UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DAVID ANDERSON BEZERRA SILVA

PROJETO E IMPLEMENTAÇÃO DE UM QUIZ MULTIUSUÁRIO BASEADO EM UM CENÁRIO DE REDES DEFINIDAS POR SOFTWARE

MOSSORÓ
2017

DAVID ANDERSON BEZERRA SILVA

PROJETO E IMPLEMENTAÇÃO DE UM QUIZ MULTIUSUÁRIO BASEADO EM UM CENÁRIO DE REDES DEFINIDAS POR SOFTWARE

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Helcio Wagner da Silva, Prof. Dr.

MOSSORÓ

2017

S586p Silva, David Anderson Bezerra.
Projeto e implementação de um Quiz multiusuário baseado em um cenário de Redes Definidas por Software / David Anderson Bezerra Silva. - 2017.
31 f. : il.

Orientador: Helcio Wagner da Silva.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Ciência da Computação, 2017.

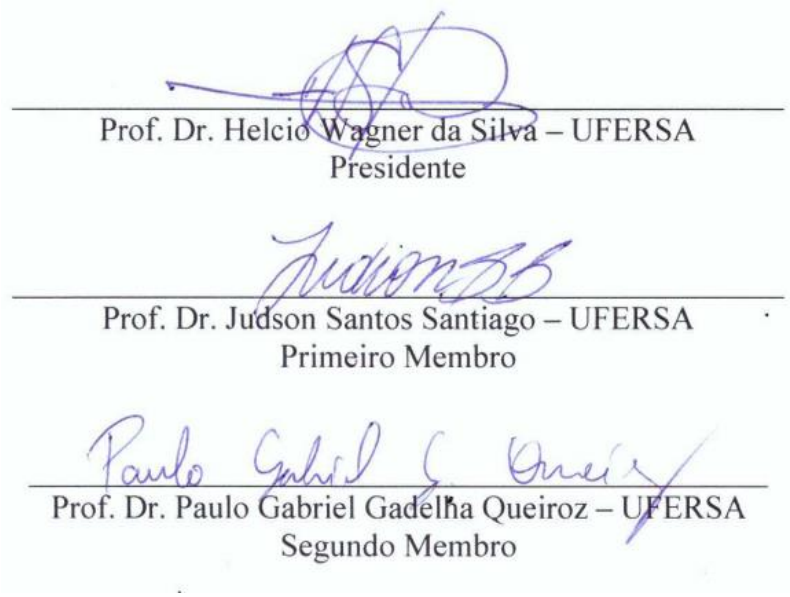
1. Redes Definidas por Software. 2. OpenFlow. 3. Quiz multiusuário. 4. REST. I. da Silva, Helcio Wagner, orient. II. Título.

PROJETO E IMPLEMENTAÇÃO DE UM QUIZ MULTIUSUÁRIO BASEADO EM UM CENÁRIO DE REDES DEFINIDAS POR SOFTWARE

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Ciência da Computação.

Defendida em: 12 / 05 / 2017 .

BANCA EXAMINADORA



Prof. Dr. Helcio Wagner da Silva – UFERSA
Presidente

Prof. Dr. Judson Santos Santiago – UFERSA
Primeiro Membro

Prof. Dr. Paulo Gabriel Gadelha Queiroz – UFERSA
Segundo Membro

AGRADECIMENTOS

Primeiramente agradeço a Deus por tudo que tem feito em minha vida, por ser perfeito e sempre me proporcionar o melhor.

Agradeço aos meus pais José Ivan e Maria da Conceição por sempre estarem do meu lado me apoiando mesmo não conhecendo bem essa área de tecnologia.

Agradeço a minha irmã Daísa Carla que teve paciência de ler o meu trabalho e sempre está disponível para me ouvir.

Agradeço a minha namorada Fábia Elizabeti, que compartilhou comigo esse momento, foi muito paciente em minhas ausências e me ajudou bastante dando dicas e apoio moral para o desenvolvimento deste.

Agradeço ao meu orientador Helcio Wagner que idealizou este projeto, pela força e dedicação que aplicou a este trabalho.

Agradeço a Banca Examinadora, Paulo Gabriel e Judson Santiago por se disponibilizarem a avaliar este trabalho e serem excelentes docentes durante a minha graduação. Aos professores Leonardo e Danielle Casillo pela grande ajuda durante esse período na universidade e na carreira profissional.

Agradeço aos meus amigos da universidade, Luiz Felipe, Ulysses, Ramiro, Ademar, Felipe Leão, Dijon, Paulo, Alessandro, Andrezza e os demais, que me ajudaram durante a graduação, amizade e a proporcionar momentos de gargalhadas.

Se existem várias maneiras de executar a mesma tarefa, escolha apenas uma. Ter duas ou mais maneiras de fazer isso é pedir problemas.

Andrew Tanenbaum

RESUMO

Diante de um mundo cada vez mais conectado, as redes de computadores precisaram evoluir para acompanhar esse crescimento. Suas tecnologias não evoluíram tão rápido quanto as suas necessidades, seus equipamentos de redes eram implementados por um software fechado e executado em um hardware proprietário. Nessa situação, ficou cada vez mais difícil a gestão das redes junto com a inflexibilidade das tecnologias e das arquiteturas. Com o avanço da padronização do protocolo OpenFlow para programar o plano de dados dos equipamentos de redes, surgiram as Redes Definidas por Software. Neste novo paradigma, a rede passa a ser controlada e gerenciada por uma plataforma de software central separada do hardware que encaminha o tráfego, atendendo os problemas de inflexibilidade e suprindo as necessidades de aplicações de redes futuras por meio de uma arquitetura de rede programável. Neste trabalho, foi projetado um jogo multiusuário do tipo “quiz” numa Rede Definida por Software na qual o controlador desta rede assume o papel de Servidor do jogo. Através disto, são demonstrados os benefícios alcançados quando a funcionalidade do jogo está junto ao controlador e as decisões são elevadas às camadas acima utilizando uma API REST desenvolvida para a comunicação entre controlador e aplicação.

Palavras-chave: Redes Definidas por Software. OpenFlow. Quiz multiusuário. REST.

LISTA DE FIGURAS

Figura 1 – Visão geral do paradigma SDN.....12
Figura 2 – Uma rede operando com um protocolo de roteamento dinâmico 13
Figura 3 – A mesma rede operando de acordo com o paradigma SDN.....14
Figura 4 – Arquitetura SDN com suas APIs 15
Figura 5 – Arquitetura de um switch OpenFlow 16
Figura 6 – Estrutura de uma tabela de fluxo.....17
Figura 7 – Quiz no cenário SDN com o Controlador assumindo função de Servidor 21
Figura 8 – Visão geral da arquitetura 22
Figura 9 – API disponibilizada pelo Controlador – Método para criação do Quiz.....23
Figura 10 – API disponibilizada pelo Controlador – Método para a remoção do Quiz.....23
Figura 11 – API disponibilizada pela Aplicação – Método para criação dos status.....23
Figura 12 – API disponibilizada pela Aplicação – Método para atualização dos status.....24
Figura 13 – Sequência das mensagens entre Servidor e Jogador durante o Quiz.....25
Figura 14 – Estrutura da mensagem de inicialização do Quiz.....26
Figura 15 – Estrutura da mensagem de envio de questões e da resposta do jogador.....26
Figura 16 – Estrutura da mensagem de finalização do Quiz para todos os Jogadores 26
Figura 17 – Janela da Aplicação 29

LISTA DE ABREVIATURAS E SIGLAS

SDN	Software Defined Networking
API	Application Programming Interface
REST	Representational State Transfer
IP	Internet Procotol
MAC	Media Access Control
NETCONF	Network Configuration
SSL	Secure Socket Layer
SNAC	Simple Network Access Control
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
ONF	Open Networking Foundation
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UDP	User Datagram Protocol

SUMÁRIO

1	**INTRODUÇÃO ……...……**	10
2	**REDES DEFINIDAS POR SOFTWARE (SDN)…….....**	12
2.1	**Openflow …….....**	15
2.1.1	Componentes da arquitetura Openflow.....	17
2.1.2	Mensagens OpenFlow.....	18
3	**REST …….....**	19
4	**A FERRAMENTA DESENVOLVIDA ….....…….....**	21
4.1	**Visão geral da arquitetura.....**	21
4.2	**API REST do Controlador.....**	22
4.3	**API REST da Aplicação.....**	23
4.4	**Procedimentos do Quiz – Controlador e Jogador.....**	24
4.5	**Aspectos de Implementação.....**	27
5	**CONSIDERAÇÕES FINAIS ……...**	30
	REFERÊNCIAS ……...	31

1 INTRODUÇÃO

Diante de um mundo cada vez mais conectado, as redes de computadores passaram a ser uma das áreas de grande interesse para a sociedade. A interconexão mundial de redes de computadores, chamada de Internet, apresenta um crescimento contínuo desde 1980 quando era um projeto de pesquisa que envolvia a criação de sites e agora chega a ser um sistema de comunicação que alcança pessoas em todos os países do mundo (COMER, 2015). O crescimento das redes de computadores gerou um grande impacto econômico, ocasionando a criação de indústrias para desenvolvimento de tecnologias de rede, produtos e serviços.

Apesar da evolução da Internet, suas tecnologias não evoluíram tão rápido quanto as suas necessidades. À medida em que a Internet tornou-se comercial, seus equipamentos de comutação tornaram-se “caixas pretas” ou seja, são soluções baseadas em software fechado executando em hardware proprietário, originando o engessamento da Internet (CHOWDHURY; BOUTABA, 2009). Com esse resultado, ficou cada vez mais difícil à gestão das redes junto com a inflexibilidade das tecnologias e das arquiteturas.

Esses obstáculos forçaram a comunidade científica a estudar novas propostas com vistas à maior abertura e flexibilidade dos equipamentos de comutação. Embora tais propostas fossem capazes de atender as necessidades, sua implementação e experimentação, eram dificultadas por se tratar, a internet, de uma infraestrutura em constante uso pela sociedade. Assim, elas não poderiam ser avaliadas em redes reais sem provocar a mudança de milhares de equipamentos de comutação instalados e acabaram sendo descartadas pelos administradores das redes.

Por outro lado, se o controle das tomadas de decisão fosse logicamente centralizado, não necessitando configurar equipamentos de comutação individualmente, haveria a possibilidade da definição do comportamento da rede através de um software. Isto resolveria o problema da inflexibilidade, haja vista que os equipamentos poderiam ser configurados a bel prazer pelos operadores de redes.

As Redes Definidas por Software (SDN – Software Defined Networking) constituem esse novo paradigma para a operação e gerência das redes de computadores. Este paradigma foi criado a partir de um projeto de seis anos desenvolvido em cima do conceito de *open source*, aberto a toda comunidade dos pesquisadores de tecnologias e soluções de redes, numa colaboração entre a Universidade de Stanford e a Universidade da Califórnia em Berkeley (MACKEOWN, 2008). O paradigma SDN facilita testes de novas propostas em redes reais sem impactar demasiadamente no desempenho daquelas redes. Dessa forma, as necessidades

de aplicações de redes futuras poderão ser atendidas por meio de uma arquitetura de rede programável.

Neste trabalho, foi projetado um jogo multiusuário do tipo “quiz” num cenário SDN no qual o Servidor do Quiz está incorporado ao Controlador, este sendo o principal componente e responsável por concentrar a comunicação com todos os equipamentos de comutação da Rede Definida por Software, e uma aplicação utilizando o estilo Transferência de Estado Representacional (REST – *Representational State Transfer*) é criada para administrar esta função no Controlador.

Este projeto tem o objetivo de demonstrar benefícios alcançados quando as decisões são elevadas a camadas acima (Aplicação) e os serviços (Servidor) são movidos para próximo ao Controlador ou, neste caso, incorporados àquele componente.

Foi desenvolvido um protocolo para a criação dos jogadores e do servidor. Em particular, o desenvolvimento do servidor consiste na incorporação da lógica do serviço ao Controlador Ryu na forma de um módulo específico. O desenvolvimento da aplicação REST requer a definição de métodos e a sua implementação utilizando a Application Programming Interface (API) correspondente, disponibilizada pelo Ryu.

Este trabalho organiza-se em cinco capítulos. No Capítulo 2, aborda-se o conceito de SDN e o protocolo OpenFlow, além de uma comparação entre as redes simples e as Redes Definidas por Software, mostra seus componentes de redes e arquitetura. No Capítulo 3, apresenta-se o estilo arquitetônico REST, suas vantagens e seus requisitos para serem usados na aplicação do Quiz. No Capítulo 4, apresenta-se a ferramenta desenvolvida, o que foi implementado para a criação desse projeto com base no estudo dos capítulos anteriores. No Capítulo 5, é feita uma análise final da vantagem de se criar um Quiz multiusuário em um cenário SDN com o Controlador fazendo a função de Servidor em vez de uma rede simples.

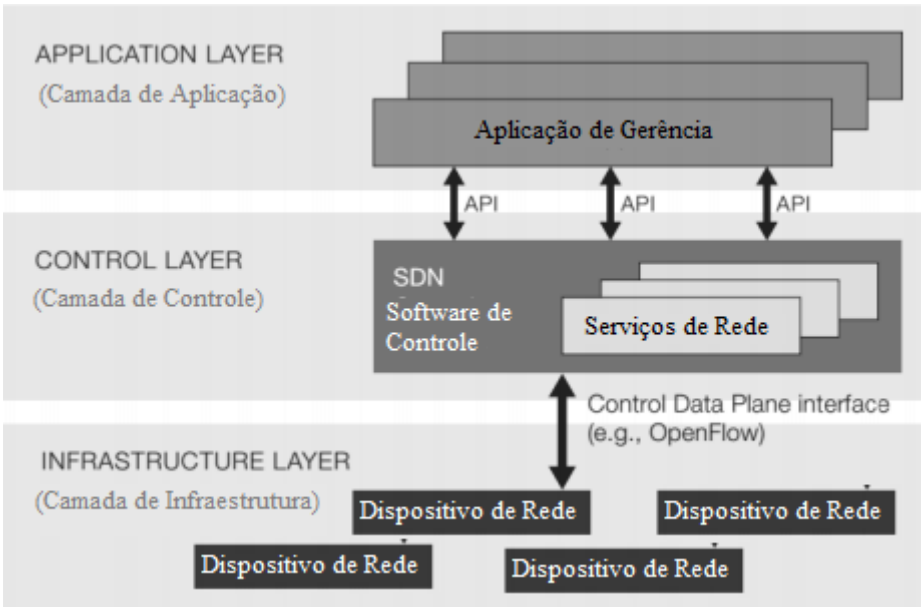
2 REDES DEFINIDAS POR SOFTWARE (SDN)

Redes Definidas por Software são um novo paradigma para a gerência de redes de computadores. Ele visa desacoplar os Planos de Controle e de Dados em equipamentos de rede. Assim, a rede passa a ser controlada e gerenciada por uma plataforma de software central separada do hardware que encaminha o tráfego.

O objetivo principal da SDN é criar uma rede ágil e flexível às mudanças rápidas com base nas necessidades e exigências das empresas e das aplicações, tudo a partir de uma plataforma programável centralizada (SDXCENTRAL, 2017).

A Figura 1 foi extraída de um *whitepaper* da *Open Networking Foundation* (ONF), um consórcio industrial sem fins lucrativos voltado à padronização de elementos críticos da SDN. Ela ilustra que toda a inteligência dos equipamentos é movida para uma entidade de software centralizada que na prática, é denominada de Controlador.

Figura 1 – Visão geral do paradigma SDN.



Fonte: Extraída de (OPEN NETWORKING FOUNDATION, 2012).

A inteligência de rede é logicamente centralizada em controladores SDN baseados em software que mantêm uma visão global da rede, que aparece para os aplicativos e os mecanismos de diretiva como um único comutador lógico (OPEN NETWORKING FOUNDATION, 2012). Dessa forma não será preciso configurar cada equipamento separadamente e sim somente ao controlador SDN, deixando mais gerenciável a rede.

Outro conceito importante de SDN é que ele é baseado numa estratégia de plataforma aberta, tendo a ideia de criar uma tecnologia que funcione em diferentes sistemas de hardware

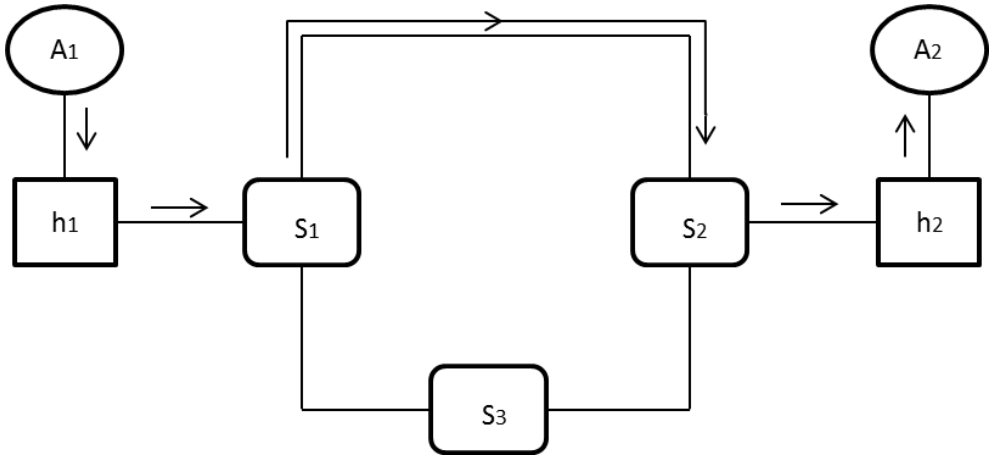
e software de rede para incentivar a inovação e aumentar a flexibilidade (UNDERDAHL; KINGHORN, 2015).

A SDN simplifica o projeto e operação da rede porque as instruções são fornecidas pelos controladores de SDN em vez de dispositivos específicos de fornecedores e protocolos, permitindo abrir as portas para novas soluções de hardware.

Diante de todas essas definições do paradigma SDN, se permite dizer que este apresenta uma flexibilidade inalcançável com os protocolos de roteamento hoje em operação.

O fato é que os protocolos de roteamento, uma vez que calculam o melhor caminho até um determinado destino, descartam todos os demais percursos possíveis. A Figura 2 ilustra uma interconexão de rede simples, que dá suporte a comunicação entre aplicações A_1 e A_2.

Figura 2 – Uma rede operando com um protocolo de roteamento dinâmico.



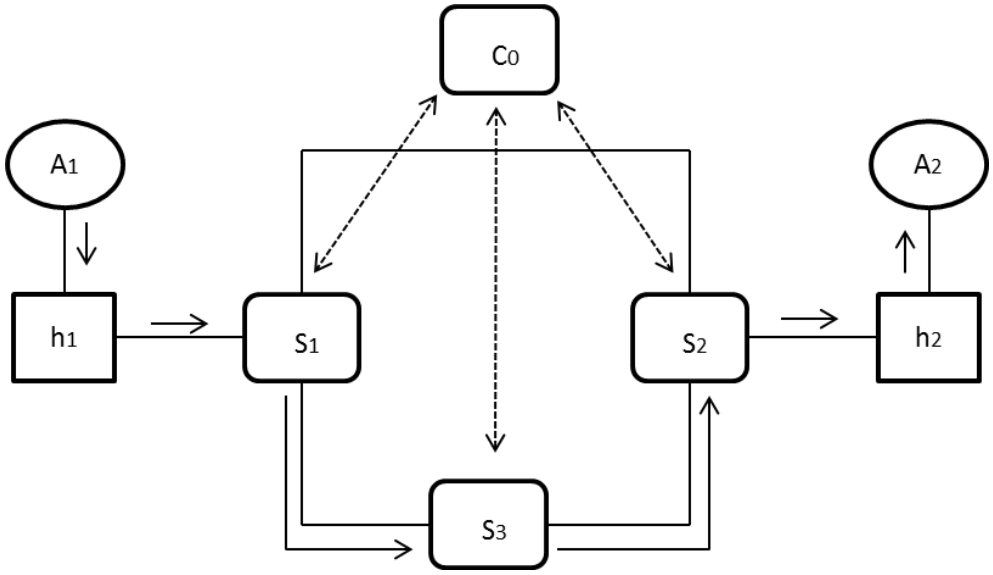
Fonte: Autoria própria (2017).

Supondo-se que os equipamentos de redes S_1, S_2 e S_3 sejam switches, também conhecido como comutador, é um equipamento cuja função é examinar os quadros de dados que chegam a cada uma de suas portas e decidir se estes estão endereçados para uma estação conectada em uma das portas. Um protocolo de roteamento dinâmico, que compartilha informações sobre alcançabilidade e estado das redes, dita que o tráfego entre as aplicações sempre seguirá o caminho mostrado – isto é, o tráfego sempre seguirá o caminho S_1-S_2 que é menor em detrimento do caminho alternativo, S_1-S_3-S_2. O paradigma SDN visa flexibilizar esta decisão.

As aplicações em uma rede são naturalmente dispersas, e elas utilizam simultaneamente os enlaces da rede, compartilhando sua largura de banda entre si. Não há diferenciação, por parte dos equipamentos de comutação, entre pacotes oriundos de uma determinada aplicação e pacotes das demais aplicações. Considerando esta afirmação, pode-se

imaginar na Figura 2 que a decisão tomada pelo protocolo de roteamento não seja a melhor possível em um cenário no qual o enlace S_1-S_2 esteja fortemente carregado pelo acúmulo de tráfego de outras aplicações não ilustradas na figura, enquanto o caminho S_1-S_3-S_2 esteja apto a transportar o tráfego de A_1 para A_2 com qualidade. É neste contexto, ilustrado pela Figura 3, que a SDN tem sua aplicação.

Figura 3 – A mesma rede operando de acordo com o paradigma SDN.



Fonte: Autoria própria (2017).

Na Figura 3, o Controlador (C_0) tem acesso, via uma rede sobreposta (*overlay*), aos equipamentos de comutação (genericamente denominados de switches) para alterar suas tabelas de encaminhamento (genericamente denominadas de tabelas de fluxo). Isto permite que o caminho S_1-S_3-S_2 seja escolhido quando o cenário expresso anteriormente se manifeste. É importante frisar que o tráfego entre A_1 e A_2 (denominado de fluxo) continua a seguir pelos enlaces da rede, misturando-se aos demais fluxos.

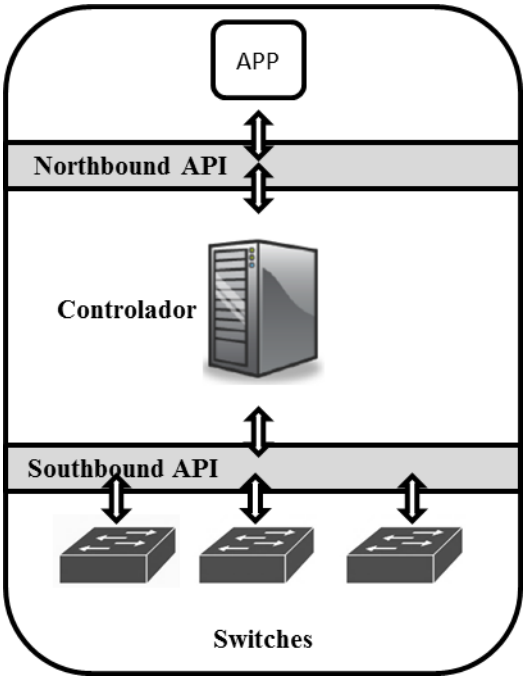
Em SDN, as interfaces de comunicação possuem denominações específicas. Enquanto a interface entre os planos de Aplicação (aplicação para gerenciar o controlador) e Controle é denominada de API Northbound, o princípio por trás da SDN é a capacidade de controlar o plano de encaminhamento de pacotes através de uma interface chamada de API Southbound.

Na API Southbound, um dos padrões definidos desde a origem do paradigma SDN é o OpenFlow que tem como principal objetivo permitir que se utilize equipamentos de comutação comerciais para pesquisa e experimentação de novos protocolos de rede sem prejudicar a operação normal das redes, por ser utilizado em uma rede sobreposta. Isso é

conseguido com a definição de uma interface de programação que permite ao desenvolvedor controlar diretamente os elementos de encaminhamento de pacotes presentes no dispositivo.
Há Controladores desenvolvidos em diversas linguagens, existem controladores em Java como o Maestro, que explora o paralelismo de uma máquina ao máximo para obter um maior desempenho do sistema de transferência entre o controlador e o switch, existe também na linguagem C++, como o Simples Controle de Acesso à Rede (SNAC – Simple Network Access Control), sua interface é baseada em uma ferramenta web que mostra os status e as características da rede. Outro controlador que tem como referência para as primeiras aplicações OpenFlow é o NOX que contém APIs desenvolvidas em C++ e em Python. Além do NOX, existe também o controlador Ryu. Ambos tem serviços que gerenciam as entradas de fluxo nos switches OpenFlow. Este último comporta as mais recentes versões do OpenFlow com linguagem em Python e ele foi o escolhido para ser utilizado neste trabalho.

É visto na Figura 4, uma arquitetura SDN em cujo centro se encontra um Controlador que na comunicação da parte de baixo controla os comportamentos dos elementos de encaminhamento de dados subjacentes. E na parte de cima, o Controlador fornece uma abstração das funções de rede com uma interface programável para a Aplicação consumir os serviços de rede e configurar a rede dinamicamente.

Figura 4 – Arquitetura SDN com suas APIs.



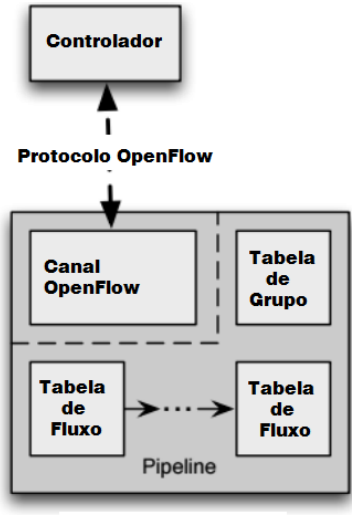
Fonte: Autoria própria (2017).

2.1 Openflow

Enquanto seja tecnicamente possível utilizar várias soluções para a comunicação entre o Controlador e os equipamentos de comutação – tais como os protocolos Protocolo Simples de gerenciamento de redes SNMP (*Simple Network Management Protocol*) e Configuração de Rede NETCONF (*Network Configuration*) – a solução mais aceita até o momento é o protocolo OpenFlow.

Na Figura 5, extraída da especificação mais recente deste protocolo, ilustra a arquitetura de um switch com suporte a OpenFlow. De acordo com (OPEN NETWORKING FOUNDATION, 2014), um switch OpenFlow possui uma tabela de grupo e uma ou mais tabelas de fluxo, utilizadas para análise e encaminhamento de pacotes. Através de um canal OpenFlow, um Controlador pode adicionar, remover ou atualizar entradas de fluxo em tabelas de fluxo.

Figura 5 – Arquitetura de um switch OpenFlow.

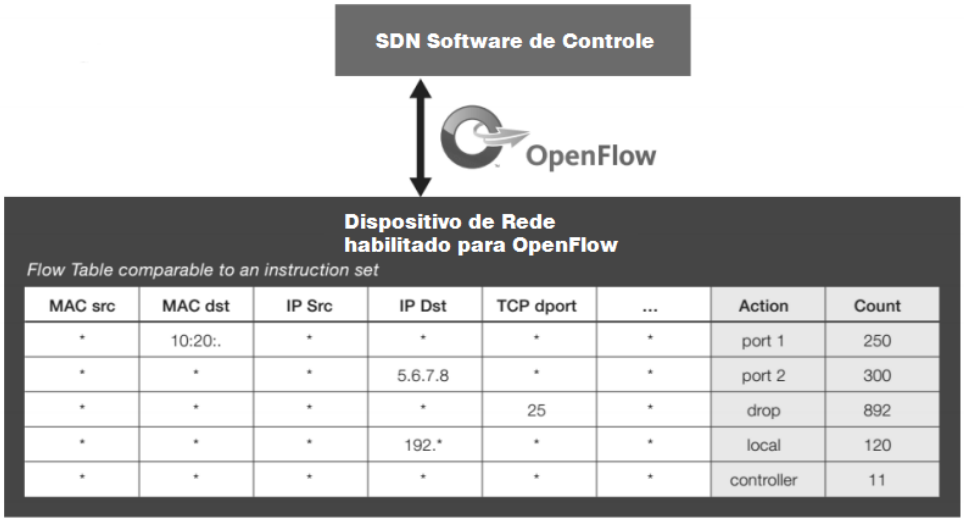


Fonte: Extraída de (OPEN NETWORKING FOUNDATION, 2014).

A Figura 6 ilustra um exemplo de tabela de fluxo com cinco entradas de fluxo. Cada entrada possui campos para correspondência (matching fields), definidos em termos de endereços de Controle de Acesso de mídia (MAC – *Media Access Control*), endereços Protocolo de Internet (IP – *Internet Protocol*), portas do Protocolo de Controle de Transmissão (TCP – *Transmission Control Protocol*), Protocolo de Datagrama do Usuário (UDP – *User Datagram Protocol*), etc. Uma vez que um pacote possua campos que correspondam aos da tabela, uma determinada ação é realizada – pode ser o caminhamento do pacote através de uma porta específica do switch, o envio do pacote para o Controlador ou

simplesmente o descarte do pacote, por exemplo. Sempre que uma correspondência for estabelecida, uma variável contadora é incrementada, para efeitos de estatística.

Figura 6 – Estrutura de uma tabela de fluxo.



MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:.	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Fonte: Extraída de (OPEN NETWORKING FOUNDATION, 2012).

2.1.1 Componentes da arquitetura Openflow

- Controlador

  É o principal componente de uma rede OpenFlow. Nele, são executadas as aplicações que gerenciam a rede, dispõe de todas as informações necessárias para determinar as funções desempenhadas pelos switches, implementando o plano de dados de uma rede SDN.

- Switch

  É o elemento responsável pelo encaminhamento dos pacotes pela rede. É nele que tem a tabela de fluxos que mantêm as informações de como serão processados os pacotes. É a infraestrutura de uma rede SDN.

- Canal Seguro

  É um canal que estabelece e finaliza as conexões entre o switch e o Controlador, usando configurações e procedimentos de conexão do tipo *Secure Socket Layer* (SSL) que confere segurança na comunicação entre Controlador e switch, permitindo autenticações mutuas através da troca de certificados, fornecendo uma chave privada.

2.1.2 Mensagens OpenFlow

O OpenFlow define várias mensagens que são trocadas entre o Controlador e um switch. Entre elas, está a mensagem *Packet_In*, enviada por um switch quando não há nenhuma correspondência entre um pacote recebido por uma de suas portas e as entradas na sua tabela de fluxo, transferindo o controle para o Controlador que especifica as medidas a serem tomadas. Outra mensagem de destaque é a *Packet_Out*, enviada pelo Controlador informando ao switch o que fazer imediatamente com um pacote. As operações sobre a tabela de fluxo de um switch (adição, remoção ou atualização de entradas de fluxo) ficam a cargo de mensagens *Flow_Mod*, enviadas pelo Controlador para aquele switch.

3 REST

O REST é um estilo arquitetônico para sistemas distribuídos criado por Roy Fielding na sua tese de doutorado (FIELDING, 2000). Fielding desenvolveu esta arquitetura de desenvolvimento orientado a serviços web através de vários outros estilos arquitetônicos para softwares de redes e adicionou outros requisitos para o seu REST.

Na prática, a especificação de uma API REST, baseia-se na utilização de Localizador de Recurso Uniforme (URL – *Uniform Resource Locator*) para definição de recursos e do Protocolo de Transferência de Hipertexto (HTTP – *Hypertext Transfer Protocol*) para realização de operações CRUD (Create, Read, Update, Delete) naqueles recursos em um servidor.

As características desse estilo são definidas através de requisitos aplicados dentro da sua arquitetura. Dentre desses requisitos está o uso de cliente-servidor em que os servidores mantêm recursos identificados por Identificador de Recurso Uniforme (URI – *Uniform Resource Identifier*), esta adoção proporciona um princípio de separação de preocupações, o cliente não precisa saber onde está os dados no servidor e nem como estão armazenados, o que determina uma melhora na portabilidade da interface do usuário.

Outro requisito é chamado de sem estado (*stateless)*, no qual o cliente-servidor deve ter uma comunicação sem estado, de forma que a cada solicitação do cliente para o servidor deve conter todas as informações necessárias para responder a requisição.

O estado da sessão é mantido somente no cliente, proporcionando algumas propriedades como a de visibilidade pelo fato do servidor não precisar olhar além dos dados da solicitação para responder, a confiabilidade por facilitar a tarefa de recuperação de falhas em um sistema distribuído e a escalabilidade devido o servidor não precisar armazenar os estados de cada sessão.

Na prática, a solicitação HTTP enviada por um cliente deve conter todas as informações de estado necessárias para a criação, leitura, atualização ou remoção de um recurso em um servidor junto com os dados necessários no corpo desta mensagem expressas em *JavaScript Object Notation* (JSON) ou *eXtensible Markup Language* (XML).

Não há regras para a definição de uma API REST, mas há boas práticas. Recomenda-se que recursos sejam nomeados de forma que façam sentido para o consumidor da API (os clientes), e que uma ação a ser realizada com base em um recurso seja definida pelo método HTTP utilizada na solicitação correspondente. Neste contexto, recomenda-se a utilização do método POST para a criação de um recurso e que não utilize verbos, mas nomes no plural

para este recurso, a utilização do método GET para obtenção das informações de estado daquele recurso, a utilização do método PUT para atualização daquelas informações de estado e a utilização do método DELETE para a remoção do recurso (FREDRICH, 2013).

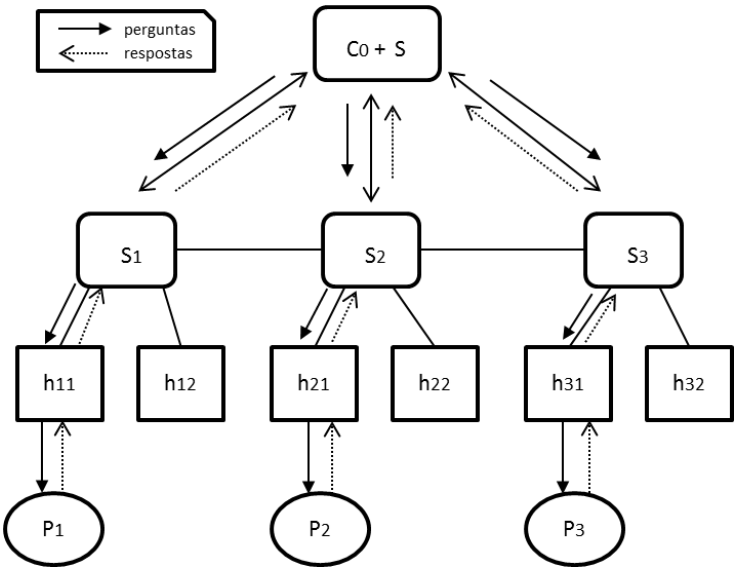
Em continuação dos requisitos definidos por Roy Fielding, para melhorar a eficiência da rede diminuindo as interações, exige outro requisito que é o cache, onde os dados dentro de uma resposta a uma solicitação precisam ser identificados de forma implícita ou explícita para avisar se tem ou não cache. Se uma resposta for armazenada em cache, o cache do cliente receberá o direito de reutilizar os dados de resposta para solicitações equivalentes posteriores.

E por fim, existe o sistema de camadas no qual cada componente não precisa saber além da camada que esteja interagindo para proporcionar ainda mais uma melhor escalabilidade. Assim, ao seguir todos estes requisitos a API REST passa a ser definida como API RESTful.

4 A FERRAMENTA DESENVOLVIDA

Neste trabalho, procura-se explorar a rede overlay entre o Controlador e os switches. Em uma aplicação distribuída do tipo Quiz, um jogo simples de perguntas e respostas. Assumindo como Servidor (S) do Quiz o Controlador, todo o tráfego relativo ao Quiz (tanto as perguntas quanto às respostas) segue através de rede overlay, assim este tráfego não tem influência alguma sobre o tráfego das demais aplicações. Toma-se como base a Figura 7.

Figura 7 – Quiz no cenário SDN com o Controlador assumindo função de Servidor.



Fonte: Autoria própria (2017).

4.1 Visão geral da arquitetura

A arquitetura geral deste trabalho é ilustrada pela Figura 8. Seus principais componentes são a Aplicação de gerência do Quiz (APP), o Controlador com função de Servidor e nos hosts a aplicação do Jogador, todos eles personalizados para o funcionamento do Quiz.

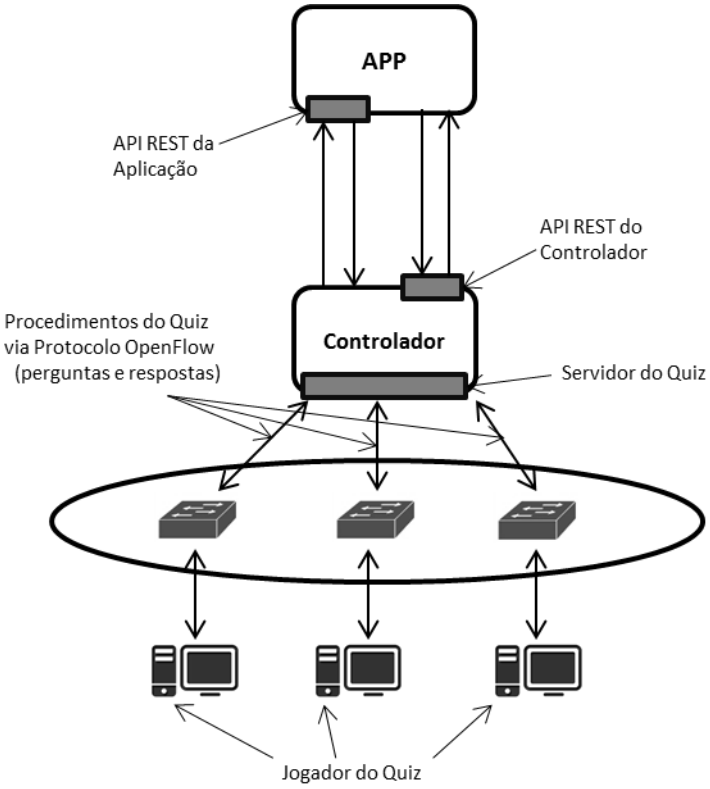
Na Aplicação de gerência é disponibilizada ao usuário uma interface gráfica que lhe permite configurar, inicializar e acompanhar o andamento do Quiz. No Controlador além da sua atribuição normal de preencher as tabelas de fluxos dos switches para que eles realizem as comutações dos pacotes corretamente, ele é capaz de se tornar um Servidor para o jogo.

Este por sua vez invocado pela Aplicação de como serviria aos jogadores na determinação do assunto, quantidade das perguntas e número de jogadores necessário para

inicialização. E nos hosts, o Jogador que realiza a requisição para participar do jogo e assim permitir receber as perguntas do Servidor (Controlador) e enviar as suas respostas de volta.

As APIs REST disponibilizadas pelo Controlador e pela Aplicação, bem como os procedimentos executados pelo Controlador e Jogador para realizar o Quiz, são descritos pelas seções seguintes.

Figura 8 – Visão geral da arquitetura.

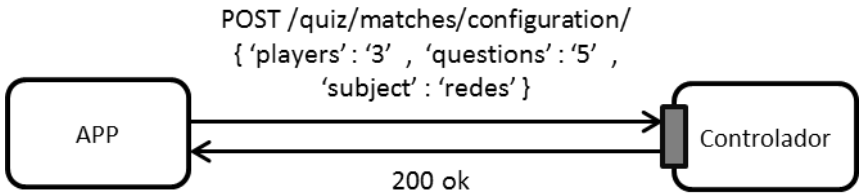


Fonte: Autoria própria (2017).

4.2 API REST do Controlador

Na API disponibilizada pelo Controlador, uma partida do Quiz só pode ser iniciada quando a Aplicação envia uma mensagem HTTP com o método POST com a sua composição ilustrada na Figura 9, no qual este método é usado para criar um recurso no Controlador chamado *match* (partida) definido pelo URL enviado por este método junto com seus parâmetros em formato JSON que definirá como será o jogo, determinando a quantidade de jogadores, questões e qual assunto será abordado.

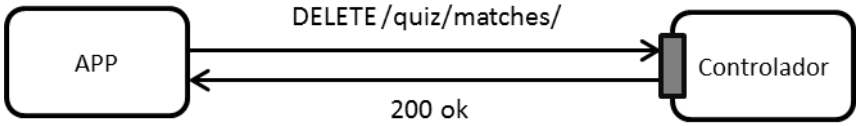
Figura 9 – API disponibilizada pelo Controlador – Método para criação do Quiz.



Fonte: Autoria própria (2017).

A Aplicação pode enviar uma mensagem HTTP com o método DELETE para finalizar a partida caso já se tenha um vencedor (pontuação não mais alcançada pelos demais jogadores) ou por outra razão não queira mais continuar, então ele irá excluir o recurso da partida servido pelo Controlador e encerrando o Quiz, definido na Figura 10.

Figura 10 – API disponibilizada pelo Controlador – Método para a remoção do Quiz.

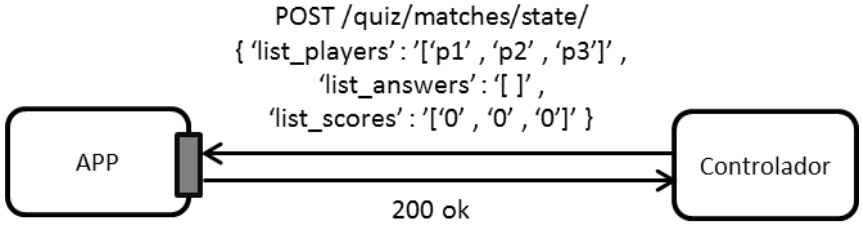


Fonte: Autoria própria (2017).

4.3 API REST da Aplicação

Após o Controlador criar o recurso da partida, ele então começa a receber as solicitações de participação dos jogadores e ao preencher a quantidade de jogadores para iniciar o jogo, é enviada uma mensagem HTTP com o método POST do Controlador para a Aplicação de gerência que cria um recurso *state* para armazenar a lista de jogadores com as suas identificações, sua lista de respostas vazias (por não serem iniciadas ainda as perguntas) e a lista de pontuação preenchida inicialmente em zeros para cada jogador, como mostra na Figura 11.

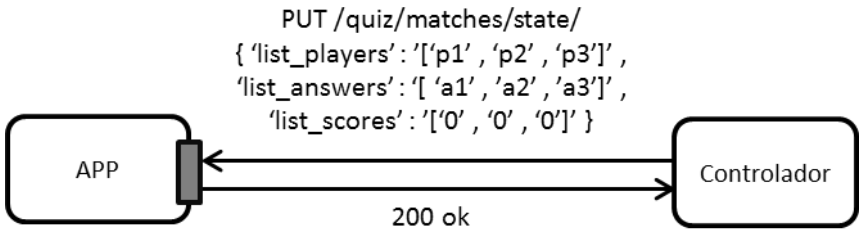
Figura 11 – API disponibilizada pela Aplicação – Método para criação dos status.



Fonte: Autoria própria (2017).

Ao cadastrar os jogadores na Aplicação por meio do recurso criado na figura anterior, o Controlador começa a enviar atualizações a cada rodada de questões com os valores determinados no cadastro que seriam as respostas dadas pelos jogadores nessa rodada e a pontuação atualizada caso tenham acertado a questão. Essa atualização é enviada pelo método PUT como mostra na Figura 12.

Figura 12 – API disponibilizada pela Aplicação – Método para atualização dos status.

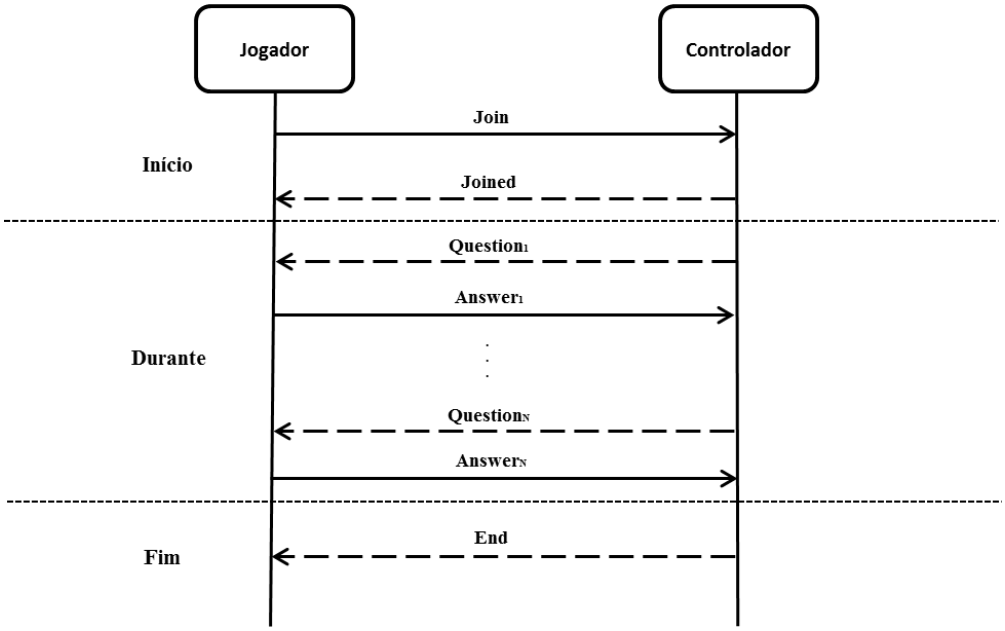


Fonte: Autoria própria (2017).

4.4 Procedimentos do Quiz – Controlador e Jogador

Após a aplicação enviar as configurações da partida, o Controlador passa a assumir também a função de Servidor. Nele, além de receber pacotes direcionados dos switches para preenchimento das tabelas de fluxo na sua atribuição normal de Controlador, ele passa a observar os que apresentam o endereço ip de destino igual a 224.0.0.10 (Servidor), os quais são de origem dos jogadores. Ao perceber que o destino é para o Servidor, do pacote é extraído a mensagem contida nele e é feita a interpretação dessa mensagem com base no Protocolo definido entre Controlador (Servidor) e Jogador. Pode ser visto na Figura 13 a sequência das mensagens entre eles durante o jogo.

Figura 13 – Sequência das mensagens entre Servidor e Jogador durante o Quiz.



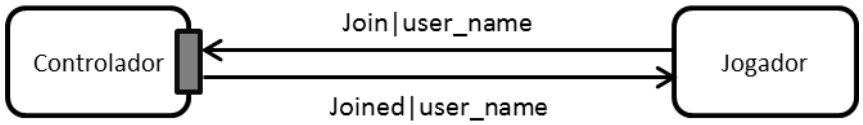
Fonte: Autoria própria (2017).

Estas mensagens tratam-se de um protocolo modo texto, em que o tipo de Unidade de Dados de Protocolo (PDU) é definido por uma palavra "mágica" e os campos delimitados por um caractere especial. Este caractere é a barra vertical '|', permitindo ler os campos separadamente. A construção dessas mensagens é explicada nos parágrafos seguintes.

Estas mensagens são encapsuladas em datagramas *multicast*. As mensagens dos jogadores são endereçadas com o ip determinado para o Servidor e as respostas enviadas do Controlador são encaminhadas utilizando o endereço de origem dos jogadores (observando qual o Switch e porta). Todas as PDUs com o destino do Controlador que chegam ao Switch são encapsuladas por mensagem OpenFlow do tipo *Packet_In* e enviadas por um canal seguro para o destino do Controlador (Servidor), no sentido contrário (Controlador – Switch), a PDU é encapsulada por mensagem OpenFlow do tipo *Packet_Out*.

Na Figura 14, o Jogador pede para participar de uma partida do Quiz enviando uma mensagem com a palavra "mágica" `Join` no primeiro campo (antes da primeira barra vertical) e no segundo campo o nome do jogador. É então retornada uma mensagem com o primeiro campo escrito `Joined` e no segundo o nome do jogador para confirmar a entrada deste ao Quiz.

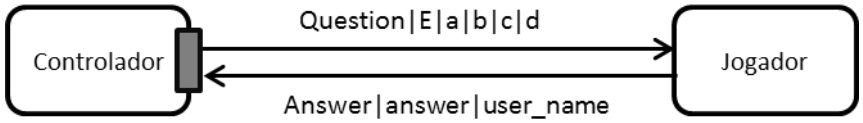
Figura 12 – Estrutura da mensagem de inicialização do Quiz.



Fonte: Autoria própria (2017).

Quando o número de jogadores que requisitaram a participação do Quiz for igual ao número de jogadores determinado pela Aplicação, é então iniciada a partida e assim o Controlador envia a primeira pergunta com a estrutura da mensagem no primeiro campo a palavra `Question`, no segundo o enunciado da questão e do terceiro ao sexto campo as alternativas. O Jogador tem até 30 (trinta) segundos para responder a questão sem que seja considerada como errada e enviada à próxima pergunta. Caso ele responda, sua resposta é criada com o nome `Answer` definido no primeiro campo e a resposta definida pelo jogador no segundo campo e no terceiro campo o nome do jogador que respondeu. Estas mensagens podem ser vistas na Figura 15.

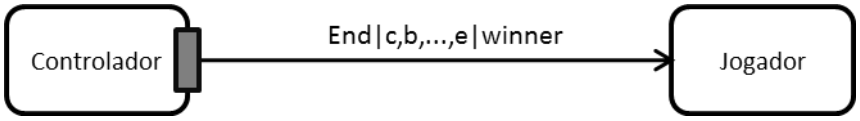
Figura 15 – Estrutura da mensagem de envio de questões e da resposta do jogador.



Fonte: Autoria própria (2017).

Ao terminar o envio de todas as questões do Controlador para o Jogador determinadas pela Aplicação é enviado uma mensagem com o nome de `End` no primeiro campo, as alternativas corretas das questões no segundo campo e por fim o nome do jogador ganhador para todos os Jogadores determinando o fim do Quiz, conforme na Figura 16.

Figura 16 – Estrutura da mensagem de finalização do Quiz para todos os Jogadores.



Fonte: Autoria própria (2017).

4.5 Aspectos de Implementação

Para a criação do quiz multiusuário baseado em um cenário de Redes Definidas por Software é necessário ter os componentes da arquitetura OpenFlow. Inicialmente é preciso ter uma máquina controladora (Controlador) que em uma analogia direta das funcionalidades de um sistema operacional, esse elemento age como um sistema operacional para a rede: provendo o controle direto dos dispositivos de comutação. Também é necessário ter elementos de comutação que tenham plano de controle separado do plano de dados que seriam os switches OpenFlow e uma estrutura física que interligue todos esses componentes por um canal seguro.

O problema é que a criação desse cenário possui um custo elevado. Para resolver esse problema é possível usar um simulador de Redes Definidas por Software. Este simulador seria o MiniNet que é um emulador de rede que permite a rápida prototipação de uma grande infraestrutura virtual de rede com a utilização de um único kernel do Linux. O MiniNet usa uma virtualização para fazer com que um único sistema pareça uma rede completa.

Uma característica deste emulador é fornecer uma maneira simples e barata para a realização de testes em redes para desenvolvimento de aplicações OpenFlow, por ser uma virtualização leve, permite o teste de uma topologia grande e complexa sem a necessidade de uma rede física.

Outra característica do MiniNet é oferecer APIs simples em Python para criação e experimentação de redes. E para o cenário deste trabalho foi criada uma topologia personalizada por meio da implementação de uma API oferecida pelo MiniNet. Nesta API, foi implementada em Python com a criação de seis hosts, três switches OpenFlow e um controlador, no qual a cada dois hosts são ligados a um dos três switches, estes, ligados entre si e todos com uma ligação ao o único controlador.

Dentre os vários controladores disponíveis, o controlador escolhido para a ferramenta foi o Ryu que se trata de um controlador gratuito. O Controlador Ryu segundo a RYU SDN FRAMEWORK COMMUNITY em 2014, é baseado na linguagem Python e suporta as versões mais recentes do OpenFlow, além de oferecer uma API Northbound RESTful que seria uma API REST que segue todos os requisitos determinados no capítulo do REST.

O Ryu é executado com base em um ou mais módulos Python fornecidos em tempo de execução. Um destes módulos faz com que os switches gerenciados aprendam a encaminhar quadros da mesma forma que switches ordinários.

Foi então modificado este módulo escrito também na linguagem Python com o objetivo de criar um Servidor do Quiz dentro do Controlador Ryu e continuar exercendo a sua função normal de gerenciar os switches. Nele, há um método que é invocado quando recebe um evento *packetIn* e dentro dele foi implementado a função do Servidor.

Quando o pacote é recebido no Controlador, é feita uma leitura dele através da biblioteca de pacotes do Ryu para verificar se o destino dele é para o Servidor como foi dito na seção anterior. Foi criado um arquivo na linguagem Python dentro desta biblioteca de pacotes do Ryu para aceitar envio de pacotes com dados de tamanhos maiores que seria preciso para enviar as questões e suas alternativas neste trabalho.

Se o pacote tiver o destino do Servidor, então dentro dele são retirados os dados para verificar qual tipo de mensagem (`Join` ou `Answer`), do qual já explicado as ações para cada tipo de mensagem.

As mensagens entre o Controlador e Switch, são realizadas por meio do protocolo OpenFlow versão 1.3 que foi escolhida para o Controlador Ryu por ser a versão mais recente que tem suporte para os switches.

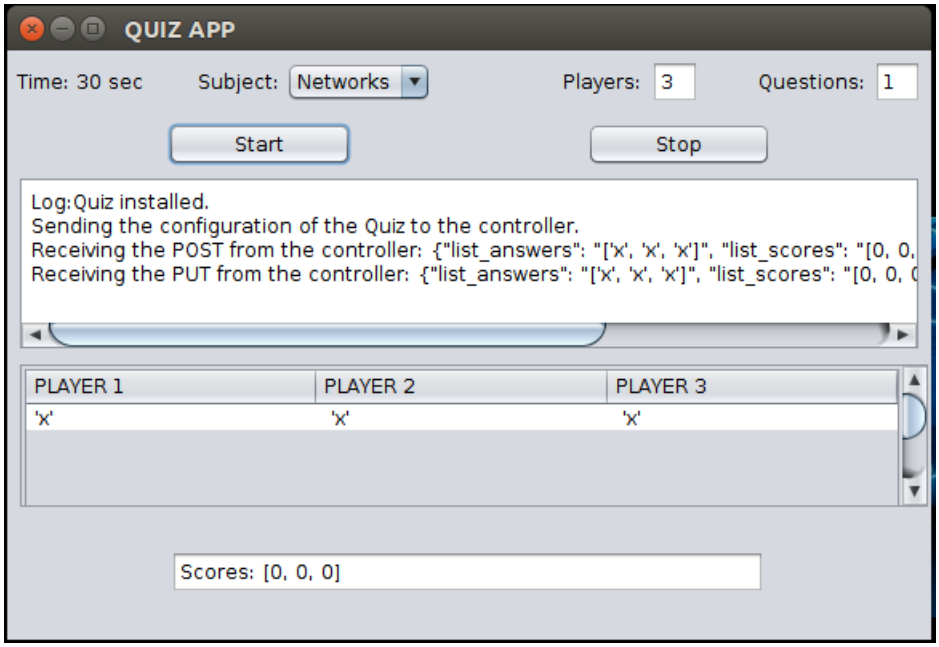
Já na implementação do Jogador que também foi utilizada a linguagem Python, suas mensagens são enviadas por meio de sockets. É criado um socket com o protocolo UDP no qual determina o caminho das mensagens do tipo `Join` ou `Answer` que são enviadas do Jogador para o Controlador.

Ainda na aplicação do Jogador, após ter o envio de pedido de entrada para o Quiz, é criada uma thread para ficar recebendo as respostas do Servidor e nela é implementada o tratamento das mensagens seguindo o Protocolo definido para cada tipo de mensagem (`Joined`, `Question` e `End`).

Visto a implementação da comunicação entre dois dos principais componentes deste trabalho, a próxima é a comunicação entre o Controlador e a Aplicação que é responsável por exercer a função de administradora do Controlador e por consequência o jogo do Quiz.

Pode ser visto na Figura 17, a interface gráfica da Aplicação que foi desenvolvida com a linguagem Java através da IDE Netbeans e utilizando a classe do *javax.swing*, a *JFrame*. E para a sua comunicação com o Controlador foram criadas duas classes, uma para enviar mensagens chamada de *ClientRestAPP* e a outra para receber as mensagens chamada *ServerRestAPP*.

Figura 17 – Janela da Aplicação.



Fonte: Autoria própria (2017).

Na classe *ClienteRestAPP* foi importada do java.net a classe *httpURLConnection*, responsável por criar a conexão entre o aplicativo e o controlador por meio do URL. Esta é uma derivação da URI que utiliza o endereço para identificar e junto com o uso específico do protocolo HTTP, envia os métodos no formato REST para a API REST do Controlador.

Para a API REST do Controlador foi aproveitada uma função disponível pelo Controlador Ryu, uma função de servidor Web correspondente a WSGI (Web Server Gateway Interface) que é um framework unificado para conectar aplicativos web e servidores Web em Python. Foi modificado nessa função o formato das URL e o tratamento de cada um já explicado no seu próprio tópico anteriormente.

Na classe *ServerRestAPP* foi importada a classe *HttpServer* que é um servidor HTTP do Java. Ele é vinculado a um endereço ip local e a um número de porta definido para ficar na espera de solicitações HTTP recebidas do Controlador neste endereço. São criadas threads para todos os URL determinado pelas mensagens REST enviadas do Controlador, e neles são analisados os dados e feitos às ações necessárias já explicadas na API REST da Aplicação.

5 CONSIDERAÇÕES FINAIS

O paradigma de Redes Definidas por Software é uma grande promessa para atender as diversas aplicações de redes futuras. Mas seu potencial já esta começando a ser explorado e já possui grande interesse da área acadêmica e industrial. Já existe no mercado diversos dispositivos comerciais com o protocolo OpenFlow habilitado, o que confirma que esse paradigma seja muito promissor.

Neste trabalho, foi observado que, por ser numa arquitetura de rede programável, os pacotes de uma partida do Quiz foram encaminhados diretamente para o Controlador, diminuindo a quantidade de fluxos do jogo e não misturando com a de outras aplicações, por não precisar repassá-los para outros Switches com o objetivo de localizar o servidor já que todos eles foram configurados pelo Controlador. Os Switches já enviam os pacotes diretamente para o Controlador e assim não precisaria ter a responsabilidade de analisar para onde devem enviá-los, direcionando imediatamente para o mesmo por uma rede sobreposta (overlay).

No Controlador, as decisões do que fazer com os pacotes do jogo também já foi configurado por uma aplicação através da API REST, deixando as decisões nas mãos do usuário da aplicação, o Controlador pode agir normalmente como um gerenciador das tabelas de fluxo dos switches conectados a ele.

Assim, quando o usuário for desejar que o Controlador também assuma o papel de servidor, as configurações que ele determinar para a iniciação do servidor Quiz são enviadas através da aplicação. Permitindo que a tomada de decisões seja de um usuário no momento que ele determinar e não de máquinas já configuradas. Realizando o principal benefício deste projeto, diminuição do caminho dos pacotes do jogo e as decisões feitas pelo usuário dono da aplicação de gerência.

REFERÊNCIAS

CHOWDHURY, NM Mosharaf Kabir; BOUTABA, Raouf. Network virtualization: state of the art and research challenges. **IEEE Communications magazine**, v. 47, n. 7, 2009.

MCKEOWN, Nick et al. OpenFlow: enabling innovation in campus networks. **ACM SIGCOMM Computer Communication Review**, v. 38, n. 2, p. 69-74, 2008.

SEZER, Sakir et al. Are we ready for SDN? Implementation challenges for software-defined networks. **IEEE Communications Magazine**, v. 51, n. 7, p. 36-43, 2013.

UNDERDAHL, B.; KINGHORN, G. **Software Defined Networking FOR DUMMIES.** Cisco Special Edition. New Jersey: John Wiley & Sons, Inc, 2015. 11 p.

COMER, D. E. **Computer Networks and Internets.** 6 ed. Pearson Education, Inc, 2015. 3 p.

SDX CENTRAL. **WHAT'S SOFTWARE-DEFINED NETWORKING (SDN)? DEFINITION.** SDN Resources, 2014. Disponível em: <https://www.sdxcentral.com/sdn/definitions/what-the-definition-of-software-defined-networking-sdn/ >. Acesso em 26 abr. 2017.

OPEN NETWORKING FOUNDATION. **Software-Defined Networking (SDN) Definition: What is SDN?.** 13 de abril de 2012. Disponível em: <https://www.opennetworking.org/sdn-resources/sdn-definition >. Acesso em 26 abr. 2017.

FIELDING, Roy Thomas. **Architectural styles and the design of network-based software architectures**. 2000. Tese de Doutorado. University of California, Irvine.

FREDRICH, T. **RESTful Service Best Practices – Recommendations for Creating Web Services.** 2013. Disponível em: < http://www.restapitutorial.com>. Acesso em 26 abr. 2017.

OPEN NETWORKING FOUNDATION. **OpenFlow switch specification – version 1.3.4 (protocol version 0x04).** 27 de março de 2014.