Koç University

College of Engineering

Department of Electrical & Electronics Engineering

ELEC 291

Summer Practice Report

Osman Raşit Kültür

SUMMER PRACTICE REPORT

Student Name	:	Osman Raşit Kültür
Starting Date	:	01 August
Completion Date	:	20 August
Total Working Days	:	20
Summer Practice #	:	291
Company	:	ARGELA Yazılım ve Bilişim Tekn. San. Ve Tic.
Department	:	R & D Department
Address	:	Reşitpaşa Mah. Katar Cad. No:4 İTÜ ARI Teknokent 3 İç Kapı No:502/601 Maslak Sarıyer Istanbul, 34469 İstanbul

Table of Contents

1. Works and Projects

In ARGELA, I had a long task to research on it. It was about developing an algorithm that finds the shortest paths with some constraints. But this algorithm could be used in software define networking based communication systems. That's why I had to build a background.

In my first week at the internship, I focused on the Software Defined Networking and I tried to understand how SDN works. Second week I studied Open Networking Operating System (ONOS) because it was an important open source project that would be useful for SDN based project. At the third week, I focused on algorithms. To examine better, I first reviewed data structures and some well-known algorithms quickly like Breadth First Search, Depth First Search, K-Shortest Path Search and Dijkstra's algorithms. I also tried to implement these algorithms in java. Last week, I installed Ubuntu Operating System in my PC and then I installed IntelliJ Integrated Development Environment in order to survey ONOS source code so that I could understands how the algorithms are implemented in SDN based system.

In four weeks, I've successfully gained understanding on SDN and graph algorithms. Since it was a new concept for me, I showed extra effort to understand them. I continued to study them and read related papers & textbook when I was home after the work. At the end of the day, they all made sense to me as a whole.

1.1. Software Defined Networking Studies

The networking technology is started when the internet is invented. At that time, first networks were made between computers and then the networks among the places got bigger and bigger. Nowadays we have networks all over the world, they even exist in little places like a home or some small business buildings that operates via computers. Some of the network topologies are shown at the figure 1 as following.

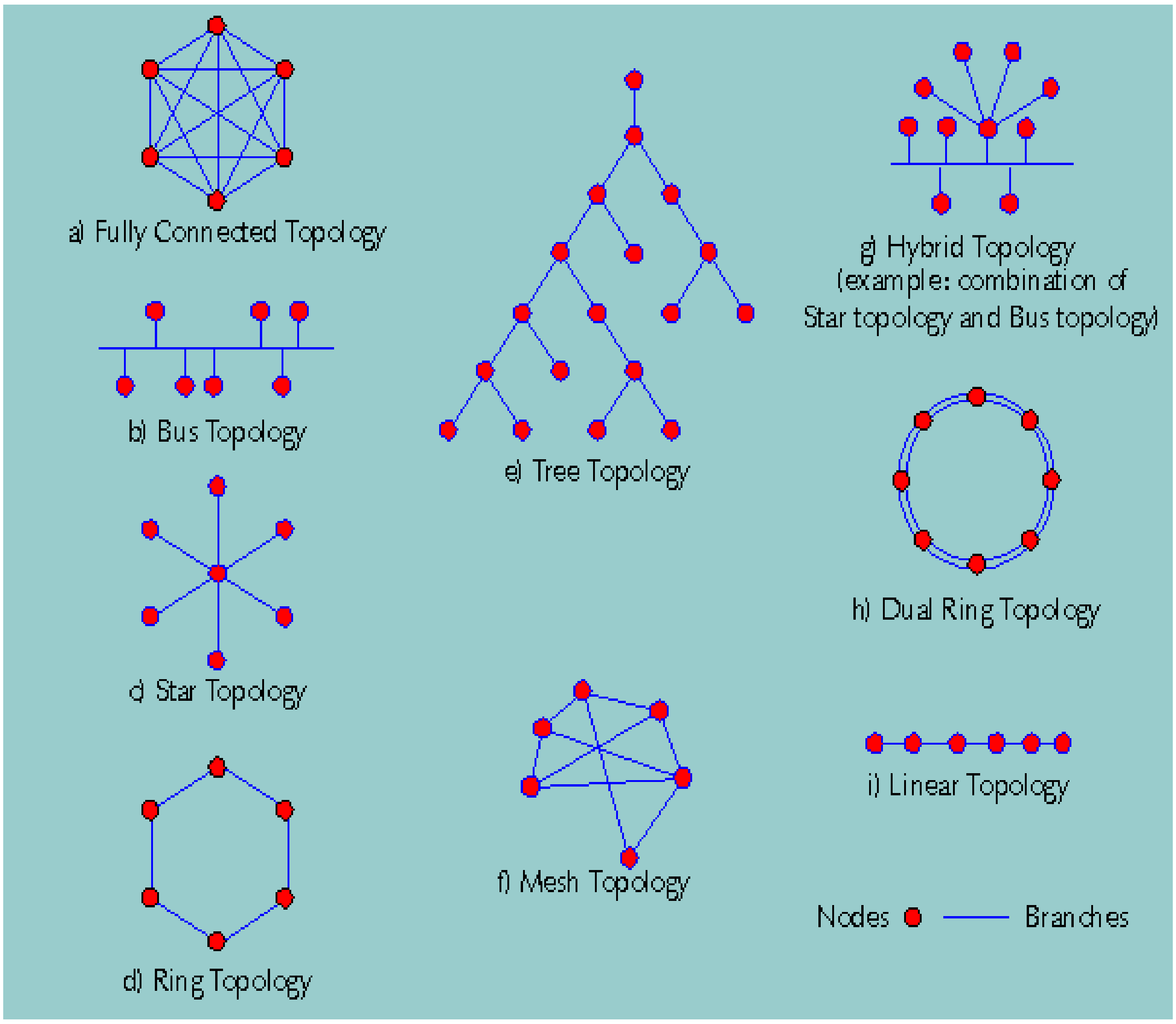
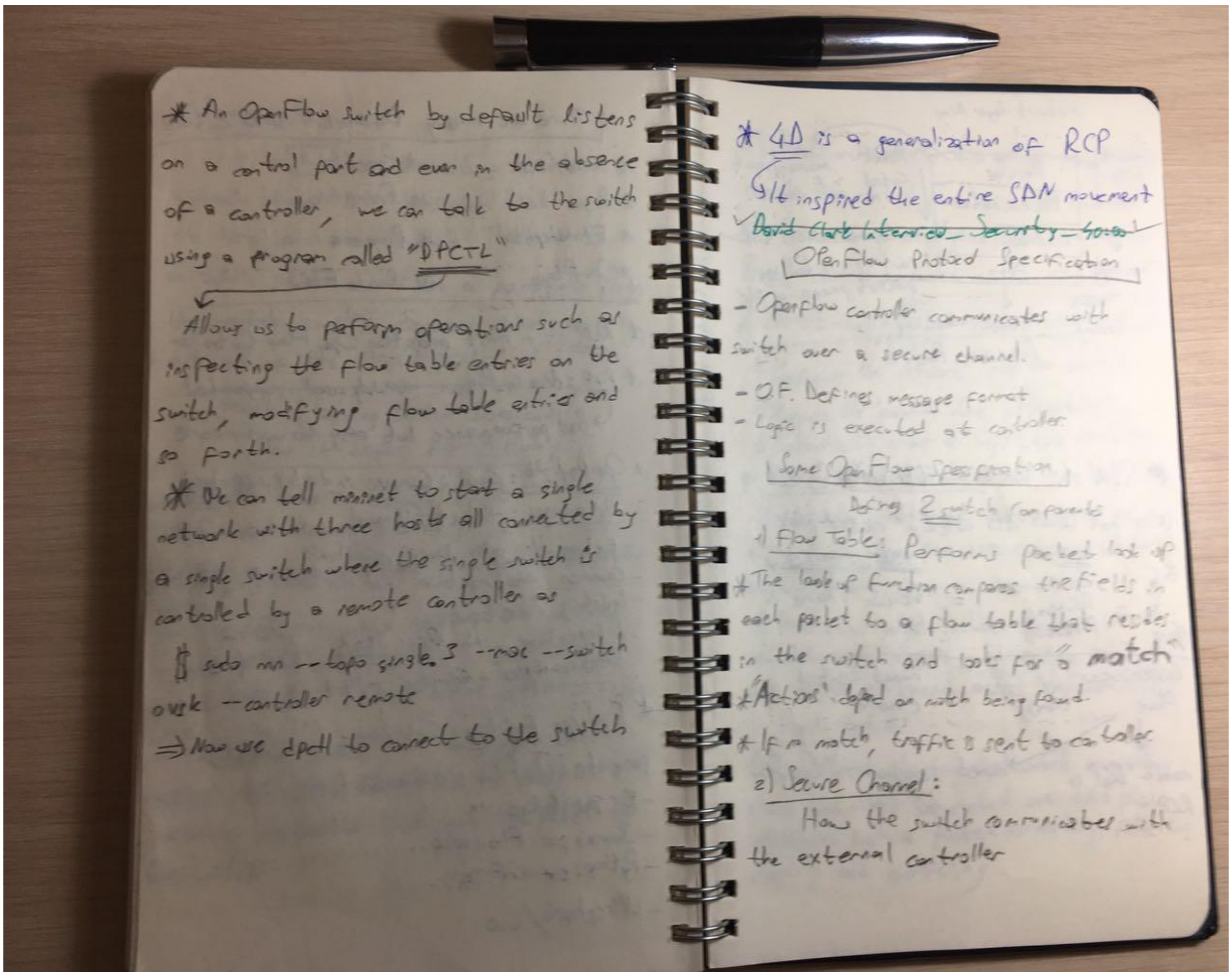


Figure 1: Network Topologies

These classical topologies always exist but controlling the topologies can be changed and get better. This is where SDN technology comes in. The topology, network and the data traffic can be controlled by Software Defined Networking. Network administrators can program any switch in the network by leveraging SDN technology. The fundamental network elements like switches, hubs etc. apply the commands coming from the administrators. In this way, administrators can see all the network traffic from just one center and canalize the traffic accordingly. For instance, in an emergency situation like an earthquake, the administrators can canalize the traffic to the most important areas and they can change the traffic routes by changing the fundamental elements in-use by reprograming the network. Programmability and flexibility of the network provide efficiency to the network. This also brings the advantage of robust cyber security.

Following figures are from my study notes on the internship. The purpose of providing these is to show more clearly how I worked in my internship.



* An OpenFlow switch by default listens on a control port and even in the absence of a controller, we can talk to the switch using a program called "DPCTL"

Allows us to perform operations such as inspecting the flow table entries on the switch, modifying flow table entries and so forth.

* We can tell mininet to start a single network with three hosts all connected by a single switch where the single switch is controlled by a remote controller as

```
$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

⇒ Now use dpctl to connect to the switch

* 4D is a generalization of RCP

It inspired the entire SDN movement

✓ ~~David Clark Interview _ Security _ 40:00~~

OpenFlow Protocol Specification

- Openflow controller communicates with switch over a secure channel.
- O.F. Defines message format
- Logic is executed at controller.

Some OpenFlow Specification

Defines 2 switch components

1) Flow Table: Performs packet look up

* The look up function compares the fields in each packet to a flow table that resides in the switch and looks for a "match"

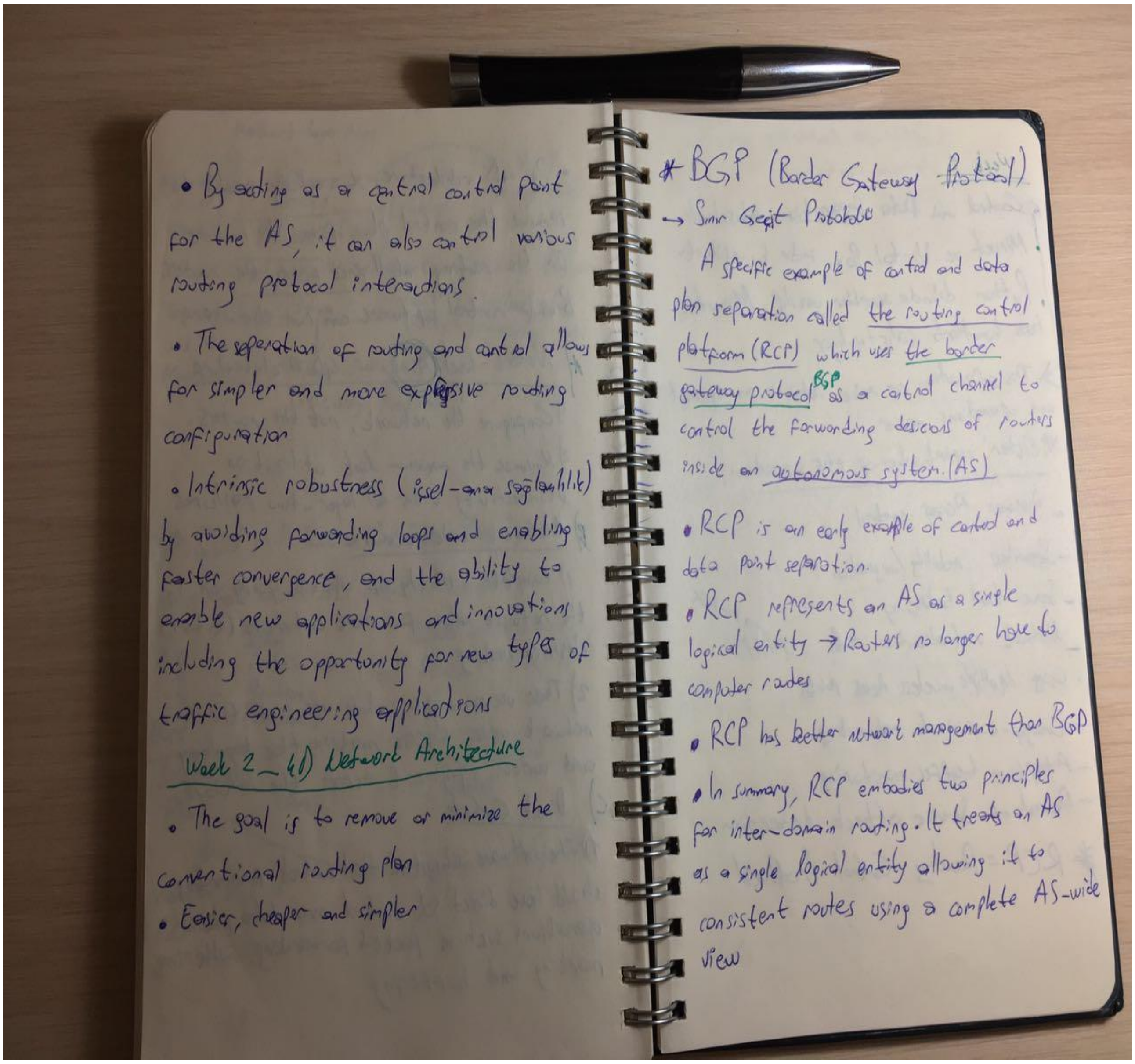
* "Actions" depend on match being found.

* If no match, traffic is sent to controller

2) Secure Channel:

How the switch communicates with the external controller

Figure 2: My Notes



• By acting as a central control point for the AS, it can also control various routing protocol interactions.

• The seperation of routing and control allows for simpler and more expressive routing configuration

• Intrinsic robustness (işsel-ana sağlamlılık) by avoiding forwarding loops and enabling faster convergence, and the ability to enable new applications and innovations including the opportunity for new types of traffic engineering applications

Week 2 _ 4) Network Architecture

• The goal is to remove or minimize the conventional routing plan

• Easier, cheaper and simpler

# BGP (Border Gateway Protocol)

→ Sınır Geçit Protokolü

A specific example of control and data plan separation called the routing control platform (RCP) which uses the border gateway protocol ^{BGP} as a control channel to control the forwarding desicions of routers inside an autonomous system. (AS)

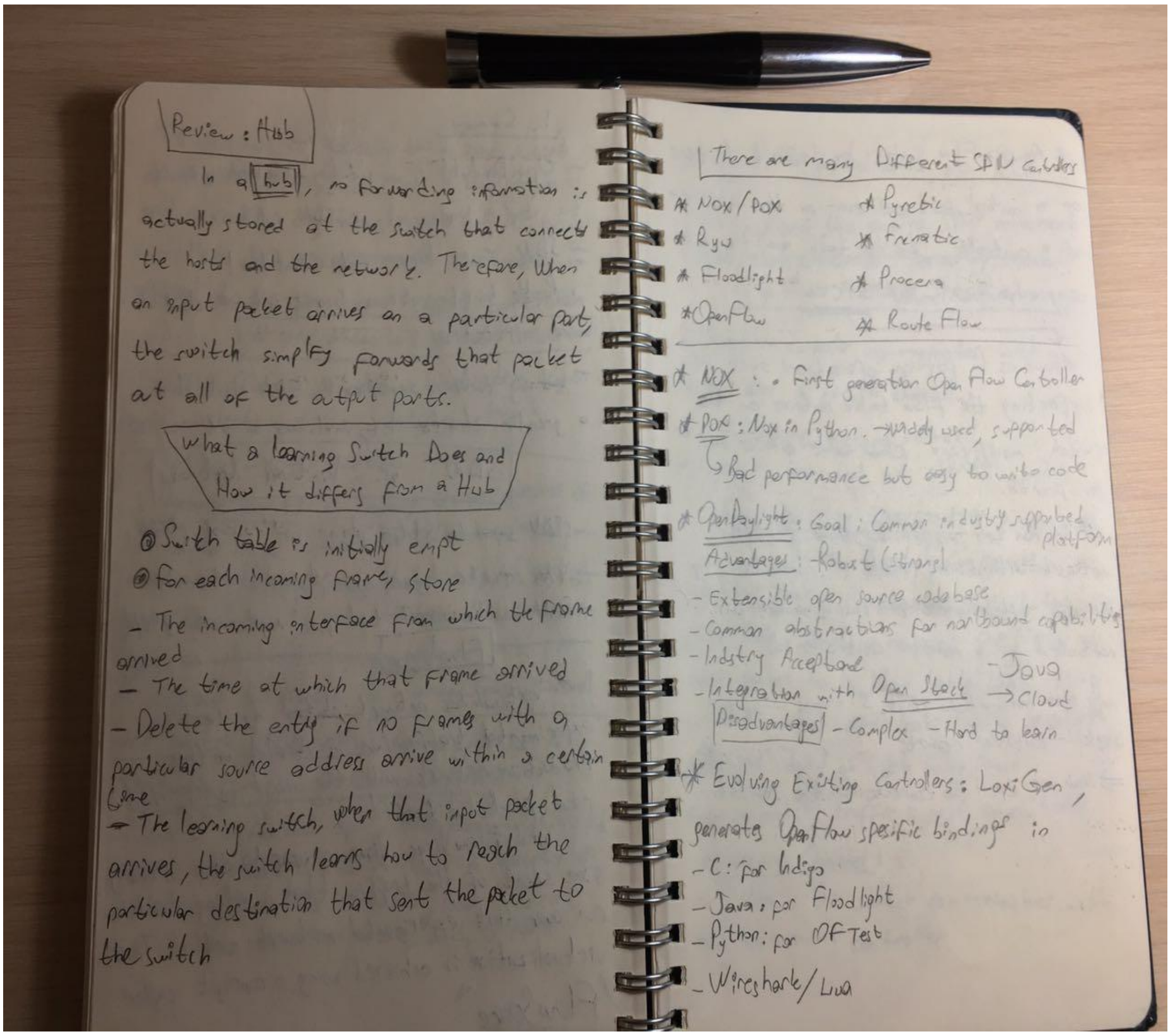
• RCP is an early example of control and data point separation.

• RCP represents an AS as a single logical entity → Routers no longer have to computer rodes

• RCP has better network management than BGP

• In summary, RCP embodies two principles for inter-domain routing. It treats an AS as a single logical entity allowing it to consistent routes using a complete AS-wide view

Figure 3: My Notes (cont'd)



Review : Hub

In a hub, no forwarding information is actually stored at the switch that connects the hosts and the network. Therefore, When an input packet arrives on a particular port, the switch simply forwards that packet at all of the output ports.

What a learning Switch Does and How it differs from a Hub

① Switch table is initially empt
② for each incoming frame, store
- The incoming interface from which the frame arrived
- The time at which that frame arrived
- Delete the entry if no frames with a particular source address arrive within a certain time
- The learning switch, when that input packet arrives, the switch learns how to reach the particular destination that sent the packet to the switch

There are many Different SDN Controllers

* Nox / POX
* Ryu
* Floodlight
* OpenFlow
* Pyretic
* Frenetic
* Procera
* Route Flow

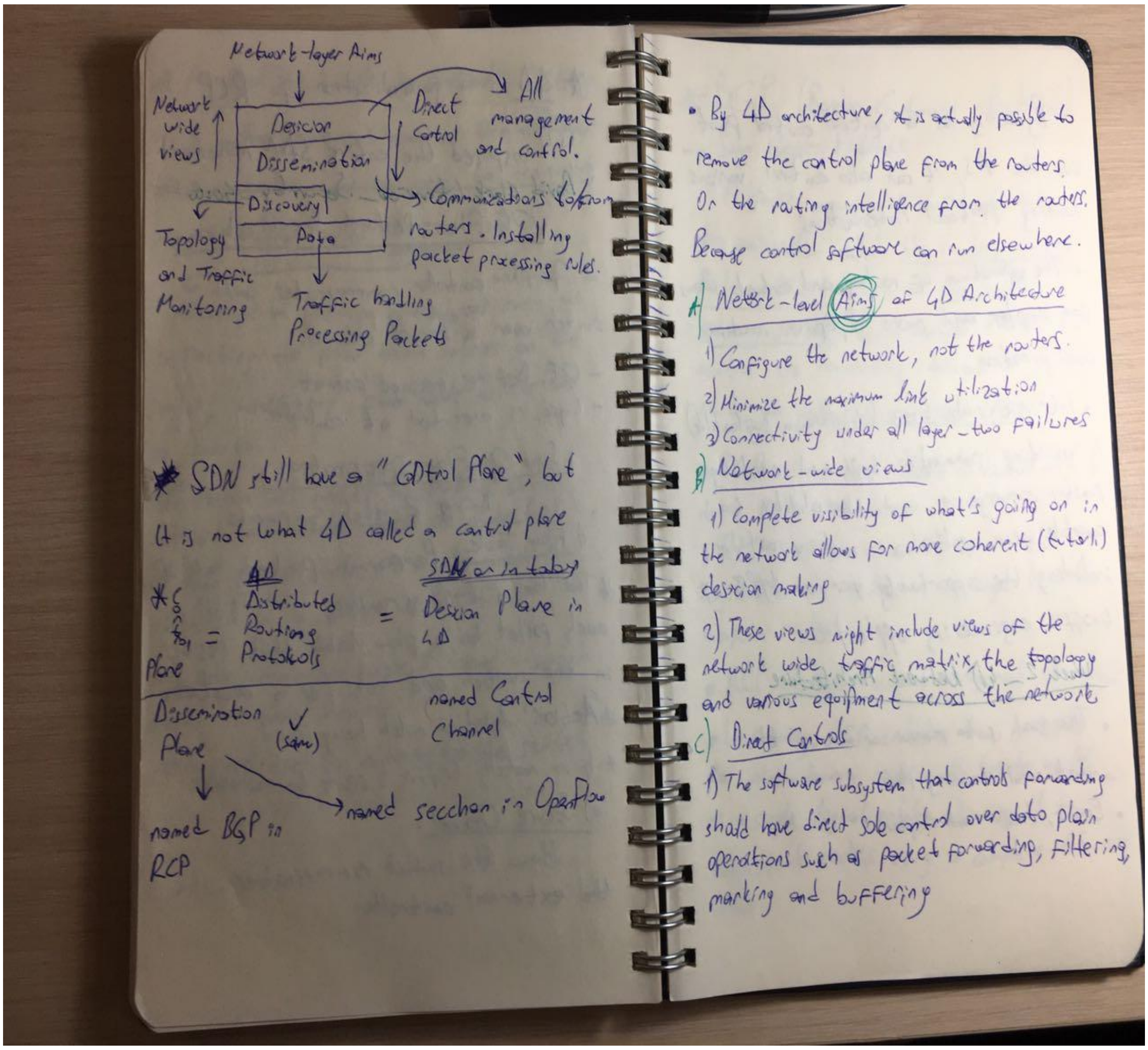
* NOX : • First generation Open Flow Controller
* POX : Nox in Python. → widely used, supported
↳ Bad performance but easy to write code
* OpenDaylight : Goal : Common industry supported platform

Advantages : - Robust (Strong)
- Extensible open source codebase
- Common abstractions for northbound capabilities
- Industry Acceptance
- Integration with Open Stack → Cloud
- Java

Disadvantages - Complex - Hard to learn

* Evolving Existing Controllers : LoxiGen, generates OpenFlow spesific bindings in
- C : for Indigo
- Java : for Floodlight
- Python : for OFTest
- Wireshark / Lua

Figure 4: My Notes (cont'd)



Network-layer Aims

Network wide views

Desicion

Dissemination

Discovery

Data

Direct Control

All management and control.

Communications to/from routers. Installing packet processing rules.

Topology and Traffic Monitoring

Traffic handling Processing Packets

✱ SDN still have a "Control Plane", but
It is not what 4D called a control plane

	4D		SDN on today
✱ Control Plane =	Distributed Routing Protocols	=	Desicion Plane in 4D

Dissemination Plane ✓ (same) named Control Channel

named BGP in RCP

named secchan in OpenFlow

• By 4D architecture, it is actually possible to remove the control plane from the routers. Or the routing intelligence from the routers. Because control software can run elsewhere.

A) Network-level Aims of 4D Architecture

1) Configure the network, not the routers.
2) Minimize the maximum link utilization
3) Connectivity under all layer-two failures

B) Network-wide views

1) Complete visibility of what's going on in the network allows for more coherent (tutarlı) desicion making

2) These views might include views of the network wide traffic matrix, the topology and various equipment across the network

C) Direct Controls

1) The software subsystem that controls forwarding should have direct sole control over data plain operations such as packet forwarding, filtering, marking and buffering

Figure 5: My Notes (cont'd)

The architecture of the Software Defined Networking can be divided into 3 parts: Data Plane, Control Plane and Application Plane.

Basically, Application Plane do abstractions and in this plane so that applications can run as they meant to.

Protocols run on the Data Plane and the network traffic is handled there.

In Control Plane, the routing is done.

In classical networking systems, Data Plane, Control Plane and Application plane are used in the routers and switches. Whereas, in Software Defined Networking, the data and control plane remove the control plane and make the control plane abstract. Instead of using hardware, it implements programs. Therefore, SDN based network administration gets much more flexible than the conventional one.

A high level overview of the SDN architecture is shown in the figure 6 as follows.

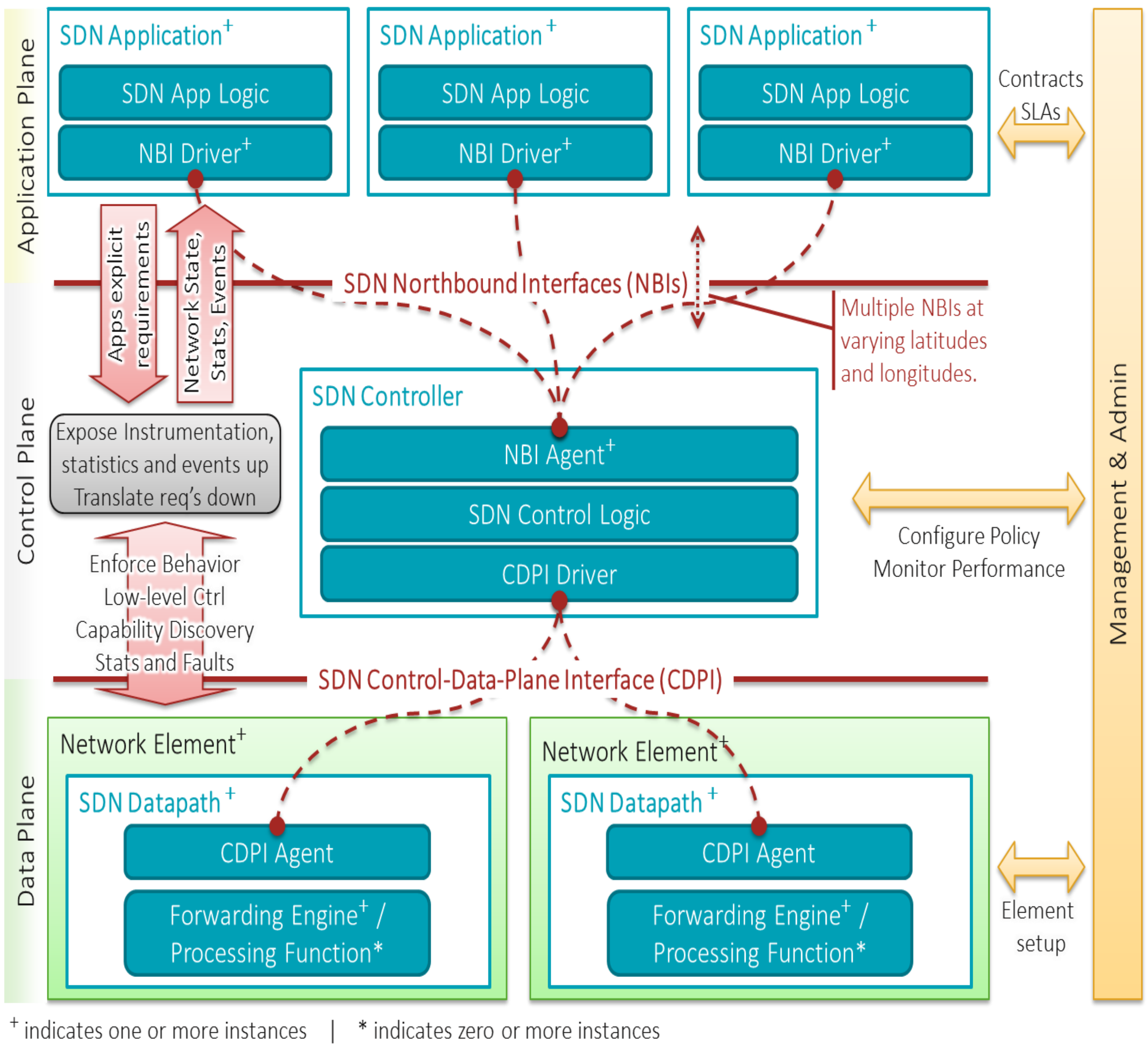


Figure 6: Overview of Software-Defined Networking Architecture

1.2 Open Networking Operating System

The ONOS (Open Networking Operating System) is a kind of operating system that basically controls SDN based networks. It has many abstractions to create applications and services at application plane of the SDN architecture. It is an open source project so that anybody can use it and improve it. The main purpose of the ONOS is to operate the SDN based systems to communicate service providers that is used for scalability, high performance, and high ability and It is written in Java. In Koç University, students learn java in the introduction to programming class. So it was good for me to understand implementations of the algorithms and the networks on the ONOS.

Figure 7: ONOS Logo

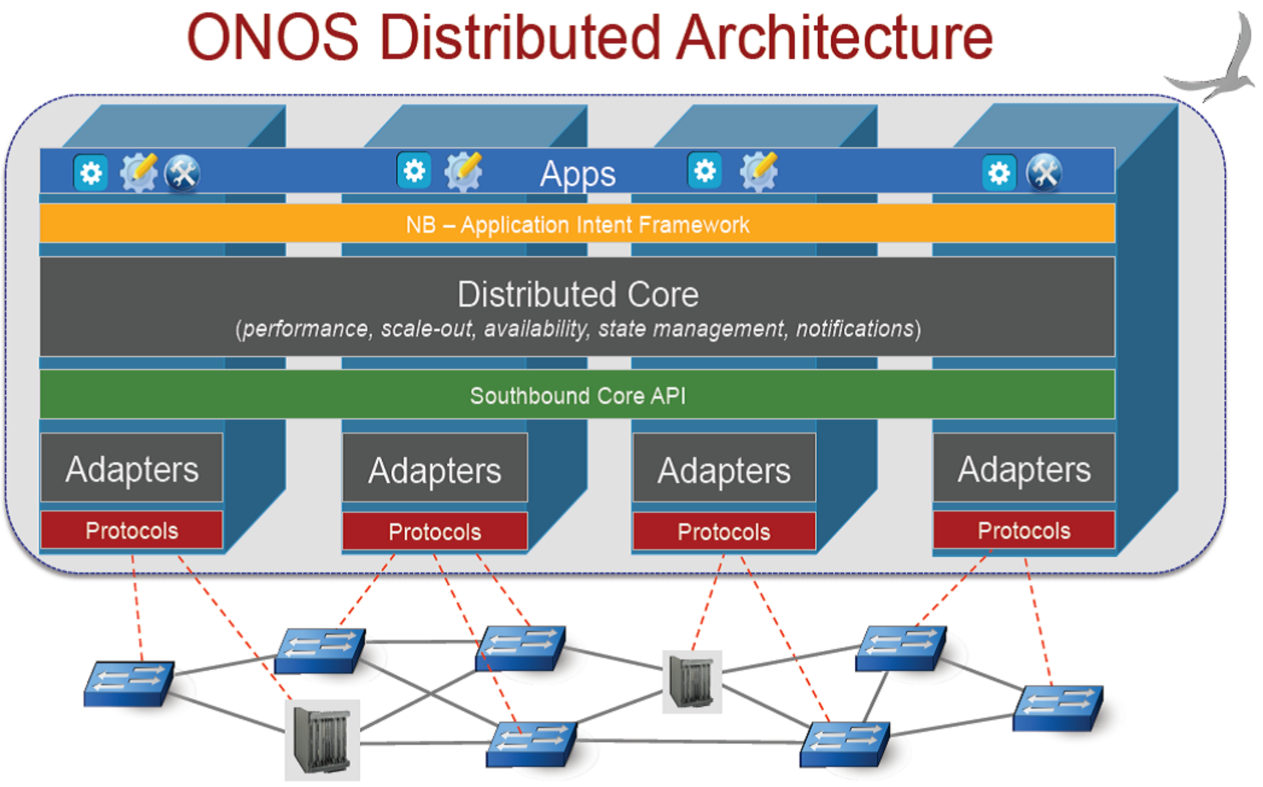

ONOS Distributed Architecture
Apps
NB – Application Intent Framework
Distributed Core
(performance, scale-out, availability, state management, notifications)
Southbound Core API
Adapters
Protocols
Adapters
Protocols
Adapters
Protocols
Adapters
Protocols

Figure 8: ONOS Distributed Architecture

1.3. Shortest Path Algorithms and My Responsibilities

In my internship, it was important to learn how Breadth First Search, Depth First Search, and Dijkstra's algorithms work. Because, in the project, which my supervisor involved, the shortest paths shall be found in an SDN based network and those paths shall satisfy some constraints such as bandwidth and network speed. At the end of the day, the company developed the algorithm which chooses the shortest path for data flow with taking into account the constraints. I was involved with the processes as doing research on how to develop a shortest path algorithm with different constraints, but I can provide no more information about the algorithm because of the non-disclosing liabilities. In this section, those three algorithms are mentioned briefly, the team chose to continue with an algorithm close to Dijkstra's Algorithm.

1.3.1. Breadth First Search Algorithm

Breadth First Search algorithms firstly, explores the neighbor nodes Secondly, moves to the next level neighbor nodes. For instance, a transportation company wants to use the shortest way from Frankfurt to München in order to spend less gas. Then, the result should be following.

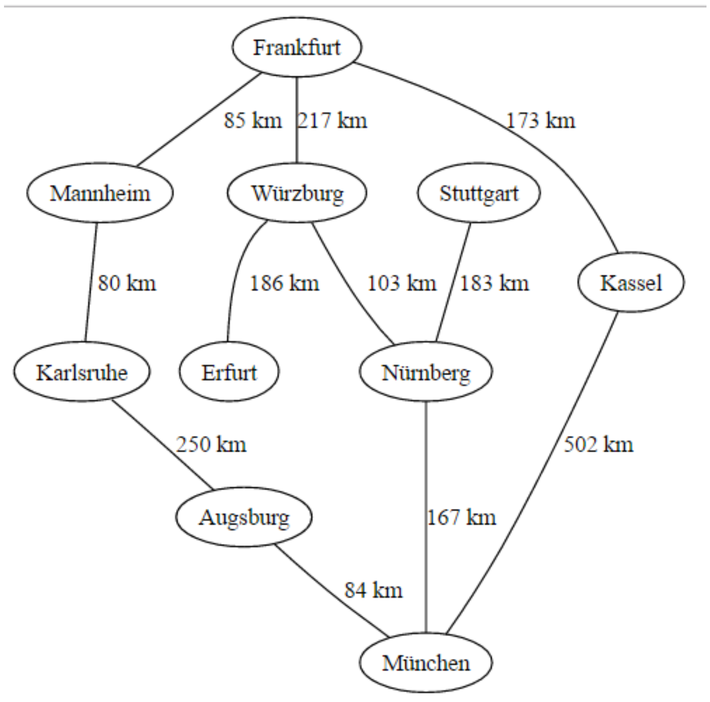


Figure 9: An example map of Germany with some connections between cities

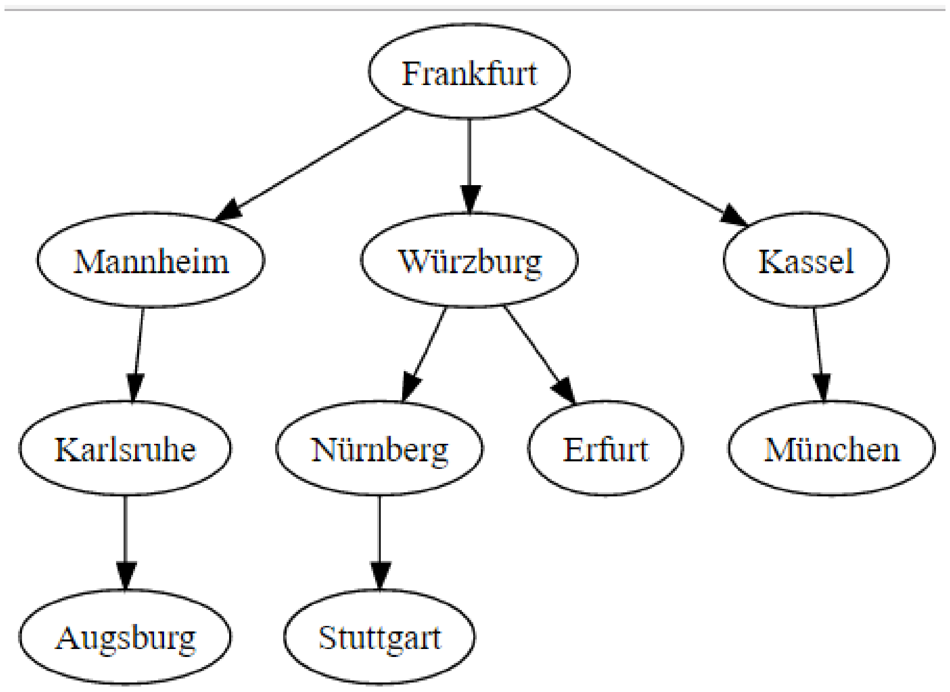


Figure 10: The breadth-first tree obtained when running BFS on the given map and starting in Frankfurt

According to the Breadth First Search algorithm, the company chooses the way from Frankfurt to Kassel, then Kassel to München as shown in the figure 10.

1.3.2. Depth First Search Algorithm

Depth First Search algorithm firstly explores as far as possible along each branch and then come to neighbor nodes. For instance, in the Figure 11, we have a graph in a binary tree structure. According to Depth First Search algorithm, the preferred way should be 3,2,7,1,9,8,5.

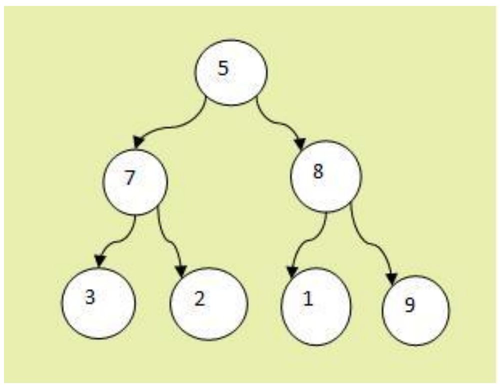


Figure 11: Binary Tree

1.3.3. Dijkstra's Shortest Path Algorithm

Dijkstra's algorithm finds the shortest path in a graph. It basically finds the shortest path from one to other nodes in a graph. It is good to give an example to illustrate the way of thing of Dijkstra's algorithm. For instance, we start at node A and then the algorithm thinks that there is no access to any other nodes so the cost for the nodes is infinity shown at the following figures. Next, the algorithm updates the cost by going around the nearby neighbor nodes and then it chooses the shortest distance among the costs. The process continues in every step like this till the reach destination.

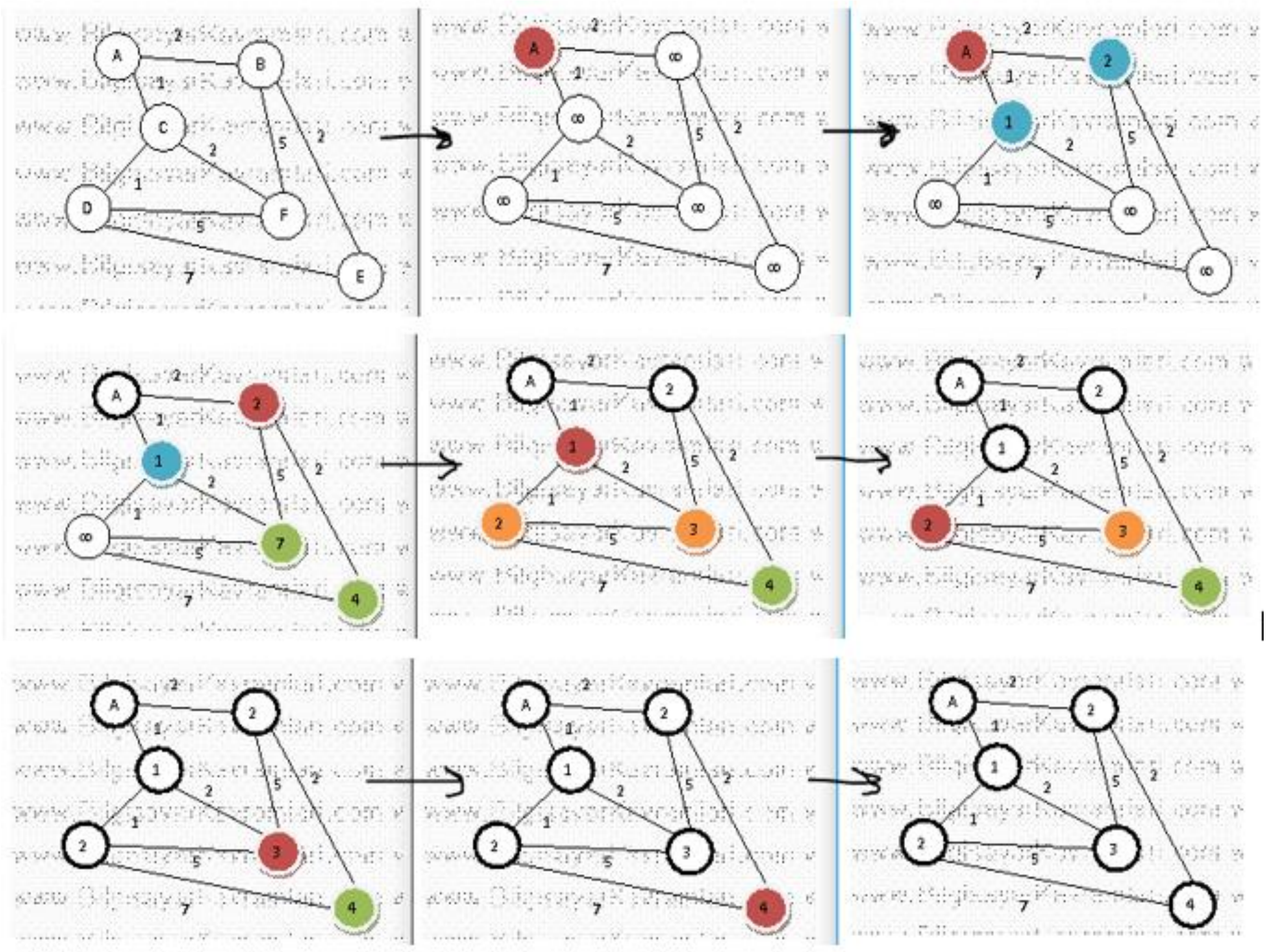


Figure 12: Dijkstra's Algorithm

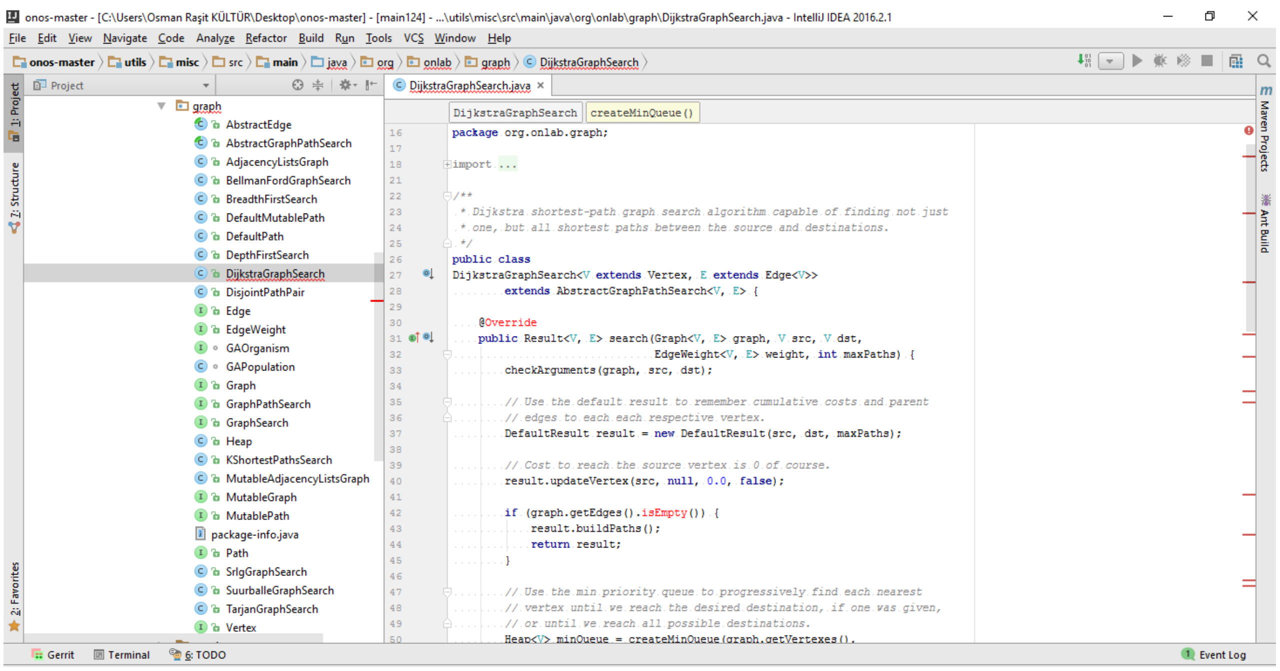
In summary, these tree algorithms are used and considered to find the shortest paths in an SDN based networks graph. Switches and hubs are thought as nodes in the network. It was important to understand the basic concepts of them.

2. ONOS Source Code

In the Internship, it was good to see how the algorithms I studied are implemented in an SDN based network. To see the implementation, In Ubuntu 14.04, I installed IntelliJ IDE and then using terminal I cloned the ONOS source code so that I can change some piece of codes and do some testing in the code. To download the source code, it is enough to copy and paste the following command:

```
$ git clone https://gerrit.onosproject.org/onos
```

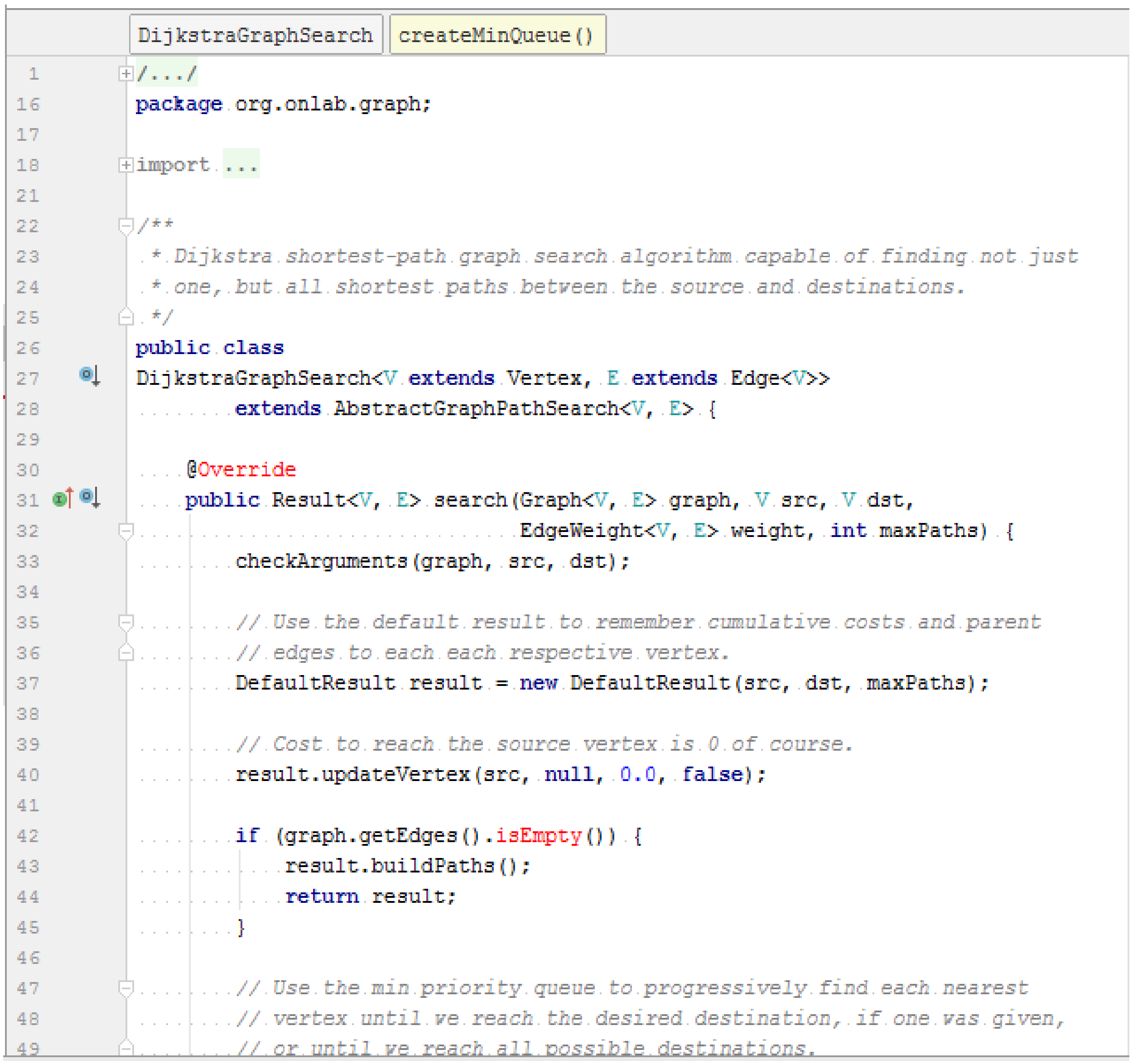
After the required installation, there is an implementation of Dijkstra's algorithm under the graph folder. I provide the following figures to see the development environment clearly



```java
package org.onlab.graph;

import ...

/**
 * Dijkstra shortest-path graph search algorithm capable of finding not just
 * one, but all shortest paths between the source and destinations.
 */
public class
DijkstraGraphSearch<V extends Vertex, E extends Edge<V>>
        extends AbstractGraphPathSearch<V, E> {

    @Override
    public Result<V, E> search(Graph<V, E> graph, V src, V dst,
                               EdgeWeight<V, E> weight, int maxPaths) {
        checkArguments(graph, src, dst);

        // Use the default result to remember cumulative costs and parent
        // edges to each each respective vertex.
        DefaultResult result = new DefaultResult(src, dst, maxPaths);

        // Cost to reach the source vertex is 0 of course.
        result.updateVertex(src, null, 0.0, false);

        if (graph.getEdges().isEmpty()) {
            result.buildPaths();
            return result;
        }

        // Use the min priority queue to progressively find each nearest
        // vertex until we reach the desired destination, if one was given,
        // or until we reach all possible destinations.
        Heap<V> minQueue = createMinQueue(graph.getVertexes(),
```

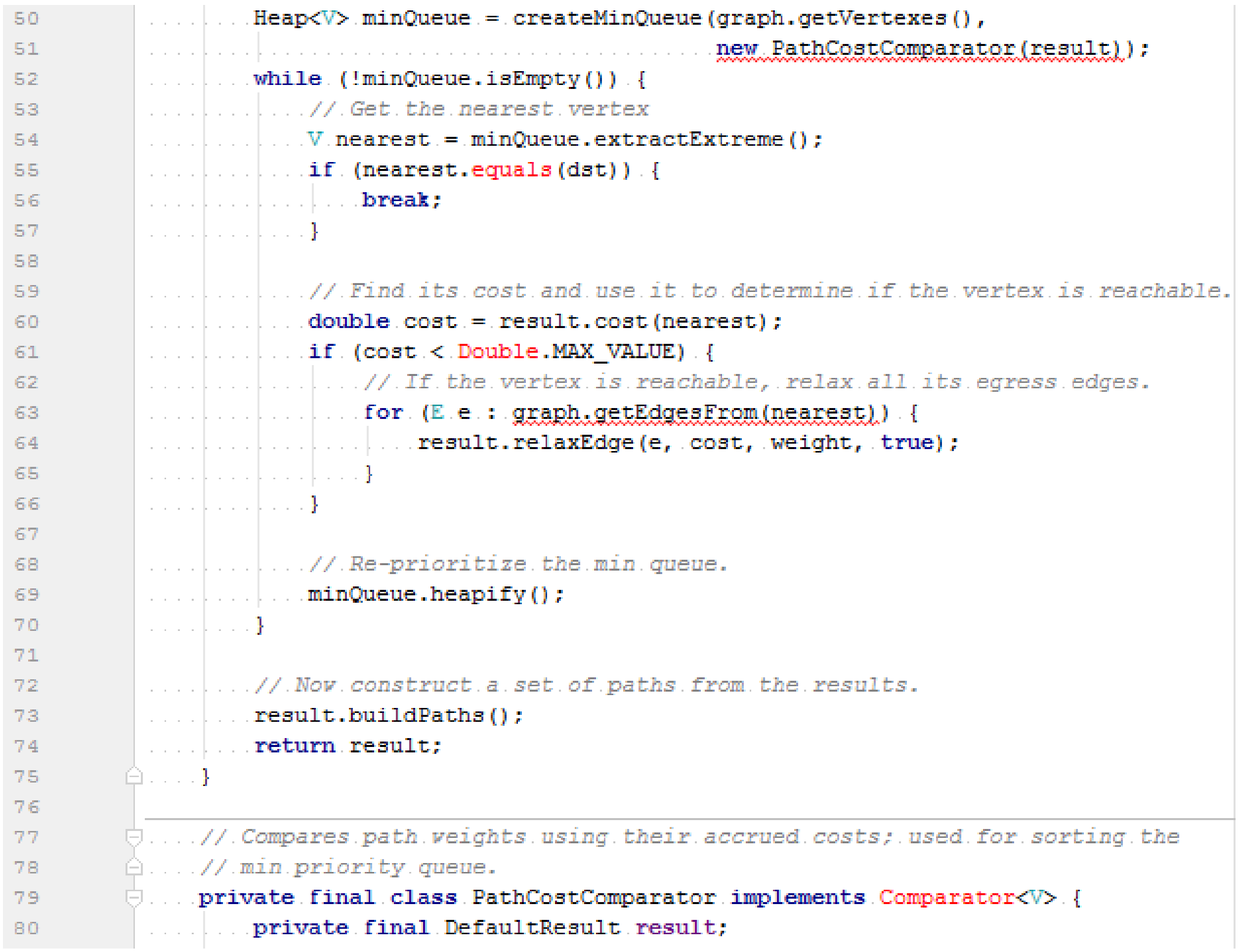
Figure 13: ONOS Source Code

Following three figures shows the implementation of Dijkstra’s algorithm in ONOS. This is just to give an insight. To understand the implementation of Dijkstra’s algorithm better, one needs to pay attention on the abstractions, inheritances and function callings.



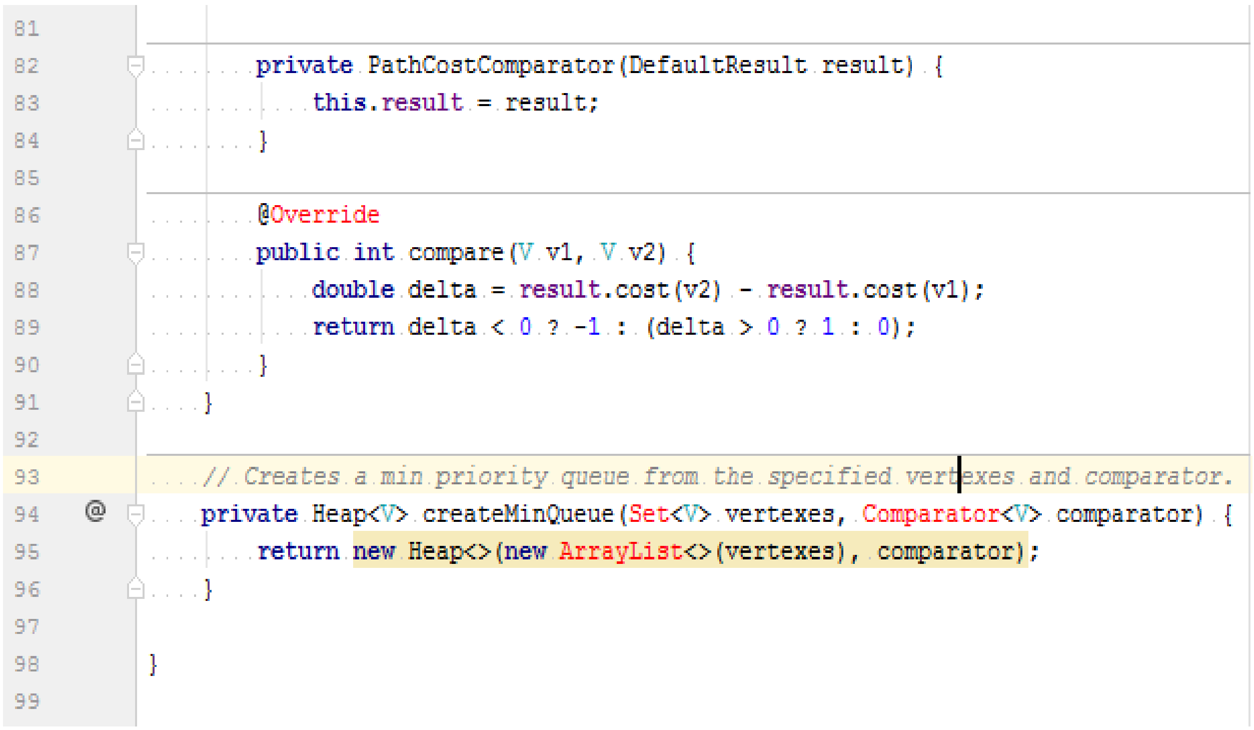
```
/.../
package org.onlab.graph;

import ...

/**
 * Dijkstra shortest-path graph search algorithm capable of finding not just
 * one, but all shortest paths between the source and destinations.
 */
public class
DijkstraGraphSearch<V extends Vertex, E extends Edge<V>>
        extends AbstractGraphPathSearch<V, E> {

    @Override
    public Result<V, E> search(Graph<V, E> graph, V src, V dst,
                               EdgeWeight<V, E> weight, int maxPaths) {
        checkArguments(graph, src, dst);

        // Use the default result to remember cumulative costs and parent
        // edges to each each respective vertex.
        DefaultResult result = new DefaultResult(src, dst, maxPaths);

        // Cost to reach the source vertex is 0 of course.
        result.updateVertex(src, null, 0.0, false);

        if (graph.getEdges().isEmpty()) {
            result.buildPaths();
            return result;
        }

        // Use the min priority queue to progressively find each nearest
        // vertex until we reach the desired destination, if one was given,
        // or until we reach all possible destinations.
```

Figure 14.1: Dijkstra’s Algorithm Implementation in ONOS



```
        Heap<V> minQueue = createMinQueue(graph.getVertexes(),
                                          new PathCostComparator(result));
        while (!minQueue.isEmpty()) {
            // Get the nearest vertex
            V nearest = minQueue.extractExtreme();
            if (nearest.equals(dst)) {
                break;
            }

            // Find its cost and use it to determine if the vertex is reachable.
            double cost = result.cost(nearest);
            if (cost < Double.MAX_VALUE) {
                // If the vertex is reachable, relax all its egress edges.
                for (E e : graph.getEdgesFrom(nearest)) {
                    result.relaxEdge(e, cost, weight, true);
                }
            }

            // Re-prioritize the min queue.
            minQueue.heapify();
        }

        // Now construct a set of paths from the results.
        result.buildPaths();
        return result;
    }

    // Compares path weights using their accrued costs; used for sorting the
    // min priority queue.
    private final class PathCostComparator implements Comparator<V> {
        private final DefaultResult result;
```

Figure 14.2: Dijkstra's Algorithm Implementation in ONOS



```

        private PathCostComparator(DefaultResult result) {
            this.result = result;
        }

        @Override
        public int compare(V v1, V v2) {
            double delta = result.cost(v2) - result.cost(v1);
            return delta < 0 ? -1 : (delta > 0 ? 1 : 0);
        }
    }

    // Creates a min priority queue from the specified vertexes and comparator.
    private Heap<V> createMinQueue(Set<V> vertexes, Comparator<V> comparator) {
        return new Heap<>(new ArrayList<>(vertexes), comparator);
    }

}
```

Figure 14.3: Dijkstra's Algorithm Implementation in ONOS

3. Summary and Concluding Remarks

I am glad to have an internship in R & D department at Argela. I want to be an engineer who do Research & Development. So, it was a good opportunity to experience how an R & D engineer works.

I also learn many valuable things from computer science. My major is electrical & electronics engineering, however. Nowadays, technology is getting complicated and interdisciplinary. It was good to know that an electrical & electronics engineer who wants to do R & D should understand from computer science well. I hope I will lead my career by knowing this.

4. References

Figure 1: Network Topologies. Retrieved from http://www.its.bldrdoc.gov/fs-1037/images/nettopoc.gif

Figure 6: Overview of Software-Defined Networking Architecture. Retrieved from https://commons.wikimedia.org/wiki/File:SDN-architecture-overview-transparent.png#/media/File:SDN-architecture-overview-transparent.png

Figure 7: ONOS Logo. Retrieved from http://onosproject.org/

Figure 8: ONOS Distributed Architecture. Retrieved from https://kshahzadblog.files.wordpress.com/2015/04/onos-distributed-architecture.png

Figure 9: An example map of Germany with some connections between cities. Retrieved from https://en.wikipedia.org/wiki/Breadth-first_search#/media/File:MapGermanyGraph.svg

Figure 10: The breadth-first tree obtained when running BFS on the given map and starting in Frankfurt. Retrieved from https://en.wikipedia.org/wiki/Breadth-first_search#/media/File:GermanyBFS.svg

Figure 11: Binary Tree. Retrieved from http://www.bilgisayarkavramlari.com/2008/11/13/derin-oncelikli-arama-depth-first-search/

Figure 12: Dijkstra’s Algorithm. Retrieved from http://bilgisayarkavramlari.sadievrenseker.com/2010/05/13/dijkstra-algoritmasi-2/

Figure 13: ONOS Source Code. Retrieved from https://github.com/opennetworkinglab/onos

Figure 14.1, 14.2, and 14.3: Dijkstra’s Algorithm Implementation in ONOS. Retrieved from https://github.com/opennetworkinglab/onos