consiste em um conjunto de controladores trabalhando em conjunto, sendo que e a rede pode ser dividida em vários domínios e cada controlador gerenciando seu próprio domínio, como pode ser visto na Figura 4 (HU *et al.*, 2018).

Figura 4 – Exemplo de um *Cluster* SDN



Fonte: Adaptado de (HU *et al.*, 2018).

O uso de um *cluster* SDN resolve o problema de falha em um único controlador, mas é um desafio manter a consistência entre os controladores. Para garantir que os pacotes sejam transmitidos corretamente, os controladores devem interagir uns com os outros buscando informações do domínio para manter a consistência da rede. O problema de consistência afeta o plano de dados, onde todos os *switches* devem executar um mesmo conjunto de regras (YU *et al.*, 2018).

2.1.4.1 Consistência na rede SDN

Consistência em uma rede SDN distribuída significa que todos os controladores tem a mesma visão sobre o estado da rede, como por exemplo, o número de *switches* e como eles estão conectados entre eles. A consistência consiste em cópias do estado da rede, deste modo todas as alterações dos controladores devem ser propagadas aos demais controladores, quando isso não ocorre, é criada uma inconsistência que acarreta no mal funcionamento da rede. Existem dois principais modelos de consistência utilizados nas plataformas dos controladores SDN: modelo de consistência eventual e modelo de consistência forte (MUQADDAS *et al.*, 2017).

O modelo de consistência eventual fornece uma maneira fraca de consistência quando analisada a forma como as modificações dos dados são propagadas nos controladores. Por exemplo, quando um controlador recebe uma nova informação do plano de dados, essa informação deve ser propagada aos demais controladores. Mas em virtude do atraso na atualização, alguns controladores podem ter temporariamente uma visão não atualizada da rede causando o comportamento incorreto de algumas aplicações. Com o tempo, todos os controladores terão valores atualizados devido a troca de informações entre os mesmos, ou seja, no início das atualizações, pode haver inconsistência e após todas as atualizações, a rede se torna consistente. Alguns aplicativos adotam esse modelo de consistência quando o objetivo é garantir a alta disponibilidade e desempenho devido ao rápido acesso aos dados em detrimento de informações consistentes entre os controladores. O protocolo *Anti-Entropy* faz o uso desse modelo de consistência (BANNOUR *et al.*, 2018a).

O modelo de consistência forte garante que o controlador sempre leia a versão mais atualizada dos dados. Porem isso tem um custo maior de atraso na sincronização de estado e sobrecarga de comunicação. Nesse modelo, os dados que não foram atualizados pela maioria dos controladores, não podem ser lidos, o que evita que o plano de dados tenha acesso a informações desatualizadas. Isso favorece a consistência ao invés da disponibilidade. Quando um *cluster* é fortemente consistente, toda a atualização é iniciada por um cliente do *cluster* presente em uma das réplicas do controlador onde essa réplica é responsável por enviar as solicitações ao líder do *cluster* que por sua vez atualiza o estado da rede. Após essa atualização, os dados são replicados e somente armazenados após a maioria das réplicas concordarem com a atualização garantindo que todas as réplicas concordem com os mesmos valores tornando esse modelo fortemente consistente. Certos aplicativos possuem como requisito a consistência forte e o modelo é implementado com frequência em *clusters* que exigem uma sincronização antes de qualquer operação. O protocolo Raft faz uso desse modelo de consistência (SAKIC; KELLERER, 2017).

A influência da implementação dos modelos de consistência usados entre os controladores é analisada em Muqaddas *et al.* (2017), que destaca a influência desses modelos na quantidade de tráfego entre os controladores e classifica esse tráfego através de uma combinação dos seguintes tipos de atualizações:

- Atualizações Incrementais ou Atualizações Completas: as atualizações incrementais apenas alteram as diferenças em relação as atualizações anteriores. Esse modelo depende

de atualizações coerentes dos estados anteriores sendo normalmente empregadas em modelos de consistência forte. As atualizações completas geralmente alteram estruturas de dados eventualmente consistentes por causa da coerência do estado não confiável entre as estruturas de dados;

- Atualizações Periódicas ou Atualizações Baseadas em Eventos: atualizações periódicas são geradas de tempo em tempo e atualizações baseadas em eventos são acrescentadas por alterações específicas de eventos ou estados.

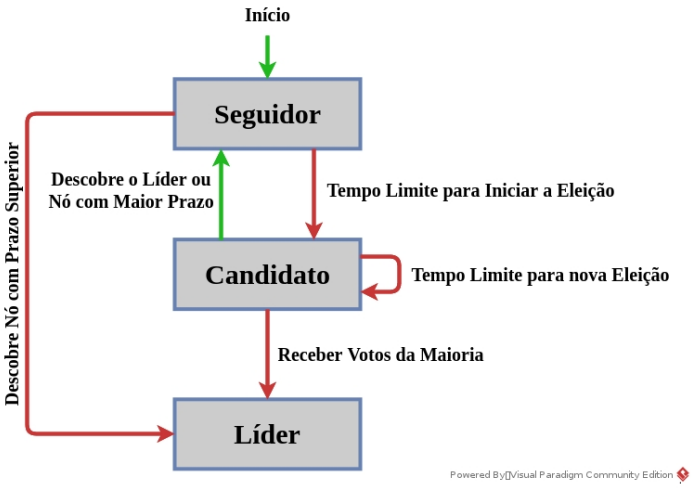
Os autores citam que todas as 4 combinações das 2 características acima são possíveis de ser implementadas e vão influenciar diretamente a quantidade de tráfego entre os controladores, o que será demonstrado na seção de resultados.

2.1.4.2 Protocolo Raft

O Raft consiste em um protocolo de consistência tolerante a falhas. O Raft separa o seu funcionamento em 3 partes: eleição do líder, replicação de *logs* e segurança (ONGARO; OUSTERHOUT, 2014).

- Eleição do líder: no protocolo Raft a comunicação dos controladores é feita através das chamadas de procedimento remoto (RPCs). O Raft utiliza uma mensagem conhecida como *heartbeat*, que segue o principio de batimentos cardíacos sendo responsável por manter a comunicação entre os controladores através de mensagens. Um controlador pode estar em um de três estados: seguidor, candidato e líder, que podem ser observados na Figura 5 (VIZARRETA *et al.*, 2020).

Figura 5 – Eleição do Raft

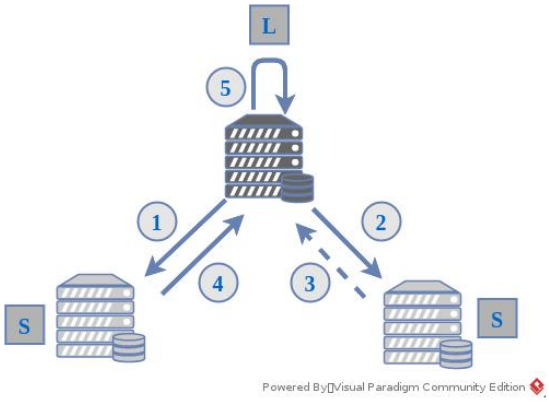


Fonte: Adaptado de (VIZARRETA *et al.*, 2020).

O processo da eleição pode ser visto na Figura 5. Ao ser iniciado o *cluster*, todos os controladores são seguidores. Não havendo um líder, qualquer um deles pode se tornar um candidato sendo iniciada uma eleição. Os candidatos solicitam votos de outros controladores onde é possível votar em apenas um candidato garantindo que apenas um líder possa ser eleito. Se dois controladores se tornarem candidatos ao mesmo tempo e ocorra empate, é iniciada uma nova eleição. Após as respostas, o candidato com o maior número de votos se torna o líder passando a ser responsável por todas as alterações no sistema. O líder passa a enviar mensagens aos demais controladores que respondem a cada mensagem e estas são enviadas em intervalos periódicos. Para manter seu mandato é necessário que a comunicação entre os controladores não seja interrompida. Caso houver uma falha de comunicação e um seguidor não receba uma mensagem *heartbeat* em um período de tempo predefinido, o controlador assume que não há um líder viável, passa a ser candidato e uma nova eleição é feita (ONGARO; OUSTERHOUT, 2014) (WOOS *et al.*, 2016).

- Replicação do *log*: Após eleger um líder é necessário replicar todas as alterações do sistema em todos os controladores, como pode ser visto na Figura 6. Primeiramente o líder L envia uma atualização para os seguidores S (1 e 2). Na próxima mensagem *heartbeat*, os seguidores L confirmam a atualização e respondem para o líder (3 e 4). Por fim essa atualização é anexada ao *log* do líder (5). Uma entrada é confirmada quando a maioria dos seguidores a reconhece. Ao concluir, o líder notifica que a entrada foi confirmada resultando no consenso sobre o estado do sistema (ONGARO; OUSTERHOUT, 2014) (WOOS *et al.*, 2016).

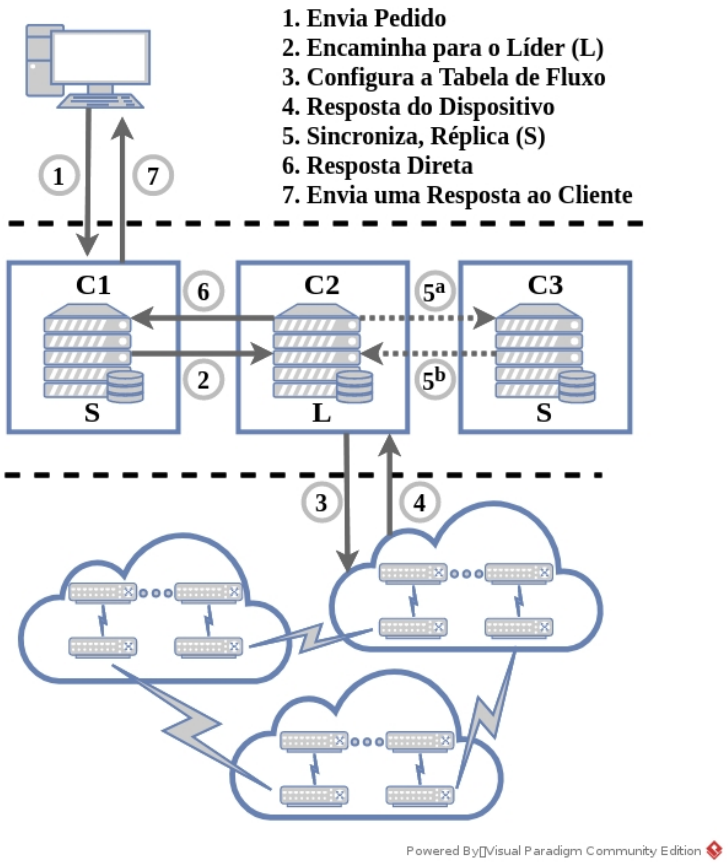
Figura 6 – Replicação do *log*



Fonte: Adaptado de (VIZARRETA *et al.*, 2020).

- Segurança: para garantir a segurança o protocolo Raft adiciona uma restrição que determina os controladores que podem ser eleitos líderes. Essa restrição garante que o líder contenha todas as entradas confirmadas, evitando que máquinas de estado possam executar diferentes sequências de comandos, que poderiam acarretar na substituição dos *logs* de entradas nas réplicas (ONGARO; OUSTERHOUT, 2014).

Figura 7 – Funcionamento do protocolo Raft



Fonte: Adaptado de (VIZARRETA *et al.*, 2020).

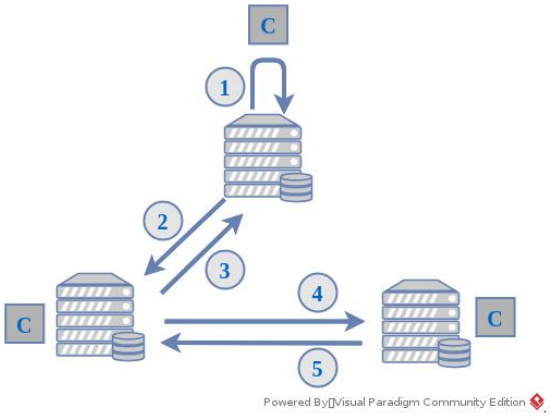
Em redes de grande escala o estado da rede é particionado em pedaços menores também denominados fragmentos (*shards*). Para o provisionamento de sistemas tolerantes a falhas é necessário que os fragmentos sejam replicados em vários nós (VIZARRETA *et al.*, 2020). A Figura 7 mostra o funcionamento de uma rede utilizando o protocolo Raft. A primeira etapa do funcionamento do protocolo Raft consiste em enviar o pedido (1) que por conseguinte envia para o líder (2), o líder por sua vez é responsável pela configuração da tabela de fluxo (3), após essa configuração ocorre a resposta do dispositivo (4), o líder sincroniza e replica as informações atualizadas para os demais controladores (5), então ocorre a resposta direta para o controlador com o cliente que efetuou a solicitação (6), por fim é enviada a resposta para o cliente (7).

2.1.4.3 Protocolo Anti-Entropy

O protocolo *Anti-Entropy* é responsável por garantir que as réplicas estejam sincronizadas nos controladores. É baseado em um algoritmo simples no qual um controlador escolhe aleatoriamente outro controlador no *cluster* para enviar suas informações. Seu processo é executado através de uma conexão ponto a ponto onde envia uma mensagem contendo o registro de data e hora de cada entrada para comparar o conteúdo (BANNOUR *et al.*, 2018a).

Após a sincronização das mensagens trocadas, os armazenamentos são atualizados com base no registro tornando os dois controladores mutuamente consistentes. Isso garante que todos os controladores obtenham consenso de acordo com um modelo eventualmente consistente. Como pode ser observado na Figura 8, sempre que ocorre uma atualização no armazenamento gerenciado por um controlador, esta nova informação é transmitida para todos os outros controladores no *cluster*. Essa abordagem se mostra útil na fixação de controladores quando seu estado varia ligeiramente, e na sincronização de novos controladores recém ingressados no *cluster* (MUQADDAS *et al.*, 2017).

Figura 8 – Funcionamento do protocolo *Anti-Entropy*



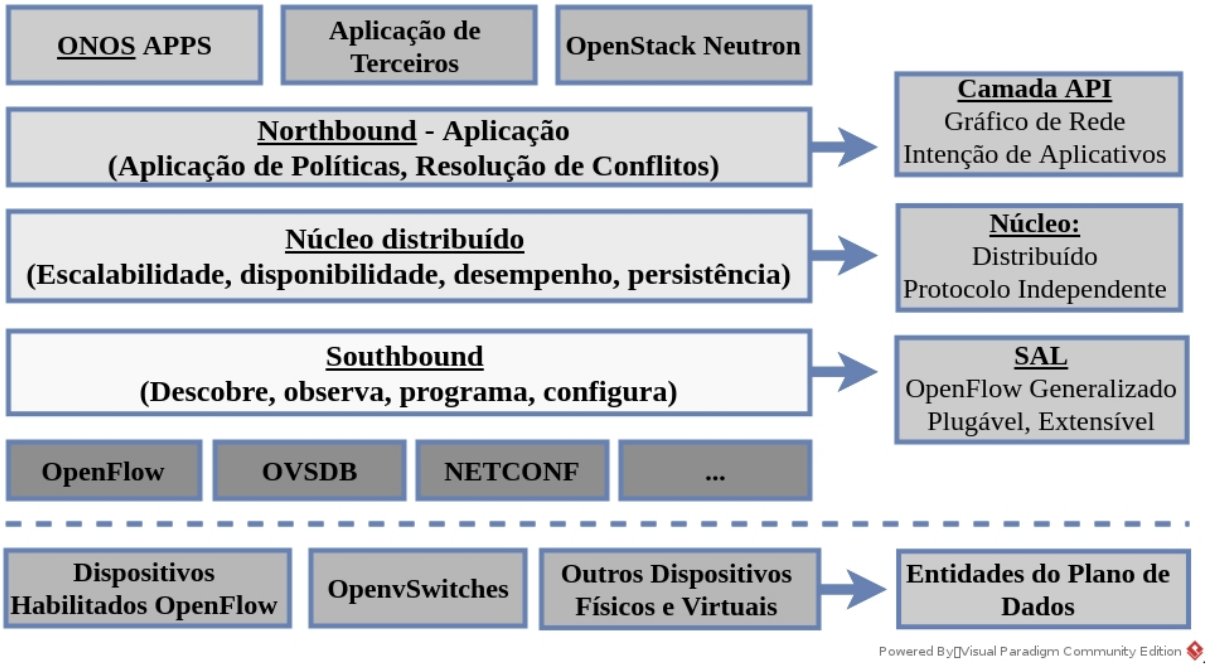
Fonte: Adaptado de (VIZARRETA *et al.*, 2020).

A Figura 8 mostra um modelo de consistência eventual que adota o estilo epidêmico de propagação. Esse modelo utiliza pares aleatórios compostos por vizinhos que comparam a sua versão de dados e concordam em um estado final apropriado quando atualizações simultâneas ocorrem (VIZARRETA *et al.*, 2020). Como é possível observar na figura, o controlador recebe uma atualização (1), transmite para seu vizinho (2), que por sua vez concorda com a atualização (3), o mesmo replica para seu vizinho (4), que também concorda com a atualização (5), chegando ao estado final também conhecido como reconciliação.

2.2 OPEN NETWORK OPERATING SYSTEM (ONOS)

O ONOS é um controlador SDN que oferece um sistema operacional de rede. O projeto é de código aberto desenvolvido em Java e pertence a *Open Networking Lab* (ON.Lab). A arquitetura do ONOS fornece APIs para o desenvolvimento de aplicativos, como por exemplo, gerenciamento e monitoramento de rede fornecendo um ambiente de virtualização, isolamento e acesso seguro. Devido a sua arquitetura flexível, permite que novos *hardwares* sejam facilmente integrados a sua estrutura (SUBRAMANIAN; VORUGANTI, 2016). A Figura 9 representa a sua arquitetura operacional.

Figura 9 – Arquitetura Operacional ONOS



Fonte: Adaptado de (SUBRAMANIAN; VORUGANTI, 2016).

A arquitetura do ONOS é projetada para requisitos de desempenho, alta disponibilidade e escalabilidade. De acordo com Subramanian e Voruganti, (2016) os principais componentes do ONOS são:

- Núcleo Distribuído: é onde fica o sistema operacional SDN, que foi projetado para ser executado em um *cluster* implementando requisitos de rede como agilidade, tolerância a falhas e alto desempenho com base nas demanda de aplicativos;
- *Northbound* APIs: fornece serviços de configuração e gerenciamento para o desenvolvimento de aplicativos SDN. Permite que aplicativos especifiquem seus requisitos de controle de rede na forma de políticas e facilita o desenvolvimento de aplicativos;
- *Southbound* APIs: fornece *plugins* de protocolos como o OpenFlow utilizado para a comunicação com dispositivos de rede.

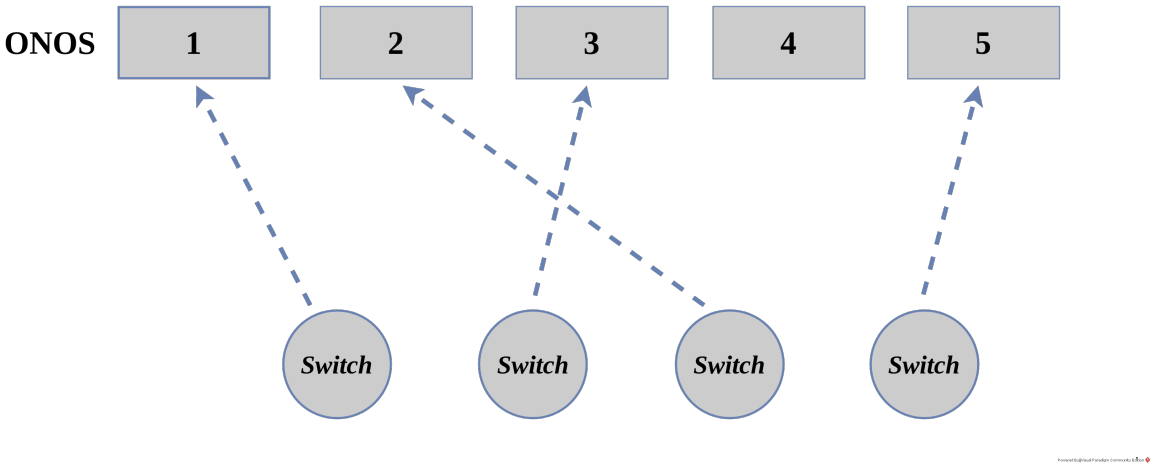
Um *cluster* ONOS utiliza os protocolos de consistência *Anti-Entropy*, Raft ou ambos. Eles são adaptados para garantir um nível específico de consistência no gerenciamento de suas *stores* distribuídas. De acordo com Muqaddas *et al.* (2017), o armazenamento das informações é composto por estruturas de dados distribuídas onde suas principais *stores* são:

- *Mastership Store*: gerenciado pelo Raft e é responsável por manter o mapeamento entre cada *switch* para seu controlador;
- *Network Topology Store*: gerenciado pelo *Anti-Entropy* e é responsável por descrever a topologia da rede em termos de enlaces e *switches*;
- *Flow Store*: gerenciado pelo *Anti-Entropy* e é encarregado pelas atualizações dos fluxos do controlador líder para o controlador seguidor ao detectar mudanças na tabela de fluxo;
- *Host Store*: incumbido de manter a lista de *hosts* da rede sendo gerenciado pelo Raft;
- *Application Store*: gerencia o estoque de aplicativos e utiliza o *Anti-Entropy*;
- *Intent Store*: gerencia o estoque de *intents*, que fazem parte da estrutura do ONOS e definem as políticas de operação da rede utilizadas pelos aplicativos e utiliza o *Anti-Entropy*;
- *Component Configuration Store*: responsável por todo o armazenamento do sistema para vários componentes de *software*. Emprega o *Anti-Entropy*;
- *Network Configuration Store*: armazena as configurações de rede inseridas no ONOS pela interface *northbound* como por exemplo a API REST, ou interface *southbound* como por exemplo o *OpenFlow*. Opera com o Raft;
- *Security Mode Store*: encarregado do gerenciamento de permissões concedidas aos aplicativos que utilizam o Raft. No entanto as violações de segurança são gerenciadas pelo *Anti-Entropy*.

Percebe-se que o ONOS usa o Raft ou o *Anti-Entropy* dependendo da estrutura de dados que vai ser armazenada.

Em versões anteriores a 1.14 os nós formavam um *cluster* comunicando-se uns com os outros e elegendo um líder através do Raft. Sua configuração utiliza um *script* que lista os pares de cada nó e a distribuição do Raft no *cluster* formando a arquitetura operacional, que pode ser observada na Figura 10 (ONOS, 2019).

Figura 10 – Arquitetura Operacional do *cluster* ONOS



Fonte: Adaptado de (HALTERMAN, 2018).

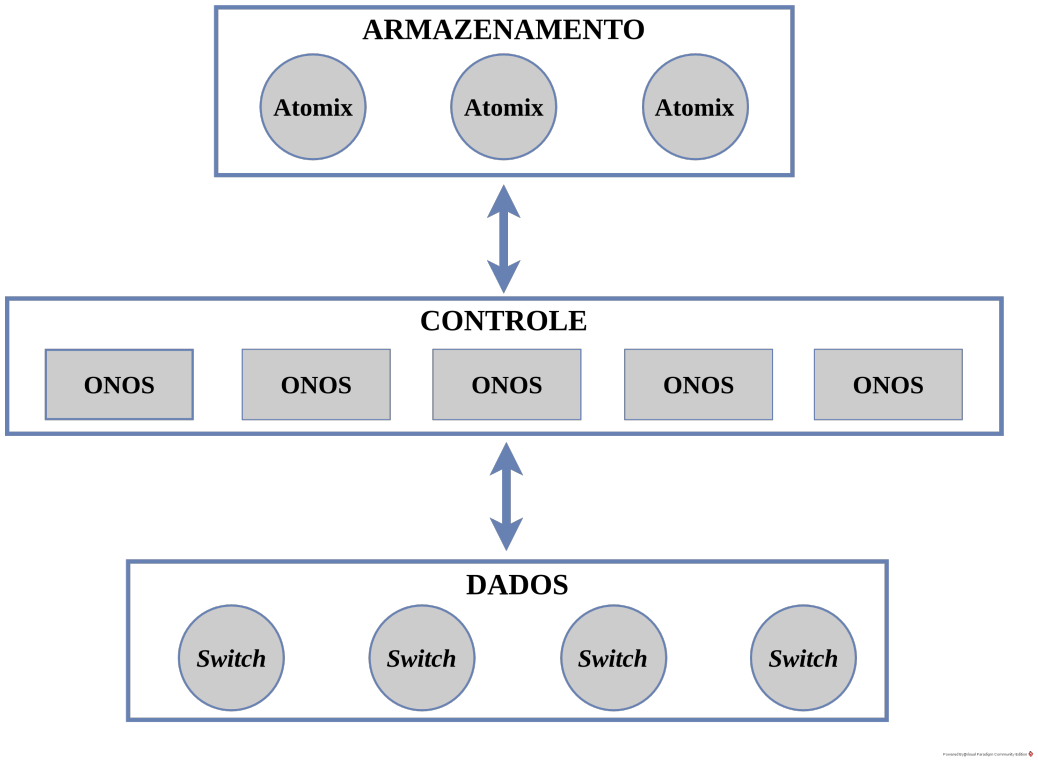
Como pode ser observado na Figura 10, em versões anteriores a 1.14, o *cluster* ONOS é composto apenas por controladores. Um controlador líder é responsável pelas decisões e consistência da rede. A comunicação do *cluster* é feita por meio da mensagem *heartbeat* do protocolo Raft, que consiste no envio de mensagens constantes entre os controladores, e se uma mensagem não for recebida, o par é marcado como inativo.

Esse modelo de *cluster* traz alguns problemas como particionamento e replicação. Cargas nas partições Raft da interface *southbound* também podem ocasionar atraso nas mensagens *heartbeat* iniciando eleições de líderes no *cluster*. Nesse modelo também é necessário um número mínimo de nós sempre tem que ser mantido (HALTERMAN, 2018). Com o objetivo de solucionar os problemas da arquitetura anterior, o projeto ONOS desenvolveu a arquitetura do *cluster* ONOS gerenciado pelo *framework* Atomix.

2.3 ATOMIX

O Atomix é um *framework* Java desenvolvido para o gerenciamento e comunicação de sistemas distribuídos escaláveis. Foi inicialmente desenvolvido pelo projeto ONOS e oferece opções para construir diferentes arquiteturas de *clusters* (ATOMIX, 2021). O Atomix é baseado no protocolo Raft mas fornece opções que vão além do algoritmo principal do Raft, como por exemplo, o acesso fácil ao estado da rede pelos aplicativos de controle. O ONOS utiliza o Atomix como uma loja que oferece um conjunto rico de primitivas de programação utilizados no gerenciamento do estado distribuído do ONOS (PETERSON CARMELO CASCONE; DAVIE, 2020). A Figura 11 mostra a arquitetura operacional do Atomix.

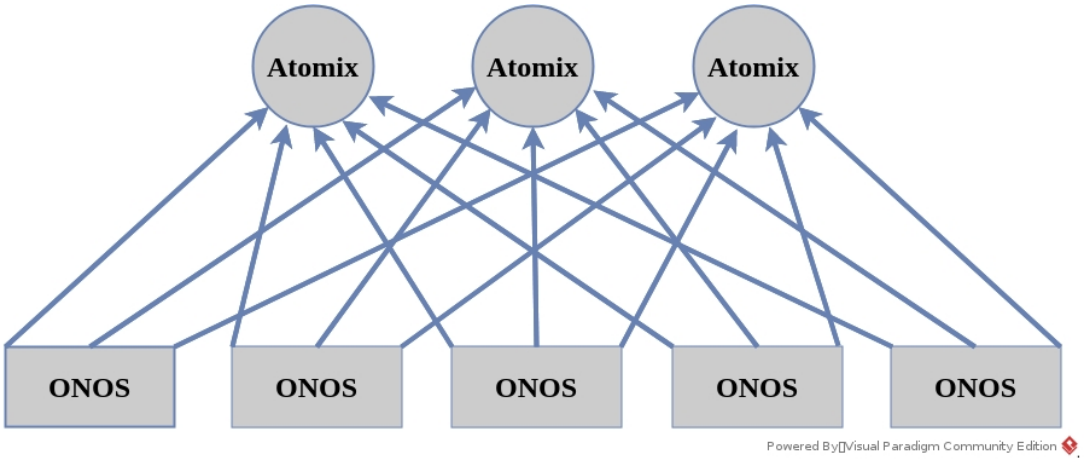
Figura 11 – Arquitetura Operacional do Atomix



Fonte: Adaptado de (HALTERMAN, 2018).

Como pode ser observado na Figura 11, o Atomix separa o armazenamento do controle. Para resistir a falhas dos controladores ONOS sem a perda de dados é adicionada uma estrutura Atomix externa para o armazenamento e gerenciamento. Como pode ser observado na Figura 12, o gerenciamento do *cluster* migrou para o Atomix juntamente com o armazenamento alterando a estrutura de comunicação do *cluster*. Isso tornou a estrutura fortemente consistente suportando uma queda simultânea de todos os controladores ONOS (HALTERMAN, 2018).

Figura 12 – Arquitetura Operacional ONOS Atomix



Fonte: Adaptado de (HALTERMAN, 2018).

Em um *cluster* Atomix, o protocolo Raft é movido para o Atomix removendo qualquer requisito para associação ao *cluster* do controlador. Os nós não tem conhecimento de seus pares ou partições disponíveis. Sua configuração é utilizada apenas para localizar e conectar ao Atomix para descobrir uns aos outros e armazenar o estado. Dessa forma o *cluster* poderia tolerar a falha de todos os controladores (ONOS, 2019).

A partir da versão 3.0 do Atomix a mensagem *heartbeat* do protocolo Raft foi substituído pelo protocolo *Scalable Weakly-consistent Infection-style Process Group Membership* (SWIM). Diferente da mensagem *heartbeat* que quando um nó não responde ele é marcado como inativo, o protocolo SWIM ao detectar uma falha tenta outras rotas para determinar se o problema está no nó ou por exemplo em um enlace. De acordo com Halterman (2018) o protocolo SWIM reduz a carga da rede, melhora a escalabilidade e reduz falsos positivos. O Quadro 1 mostra suas principais diferenças.

Quadro 1 – Comparativo da mensagem *heartbeat* do Raft e SWIM

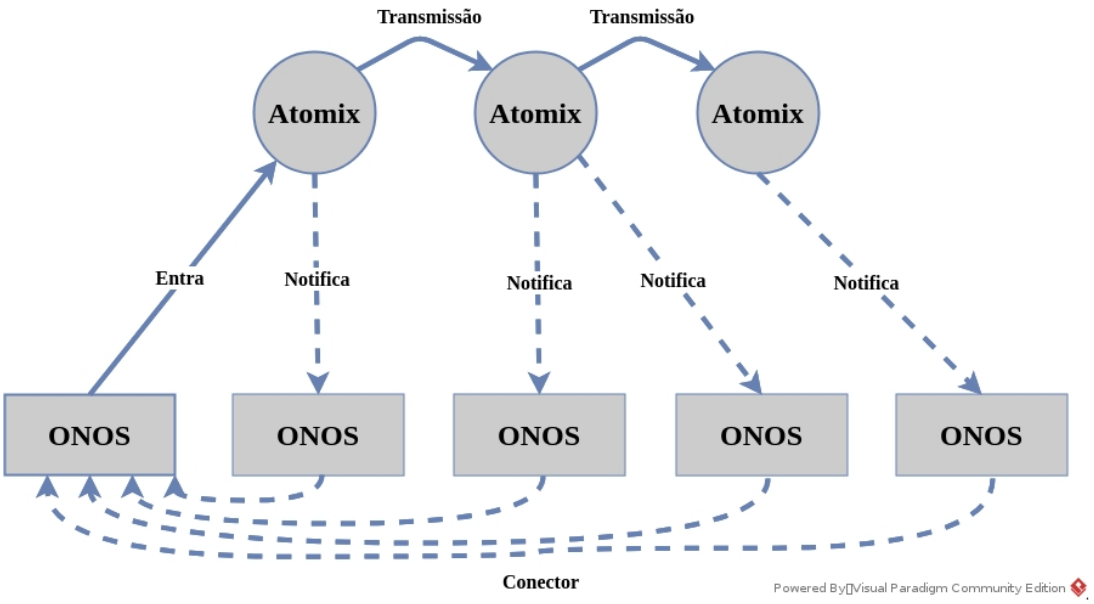
Mensagem *heartbeat* do Raft	**Protocolo SWIM**
Não dimensiona bem	Escala bem
Crescimento exponencial no tráfego	Crescimento linear no tráfego da rede
Não lida com partições de rede simples	Lida com partições de rede básicas
Falsos positivos mais frequentes	Evita falsos positivos
Pode detectar falhas mais rapidamente	Pode demorar mais para detectar falhas

Fonte: Autoria Própria.

Ao ser iniciado, um nó ONOS se conecta ao *cluster* Atomix externo para armazenamento e coordenação. Após o nó ONOS notificar o Atomix de sua existência e localização, as informações são replicadas para todos os demais nós ONOS pertencentes ao *cluster* que subsequentemente se conectam diretamente ao novo nó para a comunicação ponto a ponto. Esse processo pode ser observado na Figura 13. Sua configuração requer três componentes (ONOS, 2019):

1. *Cluster*: especifica como os nós Atomix descobrem e se comunicam uns com os outros;
2. *Management Group*: responsável pela configuração do grupo de partição para o gerenciamento de primitivos;
3. *Partition Groups*: atua na configuração do conjunto de grupos de partições, replicação e armazenamento.

Figura 13 – Descoberta do *cluster* Atomix



Fonte: Adaptado de (HALTERMAN, 2018).

Através de um conjunto de APIs de comunicação, o Atomix oferece uma série de primitivos de alto nível para a construção de sistemas distribuídos resolvendo problemas como particionamento e replicação, carga no controlador e eleição de líder (CAMPANELLA *et al.*, 2020). Cada primitivo pode ser replicado utilizando uma variedade de protocolos de sistemas distribuídos configuráveis, como os protocolos Raft e *Anti-Entropy* discutidos anteriormente (ATOMIX, 2021). De acordo com Peterson *et al.* (2020) o Atomix oferece suporte para:

- Estruturas de dados distribuídas, como por exemplo, árvores, contadores, mapas e conjuntos;
- Comunicação distribuída com mensagens diretas e publicação/assinatura;
- Coordenação distribuída que inclui eleições de líderes;
- Gerenciamento de membros do grupo.

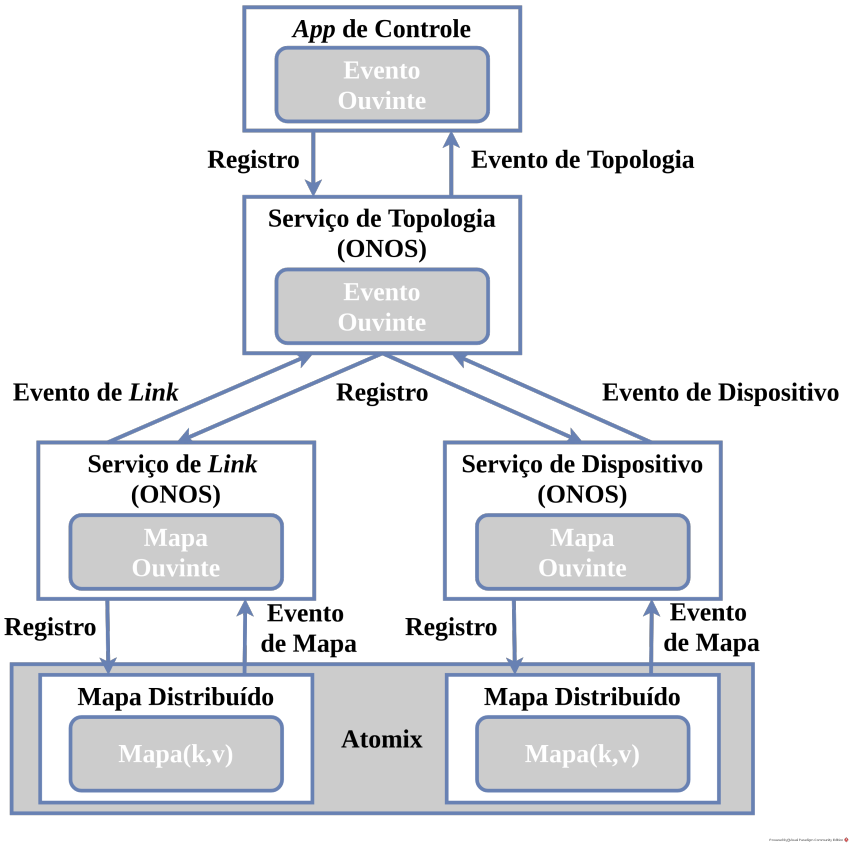
No Atomix existem duas categorias de primitivas distribuídas: dados primitivos e primitivas de coordenação.

Dados primitivos consistem em estruturas de dados simples para replicar o estado. Dois exemplos de dados primitivos são *AtomicMap* e *DistribuitedMap*. O primitivo *AtomicMap* executa atualizações *atomic* que utiliza bloqueios otimistas, dessa forma as operações são garantidas pelo *atomic*, por exemplo, esse primitivo fornece suporte ao protocolo *MultiRaftProtocol*. *Atomic*

consiste em uma variável Java onde as operações sempre são executadas juntas, desse modo todos executam juntos ou nenhuma é executa, dessa forma uma operação *atomic* não pode ser parcialmente concluída. Diferente do *AtomicMap*, o primitivo *DistributedMap* não requer consistência forte, desse modo possui suporte a protocolos de consistência eventual como por exemplo o primitivo *AntiEntropyProtocol*. Ambos modelos tem suporte a notificações baseadas em eventos de mudanças no mapa correspondente (PETERSON CARMELO CASCONE; DAVIE, 2020) (ATOMIX, 2021).

Primitivas de coordenação consistem em objetos para coordenar as mudanças de estado entre os nós. Um exemplo é o primitivo *LeaderElection* que é utilizado para eleger um único líder entre um conjunto de candidatos. Uma máquina de estado replicada é responsável pela detecção de falhas e eleição de candidatos. Esse primitivo fornece suporte ao protocolo *MultiRaftProtocol* (ATOMIX, 2021). A Figura 14 mostra o exemplo dos serviços de Topologia, Dispositivo e enlace implementado no Atomix.

Figura 14 – Exemplo dos serviços de Topologia, Dispositivo e enlace implementado no Atomix



Fonte: Adaptado de (PETERSON CARMELO CASCONE; DAVIE, 2020).

O ONOS é fundamentado no Atomix por meio de um conjunto básico de tabelas (mapas) empacotados como serviços. Os serviços são utilizados para o controle de aplicativos e outros

serviços como enlaces, dispositivos e topologia. Como pode ser observado na Figura 14, o serviço de topologia acessa indiretamente os mapas definidos pelos serviços de dispositivos e enlaces removendo a necessidade de ter um mapa associado a ele. Dessa forma o acesso do estado da rede é somente leitura e de baixa latência, e o armazenamento do gráfico de rede resultante na memória é feito em *cache*. Para garantir a visão dos aplicativos sobre a mesma árvore de transmissão, uma árvore de abrangência do gráfico é calculada (PETERSON CARMELO CASCONE; DAVIE, 2020).

2.4 TRABALHOS RELACIONADOS

A necessidade da arquitetura distribuída tem sido defendida na literatura resultando em várias pesquisas e experimentos. Dentre os problemas pesquisados, a consistência dos controladores distribuídos trouxe desafios. Alguns autores tem estudado a quantidade de tráfego gerado pelos protocolos de consistência, o que afeta diretamente o dimensionamento dos enlaces entre os controladores. Os protocolos *Anty-Entropy* e Raft presentes nos controladores ONOS e ONOS/Atomix trazem abordagens diferentes para tratar esse problema e a seguir serão analisados alguns trabalhos relacionados com o problema citado acima.

2.4.1 Trabalhos Relacionados com a Proposta

O estudo de Müller e Köhler-Bussmeier (2021) compara as arquitetura ONOS, e ONOS/Atomix em termos de disponibilidade. Através de um *cluster* de controladores do ONOS e ONOS/Atomix o autor executa testes de disponibilidade como a taxa de atualização do *cluster*, falha de enlace e sua taxa de recuperação e a falha de *hardware* e sua taxa de recuperação. A validação do ambiente é feita através da ferramenta GreatSPN que é utilizada na validação e avaliação de desempenho de sistemas distribuídos usando redes de Petri. O autor destaca o ganho de disponibilidade da arquitetura gerenciada pelo Atomix quando comparada ao modelo anterior, principalmente na adição de nós e enlaces devido a seu grau de replicação mais alto. Desse modo o autor recomenda a utilização de versões mais novas do controlador ONOS em conjunto com o Atomix porem salienta como projeto futuro um ambiente mais realista (MÜLLER; KÖHLER-BUSSMEIER, 2021).

Em Cho *et al.* (2019) é proposto um *Gateway* Independente (IGW) em um *cluster* SDN. Os *cluster* SDN e Atomix necessitam troca de mensagem em tempo real para manter sua

consistência e funcionamento. De acordo com o autor, quando o plano de dados se torna mais amplo, o *cluster* não consegue entregar suas mensagens em tempo real. Então a falha de um controlador afetaria os demais. Para tentar resolver o problema do mapeamento em tempo real o autor efetua conexões VXLANs *(Virtual Extensible Local Area Network)* em ambiente de rede IP ao invés de entre *host*. O experimento compara um *cluster* ONOS, um *cluster* ONOS/ Atomix e outro com a proposta IGW no quesito desempenho. O autor conclui que os modelos de *cluster* ONOS e ONOS/Atomix não são projetados para expansão em ambientes de rede IP ocasionando a deterioração do desempenho devido o aumento de mensagens de sincronização dos controladores. O IGW foi projetados para redes IP, então as mensagens de solicitações de troca entre os controladores ocorrem no IGW. O *cluster* ONOS pode compreender que havia troca de mensagens simultâneas entre os controladores. Porem o *cluster* ONOS/Atomix mostra que não há um armazenamento compartilhado no IGW ocasionando o aumento no número de mensagens (CHO *et al.*, 2019).

Com o objetivo de desenvolver uma estrutura de IoT para gerenciar a escala e sistema distribuído utilizando virtualização de funções de rede, o trabalho de e Gonzalez; Flauzac e Nolot (2019) utiliza um *cluster* ONOS/Atomix. O autor virtualiza um *cluster* com o controlador ONOS gerenciado pelo Atomix com o proposito de utilizar a estrutura para simulações de *testbed*. O *testbed* desenvolvido foi capaz de realizar simulações com mais de 1.000 dispositivos mostrando o alto grau de escalabilidade e alta tolerância a falhas do ONOS/Atomix. A avaliação do *cluster* apresentou adaptabilidade eficaz na integração com ferramentas de testes e sistema operacional embutido para IoT (GONZALEZ *et al.*, 2019).

O trabalho de Zhang *et al.* (2017) ilustram alguns cenários de falhas de rede que podem surgir com a utilização do Raft em um *cluster* SDN. Seu objetivo consiste em demonstrar como esses cenários de falhas podem afetar seriamente as operações do Raft em que no melhor caso é reduzido significativamente o tempo de operação disponível e no pior caso a incapacidade de alcançar um consenso e não eleger um líder. O autor compara resultados obtidos com a proposta *Vanilla* Raft e Raft apresentando o diagrama de mudança de líder, estatísticas de tentativas fracassadas dos clientes ao acessar o líder do *cluster* e estatísticas de disponibilidade do *cluster*. Os resultados obtidos revelam a eficácia da proposta do autor na melhoria da disponibilidade de liderança do Raft em um *cluster* SDN (ZHANG *et al.*, 2017).

O trabalho de Campanella *et al.* (2020) demostra a utilização do controlar ONOS com Atomix em uma rede óptica com *Hardware* real. O estudo simula três tipos de falhas; (1) Perda

da configuração que emula uma reinicialização do dispositivo, uma perda de configuração ou uma configuração incorreta na qual um determinado serviço é removido da rede; (2) Corte de fibra para simular a falha de enlaces; e (3) Falha do dispositivo para analisar a recuperação de falhas. O estudo demostrou que o Atomix oferece alta disponibilidade e proporciona um nível de robustez contra falhas de rede (CAMPANELLA *et al.*, 2020).

O estudo de Hanmer *et al.* (2018) faz uma análise do protocolo Raft implementado com ONOS 1.10 e Atomix 2.0. O autor destaca que algoritmos de consistência forte como o Raft podem não se comportar bem em condições de sobrecarga afetando negativamente a disponibilidade da rede. Foi observado que o *cluster* ONOS se torna indisponível quando há sobrecarga de mensagens entre os controladores que afetam o protocolo Raft. A sobrecarga provocou atraso nas mensagens *heartbeat* resultando na perda de solicitações e eventualmente no travamento de toda a rede. A mensagem *heartbeat* consiste no intervalo entre duas mensagens consecutivas enviadas pelo líder aos outros controladores. Quando o tempo limite desse intervalo é configurado muito baixo podem ocorrer eleições repetidas de líder tornando o sistema indisponível, e quando é muito alto, uma falha do líder não será detectada em um tempo hábil ocasionando a indisponibilidade da rede nesse período. O autor desenvolve uma solução para o problema através de um algoritmo que ajusta dinamicamente os tempos limites de eleição quando o sistema está sobrecarregado. O algoritmo forneceu uma estabilidade significativamente maior em condições de sobrecarga (HANMER *et al.*, 2018).

De acordo com Almadani; Beg e Mahmoud (2021) a falta de um padrão de comunicação entre controladores distribuídos apresenta um desafio na adoção da SDN em redes distribuídas de grande escala. O autor propõe um modelo de estrutura denominado DSF para sincronização de topologias utilizando um paradigma de publicação/assinatura em tempo real. Através da implementação do modelo DSF em controladores como o ONOS o autor apresenta uma comparação entre o controlador ONOS com DSF e o controlador ONOS baseado em Atomix. Os resultados dos experimentos mostram que ambos modelos em um *cluster* de 2 controladores recebem e processam a atualização do enlace dentro de 3 milissegundos, para 4 controladores o padrão foi semelhante. O autor destaca que configurações baseadas em Atomix com três agentes são sincronizadas mais rapidamente em *clusters* com um número menor de controladores, porem com o aumento de 3 controladores para 6 ocorre um atraso maior dos 3 últimos controladores de mais de 40 milissegundos em comparação com 30 milissegundos do *cluster* ONOS com DSF (ALMADANI *et al.*, 2021).

Segundo Liu; Steinert e Kostic (2018) a implementação de mais controladores pode contribuir com a redução na latência de controle evitando a sobrecarga de apenas um controlador. Mas consequentemente com o aumento no número de controladores ocorre o aumento no tráfego de controle independente do modelo de consistência ser forte ou fraco. De acordo com o autor ignorar esse tráfego de controle pode causar sobrecarga nos enlaces quando a topologia de rede é distribuída de forma inadequada. Desse modo o autor propõe uma formalização na otimização da distribuição do plano de controle através de uma ferramenta de previsão genérica para o protocolo OpenFlow. Os experimentos indicaram que ignorar o posicionamento do controlador sem considerar o tráfego de controle pode causar o uso excessivo da largura de banda onde nos piores casos variou de 20,1% a 50,1% a mais no tráfego em comparação com a abordagem desenvolvida pelo autor (LIU *et al.*, 2018).

O trabalho de Bannour; Souihi e Mellouk (2020) apresenta um modelo de consistência adaptável e contínua para a implementações de grande escala para controladores SDN. O objetivo do modelo é possibilitar a construção de aplicativos de redes centradas em informações (ICN) como por exemplo serviços de distribuição de vídeo. Desse modo o autor propõe transformar a técnica de replicação padrão utilizada no controlador ONOS em uma estratégia escalonável e inteligente por meio da consistência replicada por Quorum. A consistência de Quorum é utilizada principalmente em sistemas onde a consistência é mais importante que a disponibilidade mas razoavelmente disponível. A estratégia de adaptação de consistência utilizada pelo autor consiste principalmente em transformar a técnica de replicação otimista do ONOS em uma estratégia de replicação baseada no Quorum. O autor aponta que o *cluster* ONOS com a estratégia baseada em Quorum se mostrou eficiente em encontrar parâmetros de replicação de leitura e gravação aprimorados em tempo de execução. As configurações ajustáveis do Quorum alcançaram compensações equilibradas entre desempenho contínuo e requisitos de consistência. De acordo com o autor essas compensações em tempo real garantiram uma redução substancial na sobrecarga de leitura e gravação do controlador fornecendo soluções para mudanças frequentes no estado e conteúdo da topologia de rede. O autor destaca que mecanismos de consistência auto-adaptáveis e automatizados podem trazer benefícios a aplicações de redes distribuídas (BANNOUR *et al.*, 2020).

Os trabalhos relacionados acima descrevem estudos de análise de desempenho com os controladores ONOS e ONOS/Atomix, ou seja, mostram que é um tópico de pesquisa importante para a comunidade acadêmica e serviram de inspiração para o trabalho proposto. O trabalho

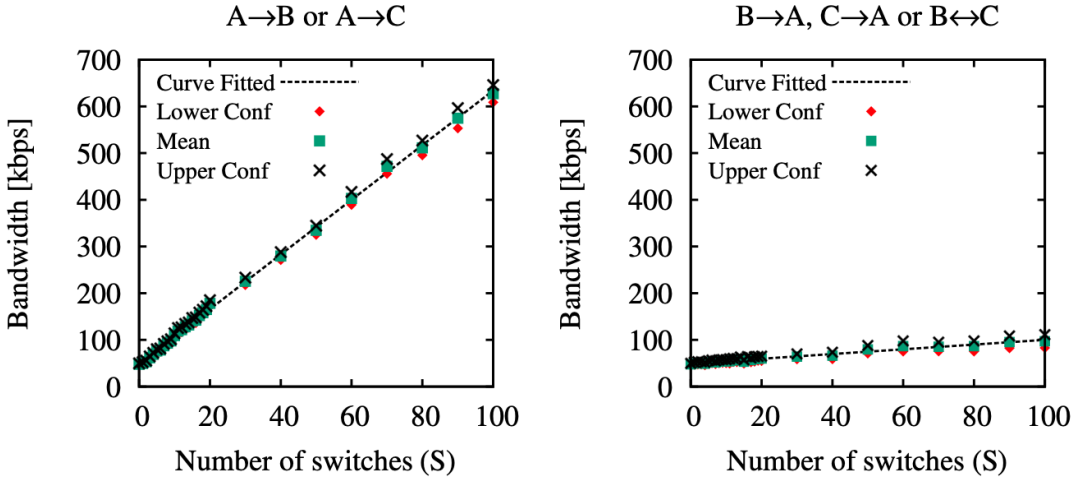
descrito a seguir está diretamente relacionado com essa proposta e é explicado com mais detalhes porque é usado como base para o trabalho atual e sera utilizado para a comparação com os resultados obtidos das simulações mostrados no capítulo 3 e 4.

2.4.1.1 Trabalho Relacionado com Modelo de Tráfego

O trabalho de Muqaddas *et al.* (2017) utiliza um *cluster* ONOS e realiza um estudo sobre o impacto dos protocolos de consistência no tráfego de dados entre os controladores. Para analisar o tráfego entre os controladores os autores desenvolveram um modelo analítico e fizeram simulações para avaliar o modelo. Os autores desenvolveram uma metodologia para obter equações que representam o tráfego de dados entre os controladores em função do número de *switches* e enlaces (MUQADDAS *et al.*, 2017).

O Gráfico 1 representa os resultados de uma topologia linear adicionada no controlador líder em um cenário com 3 controladores. O gráfico da esquerda mostra que o tráfego saindo do líder é maior do que o tráfego entre os controladores seguidores B e C, representado no gráfico da direita. Pode ser observado que o tráfego é diretamente proporcional ao número de enlaces e *switches*. O modelo desenvolvido pelo autor será descrito no próximo capítulo.

Gráfico 1 – Tráfego com 3 controladores



Fonte: Adaptado de (MUQADDAS *et al.*, 2017).

3 DESENVOLVIMENTO DO MODELO DE TRÁFEGO

Este capítulo descreve a escolha do ambiente de simulação desenvolvido para possibilitar a análise do tráfego no *cluster* ONOS/Atomix e a proposta de modelo de tráfego. A seção 3.1 descreve o ambiente desenvolvido e os *softwares* utilizados. A seção 3.2 explica a configuração de todo o ambiente. A seção 3.3 descreve o modelo de tráfego. Por fim a seção 3.4 apresenta a conclusão do capítulo.

3.1 DESCRIÇÃO DO AMBIENTE DESENVOLVIDO

Os testes foram realizados em um servidor IBM X3850 M2 com 4x Intel® Xeon™ Processor X7350 CPU @ 2.93GHz (4 núcleos) e 64 GB de memória RAM. O sistema operacional instalado no servidor foi o Ubuntu Server 20.04. Com o objetivo de construir um *cluster*, foram utilizados contêineres *Docker* com imagens do repositório oficial do ONOS pré configurados com as versões utilizadas. Os controladores testados foram o ONOS 1.13.10 Nightingale para o *cluster* ONOS e o ONOS 2.2.2 Sparrow com Atomix 3.1.5 para o *cluster* ONOS/Atomix. O Quadro 2 descreve as características do ambiente de simulação.

Quadro 2 – Características do ambiente de simulação

Servidor	IBM X3850 M2
Sistema Operacional	Ubuntu Server 20.04
Imagens Docker	Repositório Oficial ONOS
Controlador ONOS	ONOS 1.13.10 Nightingale
Controlador ONOS/Atomix	ONOS 2.2.2 Sparrow
Atomix	3.1.5
Mininet	2.2.2
OpenVSwitch	2.10.7
Memória RAM	64 GB
CPU	4x Intel® Xeon™ E7350 @ 16x 2.93GHz
Disco Rígido	2x 146GB 10k SAS

Fonte: Autoria Própria.

Todas as simulações utilizaram um ambiente virtualizado e o sistema operacional Ubuntu foi selecionado por sua confiabilidade e bom desempenho para a utilização de contêineres. Em cima do sistema operacional foi utilizado contêiner em vez de máquina virtual, visto que é composto por uma versão minimalista do sistema operacional, e, no caso do Linux, torna fácil a personalização da distribuição contendo os principais *softwares* necessários. Entre as várias opções de contêineres foi escolhido o Docker, porque consiste em uma das principais

tecnologias existentes para se trabalhar com contêineres. O baixo uso de recursos e velocidade no gerenciamento dos contêineres tornam o Docker uma excelente escolha para se trabalhar com o *cluster* necessário para a execução dos testes propostos.

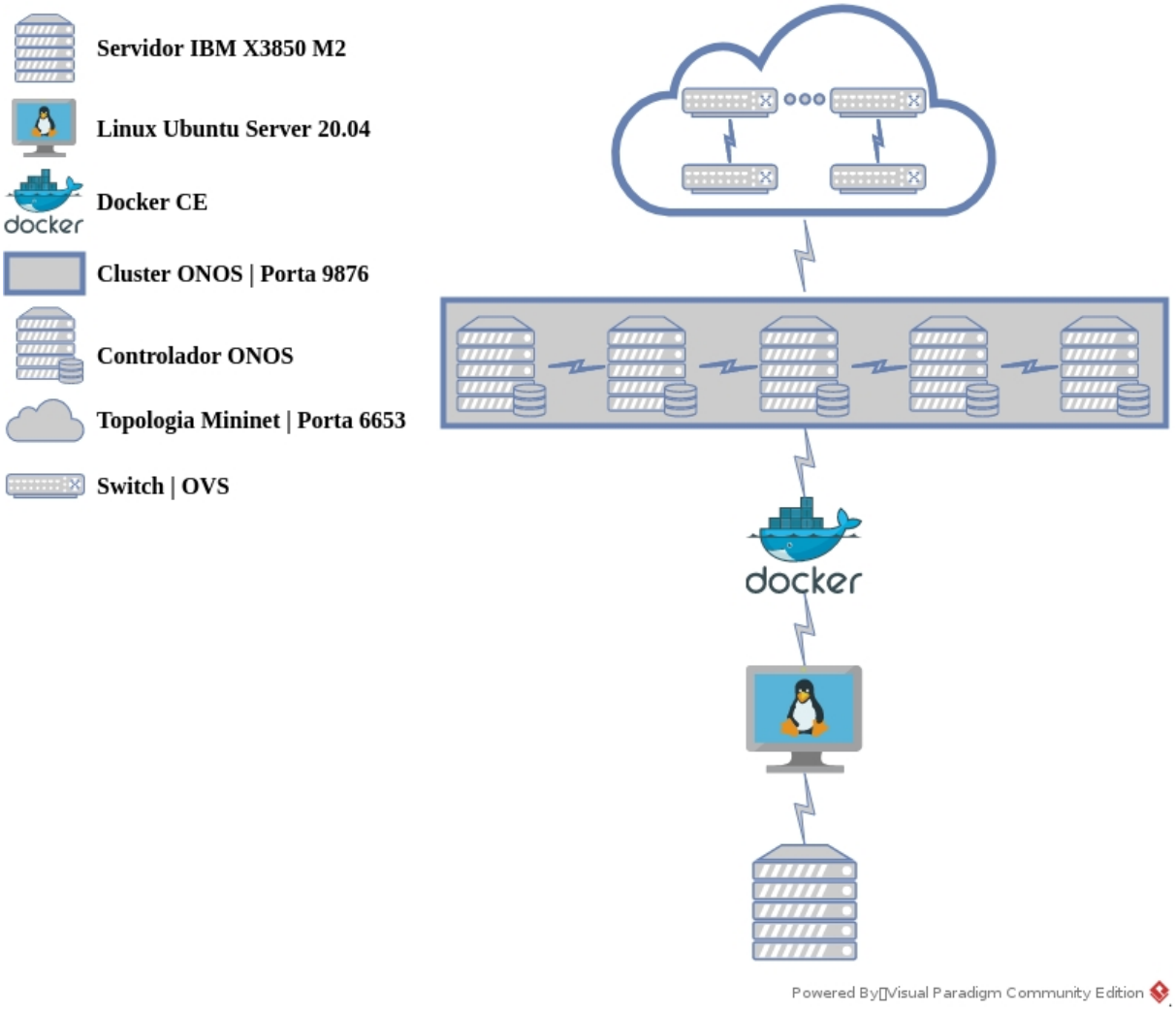
Existem várias versões do controlador ONOS. A versão do ONOS 1.13.10 Nightingale foi escolhida por ser a última versão que não utiliza o Atomix como arquitetura padrão para o *cluster*. A versão do ONOS 2.2.2 Sparrow foi seleciona devido a compatibilidade com Atomix e por se tratar da versão mais atual com suporte a longo prazo disponível no início do trabalho.

O Mininet é um emulador de rede utilizado para executar *hosts*, *switches*, roteadores e enlaces. Através da virtualização o Mininet faz a simulação de interfaces *ethernet*, dessa forma o seu comportamento é semelhante a uma rede composta por *hardware*. O suporte as topologias parametrizadas e flexíveis tornaram o Mininet a ferramenta utilizada na execução das topologias utilizadas para as simulações deste trabalho. O Mininet utiliza o *software* OpenvSwitch (OvS) para a virtualização das placas de rede utilizadas pelo *cluster*.

3.2 DESCRIÇÃO DA CONFIGURAÇÃO DO AMBIENTE

O ambiente proposto exigiu a configuração de duas arquiteturas distintas na configuração do *cluster*. Para a primeira arquitetura com a versão do ONOS 1.13.10 Nightingale foi utilizado o *script onos-form-cluster.sh* disponível no controlador para construção do *cluster*, que pode ser visto no Anexo A. Toda a comunicação do *cluster* é realizada através da porta 9876. A Figura 15 mostra a arquitetura do ambiente.

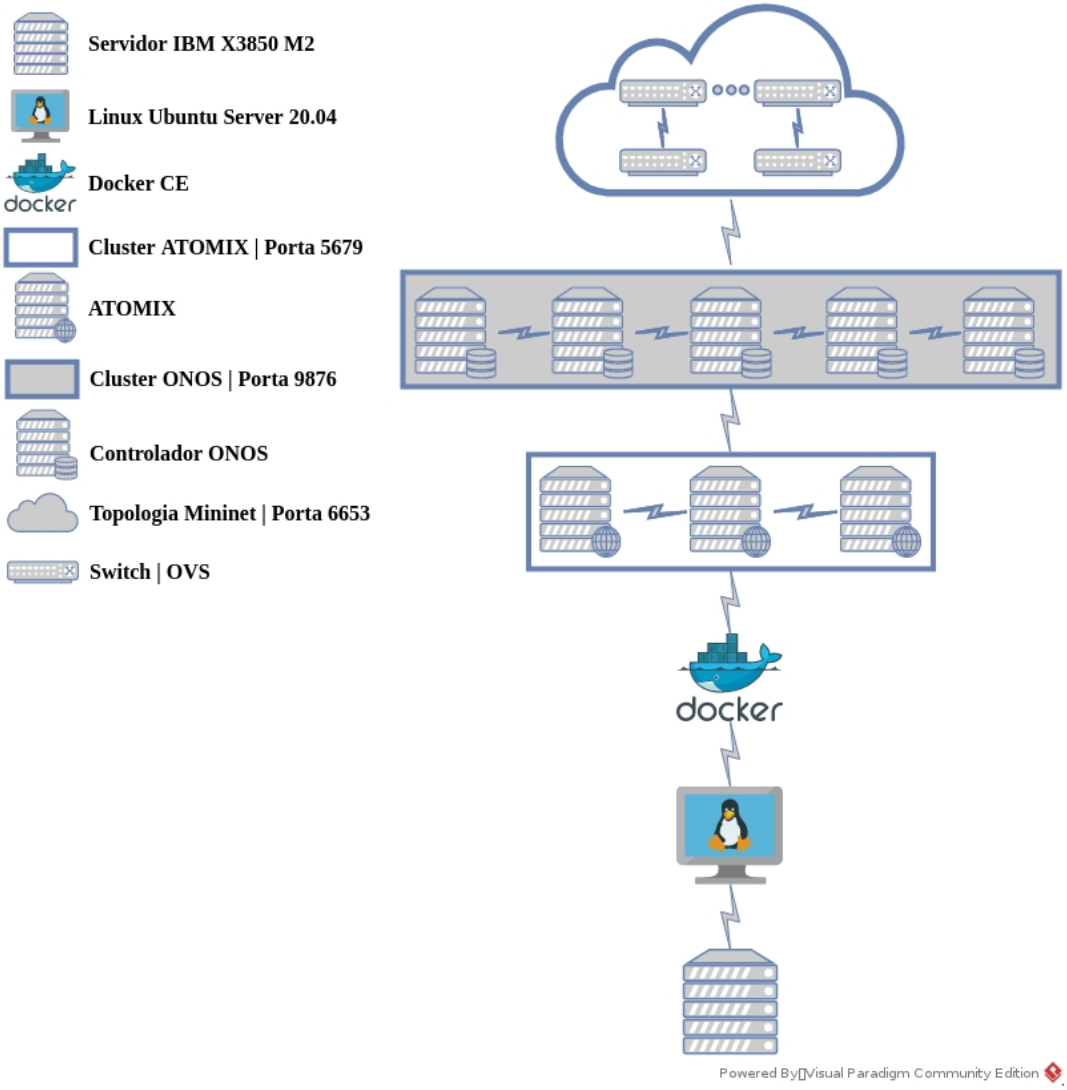
Figura 15 – Ambiente de Simulação do *cluster* ONOS



Fonte: Autoria Própria.

A segunda arquitetura consistiu em um *cluster* com o ONOS 2.2.2 Sparrow e Atomix 3.1.5. Para a configuração do *cluster* foram utilizados dois *scripts*, sendo o *atomix-gen-config* nos contêineres com Atomix e o *onos-gen-config* nos contêineres com o controlador ONOS, que podem ser vistos no Anexo B e C, respectivamente, ambos os *scripts* fornecidos pelo projeto ONOS. A comunicação entre os controladores do *cluster* se manteve na porta 9876 e a comunicação entre os nós do Atomix utiliza a porta 5679. A Figura 16 mostra a arquitetura do *cluster* ONOS/Atomix.

Figura 16 – Ambiente de Simulação do *cluster* ONOS/Atomix



Fonte: Autoria Própria.

Os *scripts* contendo as topologias utilizadas para ambas as arquiteturas foram desenvolvidos em Python. O *script* com a topologia para 20 controladores utilizado na elaboração do modelo está disponível no Apêndice A. Devido aos diferentes tipos de experimentos, foi necessária a personalização da topologia de rede de acordo com o número de controladores e *switches*, bem como a distribuição dos *switches* entre os controladores. Em todas as simulações foi utilizada a porta de número 6653 para a conexão entre o Mininet e o controlador.

Todas as medições foram executadas cinco vezes para cada número de *switches*. Após a inicialização do controlador, aguardou-se o carregamento e estabilização do *cluster*. Em seguida, a topologia foi iniciada e aguardado a normalização do fluxo de dados antes da captura ser iniciada. Por fim, foi iniciada a captura do tráfego através do *tshark* durante 300 segundos. Na arquitetura ONOS, os dados são capturados na porta 9876. Para a arquitetura ONOS/Atomix,

além da porta 9876, foi incluída a porta 5679, que compõe a interconexão entre os nós do Atomix.

3.3 MODELO DE TRÁFEGO

O cenário de simulação descrito nesta seção tem como objetivo medir o tráfego no *cluster* ONOS/Atomix e usar estes dados para desenvolver o modelo de tráfego. Foi usado como referência a metodologia de Muqaddas *et al.* (2017), que desenvolveu um modelo de tráfego para o *cluster* ONOS. A partir do citado trabalho foi desenvolvido o modelo de tráfego para o *cluster* ONOS/Atomix.

O Quadro 3 e Quadro 4 descrevem a notação utilizada nas equações do modelo de tráfego.

Quadro 3 – Notação para tráfego no cenário com 3 controladores x e y

Símbolo	Significado
B	vazão unidirecional genérica
b^0	vazão zero genérica
b^s	vazão unidirecional média por *switch*
b^l	vazão unidirecional média por enlace intra-domínio
b^d	vazão unidirecional média por enlace entre domínios, (compartilhado também pelo controlador de destino para o cenário de 3 controladores)
b^e	vazão unidirecional média por enlace entre domínios, (externo ao controlador de destino para o cenário de 3 controladores)
$B^I_{x\to y}$	vazão de x a y em topologia isolada
$B^L_{x\to y}$	vazão de x a y em topologia linear
$B^*_{x\to y}$	vazão de x a y em topologia estrela
$b^s_{x\to y}$	vazão média de x a y por *switch*
$b^l_{x\to y}$	vazão média de x a y por enlace intra-domínio

Fonte: Adaptado de (MUQADDAS *et al.*, 2017).

Quadro 4 – Notação que descreve a topologia da rede no cenário com 3 controladores. Sejam x, y dois controladores distintos, com $x, y \in \{C1, C2, C3\}$

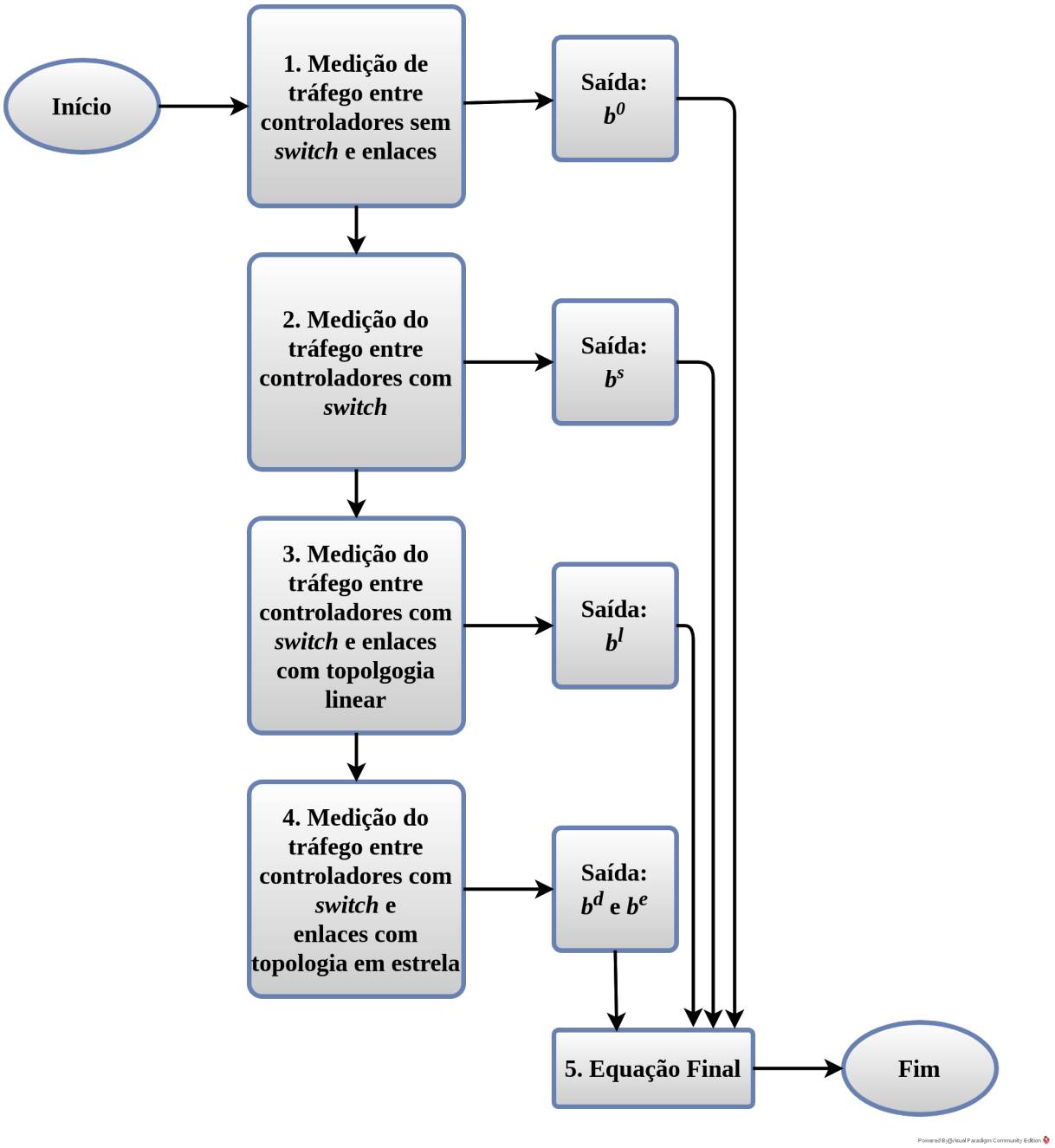
Símbolo	Significado
S_x	número de *switches* no domínio do controlador x
$S = S_{\mathrm{C1}} + S_{\mathrm{C2}} + S_{\mathrm{C3}}$	número total de *switches* na rede
L_x	número de enlace intra-domínio do controlador x
L_{xy}	número de enlaces entre domínios entre x e y
$L = L_{\mathrm{C1}} + L_{\mathrm{C2}} + L_{\mathrm{C3}+L_{\mathrm{C1C2}}+L_{\mathrm{C2C3}}+L_{\mathrm{C1C3}}}$	número total de enlaces na rede
L_{ID}	número total de enlaces entre domínios na rede

Fonte: Adaptado de (MUQADDAS *et al.*, 2017).

Para qualquer seleção de controladores distintos $x, y, z \in \{C1, C2, C3\}$(ou seja, tal que $x \neq y, x \neq z$ e $y \neq z$).

A Figura 17 mostra o fluxograma com as etapas usadas por Muqaddas *et al.* (2017) para desenvolver o modelo de tráfego do *cluster* ONOS, que são descritas a seguir em detalhes.

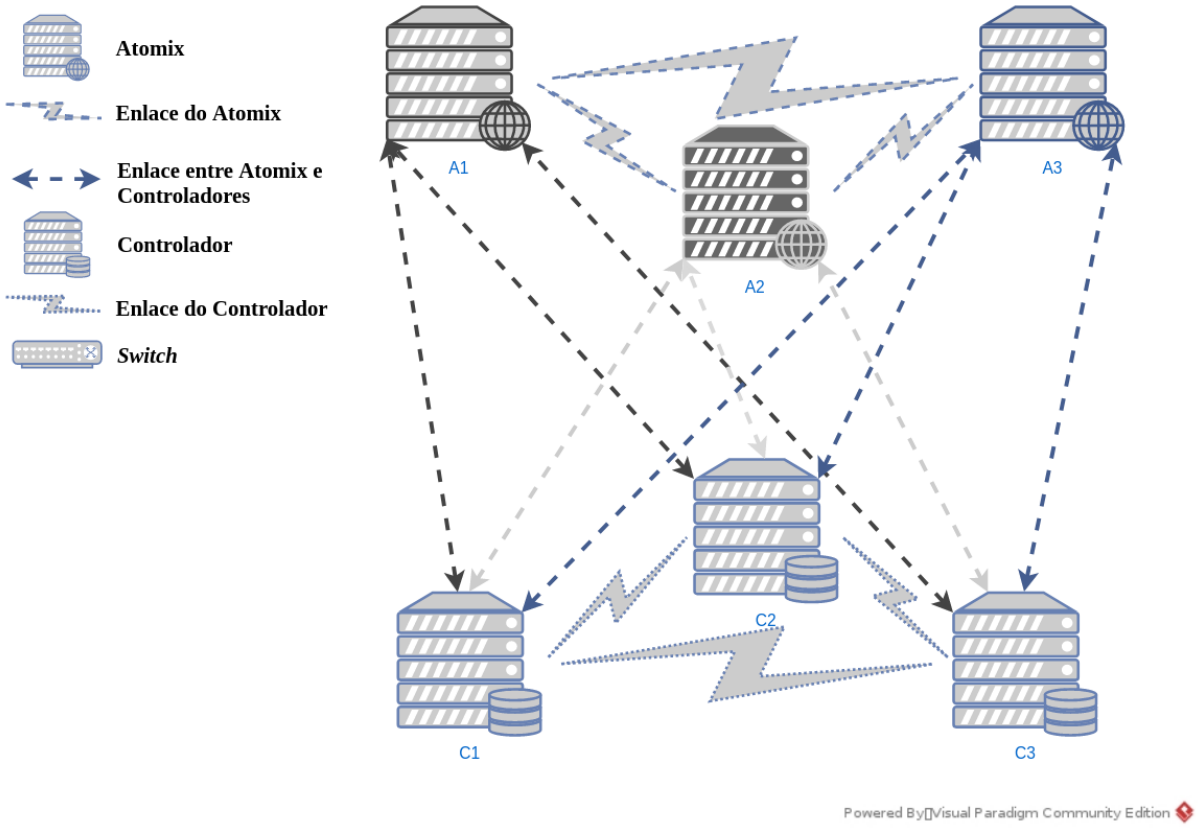
Figura 17 – Fluxograma do Modelo de Tráfego



Fonte: Autoria Própria.

1. O primeiro passo é determinar o valor do coeficiente b^0, que corresponde a vazão entre os controladores sem nenhum topologia. Como pode ser observado na Figura 18, para este experimento são iniciados todos os controladores mas não é carregada nenhuma topologia, assim sendo é possível obter a vazão entre os controladores devido aos protocolos de consistência *Anti-Entropy* e Raft.

Figura 18 – Medição de tráfego no *cluster* ONOS/Atomix sem *switch* e enlaces



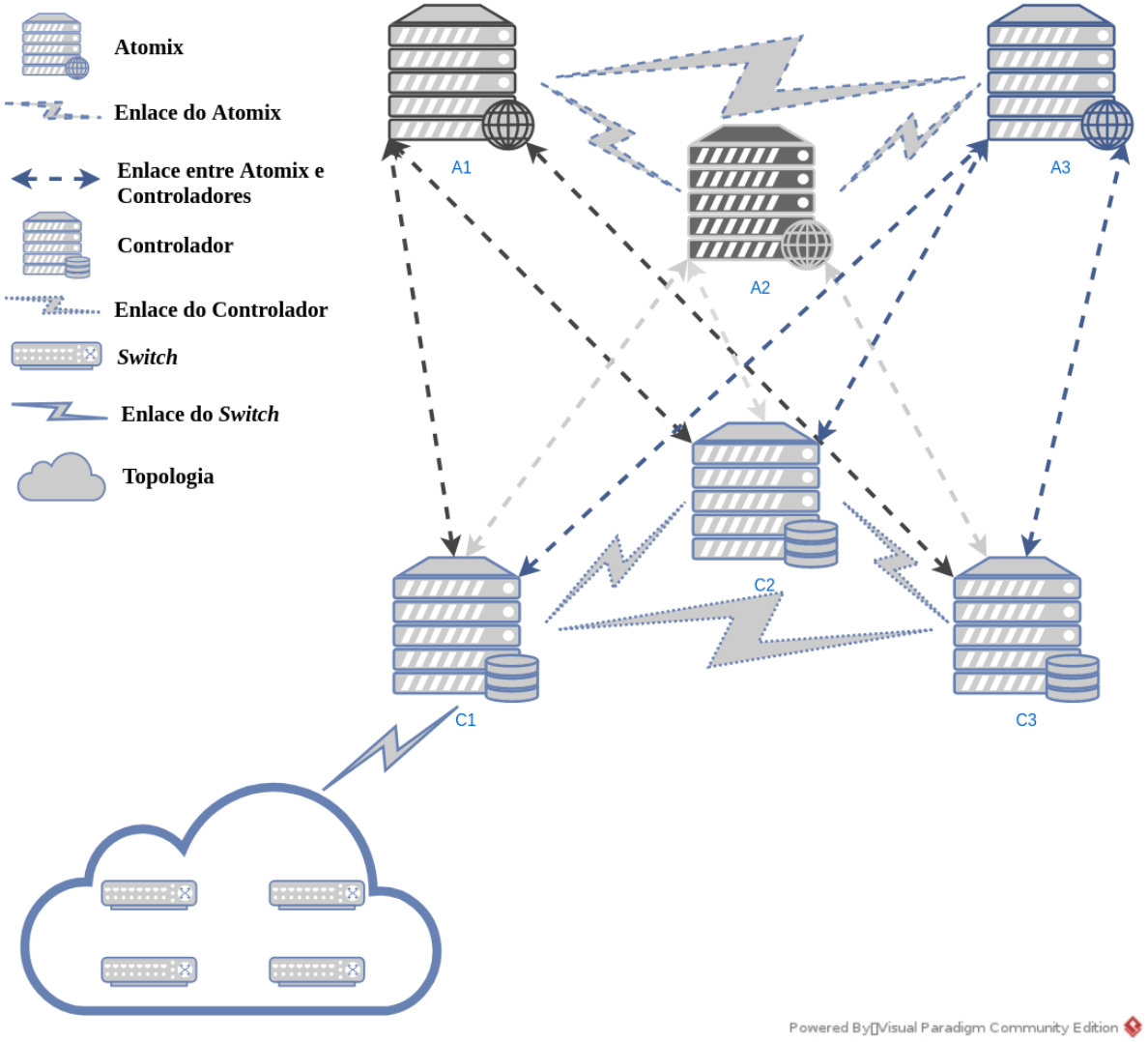
Fonte: Autoria Própria.

2. Esta etapa determina o valor do coeficiente b^s, que consiste na vazão unidirecional média por *switch*. Para obter esse valor foi necessária a execução de experimentos compostos apenas por *switches*, como pode ser observado na Figura 19. A Equação 1 é utilizada para determinar o valor de b^s no sentido $(C1 \rightarrow C2)$, e a Equação 2 é utilizada para determinar o valor do coeficiente b^s no sentido $(C2 \rightarrow C1)$.

$$B^I_{\mathrm{C1}\rightarrow\mathrm{C2}} = S \cdot b^s_{\mathrm{C1}\rightarrow\mathrm{C2}} + b^0 \tag{1}$$

$$B^I_{\mathrm{C2}\rightarrow\mathrm{C1}} = S \cdot b^s_{\mathrm{C2}\rightarrow\mathrm{C1}} + b^0 \tag{2}$$

Figura 19 – Medição de tráfego no *cluster* ONOS/Atomix com *switch*



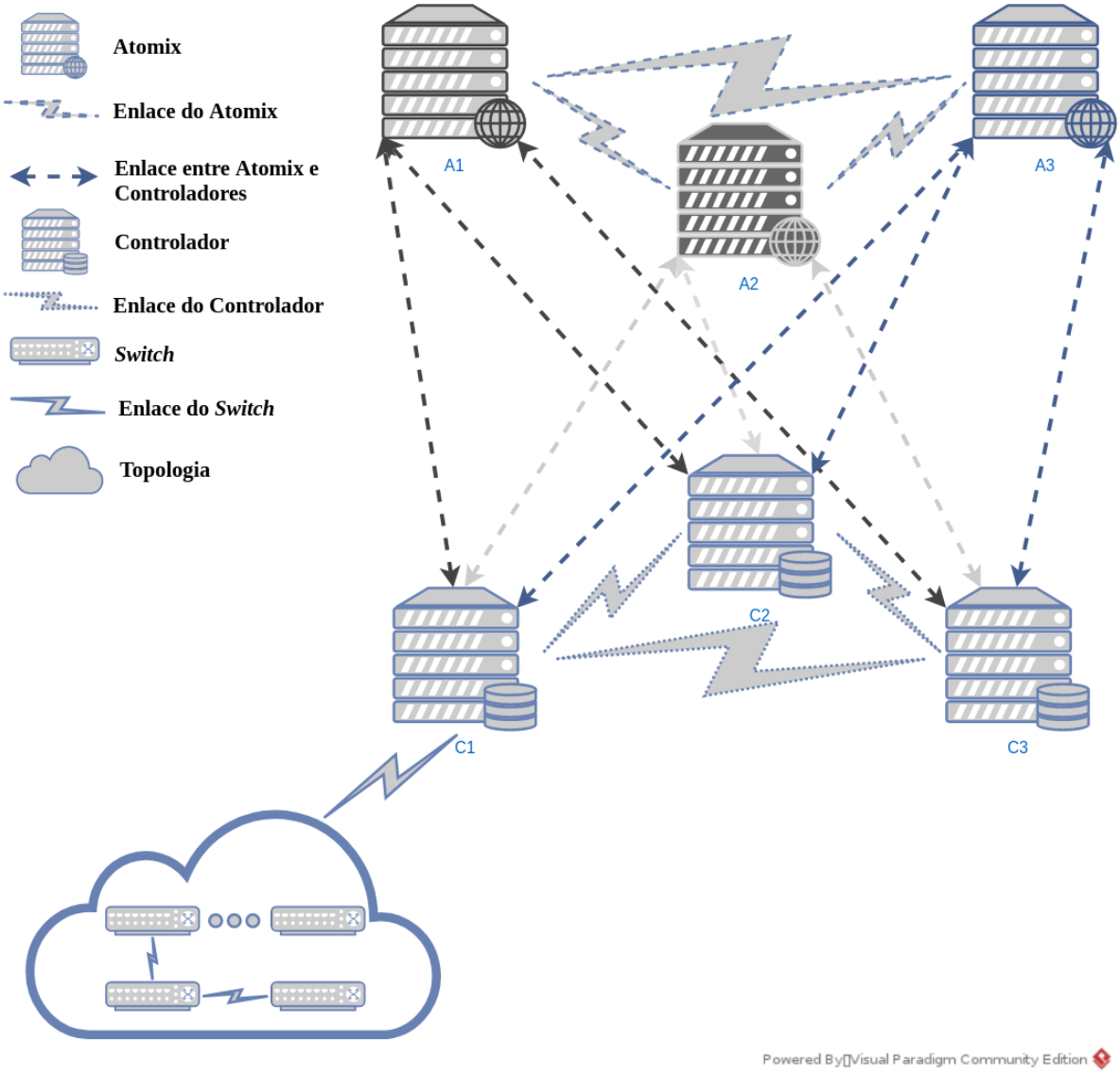
Fonte: Autoria Própria.

3. Esta etapa determina o valor do coeficiente b^l, que representa a vazão unidirecional média por enlace intra-domínios. Como pode ser observado na Figura 20, para este experimento é considerada uma topologia composta por *switches* e enlaces em topologia linear. A Equação 3 é utilizada para determinar o valor de b^l no sentido $(C1 \rightarrow C2)$, e a Equação 4 é utilizada para determinar o valor do coeficiente b^l no sentido $(C2 \rightarrow C1)$.

$$B^L_{\mathrm{C1}\rightarrow\mathrm{C2}} = S \cdot b^s_{\mathrm{C1}\rightarrow\mathrm{C2}} + (S-1) \cdot b^l_{\mathrm{C1}\rightarrow\mathrm{C2}} + b^0 \tag{3}$$

$$B^L_{\mathrm{C2}\rightarrow\mathrm{C1}} = S \cdot b^s_{\mathrm{C2}\rightarrow\mathrm{C1}} + (S-1) \cdot b^l_{\mathrm{C2}\rightarrow\mathrm{C1}} + b^0 \tag{4}$$

Figura 20 – Medição de tráfego no *cluster* ONOS/Atomix com *switch* e enlaces com topologia linear



Fonte: Autoria Própria.

4. Para estender o modelo a topologias com particionamento arbitrário é utilizado um modelo em estrela para determinar o valor da vazão entre domínios, que é representado pelos coeficientes b^d e b^e. Como pode ser observado na Figura 21, para determinar estes valores foi elaborado um experimento com uma topologia em estrela, onde foram adicionados (S-1) *switches* no controlador $C1$, um *switch* no controlador $C2$ e zero *switch* no controlador $C3$ com o objetivo de obter a vazão unidirecional média por enlace entre domínios.

   A Equação 5 é utilizada para determinar o valor de b^d e b^e no sentido ($C1 \rightarrow C2$), e a Equação 6 é utilizada para determinar o valor do coeficiente b^d e b^e no sentido ($C2 \rightarrow C1$). A Equação 7 é utilizada para determinar o valor de b^d e b^e no sentido ($C1 \rightarrow C3$), e

a Equação 8 é utilizada para determinar o valor do coeficiente b^d e b^e no sentido ($C3 \rightarrow C1$).

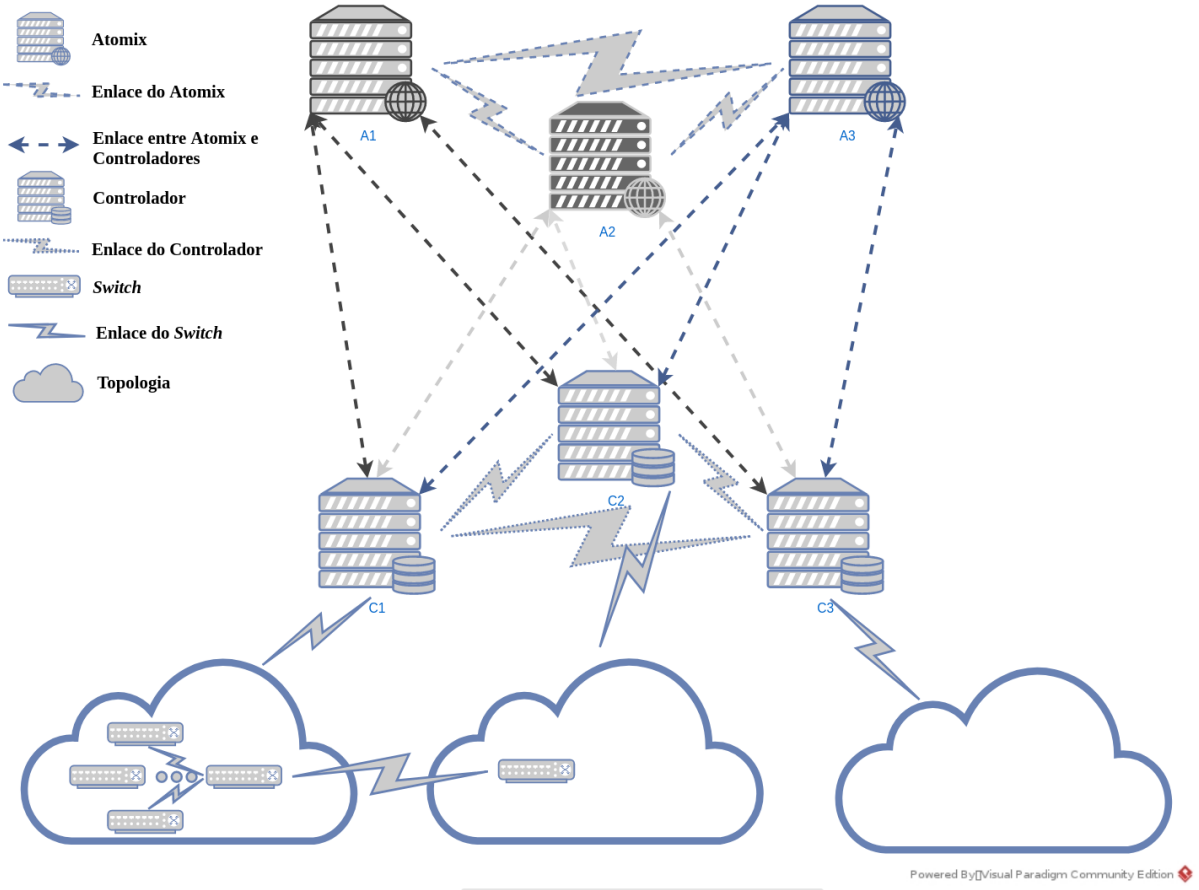
$$B^*_{\text{C1}\rightarrow\text{C2}} = 1 \cdot b^s_{\text{C1}\rightarrow\text{C2}} + (S-1) \cdot b^s_{\text{C2}\rightarrow\text{C1}} + (S-1)b^d \tag{5}$$

$$B^*_{\text{C2}\rightarrow\text{C1}} = (S-1) \cdot b^s_{\text{C1}\rightarrow\text{C2}} + 1 \cdot b^s_{\text{C2}\rightarrow\text{C1}} + (S-1)b^d \tag{6}$$

$$B^*_{\text{C1}\rightarrow\text{C3}} = 1 \cdot b^s_{\text{C1}\rightarrow\text{C3}} + (S-1) \cdot b^s_{\text{C3}\rightarrow\text{C1}} + (S-1)b^e \tag{7}$$

$$B^*_{\text{C3}\rightarrow\text{C1}} = (S-1) \cdot b^s_{\text{C1}\rightarrow\text{C3}} + 1 \cdot b^s_{\text{C3}\rightarrow\text{C1}} + (S-1)b^e \tag{8}$$

Figura 21 – Medição de tráfego no *cluster* ONOS/Atomix com *switch* e enlaces com topologia em estrela



Fonte: Autoria Própria.

5. Então para uma rede arbitrária gerenciada por um *cluster* de 3 controladores o tráfego trocado entre os controladores x e y é dado pela Equação 9:

$$
\begin{aligned}
B_{x\to y} = b^0 + (b^s_{\mathrm{C1}\to\mathrm{C2}} = b^s_{\mathrm{C1}\to\mathrm{C3}}) \cdot S_x + (b^l_{\mathrm{C1}\to\mathrm{C2}} = b^l_{\mathrm{C1}\to\mathrm{C3}}) \cdot L_x \\
+(b^s_{\mathrm{C2}\to\mathrm{C1}} = b^s_{\mathrm{C2}\to\mathrm{C3}} = b^s_{\mathrm{C3}\to\mathrm{C1}} = b^s_{\mathrm{C3}\to\mathrm{C2}}) \cdot (S_y + S_z) \\
+(b^l_{\mathrm{C2}\to\mathrm{C1}} = b^l_{\mathrm{C2}\to\mathrm{C3}} = b^l_{\mathrm{C3}\to\mathrm{C1}} = b^l_{\mathrm{C3}\to\mathrm{C2}}) \cdot (L_y + L_z) \\
+b^d \cdot (L_{xy} + L_{xz}) + b^e \cdot L_{yz}[\mathrm{kbps}]
\end{aligned}
\tag{9}
$$

O modelo descrito nessa seção foi desenvolvido por Muqaddas *et al.* (2017) para o *cluster* ONOS. O objetivo do trabalho atual é aplicar este modelo no *cluster* ONOS/Atomix com as devidas modificações, caso necessárias. A seção 3.3.1 é composta pelos experimentos com o *cluster* ONOS com objetivo de aplicar o modelo descrito na seção 3.3 em uma versão mais nova do ONOS que a utilizada pelo autor. A seção 3.3.2 reproduz os mesmos experimentos no *cluster* ONOS/Atomix com o objetivo de estender o modelo para a arquitetura de *cluster* ONOS/Atomix.

3.3.1 Modelo de tráfego entre controladores no *cluster* ONOS

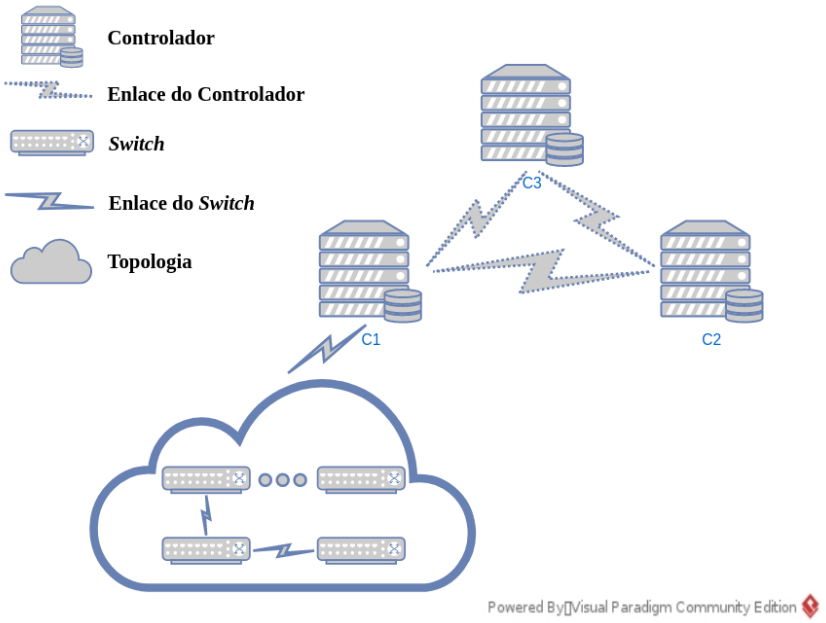
A Figura 22 mostra a topologia utilizada para as medições de tráfego na arquitetura ONOS. O *cluster* ONOS tem 3 controladores e foram feitas 6 medições com o número de *switches* descritos na Tabela 1, sendo todos linearmente conectados no controlador $C1$.

Tabela 1 – Número de *switches* em cada controlador no modelo de tráfego

Topologia	**Controlador 1** ($C1$)	**Controlador 2** ($C2$)	**Controlador 3** ($C3$)
1	0	0	0
2	20	0	0
3	40	0	0
4	60	0	0
5	80	0	0
6	100	0	0

Fonte: Autoria própria.

Figura 22 – Medição de tráfego no *cluster* ONOS



Fonte: Autoria Própria.

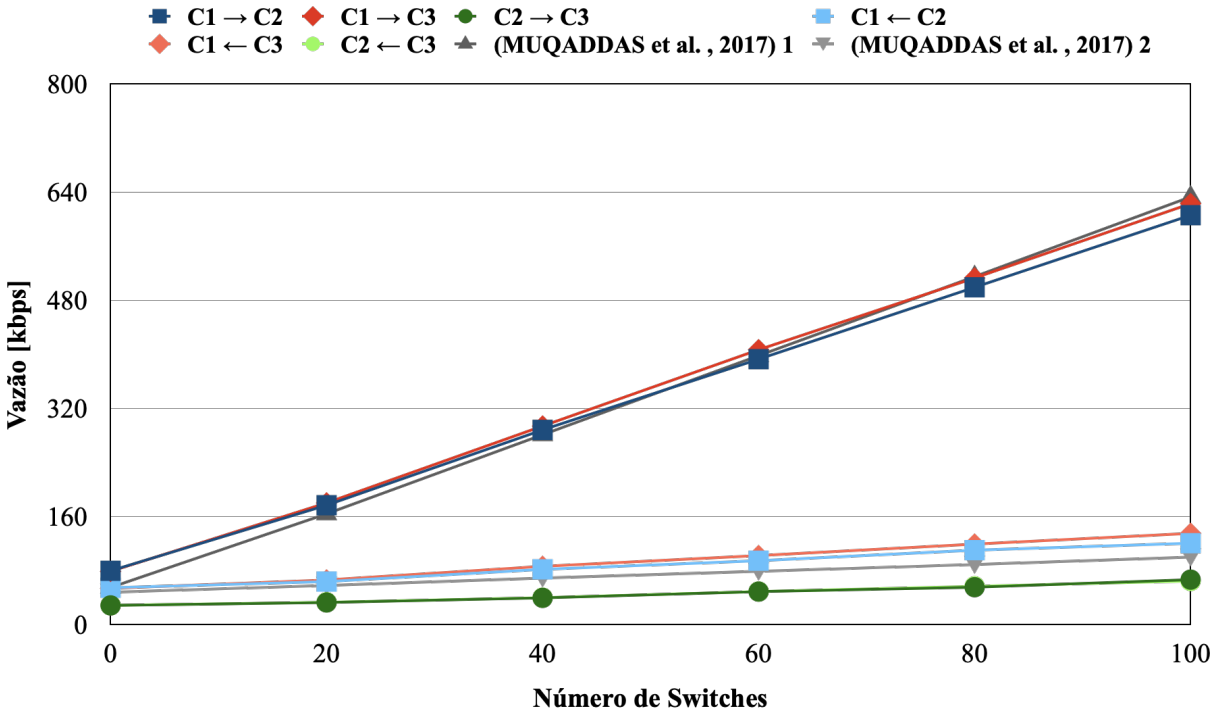
O Gráfico 2 apresenta o resultado do tráfego entre os 3 controladores na arquitetura ONOS nos 2 sentidos. Como todos os *switches* estão conectados no controlador $C1$, o tráfego deste para os controladores $C2$ e $C3$ cresce proporcionalmente com o número de *switches* ($C1 \rightarrow C2$ e $C1 \rightarrow C3$), sendo que o tráfego entre os controladores $C2$ e $C3$ é menor que o tráfego do controlador $C1$ ($C2 \rightarrow C3$ e $C3 \rightarrow C2$), assim como tráfego no sentido do líder também é menor ($C2 \rightarrow C1$ e $C3 \rightarrow C1$). As curvas 1 e 2 foram obtidas com a Equação 10 do modelo de tráfego do trabalho de Muqaddas *et al.* (2017).

A Tabela 2 mostra os valores dos coeficientes das equações obtidos por Muqaddas *et al.* (2017). O experimento do autor utilizou a versão 1.4.1 do ONOS, enquanto a versão deste trabalho é 1.13.10 para ONOS. Pode ser observado que a mudança de versão do controlador ONOS não influenciou no resultado final da equação do tráfego e as curvas do autor no Gráfico 1 são praticamente os mesmos medidos neste trabalho, mostrados no Gráfico 2.

Ainda de acordo com Muqaddas *et al.* (2017), esse comportamento no ONOS é consequência da topologia ser periodicamente atualizada por meio de pacotes *Link Layer Discovery Protocol* (LLDP) recebidos na interface *southbound* mesmo quando a topologia não sofre alterações. Essa atualização é responsável pelo tráfego adicional no controlador $C1$ por causa das estruturas de dados internas do ONOS onde a utilização de memória cresce linearmente de acordo com o número de nós e enlaces. A atualização do controlador líder para os seguidores é através do protocolo *Anti-Entropy*, que escolhe aleatoriamente outro controlador para fazer a

atualização das informações de topologia periodicamente. Ou seja, o controlador ONOS gera tráfego sempre a partir do controlador onde estão conectados os *switches*.

Gráfico 2 – Média do Tráfego dos Enlaces entre os Controladores no *cluster* ONOS



Fonte: Autoria Própria.

Tabela 2 – Resultados obtidos por Muqaddas *et al.* (2017) para o *cluster* ONOS

Coeficiente	Vazão (kbps)
b^0	47.81
$b^l_{\text{C1}\to\text{C2}} = b^l_{\text{C1}\to\text{C3}}$	1.43
$b^l_{\text{C2}\to\text{C1}} = b^l_{\text{C2}\to\text{C3}} = b^l_{\text{C3}\to\text{C1}} = b^l_{\text{C3}\to\text{C2}}$	0.06
$b^s_{\text{C1}\to\text{C2}} = b^s_{\text{C1}\to\text{C3}}$	4.43
$b^s_{\text{C2}\to\text{C1}} = b^s_{\text{C2}\to\text{C3}} = b^s_{\text{C3}\to\text{C1}} = b^s_{\text{C3}\to\text{C2}}$	0.46
b^d	0.77
b^e	0.15

Fonte: Adaptado de (MUQADDAS *et al.*, 2017).

Ao substituir os coeficientes da Equação 9 pelos valores obtidos nos experimentos de Muqaddas *et al.* (2017), o tráfego trocado entre os controladores x e y em uma rede arbitrária gerenciada por um *cluster* ONOS de 3 controladores é dado pela Equação 10:

$$\begin{aligned} B_{x\to y} =& 47.81 + 4.43 \cdot S_x + 1.43 \cdot L_x + 0.46 \cdot (S_y + S_z) + \\ & 0.06 \cdot (L_y + L_z) + 0.77 \cdot (L_{xy} + L_{xz}) + 0.15 \cdot L_{yz} [\text{kbps}] \end{aligned} \tag{10}$$

Nesta seção foram descritas as medições feitas neste trabalho com 0, 20, 40, 60, 80 e 100 *switches* conectados no controlador $C1$ no *cluster* ONOS. Os valores da Tabela 2 e da

Equação 10 foram obtidos pelo trabalho do mesmo autor e utilizados nos valores teóricos 1 e 2 para o controlador ONOS. Como foi observado a diferença de versão não teve influencia no tráfego e os valores obtidos pelo mesmo autor com a versão 1.4.1 podem ser utilizados pela versão 1.13.10 do ONOS.

3.3.2 Modelo de tráfego do *cluster* ONOS/Atomix

Ao comparar a arquitetura do *cluster* ONOS que pode ser observada na Figura 10 da seção 2.2 com a arquitetura do *cluster* ONOS/Atomix que pode ser observada na Figura 11 da seção 2.3 é possível perceber que o *cluster* ONOS/Atomix separa o armazenamento do controle. Desse modo, além da comunicação entre os controladores, também existe a troca de mensagens entre controladores e Atomix, e entre os nós do Atomix. A Figura 16 da seção 3.2 mostra essa divisão no ambiente de simulação. Então para todos os cenários do *cluster* ONOS/Atomix existe um conjunto composto por três nós de agentes Atomix. Com o objetivo de estender o modelo de tráfego elaborado por Muqaddas *et al.* (2017).

O *cluster* ONOS/Atomix utilizou o mesmo número de *switches* observados na Tabela 1 da seção 3.3.1 do *cluster* ONOS. A análise da arquitetura *cluster* ONOS/Atomix foi dividido em três partes. A seção 3.3.2.1 apresenta o modelo de tráfego entre os controladores no *cluster* ONOS/Atomix. A seção 3.3.2.2 aborda o modelo de tráfego entre controladores e nós Atomix no *cluster* ONOS/Atomix e por fim a seção 3.3.2.3 descreve o modelo de tráfego entre os nós Atomix no *cluster* ONOS/Atomix.

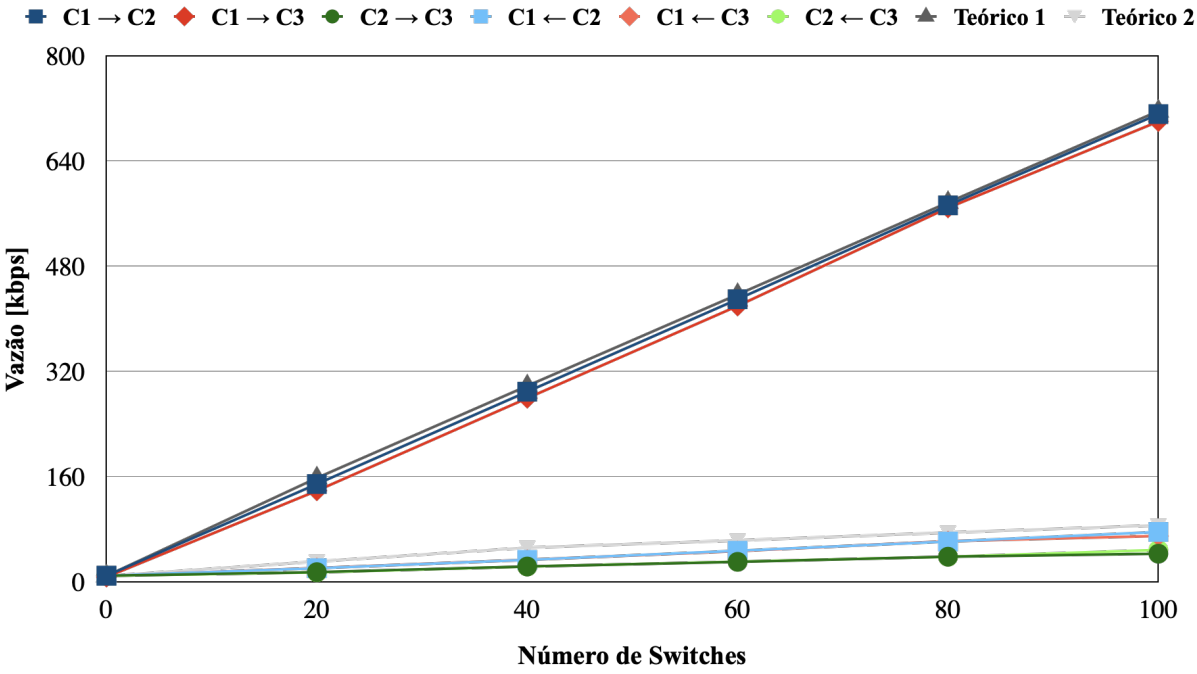
3.3.2.1 Modelo de tráfego entre controladores no *cluster* ONOS/Atomix

A Figura 20 da seção 3.3, mostrou uma topologia composta por nós do Atomix, controladores, *switches* e enlaces em um *cluster* ONOS/Atomix. Assim servindo como exemplo para a topologia linear usada neste experimento. Essa seção é referente ao tráfego dos enlaces entre os controladores. Foram feitas 6 medições com o seguinte número de *switches*: 0, 20, 40, 60, 80 e 100, sendo todos conectados no controlador $C1$.

Foi usada a metodologia descrita na seção 3.3 e os valores dos testes com 20 *switches* para obter os coeficientes da equação de tráfego, que podem ser vistos na Tabela 3. A mesma equação foi utilizada com os valores de 40, 60, 80 e 100 *switches* e os resultados podem ser observados nas curvas Teórico 1 e Teórico 2 do Gráfico 3. Assim a Equação 11 pode ser utilizada

para determinar o tráfego trocado entre os controladores do *cluster* ONOS/Atomix para qualquer número de *switches*.

Gráfico 3 – Média do Tráfego dos Enlaces entre os Controladores no *cluster* ONOS/Atomix



Fonte: Autoria Própria.

Tabela 3 – Resultados obtidos para o tráfego entre controladores no *cluster* ONOS/Atomix

Coeficiente	Vazão (kbps)
b^0	9.4
$b^l_{\text{C1}\to\text{C2}} = b^l_{\text{C1}\to\text{C3}}$	0.35
$b^l_{\text{C2}\to\text{C1}} = b^l_{\text{C2}\to\text{C3}} = b^l_{\text{C3}\to\text{C1}} = b^l_{\text{C3}\to\text{C2}}$	0.22
$b^s_{\text{C1}\to\text{C2}} = b^s_{\text{C1}\to\text{C3}}$	6.62
$b^s_{\text{C2}\to\text{C1}} = b^s_{\text{C2}\to\text{C3}} = b^s_{\text{C3}\to\text{C1}} = b^s_{\text{C3}\to\text{C2}}$	0.35
b^d	0.97
b^e	6.56

Fonte: Autoria Própria.

Ao substituir os coeficientes na Equação 9, o tráfego trocado entre os controladores x e y em uma rede arbitrária gerenciada por um *cluster* de 3 controladores é dado pela Equação 11:

$$\begin{aligned} B_{x\to y} =& 9.4 + 6.62 \cdot S_x + 0.35 \cdot L_x + 0.35 \cdot (S_y + S_z) + \\ & 0.22 \cdot (L_y + L_z) + 0.97 \cdot (L_{xy} + L_{xz}) + 6.56 \cdot L_{yz} [\text{kbps}] \end{aligned} \tag{11}$$

O Gráfico 3 mostra o resultado do tráfego entre os 3 controladores na arquitetura ONOS/Atomix nos 2 sentidos. Pode ser observado que o comportamento do tráfego é similar a arquitetura ONOS. Ou seja, o tráfego é maior no sentido do controlador onde os *switches* estão conectados, que é o $C1$, para os outros controladores. Ainda, pode ser observado que o tráfego

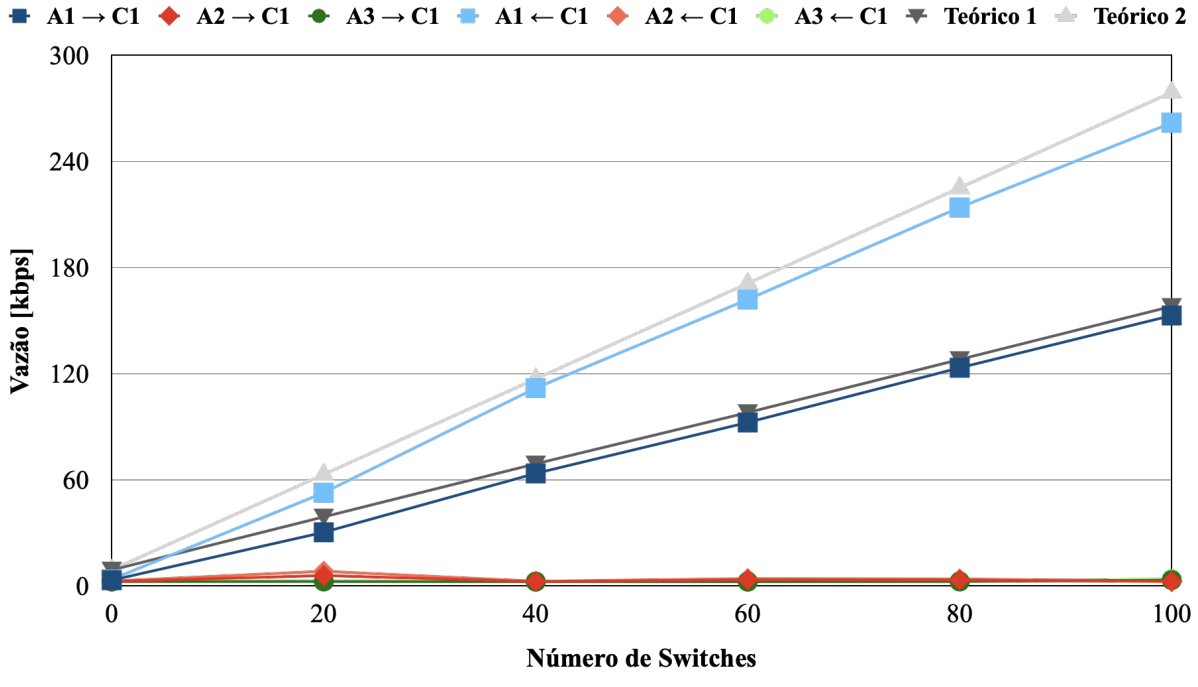
cresce proporcionalmente com o número de *switches* na rede. O comportamento deste tráfego é devido as atualizações periódicas de pacotes LLDP recebidos na interface *southbound*.

3.3.2.2 Modelo de tráfego entre controladores e nós Atomix no *cluster* ONOS/Atomix

Foi observado que o comportamento do tráfego entre os controladores e os nós do Atomix segue um perfil parecido com o observado entre controladores. Como pode ser observado no Gráfico 4, também existe um aumento no tráfego de acordo com o número de *switches*, mas o tráfego gerado do controlador até o nó do Atomix é maior que do Atomix para com os controladores. No trabalhos de Muqaddas *et al.* (2017), sempre é considerando um tráfego maior por parte do líder até seus seguidores do *cluster* ONOS, como pode ser observado no Gráfico 2, mas o líder do *cluster* ONOX/Atomix é o $A1$. Assim, a equação desenvolvida por Muqaddas *et al.* (2017) foi adaptada para considerar o maior tráfego do controlador com mais *switches* até o nó do Atomix líder.

Para obter os valores teóricos do pior caso, foram utilizadas duas equações adaptadas a partir do trabalho de Muqaddas *et al.* (2017). Para obter o valor teórico entre o Atomix e o controlador representado por $A1 \rightarrow C1$, foram utilizados os valores da Tabela 4 na Equação 12. Ao substituir os valores, foi obtida a Equação 13, responsável pelo valor Teórico 1 do Gráfico 4.

Gráfico 4 – Média do Tráfego dos Enlaces entre um nó Atomix e o Controlador 1 no *cluster* ONOS/Atomix



Fonte: Autoria Própria.

Tabela 4 – Resultados obtidos para o tráfego entre os nós do Atomix e controladores para o *cluster* ONOS/ Atomix Teórico 1

Coeficiente	Vazão (kbps)
b^0	9.4
$b^{l^a}_{\text{A1}\rightarrow\text{C1}}$	0.07
$b^{l^b}_{\text{A1}\rightarrow\text{C1}}$	0.047
$b^{s^a}_{\text{A1}\rightarrow\text{C1}}$	1.41
$b^{s^b}_{\text{A1}\rightarrow\text{C1}}$	0.074
b^d	0.21
b^e	1.4

Fonte: Autoria Própria.

$$\begin{aligned} B_{x\rightarrow y} =& b^0 + b^{s^a}_{\text{A1}\rightarrow\text{C1}} \cdot S_x + b^{s^b}_{\text{A1}\rightarrow\text{C1}} \cdot L_x + b^{l^a}_{\text{A1}\rightarrow\text{C1}} \cdot (S_y + S_z) + \\ & b^{l^b}_{\text{A1}\rightarrow\text{C1}} \cdot (L_y + L_z) + b^d \cdot (L_{xy} + L_{xz}) + b^e \cdot L_{yz}[\text{kbps}] \end{aligned} \tag{12}$$

$$\begin{aligned} B_{x\rightarrow y} =& 9.4 + 1.41 \cdot S_x + 0.074 \cdot L_x + 0.07 \cdot (S_y + S_z) + \\ & 0.047 \cdot (L_y + L_z) + 0.21 \cdot (L_{xy} + L_{xz}) + 1.4 \cdot L_{yz}[\text{kbps}] \end{aligned} \tag{13}$$

Para obter o valor teórico entre o controlador e o Atomix representado por $C1 \rightarrow A1$ foram utilizados os valores da Tabela 5 na Equação 14. Ao substituir os valores foi obtida a Equação 15, responsável pelo valor Teórico 2.

Tabela 5 – Resultados obtidos para o tráfego entre os nós do Atomix e controladores para o *cluster* ONOS/ Atomix Teórico 2

Coeficiente	Vazão (kbps)
b^0	9.4
$b^{l^a}_{\text{C1}\rightarrow\text{A1}}$	0.13
$b^{l^b}_{\text{C1}\rightarrow\text{A1}}$	0.08
$b^{s^a}_{\text{C1}\rightarrow\text{A1}}$	2.56
$b^{s^b}_{\text{C1}\rightarrow\text{A1}}$	0.14
b^d	0.37
b^e	2.53

Fonte: Autoria Própria.

$$\begin{aligned} B_{x\rightarrow y} =& b^0 + b^{s^a}_{\text{C1}\rightarrow\text{A1}} \cdot S_x + b^{s^b}_{\text{C1}\rightarrow\text{A1}} \cdot L_x + b^{l^a}_{\text{C1}\rightarrow\text{A1}} \cdot (S_y + S_z) + \\ & b^{l^b}_{\text{C1}\rightarrow\text{A1}} \cdot (L_y + L_z) + b^d \cdot (L_{xy} + L_{xz}) + b^e \cdot L_{yz}[\text{kbps}] \end{aligned} \tag{14}$$

$$\begin{aligned} B_{x\rightarrow y} =& 9.4 + 2.56 \cdot S_x + 0.14 \cdot L_x + 0.13 \cdot (S_y + S_z) + \\ & 0.08 \cdot (L_y + L_z) + 0.37 \cdot (L_{xy} + L_{xz}) + 2.53 \cdot L_{yz}[\text{kbps}] \end{aligned} \tag{15}$$

O Gráfico 4 mostra o tráfego dos enlaces entre o nó Atomix e o controlador $C1$, que é o pior caso levando em conta que todos os *switches* estão no $C1$. O resultado do tráfego a partir dos controladores $C2$ e $C3$ foram omitidos por serem menores do que o gerado a partir

de $C1$. Os resultados mostram um crescimento linear por parte dos enlaces do Atomix similar ao observado entre controladores ONOS com o aumento da topologia. Ou seja, as atualizações gerados pelos pacotes LLDP nos controladores ONOS são transmitidos para os nós Atomix a partir do controlador $C1$, sendo que o maior tráfego é enviado para o líder $A1$ dos nós Atomix. Ainda, o perfil de tráfego entre os controladores ONOS é o mesmo que o perfil de tráfego entre os controladores e os nós Atomix.

3.3.2.3 Modelo de tráfego entre os nós Atomix no *cluster* ONOS/Atomix

O Gráfico 5 mostra a média do tráfego dos enlaces entre os nós do Atomix. Como pode ser observado, todos os enlaces que fazem uma conexão com o nó $A1$ geram mais tráfego que os demais pelo fato deste ser o líder, o que é uma característica do protocolo Raft. Também pode ser observado que o tráfego que sai do líder ($A1 \rightarrow A2$ e $A1 \rightarrow A3$) é maior do que o tráfego que entra no líder ($A1 \leftarrow A2$ e $A1 \leftarrow A3$), e que o tráfego entre os seguidores é relativamente baixa ($A2 \rightarrow A3$ e $A2 \leftarrow A3$).

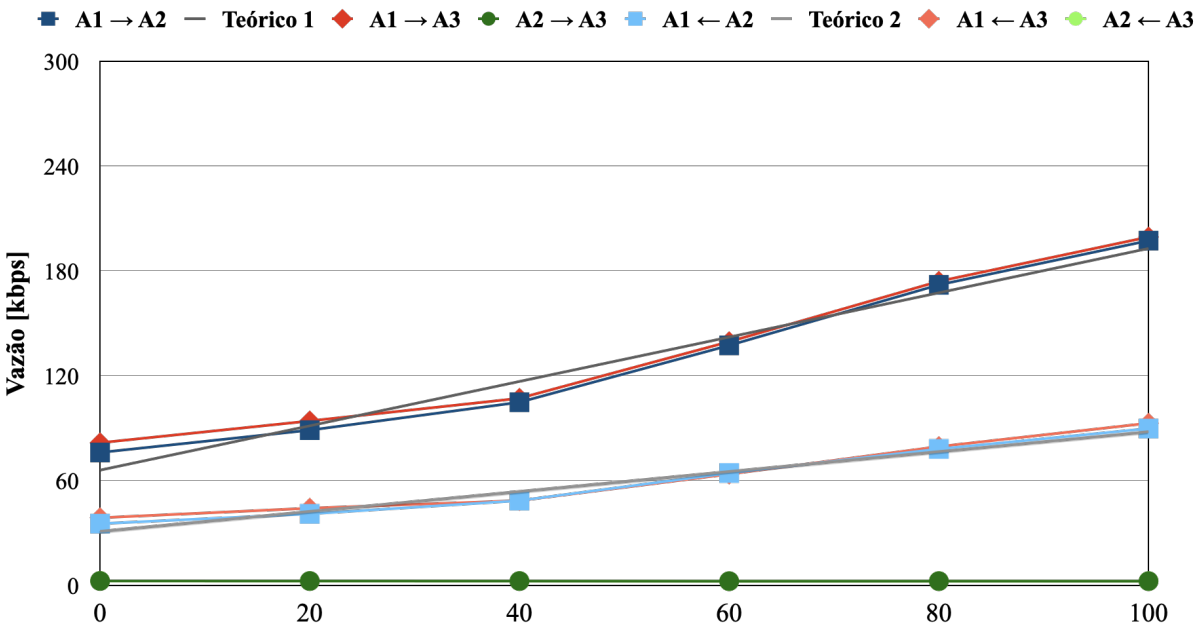
Como pode ser observado no Quadro 3 da seção 3.3, os coeficientes de b^d e b^e são equivalentes a um particionamento arbitrário. O Atomix faz um acesso indireto do mapa da topologia, dessa forma considera apenas o número de *switches* e não a sua distribuição, como será explicado no próximo capitulo que considera a distribuição de *switches* no *cluster*. Desse modo, as equações de Muqaddas *et al.* (2017) não podem ser utilizadas para o tráfego entre os nós do Atomix.

Através da análise dos resultados obtidos nos experimentos, foi observada a possibilidade da utilização de equações de regressão linear para determinar os valores teóricos. As retas Teórico 1 e Teórico 2 do Gráfico 5 foram utilizadas para obter as Equações 16 e 17. Nestas equações é possível observar que o valor encontrado torna a igualdade verdadeira, assim estimando o valor esperado que determina a relação entre ambas as variáveis, e que satisfaz a equação formando uma reta no plano cartesiano.

$$\text{A1} \rightarrow A2y = 1{,}269x + 65{,}849 \tag{16}$$

$$\text{A1} \leftarrow A2y = 0{,}5706x + 30{,}853 \tag{17}$$

Gráfico 5 – Média do Tráfego dos Enlaces entre os nós Atomix no *cluster* ONOS/Atomix

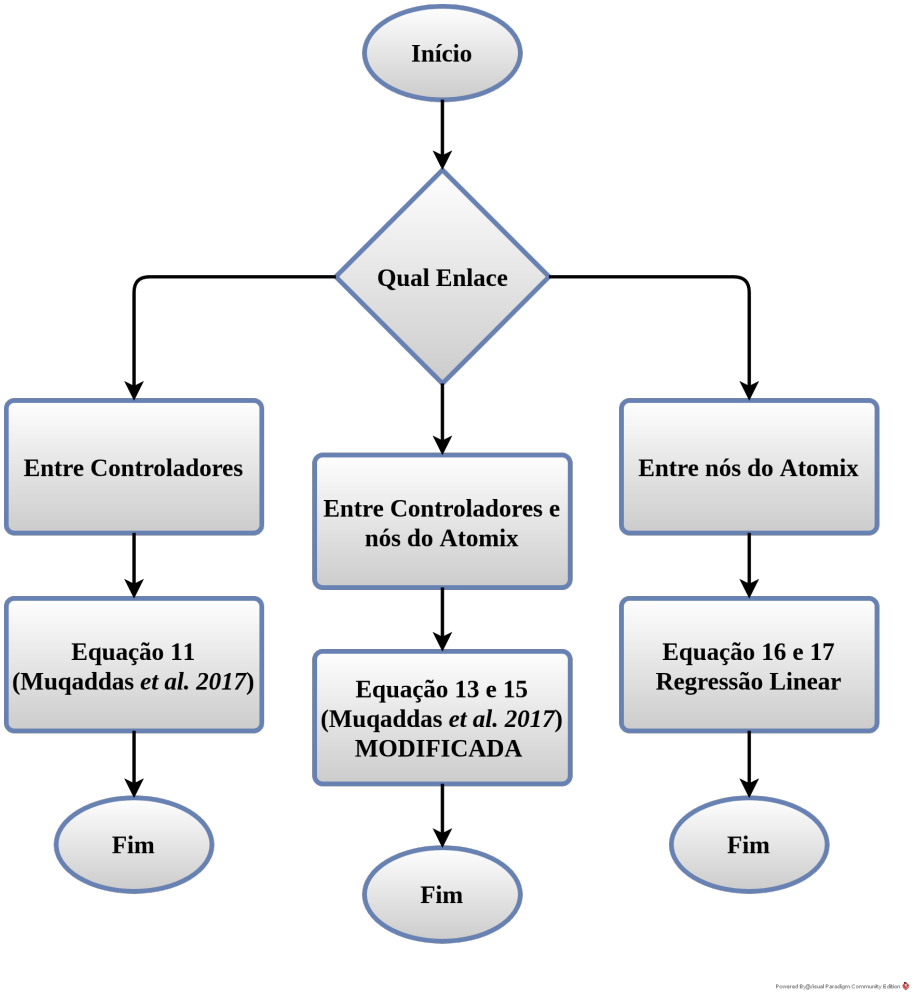


Fonte: Autoria Própria.

3.4 CONCLUSÃO DO CAPÍTULO

Como pode ser observado na seção 3.3.1, os valores teóricos obtidos por Muqaddas *et al.* (2017) para o controlador ONOS foram de encontro com os valores medidos durante os experimentos, tornando os resultados do *cluster* ONOS consistentes. A seção 3.3.2 com modelo do ONOS/Atomix foi dividido em três partes, a Figura 23 mostra essa divisão onde a seção 3.3.2.1 foi referente ao enlace entre os controladores do *cluster*, desse modo foi possível utilizar as equações desenvolvidas por Muqaddas *et al.* (2017) mas com os valores obtidos nos experimentos. Para seção 3.3.2.2 que foi referente ao enlace dos nós do Atomix entre os controladores a equação de Muqaddas *et al.* (2017) foi adaptada e utilizados os valores testados nos experimentos resultando em duas equações para determinar os valores teóricos. A seção 3.3.2.3 foi referente ao enlace entre os nós do Atomix onde foi possível determinar os valores teóricos a partir de regressão linear utilizando os valores testados nos experimentos.

Figura 23 – Fluxograma para o modelo de tráfego do *cluster* ONOS/Atomix



Fonte: Autoria Própria.

Os resultados obtidos nos experimentos do *cluster* ONOS/Atomix descritos nas seções 3.3.2.1, 3.3.2.2 e 3.3.2.3 serão utilizados para comparar os resultados obtidos no próximo capítulo, onde são consideradas distribuições diferentes de *switches* em cada controlador.

4 RESULTADOS E ANÁLISE DOS TESTES

Este capítulo descreve os cenários de testes para medir o tráfego nas arquiteturas ONOS e ONOS/Atomix e a análise dos resultados. A seção 4.1.1 utiliza um cenário similar ao descrito anteriormente mas com uma divisão arbitrária do número de *switches* entre 3 controladores. A seção 4.1.2 faz a análise do aumento de controladores na rede. Os valores e equações obtidas no capítulo 3 serão utilizadas para comparar com a medições efetuadas nesse capítulo.

4.1 ANÁLISE DE TRÁFEGO ENTRE CONTROLADORES

Esta seção apresenta os 2 cenários realizados para a medição do tráfego. O cenário 1 distribui os *switches* entre os 3 controladores para retratar uma situação real e buscar uma distribuição de *switches* com menor tráfego. Para analisar o efeito do número de controladores no tráfego, foi desenvolvido o cenário 2, que considera diferentes números de controladores no *cluster*.

4.1.1 Cenário 1

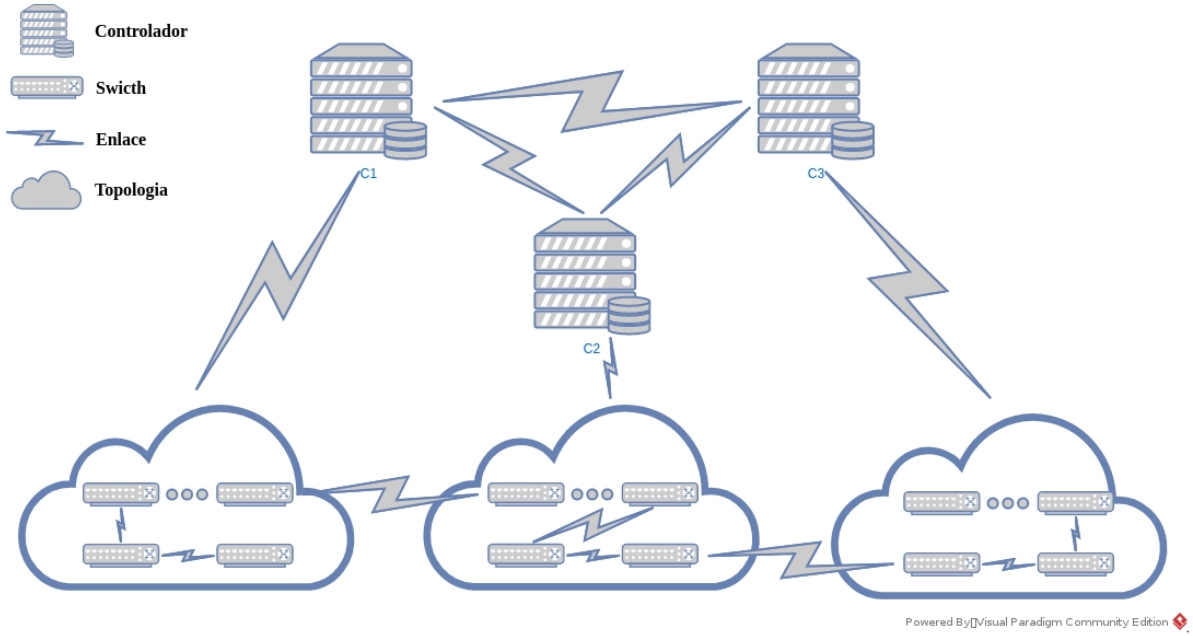
O cenário 1 foi executado em um *cluster* com três controladores onde as topologias foram divididas linearmente. O número de *switches* pode ser observado na Tabela 6. A Figura 24 mostra o *cluster* ONOS, e a Figura 25 mostra o *cluster* ONOS/Atomix. O objetivo de dividir o número de *switches* é simular uma rede mais próxima de uma topologia real e buscar uma distribuição de *switches* com o menor tráfego.

Tabela 6 – Número de *switches* em cada controlador no cenário 1

Topologia	**Controlador 1** ($C1$)	**Controlador 2** ($C2$)	**Controlador** ($C3$)
1	33	33	33
2	40	30	30
3	60	20	20
4	80	10	10

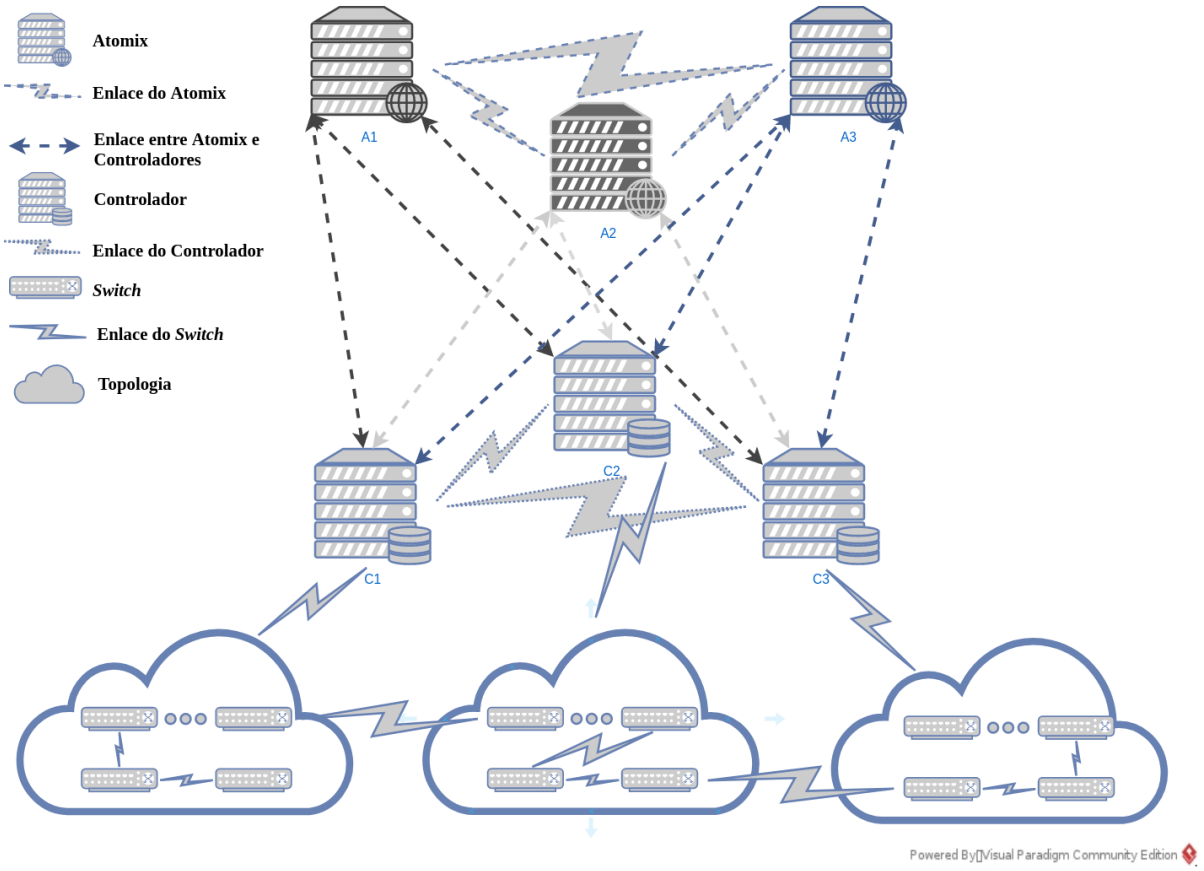
Fonte: Autoria Própria.

Figura 24 – Medição de tráfego do *cluster* ONOS do cenário 1



Fonte: Autoria Própria.

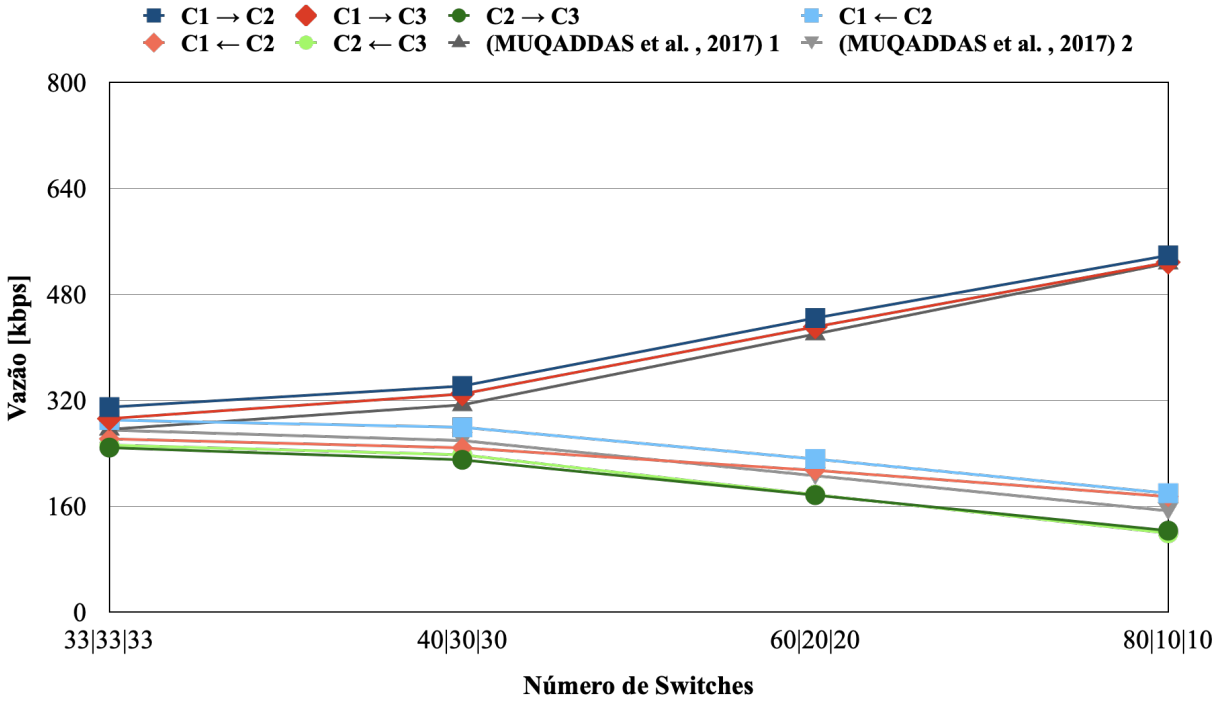
Figura 25 – Medição de tráfego do *cluster* ONOS/Atomix do cenário 1



Fonte: Autoria Própria.

O Gráfico 6 mostra o resultado do experimento no *cluster* ONOS. Como observado anteriormente no Gráfico 2 da seção 3.3.1, o tráfego aumentou proporcionalmente de acordo com o número de *switches*. É possível perceber que esse aumento de tráfego também ocorre em uma divisão desproporcional dos *switches* na rede. Como visto anteriormente no Gráfico 3 da seção 3.3.2.1, essa característica também está presente no *cluster* ONOS/Atomix. Um comportamento similar pode ser observado no Gráfico 7 onde os *switches* estão distribuídos no *cluster* ONOS/Atomix.

Gráfico 6 – Média do Tráfego dos Enlaces entre os Controladores ONOS

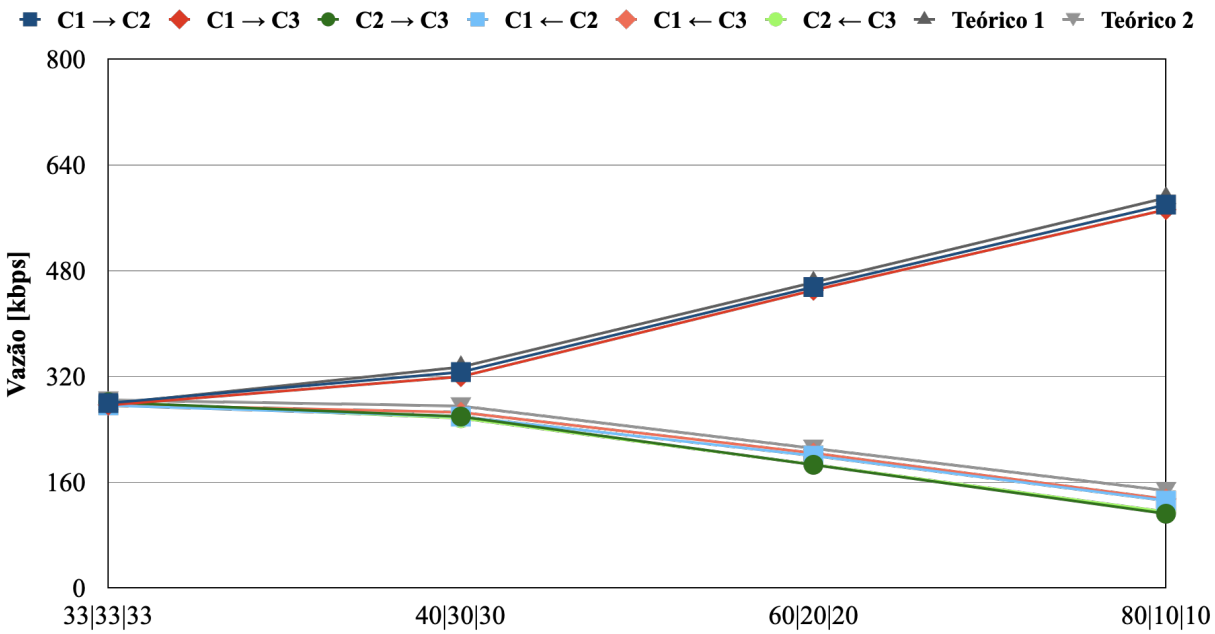


Fonte: Autoria Própria.

Conforme ocorre o aumento de *switches* no controlador $C1$, é possível perceber o aumento do tráfego nele e a redução nos controladores $C2$ e $C3$. Essa concentração de tráfego no controlador $C1$ pode ocasionar um congestionamento em seu enlace que pode ser evitado com uma estratégia de divisão mais igualitária da topologia.

De acordo com Muqaddas *et al.* (2017), as equações descritas na seção 3.3 com o modelo de tráfego também podem ser utilizadas em topologias com uma divisão arbitrária de *switches* entre os controladores. Então, os valores teóricos 1 e 2 no Gráfico 6 foram obtidos através da Equação 10 da seção 3.3.1, que é composta pelos resultados obtidos pelo mesmo autor descritos na Tabela 3 da seção 3.3.2.1.

Gráfico 7 – Média do Tráfego dos Enlaces entre os Controladores ONOS/Atomix



Fonte: Autoria Própria.

Com o objetivo de incluir os demais controladores, foram adicionados quatro valores teóricos como pode ser observado no Gráfico 8. O número de *switches* nos controladores $C2$ e $C3$ é o mesmo, desse modo os valores Teórico 1 e Teórico 3 do Gráfico 8 trazem os valores para o controlador $C1$ e os valores Teórico 2 e Teórico 4 da mesma figura compõem os valores teóricos para os controladores $C2$ e $C3$. Para determinar os valores Teórico 1 e Teórico 3 foi utilizada a Equação 13 da seção 3.3.2.2, e para os valores Teórico 2 e Teórico 4 foi utilizada a Equação 15 da mesma seção.

Os valores Teórico 1 e Teórico 2 que podem ser observados no Gráfico 7 foram obtidos através da Equação 11 da seção 3.3.2.1, com os valores medidos neste trabalho que estão descritos na Tabela 3 da seção 3.3.2.1. Como pode ser observado os valores teóricos vieram de encontro com o valores medidos neste trabalho. Então a equação e os valores obtidos através das medições de tráfego desse trabalho podem ser utilizadas para prever a vazão para qualquer distribuição de *switches* entre controladores no *cluster* ONOS/Atomix.

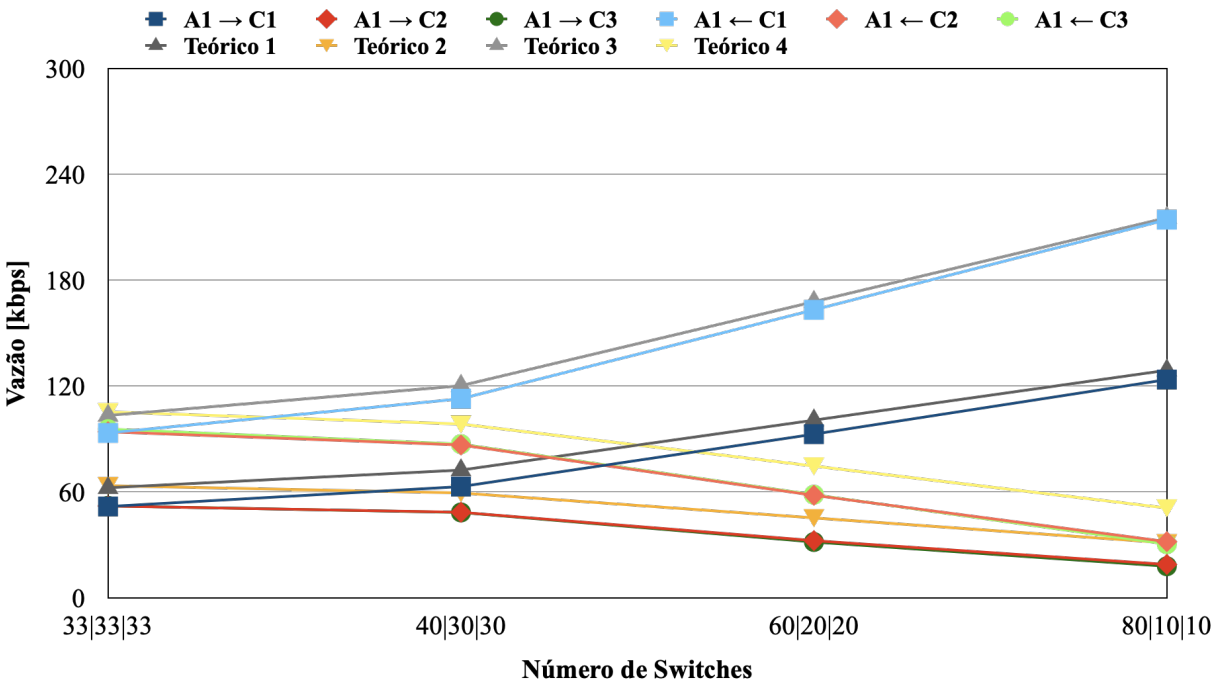
O Gráfico 8 mostra o tráfego entre os controladores com o nó do Atomix líder. Como foi observado no Gráfico 4 da seção 3.3.2.2, existe o aumento de trafego no enlace do controlador e o nó do Atomix com o aumento de *switches*. Esse comportamento também é observado em uma divisão arbitrária da topologia. Essa diferença pode ser observada no Gráfico 8, que compõe o enlace do nó do Atomix $A1$, que é líder, e o controlador $C1$ que representa o pior caso em

virtude do aumento de *switches* no controlador $C1$ e redução no $C2$ e $C3$.

Diferentemente do experimento apresentado no Gráfico 4 da seção 3.3.2.2, que compõe o *cluster* ONOS/Atomix com todos os *switches* em apenas um controlador, quando adicionados *switches* nos demais controladores consequentemente existe um maior tráfego em seus enlaces. Ao observar a Figura 4 é possível perceber que o tráfego dos controlares que não possuem *switches* é próximo a zero. Ao observar o Gráfico 8 é possível perceber que com a divisão arbitrária os demais controladores também possuem um maior tráfego em seus enlaces, que varia de acordo com o número de *switches*.

Com o objetivo de incluir os demais controladores, foram adicionados quatro valores teóricos como pode ser observado no Gráfico 8. O número de *switches* nos controladores $C2$ e $C3$ é o mesmo, desse modo os valores Teórico 1 e Teórico 3 do Gráfico 8 trazem os valores para o controlador $C1$ e os valores Teórico 2 e Teórico 4 da mesma figura compõem os valores teóricos para os controladores $C2$ e $C3$. Para determinar os valores Teórico 1 e Teórico 3 foi utilizada a Equação 13 da seção 3.3.2.2, e para os valores Teórico 2 e Teórico 4 foi utilizada a Equação 15 da mesma seção.

Gráfico 8 – Média do Tráfego dos Enlaces entre o Atomix e o Controlador 1



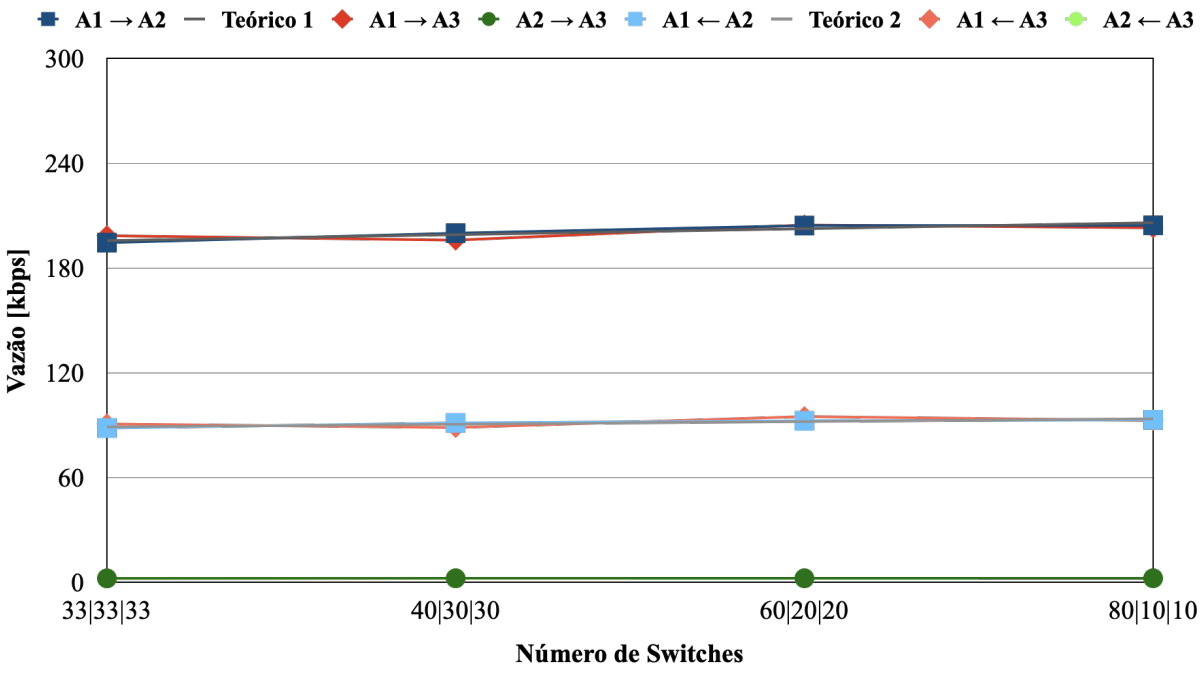
Fonte: Autoria Própria.

É possível perceber pela análise dos resultados dos experimentos que, sempre que ocorre o aumento de *switches* também ocorre o aumento do tráfego, e, como observado no

Gráfico 8, a divisão dos *switches* também determina esse tráfego. Além disso, o controlador $C1$, por possuir o maior número de *switches*, também é o controlador com maior tráfego. Os controladores $C2$ e $C3$, que possuem um mesmo número de *switches*, obtiveram um resultado similar, deixando evidente que o número de *switches* e sua posição é responsável pelo tráfego na rede. Novamente os experimentos mostraram a importância de uma divisão equilibrada para evitar sobrecarga no enlace de um controlador.

No Gráfico 5 é possível perceber o aumento do tráfego de acordo com o aumento no número de *switches* na rede. Mas como pode ser observado no Gráfico 9, quando não há o aumento de *switches* na rede, o tráfego entre os nós do Atomix se mantém o mesmo.

Gráfico 9 – Média do Tráfego dos Enlaces entre os nós Atomix



Fonte: Autoria Própria.

Diferentemente do experimento apresentado no Gráfico 5 onde ocorreu o aumento da vazão com o aumento do número de *switches*, a distribuição de *switches* não teve efeito na vazão dos enlaces entre os nós do Atomix, embora o valor observado em 100 *switches* no Gráfico 5 tenha sido o mesmo obtido neste experimento. Isso tornou possível a utilização das equações lineares 16 e 17 utilizada na seção 3.3.2.3. Os valores teóricos que podem ser observados no Gráfico 9 foram obtidos com os valores medidos neste trabalho presentes na seção 3.3.2.3. Para obter o Teórico 1, foi utilizada a Equação 16. Para obter o valor Teórico 2, foi utilizada a Equação 17. Ambos os valores teóricos mostram o pior caso como sendo o de $A1$, que é o enlace do nó do Atomix líder.

4.1.2 Cenário 2

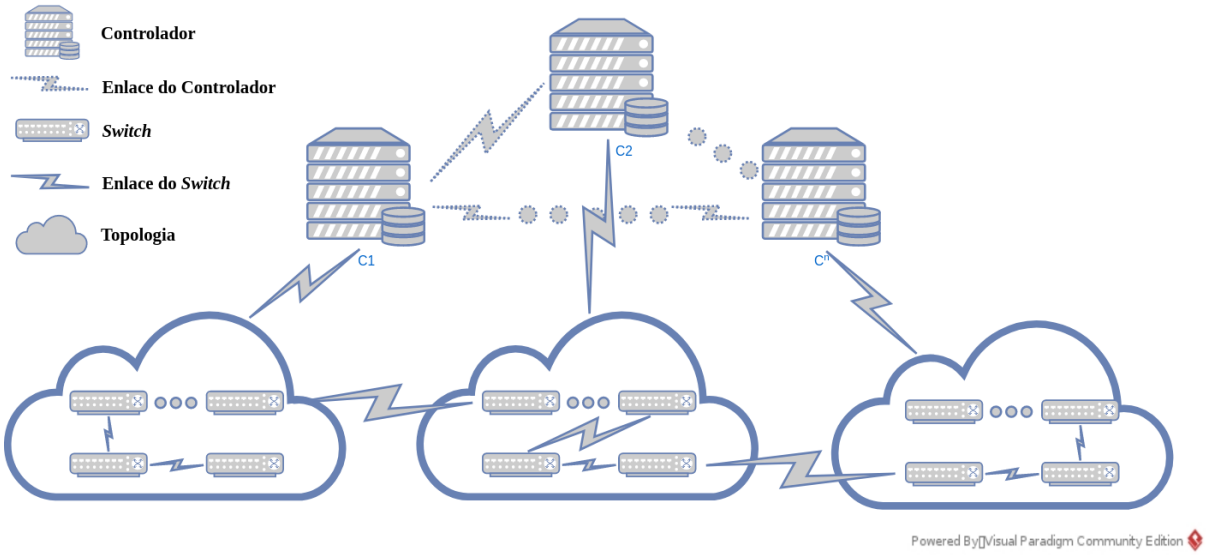
Como descrito nas seções anteriores, os experimentos foram efetuados através do aumento no número de *switches* e posteriormente a sua distribuição no *cluster* ONOS e *cluster* ONOS/Atomix. A seguir, para esse cenário, foi considerado o aumento no número de controladores com um número fixo de *switches* distribuídos igualmente e de forma linear na topologia. Essa distribuição de *switches* pode ser observada na Tabela 7. A Figura 26 mostra o *cluster* ONOS, e a Figura 27 mostra o *cluster* ONOS/Atomix.

Tabela 7 – Número de controladores e *switches* do cenário 2

Controladores	*Switches* por controlador	Total de *Switches*
3	33	99
5	20	100
7	14	98
9	11	99

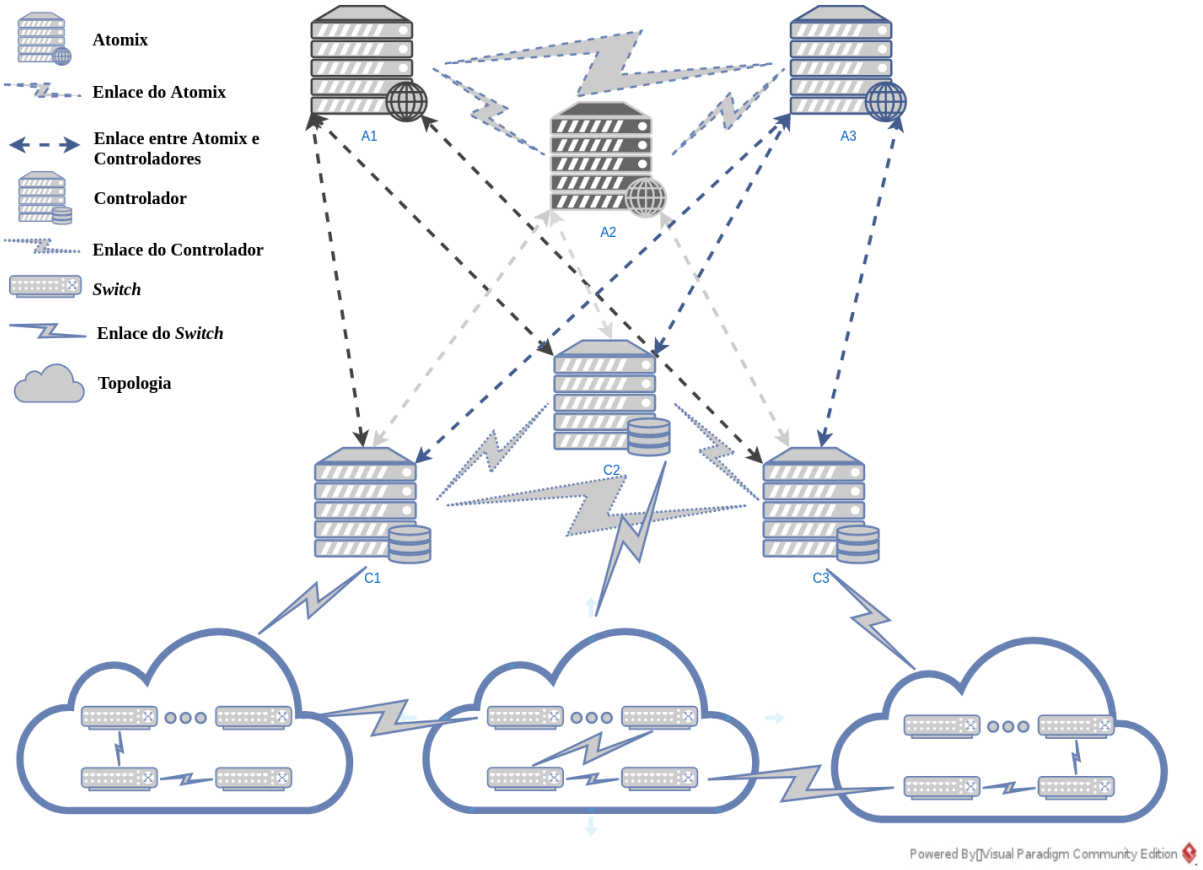
Fonte: Autoria Própria.

Figura 26 – Medição de tráfego do *cluster* ONOS do cenário 2



Fonte: Autoria Própria.

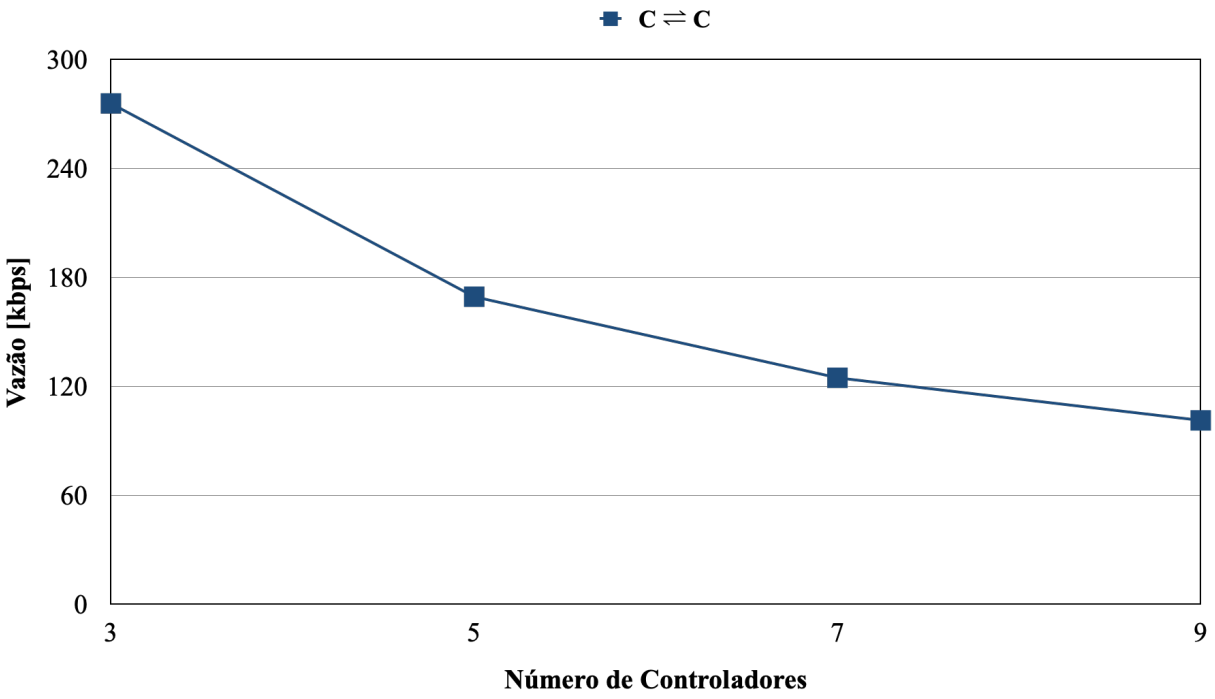
Figura 27 – Medição de tráfego do *cluster cluster* ONOS/Atomix do cenário 2



Fonte: Autoria Própria.

O Gráfico 10 mostra os resultados obtidos através dos experimentos no *cluster* ONOS. É possível perceber que com o aumento no número de controladores ocorre uma redução no tráfego gerado entre os controladores do *cluster*. De acordo com Muqaddas *et al.* (2017), isso ocorre devido ao protocolo *Anti-Entropy* que periodicamente seleciona aleatoriamente outro controlador para sincronizar a topologia de rede. Desse modo, o autor descreve a taxa de sincronização para cada controlador como λ, concluindo que a média de contribuição desse processo em cada enlace para o cenário com 3 controladores é de $3\lambda/6 = \lambda/2$, visto que são 6 enlaces possíveis. Ou seja, o tráfego em cada enlace diminui com o aumento do número de controladores.

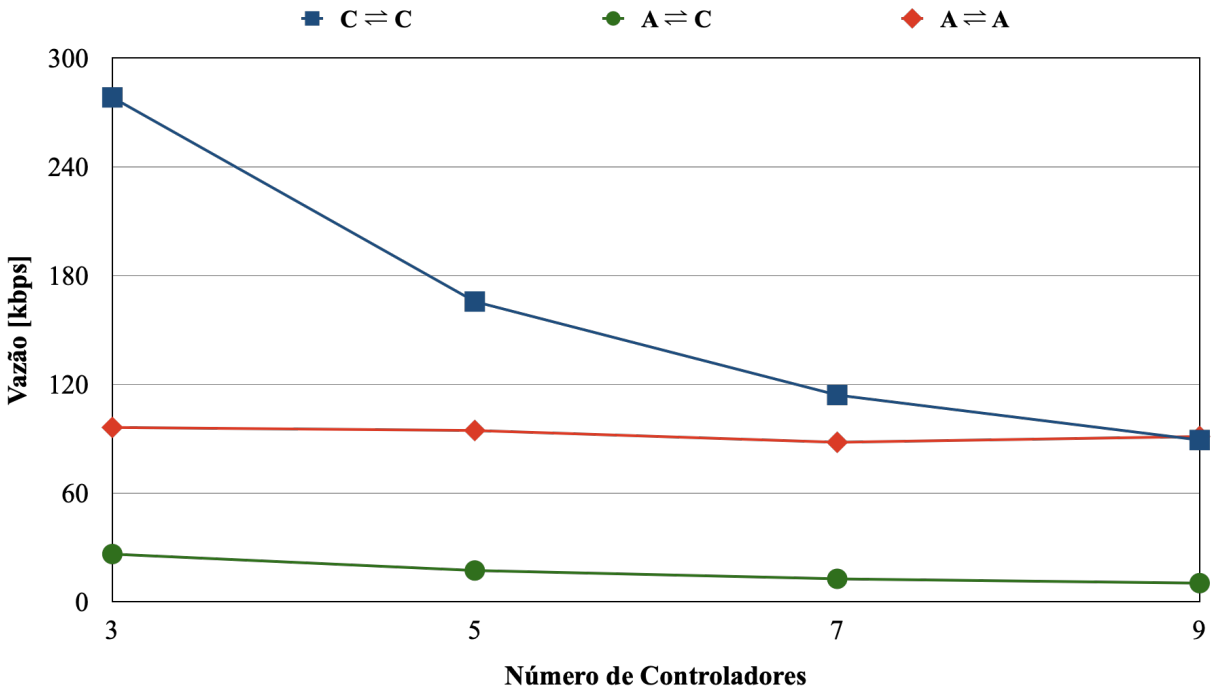
Gráfico 10 – Média do Tráfego dos Enlaces do ONOS



Fonte: Autoria Própria.

O Gráfico 11 mostra os resultados obtidos através dos experimentos no *cluster* ONOS/ Atomix. Como pode ser observado o tráfego entre os controladores teve o mesmo comportamento observado no Gráfico 10 do *cluster* ONOS. Essa similaridade no tráfego entre os controladores também foi observada nos demais experimentos das seções anteriores. O enlace entre os nós do Atomix e os controladores não sofreu alteração no tráfego com o aumento de controladores no *cluster*. Isso ocorre pelo fato de não haver um aumento significativo no número de *switches*, e por sua distribuição igualitária dos *switches* entre os controladores. Foi observado no Gráfico 8 que é o número de *switches* e sua distribuição que são responsáveis pelo tráfego entre o Atomix e o controlador. Dessa forma, ocorrendo o aumento no número de controladores os *switches* e a sua distribuição compõem o trafego entre os nós do Atomix e controladores.

Gráfico 11 – Média do Tráfego dos Enlaces do ONOS/Atomix



Fonte: Autoria Própria.

Os enlaces entre os nós do Atomix também não sofreram alterações significativas no tráfego. O mesmo comportamento foi observado no Gráfico 9, e ocorre por não existir um aumento considerável no número de *switches*. Como descrito anteriormente, para os enlaces entre os nós do Atomix, apenas o número de *switches* foi responsável pela alteração no tráfego. Assim, como pode ser observado no Gráfico 11, nem o aumento de controladores ou a distribuição de *switches* entre os controladores tem influencia no tráfego entre os nós do Atomix.

4.2 CONCLUSÃO DO CAPÍTULO

Este capítulo teve como objetivo a comparação da vazão obtida com o modelo desenvolvido no capítulo 3 e os valores obtidos a partir de medições no ambiente de simulação. Também foi analisada a influência do aumento de controladores na vazão. A seção 4.1.1 conteve o primeiro cenário composto por quatro experimentos em uma topologia arbitrária que teve como objetivo simular uma rede mais próxima de uma topologia real com o propósito de buscar uma distribuição de *switches* com um menor tráfego e validar o modelo desenvolvido. A seção 4.1.2 apresentou o segundo cenário com o objetivo de mostrar a influência que a adição de controladores teve no tráfego do *cluster*.

Os resultados do cenário 1 mostram que os valores teóricos da vazão são próximos

aos valores obtidos com as medições no ambiente de simulação, o que demostra que o modelo pode ser usado para determinar a vazão para qualquer distribuição de *switches* na rede. Ou seja, não é necessário montar um ambiente de simulação para obter os valores de vazão nos enlaces. Os resultados mostram também que uma distribuição igual de *switches* nos controladores é a topologia com o menor tráfego e se mostrou com a opção mais viável para ser utilizada em um projeto de redes SDN.

O cenário 2 mostra que o aumento de controladores na rede faz com que o tráfego de cada enlace diminua. Mas deve ser lembrado que há o custo de um número maior de controladores e o número de enlaces. Ou seja, com 3 controladores, a vazão de cada enlace é de aproximadamente 280 kbps para cada um dos 3 enlaces. Para 9 controladores, a vazão é de aproximadamente 100 kbps, mas o número de enlaces é de 36. Ou seja, o número de controladores deve ser o menor possível desde que suporte a carga de *switches* na rede.

5 CONCLUSÃO

O objetivo geral desse trabalho foi o desenvolvimento de um modelo para o tráfego em um *cluster* com o controlador ONOS gerenciado pelo *framework* Atomix. Desse modo foi necessário definir um ambiente que possibilitasse essa análise bem como a medição do tráfego gerado no *cluster*. O modelo do *cluster* para o ONOS sem a utilização do Atomix foi acrescentado para a comparação e compreensão das modificações que o *cluster* sofreu, bem como a existência de um modelo de tráfego para ele. Através da análise dos resultados obtidos com as medições no *cluster* ONOS e *cluster* ONOS/Atomix, foram possíveis as seguintes conclusões:

1. A mudança de versão do ONOS 1.4.1 utilizada no trabalho de Muqaddas *et al.* (2017) para a versão 1.13.10 utilizada nesse trabalho obteve um tráfego similar possibilitando a utilização dos valores teóricos no *cluster* ONOS;

2. O tráfego gerado entre os controladores do *cluster* ONOS/Atomix seguem um padrão similar ao observado no *cluster* ONOS, o que possibilitou a utilização do modelo de tráfego desenvolvido por Muqaddas *et al.* (2017) com os valores atualizados para o *cluster* ONOS/Atomix, obtidos através dos experimentos deste trabalho, utilizados como base para os valores teóricos do cenário 1 e cenário 2;

3. O tráfego gerado entre os controladores e entre o Atomix e o controlador é proporcional ao número de *switches* existentes em cada controlador da topologia de rede;

4. A distribuição arbitrária dos *switches* revelou um aumento no tráfego por parte do controlador com o maior número de *switches*, tanto no enlace que compõem a comunicação entre os controladores, quanto ao enlace entre o nó do Atomix líder para o controlador com o maior número de *switches*. Desse modo, uma divisão igualitária dos *switches* pode evitar a sobrecarga dos enlaces na rede;

5. O aumento no número de controladores revelou a redução no tráfego gerado entre os controladores, embora o tráfego global cresça proporcionalmente devido ao aumento no número de enlaces;

6. O aumento no número de controladores não teve efeito no enlace que constitui a comunicação entre os nós do Atomix.

O atual trabalho possibilitou uma melhor compreensão da arquitetura de *cluster* do ONOS com o Atomix. Através dos valores obtidos nos experimentos em conjunto das equações, é possível prever o tráfego com qualquer número de *switches* independentemente de sua distribuição para o pior caso em um *cluster* com 3 controladores. Essa previsão pode ser feita nos enlaces que constituem o tráfego entre controladores, entre controladores e o líder do Atomix, e entre os nós do Atomix sem a necessidade de experimentos.

Os resultados também possibilitaram uma análise de como o número de *switches* e a sua distribuição afetam o tráfego do *cluster*. Este estudo trouxe conclusões que podem contribuir para um rendimento superior na elaboração de projetos em redes SDN com o modelo do *cluster* ONOS e para o modelo do *cluster* ONOS gerenciado pelo Atomix.

5.1 TRABALHOS FUTUROS

Como descrito anteriormente foi possível entender o funcionamento do *cluster* ONOS/ Atomix, e através dos experimentos, determinar o que ocasiona o tráfego entre os enlaces bem como o seu custo na topologia. Desse modo, foram obtidos os valores descritos no modelo devido aos enlaces e *switches*. O próximo passo seria considerar no modelo a adição de *host* na topologia, tornando necessária a elaboração de uma série de experimentos com um número diversificado de *hosts* e, posteriormente, a divisão arbitrária dos mesmo entre os controladores. Algo semelhante ao mostrado nos experimentos com os *switches*. Assim sendo, um trabalho futuro seria o desenvolvimento de um modelo que inclua o tráfego devido a adição de *hosts* para o *cluster* ONOS/Atomix.

Durante o desenvolvimento do trabalho bem como a execução dos experimentos foi possível entender como protocolos de consistência como o *Anti-Entropy* e o Raft atuam no *cluster*. Porém, para analisar dados como tempo de eleição de um líder ou problemas relacionados a consistência da base de dados e tempo de inatividade da rede, seriam necessários experimentos que simulassem determinadas falhas. Outra possibilidade seria o desenvolvimento de um ambiente que simulasse no *cluster* ONOS/Atomix, falhas de enlaces, *switches*, controladores e nós do Atomix com o objetivo de alcançar um cenário que consiga ter um bom desempenho com tais adversidades.

REFERÊNCIAS

ALMADANI, Basem; BEG, Abdurrahman; MAHMOUD, Ashraf. Dsf: A distributed sdn control plane framework for the east/west interface. **IEEE Access**, IEEE, v. 9, p. 26735–26754, 2021.

ATOMIX. **Atomix User Manual**. 2021. *Disponível em: https://atomix.io/docs/latest/user-manual/*. Acesso em Abril de 2021.

BANNOUR, Fetia; SOUIHI, Sami; MELLOUK, Abdelhamid. Adaptive state consistency for distributed onos controllers. *In*: IEEE. **2018 IEEE Global Communications Conference (GLOBECOM)**. *[S.l.]*, 2018. p. 1–7.

BANNOUR, Fetia; SOUIHI, Sami; MELLOUK, Abdelhamid. Distributed sdn control: Survey, taxonomy, and challenges. **IEEE Communications Surveys & Tutorials**, IEEE, v. 20, n. 1, p. 333–354, 2018.

BANNOUR, Fetia; SOUIHI, Sami; MELLOUK, Abdelhamid. Adaptive distributed sdn controllers: Application to content-centric delivery networks. **Future Generation Computer Systems**, Elsevier, v. 113, p. 78–93, 2020.

CAMPANELLA, Andrea; YAN, Boyuan; CASELLAS, Ramon; GIORGETTI, Alessio; LOPEZ, Victor; ZHAO, Yongli; MAYORAL, Arturo. Reliable optical networks with odtn: resiliency and fail-over in data and control planes. **Journal of Lightwave Technology**, IEEE, v. 38, n. 10, p. 2755–2764, 2020.

CHO, Sungchol; LEE, SeungHun; PANICHPATTANAKU, Wasimon; THAENCHAIKUN, Aj Chakadkit; HAN, Sunyoung. Architecture for sdn-independent gateway. *In*: IEEE. **2019 23rd International Computer Science and Engineering Conference (ICSEC)**. *[S.l.]*, 2019. p. 76–81.

COKER, Oswald; AZODOLMOLKY, Siamak. **Software-defined Networking with OpenFlow: Deliver Innovative Business Solutions**. *[S.l.]*: Packt Publishing Ltd, 2017.

EL-GEDER, Suad. **Performance evaluation using multiple controllers with different flow setup modes in the software defined network architecture**. 2017. Tese (Doutorado) — Brunel University London, 2017.

GONZALEZ, Carlos J; FLAUZAC, Olivier; NOLOT, Florent. The network virtualization to support the scalability of the internet of things. *In*: SPRINGER, CHAM. **International Conference on Smart Technologies, Systems and Applications**. *[S.l.]*, 2019. p. 40–51.

HALTERMAN, Jordan. **Distributed Systems in ONOS with Atomix 3; Architecture and Implementation**. 2018. *Disponível em: https://opennetworking.org/wp-content/uploads/2018/12/Distributed-Systems-in-ONOS-with-Atomix-3.pdf*. Acesso em Fevereiro de 2021.

HANMER, Robert; JAGADEESAN, Lalita; MENDIRATTA, Veena; ZHANG, Heng. Friend or foe: Strong consistency vs. overload in high-availability distributed systems and sdn. *In*: IEEE. **2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)**. *[S.l.]*, 2018. p. 59–64.

HU, Fei. **Network Innovation through OpenFlow and SDN: Principles and Design**. *[S.l.]*: Crc Press, 2014.

HU, Fei; HAO, Qi; BAO, Ke. A survey on software-defined network and openflow: From concept to implementation. **IEEE Communications Surveys & Tutorials**, IEEE, v. 16, n. 4, p. 2181–2206, 2014.

HU, Tao; GUO, Zehua; YI, Peng; BAKER, Thar; LAN, Julong. Multi-controller based software-defined networking: A survey. **IEEE Access**, IEEE, v. 6, p. 15980–15996, 2018.

KARAKUS, Murat; DURRESI, Arjan. A survey: Control plane scalability issues and approaches in software-defined networking (sdn). **Computer Networks**, Elsevier, v. 112, p. 279–293, 2017.

KREUTZ, Diego; RAMOS, Fernando; VERISSIMO, Paulo; ROTHENBERG, Christian Esteve; AZODOLMOLKY, Siamak; UHLIG, Steve. Software-defined networking: A comprehensive survey. **arXiv preprint arXiv:1406.0440**, 2014.

LARA, Adrian; KOLASANI, Anisha; RAMAMURTHY, Byrav. Network innovation using openflow: A survey. **IEEE communications surveys & tutorials**, IEEE, v. 16, n. 1, p. 493–512, 2013.

LIU, Shaoteng; STEINERT, Rebecca; KOSTIC, Dejan. Flexible distributed control plane deployment. *In*: IEEE. **NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium**. *[S.l.]*, 2018. p. 1–7.

MARSCHKE, Doug; DOYLE, Jeff; MOYER, Pete. **Software Defined Networking (SDN): Anatomy of OpenFlow Volume I**. *[S.l.]*: Lulu. com, 2015. v. 1.

MÜLLER, Michael; KÖHLER-BUSSMEIER, Michael. Availability analysis of the onos architecture. 2021.

MUQADDAS, Abubakar Siddique; GIACCONE, Paolo; BIANCO, Andrea; MAIER, Guido. Inter-controller traffic to support consistency in onos clusters. **IEEE Transactions on Network and Service Management**, IEEE, v. 14, n. 4, p. 1018–1031, 2017.

NADEAU, Thomas D; GRAY, Ken. **SDN: Software Defined Networks: an authoritative review of network programmability technologies**. *[S.l.]*: "O'Reilly Media, Inc.", 2013.

NUNES, Bruno Astuto A; MENDONCA, Marc; NGUYEN, Xuan-Nam; OBRACZKA, Katia; TURLETTI, Thierry. A survey of software-defined networking: Past, present, and future of programmable networks. **IEEE Communications Surveys & Tutorials**, IEEE, v. 16, n. 3, p. 1617–1634, 2014.

ONGARO, Diego; OUSTERHOUT, John. In search of an understandable consensus algorithm. *In*: **2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)**. *[S.l.: s.n.]*, 2014. p. 305–319.

ONOS. **ONOS User Guide**. 2019. *Disponível em: https://wiki.onosproject.org/display/ONOS/Guides*. Acesso em Setembro de 2019.

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

PETERSON CARMELO CASCONE, Brian O'Connor Thomas Vachuska Larry; DAVIE, Bruce. **Software-Defined Networks: A Systems Approach**. *[S.l.]*: "Systems Approach LLC ", 2020.

QI, Heng; LI, Keqiu. **Software defined networking applications in distributed datacenters**. *[S.l.]*: Springer, 2016.

SAKIC, Ermin; KELLERER, Wolfgang. Response time and availability study of raft consensus in distributed sdn control plane. **IEEE Transactions on Network and Service Management**, IEEE, v. 15, n. 1, p. 304–318, 2017.

SUBRAMANIAN, Sriram; VORUGANTI, Sreenivas. **Software-Defined Networking (SDN) with OpenStack**. *[S.l.]*: Packt Publishing Ltd, 2016.

VIZARRETA, Petra; TRIVEDI, Kishor; MENDIRATTA, Veena; KELLERER, Wolfgang; MAS-MACHUCA, Carmen. Dason: Dependability assessment framework for imperfect distributed sdn implementations. **IEEE Transactions on Network and Service Management**, IEEE, v. 17, n. 2, p. 652–667, 2020.

WOOS, Doug; WILCOX, James R; ANTON, Steve; TATLOCK, Zachary; ERNST, Michael D; ANDERSON, Thomas. Planning for change in a formal verification of the raft consensus protocol. *In*: **Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs**. *[S.l.: s.n.]*, 2016. p. 154–165.

YU, Tao; HONG, Yang; CUI, Hongyan; JIANG, Hongxiang. A survey of multi-controllers consistency on sdn. *In*: IEEE. **2018 4th International Conference on Universal Village (UV)**. *[S.l.]*, 2018. p. 1–6.

ZHANG, Yang; RAMADAN, Eman; MEKKY, Hesham; ZHANG, Zhi-Li. When raft meets sdn: How to elect a leader and reach consensus in an unruly network. *In*: **Proceedings of the First Asia-Pacific Workshop on Networking**. *[S.l.: s.n.]*, 2017. p. 1–7.

APÊNDICE A - *Script* Topologia

APÊNDICE A – *SCRIPT* TOPOLOGIA

Script da topologia com três controladores e vinte *switches* em topologia linear utilizado no modelo de tráfego. Os demais *Scripts* com as topologias utilizadas para este trabalho apenas diferem o número de controladores e *switches*. Assim como a distribuição dos *switches* nos controladores de acordo com cada cenário. Esse formato de topologia foi utilizado para o *cluster* ONOS e ONOS/Atomix.

```
from mininet.net import Mininet
from mininet.node import Controller, RemoteController,
↪  OVSController, OVSSwitch
from mininet.node import CPULimitedHost, Host, Node
from mininet.node import OVSKernelSwitch, UserSwitch
from mininet.node import IVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf
from subprocess import call

def 3c20s():

    net = Mininet( controller=RemoteController,
    ↪  switch=OVSSwitch )

    print "Criando Controladores"

    c1 = net.addController( 'c1', ip='172.17.0.5',port=6653 )
    c2 = net.addController( 'c2', ip='172.17.0.6',port=6653 )
    c3 = net.addController( 'c3', ip='172.17.0.7',port=6653 )

    print "Criando Switches"

    s1 = net.addSwitch( 's1', cls=OVSKernelSwitch )
```

```
    s2 = net.addSwitch( 's2', cls=OVSKernelSwitch )
    s3 = net.addSwitch( 's3', cls=OVSKernelSwitch )
    s4 = net.addSwitch( 's4', cls=OVSKernelSwitch )
    s5 = net.addSwitch( 's5', cls=OVSKernelSwitch )
    s6 = net.addSwitch( 's6', cls=OVSKernelSwitch )
    s7 = net.addSwitch( 's7', cls=OVSKernelSwitch )
    s8 = net.addSwitch( 's8', cls=OVSKernelSwitch )
    s9 = net.addSwitch( 's9', cls=OVSKernelSwitch )
    s10 = net.addSwitch( 's10', cls=OVSKernelSwitch )
    s11 = net.addSwitch( 's11', cls=OVSKernelSwitch )
    s12 = net.addSwitch( 's12', cls=OVSKernelSwitch )
    s13 = net.addSwitch( 's13', cls=OVSKernelSwitch )
    s14 = net.addSwitch( 's14', cls=OVSKernelSwitch )
    s15 = net.addSwitch( 's15', cls=OVSKernelSwitch )
    s16 = net.addSwitch( 's16', cls=OVSKernelSwitch )
    s17 = net.addSwitch( 's17', cls=OVSKernelSwitch )
    s18 = net.addSwitch( 's18', cls=OVSKernelSwitch )
    s19 = net.addSwitch( 's19', cls=OVSKernelSwitch )
    s20 = net.addSwitch( 's20', cls=OVSKernelSwitch )

    print "Criando Enlaces entre Switches (s)"

    net.addLink( s1, s2 )
    net.addLink( s2, s3 )
    net.addLink( s3, s4 )
    net.addLink( s4, s5 )
    net.addLink( s5, s6 )
    net.addLink( s6, s7 )
    net.addLink( s7, s8 )
    net.addLink( s8, s9 )
    net.addLink( s9, s10 )
    net.addLink( s10, s11 )
```

```
    net.addLink( s11, s12 )
    net.addLink( s12, s13 )
    net.addLink( s13, s14 )
    net.addLink( s14, s15 )
    net.addLink( s15, s16 )
    net.addLink( s16, s17 )
    net.addLink( s17, s18 )
    net.addLink( s18, s19 )
    net.addLink( s19, s20 )

    net.build()

    print "Iniciando Controladores"

    c1.start()
    c2.start()
    c3.start()

    print "Iniciando Switches (s) no controlador"

    s1.start( [ c1 ] )
    s2.start( [ c1 ] )
    s3.start( [ c1 ] )
    s4.start( [ c1 ] )
    s5.start( [ c1 ] )
    s6.start( [ c1 ] )
    s7.start( [ c1 ] )
    s8.start( [ c1 ] )
    s9.start( [ c1 ] )
    s10.start( [ c1 ] )
    s11.start( [ c1 ] )
    s12.start( [ c1 ] )
```

```
    s13.start( [ c1 ] )
    s14.start( [ c1 ] )
    s15.start( [ c1 ] )
    s16.start( [ c1 ] )
    s17.start( [ c1 ] )
    s18.start( [ c1 ] )
    s19.start( [ c1 ] )
    s20.start( [ c1 ] )

    print "Executando Cliente"

    CLI( net )

    print "Parando Rede"

    net.stop()

if __name__ == '__main__':

    setLogLevel( 'info' )  # for CLI output
    3c20s()
```

ANEXO A - *Script onos-form-cluster.sh*

ANEXO A – *SCRIPT ONOS-FORM-CLUSTER.SH*

Script onos-form-cluster.sh utilizado na configuração do *cluster* ONOS 1.13.10. *Script* disponibilizado pelo projeto ONOS (ONOS, 2019).

```
#!/bin/bash
#
 ↪ ----------------------------------------------------------------
# Forms ONOS cluster using REST API of each separate instance.
#
 ↪ ----------------------------------------------------------------
usage() {
  echo "usage: $PROG -u <user> -p <password> -h ip1 ip2
   ↪  ip3..."
  exit 1
}
nodes=
while getopts u:p:h o; do
    case "$o" in
        u) user=$OPTARG;;
        p) password=$OPTARG;;
        h) usage;;
        *) echo $o ;;
    esac
done

shift $(($OPTIND - 1))

[ -z "$user" -o -z "$password" ] && usage

[ $# -lt 2 ] && usage

ip=$1
```

```
shift
nodes=$*

ipPrefix=${ip%.*}

aux=/tmp/${ipPrefix}.cluster.json
trap "rm -f $aux" EXIT

echo "{ \"nodes\": [ { \"ip\": \"$ip\" }" > $aux
for node in $nodes; do
    echo ", { \"ip\": \"$node\" }" >> $aux
done
echo "], \"ipPrefix\": \"$ipPrefix.*\" }" >> $aux

for node in $ip $nodes; do
    echo "Forming cluster on $node..."
    curl -X POST -u $user:$password
        http://$node:8181/onos/v1/cluster/configuration -d
        ↪  @$aux
done
```

ANEXO B - *Script atomix-gen-config*

ANEXO B – *SCRIPT ATOMIX-GEN-CONFIG*

Script atomix-gen-config utilizado para a configuração dos nós do Atomix 3.1.5. *Script* disponibilizado pelo projeto ONOS (ONOS, 2019).

```
#!/usr/bin/env python
"""
usage: Atomix-gen-config [-h] [-s PARTITION_SIZE] [-n
 ↪  NUM_PARTITIONS]
                         [node_ip] [filename] [node_ip
 ↪  [node_ip ...]]

Generate the partitions json file given a list of IPs or from
 ↪  the $OCC*
environment variables.

positional arguments:
  filename              File to write output to. If none is
 ↪  provided, output
                        is written to stdout.
  node_ip               IP Address(es) of the node(s) in the
 ↪  cluster. If no
                        IPs are given, will use the $OCC*
 ↪  environment
                        variables. NOTE: these arguemnts are
 ↪  only processed
                        after the filename argument.

optional arguments:
  -h, --help            show this help message and exit
  -s PARTITION_SIZE, --partition-size PARTITION_SIZE
                        Number of nodes per partition. Note
 ↪  that partition
```

```
                        sizes smaller than 3 are not fault
↪  tolerant. Defaults
                        to 3.
  -n NUM_PARTITIONS, --num-partitions NUM_PARTITIONS
                        Number of partitions. Defaults to the
↪  number of nodes
                        in the cluster.
"""

from os import environ
import argparse
import re
import json

convert = lambda text: int(text) if text.isdigit() else
↪  text.lower()
alphanum_key = lambda key: [convert(c) for c in
↪  re.split('([0-9]+)', key)]

def get_vars_by_type(type):
    vars = []
    for var in environ:
        if re.match(r"{}[0-9]+".format(type), var):
            vars.append(var)
    return sorted(vars, key=alphanum_key)


def get_vars():
    vars = get_vars_by_type('OCC')
    if len(vars) == 0:
        vars = get_vars_by_type('OC')
    return vars
```

```
def get_local_node(node, ips=None):
    if not ips:
        ips = [environ[v] for v in get_vars()]
    return 'Atomix-{}'.format(ips.index(node) + 1)


def get_nodes(ips=None, default_port=5679):
    node = lambda id, ip, port: {'id': id, 'address':
    ↪  '{}:{}'.format(ip, port)}
    result = []
    if not ips:
        ips = [environ[v] for v in get_vars()]
    i = 1
    for ip_string in ips:
        address_tuple = ip_string.split(":")
        if len(address_tuple) == 3:
            id=address_tuple[0]
            ip=address_tuple[1]
            port=int(address_tuple[2])
        else:
            id='Atomix-{}'.format(i)
            i += 1
            ip=ip_string
            port=default_port
        result.append(node(id, ip, port))
    return result


def get_local_address(node, ips=None, default_port=5679):
    if not ips:
```

```
        ips = [environ[v] for v in get_vars()]
    for ip_string in ips:
        address_tuple = ip_string.split(":")
        if len(address_tuple) == 3:
            id=address_tuple[0]
            ip=address_tuple[1]
            port=int(address_tuple[2])
            if node == id or node == ip:
                return '{}:{}'.format(ip, port)
    return '{}:{}'.format(node, default_port)

if __name__ == '__main__':
  parser = argparse.ArgumentParser(
      description="Generate the partitions json file given a
      ↪  list of IPs or from the $OCC* environment
      ↪  variables.")
  parser.add_argument(
      '-s', '--partition-size', type=int, default=3,
      help="Number of nodes per partition. Note that partition
      ↪  sizes smaller than 3 are not fault tolerant.
      ↪  Defaults to 3." )
  parser.add_argument(
      '-n', '--num-partitions', type=int,
      help="Number of partitions. Defaults to the number of
      ↪  nodes in the cluster." )
 # TODO: make filename and nodes independent. This will break
 ↪  backwards compatibility with existing usage.
  parser.add_argument(
      'node', metavar='node_ip', type=str, help='IP address of
      ↪  the node for which to generate the configuration')
  parser.add_argument(
     'filename', metavar='filename', type=str, nargs='?',
```

```
    help='File to write output to. If none is provided,
    ↪  output is written to stdout.')
parser.add_argument(
    'nodes', metavar='node_ip', type=str, nargs='*',
    help='IP Address(es) of the node(s) in the cluster. If
    ↪  no IPs are given, ' +
         'will use the $OCC* environment variables. NOTE:
         ↪  these arguemnts' +
         ' are only processed after the filename argument.')

args = parser.parse_args()
filename = args.filename
partition_size = args.partition_size
local_member_id = get_local_node(args.node)
local_member_address = get_local_address(args.node,
↪  args.nodes)
nodes = get_nodes(args.nodes)
num_partitions = args.num_partitions
if not num_partitions:
  num_partitions = len(nodes)

data = {
    'cluster': {
        'clusterId': 'onos',
        'node': {
            'id': local_member_id,
            'address': local_member_address
        },
        'discovery': {
            'type': 'bootstrap',
            'nodes': nodes
        }
```

```
    },
    'managementGroup': {
        'type': 'raft',
        'partitions': 1,
        'partitionSize': len(nodes),
        'members': [node['id'] for node in nodes],
        'storage': {
            'level': 'mapped'
        }
    },
    'partitionGroups': {
        'raft': {
            'type': 'raft',
            'partitions': num_partitions,
            'partitionSize': partition_size,
            'members': [node['id'] for node in nodes],
            'storage': {
                'level': 'mapped'
            }
        }
    }
}
output = json.dumps(data, indent=4)

if filename:
  with open(filename, 'w') as f:
    f.write(output)
else:
  print output
```

ANEXO C - *Script onos-gen-config*

ANEXO C – *SCRIPT ONOS-GEN-CONFIG*

Script onos-gen-config utilizado na configuração dos controladores ONOS 2.2.2 para com os nós do Atomix 3.1.5. *Script* disponibilizado pelo projeto ONOS (ONOS, 2019).

```
#!/usr/bin/env python
"""
usage: onos-gen-config [-h] [-s PARTITION_SIZE] [-n
 ↪  NUM_PARTITIONS]
                              [filename] [node_ip [node_ip ...]]

Generate the partitions json file given a list of IPs or from
 ↪  the $OCC*
environment variables.

positional arguments:
  filename              File to write output to. If none is
 ↪  provided, output
                        is written to stdout.
  node_ip               IP Address(es) of the node(s) in the
 ↪  cluster. If no
                        IPs are given, will use the $OC*
 ↪  environment
                        variables. NOTE: these arguments are
 ↪  only processed
                        after the filename argument.

optional arguments:
  -h, --help            show this help message and exit
  -s PARTITION_SIZE, --partition-size PARTITION_SIZE
                        Number of nodes per partition. Note
 ↪  that partition
```

```
                        sizes smaller than 3 are not fault
↪  tolerant. Defaults
                        to 3.
  -n NUM_PARTITIONS, --num-partitions NUM_PARTITIONS
                        Number of partitions. Defaults to the
↪  number of nodes
                        in the cluster.
"""

from os import environ
import argparse
import re
import json

convert = lambda text: int(text) if text.isdigit() else
↪  text.lower()
alphanum_key = lambda key: [convert(c) for c in
↪  re.split('([0-9]+)', key)]


def get_vars_by_type(type):
    vars = []
    for var in environ:
        if re.match(r"{}[0-9]+".format(type), var):
            vars.append(var)
    return sorted(vars, key=alphanum_key)


def get_vars():
    vars = get_vars_by_type('OCC')
    if len(vars) == 0:
        vars = get_vars_by_type('OC')
```

```
    return vars


def get_nodes(ips=None, default_port=5679):
    node = lambda id, ip, port: {'id': id, 'ip': ip, 'port':
    ↪  port}
    result = []
    if not ips:
        ips = [environ[v] for v in get_vars()]
    i = 1
    for ip_string in ips:
        address_tuple = ip_string.split(":")
        if len(address_tuple) == 3:
            id = address_tuple[0]
            ip = address_tuple[1]
            port = int(address_tuple[2])
        else:
            id = 'Atomix-{}'.format(i)
            i += 1
            ip = ip_string
            port = default_port
        result.append(node(id, ip, port))
    return result


if __name__ == '__main__':
    parser = argparse.ArgumentParser(
        description="Generate the partitions json file given a
        ↪  list of IPs or from environment variables.")
    parser.add_argument(
        'node', metavar='node_ip', type=str, nargs='?',
```

```
        help='The node for which to generate the
        ↪  configuration')
    parser.add_argument(
        'filename', metavar='filename', type=str, nargs='?',
        help='File to write output to. If none is provided,
        ↪  output is written to stdout.')
    parser.add_argument(
        '--nodes', '-n', metavar='node_ip', type=str,
        ↪  nargs='+',
        help='IP Address(es) of the storage nodes. If no IPs
        ↪  are given, ' +
             'will use the $OCC* or $OC* environment
             ↪  variables. NOTE: these arguments' +
             ' are only processed after the filename
             ↪  argument.')

    args = parser.parse_args()
    node = args.node
    print node
    filename = args.filename
    nodes = get_nodes(args.nodes)

    data = {
        'name': 'onos',
        'node': {
            'id': node,
            'ip': node,
            'port': 9876
        },
        'storage': nodes
    }
    output = json.dumps(data, indent=4)
```

```
    if filename:
        with open(filename, 'w') as f:
            f.write(output)
    else:
        print output
```