COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Optimal time route planning in dynamic wireless software defined networks

Autor
Álvaro del Águila Martos

Directora
Dr. Janise McNair

Madrid
2020

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Optimal time route planning in dynamic wireless software defined networks

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2019/20 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Álvaro del Águila Fecha: 09/Julio/ 2020

Álvaro

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Dr. Janise McNair Fecha: 09/Julio/2020

[illegible]

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Optimal time route planning in dynamic wireless software defined networks

Autor
Álvaro del Águila Martos

Directora
Dr. Janise McNair

Madrid
2020

Acknowledgements

I would like to thank every person that has walked with me during the path of my career these past 4 years. To my professors for giving me an excellent education with professional knowledge and life values, and specially to Allen Starke and Dr. Janise McNair for their help and support during this project.

I would like to give special thanks to my family for making all of this possible and always believing in me, and to my friends who made these past years so special to me.

Tiempo de planificación de ruta óptimo en redes de software definidas dinámicas e inalámbricas

Autor: Álvaro del Águila Martos
Directora: Dr. Janise McNair

Palabras clave: SDN, ONOS, Mininet-WiFi, QoS

Resumen

Este proyecto presenta el desarrollo de un controlador para las redes definidas por software (SDN) utilizando el controlador de ONOS. Se centra en los protocolos de Reactive Forwarding y Segment Routing, modificando la aplicaión de Reactive Forwarding, para asegurar una calidad de servicio (QoS) óptima en una red de software definida dinámica e inalámbrica que siempre se cumpla.

Introducción

La red definida por software (SDN) es una arquitectura de red emergente que está ganando importancia y peso en la actualidad. Es una tecnología de red eficiente que cumple las necesidaes de funcionalidad dinámica de nuestras futuras redes con el aspecto crítico de simplificar hardware con software, dando beneficios tales como la reducción de costes.[1] Las SDN diferencian y separan el plano de control del plano de datos del hardware que realiza el switching. El software es empleado para realizar las funciones del plano de control, reduciendo el uso de hardware y sus requerimientos. Este software se llama controlador, el cual es centralizado y tiene una visión completa de la red. El controlador se construye a base de aplicaciones en código Java que pueden ser creadas por cualquier fuente externa o usuario. Este proyecto se centra en encontrar la mejor combinación de aplicaciones y su modificación, para obtener el controlador de ONOS que garantice una calidad de servicio constante en una red definida por software dinámica e inalámbrica.

Descripción del sistema

Todos los experimentos fueron realizados en una máquina virtual de Ubuntu, con 8GB de RAM y 2 CPUs, en un equipo con 16GB de RAM y un Intel(R)Core(TM)i7-3520M CPU @ 2.90GHz.

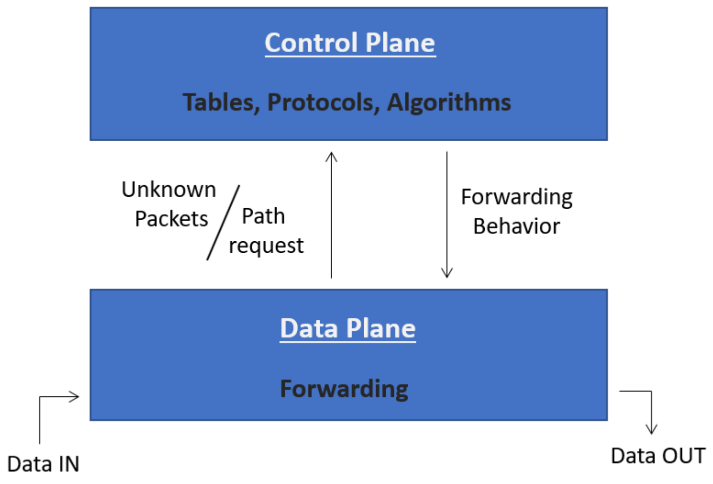


Figure 1: Plano de control y plano de datos en el que se basan las SDN

Los programas empleados fueron Mininet con su extensión Mininet-WiFi para crear la topología dinámica usando código en Python, ONOS para establecer el controlador con aplicaciones creadas en código Java, Wireshark para capturar el flujo de paquetes y evaluar los datos obtenidos, y HTOP para visualizar el uso de hardware.

Para ejecutar los experimentos, primero se inicializa el controlador de ONOS cargando todas sus aplicaciones. Después, una topología inalámbrica y dinámica se crea de manera virtual con Mininet-WiFi, que conecta sus puntos de acceso (AP) al controlador. El comando "pingall" se ejecuta, que hace que cada host virtual de la topología realice ping a cada uno de ellos, para así observar qué aplicaciones dan a la red una mayor conectividad, con menor pérdida de paquetes. Una vez elegidas las aplicaciones más eficientes, su código es evaluado, cambiado y testeado para obtener el controlador que elija la ruta más corta en términos del número de APs por los que la informacion fluye (hop count) al enviar los paquetes de información.

Resultados

Las aplicaciones que mostraron los mejores resultados y que fueron seleccionadas fueron las de Reactive Forwarding y Segment Routing. La aplicación de Reactive Forwarding se modificó para seleccionar la ruta más corta de entre todas las posibles para enviar los paquetes.

En función de la topología empleada, diferentes resultados con mayor o menor mejora fueron observados. Los experimentos finales fueron realizados en una topología de árbol ya que es la topología empleada en los campus universitar-

	Original	Modified
Max. tiempo para seleccionar una ruta	1ms	5ms
Media de p/seg del reactive processor	572.6	574.3
Uso máximo de CPU	84.4%	85.2%
Uso máximo de memoria RAM	3.61GB	4.06GB

Table 1: Comparación de datos durante el experimento final.

	Original	Modified
Coste = 1.0	10%	13%
Coste = 2.0	34%	42%
Coste = 3.0	34%	25%
Coste = 4.0	10%	12%
Coste = 5.0	12%	8%

Table 2: Coste de rutas durante el experimento final.

ios[2], con un área de 1.96km^2, 40 hosts, y 15 APs con un rango de alcance de 200m. Estas cifras son basadas en el campus de la Universidad de Florida[3], reduciendo las cantidades para así poder realizar la simulación, pero manteniendo los mismos rangos y relaciones para así realizar los análisis más realistas posibles.

Se observa una gran mejora en el coste de las rutas, siendo el coste el número de saltos que dan los packetes, o el número de APs por los que fluye la información para ir de origen a destino, seleccionando los más cortos. Sin embargo, esta reducción de coste requiere un incremento en pocos milisegundos en el tiempo de selección de la ruta, debido al cambio de código, y también un ligero incremento en el uso del hardware que además de deberse al cambio del algoritmo, también se daría por los programas ejecutados en segundo plano para realizar las mediciones necesarias para el análisis, que pudieron haber funcionado consumiendo más recursos.

Esto supone una gran mejora para las redes que tengan mayor dependencia de la ruta a seguir, como es el caso de las redes WiFi[4], ; suponinendo una disminución de latencia y mejorando la calidad de servicio (QoS), y futuros trabajos podrían evaluar los avances realizados en este proyecto en redes de mayor tamaño para así analizar cómo afectaría a la red.

Optimal time route planning in dynamic wireless software defined networks

Author: Álvaro del Águila Martos
Director: Dr. Janise McNair

Key words: SDN, ONOS, Mininet-WiFi, QoS

Abstract

This paper presents a SDN controller app development using an ONOS controller. It focuses on the Reactive Forwarding and Segment Routing protocols, updating the Reactive Forwarding application, to ensure that an optimal Quality of Service in a dynamic wireless software defined network is always met.

Introduction

As an emerging networking architecture, Software defined networking (SDN) is one of the most important new concepts that have been gaining weight these past years. It is an efficient networking technology that supports the dynamic nature functions of our future networks with the crucial aspect of simplifying hardware through software, with many benefits such as lowering costs. [1] It differentiates and separates the control plane and the data plane of the switching hardware. Using software for the control plane functions, it reduces hardware usage and requirements. That software is known as a controller, which is centralized and has a complete view of the network. The controller is built with applications that can be programmed by any external source or user in Java code. This paper is focused on finding the best combination of applications and their modification, to build an ONOS controller that would ensure that the Quality of Service in a dynamic wireless software defined network is always met.

System Description

All the experiments were run in an Ubuntu Virtual Machine with 8GB of RAM and 2 CPUs, in a computer with 16GB of RAM and an Intel(R)Core(TM)i7-3520M CPU @ 2.90GHz.

The programs used were Mininet with its Mininet-WiFi extension to create the dynamic topology with python code, ONOS to establish the controller with

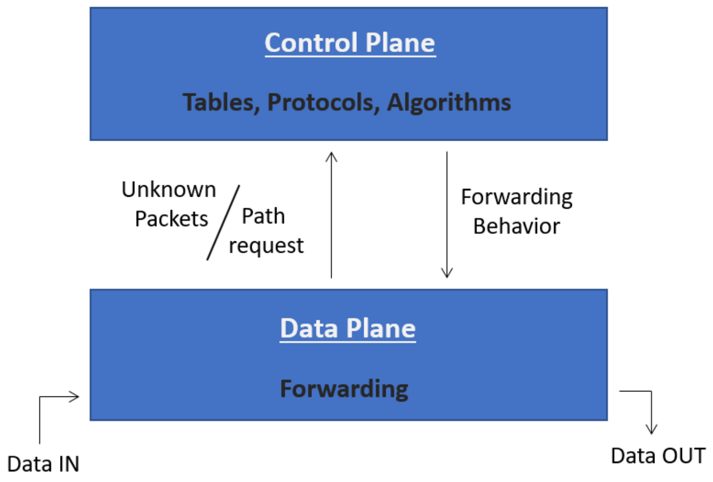


Figure 2: Control plane and data plane schemes regarding SDN

java code applications, Wireshark to capture the packets and evaluate the data obtained, and HTOP to record hardware usage.

To run the experiments, first the ONOS controller was initiated and had all its applications loaded. Then, a wireless dynamic topology was virtually created using Mininet-WiFi, that would connect its Access Points to the controller. The command ”pingall” was made, which would make every virtualized host do a ping to each other to discover which applications gave the network a better connectivity with less packet loss. After selecting the most efficient applications, their code was evaluated, changed and tested to have the controller choose the shortest possible paths in terms of AP hop count when sending packages of information.

Results

The applications that showed the best results and that were selected were the Reactive Forwarding app and the Segment Routing app. The Reactive Forwarding application was modified to select the shortest path from the possible ones.

Different results with more/less improvements were observed while running tests on different topologies. The final tests were run on a tree topology with 40 hosts, 15 access points with a range of 200m, in an area of 1.96km^2, since that is the perfect topology for a university's campus[2]. The tests were based on the University of Florida's campus[3], bringing the numbers down to be able to simulate it, but keeping the same ratios and trying to be as realistic as possible.

A noticeable improvement can be seen in the path costs, being the number of APs the packages had to go through from the source to their destination. However,

	Original	Modified
Max. time to select a route	1ms	5ms
Average reactive processor p/sec	572.6	574.3
Max. CPU usage	84.4%	85.2%
Max. RAM memory	3.61GB	4.06GB

Table 3: Data comparison during the final experiment.

	Original	Modified
Cost = 1.0	10%	13%
Cost = 2.0	34%	42%
Cost = 3.0	34%	25%
Cost = 4.0	10%	12%
Cost = 5.0	12%	8%

Table 4: Path costs during the final experiment.

there is a slight increase in the time to choose the path due to the algorithm modification and in the hardware usage not just because of the algorithm modification but also due to the programs used to record the data that could be running less efficiently.

This is a major improvement in networks that rely on the path to choose, like WiFi based networks[4], reducing its latency and improving its quality of service (QoS), and future works should test it on bigger networks to evaluate how it would react.

Bibliography

[1] "[Yuanguo Bi, Guangjie Han, Chuan Lin, Yan Peng, Huayan Pu and Yazhou Jia] Intelligent Quality of Service Aware Traffic Forwarding for Software-Defined Networking/Open Shortest Path First Hybrid Industrial Internet". In: *IEEE Transactions on Industrial Informatics* 16.2 (February 2020).

[2] *Campus Topology.* URL: `https://www.conceptdraw.com/examples/what-kind-of-topology-used-in-college`.

[3] *UF campus wireless network.* URL: `https://net-services.ufl.edu/provided-services/wireless/`.

[4] *The latency of a WiFi network.* URL: `https://www.quora.com/What-is-the-average-latency-of-a-WiFi-network`.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Defining SDN and its current state

Moore's Law states that every two years, the amount of transistors in a microprocessor doubles in number.

This law has not been far from reality, and our society has experienced one of the fastest technology growths this past decade, affecting specially the telecommunications field. New technologies have been created such as the Internet of Things(IoT) and Big Data.

As new technologies develop, the coordination and management of large numbers of devices will both be a requirement and a significant challenge. From Internet of Things (IoT) devices to smart phones, tablets and laptops, to Unmanned Air Vehicles (UAVs), to Small Satellites. Interoperability and on-demand communication paths require an adaptive approach to network management techniques that can be employed to create a network management architecture.

As an emerging networking architecture, Software Defined Networking (SDN) is one of the most important new concepts that have been gaining weight these past years. It is an efficient networking technology that supports the dynamic nature functions of our future networks with the crucial aspect of simplifying hardware through software, with many benefits such as lowering costs. [1] It aims to make the networks agile and flexible. The main characteristics that differentiate SDN from any other network layer architectures are:

- Differentiation and separation of the control plane and the data plane of the switching hardware.

  The control plane makes decisions about how the packets travel around the network, and the data plane moves the packets from host to host.

- Using software for the control plane functions, reducing hardware usage and requirements. That software is known as a controller, which is centralized and has a complete view of the network.

  When a packet arrives at a data plane device (a network switch or access point), it gives information about that traffic to the controller, and that centralized controller sends rules to the data plane device to handle the packet.

  The controller would act as the "brain" in a SDN.

- Open interfaces between the controllers and the devices located in the data plane.

- Customization of the controller through applications that can be programmed by any external source or user; which is what this paper is focused on.

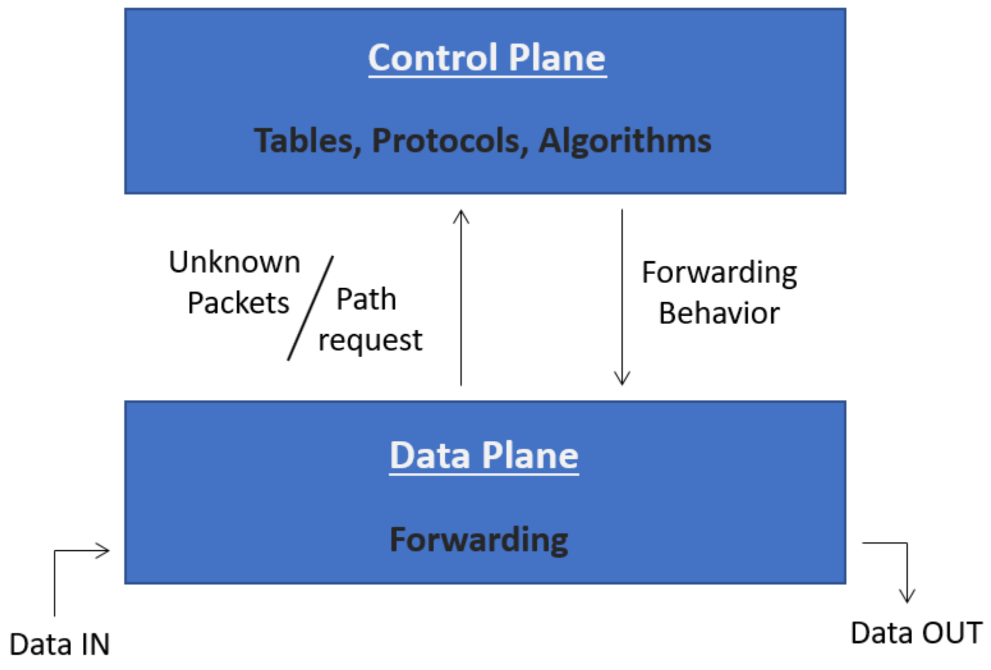


Figure 1.1: Control plane and data plane schemes regarding SDN

In a software-defined network, the administrator can configure traffic from a centralized control console without having to modify any individual switches or access points in the network. [2]

This process introduces a major improvement from traditional network architectures, where individual network devices direct the traffic based on their own configured routing tables.

This central controller presents a single point of failure, which can be seen as a security benefit, but also a concern because if it was targeted by an attacker, it could be fatal to the network. It has more targeted protection and simplifies its firewall.

SDNs would be beneficial in many scenarios; for example in a campus network where there is a need to unify WiFi and Ethernet networks. This paper will dig more into this specific scenario.

Due to economy and compatibility reasons, having a full deployment of SDN in just one step would not be a realistic approach, but with an increasing speed, it will play an important role in our society in the near future. [3]

As a revolutionary concept, SDN is evolving in a fast pace. There are many applications for different controllers that give different features and capabilities to the network, with many possibilities for development and improvement.[4]

1.2 Objectives of the project

This project is focused on dynamic wireless software defined networks, to ensure an optimal and constant quality of service using the Open Network Operating System (ONOS) controller.

This project will evaluate all the options available to develop the optimal controller:

- Finding the most suitable applications to build the best ONOS controller for this topology characteristics.
- Revising and modifying their code to improve their efficiency and the QoS in the network.

Establishing a working virtual machine capable of creating functional virtual networks, with all the programs required, and learning how to use them all combined, will be the first step to complete the previous stated objectives.

1.3 Motivation

As stated previously, this is a project focused in the future of the telecommunications field that brings a dynamic aspect to the network.

It is exciting to be part of a project that works on our future. Researching about it and realizing its potential and how beneficial it is, before it being very popular or well known. That gives a lot of motivation.

It brings more flexibility to configure the network, and improves the efficiency of the network. It would also decrease the need of hardware and as a consequence, it would lower costs. This new technology will bring many new possibilities for our future.

Chapter 2

Background Information

2.1 Quality of Service

The Quality of Service QoS are a set of requirements that must be met in order to have a proper network functionality and performance. [5]

There are different parameters that can measure the QoS of a network:

- Packet loss: When packages are dropped and not received to its destination.
- Latency: It is the time a packet takes to travel from the source to the destination. It should be as close to zero as possible.
- Jitter: Congestion of the network, making the network slower.
- Bandwidth: It is the capacity of a network's link to send the maximum amount of data from one point to another in a determined time frame.
- Availability: Amount of time that the network is fully operating.
- Resilience: Capacity to recover in case a network failure occurs.

This project will be focused first in finding the applications that build the controller that provides the network with less packet loss, and then a modification of those to get as less latency as possible.

2.1.1 Other definitions

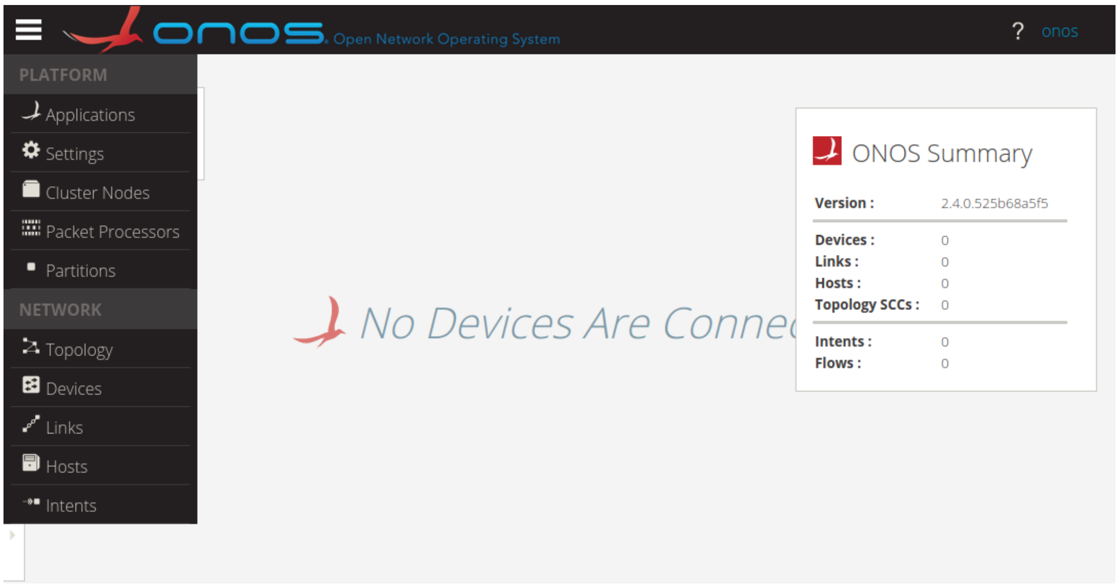
- Switch: A hardware device that filters and forwards network packets.
- Access Point: Device that creates a wireless local area network, and transmits data, connected to the controller with cables.
- Host: Device connected to the network that can send and receive information.
- Link: A connection between two network devices.
- Virtual Machine (VM): A virtual machine (VM) is a virtual environment that functions as a virtual computer system with its own CPU, memory, network interface, and storage, created on a physical hardware system (located off- or on-premises). Software called a hypervisor separates the machine's resources from the hardware and provisions them appropriately so they can be used by the VM. [6]
- Hardware: Every physical electronic device. [7]
- Software: Programs running on the software. For example an operating system like Windows.
- Firmware: Software semi-permanently placed in the hardware. It does not "disappear when the hardware is turned off and to change it a special tool or installation has to be used. For example the BIOS in a computer.
- Whitelist: In this scenario, a set of paths that have a priority to be used when possible. Those paths are known to be fast and reliable.[8]
- Blacklist: Set of paths that are not allowed to be used. In this project's scenario; it would be paths that are known to fail.
- Ping: It is a basic tool that allows the user to know if an IP address exists and can accept requests. Ping works by sending an Internet Control Message Protocol (ICMP) Echo Request to a specified host on the network and waiting for a reply. It can be used to test connectivity and determine response time. The ping size in this project is 56 bytes.[9]

2.2 The ONOS Controller

The controller used for this paper is the ONOS controller; which stands for Open Network Operating System.[10]

It is a modular software that combining independent and different applications and protocols to build a custom controller for a SDN, using Java code. The version of ONOS installed is the developers edition, which lets you access the source code and allows you to modify it.

It offers a graphical user interface (GUI) from where the controller can be modified, the topology of the network that it is connected can be seen and analyzed, as well as showing real time information of the network's traffic from every networking element, which will be used in this project.



ONOS Summary

Version :	2.4.0.525b68a5f5
Devices :	0
Links :	0
Hosts :	0
Topology SCCs :	0
Intents :	0
Flows :	0

Figure 2.1: ONOS GUI.

2.2.1 Open-Flow

As mentioned previously, the SDN controller is built with different applications written in Java code. The most important one is the OpenFlow protocol; which gives access to the forwarding plane of the network data devices such as a switch, and allows the controller to determine the route of the packets across a network. It is in charge of connecting the controller with the data plane.

2.3 Mininet-WiFi

Mininet is a SDN network emulator that simulates network devices such as virtual hosts, switches, links and adds controllers, that run with the Linux network software.[11]

Mininet-WiFi is an extension that adds WiFi stations and Access Points (AP) based on wireless Linux drivers, which perfectly suits the wireless dynamic SDN network needs that are demanded for this project. It supports OpenFlow, and it can get connected to the ONOS controller, which is the controller used for this set up.

Mininet-WiFi can run scripts programmed on Python, which gives freedom to create different topologies for different tests and purposes. For the topic of this paper, Mininet-WiFi gives the tools to create your custom dynamic SDN network, adding as many hosts and APs as needed, with a defined range, and mobility with different speeds and paths that can be programmed. That can provide the perfect set up since the topology can experience different scenarios such as hosts getting out of reach, or conflicts of hosts being in range with more than one AP.

In the next figure, a dynamic topology can be observed during different time frames with its range. The numbers 0 to 39, represent the hosts with their range in green colour; and from 40 to 45, access points with their range represented in blue colour.

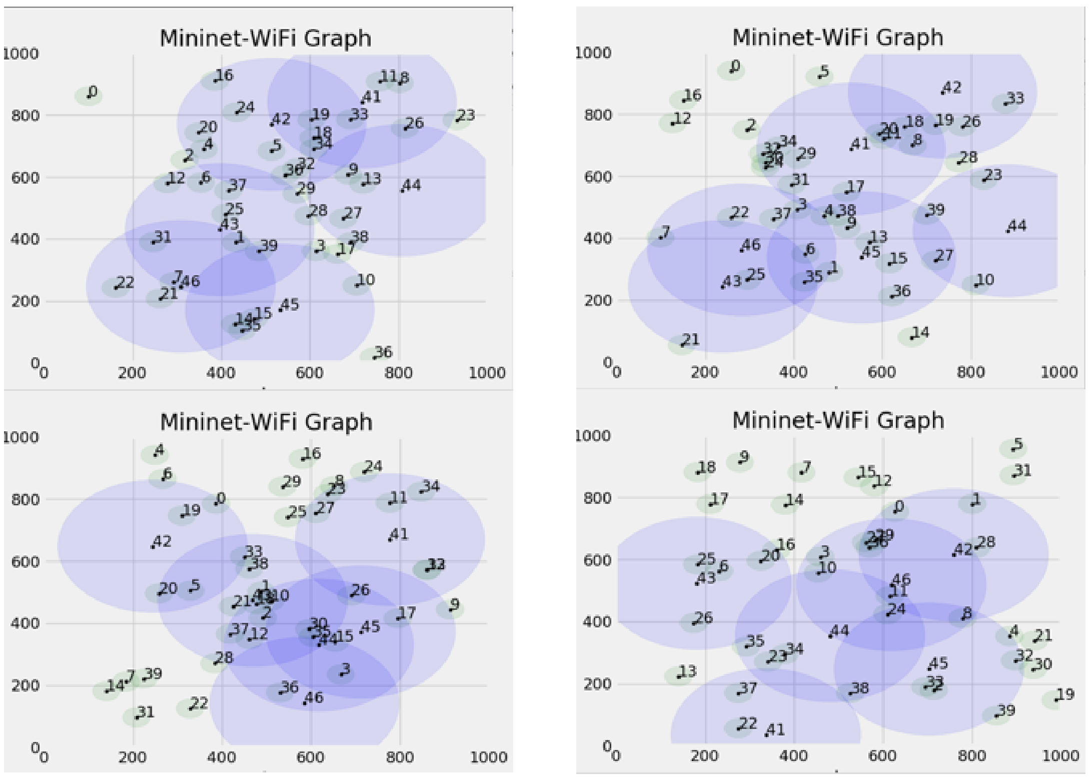


Figure 2.2: Example of a dynamic topology run in Mininet-WiFi, with 40 hosts and 6 APs.

2.4 Resources

All the experiments were run in an Ubuntu 18.04 Virtual Machine with 8GB of RAM and 2CPUs, in a computer with 16GB of RAM and an Intel(R)Core(TM)i7-3520M CPU @ 2.90GHz.

The programs used were Mininet with its Mininet-WiFi extension to create the dynamic topology, ONOS to establish the controller and its GUI to get network information, Wireshark; a network packet analyzer, to capture the packets and evaluate the data obtained, and HTOP; an interactive system-monitor process-viewer and process-manager, to record the hardware's usage.

Chapter 3

Development guides

3.1 Installation guide

One of the most important processes was to get a fully working virtual machine capable of creating dynamic wireless software defined networks with Mininet-WiFi and using the ONOS controller. The virtual machine holder/creator used is the Oracle VM VirtualBox.

There is more than one option to install every program, with more than one optional extension as well. Those are open source programs that are not very extended or popular, with fewer coverage of problems, and Mininet is not in its final version at the moment, so some problems were found through the process of having a working VM with all the needed features.

Here is a tutorial on how to install Mininet-WiFi and ONOS developers edition in an Ubuntu 18.04 Virtual Machine:

Step0. Update and upgrade the VM:

- Start a new terminal and execute "sudo apt-get update" to check if thee are any updates in the firmware or in any software, registering them without making any installations.
- Execute "sudo apt-get upgrade", which downloads and installs the software updates if found in the previous update command.
- Go to Step1.

Step1. Install Linux networking software and drivers, necessary to operate Mininet correctly.

- Start a new terminal and execute "sudo apt install net-tools"

- Go to Step2.

Step2. Install Mininet:
The website used was "http://mininet.org/download/", following the second option.

- Start a new terminal and execute "sudo apt-get install git" to install GitHub; a repository hostig service.
- Execute "git clone git://github.com/mininet/mininet" to copy the Mininet installation files.
- Run "mininet/util/install.sh -a"
- Execute Step0 again to check that the software is in its lates version and then go to Step3.

Step3. Install Mininet-WiFi:
This is the website used for this step: "https://github.com/intrig-unicamp /mininet-wifi"

- Start a new terminal and execute "git clone https://github.com/intrig-unicamp/mininet-wifi"
- Execute "cd mininet-wifi"
- Execute "sudo util/install.sh -Wlnfv"
- Execute Step0 again to check if there are any software updates and then go to Step4.

Step4. Install the ONOS controller: This is the website used: "https://wiki.onosproject.org/display/ONOS/Developer+Quick+Start"

- Go to the Bazel website to install bazel; a program needed to install and run the ONOS controller: "https://docs.bazel.build/versions/master/install.html"
- Click on "Ubuntu Linux".
- Follow the "Using Bazel's apt repository"
- Start a new terminal and execute "sudo apt install curl"; a program needed to install bazel; which is needed to install and run ONOS.
- Execute "curl https://bazel.build/bazel-release.pub.gpg — sudo apt-key add -"

- Execute ”echo ”deb [arch=amd64] https://storage.googleapis.com/bazel-apt stable jdk1.8” — sudo tee /etc/apt/sources.list.d/bazel.list”
- Execute ”sudo apt update && sudo apt install bazel”
- Execute ”sudo apt update && sudo apt full-upgrade”
- Execute ”sudo apt install openjdk-11-jdk”
- Go back to ONOS develope quick guide - https://wiki.onosproject.org/display/ONOS/Developer+Quick+Start
- Install all of the ”Other dependencies” on this page by executing ”sudo apt-get install ¡dependency¿”, replacing ¡dependency¿ with each name in the list.
- Execute ”git clone https://gerrit.onosproject.org/onos”
- Execute ”cd onos”
- Execute ”bazel build onos”

Two tutorial VMs with Mininet and ONOS already installed with tutorials to test them and get to know them can be found at:

- https://wiki.onosproject.org/plugins/servlet/mobile?contentId=622595#content/view/1638475
- http://sdnhub.org/tutorials/sdn-tutorial-vm/

Those virtual machines are a great source to learn and get started, before building your own VM with all your needs.

3.2 Building the system

In order to run experiments; a process has to be followed to build the system.
Step0. Cleaning residual Mininet achieves.

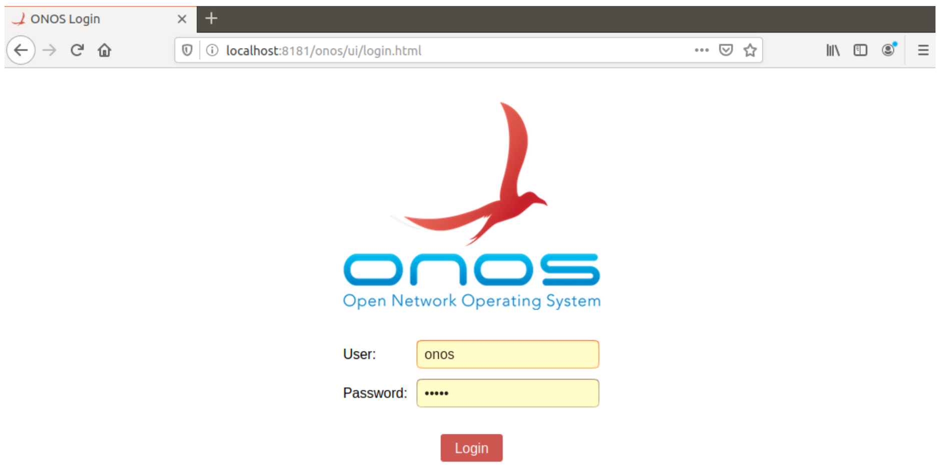
- Start a new terminal and go to the folder where the Mininet python script is located with the ”cd” command.
- Execute ”sudo mn -c” to get rid of all the residual files generated in other runs in order to start a new clean run.

Step1. Start a clean ONOS controller

- Start a new terminal and go to the onos folder with the ”cd” command
- Execute ”sudo bazel run onos-local – clean debug” to start a clean controller with no residual files or configurations from other runs. This process will take a few minutes. After the READY status is stated, go to Step2.

Step2. Build the ONOS controller

- Open a web browser and go to ”http://localhost:8181/onos/ui”
- Access by user:”onos” and password:”rocks”


ONOS Login
localhost:8181/onos/ui/login.html
onos
Open Network Operating System
Login

User: onos
Password: •••••

Figure 3.1: ONOS Log in page.

- Go to the applications section in the ONOS GUI and activate the apps that will build the controller.

Step3. Start the Mininet network.

- Start a new terminal and go to the folder where the Mininet python script is located with the "cd" command.
- Execute "sudo python NameOfTheScript.py"

After those steps, everything will be built in order to run new tests and experiments. When finished, it is recommended to run Step0.

Chapter 4

Methodology

This chapter will explain, in order, all the processes followed for the development of this project.

4.1 Selecting the ONOS applications

The ONOS controller is built with the combination of applications that will add different features or characteristic to the controller.

The applications preinstalled when starting the controller are "Default Drivers" and "ONOS GUI2". Installing "OpenFlow Provider Suite" which automatically installs "OpenFlow Base Provider" is essential in order to connect the controller to the Mininet network. The applications "Reactive Forwarding" and "Segment Routing" are the additions that will be explained in this project, which will build the optimal ONOS controller.

4.1.1 Reactive Forwarding: The forwarding application

When considering the forwarding application, possible options that ONOS offered and that were evaluated are:

- Multicast Forwarding; which sends in one transmission information from a host to a group of hosts. This application is more efficient when there are established groups that do not change much, which means that it works better in a proactive manner in which the routes are established prior to sending the data. It would not be mostly efficient in a dynamic network.
- Reactive Forwarding; which installs forwarding entries to the network switches on-demand after a sender starts transmitting packages. When a package is sent, the app handles the packet and configures the package accordingly. It works choosing from the shortest paths in hops count, using the shortest path algorithm. In this scenario, a hop count is the number of APs that a package travels through from the source to the destination. It is ideal for dynamic networks.

After running tests on a dynamic topology, with the multicast forwarding protocol, over 15% of more packages were lost, while with the reactive forwarding protocol, an important increase of packages per second, as will be shown in the "App comparison" section's Figure, combining all the final chosen applications, and differentiating this two forwarding apps.

4.1.2 Segment Routing: A great addition.

The controller is built from combining applications, and a great addition to the collection that increases the Quality of Service (QoS) and has other benefits is Segment Routing.

[12]It divides the network in segments and each node and link have their own segment ID or SID, while using the shortest path algorithm for their segments, which is the same approach that the reactive forwarding application does. That optimizes the fast reroute schemes and gives simplicity and scalability attributes combined with advanced protection.

4.1.3 App comparison

The following tests were acquired with all the applications installed but differentiating the forwarding application.

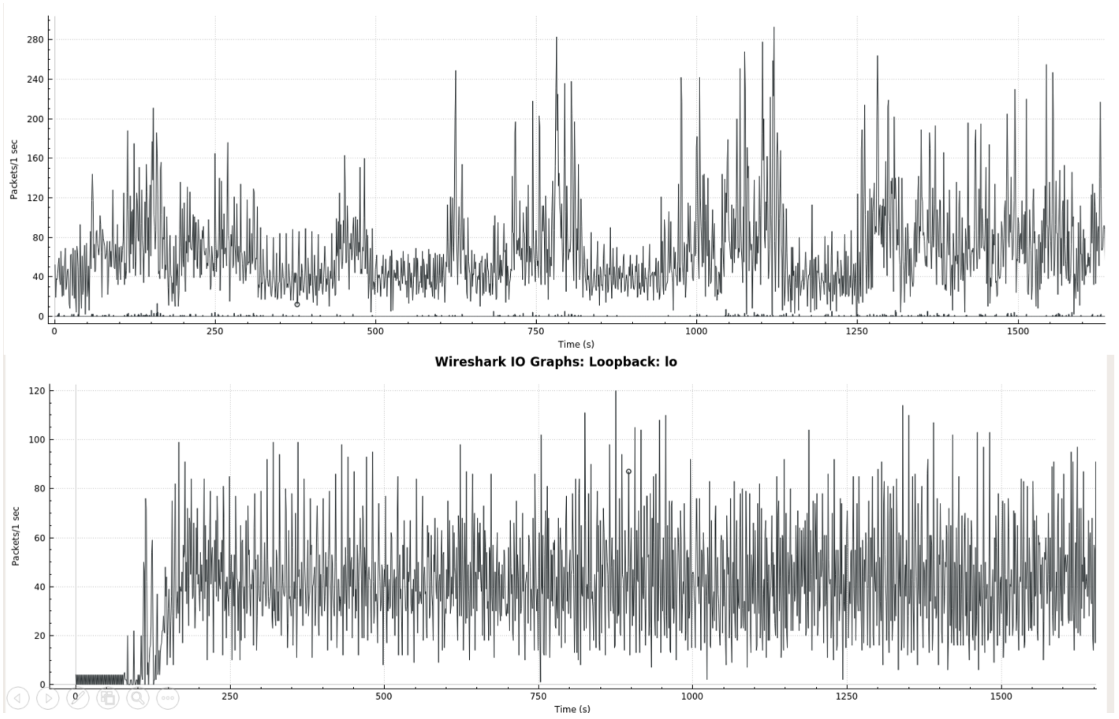


Figure 4.1: The top picture shows the results obtained with reactive forwarding and the bottom one with multicast forwarding.

An important increase of packages per second is observed. Reactive forwarding reaches more than 280 packages/second and multicast forwarding only reaches 120. Those graphs were obtained with the program Wireshark. The dynamic wireless software defined network used to run the tests had 40 hosts and 6 access points in a linear topology, doing ”pingall” to compare the packages received/lost.

Tests run with only the reactive forwarding application reveal that after adding the segment routing application, the number of delivered packages increased 4% and the maximum number of packages per second increased from 200 to over 280, improving the QoS.

4.2 Reactive forwarding modification

Once the final applications were chosen, their code was reviewed to check what could be done to improve the final controller. There were TODO sections, with features that could be added, and other comments in which functions could be changed to have a better functionality.

However, the focus was held in the process of selecting the path to follow, which was held by the Reactive Forwarding app, in the script "ReactiveForwarding.java". The function "Set Path getPaths(Topology topology, DeviceId src, DeviceId dst)" returns a set of all the shortest paths, precomputed in terms of hop-count, between the specified source and destination devices, as stated in the script "TopologyService.java".

Different functions to get the shortest paths were studied to see if it could mean an improvement to the controller, although it was not possible to make them compatible with the Reactive Forwarding app, as it will be explained in the following points:

- The k-shortest path is already implemented in ONOS. However, it gives a Stream of paths, and the Reactive Forwarding application works with a Set of paths, giving incompatibilities in some functions.
- Yen's algorithm [13] is a modified version of the k-shortest path. It works with graphs, and the Reactive Forwarding app works with topologies, and its definition of basic elements and their functions such as Paths or Links are different, not being able to work together without a complete transformation.

After that process, the function "Path pickForwardPathIfPossible(Set Path paths, PortNumber notToPort)" was changed.

Originally, the chosen function would pick a random path of the Set of shortest paths which would not lead back to the specified port of origin. That approach could be beneficial with less time and less resources consumed to select the path to follow, knowing that the chosen path would already be one of the shorter ones from the "getPath" function.

However, that function was modified to check all the paths and keep the one with the lower cost, instantly returning a direct path with cost 1.0 if found. That would make sure that the shortest path in terms of cost was followed. The cost is measured in the AP hops count. That is how many APs a package goes through from the source to the destination. In the Experimental Analysis chapter, the results and the impact in that change will be discussed.

```java
// Selects a path from the given set that does not lead back to the
    specified port if possible.
private Path pickForwardPathIfPossible(Set<Path> paths, PortNumber
    notToPort) {
 Path bestPath = null;
  for (Path path : paths) { //goes through all the paths
      if (!path.src().port().equals(notToPort)) {
        if((bestPath == null) || (path.cost()<bestPath.cost())){
          bestPath=path;
          if(bestPath.cost() == 1.0){
          return bestPath;
          }
        }
      }
  }
  return bestPath;
}


    /*private Path pickForwardPathIfPossible(Set<Path> paths,
    PortNumber notToPort) {
         for (Path path : paths) {
              if (!path.src().port().equals(notToPort)) {
                  return path;
              }
         }
         return null;
    }*/
```

Listing 4.1: Modified Java code in the upper part vs original code in the commented lower section

4.3 Finding the appropriate network for the final testings

In order to get relevant results, a realistic topology structure was created.

The wireless dynamic topology was based on the University of Florida's campus. UF's campus has an area of 8km^2, and around 52000 students registered; from which not all of them would be connected to the campus network at once. In 2012, there were 1700 Access Points 802.11n, which have a range of 820 meters. [14]

However, due to hardware limitations, simulating a topology with that amount of components was not possible, and the numbers had to be measured down.

After tests with different number of elements, the biggest topology that the testing machine could simulate and handle had 15 APs with a range of 200m, 40 hosts, and an area of 1.96km^2. The range and area were kept approximately with the same ratio as the UF's campus has, and a higher ratio of APs per host was chosen since the following tests and simulations would be made to observe the cost of the paths that would be chosen when sending packages from host to host. The velocity of the movement of the APs and hosts positions was set between 0.5 and 5 m/sec and the original seed was 3.
The code with every value to create it will be explained in the following Listing:

```
'Setting mechanism to optimize the use of APs'

from mininet.node import RemoteController
from mininet.log import setLogLevel, info
from mn_wifi.cli import CLI
from mn_wifi.net import Mininet_wifi
import math


def topology():
    "Create a network."
    net = Mininet_wifi(controller=RemoteController)

    print "*** Creating nodes"

    stat = []
    apt = []
```

```
n = 40 """40 hosts"""
m = 15 """15 APs"""

"""Creating and adding the APs and hosts to the network"""

for i in range(n):
stat.append('sta'+str(i))

for i in range(m):
apt.append('ap'+str(i))

sta = []
ap = []
print stat

for i in range(n):
sta.append(str(i))

print sta

for j in range(n):
stat[j] = net.addStation(sta[j])

for i in range(n+1,n+m+1):
ap.append(str(i))

for j in range(m):
apt[j] = net.addAccessPoint(ap[j], range=200) """The APs have a
range of 200m"""

print ap


c0 = net.addController('c0', controller=RemoteController)

print "*** Configuring wifi nodes"
net.configureWifiNodes()

print "*** Creating tree links"


net.addLink(ap[0], ap[1])
net.addLink(ap[0], ap[2])
net.addLink(ap[1], ap[3])
net.addLink(ap[1], ap[4])
net.addLink(ap[2], ap[5])
net.addLink(ap[2], ap[6])
net.addLink(ap[3], ap[7])
net.addLink(ap[3], ap[8])
```

```
 net.addLink(ap[4], ap[9])
 net.addLink(ap[4], ap[10])
 net.addLink(ap[5], ap[11])
 net.addLink(ap[5], ap[12])
 net.addLink(ap[6], ap[13])
 net.addLink(ap[6], ap[14])

 """
 Log-Distance path loss model is a model that predicts the signal
loss around buildings or with a lot of users around its distance.
 The bigger the exponent, the bigger the loss.
 The exponent of 5 was chosen as if there was much indoors
interference such as buildings, labs, sportive areas, or a stadium
 """
 net.setPropagationModel(model="logDistance", exp=5)

 """plotting graph, and setting its wide and length in meters"""
 net.plotGraph(max_x=1400, max_y=1400)

 """Set mobility for each node"""

 net.setMobilityModel(time=0, model='RandomWayPoint', max_x=1400,
max_y=1400, min_v=0.5, max_v=5, seed=3, ac_method='llf')
 """
 time: time (in seconds) in which the mobility is started
 model: mobility model
 max_x: maximum x position
 max_y: maximum y position
 min_v: minimum moving velocity in m/s
 max_v: maximum moving velocity in m/s
 seed: defines a starting value for a pseudo random sequence
 ac_method: handover association mechanism. The one used is: last
loaded first
 """

 print "*** Starting network"
 net.build()
 c0.start()

 for o in range(len(apt)):
 apt[o].start([c0])


 info("*** Running CLI\n")
 CLI(net)

 info("*** Stopping network\n")
 net.stop()
```

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

Listing 4.2: Python script for the final topology used in the concluding experiments.

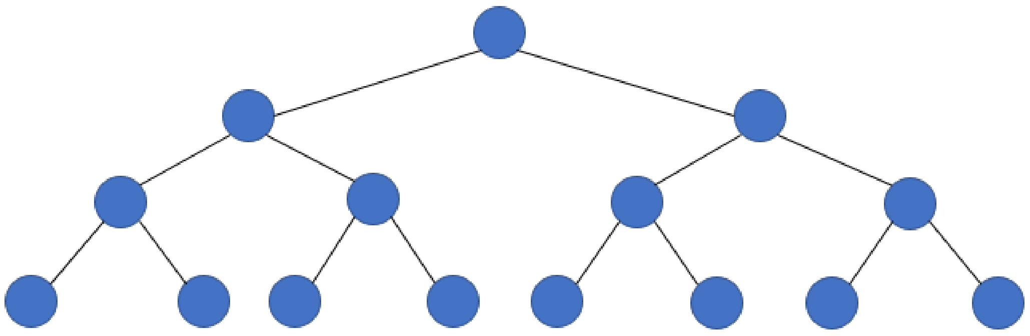


Figure 4.2: Representation of a 15 AP tree topology

Regarding the APs connections, a tree topology was chosen. [15] Campus Area Networks are generally based on a tree topology, which is a hybrid topology between the star and the bus topology. It is a more flexible and scalable network since new "branches" and APs can be added or modified easily, with a point to point connection and featuring a centralized monitoring with easier maintenance and fault finding. It is ideal for a large network such as the campus of a university. However, it needs a lot of maintenance and the backbone (or main AP) forms a point of failure.

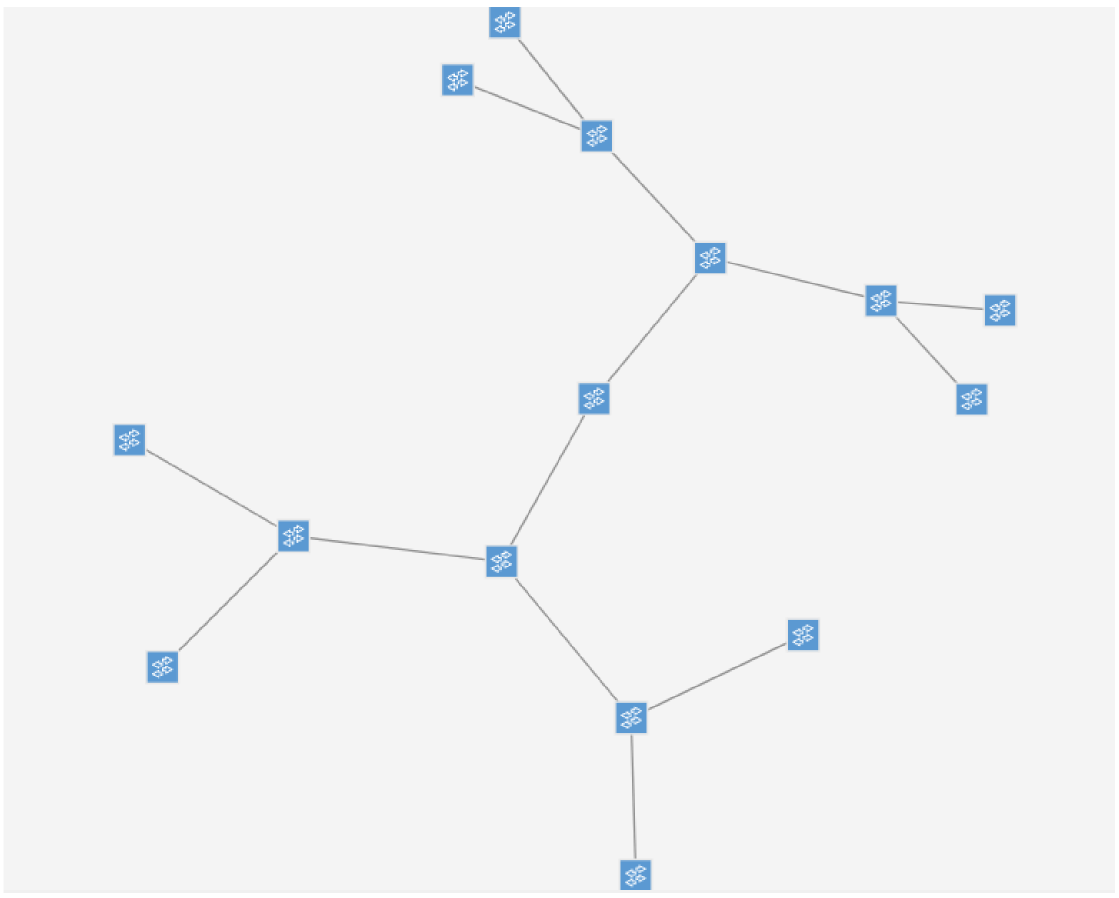


Figure 4.3: Representation of the dynamic tree topology from the ONOS GUI

Chapter 5

Experimental Analysis

The following test results were made with a dynamic network of 15 APs in a tree structure topology, and with 40 host that will ping all with each other, as stated in the Methodology chapter.

There were different results when tested in different starting times, and each test would show different variations of results and improvements. Nevertheless, some critical analysis can show the contrast in using each version of the algorithm. The packages delivered/lost had no noticeable difference, but the cost of the followed paths showed a differentiation.

	Original	Modified
Max. time to select a route	1ms	5ms
Average reactive processor p/sec	572.6	574.3
Max. CPU usage	84.4%	85.2%
Max. RAM memory	3.61GB	4.06GB

Table 5.1: Data comparison during the final experiment.

As the upper table shows, this algorithm change does not show a significant amount of extra hardware usage. Those results were obtained using HTOP, the ONOS GUI, and Wireshark, which can sometimes saturate the system and be a reason of why for example there is a 0.45GB difference in the RAM memory usage.

The time to select a package was mostly at very low values. However, in very rare occasions, that time was increased to higher values such as 15 ms in both versions.
Since the process in selecting a path is more complex in the newer version, it is expected to have a very few extra milliseconds in selecting the shortest path. That slight difference did not make any noticeable impact in the amount of delivered packages. Nevertheless, this algorithm should be tested on a bigger network to see how it would affect a bigger scale network such as the campus of a university.

	Original	Modified
Cost = 1.0	10%	13%
Cost = 2.0	34%	42%
Cost = 3.0	34%	25%
Cost = 4.0	10%	12%
Cost = 5.0	12%	8%

Table 5.2: Path costs during the final experiment.

The upper table shows the main difference when changing the algorithm. As stated previously, the contrast can be stronger or lighter depending on the topology and its current state. From this particular example, it can be perceived that there were numerous cases in which there was only one possible path to follow, and some other situations in which the original random algorithm chose the best possible path. However, it is noticed that there is a major improvement in the path costs, with major differences specially in the paths with 2 hops, focusing most of the weight of the network in shorter and faster paths.

With all this data, depending on the speed of the network, this could mean a major improvement in the QoS of a network.

- On a very fast optimized network that prioritizes sending the data as soon as it gets received, a few extra milliseconds in selecting a path to only loose a smaller time from using a shorter path, could lead into a slight increase in the network's latency, making the network slower. This case could apply to wired networks which have a latency of less than 1 ms and specially on fiber optics; which latency is 4.9 microseconds ($4.9*10^{-3}$ ms) per kilometer.[16]

- When the network relies more on what path to choose from, this algorithm could make a major improvement.

  On WiFi (wireless fidelity) based networks, if the network is running optimally, it would take about 1-4 ms per AP to go through for just a ping. In this scenario, going through the path with less APs hop count would make a major improvement.[17]

Another feature considered that could be useful would be to have a whitelist/blacklist system to choose from paths that are already known to be reliable and efficient and discard the ones that frequently experience failures or slower speeds.
In the end, that feature was not added to this project due to the nature of a dynamic network, having all the APs and hosts changing position, having situations in which paths from those lists could not exist anymore. If the APs did not

change much or if they were static, that system could be a great addition for a future project.

Chapter 6

Conclusion

In this paper, it has been discussed how a combination of the reactive forwarding app and the segment routing app could build a great ONOS controller for a wireless dynamic Software Defined Network.

A new process in selecting the path to follow from the reactive forwarding app is proposed, to choose the path with the least AP hops, which would be beneficial specially for wireless networks.

The objectives proposed have been met, and future works are proposed to improve the final controller.

Chapter 7

Future works

Future works in this topic to improve the ONOS controller could be:

- Test the built controller with the new code in a bigger network to experience more noticeable differences.
- An implementation of a k-shortest path type algorithm to make the selection of shortest paths.
- An implementation of a whitelist/blacklist system.

Appendix A

Sustainable Development Goals of the project

This technology meets the goal number 9 of sustainable development goals established by the United Nations which is "Industry, Innovation and Infrastructure". This project meets that objective because it presents a system in which there will be less hardware needed in processes where it would be necessary.

In this case it is with the software controller substituting the control plane hardware, saving materials and energy that the extra hardware would require to run. As a consequence, less energy would be used by less hardware to run the same operations. Less hardware would be produced, saving materials, saving the production energy, and as a consequence, less pollution would be thrown to the Earth, and there would be less future residuals.
That would also benefit economically with less expenses.

Appendix B

Scientific paper

During this project, a scientific paper was written.
It will be published when a conference or opportunity comes along.

Optimal time route planning in dynamic wireless software defined networks

Alvaro del Aguila, Allen Starke, Dr. Janise McNair
Dept. of Electrical & Computer Engineering University of Florida; Gainesville, FL 32611
delaguila.alvaro@hotmail.com, allen1.starke@ufl.edu, mcnair@ece.ufl.edu

***Abstract*—This paper presents a SDN controller app development using an ONOS controller. It focuses on the Reactive Forwarding and Segment Routing protocols, updating and upgrading them, to ensure that the Quality of Service in a dynamic wireless software defined network is always met.**

I. Introduction

Moore's Law states that every two years, the amount of transistors in a microprocessor doubles in number. This law has not been far from reality, and our society has experienced one of the fastest technology growths this past decade, affecting specially the telecommunications.
New technologies have been created such as the Internet of Things (IoT) or Big Data.

As new technologies develop, the coordination and management of large numbers of devices will both be a requirement and a significant challenge. From Internet of Things (IoT) devices to smart phones, tablets and laptops, to Unmanned Air Vehicles (UAVs), to Small Satellites. Interoperability and on-demand communication paths require an adaptive approach to network management techniques that can be employed to create a network management architecture.

Software defined networking (SDN) is one of the most important new concepts that have been gaining weight these past years, being able to separate the control plane from the data plane, and to centralize the network with a controller, which is software built with applications written on a high level code, giving the opportunity for a dynamic maintenance of the network. It brings more flexibility to configure the network, and improves the efficiency of the network. It would also decrease the need of hardware and as a consequence, it would lower costs. More details on SDN will be given in the next section.

However, due to economy and compatibility reasons, having a full deployment of SDN in just one step would not be a realistic approach, but with an increasing speed, it will play an important role in our society in the near future. [1]

As a revolutionary concept, SDN is evolving in a fast pace. There are many applications for different controllers that give different features and capabilities to the network, with many possibilities for development and improvement.[2]

This project is focused on dynamic wireless software defined networks, to ensure an optimal and constant quality of service using the ONOS controller. In this paper, it will be discussed how segment routing and reactive forwarding improve the QoS of this specific type of network. Those two applications have open code that can be modified, adding features that are stated to be missing and left to develop, and creating a configuration that would optimize the controller. The goal of this paper is to create a reliable controller that assures an optimal and constant QoS.

II. Background Information

A. *Defining SDN*

As an emerging networking architecture, Software Defined Networking (SDN) is established as a promising concept for our future. It is an efficient networking technology that supports the dynamic nature functions of our future networks with the crucial aspect of simplifying hardware through software, with many benefits such as lowering costs. [3]

The main characteristics that differentiate SDN from any other network layer architectures are:

- Differentiation and separation of the control plane and the data plane of the switching hardware.
- Using software for the control plane functions, reducing hardware usage and requirements. That software is known as a controller, which is centralized and has a complete view of the network.
- Open interfaces between the controllers and the devices located in the data plane.
- Customization of the controller through applications that can be programmed by any external source or user; which is what this paper is focused on.

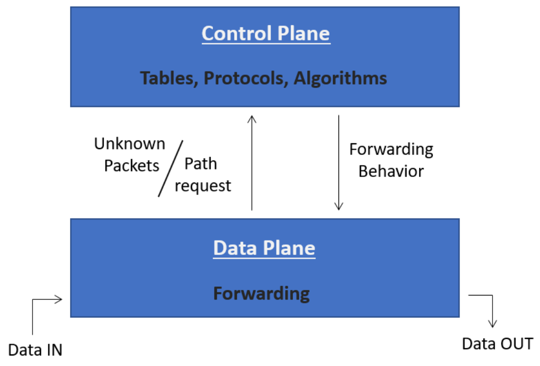


Fig. 1. Control plane and data plane schemes regarding SDN

B. The ONOS controller

The controller used for this paper is the ONOS controller; which stands for Open Network Operating System.[4]

As mentioned previously, the SDN controller is built with different applications written in Java code.
One of the most important ones is the OpenFlow protocol; which has a very important role in the controller, giving access to the forwarding plane of a network switch or router, and allowing the controller to determine the route of the packets across a network. It is layered on top of the Transmission Control Protocol (TCP) and establishes the use of the Transport Layer Security (TLS).

When considering the forwarding application, possible options that ONOS offered and that were evaluated are:

- Multicast Forwarding; which sends in one transmission information from a host to a group of hosts. This application is more efficient when there are established groups that do not change much, which means that it works better in a proactive manner in which the routes are established prior to sending the data. It would not be mostly efficient in a dynamic network.
- Reactive Forwarding; which installs forwarding entries to the network switches on-demand after a sender starts transmitting packages. When a package is sent, the app handles the packet and configures the package accordingly. It works choosing from the shortest paths in hops count, using the shortest path algorithm. It is ideal for dynamic networks.

After running tests on a dynamic topology, it is observed that with the multicast forwarding protocol, over 15% of packages were lost, while with the reactive forwarding protocol, an increase of packages per second could be observed, as the Figure 2 shows. Reactive forwarding reaches more than 280 packages/second and multicast forwarding only reaches 120. Those graphs were obtained with the program Wireshark. The dynamic topology used to run the tests had 40 hosts and 6 access points in a mobile network, doing "ping alls" with all the hosts to compare the packages received/lost.

The controller is built from combining applications. A great addition that increases the Quality of Service (QoS) and has other benefits is Segment Routing. [5]It divides the network in segments and each node and link have their own segment ID or SID, while using the shortest path algorithm for their segments, which what the reactive forwarding application does. That optimizes the fast reroute schemes and gives simplicity and scalability attributes combined with advanced protection. The Figure 2 tests were acquired with the segment routing application installed already. Tests run with only the reactive forwarding application reveal that after adding the segment routing application, the number of delivered packages increased 4% and the maximum number of packages per second increased from 200 to over 280, improving the QoS and making it the starting point for the development of this project.

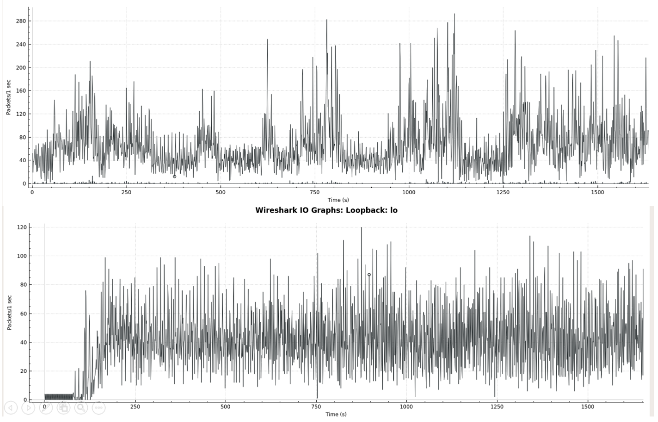


Fig. 2. The top picture shows the results obtained with reactive forwarding and the bottom one with multicast forwarding.

C. Mininet-WiFi

Mininet is a SDN network emulator that simulates virtual hosts, switches, links and controllers, that run Linux network software.[6]

Mininet-WiFi is an extension that adds WiFi stations and Access Points (AP) based on wireless Linux drivers, which perfectly suits the wireless dynamic SDN network needs that are demanded for this project. It supports OpenFlow, and it can get connected to the ONOS controller, which is the controller used for this set up.

Mininet-WiFi can run scripts programmed on Python, which gives freedom to create different topologies for different tests. For the topic of this paper, Mininet-WiFi gives the tools to create your custom dynamic SDN network, adding as many hosts and APs as needed, with a defined range, and mobility with different speeds and paths that can be programmed. That can provide the perfect set up since you can experience different scenarios such as hosts getting out of reach, or conflicts of hosts being in range with more than one AP.

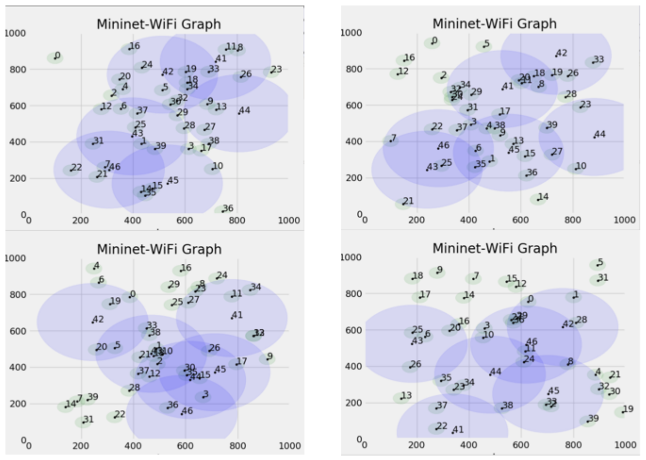


Fig. 3. Example of a dynamic topology run in Mininet-WiFi, with 40 hosts and 6 APs.

III. Methodology

During this project, alternative approaches were tested.

The first tests were run on a smaller and less complex dynamic topology. It had 40 nodes, 6 APs, and a linear structure, doing ”ping alls” to test the percentage of packages that would reach their destination.
Once the final applications were chosen, their code was reviewed to check what could be done to improve the final controller. There were TODO sections, with features that could be added, and other comments in which functions could be changed to have a better functionality.

However, the focus was held in the process of selecting the path to follow, which was held by the Reactive Forwarding app, in the script ”ReactiveForwarding.java”. The function ”Set Path getPaths(Topology topology, DeviceId src, DeviceId dst)” returns a set of all the shortest paths, precomputed in terms of hop-count, between the specified source and destination devices, as stated in the script ”TopologyService.java”.

Different functions to get the shortest paths were studied to see if it could mean an improvement to the controller, although it was not possible to make them compatible with the Reactive Forwarding app, as it will be explained in the following points:

- The k-shortest path is already implemented in ONOS. However, it gives a Stream of paths, and the Reactive Forwarding application works with a Set of paths, giving incompatibilities in some functions.
- Yen's algorithm [7] is a modified version of the k-shortest path. It works with graphs, and the Reactive Forwarding app works with topologies, and its definition of basic elements and their functions such as Paths or Links are different, not being able to work together without a complete transformation.

After that process, the function ”Path pickForwardPathIfPossible(Set Path paths, PortNumber notToPort)” was changed.

Originally, it would pick a random path of the Set of shortest paths which would not lead back to the specified port of origin. That approach could be beneficial with less time and less resources consumed to select the path to follow, knowing that the chosen path would already be one of the shorter ones from the ”getPath” function.

However, that function was modified to check all the paths and keep the one with the lower cost, instantly returning a direct path with cost 1.0 if found. That would make sure that the shortest path in terms of cost was followed. The cost is measured in the hops made of the APs that a package goes through when going from a source to the destination. In the Experimental Analysis section, the results and the impact in that change will be discussed.

```
//Selects a path from the given set that does not
    lead back to the specified port if possible.
private Path pickForwardPathIfPossible(Set<Path>
    paths, PortNumber notToPort){
 Path bestPath = null;
  for(Path path : paths){ //goes through all the
    paths
      if(!path.src().port().equals(notToPort)){
        if((bestPath == null) || (path.cost()<
    bestPath.cost())){
            bestPath=path;
            if(bestPath.cost() == 1.0){
            return bestPath;
            }
          }
        }
  }
  return bestPath;
}


    /*private Path pickForwardPathIfPossible(Set<
    Path> paths, PortNumber notToPort){
        for(Path path : paths){
          if(!path.src().port().equals(notToPort)){
            return path;
          }
        }
      return null;
    }*/
```

Listing 1. Modified Java code in the upper part vs original code in the commented lower section

In order to get relevant results, a realistic topology structure was created. The wireless dynamic topology was based on the University of Florida's campus. UF's campus has an area of 8km, and around 52000 students registered; from which not all of them would be connected to the campus network at once. In 2012, there were 1700 Access Points 802.11n, which have a range of 820 meters. [8] However, due to hardware limitations, simulating a topology with that amount of components was not possible, and the numbers had to be measured down. After tests with different number of elements, the biggest topology that the testing machine could simulate and handle had 15 APs with a range of 200m, 40 hosts, and an area of 1.96km^2.
The range and area were kept approximately with the same ratio as the UF's campus has, and a higher ratio of APs per host was chosen since the following tests and simulations would be made to observe the cost of the paths that would be chosen when sending packages from host to host. The velocity of the movement of the APs and hosts positions was set between 0.5 and 5 m/sec and the original seed was 3.

Regarding the APs connections, a tree topology was chosen. [9] Campus Area Networks are generally based on a tree topology, which is a hybrid topology between the star and the bus topology. It is a more flexible and scalable network since new ”branches” and APs can be added or modified easily, with a point to point connection and featuring a centralized monitoring with easier maintenance and fault finding. It is ideal for a large network such as the campus of a university. However, it needs a lot of maintenance and the backbone (or main AP) forms a point of failure.

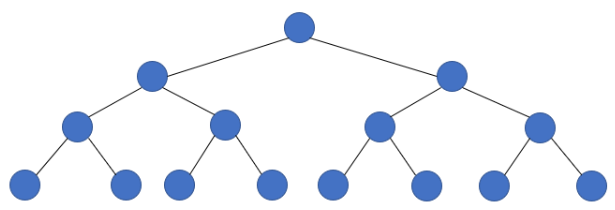


Fig. 4. Representation of a 15 AP tree topology

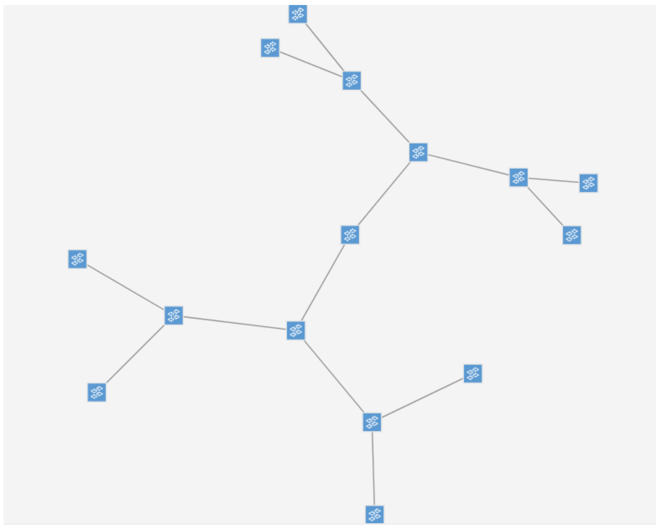


Fig. 5. Representation of the dynamic tree topology from the ONOS GUI

IV. Experimental Analysis

All the experiments were run in an Ubuntu Virtual Machine with 8GB of RAM and 2CPUs, in a computer with 16GB of RAM and an Intel(R)Core(TM)i7-3520M CPU @ 2.90GHz. The programs used were Mininet with its Mininet-WiFi extension to create the dynamic topology, ONOS to establish the controller, Wireshark to capture the packets and evaluate the data obtained, and HTOP to record the hardware's usage.

The following test results were made with a dynamic network of 15 APs in a tree structure topology, and with 40 host that will ping all with each other, as stated in the Methodology section. It can be observed in Figures 4 and 5.

There were different results when tested in different starting times, and each test would show different variations of results and improvements. Nevertheless, some critical analysis can show the contrast in using each version of the algorithm. The packages delivered/lost had no noticeable difference, but the cost of the followed paths showed a differentiation.

As Table I shows, this algorithm change does not show a significant amount of extra hardware usage. Those results were obtained using HTOP, the ONOS GUI, and Wireshark, which can sometimes saturate the system and be a reason of why for example there is a 0.45GB difference in the RAM memory usage. The time to select a package was mostly at very low values. However, in very rare occasions, that time was increased to higher values such as 15 ms in both versions. Since the process in selecting a path is more complex in the newer version, it is expected to have a very few extra milliseconds in selecting the shortest path. That slight difference did not make any noticeable impact in the amount of delivered packages. Nevertheless, this algorithm should be tested with a bigger network to see how it would affect a bigger scale network such as the campus of a university.

	Original	Modified
Max. time to select a route	1ms	5ms
Average reactive processor p/sec	572.6	574.3
Max. CPU usage	84.4%	85.2%
Max. RAM memory	3.61GB	4.06GB

TABLE I
Data comparison during the final experiment.

	Original	Modified
Cost = 1.0	10%	13%
Cost = 2.0	34%	42%
Cost = 3.0	34%	25%
Cost = 4.0	10%	12%
Cost = 5.0	12%	8%

TABLE II
Path costs during the final experiment.

Table II shows the main difference when changing the algorithm. As stated previously, the contrast can be stronger or lighter depending on the topology and its current state. From this particular example, it can be perceived that there were numerous cases in which there was only one possible path to follow, and some other situations in which the original random algorithm chose the best possible path. However, it is noticed that there is a major improvement in the path costs, with major differences specially in the paths with 2 hops, focusing most of the weight of the network in shorter and faster paths.

With all this data, depending on the speed of the network, this could mean a major improvement in the QoS of a network.

On a very fast optimized network that prioritizes sending the data as soon as it gets received, a few extra milliseconds in selecting a path to only loose a smaller time from using a shorter path, could lead into a slight increase in the network latency, making the network slower. This case could apply to wired networks which have a latency of less than 1 ms and specially on fiber optics; which latency is 4.9 microseconds ($4.9*10^{-3}$ ms) per kilometer.[10]

However, when the network relies more on what path to choose from, this algorithm could make a major improvement. On WiFi (wireless fidelity) based networks, if the network is running optimally, it would take about 1-4 ms per AP to go through for just a ping. In this scenario, going through the path with less APs hop count would make a major improvement.[11]

Another feature that could be useful would be to have a whitelist/blacklist system to choose from paths that are already known to be reliable and efficient and discard the ones that frequently experience failures or slower speeds. That feature was not added to this project due to the nature of a dynamic network, having all the APs and hosts changing position, having situations in which paths from those lists could not exist anymore. If the APs did not change much or if they were static, that system could be a great addition for a future project.

V. Conclusion

In this paper, it has been discussed how a combination of the reactive forwarding app and the segment routing app could build a great ONOS controller for a wireless dynamic Software Defined Network. A new process in selecting the path to follow from the reactive forwarding app is proposed, to choose the path with the least AP hops.

Future works in this topic could test the built controller with the new code in a bigger network to experience more noticeable differences. An implementation of a k-shortest path type algorithm to make the selection of shortest paths, and an implementation of a whitelist/blacklist system, could be future approaches to improve this ONOS controller.

References

[1] Yuanguo Bi, Guangjie Han, Chuan Lin, Yan Peng, Huayan Pu and Yazhou Jia, ”Intelligent Quality of Service Aware Traffic Forwarding for Software-Defined Networking/Open Shortest Path First Hybrid Industrial Internet”, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, VOL. 16, NO. 2, FEBRUARY 2020

[2] Allen Starke, Zixiang Nie, Morgan Hodges, Corey Baker and Janise McNair, ”Denial of Service Detection Mitigation Scheme using Responsive Autonomic Virtual Networks (RAvN)”, IEEE Military Communications Conference (MILCOM), November 2019

[3] Sakir Sezer, Sandra Scott-Hayward, Pushpinder Kaur Chouhan, Barbara Fraser, David Lake, Jim Finnegan, Niel Viljoen, Marc Miller and Navneet Rao, ”Are We Ready for SDN?Implementation Challenges for Software-Defined Networks”, IEEE Communications Magazine • July 2013

[4] ONOS https://wiki.onosproject.org/display/ONOS/ONOS

[5] ”Segment Routing 101 and the Future of MPLS” https://aviatnetworks.com/blog/segment-routing-101-and-the-future-of-mpls/

[6] Mininet http://mininet.org/

[7] Yen’s algorithm https://github.com/bsmock/k-shortest-paths/blob/master/edu/ufl/cise/bsmock/graph/ksp/Yen.java

[8] UF campus wireless network https://net-services.ufl.edu/provided-services/wireless/

[9] Campus Topology https://www.conceptdraw.com/examples/what-kind-of-topology-used-in-college

[10] Latency https://whatis.techtarget.com/definition/latency

[11] The latency of a wifi network https://www.quora.com/What-is-the-average-latency-of-a-WiFi-network

Bibliography

[1] "[Yuanguo Bi, Guangjie Han, Chuan Lin, Yan Peng, Huayan Pu and Yazhou Jia] Intelligent Quality of Service Aware Traffic Forwarding for Software-Defined Networking/Open Shortest Path First Hybrid Industrial Internet". In: *IEEE Transactions on Industrial Informatics* 16.2 (February 2020).

[2] *software-defined networking (SDN).* URL: `https://searchnetworking.techtarget.com/definition/software-defined-networking-SDN`.

[3] "[Sakir Sezer, Sandra Scott-Hayward, Pushpinder Kaur Chouhan, Barbara Fraser, David Lake, Jim Finnegan, Niel Viljoen, Marc Miller and Navneet Rao]Are We Ready for SDN? Implementation Challenges for Software-Defined Networks". In: *IEEE Communications Magazine* (July 2013).

[4] "[Allen Starke, Zixiang Nie, Morgan Hodges, Corey Bakerand Janise McNair] Denial of Service DetectionMitiga-tion Scheme using Responsive Autonomic Virtual Networks(RAvN)". In: *IEEE Military Communications Conference (MIL-COM)* (November 2019).

[5] *QoS (quality of service).* URL: `https://searchunifiedcommunications.techtarget.com/definition/QoS-Quality-of-Service`.

[6] *What is a virtual machine?* URL: `https://www.redhat.com/en/topics/virtualization/what-is-a-virtual-machine`.

[7] *Defining the terms driver, firmware, hardware, software, and utility.* URL: `https://kb.netgear.com/1070/Defining-the-terms-driver-firmware-hardware-software-and-utility`.

[8] *Whitelist and Blacklist Overview.* URL: `https://www.juniper.net/documentation/en_US/release-independent/sky-atp/help/information-products/pathway-pages/topic-98706.html`.

[9] *Ping.* URL: `https://searchnetworking.techtarget.com/definition/ping`.

[10] *ONOS.* URL: `https://wiki.onosproject.org/display/ONOS/`.

[11] *Mininet.* URL: `http://mininet.org/`.

[12] *Segment Routing 101 and the Future of MPLS*. URL: `https://aviatnetworks.com/blog/segment-routing-101-and-the-future-of-mpls/`.

[13] *Yen's algorithm*. URL: `https://github.com/bsmock/k-shortest-paths/blob/master/edu/ufl/cise/bsmock/graph/ksp/Yen.java`.

[14] *UF campus wireless network*. URL: `https://net-services.ufl.edu/provided-services/wireless/`.

[15] *Campus Topology*. URL: `https://www.conceptdraw.com/examples/what-kind-of-topology-used-in-college`.

[16] *Latency*. URL: `https://whatis.techtarget.com/definition/latency`.

[17] *The latency of a WiFi network*. URL: `https://www.quora.com/What-is-the-average-latency-of-a-WiFi-network`.