Hindawi
Mobile Information Systems
Volume 2019, Article ID 3971598, 16 pages
https://doi.org/10.1155/2019/3971598

Research Article

NSAF: An Approach for Ensuring Application-Aware Routing Based on Network QoS of Applications in SDN

Joonseok Park,[1] **Jeseung Hwang**,[2] **and Keunhyuk Yeom**[3]

[1]*Research Institute of Logistics Innovation and Networking, Pusan National University, Busandaehak-ro 63beon-gil, Geumjeong-gu, Busan 46241, Republic of Korea*
[2]*Busan Grand ICT R&D Center Associate Research, Pusan National University, Busandaehak-ro 63beon-gil, Geumjeong-gu, Busan 46241, Republic of Korea*
[3]*Department of Electrical and Computer Engineering, Pusan National University, Busandaehak-ro 63beon-gil, Geumjeong-gu, Busan 46241, Republic of Korea*

Correspondence should be addressed to Keunhyuk Yeom; yeom@pusan.ac.kr

Received 17 January 2019; Accepted 31 March 2019; Published 23 April 2019

Guest Editor: Pinar Kirci

The advent of software-defined networking (SDN) has led to the paradigm of programmable network environments. Conceptually, the structure of SDN is divided into three layers: application, control, and infrastructure. The SDN controller in the control layer can configure and execute the routing of applications to the infrastructure layer consisting of network devices, including hosts and switches. Current studies on SDN have predominantly focused on control layer aspects, such as controller development, performance aspects of the controller, and interaction among different controllers and between controllers and network devices. However, to provide seamless network services and efficiently manage network environments, application-aware routing is essential because applications may have different quality of service (QoS) requirements, such as maximum bandwidth and minimum delay. This study proposes the Network Situation-Aware Framework (NSAF) to more efficiently handle application routing based on the QoS requirements and changing network status. A mechanism for supporting the NSAF consisting of application registration, network status monitoring, violation detection, and routing control is also presented. The applicability and feasibility of the proposed NSAF are verified by implementing a prototype. The NSAF may be used as a reference model for efficiently managing SDN-based networks to ensure application QoS.

1. Introduction

Developments in network technologies have resulted in new programmable network environments based on software-defined networking (SDN) [1–5], in which the control plane is separated from the data plane. As shown in Figure 1, SDN can generally be divided into three layers: application, control, and infrastructure. The controller in the control layer can order the routing of business applications to the infrastructure layer.

Following the advent of SDN, relevant studies have primarily focused on the control layer aspects [6, 7]. Various SDN open-source projects, such as Floodlight [8], OpenDaylight [9], ONOS [10], and Ryu [11], are currently active. Research into protocols, including OpenFlow [12] and OpFlex [13], which support communication between the control and infrastructure layers, is also underway. To further spread the use of the SDN technology and maximize its network management benefits, research on the application layer aspect is needed (e.g., a method to control the network from the application point of view).

In SDN, business applications can have different network quality of service (QoS) requirements. A particular application may need maximum bandwidth, whereas another may require minimum delay. In addition, the network condition changes according to network traffic and failure. Therefore, application-aware routing [14–16] is essential. This study proposes a framework called the Network Situation-Aware Framework (NSAF), and its mechanism satisfies this need and ensures a stable execution of SDN

applications. The NSAF analyzes each application's network QoS requirements, monitors current network status, detects violation of network QoS, and finds the most appropriate network routing paths.

The remainder of this paper is organized as follows: Section 2 outlines the basic concepts; Section 3 presents the NSAF and its design and architecture; Section 4 presents a case study; Section 5 discusses related work and evaluates the proposed mechanism; and Section 6 concludes this paper and outlines future work.

2. Basic Concepts

2.1. DiffServ. Differentiated services (DiffServ) specify a simple and scalable mechanism for classifying and managing network traffic and providing QoS on modern IP networks [17]. DiffServ streamlines flow and simplify complicated packet processing in the network by clustering traffic into traffic classes according to the predefined QoS. This feature makes DiffServ lightweight and easy to implement.

Applications with similar traffic characteristics and performance requirements are mapped into DiffServ classes based on the end-to-end behavior requirements of the applications according to RFC 5127. Table 1 shows the service classes of DiffServ. Service class is used herein as an application type.

2.2. Ontology and Semantic Web Rule Language (SWRL). Ontology is a formal and explicit specification of shared conceptualization [18, 19]. The ontology is a core element of the semantic web that increases the quality of information search on the web by enabling machines to decipher and understand the data existing on the web without human involvement by assigning semantics to the data. The key elements of the ontology are the following:

(1) Classes: sets, collections, concepts, or types of objects

(2) Individuals: instances of objects

(3) Relations: ways in which classes and individuals can be related to one another

(4) Attributes: aspects, properties, features, characteristics, or parameters that can be associated with objects

SWRL [20, 21] is used to express rules in the semantic web. An SWRL rule consists of an inference relationship between antecedent and consequent.

2.3. Genetic Algorithms. Genetic algorithms (GAs) [22–24] are a class of global optimization algorithms first introduced in John Holland's book "Adaptation on Natural and Artificial Systems." A typical GA has the following execution process:

(1) Randomly initialize population (t)

(2) Determine the fitness of population (t)

(3) Loop

(a) Select parents from population (t)

(b) Perform crossover and mutation on parents to create offspring population

(c) Determine the fitness of the combined population ($t+1$)

(4) Stop when the best individual is good enough

Holland [22] suggested the notion of schemata for the convergence analysis of genetic algorithms. Schemata are bit patterns, which function as representatives of a set of binary strings. The population consists of a set of N binary strings of length L at time t, where N is the number of strings in the population, which contains the bit pattern, such as 100000, 110011, and 010010. Furthermore, H is called $o(H,t)$. Let us assume that function f has to be maximized. f is defined as overall binary strings of length L, and it is called fitness of the strings. Two parent strings from the current population are always selected for the creation of a new string. The probability that a parent string H_j will be selected from N strings $H_1, H_2, \ldots, H_N$ is shown in the following equation [23]:

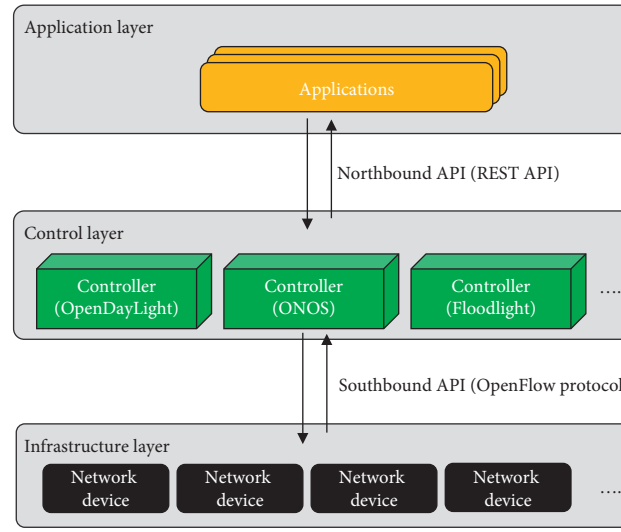


Figure 1: SDN layers.

Table 1: DiffServ classes [17].

Service class (application type)	QoS tolerance		
	Packet loss	Delay	Jitter
Network control	Low	Low	Yes
Telephony	Very low	Very low	Very low
Signaling	Low	Low	Yes
Multimedia conferencing	Low-medium	Very low	Low
Real-time interactive	Low	Very low	Low
Broadcast video	Very low	Medium	Low
Multimedia streaming	Low-medium	Medium	Yes
Low-latency data	Low	Low-medium	Yes
OAM	Low	Medium	Yes
High-throughput data	Low	Medium-high	Yes
Standard	Not specified		
Low-priority data	High	High	Yes

$$p(H_j) = \frac{f(H_j)}{\sum_{j=1}^{N} f(H_j)}. \quad (1)$$

Strings with greater fitness are more likely to be selected than those with lesser fitness. Let $f\mu$ be the average fitness of all strings in the population, as shown in the following equation [23]:

$$f\mu = \frac{1}{N} \sum_{i=1}^{N} f(H_i). \quad (2)$$

The probability $p(H_j)$ can be written as $(f(H_j))/(Nf\mu)$.

The fitness function is used to assess the excellence of the individuals in the population. That is, it evaluates whether each individual should survive into the next generation. A surviving individual becomes a parent and creates a new generation through crossover or mutation. Crossover is used to create offspring by copying positions in two parents that do not overlap with each other, while mutation creates offspring by changing the information of an individual. For the new generation, the fitness function repeatedly evaluates whether the optimization goal has been reached to solve the problem. The optimal solution is derived by repeatedly evaluating the new generation with the fitness function to determine whether the optimization goal of solving the problem has been achieved.

3. Network Situation-Aware Framework (NSAF)

The main techniques of the approach for embodying the conceptual NSAF architecture and realizing the NSAF are presented in this section. The NSAF is located between the application and control layers (Figure 2). It acts as an intermediator that manages the QoS requirement of applications and controls the SDN controller. Controller dependencies are mitigated by separating the application and the controller.

In a conventional structure, in which the application is directly connected to a controller, the application has to configure an additional module for interconnection for each controller. However, in a separated architecture, the application only needs to consider the interface of the NSAF, and the connection to each controller has the advantage of being abstracted and managed as a common NSAF interface. The advantage of monitoring and controlling the SDN controller through the NSAF API is that one does not need to know the usage and function of various kinds of SDN controllers.

3.1. Requirements and Architecture of the NSAF. Table 2 presents the identified key requirements of the NSAF.

Figure 3 shows the NSAF architecture in terms of its key requirements. The architecture consists of five modules: interface manager, application-aware service-level agreement (SLA) manager, application manager, flow rule manager, and topology manager. It has resources, including application profile, network service-level agreement (NSLA), topology information, and route information.

The interface manager utilizes access points to control the SDN controller through the NSAF by abstracting and mapping the REST API provided by the SDN controller. The application manager collects and manages NSLA information, which represents the application's detailed profile information and the network SLA of the application. The application-aware SLA manager determines whether or not the network QoS of the path used by the current application violates the NSLA and calculates new paths. The flow rule manager creates and manages rules that control routing paths. The topology manager collects, monitors, and manages network status and traffic information.

3.2. NSAF Execution. The NSAF execution covers application QoS assurance, including application profile and QoS registration, network status monitoring, identification of violations, and routing changes (Figure 4). It is divided into four phases: application registration, network status monitoring, violation detection, and routing control.

The service type of the application, basic information, and NSLA, which is the network QoS requirement information of the application, is registered in the application registration phase. Network status monitoring requests the API provided by the SDN controller to collect traffic and resource information for the connected network topology. The network status is analyzed and visualized based on the collected information. The violation detection process determines whether or not the route used by the current application violates the NSLA of the application based on the collected traffic and resource information and the application of NSLA information. Routing control is conducted if the NSLA is determined to be violated (i.e., the route currently being used by the application does not guarantee QoS). The routing control process constructs a path that satisfies the application QoS.

3.3. Application Registration. The RFC 5127 "Aggregation of DiffServ Service Class" standardized by the Internet Engineering Task Force [17] presents 12 application types: network control, telephony, signaling, multimedia conferencing, real-time interactive, multimedia streaming, broadcast video, low-latency data, OAM, high-throughput data, standard, and low-priority data. We used DiffServ's QoS tolerance as a condition for satisfying the application performance. We also analyzed the QoS factors defined in DiffServ, and the QoS metrics measurable by the SDN controller [25]. Based on this, we defined packet loss, bandwidth, delay, and jitter as the application quality factors.

As shown in Figure 5, the application profile is registered in the application registration phase. This profile consists of the application type information and the NSLA, which describes the application's network QoS requirements.

3.4. Network Monitoring. Collecting information on the current network situation is necessary in ensuring the

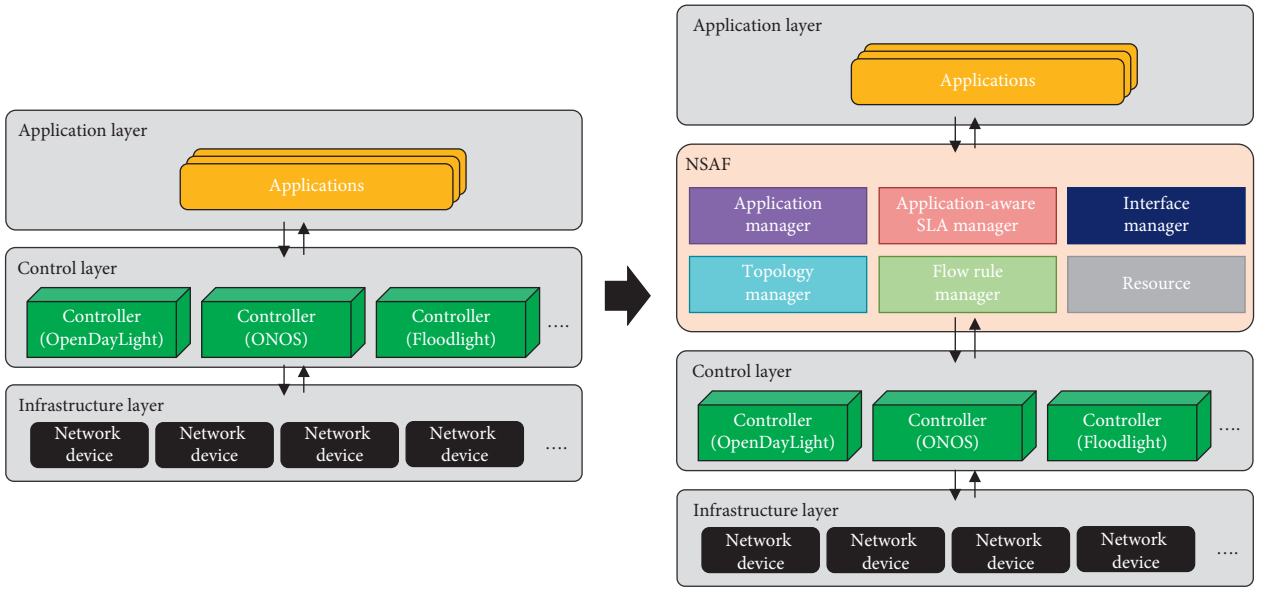


Figure 2: SDN vs extended SDN with the NSAF.

Table 2: Key requirements of the NSAF.

ID	Requirements	Description
R-01	Application profile management	Collects and analyzes the application type and application requirements
R-02	NSLA management	Manages network service-level agreement (SLA) for applications
R-03	Network status monitoring	Monitors node and link information in the network topology
R-04	Application-specific situation awareness	Ascertains QoS violation of the network route using application profile and NSLA
R-05	Application-specific path calculation	Calculates and determines path according to the network status

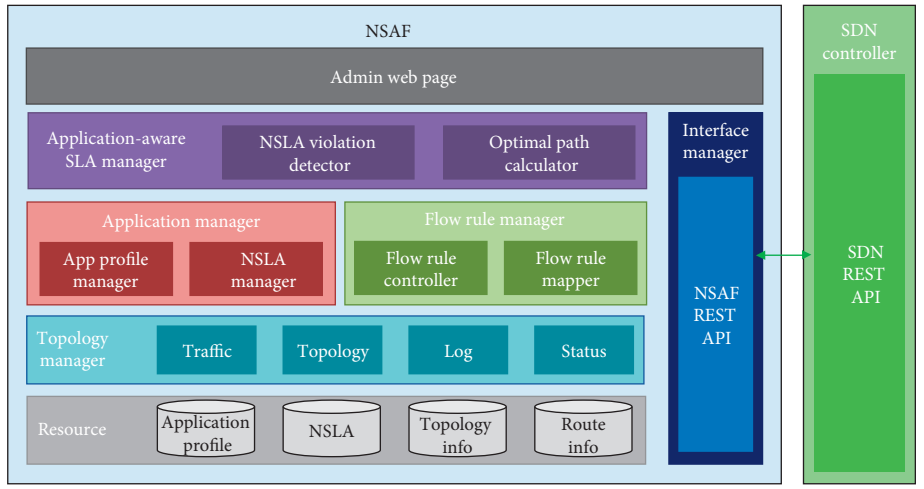


Figure 3: NSAF architecture.

application QoS. The proposed NSAF uses the REST API to determine the status of the SDN controller and collect and update the network status information in real time.

Figure 6 shows the topology model defined for collecting and managing the network status information in the NSAF.

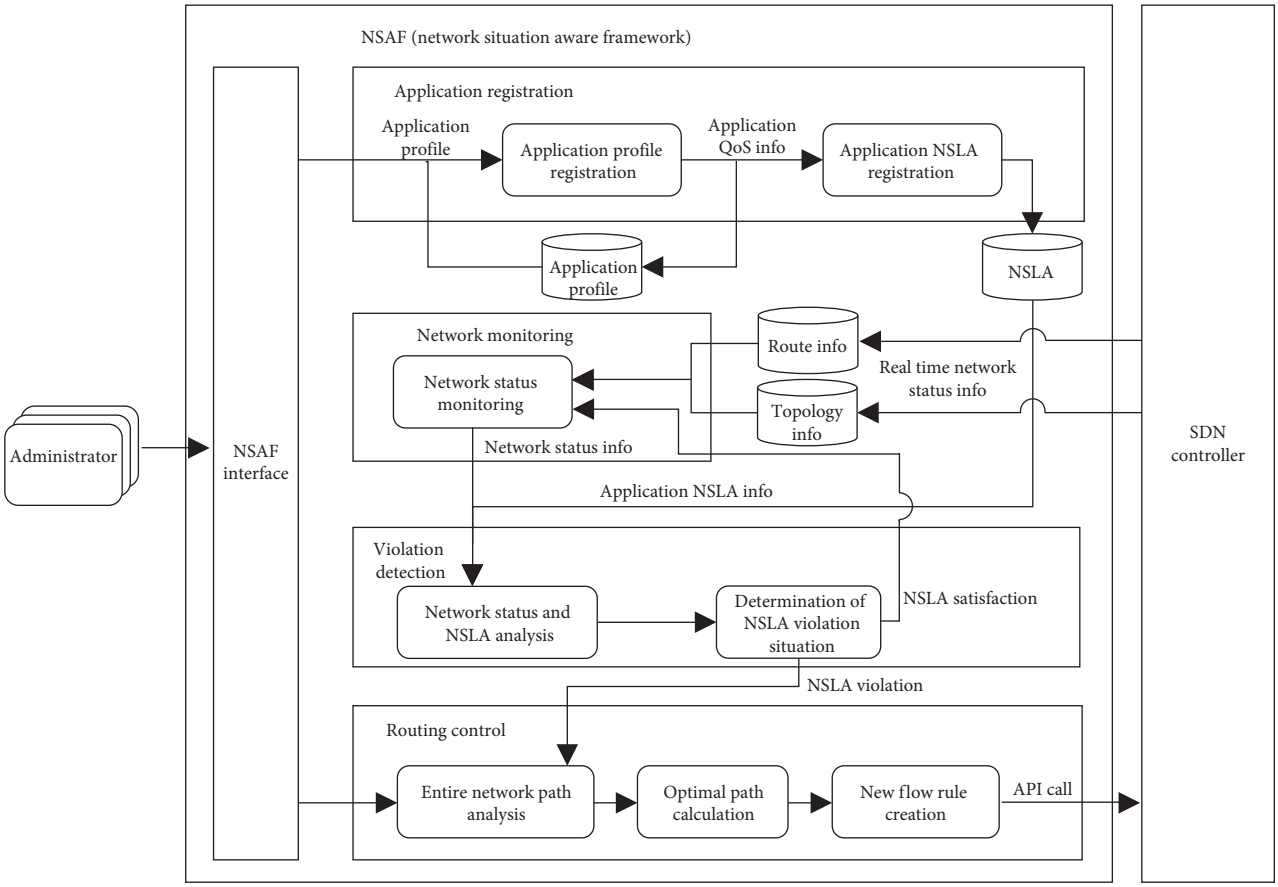


Figure 4: Execution of the NSAF.

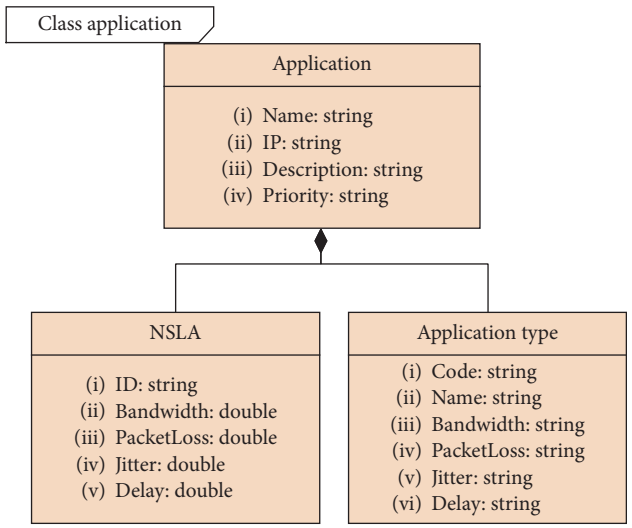


Figure 5: Application profile.

The topology for collecting the network status information models consists of at least one node and a link. Nodes are used to collect information on network devices, such as switches and routers. A link is defined to manage the connection between nodes for routing path configuration. The flow table is defined to manage information for network routing control.

3.5. Violation Detection. Two types of violations are defined to determine the application QoS violation situation. A hard fail occurs if the link between two nodes is broken. In contrast, a soft fail occurs if the connection between the nodes is not a problem, but the NSLA of the application is not satisfied. Table 3 shows the types of violations and details of the violations by type.

The ontology is applied to detect violations. Figure 7 shows the internal structure used to determine whether or not a violation has occurred. As illustrated in Figure 7, the application profile information and the network status information are sent to the NSLA Violation Controller. Context information (e.g., network QoS) and traffic information that can be measured for a link connecting nodes (e.g., switches and routers constituting the entire network topology) are extracted based on the collected information. The ontology manager reflects the extracted context information on the ontology. The ontology is loaded into the reasoning engine, and the violation situation is identified. The violation detect manager collects the violation and sends it to the monitoring manager through the NSLA violation controller to notify the NSLA of the violation of the current application.

We constructed the ontology model by abstracting the entities of the elements derived from the application registration and network state. We then defined the relationships between them. Table 4 shows an example of the attributes for the identified class, defined attribute, and data

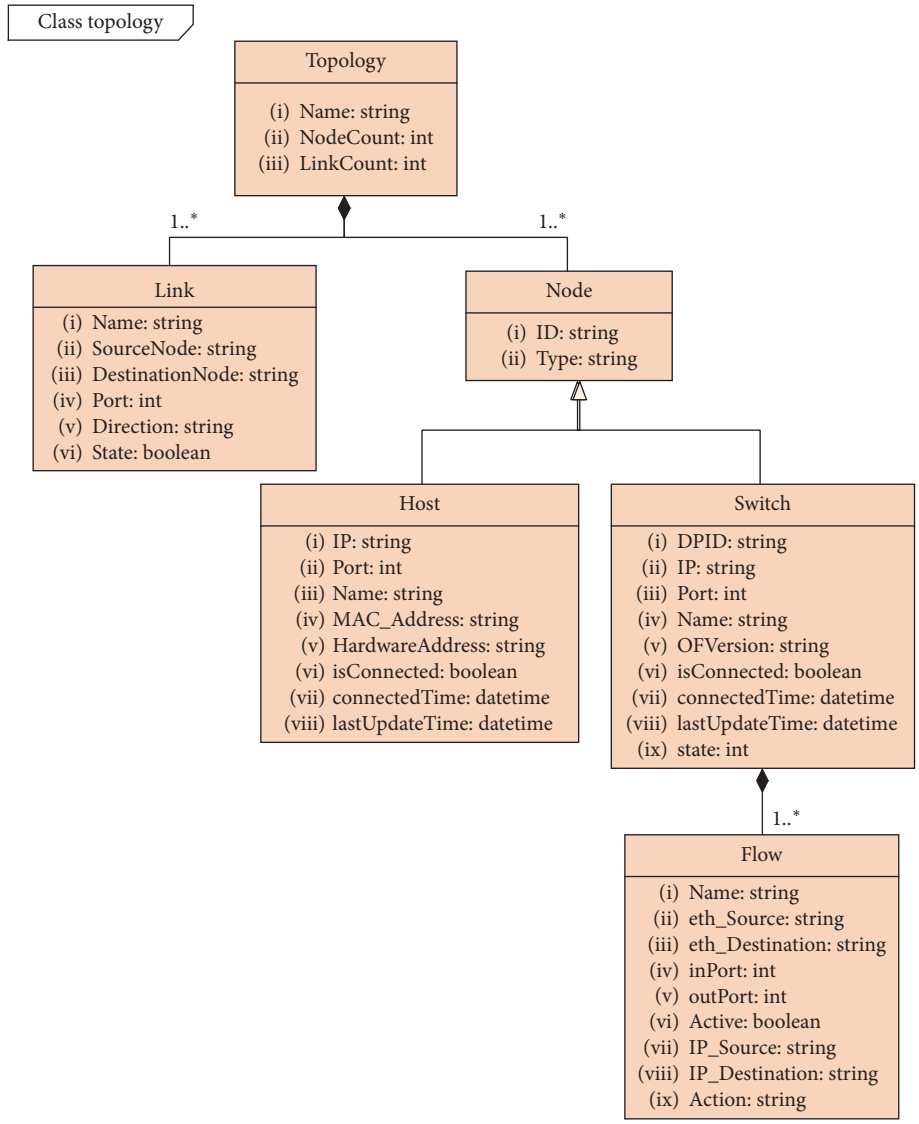


Figure 6: Topology model for the network status monitoring.

Table 3: Defined violation types.

Violation type	Details of violation type	Description
Hard fail	Node violation	The topology node is disconnected
	Link violation	The topology link is disconnected
Soft fail	Bandwidth violation	The application does not have the required bandwidth to execute
	Packet loss violation	The maximum packet loss for the application is exceeded
	Delay violation	The maximum delay for the application is exceeded
	Jitter violation	The application's jitter is violated

property elements. Figure 8 presents the ontology meta-model constructed using the elements analyzed in Table 4.

3.6. Routing Control. The network QoS of the application was ensured by applying two path algorithms to derive different paths when the defined NSLA is violated. First, the route algorithm was used to calculate the digest algorithm, which is the most used in the path algorithm, by using the QoS classified herein as the cost. In addition, a number of alternative paths can be derived by applying a GA to find optimal solutions for multiobjective problems.

The path calculation process using Dijkstra is elaborated below:

(5) Calculate the T-score using the average and standard deviation values for four kinds of application QoS. The T-score is used to set the reference value because

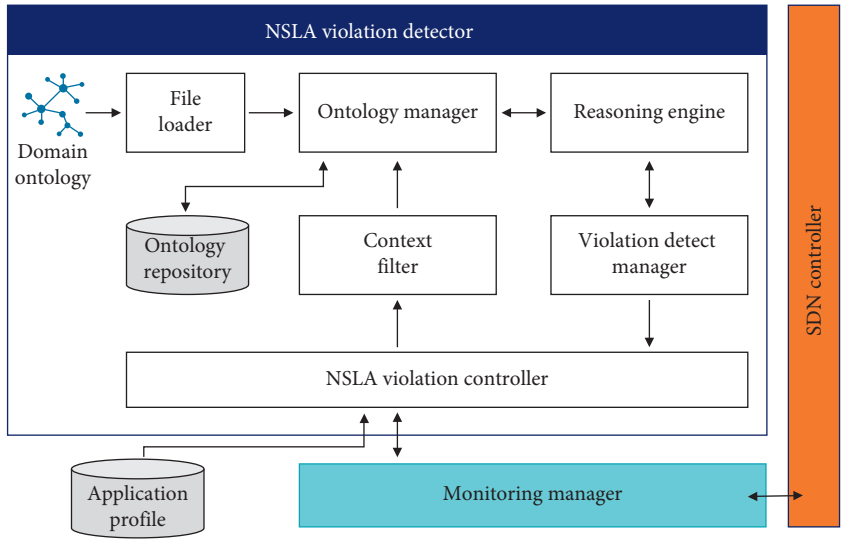


Figure 7: Internal structure used for the violation detection.

Table 4: Identified class, attribute, and data property information.

Class	Attribute	Data property	Range	Description
Application	ID, name, admin, IP address (v4, v6), application type	hasName	String	Application name
		hasAdmin	String	Application administrator
		hasIPAddress	String	Application IP address
		hasDescription	String	Application description
NSLA	Application ID, bandwidth, packet loss, delay, jitter	hasApplicationID	String	Application ID
		hasBandwidth	Int	Bandwidth
		hasPacketloss	Double	Packet loss
		hasDelay	Double	Delay time
		hasJitter	Double	Variation of delay time
Violation	Name, occurred time, target application, reason	hasName	String	Violation name
		hasOccurredTime	Datetime	Time of occurrence
		hastargetapplicaton	String	Target application
		hasreason	String	Reason for occurrence

each QoS value unit is different. Equation (3) is the T-score formula:

$$\mathrm{T-score}\,(\mathrm{QoS}) = \left(\frac{(\mathrm{QoS}) - \mathrm{average}\,(\mathrm{QoS})}{\mathrm{standard\ deviation}\,(\mathrm{QoS})} \times 20\right) + 100. \tag{3}$$

(6) Calculate each QoS cost by multiplying the calculated T-score by the weight according to the application type. The final cost is calculated by summing each QoS cost of the computed node, as presented in the following equation:

$$\begin{aligned} \text{node cost} &= \mathrm{T-score} \times \mathrm{weight}_{\mathrm{QoS}}, \\ \text{final cost} &= \sum_{i=1}^{4} \text{node cost}. \end{aligned} \tag{4}$$

(7) The final good-quality path is calculated as the minimum cost.

The GA performs the following application process:

(1) The topology of each of the four QoS choices is composed of adjacency matrices [26]. The path from the source to the destination is randomly selected and expressed as the initial population (1: selected, 0: not selected). Let T be the topology and S be the topology nodes. An initial population is formed by a bit pattern re2presented by 0 and 1 according to Equation 5:

$$\forall S_i \in T = \begin{cases} 1, & S_i \in \text{routing path}, \\ 0, & \text{else}. \end{cases} \tag{5}$$

(2) A fitness assessment is conducted for the initial population. The criterion of the fitness evaluation is the measurement of whether or not the value of the node QoS element of each path exceeds the NSLA of the application. The fitness score increases by 1 point if the QoS value of the node satisfies the NSLA. Equation (6) shows the fitness evaluation case of the NSLA bandwidth:

$\forall$ pair of node (s_i, s_j), where $i \neq j$ and directly connected,

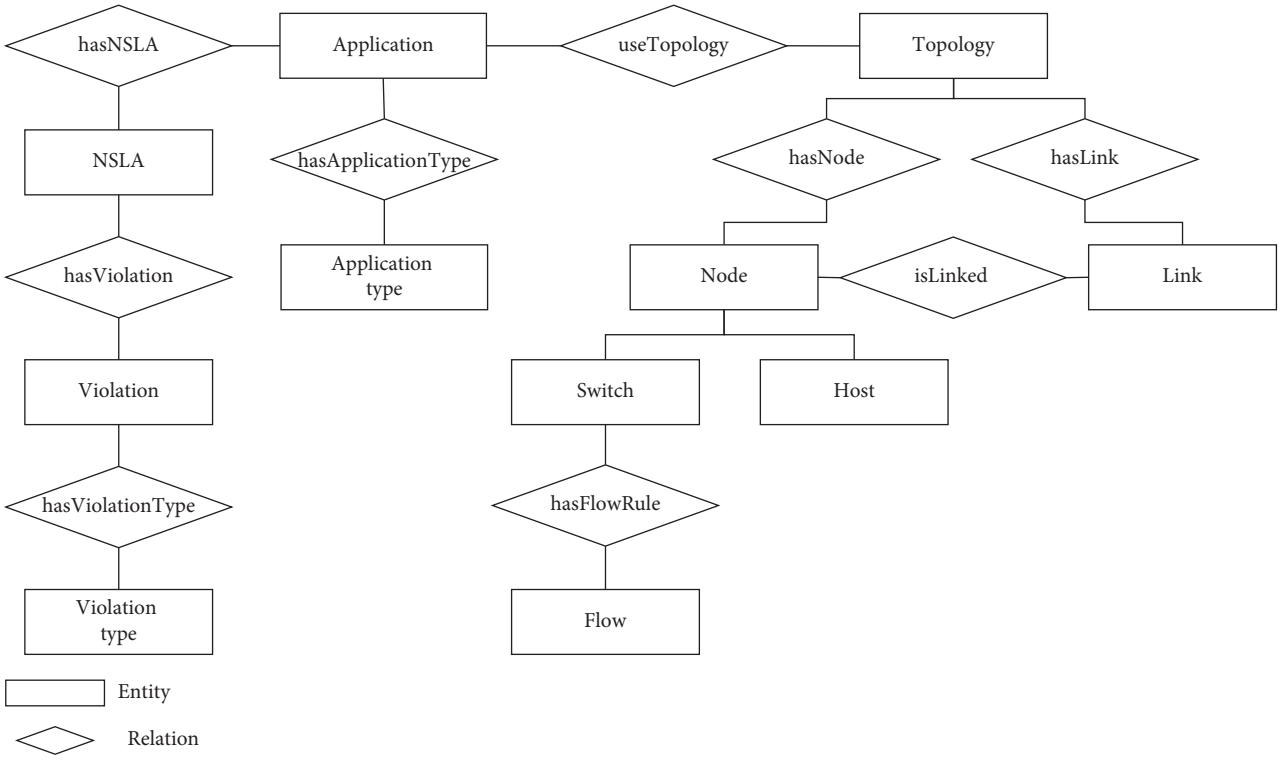


Figure 8: Ontology metamodel.

$$\text{then bandwidth QoS}_{ij} = \begin{cases} 1, & \text{if value of bandwidth QoS}_{ij} \\ & \geq \text{value of NSLA (bandwidth)}, \\ -1, & \text{else}, \end{cases}$$
$$\text{fitness evauation (bandwidth QoS)} = \sum \text{bandwidth QoS}_{ij}. \tag{6}$$

Figure 9 depicts an example in which the fitness evaluation formula is applied to determine if the bandwidth corresponding to the initial generation of the solution meets the application's NSLA.

(3) The scores from the evaluation of the fitness of the path selected as the initial generation are then divided by the number of hops and calculated as the final fitness score. Equation (7) is the fitness evaluation formula. Figure 10 shows an example of the final fitness score obtained using

$$\text{QoS fitness} = \frac{\sum_{h=1}^{\text{Hop Count}} \text{path (QoS)}}{\text{hop count}}. \tag{7}$$

(4) The top 20% of the early generations with the best fitness evaluation is judged as the dominant gene and mated or mutated to the next generation. We determined herein whether or not crossing is possible. If the gene cannot be crossed, it generates a child gene through mutation. Figure 11 shows the concept of crossover and mutation.

(5) Steps 2 to 4 are repeated on the new genes until a path with four points of target fitness score satisfying the NSLA satisfies all the four QoS factors and derives the paths.

4. Case Study of the NSAF

A prototype was implemented to evaluate the proposed NSAF and its process using Floodlight controller and Mininet [27]. Figure 12 shows the application registration. Figure 13 illustrates the network status monitoring of the NSAF that reflects the model and element described in Sections 3.3 and 3.4.

Figure 12 shows the application registration interface. The data to be input on the interface comprise the application manager member information (ID, password, manager name, and number to be contacted) and the application profile information (application name, IP address, application type, and application description). The NSLA's criteria will be set depending on the application type. The QoS criteria for each application type are based on the DiffServ standard.

S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
1	1	1	0	1	0	0	0	1	0

Initial population

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
S1		150	80							
S2	150			100	120					
S3	80			100			100	100	60	
S4		100	100		100	100				
S5		120		100		100				
S6				100	100		100			100
S7			100			100				100
S8			100						100	
S9			60					100		80
S10						100	100		80	

Bandwidth adjacency matrix

QoS	Value
Bandwidth	100
PacketLoss	5
Delay	5.0
Jitter	5.0

Application NSLA

Src	Dst	QoS	Fitness
S5	S2	120	+1
S2	S1	150	+1
S1	S3	80	−1
S3	S9	60	−1

Fitness evaluation

→ Bandwidth fitness score = 0

Figure 9: Examples of initial population fitness evaluations.

QoS	Fitness	Hop count	Final fitness
Bandwidth	0	4	0
Delay	3		0.75
Jitter	3		0.75
PacketLoss	2		0.5

= 2.0

Figure 10: Final fitness score.

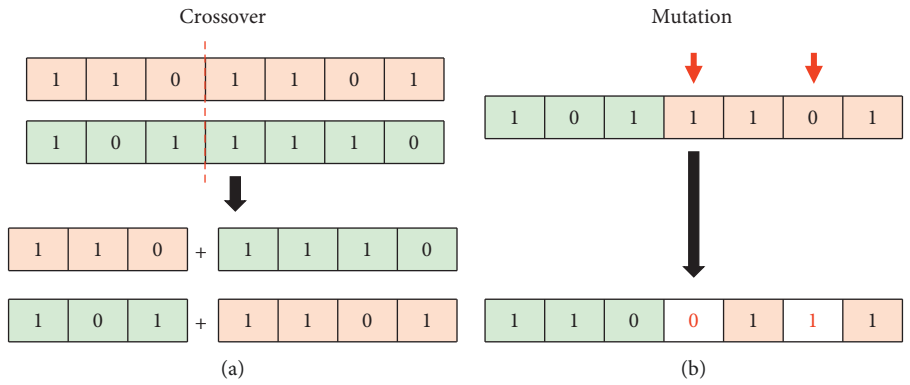


Figure 11: (a) Crossover and (b) mutation.

Figure 13 depicts that the NSAF Network Status Monitoring Interface connects and calls the Floodlight controller's data. It shows the structure of the switches and hosts that make up the topology at a glance and displays simple information on the top right by selecting each switch and host. The network monitoring provided by the NSAF supports not only topology monitoring but also information about the switches and the hosts connected to the SDN controller and dashboard monitoring that visually shows real-time network traffic.

An ontology model for detecting the NSLA violation was constructed using Protégé 5.0. Figure 14 shows the implemented ontology.

Based on the NSAF ontology, the SWRL rule shown in Table 5 was created to test the NSLA violation status defined in Section 3.5. Figure 15 shows the test application multimedia streaming application called PNU Tube. The NSLA information for the application is as follows: a bandwidth of 100 Mbps, a packet loss of 20 bytes, a delay under 1.5 ms, and a jitter under 0.5 ms.

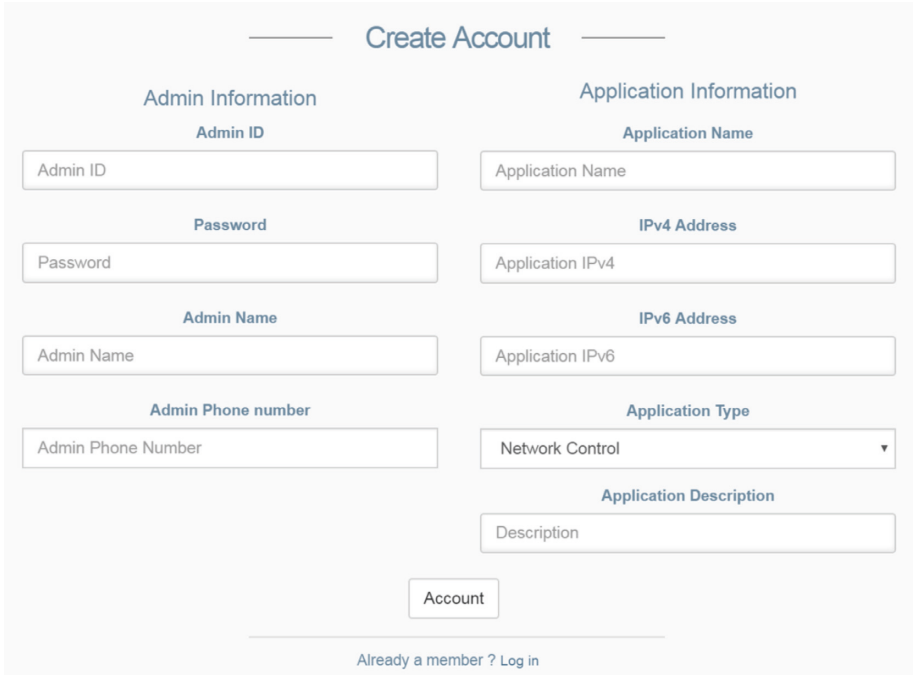


Figure 12: NSAF application registration interface.

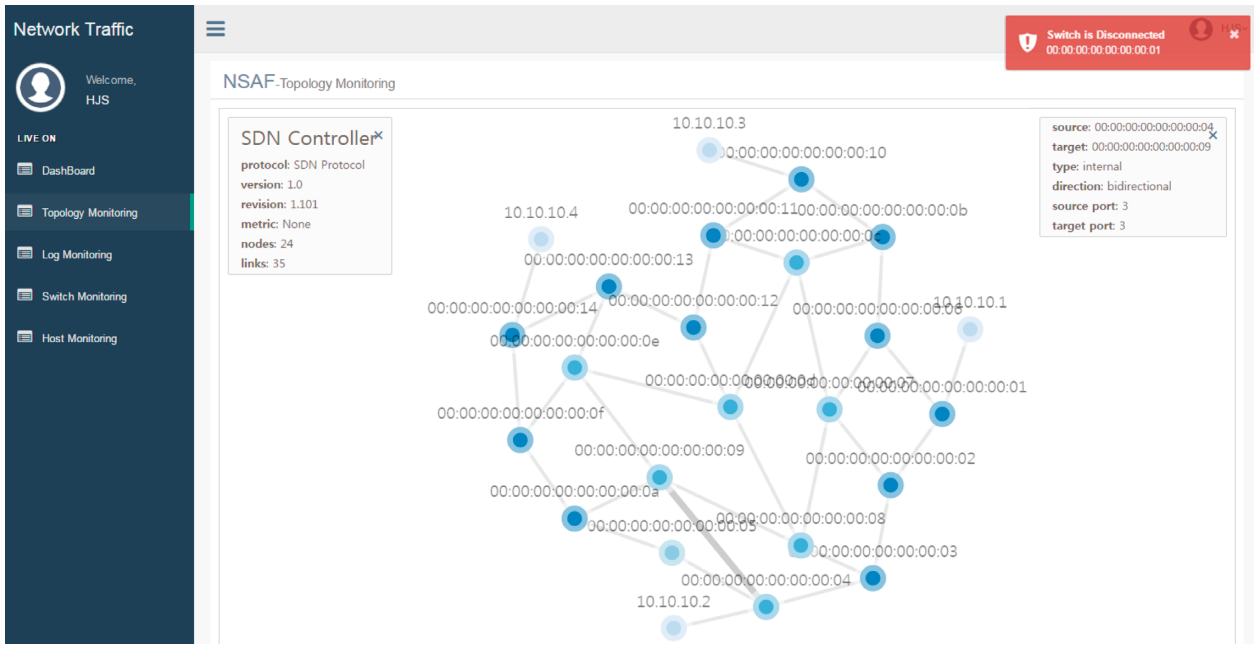


Figure 13: NSAF Network Status Monitoring Interface.

The virtual topology structure was composed of 10 switches such that the NSLA violation status can be determined. The path that the application used to send packets from H1 to destination H2 was S5 ⟶ S6 ⟶ S7 ⟶ S3 ⟶ S9. Using the application profile information and the topology QoS value, we confirmed that a soft fail situation occurred by applying the SWRL rules, as shown in the example in Table 5. When we checked the result of discrimination of the NSLA violation, three soft fail-type violations were found to have occurred. The QoS value for the S3 bandwidth, delay, and packet loss did not satisfy the NSLA; hence, bandwidth, delay, and packet violations occurred.

The following is an example of the bandwidth violation among the SWRL rules defined in Table 5: [Application (?A)] denotes the presence of an application called "a." [hasNSLA (?a, ?n)] indicates that "a" has an NSLA of "n." [hasBandwidth (?n, ?nb)] has a bandwidth value of "nb." [useTopology (?a, ?t)] uses a topology named "t," while hasNode (?t, ?s)] has a node named "s." [hasBandwidth (?s, ?sb)] indicates that node "s" has a bandwidth value of "sb." [swrl: greaterThan (?nb, ?

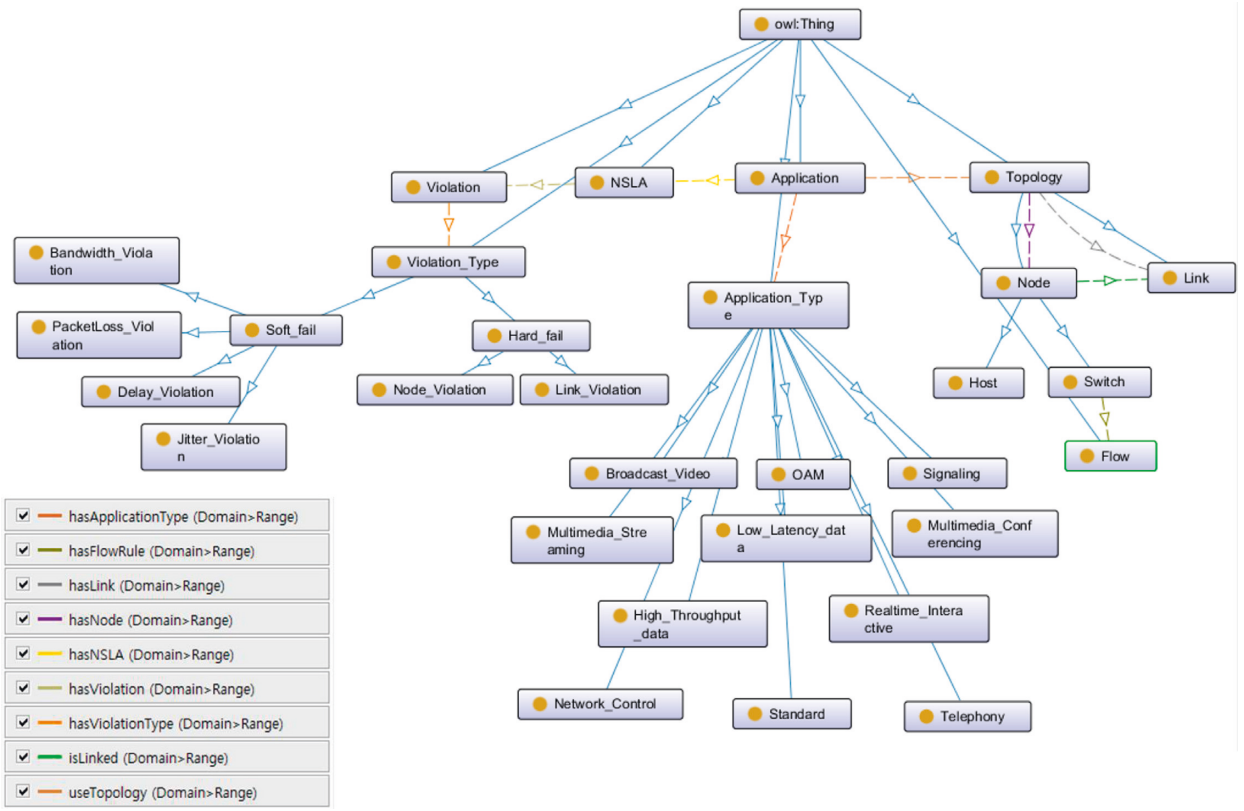


Figure 14: Implemented NSAF ontology.

Table 5: Example SWRL rule for the violation detection.

Violation type	Details of the violation type	SWRL rule
Hard fail	Node violation	Application (?a)ˆuseTopology (?a, ?t)ˆhasNode (?t, ?n)ˆhasState (?n, false) -> Node_Violation (?a)
Soft fail	Bandwidth violation	Application (?a)ˆhasNSLA (?a, ?n)ˆhasBandwidth (?n, ?nb)ˆuseTopology (?a, ?t)ˆhasNode (?t, ?s)ˆhasBandwidth (?s, ?sb)ˆswrlb:greaterThan (?nb, ?sb) -> Bandwidth_Violation (?a)

sb)] is a syntax that compares the NSLA bandwidth value "nb" with the bandwidth value of the topology node. Therefore, it is deduced that [->Bandwidth_Violation (?a)], that is, application "a" occurred as a bandwidth violation.

As an example, we tested the path that guarantees the network QoS by setting S2 as the starting node and S28 as the destination node in the virtual environment composed of 30 nodes (Figure 16) and executing Dijkstra's algorithm and the GA presented in Section 3.6.

Figure 17 presents an example of the logarithm of the values calculated in the network configuration environment of Figure 18 on the NSAF according to Dijkstra's algorithm, which was presented in Section 3.6.

As shown in the log in Figure 17, the individual T-scores for each QoS were calculated, and the node with the lowest cost was selected by summing the node cost. Finally, the path should be {S2, S7, S12, S13, S18, S23, S28}.

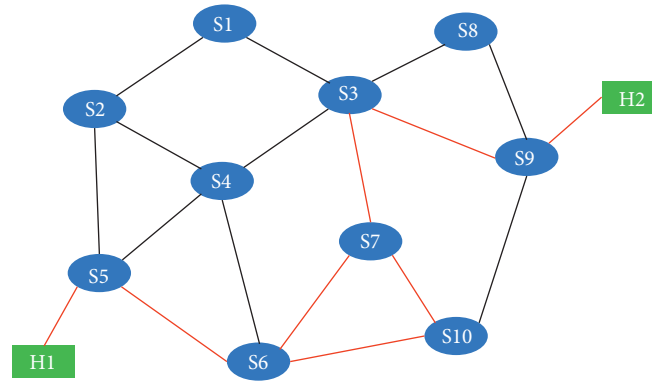
Figure 19 shows the result of the GA application. Figure 18 illustrates the calculated path (i.e., paths 1, 2, and 3). As shown in the results of the execution log of the NSAF on the left side, we performed crossover and mutation on the selected initial solution and confirmed that a number of alternative paths with a fitness score of 4 were derived.

5. Related Work and Discussion

Various studies have been conducted as regards support monitoring in SDN environments. Isolani et al. [28] proposed an SDN interactive manager to monitor, visualize, and configure SDN with the administrator in the management loop. Their proposed system includes a management plane that is responsible for managing elements in other SDN layers, such as monitoring device status and allocating resources. Their SDN interactive manager sits in this proposed

Application name	PNU tube
Application type	Multimedia streaming
IP address	164.125.0.1
NSLA	Bandwidth: 100 mbps Packetloss: 20 bytes Delay: 1.5 ms Jitter: 0.5 ms

(a)



(b)

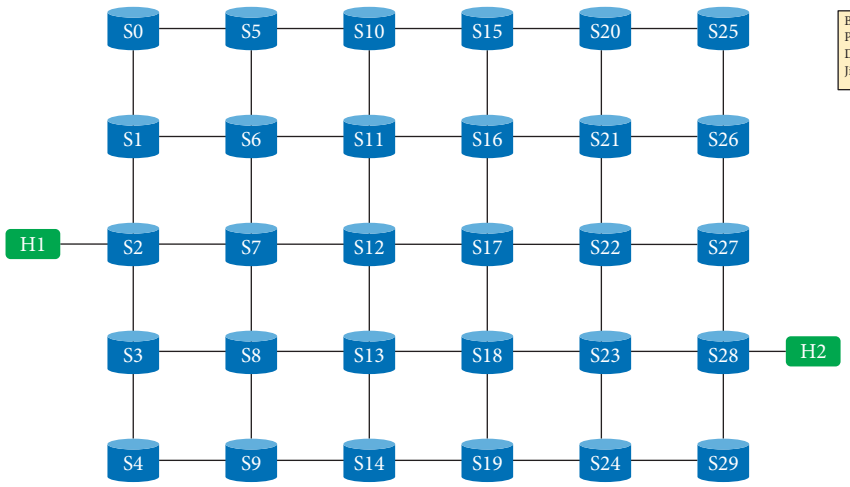
Switch	Bandwidth	Packetloss	Delay	Jitter
S5	100 mbps	10 bytes	1.0 ms	0.3 ms
S6	100 mbps	8 bytes	0.8 ms	0.4 ms
S7	100 mbps	11 bytes	1.2 ms	0.4 ms
S3	80 mbps	25 bytes	3.7 ms	0.3 ms
S9	100 mbps	8 bytes	0.9 ms	0.1 ms

(c)

```
{
  "softfail" : {
    "Delay_Violation" : "true",
    "Bandwidth_Violation" : "true",
    "PacketLoss_Violation" : "true
  },
  "hardfail" : {}
}
```

(d)

Figure 15: Violation detection: (a) application information; (b) routing path in use (red line); (c) routing table information; (d) violation detection result.



Bandwidth: 80 Mbps
PacketLoss: 20 bytes
Delay: 0.3 ms
Jitter: 0.4 ms

Application NSLA

	S2	S3	S4	…	S8	S9	…
S2		55					
S3	55		60		90		
S4		60					
…							
S8		90				120	
S9					120		
…							

Bandwidth adjacency matrix

Figure 16: Network environment and application NSLA.

plane and comprises three main components, namely, monitoring manager, visualization manager, and configuration manager.

Raumer et al. [29] proposed a flow sampling application that receives information from the analyzer module, called MonSamp, for QoS monitoring. In their proposed system, the monitor is directly connected to SDN and receives part of the network traffic information. MonSamp can reactively decide to reduce the number of flows using the received information. It is an SDN application for extraction and direct sampling of the network traffic.

Chowdhury et al. [30] proposed a network statistics collection framework called PayLess. The proposed framework operates on the top of the control layer and uses the northbound API of the controller. It consists of a request interpreter, a scheduler, a switch selector, and an aggregator and data store. The request interpreter translates high-level primitives expressed by the application to flow-level

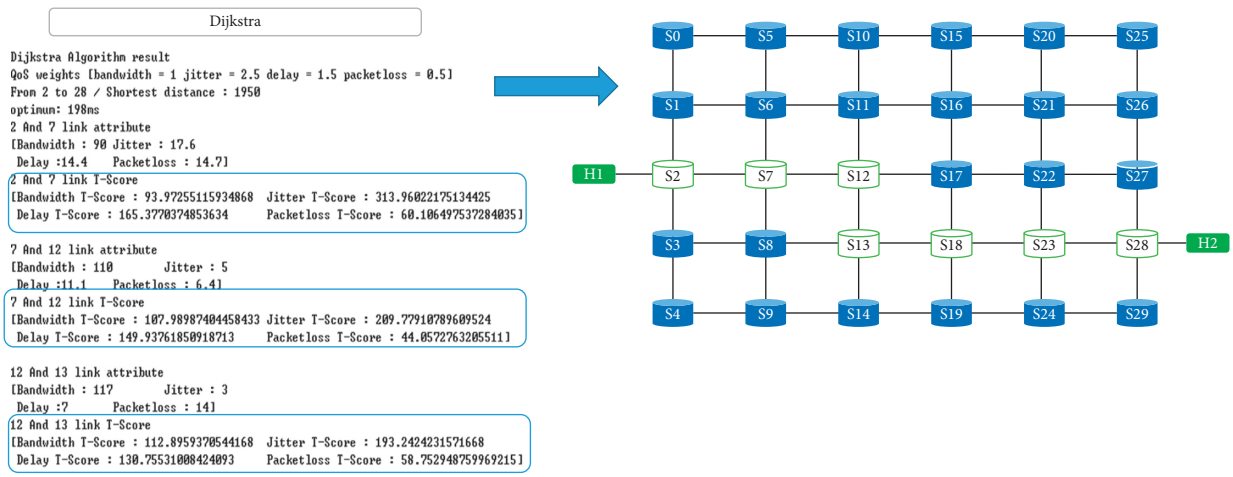


Figure 17: Dijkstra's path calculation results.

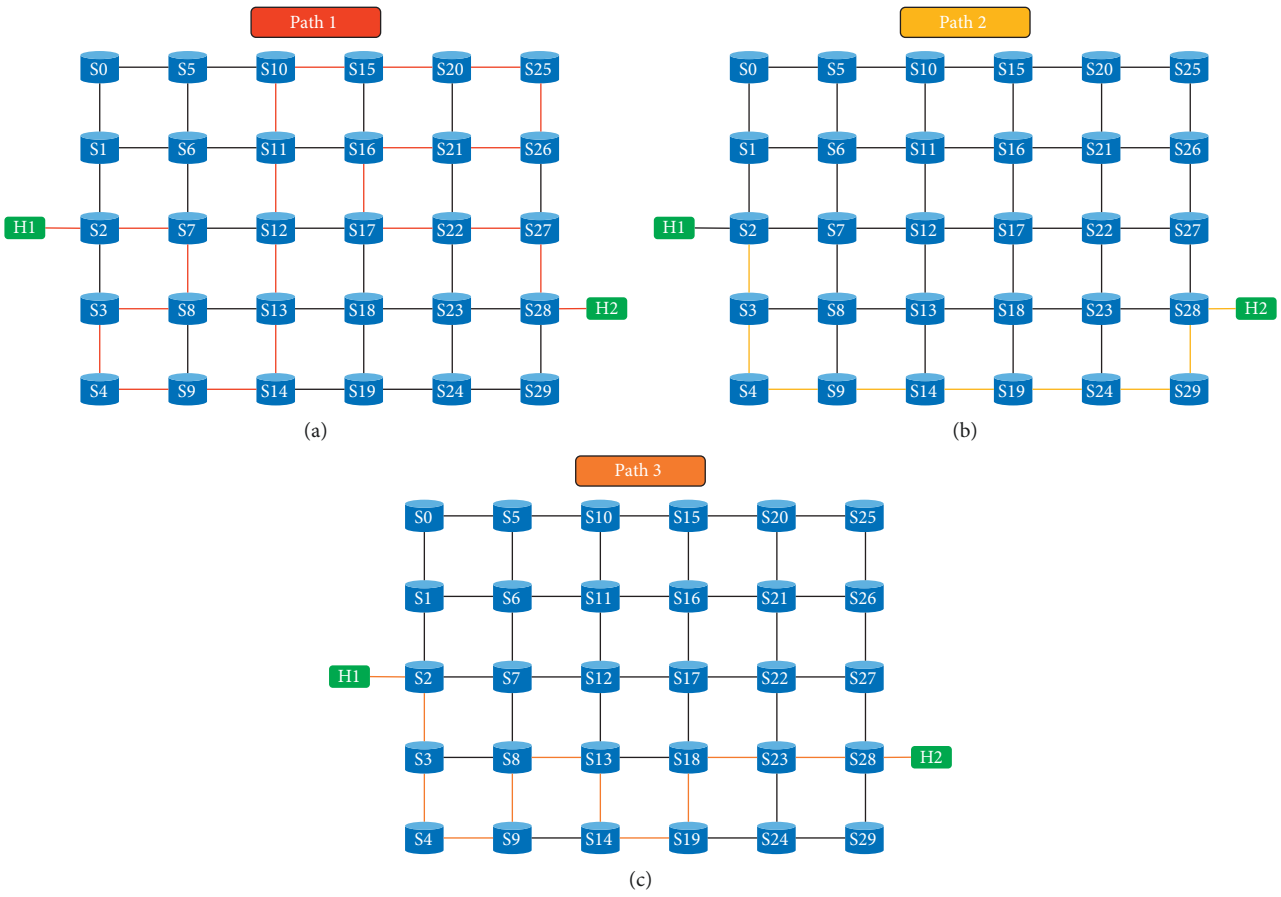


Figure 18: Path results.

primitives. The scheduler schedules polling of switches. The switch selector selects switches for the statistics collection. The aggregator and data store collects raw data from the selected switches and stores these raw data in the data store.

As outlined earlier, an element can be used to monitor the application QoS, but the focus of this element is on monitoring the network resources in the SDN environment, such as visualizing the current network status and providing the traffic statistics information, rather than the application QoS information. To the best of our knowledge, no approach that guarantees the QoS of the overall application from application QoS registration to the calculation of the QoS guaranteed path to guarantee the QoS of the application has yet been proposed.

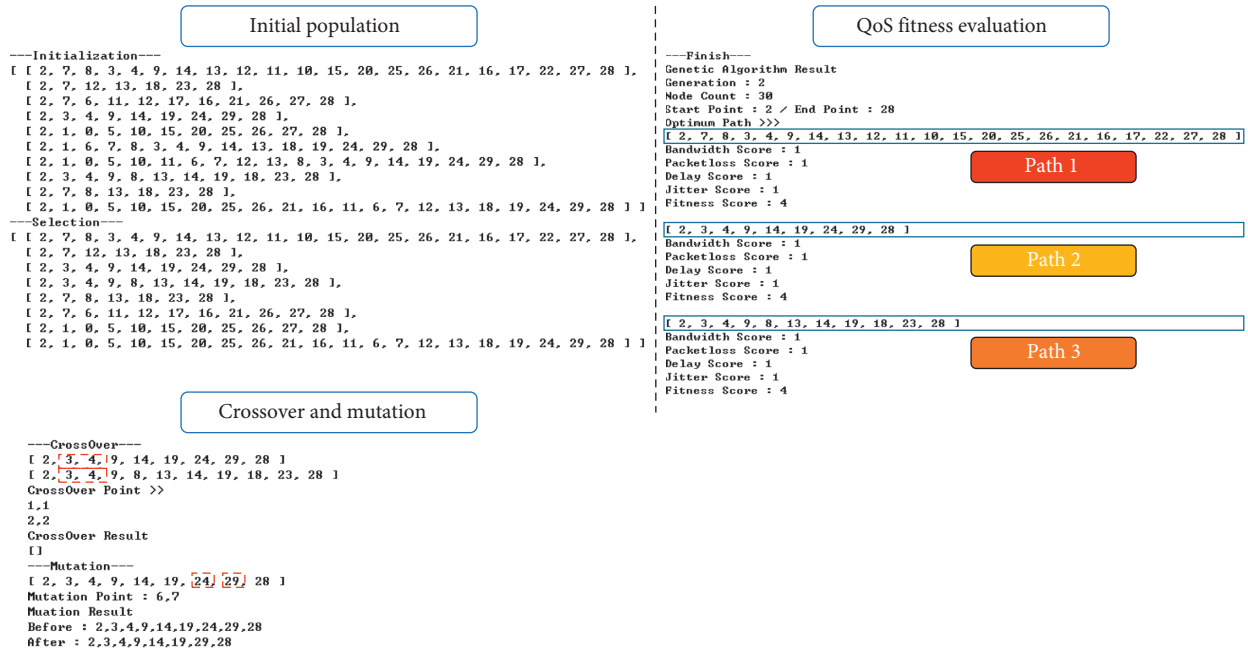


Figure 19: Genetic path calculation results.

Table 6: Execution time of the NSAF.

Framework element	Execution time
Collecting network status information	Collect network status information from the SDN controller every 10 s
Determining application QoS violations	Perform within 0.4 s to derive results
Route discovery to meet application QoS	Dijkstra's algorithm: calculate one path within 0.3 s; genetic algorithm: derive various paths satisfying the QoS within 5 s

The SDN controller herein was controlled using the NSAF without the need for expert knowledge of SDN. Moreover, application-aware routing was supported to improve usability. The NSAF operated between the control and application layers; hence, reusability can be improved by reusing the architecture even if an SDN controller integration environment is introduced in the future. The violation was identified through the ontology model; hence, in the case of another new violation situation being added, it can be simply discriminated by defining an associated SWRL rule. Extensibility can, thus, be increased.

We implemented the NSAF prototype that guarantees the application QoS and verified the feasibility of the proposed method. Table 6 shows the execution time of the proposed NSAF. The path derivation time to guarantee the QoS of the application was composed of 20, 30, 40, and 50 nodes of the virtual topology. The average time to route derivation was measured by executing the algorithm for five times.

The NSAF utilized Dijkstra's algorithm to apply T-score and GA for optimization and guarantee the application QoS. Dijkstra's algorithm is fast; hence, the route must be calculated using Dijkstra's algorithm first in case of a QoS violation. However, if a problem occurs in the path calculated using Dijkstra's algorithm, multiple paths satisfying the QoS can be found as backups using GA because Dijkstra's algorithm computes only one path satisfying the minimum cost. The NSAF applied a mechanism that adjusts the application to run continuously by changing the path to ensure the application QoS.

In addition, we conducted a qualitative comparison evaluation of the framework proposed herein and monitoring research in SDN environments. The evaluation criteria were compared with those of the monitored aspects and those of the application's QoS monitored aspects. We compared application QoS monitoring and QoS monitoring methods and investigated whether or not path control considering QoS is supported. Table 7 presents the related studies and comparative assessments.

6. Conclusion

This study proposed a framework, called NSAF, that guarantees the application QoS by recognizing the QoS requirements, ascertaining QoS violation status, and discovering network paths that satisfy the application QoS to support a stable application operation in SDN environments.

The NSAF made it possible to register 12 types of DiffServ applications and set the application QoS on measurable

Table 7: Comparison with the related studies.

Criteria		Interactive monitoring [28]	MonSamp [29]	PayLess [30]	NSAF
Monitored targets		SDN controller and device	Traffic pattern	Application QoS	Application QoS and network path that satisfy application QoS
Application QoS monitoring aspects	How to monitor QoS	Registration of the user traffic profile (configuration parameter setting)	Flow sampling application (switch flow rule)	Monitoring request object creation (specification)	NSLA based on the ontology (binding value at the ontology)
	Route control with QoS	Not supported	Feedback from the flow sampling application	Not supported (only monitoring)	Route control using NSAF
	How to ensure QoS	Not supported (visualization support about the monitoring result)	Not presented	Not presented (statistical result about monitoring)	(i) QoS violation detection using the ontology (ii) Network routing with Dijkstra's algorithm and GA considering QoS

quality factors, such as bandwidth, packet loss, delay, and jitter. The NSAF was supported by a proposed application profile model. Furthermore, a topology model was used in the proposed framework to monitor the network status in terms of the application QoS violation. We also classified the application violations as either hard fail or soft fail and identify the application QoS violations based on the ontology.

Dijkstra's algorithm was applied to compute the path cost of four QoS in the network path search satisfying the application QoS, and the alternative optimized paths were determined using GA.

The proposed framework can be used as a reference structure to change the network paths of applications according to their QoS requirements and varying network status. In the future work, we plan to expand the proposed NSAF to predict the changes in the network and control paths when applications are executed.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (nos. NRF-2014R1A1A2007061, NRF-2016R1D1A1B03935865, and NRF-2017R1D1A1B03030243).

References

[1] M. Jarschel, T. Zinner, T. Hossfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, attributes, and use cases: a compass for SDN," *IEEE Communications Magazine*, vol. 52, no. 6, pp. 210–217, 2014.

[2] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2015.

[3] A. Mediola, J. Astorga, E. Jacob, and M. Higuero, "A survey on the contributions of software-defined networking to traffic engineering," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 918–953, 2017.

[4] S. Sezer, S. Scott-Hayward, P. Chouhan et al., "Are we ready for SDN? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[5] K. M. Modieginyane, B. B. Letswamotse, R. Malekian, and A. M. Abu-Mahfouz, "Software defined wireless sensor networks application opportunities for efficient network management: a survey," *Computers & Electrical Engineering*, vol. 66, pp. 274–287, 2018.

[6] M. Jammal, T. Singh, A. Shami, R. Asal, and Y. Li, "Software defined networking: state of the art and research challenges," *Computer Networks*, vol. 72, pp. 74–98, 2014.

[7] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Distributed SDN controller system: a survey on design choice," *Computer Networks*, vol. 121, pp. 100–111, 2017.

[8] Project Floodlight, 2018, http://www.projectfloodlight.org.

[9] OpenDaylight, 2018, https://www.opendaylight.org.

[10] ONOS, 2018, http://onosproject.org.

[11] Ryu, 2018, https://osrg.github.io/ryu.

[12] OPenFlow, 2018, https://www.opennetworking.org/technical-communities/areas/specification/open-datapath.

[13] Cisco Systems:OpFlex:An Open Policy Protocol White Paper, 2018, http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.html.

[14] T. Tsai, K. Wang, and T. Y. Chao, "Dynamic flow aggregation in SDNs for application-aware routing," in *Proceedings of the 2016 10th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, Prague, Czech Republic, July 2016.

[15] Y. Desmouceaux, P. Pfister, J. Tollet, M. Townsley, and T. Clausen, "6LB: scalable and application-aware load

balancing with segment routing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 819–834, 2018.

[16] L. Cheng, K. Wang, and Y. Hsu, "Application-aware routing scheme for SDN-based cloud datacenters," in *Proceedings of the 2015 Seventh International Conference on Ubiquitous and Future Networks*, pp. 820–825, Sapporo, Japan, July 2015.

[17] J. Chan, J. Babiarz, and F. Notel Baker, *Cisco Systems: Aggregation of Diffserv Service Classes*, Network Working Group, Internet Engineering Task Force (IETF), Fremont, CA, USA, 2018, https://tools.ietf.org/html/rfc5127.

[18] OWL (web ontology language), 2018, http://www.w3.org/2004/OWL.

[19] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen, "From SHIQ and RDF to OWL: the making of a web ontology language," *Journal of Web Semantics*, vol. 1, no. 1, pp. 7–26, 2003.

[20] SWRL: A Semantic Web Rule Language Combining OWL and RuleML, 2018, https://www.w3.org/Submission/SWRL.

[21] T. M. de Farias, A. Roxin, and C. Nicolle, "SWRL rule-selection methodology for ontology interoperability," *Data & Knowledge Engineering*, vol. 105, pp. 53–72, 2016.

[22] H. J. Holland, *Adaptation in Natural and Artificial Systems*, The MIT Press, Cambridge, MA, USA, 1992.

[23] R. Rojas, *Neural Networks—A Systematic Introduction*, Springer-Verlag, Berlin, Germany, 1996.

[24] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*, John Wiley & Sons, Hoboken, NJ, USA, 2000.

[25] M. Karakus and A. Durresi, "Quality of service (QoS) in software defined networking (SDN): a survey," *Journal of Network and Computer Applications*, vol. 80, no. 15, pp. 200–218, 2017.

[26] R. B. Bapat, "Adjacency matrix," in *Graph and Matrices*, Springer, Berlin, Germany, 2010.

[27] Mininet, 2018, http://mininet.org.

[28] P. H. Isolani, J. A. Wickboldt, C. B. Both, J. Rochol, and L. Z. Granville, "Interactive monitoring, visualization, and configuration of OpenFlow-based SDN," in *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 207–215, Ottawa, Canada, May 2015.

[29] D. Raumer, L. Schwaighofer, and G. Carle, "MonSamp: a distributed SDN application for QoS monitoring," in *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, pp. 961–968, Łódź, Poland, September 2014.

[30] S. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "PayLess: a low cost network monitoring framework for Software Defined Networks," in *Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–9, Kraków, Poland, May 2014.