Journal of Universal Computer Science, vol. 28, no. 6 (2022), 648-669
submitted: 16/1/2022, accepted: 27/4/2022, appeared: 28/6/2022 CC BY-ND 4.0

Traffic Optimization with Software-Defined Network Controller on a New User Interface

Derya Yiltas-Kaplan
(Istanbul University-Cerrahpasa, Istanbul, Turkey
https://orcid.org/0000-0001-8370-8941, deryayiltas@gmail.com)

Abstract: Software-defined networking (SDN) has emerged as a solution to the cumbersome structures of classical computer networks. It separates control and data planes to give independence to devices with respect to either traffic routing or network management. The two isolated planes communicate with each other via the help of software modules, which are located in an SDN controller, such as Floodlight, NOX, or Ryu. In this study, Floodlight is used and an SDN topology with 20 switches is constructed with Python code in Mininet. All algorithms have been coded with Java. The default routing algorithm in Floodlight is Dijkstra's algorithm. Four different network optimization algorithms, namely Bellman-Ford, Ford-Fulkerson, Auction, and Dual Ascent algorithms, are utilized in ordinary network routing instead of Dijkstra's algorithm. None of these four algorithms were used in SDN before and network implementations using Ford-Fulkerson, Auction, or Dual Ascent algorithms were scarce in the literature. The results are analyzed with multiple types of normalization on a new user interface communicating with Floodlight part via HTTP requests. There has not been a user interface that performs the same operations in Floodlight. In the future, this study may possibly be improved with considering normalization processes based on various proportions among the metric values and accounting the computational time of the algorithms.

Keywords: Auction algorithm, Bellman-Ford, Dual ascent, Floodlight, Ford-Fulkerson, SDN, SDN GUI, SDN routing, Traffic optimization
Categories: C.2.2, C.2.3, C.4, H.5.2
DOI: 10.3897/jucs.80625

1 Introduction

Computer networks consist of several devices and architectures, the basic examples of which are switches, routers, data hubs, and firewalls (hardware). The number of devices increases with the number of data centers. In such structures, network management and organization become difficult and complicated; for example, increasing the number of endpoint devices in networks causes high probability of occurrence of connections and message transmissions. As a result, network traffic grows and some difficulty in processing big data arises. Similarly, the routing tables used for end-to-end packet transmission come up with confusion in classical network environments. In recent years, there have been many attempts, such as proposing fundamental changes in network structures, to eliminate those difficulties. One of the most important improvements in this area is the approach of software-defined networking (SDN). SDN presents advantages of hardware independence, simplification of network control and applications with software, flexibility, dynamic network configuration, and system scalability [Akbaş et al. 2016], [Özbek et al. 2020]. One of the most popular SDN

architectures in the world is B4 wide area network of Google [Yiltas-Kaplan 2019]. There is also an architecture of software-driven wide area network announced by Microsoft researchers, that controls the data centers from one center [Hong et al. 2013].

The layers in SDN construction are named as Infrastructure, Control, and Application, as shown in Figure 1 [Open Networking Foundation 2014]. The infrastructure layer involves network elements, namely switches, that represent their operations with the southbound interface coming from the controller and execute the controller commands. The control layer contains tasks related to the controller and serves as the center of the network logically. There are several software examples, such as Floodlight, working as the controller on the control plane. The top layer of SDN, namely the application layer, incorporates all the SDN applications and has connections to the controller via the northbound interface to provide various network requirements [Akçay and Yiltas-Kaplan 2019], [Open Networking Foundation 2014], [Yiltas-Kaplan 2019].

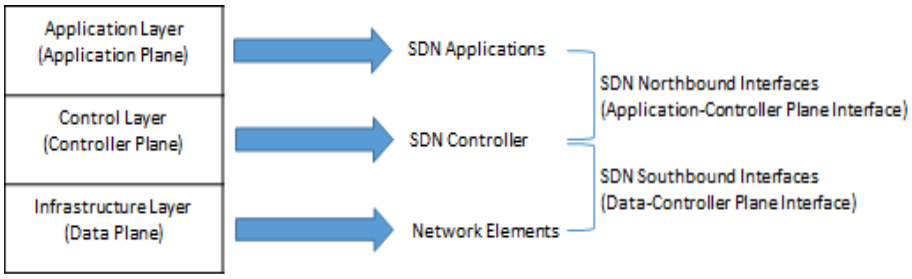

Figure 1: SDN layers

The main purpose in this study is to adapt various algorithms into Floodlight software to forward packet flows and find the least-cost paths. The algorithms employed in this paper had not been used in SDN applications before. Moreover, there had been hardly any regular implementation, like the ones used in this study, constructed in traditional computer networks. The default routing algorithm in Floodlight is Dijkstra's algorithm (DA). The algorithms evaluated in this paper include Bellman-Ford algorithm (BFA), Ford-Fulkerson algorithm (FFA), Auction algorithm (AA), and Dual Ascent algorithm (DAA). The theoretical definitions and mathematical representations of these algorithms appear in the literature [Bertsekas 1998], but they have not been implemented in any SDN topologies before. In this study, they are coded for penetrating into the Floodlight system in line with the study's fundamental goal of applying various optimization algorithms with different metrics in the Floodlight modules for the first time and improving the routing performance. This is a process that requires common data structures with Floodlight. Furthermore, it is critical to consider that the network packets are transmitted as flows in SDNs, in contrary to traditional computer networks. Thus, in this study, different network optimization algorithms are performed instead of DA by adapting to the Floodlight data structures with the purpose of representing various performance measurements visually.

Like DA, BFA is a well-defined and preferred solution for the shortest path problem. On the other hand, FFA is involved in the algorithms, which are proposed to compute the maximum flow, and AA and DAA are for minimum cost flow [Bertsekas 1998]. In this study, the applications of FFA, AA, and DAA in routing operations and

their multi-valued structures will be studied and discussed for the first time. Hence, the present study marks remarkable progress in routing optimization of two metrics while utilizing normalization rules.

The two metrics extracted from Floodlight are latency and bandwidth. The sample SDN topology in this study has 20 switches. After the coding steps of the algorithms and normalization processes, a graphical user interface (GUI) is created to provide an analysis of overall results. With this GUI, it is possible to investigate the results based on various algorithms and on each normalization type independently. Thus, the present study is organized to pioneer work on the Floodlight project by presenting ways of module changes for routing operation.

The rest of the paper is designed as follows: Section 2 focuses on the description of the software materials, algorithms, normalization process, GUI structure, and the test process. Section 3 presents the experimental results of cost and flow calculations. Finally, Section 4 provides the conclusion and future directions for this study.

2 Materials and Method

2.1 Floodlight and Mininet

SDN controller takes the central role between the southbound and the northbound interfaces. This implies that the services and the applications in the networks are dissociated from the equipment. The controllers manage the routing operations and the flows between switches via the southbound interface and provide connections to the applications via the northbound interface. Actually, the control and the routing operations diverge from each other, as shown in the southbound interface in Figure 1, and therefore, the programming of the control part can be performed directly [Özbek et al. 2020].

Floodlight is one of the most commonly used SDN controllers, and it is an important Java-based project, which has an open-source software approach [Wang 2018]. Any Java module can be easily integrated into Floodlight, which covers network applications, services, and modules supported by the REST API. Some modules in this part are Topology, Statistics, Device Manager, Load Balancer, and Web UI. Floodlight can perform application tests with both physical and virtual OpenFlow-enabled switches. Path computations for the flow transmissions between the switches are made by Topology Manager/Router, which is a core service in Floodlight. DA is the default routing algorithm for obtaining the least-cost path in the network within the class of TopologyInstance.java in the Topology module. Here, the input of each path computation consists of all the nodes in the network, the source and destination node pair, the single metric cost information for each link. The output is the shortest path with a cost value. REST API accomplishes several operations such as insertion, deletion, and interrogation of the flow inputs [Ilhan and Yiltas-Kaplan 2020], [Özbek et al. 2020].

It is highly practical to run the virtual network structures constructed with Mininet emulator for the topologies or the traffic and the service computations in SDN [Özbek et al. 2020]. It is because Mininet works on Linux kernels, a Linux-based operating system is set up either on the virtual machine or directly into the host to provide correct Floodlight and Mininet connection. For the software part in this study, various versions

were tested, and Ubuntu 18.04 was approved for Floodlight-Mininet applications at the end. Java 8 was installed as mentioned in the Floodlight setup documents, and IntelliJ IDEA was used instead of Eclipse. The Python programming language was utilized for the commands in Mininet side [Ilhan and Yiltas-Kaplan 2020] for constructing a sample network with 20 switches.

2.2 Optimization Algorithms

Initially, performance of the optimization algorithms, particularly the DA-related part in the Floodlight modules, has been investigated. After this, the algorithms have been coded and integrated into the correct places. The main part for this procedure in Floodlight is the net.floodlightcontroller.topology module in the directory src/main/java. Each algorithm's codes have been added into the related parts of the TopologyInstance.java file in the Topology module. The following subsections give more details about theoretical descriptions of the algorithms and their implementation plans regarding the software part of this study.

2.2.1 Bellman-Ford algorithm

Basically, a computer network is defined with the graph G=(V, E), where V is the set of network nodes and E the edge set. The graph is modified as G=(V, E, w) if the weight or cost function w applies to the edges. The terms weight or cost will be used alternatively throughout the study. Suppose that there are no negative-weight cycles in graph G, and d is the metric value (such as distance value) for each link. The pseudocode of BFA will then be written as in Figure 2 [Klappenecker 2022].

```
Input: G = (V, E, w)
for ∀v ∈ V {
   d[v] = infinite; parent[v] = none; }
d[s] = 0; parent[s] = s;
for i := 1 to |V| - 1 {
    for each (u,v) in E {
       if (d [u] + w(u,v) < d [v]) then { d [v] := d [u] +
w(u,v); parent[v] := u; }
    } }
```

Figure 2: Pseudocode of BFA

BFA returns the shortest path lengths from a source to all the other nodes in the graph. The result also covers whether there is a negative-weight cycle, which is a circular cycle with the sum of link metrics smaller than 0 [Demaine and Wenk 2022].

2.2.2 Ford-Fulkerson algorithm

FFA uses the logic of the breadth-first search (BFS). This study proposes a modified version of FFA (dFFA) to perform the routing and the flow control in the network. In

dFFA, initially, all the links in the network are assigned positive weight values. A specific number of paths is obtained between a predetermined source and destination pair in the network topology. The link weights of these paths are computed from the latency and bandwidth values, both derived from Floodlight. The paths, that have the minimum weights, are deemed suitable for network flows.

The coding of the algorithm can be divided into two procedures. The first one involves BFS process, and the second one obtains the most suitable paths. Figures 3 and 4 show the pseudocodes of these procedures respectively.

```
1 Define a variable q in the Queue type.

2 Define a variable path in the ArrayList type.

3 Define a variable matrix in the two-dimensional ArrayList type.

4 Insert the first node into the path.

5 Insert path to q.

6 Do the following until q becomes empty:

    6.1 path <= q.poll()  { Return the first variable of q and then delete it from
        q. }
    6.2 Define last and assign the reference of the last element of the path list
        to last.
    6.3 If last is the destination node and the method that returns the Random
        value gives true, then add the path to the matrix.
    6.4 If the number of elements in the matrix reaches 50, then end the
        process.
    6.5 Return the nodes starting by the node last sequentially and do the
        following:

        6.5.1 Give the name current to the next node and do the following if it
        has not been visited before:

            6.5.1.1 Define a new variable newPath in ArrayList<Integer> type.

            6.5.1.2 Insert current to newPath.
            6.5.1.3 Insert newPath to q.

7 Return the variable matrix.
```

Figure 3: Pseudocode of BFS part of dFFA

1 Define an *int* variable as *minPrefIndex* and initialize it with 0.

2 Define a *double* variable as *costValue*.

3 Do the following for each element (namely *onePath*) in two-dimensional array *matrix*:

3.1 Compute the weight value of each element in one dimensional array of *onePath*.

3.2 Between all *onePath* elements, assign the smallest weighted path to the variable *bestOption*.

4 Return *bestOption* as the solution set.

Figure 4: Pseudocode of the best path selection part of dFFA

In step 6.3 of Figure 3, a method returning Random value means an operation occurring with a determined probability. During the process in which the paths are obtained, a control class is constructed so that this method returns true for each path with a probability of 25%. The main goal of this operation is to prevent finding the best paths limited only to specific regions of the topology. BFS finds paths according to the starting point of the topology; therefore, its results can only be accumulated for some regions.

After the steps shown in Figure 3, a limited number of paths are obtained. In this study, this number is 50. These paths are stored in two-dimensional ArrayList of matrix having ArrayList<Integer> elements, each of which presents an overall path. After completing the procedure in Figure 3, the second pseudocode in Figure 4 is used to find the best path among those 50 paths based on the weight computations.

In accordance with the normalization rule, which is based on separate computations being performed independently, the weight computation in step 3.1 of Figure 4 is performed separately.

2.2.3 Auction algorithm

To the best of our knowledge, the current research literature reveals a lack of computer network implementation with AA. To fill that gap in the literature, seminal papers and books on AA were investigated, and significant information was drawn from four of the most effective studies [Becker et al. 2016], [Bertsekas 1991], [Bertsekas 1992], [Bertsekas 1998]. The steps of the algorithms were put in order, and the coding was planned according to this information. Consequently, numerical values of all the network nodes were stored in vectors during the implementation of AA. The relevant node was either inserted into, or deleted from, the solution set, after the required comparisons. The insertion of the destination node to the solution set is the termination criterion of this procedure.

The assumptions underlying the adaptation of AA with DA can be explained in two parts:

- Firstly, AA is commissioned with the goal of finding the shortest path between two different nodes. DA in Floodlight, on the other hand, computes every shortest path from a starting node to all the other nodes in a tree topology. In this study, DA works as usual, but AA is organized to work between two specific nodes. For performance comparison of DA and AA, the results of these two determined nodes are considered.
- Secondly, as it is generally the case with computer network structures, Floodlight gives different latency values for the opposite directions of two nodes. Here, the latency is one of the link cost metrics in the system. In this study, the smallest value is used as a basic cost metric in both directions for elimination of any disorder in two different latency values on a link.

The pseudocode of AA can be inspected from Figure 5 [Bertsekas 1992].

1 End the process if starting and ending nodes become the same point.

2 Assign the starting node to *CurrentNode*.

3 Determine *CurrentNode* as *solution*.

4 Generate an array as *Pvector* having the number of elements including all nodes. Initialize all values of the elements with 0.

5 Define the variable *nextLink* presenting the link connections.
6 Do the follows until the destination end of *nextLink* becomes the destination node:
6.1 Assign the list of connections arising out of the current node to *linksCurrentNode*.
6.2 Do follows for all nodes of *linksCurrentNode* list:
6.2.1 Assign *Pvector* value of the node in which the current connection arrived to *vectorValue*.
6.2.2 Assign the link cost to *linkValue*.
6.2.3 Assign the sum of *linkValue+vectorValue* to *sumValue*.
6.2.4 Assign the connection providing *sumValue to nextLink*.
6.3 Assign the value of *CurrentNode* in the vector *p* to *currentVectorValue*.
6.4 If *currentVectorValue* is smaller than *sumValue* perform *contractPath*, otherwise perform *extendPath*.
6.4.1 *contractPath:* Remove the last node from the solution set.
6.4.2 *extendPath:* Insert the node in which *nextLink* arrived to the solution set.

Figure 5: Pseudocode of AA

2.2.4 Dual Ascent algorithm

This algorithm is a type of primal-dual algorithm and is defined to proceed over a dual solution form of a linear program. In 1984, Richard T. Wong diversified the DAA logic and proposed a distributed algorithm. At the beginning of the algorithm there are two

data structures in the system, namely rootComponent and graph. These data structures list the node elements of the topology. Originally, there are not any nodes in the graph. The nodes are added into the graph iteratively starting from the source node. When the destination node is added into the graph list, there is exactly one shortest path in the last path set. The path between a source and destination pair is obtained, according to the tree structure [Ilhan and Yiltas-Kaplan 2020], [Wong 1984].

Before planning the coding part of DAA in this study, the source codes of a DAA sample in [GitHub 2017] were analyzed. Then, a general plan was derived from this sample. The overall design of the algorithm codes in this paper depends on the Floodlight data structures, which are totally different from the literature sample in [GitHub 2017]. The pseudocode of DAA is presented in Figure 6 [GitHub 2017], [Ilhan and Yiltas-Kaplan 2020].

1 Create the arrays of *rootComponent* and *graph* for the nodes.

2 Once a temporary *rootComponent* list is extracted from the permanent *graph* list.

3 Until the target node enters the *graph* array do the following iteratively:

3.1 Find a suitable link by starting from the *rootComponent* list. This link target node is chosen from the links in the *rootComponent* array.

3.2 The reduced cost table is updated by using this link. In other words, if the target node of each link is in the *rootComponent* list, then the link value in the reduced cost table is updated by decreasing this value with that of the link which has been chosen at the beginning and has the least-cost (link *a*) [GitHub 2017].

3.3 The node *a* is added to the *graph* chain. Then the *rootComponent* array is deleted and evaluated again copying from the *graph* chain.

4 Eliminate the nodes that have not any common link in both directions as source side or target side in the *graph* solution set. To decide these operations, some data structures are stored during the reduced cost updates.

5 All nodes in the *graph* list are checked whether the node subsets construct any cycle and these relevant subsets are solved.

Figure 6: Pseudocode of DAA [Ilhan and Yiltas-Kaplan 2020]

2.3 Normalization process

The Floodlight Project [Wang 2018] includes a function for computing the minimum cost path between two nodes by using the cost values on this path. The connection costs between the nodes are represented by the latency metric in the latest version of Floodlight (V1.2-SNAPSHOT (Master branch)) during this study. This metric is used for performing the path computations in DA of Floodlight.

In the literature of the traditional networks, there are numerous studies and methods concerning multi-metric structures for improving the quality of service during the routing processes, as can be seen in [Yiltas and Perros 2011], [Yiltas-Kaplan 2015]. There are also many studies about general decision-making processes with multi-metric evaluations. In the literature this process is called multi-criteria decision making and

one of the related studies regarding real-time traffic management of autonomous vehicles with totally 17 main and sub-criteria can be analysed in [Deveci et al. 2021].

In this study, the cost measurement is executed using two different metrics, for the purpose of performance improvement. Adapting two different metrics in Floodlight is quite complicated than the classical computer networks. Gathering large amount of network data with SDN application requires robust hardware equipment in terms of storage and processing. SDN provides the usage of virtual machines for the controller software and these machines share the hardware resources during the executions of the applications. Additionally, to construct SDN topology via Mininet covers some definitions of devices and links with Python. The number of these definition lines in a program is dependent on the number of network devices. This study aims to offer a guide for joining two metrics of different data types and units and presenting means of using the aggregated value in the routing algorithms through analyzing the performance enhancement.

Some metrics are from the space of very large integers, and some are from decimal numbers. A comparison of the metrics from various number spaces may not be practical. The usage of functions, such as the Euclidean distance, provides aggregation of several metric values into a single sum. However, this operation causes outperforming of the metrics that have large numerical values [Aggarwal 2015]. Therefore, several different techniques were investigated, and normalization rules were decided to be the most suitable method for filtering metrics. The Min-Max Normalization (Rescaling) and Z-Score Normalization (Standard Scoring) are the two rules that have emerged as ideal in the process.

2.3.1 Min-Max Normalization

This method is also called Rescaling. Suppose that the minimum and maximum values of the *j*th metric become $\min_j$ and $\max_j$ respectively. The *i*th record is $\overline{X_i}$ and the value of *j*th metric of $\overline{X_i}$ is x_i^j. As in (1), x_i^j is scaled to $[0, 1] \subset R^+$ [Aggarwal 2015].

$$y_i^j = \frac{x_i^j - \min_j}{\max_j - \min_j} \tag{1}$$

Min-Max normalization may eliminate the importance of some metrics in the relevant data set during the scaling into [0, 1] if a numerical mistake occurs in the data. For example, if the interval between the minimum and maximum values is written 100 instead of 10 by mistake, Min-Max normalization can produce irrelevant results. Z-Score normalization is preferred in such circumstances [Aggarwal 2015].

2.3.2 Z-Score Normalization

This method is also called Standard Scoring or Standardization. Suppose that the mean of the *j*th metric is μ_j, the standard deviation is σ_j and the *i*th record is $\overline{X_i}$. The *j*th metric value of $\overline{X_i}$ is x_i^j. x_i^j is normalized as in (2) [Aggarwal 2015].

$$z_i^j = \frac{x_i^j - \mu_j}{\sigma_j} \quad (2)$$

So, the values are scaled into an interval based on the normal distribution. In other words, the difference between a value and the mean in the unit of standard deviation is represented by Z-Score [Molugaram and Rao 2017].

2.4 Graphical user interface

In the topology generation, algorithm executions and normalization processes of the present study, the coding parts are related to Floodlight, Mininet, and IntelliJ IDEA. Floodlight does not have any GUI that presents the topology, the routing process, and the performance comparisons of the algorithms in the modules. The single GUI in the Floodlight project was designed by the REST API only to give some properties of Floodlight such as the status, the number of the devices and connections, and the flow summary tables of the switches [Akçay and Yiltas-Kaplan 2017].

For the purposes of the present study, a new GUI is proposed to demonstrate the steps of the process and the results from the above-mentioned software platforms. Thus, this GUI aggregates all the other parts into an entire visual structure. It represents the visual topology coded in Mininet and the final paths computed from the algorithms in IntelliJ IDEA, and yields comparative results extracted from the result files of the basic software platforms.

The application includes two fundamental sections:

- The User Interface
- The Floodlight Communication Server Software

A communication server is designed to display the results of the algorithms executed in Floodlight part from the GUI view. To this end, the results of all the algorithms executed in Floodlight are saved in a folder created in the communication server. The user interface module gets the data concerning the path output of the algorithm, the path cost, and flow simulation by sending HTTP requests to the communication server upon any requirement.

Figure 7 illustrates the architectural diagram of the structure concerned with the GUI. The user interface of the application was designed on React, which has been one of the most popular and effective user interface frameworks in recent years. All operations represented in the interface are as follows:

A. Procedures

A.1. Topology Generation: The topology information is derived from the communication layer and the nodes, and their network connections are created on the screen.

A.2. Algorithm Execution: This procedure involves three steps.

A.2.1. In the first step, latency and bandwidth values are read from the communication layer according to the selected normalization type. These values are located on the topology.

A.2.2. In the second step, the path produced with the relevant algorithm is received from the communication layer and located in the network.

A.2.3. In the third step, the path flow simulation data, that take 15 seconds between the source and destination nodes, are received from the communication layer, and visualized.

B. Comparison Graphs

B.1. Comparative cost graphs for each normalization type of each algorithm are demonstrated in the application.

B.2. A collective graph is placed for showing the comparisons of all algorithms in the same diagram.

B.3. Graph values are requested from the communication layer.

Furthermore, a node.js based server software is developed as the server framework in JavaScript language to read the data produced by Floodlight and transfer into the interface module.

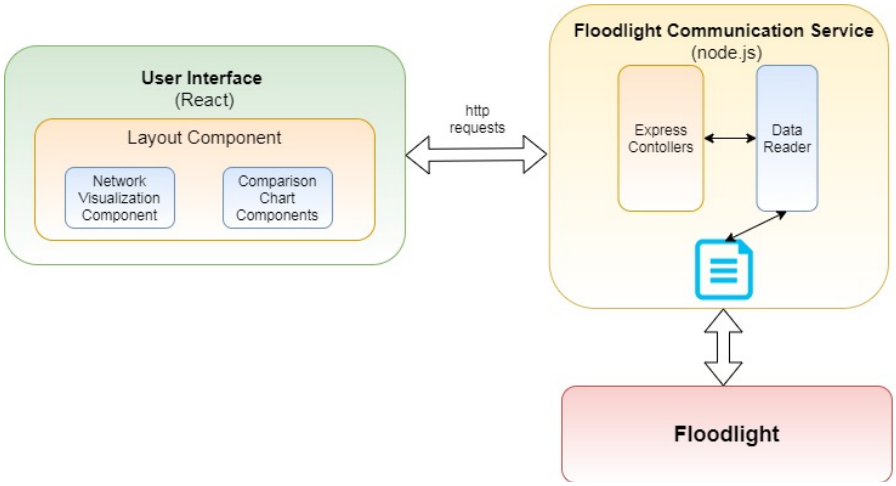

Figure 7: GUI architecture

The general view of the GUI can be seen in Figure 8.

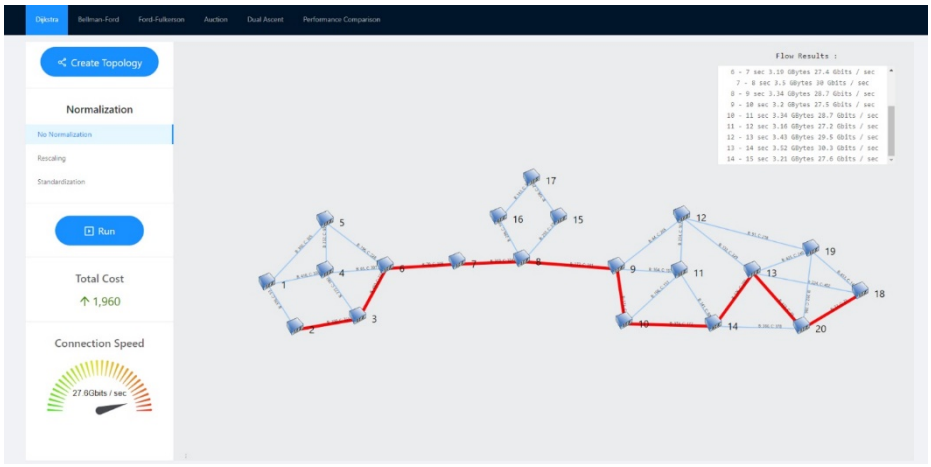

Figure 8: Screenshot of the GUI

2.5 Test Process

In this study, four different algorithms are used separately as the main routing methods in the core Topology module of Floodlight. The default routing method of Floodlight is DA, which is the fifth algorithm in the comparison results. The main purpose of routing operations in Floodlight is to find the minimum cost path between all node pairs. Original routing operations are investigated before the code development and testing steps. Here, source and destination node pairs are determined for usage in test data. Java classes of the algorithms are designed separately with constructing functions to find the minimum cost paths on the network.

Different metrics can be used to determine the basic parameters of a function that measure the costs and select the last routes. Original Floodlight function involves only latency as the link cost metric. In this study, both latency and bandwidth are used. These two metrics have different number spaces and units. At this step, the usage of normalization process becomes important.

Test process should be performed with the same input data and network topology for all algorithms. Each result should be measured as an average value of several different executions. This study gives the comparison results based on these rules clearly as seen in Section 3.

3 Discussion of Results

After running the algorithms and utilizing the GUI, the performance results are collected. This operation provides a final analysis on the performance values. Besides software codes, hardware is also important for this stage. Table 1 shows properties of the computer that was used during the computations.

Brand	Dell Inspiron N5110 (2012 Production)
RAM	8 GB RAM
Processor	Intel Core i5-2450M CPU, 2.50 GHz, Quad Core

Table 1: Configuration properties of the test environment

All mentioned software environments are presented in Table 2.

Ubuntu	Ubuntu 18.04
Java development environment	IntelliJ IDEA Ultimate
Floodlight	V1.2-SNAPSHOT (Master branch)
Mininet	V2.2.1
Java version for Floodlight integration	Java 8

Table 2: Versions of software platforms

The whole algorithm codes underpinning the software project proposed in this paper are deposited at https://bitbucket.org/deryayiltas/project-derya.git repository and

GUI codes are at https://bitbucket.org/deryayiltas/gui.git. Similarly, the data underpinning the analysis reported in this paper are deposited at "Data repository" TrafficData at https://bitbucket.org/deryayiltas/TrafficData.git.

3.1 Cost comparisons

Cost comparisons of the algorithms based on No-normalization, Min-Max normalization, and Z-Score normalization are given in Figures 9, 10, and 11, respectively. Each relevant diagram value is computed using the average of 12 different runs. The units of the two metrics are defined as millisecond (ms) for latency and Gbps (Gigabits/sec) for bandwidth. Decisions on the flow routings are given according to both metrics. The cost values represent the product of latency values with a predetermined coefficient, which is assumed to be 10 in this study. In other words, the diagram views are scaled to become completely observable. Here the most important issue is making the graphical representations easier and more comprehensible. In graphics, the differences between the algorithms are relatively important. As shown in Figure 9, the cost results of DA and BFA are very close. According to Figure 10, DAA gives the best results against all other algorithms. Therefore, using DAA with Min-Max normalization instead of DA in Floodlight has advantages with respect to the network cost values.

DA and BFA have very similar results in Figure 11, too. Figure 11 further illustrates that BFA provides lower cost values. All results affirm that the normalization processes bring forth cost-effective path selections in SDN applications.

According to overall results in Figures 9, 10, and 11, DA and BFA are generally better selections for network operations having delay value as vital metric such as real time applications, namely video or voice. Conversely, DAA with Min-Max normalization is the best selection for real time applications.

Figure 9: Comparison results according to No-Normalization

Figure 10: Comparison results according to Min-Max Normalization

Figure 11: Comparison results according to Z-Score Normalization

3.2 Flow comparisons

For an evaluation of the Data Transfer Rate (in Gbps), Average Amount of Data in one second (in GByte) and the Total Amount of Data (in Gbyte) respectively, the iPerf command in Mininet is used. A TCP traffic flow of 15 seconds is constructed. The

results are obtained for all algorithms with No-normalization, Min-Max normalization, and Z-Score normalization. As a sample, the values of dFFA are given in Table 3. The results in Table 3 demonstrate that Min-Max normalization helps dFFA to provide precious values in terms of transmitting data rates.

	0-5 seconds	**5-10 seconds**	**10-15 seconds**	**Overall (0-15 seconds)**
No-Normalization	25.98 Gbps 2.98 GBytes 14.89 GBytes	26.14 Gbps 3.04 GBytes 15.21 GBytes	25.5 Gbps 2.97 GBytes 14.84 GBytes	25.74 Gbps 3.00 GBytes 44.94 GBytes
Min-Max Normalization	27.32 Gbps 3.18 GBytes 15,9 GBytes	28.32 Gbps 3.3 GBytes 16.5 GBytes	28.52 Gbps 3.3 GBytes 16.49 GBytes	**28.05 Gbps** **3.26 GBytes** **48.89 GBytes**
Z-Score Normalization	27.32 Gbps 2.82 GBytes 14.09 GBytes	28.32 Gbps 3.21 GBytes 16.06 GBytes	28.52 Gbps 3.14 GBytes 15.7 GBytes	28.05 Gbps 3.06 GBytes 45.85 GBytes

Table 3: Average flow results of dFFA

Average results of several different executions of all algorithms are obtained and presented in Figures 12 and 13. The values of data transfer rates in Figure 12 show the average value for each second computed based on the TCP traffic flow of 15 seconds. As shown in Figures 12 and 13, normalization rules in general and Min-Max normalization in particular reveal increases in the values of the data transfer rates and total amount of data. Especially dFFA gives better results than the other algorithms. Moreover, DA and BFA give very close result values.

According to overall results in Figures 12 and 13, it is clear that dFFA is generally the best selection for network operations requiring higher throughput values as vital metric. Video or file transportations are some sample applications to these network operations. Data transfer rates are very important in such data formats, especially for video transmissions.

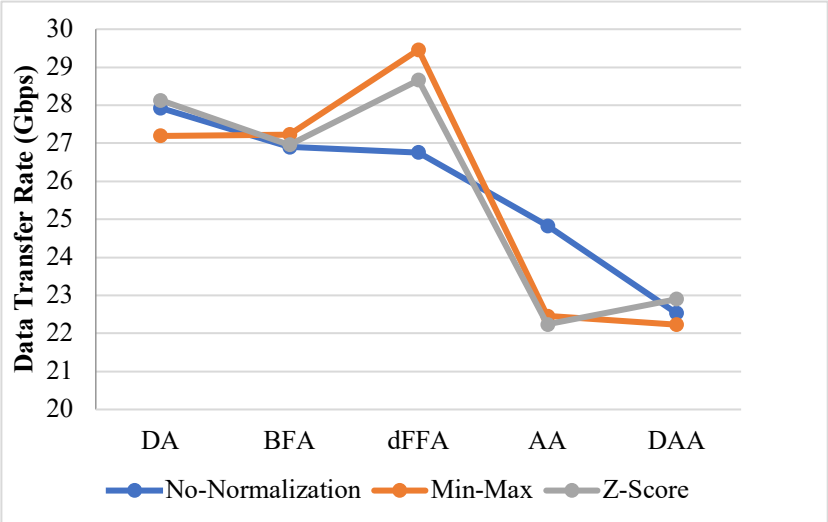

Figure 12: Comparison of the data transfer rates (15- second flow)

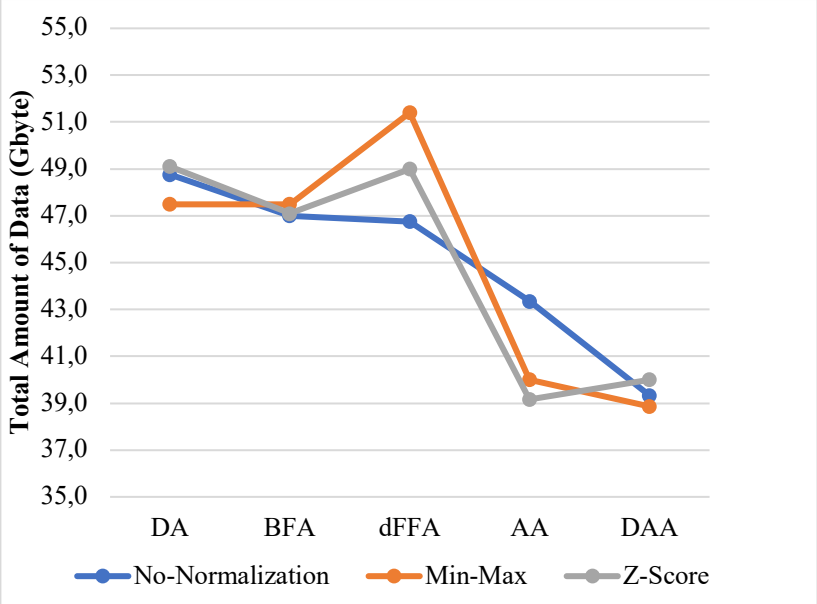

Figure 13: Comparison of total amounts of data (15- second flow)

Additional evaluations are made for TCP traffic flows of 5 seconds and 60 seconds respectively to show the difference between short and long flows. Figures 14 and 15 involve the average values of different executions at each point for traffic flows of 5

seconds. Figures 14 and 15 signify that for short flows Min-Max normalization is better than Z-Score normalization and moves closer to No-Normalization. There are not many data values for 5-second flows, for this reason Min-Max normalization cannot get a large scale for minimum and maximum values for the metrics during the evaluation of (1) and becomes similar to No-Normalization. According to Figures 14 and 15, dFFA again gives good results versus the other algorithms.

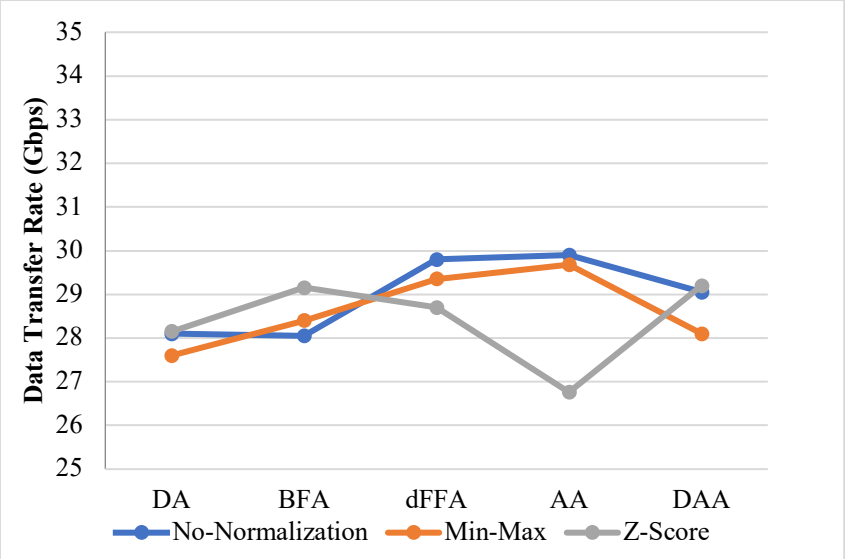

Figure 14: Comparison of the data transfer rates (5- second flow)

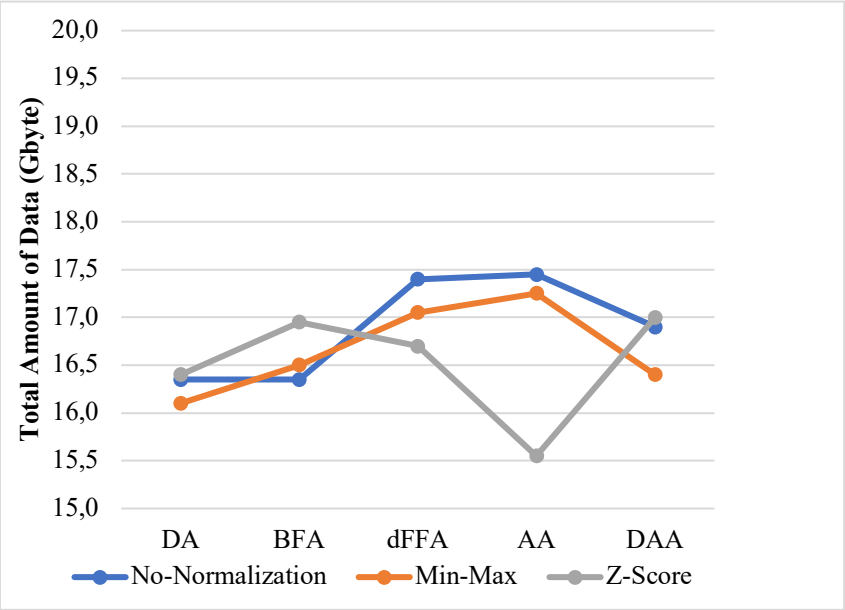

Figure 15: Comparison of total amounts of data (5- second flow)

Similarly, Figures 16 and 17 give the average results for the traffic flows of 60 seconds.

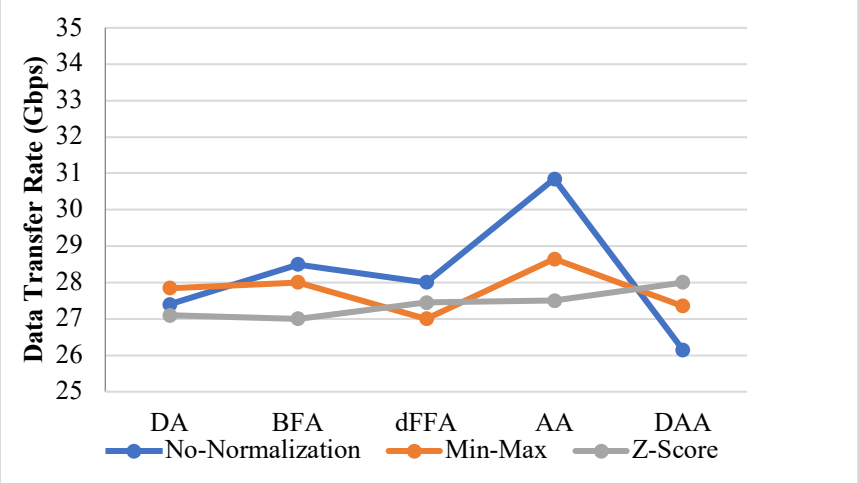

Figure 16: Comparison of the data transfer rates (60- second flow)

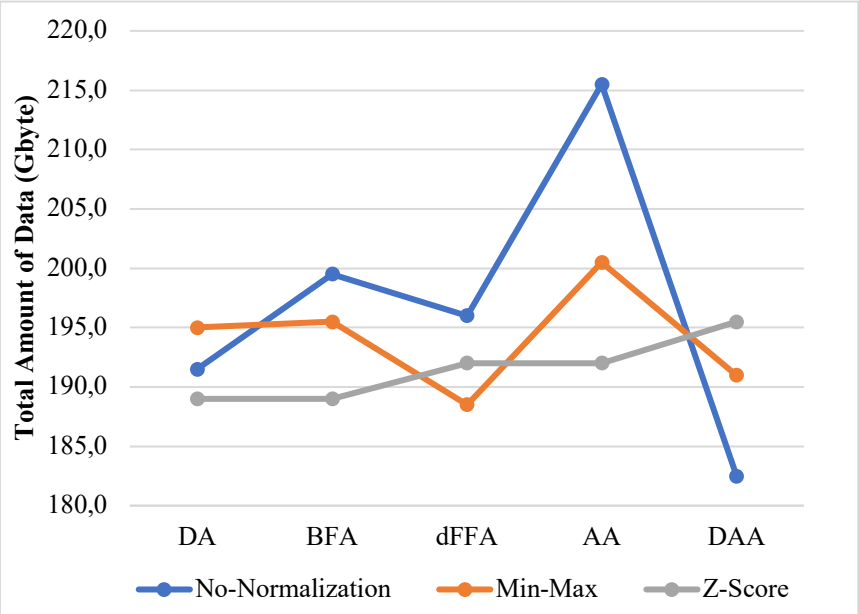

Figure 17: Comparison of total amounts of data (60- second flow)

Figures 16 and 17 show that for long flows Min-Max normalization is again better than Z-Score normalization, but No-Normalization exhibits an improvement for BFA, dFFA, and AA against Min-Max normalization. For 60-second flows, there may be numerous metric values that become outliers during Min-Max computations and this situation causes fair average performance for some scenarios. On the other hand, the results prove that any normalization method gives an opportunity to deactivate an exceptional improvement of the algorithms in No-Normalization steps caused by various random data values in long flows.

4 Conclusions

In this study, the routing module in Floodlight was changed according to four different optimization algorithms, namely BFA, dFFA, AA, and DAA. The operations with No-Normalization, Min-Max normalization, and Z-Score normalization were added to the programs by considering two metrics: latency and bandwidth. The proposed routing modules were compared against the default routing module of Floodlight using DA. A GUI was constructed to see the topology, the results of the algorithms, and the graphic comparisons of the whole processes. The results present that the normalization types, especially Min-Max, improve the cost effectiveness in general. DA and BFA give very close cost results for the paths. DAA outperforms all the other algorithms regarding Min-Max normalization results.

The data transfer rates, and the total amount of data were also evaluated based on the bandwidth. Here, in a similar manner DA and BFA again give very close results. On the other hand, dFFA outperforms all the other algorithms in most of the scenarios planned with different sizes of flows when using normalization steps.

This study performs different modules in Floodlight and provides a new GUI to see the execution and comparison results. It serves as a preliminary study for the literature with giving two-metric solutions for an SDN topology. Using several metrics instead of only one metric increases the quality of service in SDNs for real world problems. In network problems, bandwidth becomes one of the most important issues for flow transmissions and traffic optimization. Therefore, bandwidth is vital during the routing processes. In this study, individual data sets of two different metrics were brought together with normalization steps. Improvement of the results by considering normalization processes based on various proportions among the metric values is suggested for future research. Concerning the traffic metrics, another future work can be the use of several different metrics instead of two. The proposed solutions can also give directions to any other field of studies related with cost optimization problems. During the programming part of this study, all algorithms and computations were planned to become effective in terms of computational time. Before the algorithm executions, traffic flows are generated via Mininet and this process takes the relevant seconds such as 5, 15 or 60 as the computational time. Java classes do not waste any time except the switching options between the algorithms or normalization types. Finally, GUI part displays the outputs in a few seconds, namely 7 seconds for a route selection by using one algorithm and one normalization type.

Acknowledgements

This work has been supported by the Scientific and Technological Research Council of Turkey (TUBITAK) 3001 Starting R&D Projects Funding Program with the project number of 116E905.

The author would like to thank Hilmi Tunahan Ilhan, Tolga Cubukcuoglu and Hani Elubeyd for their helpful comments and skilled technical assistance during this study, and Dr. Dilek Inal, Dr. Savio S.H. Tse and Dr. Mustafa Dagtekin for proofreading the paper.

References

[Aggarwal 2015] Aggarwal, C. C.: "Data Mining: The Textbook (eBook)", Springer International Publishing.

[Akbaş et al. 2016] Akbaş, M. F., Karaarslan, E., Güngör, C.: "A Preliminary Survey on the Security of Software-Defined Networks", International Journal of Applied Mathematics, Electronics and Computers, 4(Special Issue), (2016), 184-189, https://doi.org/10.18100/ijamec.270088.

[Akçay and Yiltas-Kaplan 2017] Akcay, H., Yiltas-Kaplan, D.: "Web-Based User Interface for the Floodlight SDN Controller", Int. J. Advanced Networking and Applications, 8(5), (2017), 3175-3180.

[Akçay and Yiltas-Kaplan 2019] Akcay, H., Yiltas-Kaplan, D.: "Performing Classification for Anomaly Detection in Software Defined Networking", In Proc. 6th International Scientific Research Congress, (2019), 513-523.

[Becker et al. 2016] Becker, R., Fickert, M., Karrenbauer, A.: "A Novel Dual Ascent Algorithm for Solving the Min-Cost Flow Problem", In Proc. of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX), (2016), 151-159.

[Bertsekas 1991] Bertsekas, D. P.: "An Auction Algorithm for Shortest Paths", SIAM J. Optimization, 1(4), (1991), 425-447.

[Bertsekas 1992] Bertsekas, D. P.: "Auction Algorithms for Network Flow Problems: A Tutorial Introduction", [online] https://dspace.mit.edu/bitstream/handle/1721.1/3265/P-2108-26912652.pdf;sequence=1, (Accessed 15 January 2022).

[Bertsekas 1998] Bertsekas, D. P.: "Network Optimization: Continuous and Discrete Models", Athena Scientific.

[Demaine and Wenk 2022] Demaine, E., Wenk, C.: "Shortest Paths in Graphs Bellman-Ford Algorithm", [online] https://www2.cs.arizona.edu/classes/cs545/fall09/ShortestPath2.prn.pdf, (Accessed 15 January 2022).

[Deveci et al. 2021] Deveci, M., Pamucar, D., Gokasar, I.: "Fuzzy Power Heronian Function Based CoCoSo Method for the Advantage Prioritization of Autonomous Vehicles in Real-Time Traffic Management", Sustainable Cities and Society, 69, (2021), https://doi.org/10.1016/j.scs.2021.102846.

[GitHub 2017] GitHub: "Java code to build and run evaluation of Approximation Algorithms for the Directed Steiner Tree problem", [online] https://github.com/mouton5000/DSTAlgoEvaluation, (Accessed 15 January 2022).

[Hong et al. 2013] Hong, C.-Y., Kandula, S., Mahajan, R., Zhang, M., Gill, V., Nanduri, M., Wattenhofer, R.: "Achieving High Utilization with Software-Driven WAN", In Proc. ACM SIGCOMM, (2013), 15-26, https://doi.org/10.1145/2486001.2486012.

[Ilhan and Yiltas-Kaplan 2020] Ilhan, H. T., Yiltas-Kaplan, D.: "Changing Routing Module in Floodlight", In Proc. 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE), (2020), 1-5, 10.1109/ICECCE49384.2020.9179298.

[Klappenecker 2022] Klappenecker, A.: "The Bellman-Ford Algorithm", [online] http://faculty.cs.tamu.edu/klappi/csce411-f17/csce411-graphs6.pdf, (Accessed 15 January 2022).

[Molugaram and Rao 2017] Molugaram, K., Rao, G. S.: "Random Variables", Statistical Techniques for Transportation Engineering, Chapter 4, (2017), 113-279.

[Open Networking Foundation 2014] ONF: "SDN Architecture", [online], https://www.opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf, (Accessed 15 January 2022).

[Özbek et al. 2020] Özbek, U., Yiltas-Kaplan, D., Zengin, A., Ölmez, S.: "Traffic Analysis of a Software Defined Network", In Proc. 1st International Congress on Engineering Technologies (EngiTek), (2020), 44-49.

[Wang 2018] Wang, Q.: "Installation Guide, Floodlight" [online], https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343544/Installation+Guide#InstallationGuide-FloodlightMasterandAbove, (Accessed 15 January 2022).

[Wong 1984] Wong, R. T.: "A Dual Ascent Approach for Steiner Tree Problems on a Directed Graph", Mathematical Programming, 28(3), (1984), 271–287.

[Yiltas and Perros 2011] Yiltas, D., Perros, H.: "Quality of Service-Based Multi-Domain Routing Under Multiple Quality of Service Metrics", IET Communications, 5(3), (2011), 327-336, 10.1049/iet-com.2010.0144.

[Yiltas-Kaplan 2015] Yiltas-Kaplan, D.: "Cost Functions for Two-Metric Quality Of Service Routing", Istanbul University Journal of Electrical and Electronics Engineering, 15(2), (2015), 1913-1919.

[Yiltas-Kaplan 2019] Yiltas-Kaplan, D.: "The Usage Analysis of Machine Learning Methods for Intrusion Detection in Software-Defined Networks", Chapter 5, Artificial Intelligence and Security Challenges in Emerging Networks, Editor: Ryma Abassi, IGI Global, Hershey, PA, 124-145.