Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2022) 30: 263 – 278

doi:10.3906/elk-2104-182

An effective prediction method for network state information in SD-WAN

Erdal AKIN[1,*], **Ferdi SARAÇ**[2], **Ömer ASLAN**[3]
[1]Department of Computer Engineering, Bitlis Eren University, 13000 Bitlis, Turkey
[2]Department of Computer Engineering, Süleyman Demirel University, 32260 Isparta, Turkey,
[3]Department of Computer Engineering, Siirt University, 56100 Siirt, Turkey

Received: 26.04.2021 • **Accepted/Published Online:** 02.07.2021 • **Final Version:** 19.01.2022

Abstract: In a software-defined wide area network (SD-WAN), a logically centralized controller is responsible for computing and installing paths in order to transfer packets among geographically distributed locations and remote users. Accordingly, this would necessitate obtaining the global view and dynamic network state information (NSI) of the network. Therefore, the centralized controller periodically collects link-state information from each port of each switch at fixed time periods. While collecting NSI in short periods causes protocol overhead on the controller, collecting in longer periods leads to obtaining inaccurate NSI. In both cases, packet losses are inevitable, which is not preferred for quality of service (QoS). Packet loss needs to be reduced by minimizing the protocol overload on the controller and collecting accurate NSI to provide better QoS. This work proposes an effective prediction method for collecting NSI (PM-NSI) that significantly reduces packet loss and controller protocol load allowing the controller to collect accurate NSI in longer periods. The proposed method is compared against the existing NSI collection method, which collects NSI periodically, in use on the RYU controller and the Mininet emulator by using a dynamic routing algorithm. The test results indicated that PM-NSI reduces controller load around 1000% by collecting NSI in longer periods and so outperforms the existing periodic NSI collection method in terms of packet loss, jitter, controller load, and thus QoS.

Key words: Routing in SD-WAN, network state information, software-defined wide area network, load balancing

1. Introduction

The original IP network architecture became insufficient to meet the growing real-world needs. Patching new features to the existing network architecture to handle the requirements results in a complex distributed system [1–3].

Software-defined networking (SDN) emerged as a new technology that promises to give the ability to manage and program the Internet [1, 4]. Basically, SDN separates the control plane from multiple data-plane elements into a logically centralized controller that acquires the link state information of the network, fulfills all decisions (e.g., computing feasible paths for the requested flows), and installs these decisions to the underlying forwarding elements [1, 5–7]. SDN principles are used to manage wide area network (WAN) which is called software-defined wide area network (SD-WAN).

While SD-WAN aims to enable managing the network, it brings about new scalability issues such as signaling overhead, communication delay and reliability, fault tolerance issues, and protocol overhead [8]. Signaling overhead occurs because of communication needs between separated control and data plane. Communication

*Correspondence: e.akin@beu.edu.tr

delay and reliability happen due to the distance between the controllers and the switches. Since there is one point of management, fault tolerance issue is inevitable if the controller fails. Last but not least, the protocol overhead appears because of collecting NSI and dealing with the statistics that switches send [9]. Although these scalability issues are important and affect each other, the protocol overhead is crucial especially for the routing decisions.

Routing in SD-WAN is performed in three main steps: the controller (i) periodically communicates with the switches to collect dynamically changing statistics and NSI, (ii) uses a routing algorithms to find a feasible path for the demand of the requested flow, (iii) installs the computed routing information to the underlying forwarding elements such as switches [4, 6]. Let us explain the process better in Figure 1. The controller continuously requests NSI from the underlying network (switches S1, S2, and S3) with a predetermined period. Then, it acquires link state information such as available bandwidth and residual bandwidth for each link. If a flow request with demand from Host 1 to Host 2 arrives, it finds a path for the demand by using a routing algorithm. Assuming that the computed path is (S1, S3), the controller installs the forwarding rules on the switches S1 and S3.

The controller collects statistics such as flow, meter, queue, aggregate, table, and port (link) statistics by querying the switches via statistics messages to acquire dynamic NSI [10]. The controller needs to collect NSI more often within shorter periods to obtain accurate NSI. However, this causes messaging overhead (i.e. protocol overhead) on the controller. Nonetheless, collecting NSI with longer periods results in inaccurate NSI, as link statistics are changing quickly. In both cases, the delay and packet loss are inevitable [4]. There are a paucity of studies to address this scalability issue. The authors in [11, 12] aim at sending some traffic information to the authorized switches instead of the controller. These switches act like the controller which is unsuitable for the concept of SD-WAN. Many of the existing studies attempt to estimate traffic load on the network, but they do not take into account both the controller overload and the packet forwarding between multiple sources and destinations together [13–15].

To address the abovementioned issues, we propose a new prediction method, called the prediction method for network state information (PM-NSI), that targets the inaccuracy issues. In PM-NSI, we aim at predicting accurate NSI while collecting NSI at longer periods. To do that, we use variable packet size probing technique [16–20]. The main idea behind the technique is to measure the remaining bandwidth capacity of each link of the path by probing the demand of the flow requests. With this idea, we calculate possible bandwidth consumption for each link based on randomly chosen demands of each possible flow. Then, by using calculated possible bandwidth consumption and lastly collected NSI, we compute the possible remaining bandwidth of each link. Furthermore, we modify the dynamic shortest path (DSP) algorithm concerning the predicted link-state information to compute paths. Since PM-NSI allows the controller to collect more accurate NSI at longer periods, the protocol overhead on the controller is reduced significantly which results in less delay, jitter, and packet loss.

The periodic NSI method and the proposed PM-NSI method are implemented on the RYU controller and the Mininet emulator is used to compare them on an enhanced ANSNET topology (Figure 2). Our results show that PM-NSI outperforms the periodic collection method in terms of delay, jitter, and packet loss while significantly reducing the controller load by collecting accurate NSI at longer periods.

The main contributions of our study are outlined as follows:

- We propose a novel NSI prediction method which aims at collecting NSI at longer periods.

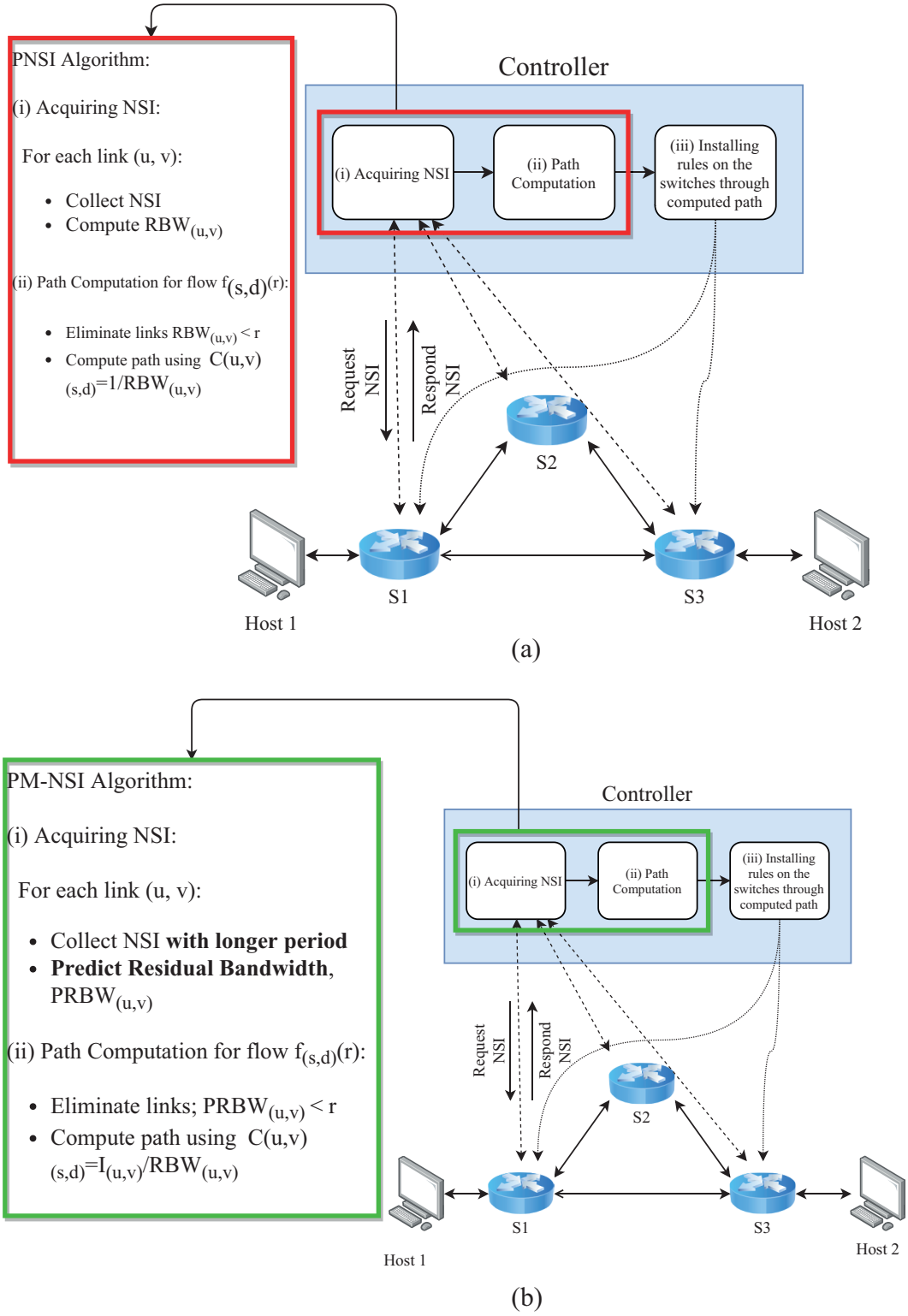

Figure 1. NSI Collection Methods (a) PNSI and (b) PM-NSI

- The effectiveness of the proposed NSI method has been extensively studied in terms of QoS by means of bench-marking against the periodic NSI collection method which is the method in use.
- It has been observed that the proposed NSI collection method obtains accurate NSI at longer periods which results in reducing the protocol overhead on the controller.
- Since the protocol overhead is reduced, the controller computes more feasible paths with better QoS in the proposed NSI collection method comparing the well-known periodic NSI collection method.

This paper is structured as follows. In Section 2, we mention the scalability issues in SD-WAN controller plane and periodic NSI collection in SD-WAN. We discuss the proposed PM-NSI method in Section 3. In Section 4, we provide the experimentation set up and evaluate the results. Finally, we state the challenges and the future studies in Section 5.

2. Background and related studies

The controller computes optimal paths by using one of the existing real-time routing algorithms concerning obtained current NSI. To acquire the current NSI, the controller queries all links (ports) of the switches via OpenFlow statistics messages. Using these messages, the controller queries the switch for information about its running states such as flow, meter, queue, aggregate, table, and port stats. Furthermore, one can create custom experimenter statistics by modifying the switches and the controller in terms of required custom statistics.

As evaluated in [4], the accuracy of NSI is crucial for the performance of a routing algorithm. However, collecting the accurate dynamically changing NSI is a very challenging issue. Unfortunately, frequently querying the distributed switches causes significant overhead and still inaccuracy because of delay and measurement errors. Moreover, collecting NSI at longer periods will result in nonoptimal path selection as it cannot provide current NSI, although it reduces overhead. Therefore, the controller needs to carefully pick a predetermined period (e.g., every $\mathscr{T}$ seconds) to balance the accuracy and protocol overhead [4, 21].

In practice, the controller communicates with all OpenFlow switches on the network by sending the port statistics. The switches reply with current total traffic, $tx^c_{(u,v)}$, for each port. The controller keeps the last recorded traffic on the port, $tx^p_{(u,v)}$. In other words, while $tx^c_{(u,v)}$ presents the total traffic on the link (u,v) at the current time t_c, $tx^p_{(u,v)}$ shows the total traffic on the link until the previously measured time t_p. The difference between t_p and t_c is as much as the predetermined period, $\mathscr{T}$ seconds. The controller, first, calculates the average current traffic ($\mathcal{ABW}_{(u,v)}$) on the link (u,v) which is formulated as follows:

$$\mathcal{ABW}_{(u,v)} = \frac{tx^c_{(u,v)} - tx^p_{(u,v)}}{t_c - t_p} \tag{1}$$

$\mathcal{ABW}_{(u,v)}$ is the ratio of the difference between $tx^c_{(u,v)}$ and $tx^p_{(u,v)}$ over the difference between t_p and t_c. Then, the controller determines residual bandwidth $\mathcal{RBW}_{(u,v)}$ for each link (u,v) by using the formula:

$$\mathcal{RBW}_{(u,v)} = \mathcal{BW}_{(u,v)} - \mathcal{ABW}_{(u,v)} \tag{2}$$

$\mathcal{BW}_{(u,v)}$ is the initial capacity of the link (u,v). Finally, the controller saves t_c and $tx^c_{(u,v)}$ as $t_p = t_c$ and $tx^p_{(u,v)} = tx^c_{(u,v)}$ for calculations in the next period.

These steps may cause scalability issues because of protocol overhead on the controller. There are some studies proposed to solve the issues. In [11, 12], the researchers give authority to some switches to prevent sending some requests to the controller to reduce protocol overhead. However, the proposed method is against the design of SD-WAN architecture.

In [22], the authors compare the centralized and the distributed controllers by taking into account all message traffic between the switches and the controllers. Despite the better performance of the multicontroller designs, they cause unnecessary delays corresponding to the routing algorithms because of the need for dynamic NSI. Therefore, to handle this issue, the authors propose a multicontroller design, called Link-prioritized NSI [21]. Although they provide more accurate NSI together with better or similar performance in controller load and total transferred bandwidth, they only take into account that the traffic goes through one direction.

In [14, 15], the number of packets, average traffic load, and path length is estimated. However, the authors focus on packet forwarding rather than controller overload and do not provide a quantitative evaluation. In [13], the authors propose a model with seven functions for predicting control channel. They achieve 94% prediction accuracy, but they do not take into account packet forwarding between multiple sources and destinations.

Accordingly, we propose a new prediction method for NSI (PM-NSI) that assumes the traffic can go through between all sources and destinations in the network. Despite proposing and testing PM-NSI for the single controller design, it can also be modified for multicontroller designs in the future. In Section 3.2, the proposed PM-NSI method is discussed in detail.

3. NSI collection methods

OpenFlow protocol has a set of messages sent from the controller to the underlying data plane elements and a set of corresponding messages responded from the data plane elements to the controller [27]. For the NSI collection, the controller requests the total traffic for all ports of all switches on the network and the switches respond back with the requested information. Then, the controller acquires link state information such as current traffic and residual bandwidth on the links and computes a feasible path for a flow request. Therefore, NSI collection methods consist of the two main steps of the routing protocol in SD-WAN (the red and green rectangle which consists of both steps in the controller in Figures 1a and 1b). In this section, we first present algorithm and the issues of the classical method PNSI which is practically in use. Then, we propose PM-NSI which targets to solve the issues of PNSI.

3.1. Periodic network state information (PNSI) collection

As we explained in detail in Section 2, the controller uses OpenFlow protocol to communicate with switches for collecting NSI. In this section, for better insight, we explain the steps of the PNSI method in Algorithm 1. Between lines 6 and 14, the controller collects NSI and computes $ABW_{(u,v)}$ and $RBW_{(u,v)}$ for each link (u,v), if the predetermined time period $\mathscr{T}$ is reached or passed. It first requests $tx^c_{(u,v)}$ from each switches in line 8. Then, as soon as the control receives $tx^c_{(u,v)}$, it computes $ABW_{(u,v)}$ and $RBW_{(u,v)}$ in lines 9 and 10 by using equations 1 and 3, respectively. Then, in lines 11 and 12, the controller saves current $tx^c_{(u,v)}$ and t_c as the previous variables in order to use for the next collection time. The controller uses lastly computed $RBW_{(u,v)}$ (line 15) for path computation. Finally, the controller computes a feasible path for the requested flow $f_{(s,d)}(r)$ in line 17. As indicated in [4], DSP is providing one of the best performances among the others. Therefore, we choose the DSP algorithm for PNSI which eliminates the links which are less than requested demand r and

Algorithm 1: Periodic Method for NSI (PNSI)

1 **Initialization:** Graph $G(N, E)$,
2 $BW(u, v)$ for $\forall(u, v) \in E$,
3 $\mathscr{T}$ is declared,
4 $f_{(s,d)}(r)$, a flow between source s and destination d with a requested demand r,
5 **Output:** $p_{(s,d)}$, a path that meets the demand r of a flow between source s and destination d.
6 **if** $t_c - t_p >= \mathscr{T}$ **then**
7 **for** *Each link* $(u, v) \in E$ **do**
8 The controller requests $tx^c_{(u,v)}$
9 $ABW_{(u,v)} = \frac{tx^c_{(u,v)} - tx^p_{(u,v)}}{t_c - t_p}$
10 $RBW_{(u,v)} = BW_{(u,v)} - ABW_{(u,v)}$
11 $tx^p_{(u,v)} = tx^c_{(u,v)}$
12 $t_p = t_c$
13 **end**
14 **else**
15 Controller uses lastly computed $RBW_{(u,v)}$.
16 **end**
17 For flow $f_{(s,d)}(r)$ Eliminate the link (u, v) that $RBW_{(u,v)} < r$, compute the shortest path $p_{(s,d)}$ using $C(u, v)_{(s,d)} = \frac{1}{RBW_{(u,v)}}$
18 **Return:** $p_{(s,d)}$

computes path $p_{(s,d)}$ based on the cost metric:

$$C(u, v)_{(s,d)} = \frac{1}{RBW_{(u,v)}} \tag{3}$$

Then, the controller installs rules to the switches on the computed path $p_{(s,d)}$. The main shortcoming in this method is that the controller needs to collect NSI in short periods (line 8) to acquire accurate NSI, which causes overload on the controller. In addition, if it collects in longer periods, it would have inaccurate NSI (line 15) which results in using unfeasible links during path computation and congestion on the links. Therefore, we aim at proposing a new method which provides a way for the controller to collect NSI in longer periods and reduces overload on the controller by predicting the link usage in advance and handling the inaccuracies. In the next section, we present the proposed method in detail.

3.2. Prediction method for NSI collection (PM-NSI)

3.2.1. Details of the proposed method

In our method, we aim at predicting usage of a link (u, v) between the last collection of NSI and the current flow request times. Since the dynamic NSI changes over time because of continuous traffic load, the controller will not have accurate NSI if it uses the previously collected NSI. Thus, instead of using inaccurate NSI, it is allowed the controller to predict current NSI at the time of flow requested. In this way, our method provides a way for the controller to collect NSI at longer periods resulting in less protocol overhead. To do that, first, it is necessary to determine Possible Flow Number ($PFN_{(u,v)}$) and Possible Bandwidth Consumption ($PBWC_{(u,v)}$) for each link (u, v). Then, by using this information, current remaining bandwidth $RBW_{(u,v)}$ on the link (u, v) can be predicted.

Algorithm 2: Proposed Prediction Method for NSI (PM-NSI)

1 **Initialization:** Graph $G(N, E)$,
2 $RBW(u, v)$ for $\forall(u, v) \in E$ computed at last collection time,
3 $f_{(s,d)}(r)$, a flow between source s and destination d with a requested demand r,
4 $I_{(u,v)}$, flow number on the link (u, v),
5 $|SD|$, number of possible (s, d) pairs in the network,
6 $M(u, v) = \{(s, d)$ pairs that using the link $(u, v)\}$ computed by Minimum-hop Disjoint Paths Algorithm [21] for $\forall(u, v) \in E$.
7 **Output:** $p_{(s,d)}$, a path that meets the demand r of a flow between source s and destination d with minimum protocol overhead on the controller.
8 $PFN_\alpha = \frac{t^c - t^p}{\alpha}$
9 $PBWC_{(s,d)} = \sum_{i=1}^{PFN_\alpha} \frac{uniform(r_{min}, r_{max})}{|SD|}$
10 **for** *Each link* $(u, v) \in E$ **do**
11 $PRBW^{temp}_{(u,v)} = RBW_{(u,v)} - PBWC_{(s,d)}$
12 $\mathscr{X} = |M(u, v)|$
13 **if** $\mathscr{X} \geq 1$ **then**
14 $PRBW_{(u,v)} = Pr(\mathscr{X}) \times PRBW^{temp}_{(u,v)}$
15 **else**
16 $PRBW_{(u,v)} = PRBW^{temp}_{(u,v)}$
17 **end**
18 **end**
19 For flow $f_{(s,d)}(r)$ Eliminate the link (u, v) that $PRBW_{(u,v)} < r$, compute the shortest path $p_{(s,d)}$ using $C(u, v)_{(s,d)} = \frac{\mathcal{I}_{(u,v)}}{PRBW_{(u,v)}}$
20 **if** $p_{(s,d)} \neq \emptyset$ **then**
21 **for** *Each link* $(u, v) \in p_{(s,d)}$ **do**
22 $\mathcal{I}_{(u,v)} = \mathcal{I}_{(u,v)} + 1$
23 **end**
24 **end**
25 **Return:** $p_{(s,d)}$

For set of nodes $N = \{u_1, u_2, u_3, ..., u_n\}$ and for the set of links $E = \{e_1, e_2, e_3, ..., e_m\}$. Each link $(u, v) \in E$ is associated with an available bandwidth parameter $BW(u, v) \geq 0$. For source node s, destination node d and bandwidth requirement r a flow is $f_{(s,d)}(r)$. For each flow f, the controller needs to find a single path p, which $BW(p) = \min\{BW(u, v) \mid (u, v) \in p\} \geq r$.

The proposed method is a combination of NSI prediction and routing algorithm that uses predicted remaining bandwidth of the links. As seen at *line* 8 in Algorithm 2, we first determine PFN_α, which is the number of the requested flows between the previous NSI collection time (t^p) and the current time (t^c) that the flow requested. Assuming that the flows are coming in every α seconds, PFN_α is the floor of the ratio of the difference between t^c and t^p to α. Then, in *line* 9 in Algorithm 2, we calculate possibly transferred bandwidth demand between t^p and t^c, which is named as Possible Bandwidth Consumption ($PBWC_{(s,d)}$). To calculate $PBWC_{(s,d)}$, we uniformly choose PFN_α demands from $uniform(r_{min}, r_{max})$ (r_{min} and r_{max} representing minimum and maximum demands, respectively) and find the average of them over the total number of possible (s, d) pairs ($|SD|$). Between *lines* 10 and 18 in Algorithm 2, we determine Predicted Remaining Bandwidth

($PRBW_{(u,v)}$) for each link (u,v). Therefore, we first compute the Temporary Predicted Remaining Bandwidth ($PRBW^{temp}_{(u,v)}$) at *line* 11. Then, we calculate the number of pairs using the link (u,v) ($\mathscr{X}$) (*line* 12). To determine $\mathscr{X}$, we need to use $M(u,v)$ which is a pre-computed parameter by using Minimum-hop Disjoint Paths Algorithm [21]. As explained in detail in [21], the algorithm computes $M(u,v)$ parameter which is the set of all possible (s,d) pairs that the minimum-hop edge-disjoint paths belonging to the pairs contain the link (u,v). Between *lines* 13 and 17, we calculate $PRBW_{(u,v)}$ that is used to determine the cost metric $C_{(u,v)}$ for path computation. To do that, it is essential to compute probability of $\mathscr{X}$ ($Pr(\mathscr{X})$) over the number of total (s,d) pairs, a.k.a $|SD|$. If $M(u,v)$ is not *null*, we assume that $Pr(\mathscr{X}) \times PRBW^{temp}_{(u,v)}$ of the link capacity is available for the flows. The goal here is to increase the cost of using the common links (*line* 14). Thus, the aim is to postpone the usage of common links to prevent early congestion. Otherwise, if $M(u,v)$ is *null*, any bandwidth capacity is not reserved to allow the routing algorithm to use the other links (non-common links) for better load balancing (*line* 16). After obtaining $PRBW_{(u,v)}$, $C(u,v)_{(s,d)}$, which is a new link cost metric, is introduced as follow:

$$C(u,v)_{(s,d)} = \frac{\mathcal{I}_{(u,v)}}{PRBW_{(u,v)}} \tag{4}$$

where $I(u,v)$ represents the total number of flows using the link (u,v). Then, DSP algorithm (which provides one of the best performance and is easy to implement among the routing algorithms in [4]) is modified in order to use this metric to compute paths (*line*19). Modified DSP finds a path for a flow $f_{(s,d)}(r)$ by using equation 4, while eliminating the links that $PRBW_{(u,v)}$ is less than the requested demand r. Then, if the path $p_{(s,d)}$ is found, $I(u,v)$ is updated for each link (u,v) belonging to $p_{(s,d)}$ (*lines* $20-24$). Then, the controller installs rules to the switches on the computed path $p_{(s,d)}$.

3.2.2. Computational time analysis

In the proposed PM-NSI method, a modified DSP algorithm is used to compute paths. Before using it, we try to predict the remaining bandwidth of the links. To do that, we use pre-computed $M(u,v)$ parameter [21]. As evaluated in detail in [21], the worst-case computation complexity of Minimum-hop Disjoint Path Algorithm is $\mathcal{O}(|N|^2\mathcal{L}(BFS))$, where $\mathcal{L}$ is the maximum degree and $|N|$ is the number of vertices in the network. However, these parameters are computed only once when the topology is created. So, the running time of the Algorithm can be ignored.

In Algorithm 2, initially possible flow numbers between the previous NSI collection time and the current time the flow has arrived are determined. Then, the total potential demand based on the possible flow numbers is estimated. The computation time complexity of predicting remaining bandwidth of the links is $\mathcal{O}(|E|)$, where $|E|$ is the number of the links in the network (Algorithm 2 lines 10-18). Between lines 20 and 24, $\mathcal{I}_{(u,v)}$ is updated for each link, which gives $\mathcal{O}(|E|)$ as the running time in the worst case. In addition, in line 19, PM-NSI executes a modified DSP algorithm (a.k.a dynamic version of well-known Dijkstra's shortest path algorithm) which uses equation 4 for path computation. AS indicated in [28], the time complexity of the Dijkstra's Algorithm is $\mathcal{O}(|E| + |E|log|N|)$, if priority queue is used as we have done for the implementation. Otherwise, it would be $\mathcal{O}(|N|^2)$ when array is used. Therefore, the computation time complexity of PM-NSI

will be $\mathcal{O}(|E|) + \mathcal{O}(|E|) + \mathcal{O}(|E| + |E|log|N|)$ which equals to $\mathcal{O}(3|E| + |E|log|N|)$. However, since 3 is a constant value, it is $\mathcal{O}(|E| + |E|log|N|)$.

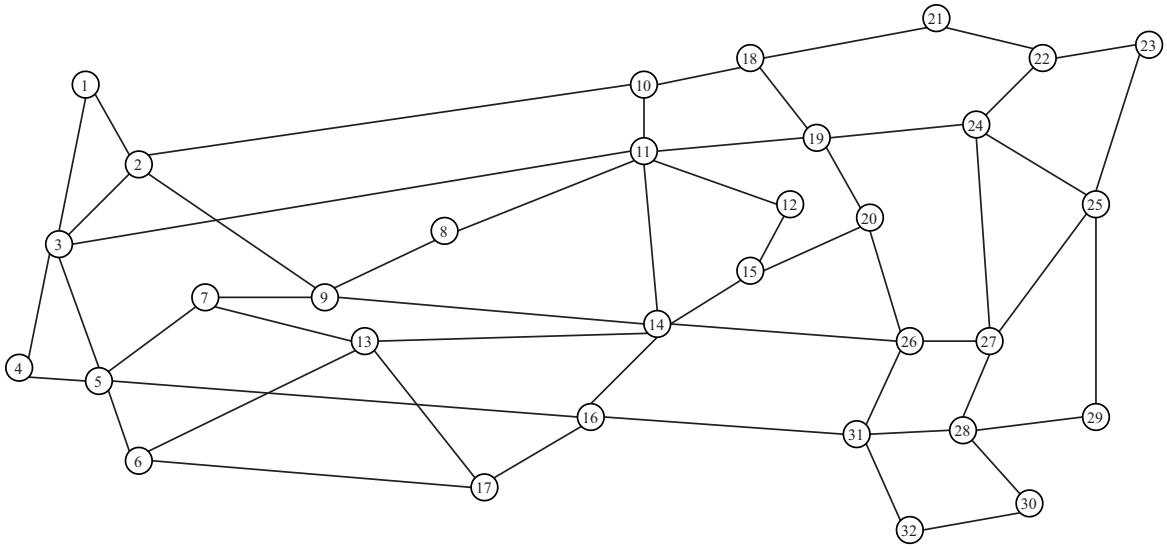

Figure 2. ANSNET topology.

4. Performance evaluation

A centralized RYU SD-WAN controller is used on a Mininet emulator to implement PM-NSI and the existing periodic NSI (PNSI) collection methods. RYU is one of the SDN controllers used for research and development purposes in academia and industry. It supports OpenFlow protocols and gives better performance than Floodlight in terms of packet loss, latency, and jitter [33]. In addition to PNSI, the DSP algorithm is chosen which provides one of the best performances among the routing algorithms evaluated in [4]. For PM-NSI, the aforementioned modified version of DSP that uses the cost metric, which is determined for PM-NSI, is implemented. If the controller computes the path for a flow, it then installs the rules on the Mininet switches on the path by using the OpenFlow protocol. Then, the flow loads the network as much as request r. The controller rejects the flow if it cannot find a feasible path for the flow.

Table . Simulation parameters of performance evaluation.

Parameter	Value
Number of links $\|E\|$	108
Number of nodes $\|N\|$	32
Bandwidth capacity $BW_{(u,v)}$	$uniform(2, 10)$ Mbps
Demand r of flow $f_{(s,d)}(r)$	$uniform(200, 500)$ kpbs
NSI collection period $\mathscr{T}$	$3, 5, 10, 20, 30, 50$ and 100 seconds
Average Rate α	3

For tests, in order to fairly compare both methods, the same values for link bandwidth, requested demands, and topology for the network as done in [4, 21] are used. Thereby, the UDP flows with randomly selected demands are generated and the performance measurements is observed on the hosts using Linux software

and iperf. Flows are randomly generated in every $uniform(1,5)$ seconds. Thus, we choose $\alpha = 3$, which is the average rate of the coming flows. For a flow $f_{(s,d)}(r)$, source s and destination d nodes (hosts) are selected randomly from any hosts and requested demand r from $uniform(200,500)$ kbps. For the network topology, the expanded ANSNET topology is used (Figure 2), which has 108 bidirectional links with 32 nodes [4]. We randomly selected the bandwidth capacity from $uniform(2,10)$ Mbps for each link. As different from [4, 21], all nodes have hosts that can generate and transfer flows between each other in our tests. The parameters used in simulation are summarized in Table .

The following performance measurement metrics have been chosen for tests:

Total traffic load on the controller (TTLC)

Cumulative length of NSI collection messages, which carries port (link) statistics between switches and the controller[1] [10].

Number of accepted flows (NAFs)

The number of routed flows that the routing algorithm is able to compute path.

Total transferred data (TTD)

The total transferred data carried through the network for the accepted flows.

Loss rate (LR)

The percentage of the lost packets of accepted flows.

Total jitter (TJ)

The total latency of packets of accepted flows when they are transferring through the network.

The proposed PM-NSI method is evaluated together with the existing periodic collection method, named PNSI. For both NSI collection methods, the controller communicates with the switches to collect port (link) statistics at every $\mathscr{T}$ seconds. Both methods are compared for every $\mathscr{T} = 3, 5, 10, 20, 30, 50$, and 100 s as in [4]. The goal is to show that PM-NSI gives better results with respect to loss rate LR and total jitter TJ at longer period time.

Collecting NSI at longer periods ($\mathscr{T}$) significantly reduces TTLC. As seen in Figure 3, TTLC is getting around $3.8, 2.30, 1.12, 0.59, 0.36, 0.23$ MB for $\mathscr{T} = 3, 5, 10, 20, 30, 50$, and 100 s, respectively.

However, if the controller performs the PNSI method, although increasing period time reduces the controller load, it causes the controller to acquire inaccurate link state information which results in worse QoS because of unfeasible path computation for the requested flows. Furthermore, decreasing the period time does not provide better performance since it causes performance degradation on the controller because of the traffic overload [4, 21]. Therefore, it is very important to provide a trade-off between NSI collection period and accuracy of obtained link state information. The proposed PM-NSI algorithm provides a way to do this trade-off.

Collecting NSI is needed to keep the controller up to date with the global view of the topology and dynamically changing link state information. Since the view of the topology is not changing frequently, it could be acceptable to collect NSI in longer period. However, having the up-to-date link state information is important in terms of packet loss [29]. For this reason, the controller needs to collect NSI in shorter periods to

[1]Flowgrammable (2021). OpenFlow [online]. Website http://flowgrammable.org/sdn/openflow/message-layer/ [accessed 23 April 2021]

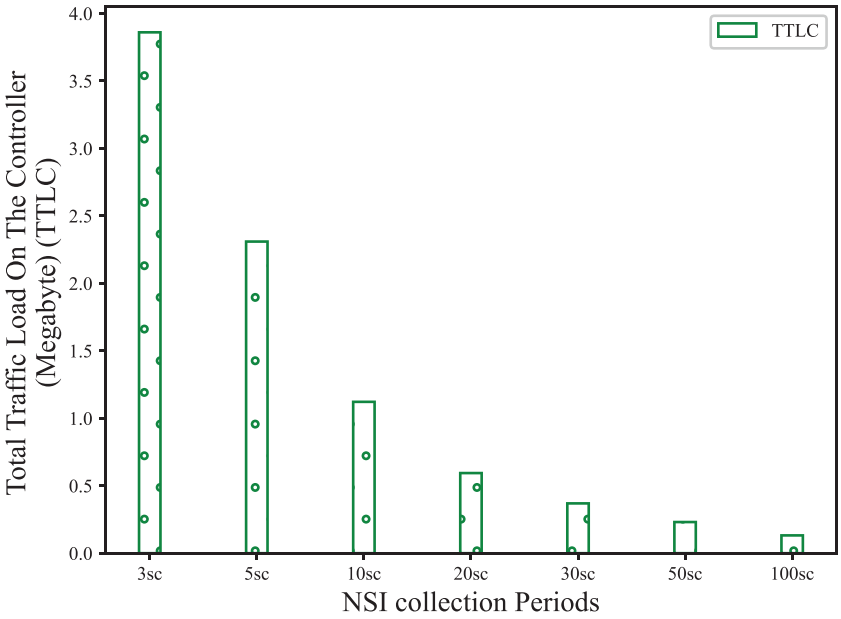

Figure 3. Total Traffic Load on the Controller during NSI collection under ANSNET topology in every $3, 5, 10, 20, 30, 50$ and 100 seconds.

obtain accurate and up-to-date link state information. However, collecting NSI in shorter periods can also cause overload on the controller, which also decreases the performance of the controller. Thus, it is crucial to pick an optimal period to balance the accuracy of the link state and the controller overload. Accordingly, as seen in Figure 4, PNSI provides the best loss rate when it collects NSI in every 5 s. The loss rate increases exponentially when the collection period increases, obviously because the controller does do not have accurate link-state information. One may observe that shorter period time to collect NSI does not give the best performance as seen for 3 s in Figure 4. The reason for that is collecting NSI in every 3 s increases TTLC, which negatively affects the responsiveness of the controller [4]. Besides, LR of PM-NSI is about 0.1% for $3 - 20$ s periods. Furthermore, PM-NSI provides a similar LR every 50 s compared to the best case of PNSI with 5 s. Thus, the proposed PM-NSI method reduces TTLC by around 1000% while providing a similar LR performance.

Jitter, which is the latency and response time in milliseconds, is also one of the important aspects of the network [31]. Especially, it is absolutely crucial for real-time latency-sensitive applications such as video conferencing, VoIP calls, online video gaming and so on [30]. If the jitter is high, it negatively affects the quality of communication that decreases transfer speeds [31]. As seen in Figure 5, PM-NSI gives better total jitter (TJ) for every collection periods. PM-NSI minimizes TJ at $\mathscr{T} = 20$ s collection period. However, TJ is not zero because of the switches-controller communication, background traffic, and path installation processes [32]. As seen in Figures 4 and 5, although PM-NSI gives much better results than PNSI, the performances of both decrease when the period increases. The reason is that the controller has old NSI which misleads it to use the same remaining bandwidth of the links and find the same paths for the different flows that have the same source-destination pairs, which results in congestion on the links. Although PM-NSI provides much better results when the collection period $\mathscr{T}$ is up to 100 s for LR and 20 s for TJ, its performance also decreases (still better than PNSI) when the collection period is higher than 50 s. To better explain this, we need to evaluate Figures 6 and 7.

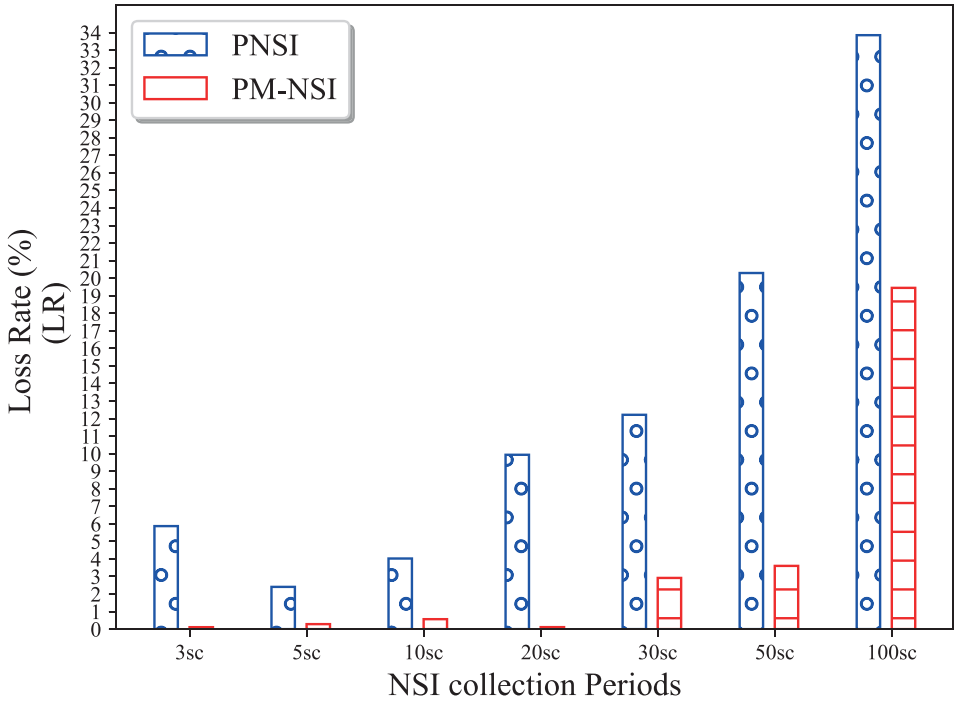

Figure 4. Loss rate (%) under ANSNET topology with PM-NSI and PNSI in every $3, 5, 10, 20, 30, 50$, and 100 s.

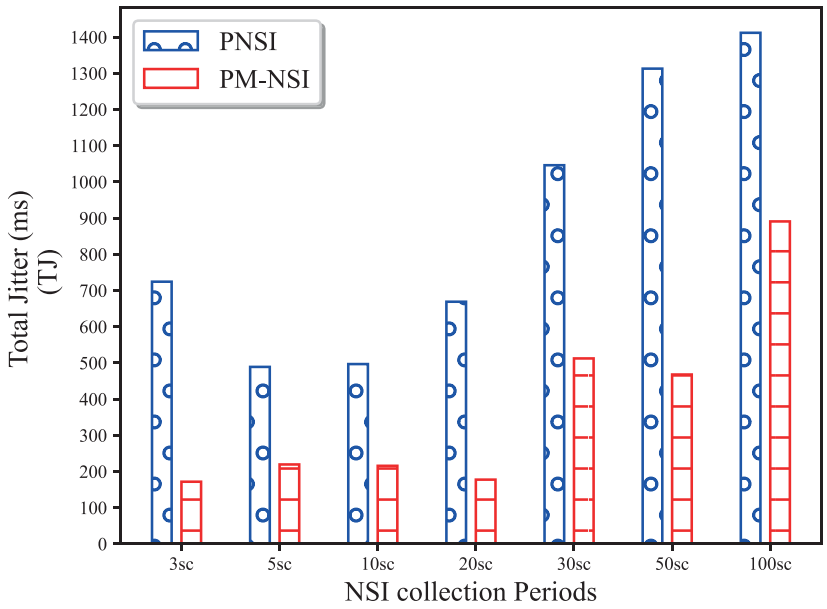

Figure 5. Total jitter (ms) under ANSNET topology with PM-NSI and PNSI in every $3, 5, 10, 20, 30, 50$, and 100 s.

As seen in Figure 6, PNSI accepts more flows than PM-NSI in every $\mathscr{T} = 3 - 100$ s. While PM-NSI accepts similar number of flows (around 63) for every collection period, the number of accepted flows NAF is increasing for PNSI when the collection period increases. However, accepting more flow does not guarantee that the method works better. As seen in Figure 7, total transferred data (TTD) decreases when the collection

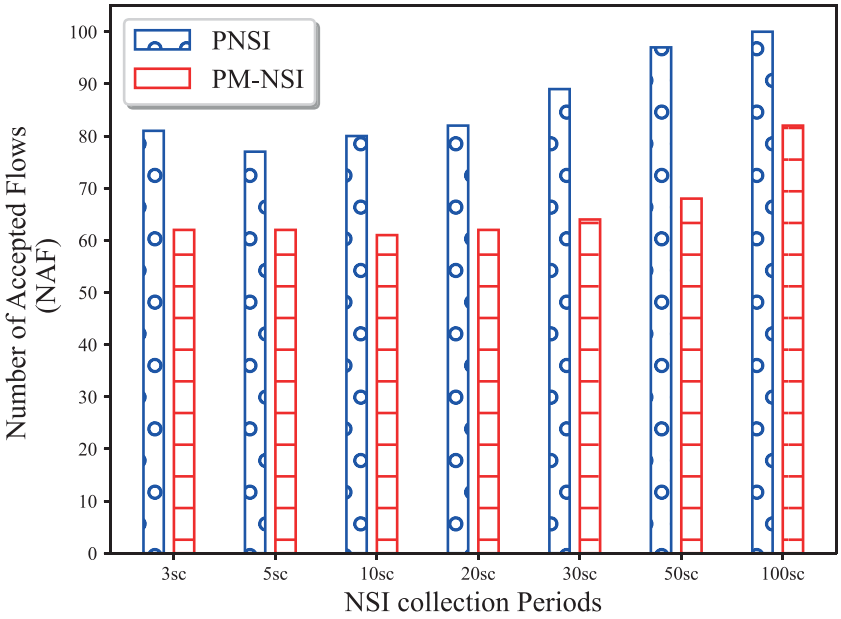

Figure 6. Number of accepted flows (NAF) under ANSNET topology with PM-NSI and PNSI in every $3, 5, 10, 20, 30, 50,$ and 100 s.

period increases, while it gives equivalent results for PM-NSI for every collection periods. Furthermore, when the collection period increases, the controller accepts more flow than the network can carry. Therefore, LR and TJ gets higher as seen in Figures 4 and 5. Thus, by predicting future possible load, PM-NSI only accepts the maximum number of flows that the network can handle. Although it seems that PM-NSI transfers less data through the network, due to lower LR and TJ of PM-NSI, it maintains quality of service (QoS) for the accepted flows.

5. Conclusion and future work

To obtain dynamically changing network state information (NSI) of a network is one of the key issues for routing in SD-WAN. Basically, a logically centralized SD-WAN controller periodically queries port (link) state information from underlying switches at predetermined time periods. However, the collected NSI is valid at the time the reply message is created by the switches. When the controller receives the message, the collected NSI on the message has already become out of date. In addition, collecting NSI with shorter periods will not be a solution for the issue, because it will cause protocol overload in the controller. Collecting at longer period will increase inaccuracy, although it decreases protocol overhead. Obviously, in both scenarios, existing periodic NSI method does not provide an effective mechanism to collect NSI. Accordingly, we proposed an effective prediction method for NSI that aims to predict NSI which allows to collect NSI at longer periods resulting in reducing controller load and providing more accurate NSI to compute more feasible paths for the flows. In our tests, we observed that PM-NSI provides more accurate NSI to the controller at longer period. Thus, the controller does not exceed the number of accepted flows that the network can carry. Therefore, it reduces loss rate and total jitter which results in better quality of service for the accepted flows. Seemingly, PM-NSI sacrifices the total transferred data. In essence, PNSI sacrifices quality of service because all accepted flows lose packets and encounter latency during transmitting through the network.

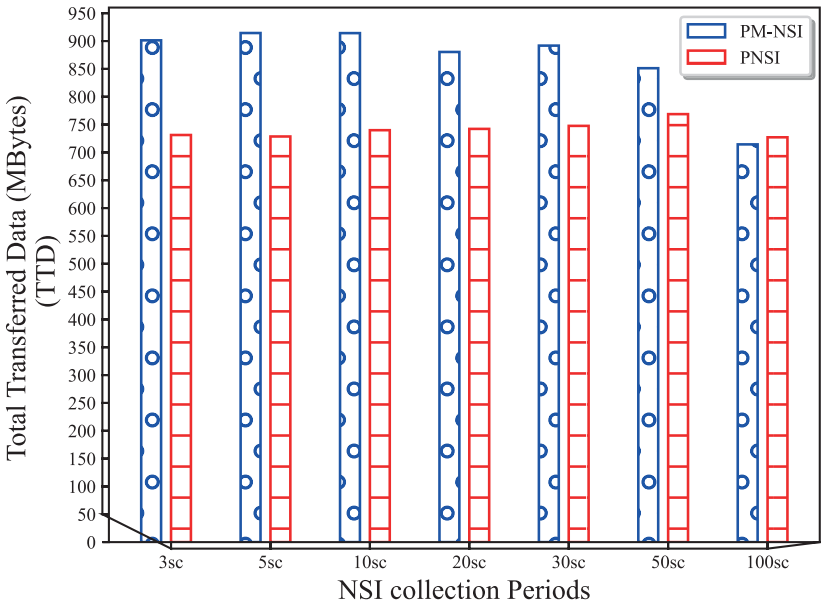

Figure 7. Total transferred data under ANSNET topology with PM-NSI and PNSI in every $3, 5, 10, 20, 30, 50$, and 100 s.

In the future, it is planned to modify the proposed PM-NSI for the SD-WAN-based network that has logically centralized multiple controllers. Furthermore, we intend to generalize PM-NSI with other topologies and also try to modify for overcoming with other scalability issues. In addition, we aim at enhancing one of the short term traffic prediction algorithms based on long short-term memory (LSTM) to predict NSI more accurately and reduce further controller load [23–26].

References

[1] Feamster N, Rexford J, Zegura E. The road to SDN: an intellectual history of programmable networks. ACM SIGCOMM Computer Communication Review. 2014; 44 (2):87-98. doi: 10.1145/2602204.2602219

[2] Kreutz D, Ramos FM, Verissimo PE, Rothenberg CE, Azodolmolky S et al. Software-defined networking: A comprehensive survey. Proceedings of the IEEE. 2014; 103 (1):14-76. doi: 10.1109/JPROC.2014.2371999

[3] Raghavan B, Casado M, Koponen T, Ratnasamy S, Ghodsi A et al. Software-defined internet architecture: decoupling architecture from infrastructure. In: 11th ACM Workshop on Hot Topics in Networks (HotNets-XI); Redmond, WA 2012. pp. 43-48.

[4] Akin E, Korkmaz T. Comparison of routing algorithms with static and dynamic link cost in software defined networking (sdn). IEEE Access 2019; 7: 148629-44. doi: 10.1109/ACCESS.2019.2946707

[5] Kim H, Feamster N. Improving network management with software defined networking. IEEE Communications Magazine. 2013; 51 (2):114-9. doi: 10.1109/MCOM.2013.6461195

[6] McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L et al. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM computer communication review. 2008;38 (2):69-74. doi: 10.1145/1355734.1355746

[7] Goransson P, Black C, Culver T. Software Defined Networks: A Comprehensive Approach. Morgan Kaufmann, 2016.

[8] Yang Z, Cui Y, Li B, Liu Y, Xu Y. Software-defined wide area network (SD-WAN): Architecture, advances and opportunities. In: 28th International Conference on Computer Communication and Networks (ICCCN); Valencia, Spain 2019. pp. 1-9. doi: 10.1109/ICCCN.2019.8847124

[9] Karakus M, Durresi A. A survey: Control plane scalability issues and approaches in software-defined networking (SDN). Computer Networks. 2017; 112:279-93. doi: 10.1016/j.comnet.2016.11.017

[10] Pfaff B, Lantz B, Heller B. Openflow switch specification, version 1.3. 0. Open Networking Foundation. 2012.

[11] Curtis AR, Mogul JC, Tourrilhes J, Yalagandula P, Sharma P et al. DevoFlow: Scaling flow management for high-performance networks. In: Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM); Toronto Ontario, Canada 2011. pp. 254-265.

[12] Yu M, Rexford J, Freedman MJ, Wang J. Scalable flow-based networking with DIFANE. ACM SIGCOMM Computer Communication Review. 2010; 40 (4): 351-62. doi: 10.1145/1851275.1851224

[13] Yu BY, Yang G, Yoo C. Comprehensive prediction models of control traffic for SDN controllers. In: 4th IEEE Conference on Network Softwarization and Workshops (NetSoft); Montral, Canada 2018. pp. 262-266.

[14] Bianco A, Giaccone P, Mashayekhi R, Ullio M, Vercellone V. Scalability of ONOS reactive forwarding applications in ISP networks. Computer Communications; 2017; 102:130-8. doi: 10.1016/j.comcom.2016.09.007

[15] Bianco A, Giaccone P, Mahmood A, Ullio M, Vercellone V. Evaluating the SDN control traffic in large ISP networks. In: IEEE International Conference on Communications (ICC); London, UK 2015. pp. 5248-5253.

[16] Mu M, Stokking H, Den Hartog F. Network delay and bandwidth estimation for cross-device synchronized media. Springer, Cham MediaSync. 2018; pp.649-676. doi: 10.1007/978-3-319-65840-7.

[17] Yu C, Liang Q, Lianghui D, Feng Y. Estimating Available Bandwidth Using Overloading Stream with Variable Packet Size. In: IEEE 3rd International Conference for Convergence in Technology (I2CT); Pune, India; 6 Apr 2018. pp. 1-6. doi:10.1109/I2CT.2018.8529381

[18] Abut F, Leischner M. An Experimental Evaluation of Tools for Estimating Bandwidth-Related Metrics. International Journal of Computer Network & Information Security. 2018;10 (7). doi:10.5815/ijcnis.2018.08.01

[19] Jasim AH, Ogren N, Minovski D, Andersson K. Packet probing study to assess sustainability in available bandwidth measurements: Case of high-speed cellular networks. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications. 2020;11 (2):106-25. doi: 10.22667/JOWUA.2020.06.30.106

[20] Al-Najjar A, Layeghy S, Portmann M, Indulska J. Enhancing quality of experience of voip traffic in SDN based end-hosts. In: IEEE 2018 28th International Telecommunication Networks and Applications Conference (ITNAC) 2018; pp. 1-8. doi: 10.1109/ATNAC.2018.8615286

[21] Akin E, Korkmaz T. Link-prioritized network state information collection in SDN. In: IEEE International Conference on Communications (ICC); Shanghai, China 2019. pp. 1-7.

[22] Karakus M, Durresi A. A scalability metric for control planes in software defined networks (sdns). In: IEEE 30th International Conference on Advanced Information Networking and Applications (AINA); Crans-Montana, Witzerland 2016. pp. 282-289.

[23] Tian Y, Zhang K, Li J, Lin X, Yang B. LSTM-based traffic flow prediction with missing data. Neurocomputing. 2018; 318:297-305. doi: 10.1016/j.neucom.2018.08.067

[24] Cao S, Liu W. Lstm network based traffic flow prediction for cellular networks. In: International Conference on Simulation Tools and Techniques (SIMUtools); Chengdu, China 2019. pp. 643-653.

[25] Li J, Gao L, Song W, Wei L, Shi Y. Short term traffic flow prediction based on LSTM. In: Ninth International Conference on Intelligent Control and Information Processing (ICICIP); Wanzhou, China 2018. pp. 251-255.

[26] Wei W, Wu H, Ma H. An autoencoder and LSTM-based traffic flow prediction method. Sensors. 2019; 19 (13) :2946. doi: 10.3390/s19132946.

[27] Göransson P, Black C, Culver T. The OpenFlow Specification. In: Göransson P, Black C, Culver T. Software Defined Networks: A Comprehensive Approach. 2nd ed. Morgan Kaufmann, 2017, pp. 89-136.

[28] Fredman ML, Tarjan RE. Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM (JACM). 1987;34 (3):596-615. doi: 10.1109/SFCS.1984.715934

[29] Alsaeedi M, Mohamad MM, Al-Roubaiey AA. Toward adaptive and scalable OpenFlow-SDN flow control: A survey. IEEE Access. 2019 Aug 1;7:107346-79. doi: 10.1109/ACCESS.2019.2932422

[30] Amiri M, Al Osman H, Shirmohammadi S,Abdallah M. An SDN Controller for Delay and Jitter Reduction in Cloud Gaming. In: Proceedings of the 23rd ACM International Conference on Multimedia (MM); Association for Computing Machinery, New York, NY, USA 2015; 1043–1046. doi:10.1145/2733373.280639

[31] Numan PE, Yusof KM, Marsono MN, Yusof SK, Fauzi MH et al. On the latency and jitter evaluation of software defined networks. Bulletin of Electrical Engineering and Informatics. 2019; 8 (4):1507-16. doi: 10.11591/eei.v8i4.1578

[32] Chahlaoui F, Dahmouni H. Towards QoS-enabled SDN networks. In: IEEE 2018 International Conference on Advanced Communication Technologies and Networking (CommNet); Marrakech, Morocco 2018; pp. 1-7. doi: 10.1109/COMMNET.2018.8360251

[33] Chouhan RK, Atulkar M, Nagwani NK. Performance Comparison of Ryu and Floodlight Controllers in Different SDN Topologies. In: 1st IEEE International Conference on Advanced Technologies in Intelligent Control, Environment, Computing & Communication Engineering (ICATIECE); Bangalore, India 2019, pp. 188-191. doi: 10.1109/ICATIECE45860.2019.9063806