# Adaptive Life-cycle Based on Traffic Prediction on ONOS Controller

Seungbeom Song, Jaiyong Lee
School of Electrical and Electronic Engineering
Yonsei University
e-mail: {glistar, jyl}@yonsei.ac.kr

***Abstract*— Smart device and Internet of Things (IoT) require high Quality of Service (QoS). A centralized network emerged as the most suitable alternative network and it is expected to be the leading future network. At this moment, the most popular centralized network is Software Defined Network (SDN) which can be separated into control plane and data plane in terms of software. SDN reduces complexity in distributed networking and manages network resources easily. Due to these advantages, SDN is undergoing a drastic increase in networking deployment. However, despite these merits, SDN still has problems with congestion. The congestion problem with inevitable performance decrease affects the QoS of the end users. In our study, we propose ALTP based on the Open Network Operating System (ONOS) controller to provide high QoS to users through adaptive monitoring and forwarding. For implementing the traffic estimating subsystem in SDN controller, we used Time Series Analysis (TSA). We got the meaningful benefit of performance while increasing overhead slightly by implementing the adaptive control of ALTP system.**

***Keywords- SDN; QoS; ONOS; Adaptive Life-cycle***

## I. INTRODUCTION

In the last few years, with the great increase in the smart device's distribution rate, various communication networks have been constructed and managed globally. Moreover, various kinds of services, leading real-time services, such as video streaming, are provided using these networks. About these communication services, users require high Quality of Service, which guarantees high throughput reliability. Network providers satisfy these demands and guarantee continuous and high throughput or control the network QoS parameters such as throughput, delay, jitter, bitrate and so on. But, in today's legacy network at the congestion situation, throughput decline is occurring rapidly. By reducing the window size through congestion control in each end host [1], it is controlled a little bit. However, it is impossible to quickly recognize the whole network state and respond to unstable situations. Because of these points, the legacy network structure has the limitation to guarantee consistent high rate throughput required by present users. To solve this limitation, the network infrastructure should handle the traffic in a more flexible way.

SDN developed by Berkeley and Stanford University is a relatively new paradigm. SDN proposed a solution of Open Shortest Path First (OSPF)'s limitation through centralized management hierarchy. At the same time, in contrast to traditional IP networks, it provides a separate data plane and control plane of the network [2]. In SDN, network control such as routing table is processed on the controller. It sends instructions to the data plane. It reduces duplicate and unnecessary calculation. It suggests complete control of the network at the controller. SDN infrastructures have a major advantage from the abundant availability of computing resources in the control plane layer which is typically hosted on high-performance commodity servers [6], reduce the complexity of distributed configuration and ease the network management tasks programmability [5]. Due to these advantages, the 5G network architecture is proposed based on SDN. However, the current controller lacks a system implementation that takes advantage of the benefits of the central control. It only consists of the existing method based on software. For example, topology-based methods such as cloud distributed routing on Quagga [6] are used instead of SDN-specific routing. Also, although it provides reactive forwarding, link connectivity only reacts based on periodic link level discovery protocols or link events. This is a non-reactive control that reflects network conditions.

In this paper, we focus on the Open Network Operating System Controller system which assures high quality using Adaptive Life-cycle of data plane based on Traffic Prediction (ALTP) in SDN environment.

The paper is organized as follows. Section II gives an overview of SDN and ONOS [3] controller. Section III introduces reasons for the need for adaptive life-cycle control in network congestion. Section IV highlights significant related work. Section V describes the proposed ALTP ONOS system. Section VI analyzes the results achieved with ALTP. We conclude and outline future work in Section VII.

## II. BACKGROUND

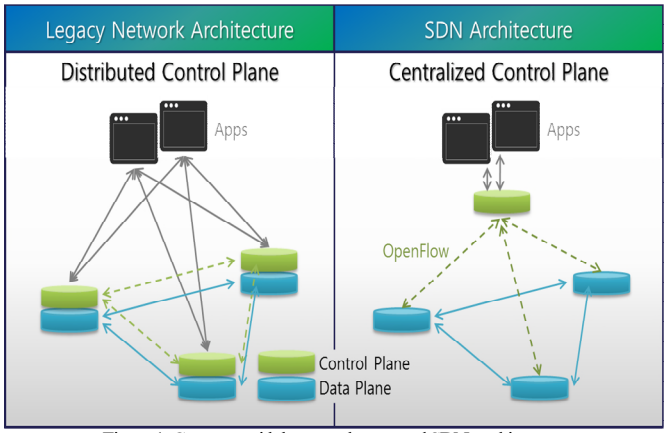


Figure 1. Compare with between legacy and SDN architecture.

 ISBN: 978-1-61208-625-5 

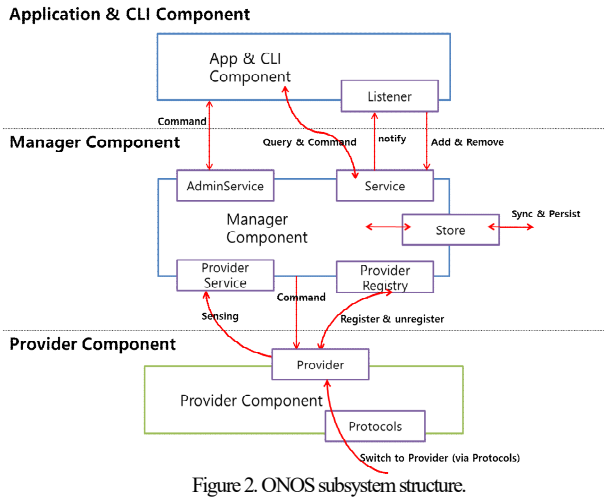


Figure 2. ONOS subsystem structure.

### A. SDN Overview

SDN [7], developed by Berkeley and Stanford University in 2008, is a new paradigm for controlling and managing networks [8]. Unlike other centralized networks such as OSPF, SDN has a unique concept, which is the separation of network control and data plane, as shown Figure 1. Network management functions based on software are centralized. SDN can help more scalable vendors independently. This means providers need not consider expensive vendor-specific devices and protocols when expanding the network, and easily recognize the whole network's state [7]. In the network configuration, these ease the network management tasks and control forwarding rules of the whole switch more efficiently [5]. For this reason, SDN lets network managers configure, manage, store, and optimize management parameters directly as basic operations [6]. Overall, SDN can become intelligent, responsive, and programmable [7].

### B. ONOS subsystem structure and OpenFlow specification

The ONOS subsystem of the control plane is composed of several elements shown in Figure 2. The provider component is the interface with data plane. The provider communicates with the data plane through south-bound protocols such as OpenFlow [7]. The manager component is the main body of the controller. It is composed of manager, service, store, and registry. The manager is the core, which operates to combine all components of manager body. The service interface has functions, which help to use methods to other components and through calling. It can receive another component's sensing or query data. Store saves methods which need to utilize and determine to synchronize or persist manager component's methods. The register interacts with the provider. The application component is network service function such as forwarding service, firewall and so on. By using the service of the manager component, applications can be more flexible. Any application has its own listener which helps receive control signaling and use the service parameter. In the overall control plane, the provider communicates with several switches, and the manager processes stats to command query and the application operates network functions.

## III. CONGESTION AND HIGH QUALITY-SERVICE IN SDN

The dissatisfaction with high quality service is the delay caused by link congestion. If congestion occurred in the network, the end host's window size is decreased through congestion control whereby the average delay is high in the traditional network. Furthermore, link congestion brings QoS degradation through the re-routing process handled by traffic engineering at the point of network management, which can bring a delay also. In this aspect, for high quality service, it is important to improve the re-routing process speed for the average delay when congestion occurred and to avoid congestion. In ONOS system, re-routing happens when the flow-table and topology are refreshed or when the switches links status is changed. Therefore, network status table, flow table, meter table and topology table should be refreshed to implement new QoS operations. In other words, updating the meter table fast and flow entry is very important for fast QoS operation. However, this frequently updating come from message communication in controller and switch in SDN environment. So, for more accurate and faster management, the controller must communicate with the switch frequently. This process triggers an increase in messages and leads to link congestion between controller and switch. Hence, the tradeoff between controller management messages and control plane resource usages will be considered.

To solve this problem, we propose the ONOS system using a method to predict the traffic variation for increasing the updating life-cycle more only on congestion switch candidates. But, though we recognize the traffic trend, it was hard to manipulate because traffic is varied too much irregularly. However, if we predict traffic a few later times in SDN environment, controlling congestion can be proactive in advanced. We suggest TSA [10] to solve this problem. TSA is one of the mathematical methods to find the trend of data flow. In this paper, we do not deal with TSA. If we find a proper trend and suggest a suitable model, forecasting future traffic models could be predicted. In other words, TSA is one of the proper methods for this study because it uses recent data and can predict instantly.

## IV. RELATED WORK

There are numerous research studies of predicting flow's fluctuation. Bozakov et al. [5] studied how to estimate the autocorrelation of network flow from monitoring data. To gather data, they use random sampling, i.e., random inter-query times. As a consequence, this trial could increase the quality of the whole network successfully without exceeding the control plane overhead issue. And there were develop a system for traffic matrix estimation using the sampling of OpenFlow counters [11]. But these studies just focused on reducing message overhead without specific network function of controller's query message. Fast recovery after a node or link failure is very important in any routing protocol. In the legacy network, the authors of [12] discussed node recovery efficiency using Lagrange multipliers and suggested 'pop-routing' which is applied in OSPF. But this approach runs only legacy environment, not a centralized network, and focused on making the recovery faster and

 ISBN: 978-1-61208-625-5 

reliable. Therefore, this new routing policy cannot help to reduce the link's own congestion phase. There was an approach of SDN's polling command and query, namely [13], where they used a polling scheme and changed dramatically based on real-time traffic in the whole network. But, the fast response of traffic change is a very important issue and real-time based scheme cannot fulfill in burst traffic due to the bottleneck effect. There was a suggestion of adaptive scheme in SDN environment. Authors of [14] studied the avoidance of congestion in SDN by re-routing when link utilization is more than 70%. Authors of [15] studied for control idle flow timeout using several time interval data. That operated like a Round Trip Time (RTT) in Transmission Control Protocol (TCP), but compared with RTT, they don't use weight function and just used raw arrival rate. Despite the fact that the study does not consider link bandwidth and congestion, the concept of adjusting interval, using special network function, is meaningful enough.

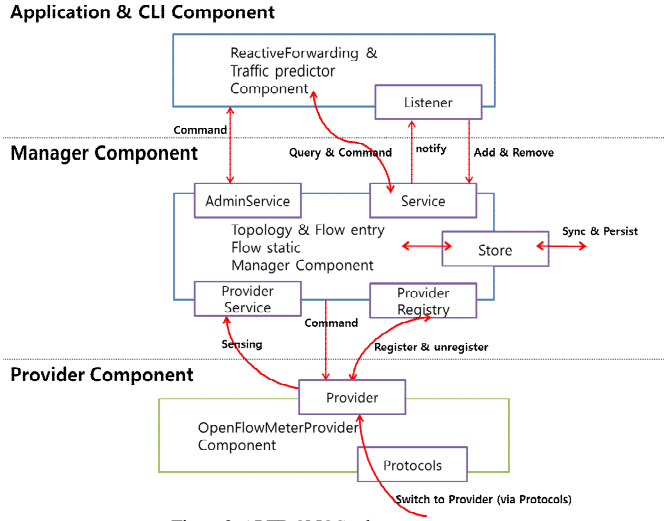


Figure 3. ALTP ONOS subsystem structure

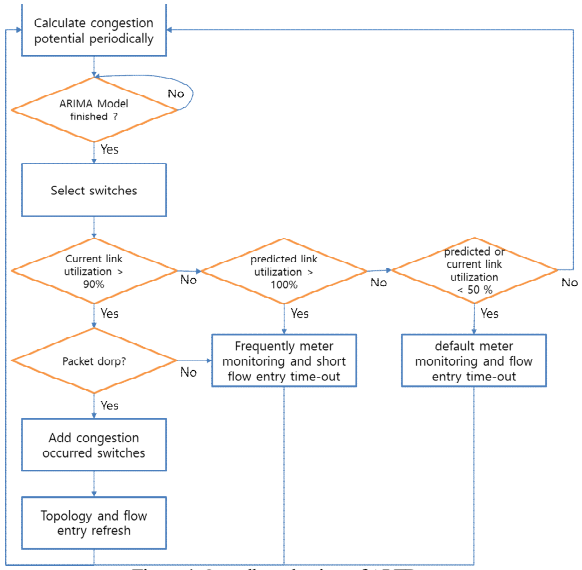


Figure 4. Overall mechanism of ALTP.

## V. IMPLEMENTATION

For improving the present SDN environment, we propose the new ONOS subsystem structure satisfying high quality service. This system's characteristic can be roughly categorized into two genres. First, we have the Traffic prediction using Time-Series Analysis, and second we have a more rapid response to congestion by the re-configuring life cycle of the network topology for Adaptive Life-cycle based on Traffic Prediction (ALTP) mechanism.

### *A. ALTP ONOS Subsystem Design Concept*

Rapid responding to congestion is essential for satisfying high QoS. In this paper, two measures are suggested for rapid responding to congestion. The first is the fast re-routing when congestion occurred, and second is applying QoS to each flow when link condition is varied by congestion, and consequently reducing delay as much as possible.

### *B. Design Requirement*

Two measures stated above, the accuracy of meter entry showing link status and flow entry in charge of routing are needed on SDN controller management. These can be executed by the frequent update of those two entries. However, the frequent update occurs inevitably message overhead between switch and controller. To improve this overhead problem in meter entry updating, updating must be executed selectively in the switches which have congested link or predicted link congestion. Also, flow entry time-out updating must be executed selectively when congestion occurred or high probability of congestion is expected.

As a result, the ALTP ONOS subsystem requires two functions, like below.

*1) The Function recognizing switches and links where a risk of congestion is high or congestion occurred.*

*2) The Function applying for adaptive meter entry and flow entry to selected switches and links, which makes message overhead become the least.*

### *C. ALTP ONOS Subsystem Sturcture*

We assume that no packet loss occurs at links and do not assume link down. Packet loss occurs only in case of congestion. Also, increasing message overhead between controller and switches only reduces controller's handling time by control plane congestion, but no control packet loss occurs between controller and switch. In this structure, shown in Figure 3, we modified the manager component and services to get the information about switches and links where congestion occurred or predicted the probability of congestion is high. In addition, to manipulate updating frequency of entry, the provider component has modified MeterStatCollector and the application component has modified ReactiveForwarding.

*1) Overall Mechanism of ALTP ONOS Subsystem*

The diagram which is shown in Figure 4 simply schematizes the mechanism proposed ALTP ONOS subsystem structure. First, for the flow path which the service requiring high service level agreement used, the ALTP ONOS subsystem calculates congestion potential using TSA for all switches using network status database updated by periodic polling messages. Congestion potential refers to the bandwidth utilization of the link interface mentioned in [14]. The utilization of the link interface is defined as currently used bit rate per ports divided by the maximum bit rate per ports. It selects the switch

wherein congestion occurred or congestion risk is high. Congestion occurred means link utilization is over 90% or packet drop has occurred. In this case, topology graph except that interface is requested immediately. In the case of a single path for that service, diverting other flow using the same link is derived by setting the weight of the link high. The flow entry is immediately refreshed. High congestion risk means that predicted link utilization exceeds 100% by TSA in the path meter polling. In this case, it sets the meter pilling interval and flow entry timeout to half. This is to compensate for imperfect predicted value. The system reacts more quickly in congestion situations by using more control messages. The setting parameters are changed to default after the predicted utilization of link is 50% or less in a sequence of the system. It is also applied when current link utilization is 50% or less. Then, adjust flow entry updating speed according to the flow that passes pertinent switches. Similarly, by adjusting meter entry updating speed for pertinent switches, responding to the congestion can be faster.

*2) Flowchart of interactions between Subsystems*

Figure 5 presents the flowchart showing how subsystems provide the adaptive updating rate of flow entry and meter entry to switches by using exchanged data between subsystems. Table I represents which data is exchanged between subsystems and what is this data's meaning. DeviceService provides port state information of all existing ports of topology to TrafficPredictor. Next, TrafficPredictor expects the potential of congestion phase of each port by using TSA and transfers the list of ports which has expected to be congested to LinkService in the mechanism of ALTP ONOS Subsystems.

LinkService returns the list of links which is connected to received port list information and is already congested. This means that LinkService returns the list of links which is expected to be congested or is already congested and transfers this list to ReactiveForwarding and DeviceService. ReactiveForwarding adjusts the life-cycle of flow entry. DeviceService, by using the information of the list of links, returns the list of devices which has the congestion-expected link or has congested link and transfer it to MeterStatsCollector. MeterStatsCollector, by using this information, adjust the updating rate of meter entry and monitoring rule.

For ALTP ONOS subsystem we proposed, we added and modified five classes in ONOS system, as follows.

Traffic predictor (Added): Traffic predictor is newly added device service function, which is involved in Manager Component. The autoregressive integrated moving average (ARIMA) of TSA models [16] is used and predicts the traffic transition. Traffic predictor collects all the amount of present traffic-bandwidth usage calculated in [14] from each port of each switch in current topology. According to this collected data, it predicts if the congestion will occur or not in that port. At this moment, for accurate prediction of traffic bandwidth usage amount, at least past 50 usage amount data is needed. So, the predicted value will not be returned before accumulate 50 data and only return present traffic bandwidth. Using predicted value, traffic predictor returns the list of ports that have a high probability of congestion, and it transmits the info to the LinkService and DeviceService.

LinkService (Modified): In modified LinkService, the transmitted port list which has high congestion probability from the Traffic Predictor is used. If any link is connected to the congested port interface, it decides predicted congestion link or has congestion now for all links in topology now. We get the list of links that have congestion or has a high probability of congestion.

DeviceService (Modified): Modified DeviceService recognizes devices expected congestion for all links. We can get the list of switches expected to be congested and already congested.

ReactiveForwarding (Modified): In modified ReactiveForwarding, when the congestion occurrs, ONOS controller processes the re-routing by using this class. When we want to make this re-routing time faster, we should update the flow entry, which sets the path of each flow faster. So, by getting the links which are the member of flow`s path, SDN controller decides the potential of congestion of links of that path using the proposed system. This also means deciding the possibility of re-routing. If the path is expected to be re-routed, it should increase the updating rate of flow entry for reacting faster to this situation. This can be executed by reducing flow timeout of flow entry.

MeterStatsCollector (Modified): In modified MeterStatCollector, when the congestion and re-routing occurrs, the new path is set and the link state that the flow uses as the path is changed. Therefore, it is necessary to provide changed QoS to each flow by updating the meter entry faster. Therefore, by increasing the updating rate of meter entry for switches that are expected to be congested or are already congested, we should provide changed QoS as fast as possible. By doing this, ALTP ONOS controller can react to congestion phase faster.

TABLE I. THE MESSAGE DESCRIPTION OF ALTP

| | Data | Description |
|---|---|---|
| 1 | Port statistics | For every period, compute whole port statistics |
| 2 | Port list | Attain and transfer port list which expected to be congested through port statistics |
| 3 | Link list | Transfer link list which expected to be congested through the port list |
| 4 | Device list | Transfer device list which expected to be congested through the port list |
| 5 | Monitoring rule | Command adjusted meter entry polling interval to selected switch |
| 6 | Forwarding rule | Command adjusted flow timeout to every flow which passes the selected switch |

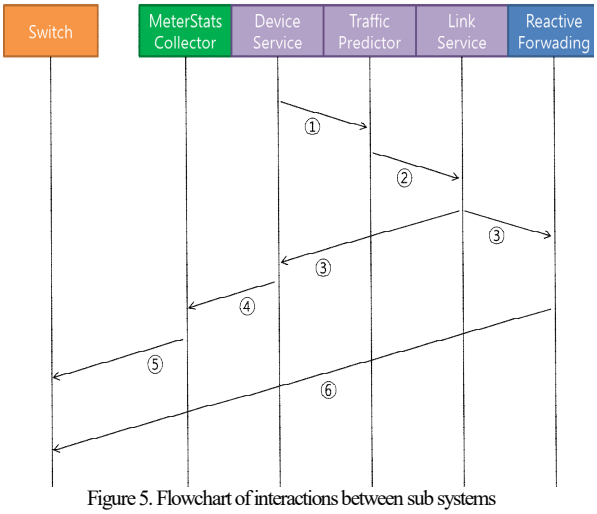


Figure 5. Flowchart of interactions between sub systems

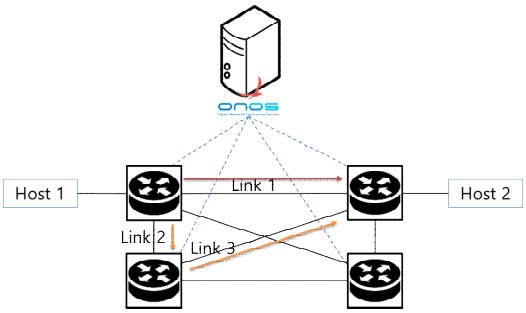


Figure 6. Topology setting for simulation

## VI. RESULTS AND DISCUSSION

Here we describe the analysis of the result of the experiment showing how much ONOS controller's performance is improved when ALTP ONOS controller we suggested is used. We use the ALTP ONOS controller based on ONOS version 1.5.2 with OpenFlow version 1.3. To verify the fast reaction of ALTP ONOS controller for congestion, we set the simple topology as experiment environment shown in Figure 6.

We generate the traffic from host 1 to host 2 in Poisson's distribution which has average 2Mbps velocity while link1 has 1Mbps bandwidth. This is for deliberate congestion occurrence. Link2 and link3 on the re-routing path have 10Mbps bandwidth, respectively. By setting like this, when the routing path is changed, which means ALTP changes the life-cycle time interval of flow entry and meter entry form 10 seconds (default life-cycle time interval) to 5 seconds when congestion is expected, we can observe how much delay is improved by faster updating meter entry. Because in case of meter entry that is applied by prior routing path, it cannot guarantee 2Mbps to each flow. However, after the meter entry update, it can guarantee 2Mbps to each flow, so the delay would be considerably improved. In other words, ALTP ONOS controller can provide High-QoS by reacting faster in case of congestion occurrence.

In Table II, by increasing the Life-cycle interval of selected switches' entries, two important factors that have a dominant effect on high QoS are considerably improved. However, this result is reasonable because it is ached by increasing message overhead between controller and switches.

However, when we compare the results of the cases which increasing updating rate is applied to selected switches or not, we can find that the performance we can achieve by increasing adaptive message is much bigger. Table II indicates the performance and message overhead in three cases. The first case is applying the default life-cycle rate in the traditional ONOS controller. The second case is applying the faster life-cycle rate to selected switches in ALTP ONOS controller. The last one is applying the faster updating rate to all switches in the traditional ONOS controller.

We achieve better performance of updating entries more frequently for all switches (not selective). However, the messages are increased too, which lead to an overhead of control plane. This indicates that ALTP ONOS controller has better performance compared to the traditional ONOS controller by increasing message overhead as small as possible.

TABLE II. RESULT DATA WITH ADPTIVE LIFE-CYCLE RATE

| (Life-cycle interval) updating rate | Default (10) Life-cycle (non-selective) | ALTP (5or10) Life-cycle (adaptive) | Double (5) Life-cycle (non-selective) |
|---|---|---|---|
| The average number of message in control plane | 3766 | 4114 | 4775 |
| Message Increase rate | - | 9.2% | 26.7% |
| Average Flow Delay(s) | 2.5356 | 2.1011 | 2.03976 |
| Average Packet drop rate | 10.24% | 8.03% | 7.76% |
| Average Re-routing Convergence time(ms) | 51.598 | 43.969 | 42.844 |

## VII. CONCLUSION AND FUTURE WORK

In this paper, to provide high QoS in IoT services, we design ALTP ONOS controller which predicts or recognizes congestion phase and executes a fast response to congestion phase. The method for congestion phase estimation is performed by TSA. Moreover, the method for fast response to congestion phase is updating flow and meter entry more frequently. In the experiment, it is verified that traffic prediction by using ARIMA model in TSA is sufficiently reliable. By using these methods, the components such as traffic delay, throughput, and drop rate which have a dominant effect on high QoS are improved. This result indicates that the suggested methods perform fast response in congestion phase. Also, adjusting entries process is performed only on selected devices. So, we can minimize message overhead increment between controller and switch due to frequent entry update.

The proposed ALTP ONOS controller induced performance improvement successfully. However, in case of ARIMA model used in ALTP, it is strong for trend analysis but weak for burst analysis. So, to perform burst analysis more accurately, an additional prediction model would be beneficial for further performance improvement. Long-term analysis can be one of the alternatives, and it is possible to predict burst potential by accumulating data for various time periods. Therefore, adjustment of flexible monitoring rule is possible. Deep-learning can be helpful for trend analysis by accumulating data.

In brief, the suggested method in this paper is capable of managing congestion with minimized message overhead.

## ACKNOWLEDGMENT

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2018-2017-0-01633) supervised by the IITP (Institute for Information & communications Technology Promotion)

## REFERENCES

[1] Network working group, "TCP Congestion Control", Purdue University, 2009.

[2] A. l-Najjar, S. Layeghy, M. Portmann, "Pushing SDN to the End-Host, Network Load Balancing using OpenFlow", IEEE 13th International Workshop on Managing Ubiquitous Communications and Services, pp. 1-6, 2016.

[3] U. Krishnaswamy et al., ''ONOS: An open source distributed SDN OS,'' 2013. [Online]. Available: http://www.slideshare.net/ umeshkrishnaswamy/open-networkoperating-system

[4] Z. Bozakov, A. Rizk, D. Bhat and M. Zink, "Measurement-based Flow Characterization in Centrally Controlled Networks", IEEE INFOCOM 2016, pp. 1-9, 2016.

[5] H. Kim and N. Feamster, "Improving network management with software defined networking", IEEE Communications Magazine, pp.114–119, Feb. 2013.

[6] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador and M. F. Magalhães, "Quagflow: partnering quagga with openflow", In ACM SIGCOMM Computer Communication Review ,Vol. 40, No. 4, pp. 441-442, 2010.

[7] Software-Defined Networking (SDN) Definition, ONF (Open Networking Foundation), Available: https://www.opennetworking.org/sdn-resources/sdn-definition/

[8] A. l-Najjar, S. Layeghy and M. Portmann, "Pushing SDN to the End-Host, Network Load Balancing using OpenFlow", IEEE 13th International Workshop on Managing Ubiquitous Communications and Services, pp. 1-6, 2016.

[9] Software Defined Networking, TechCentral, Available: http://www.techcentral.ie/software-defined-networking/

[10] Hamilton, J. D. (1994). Time series analysis (Vol. 2). Princeton: Princeton University Press.

[11] A. Tootoonchian, M. Ghobadi, Y. Ganjali, "OpenTM: Traffic matrix estimator for openflow networks", University of Toronto, p.201–210, 2010.

[12] L. Maccari, and R. L. Cigno, "Messages for Faster Route Convergence Pop-Routing: Centrality-based Tuning of Control", IEEE INFOCOM 2016, p.694, 2016.

[13] Z. Su, T. Wang, Y. Xia and M. Hamdi, "FlowCover: Low-cost Flow Monitoring Scheme in Software Defined Networks", In IEEE Global Communications Conference 2014 (GLOBECOM 2014), pp. 1956-1961, 2014

[14] S. Song, J. Lee, K. Son, H. Jung and J. Lee, "A congestion avoidance algorithm in SDN environment", IEEE 30st Intnternational Conference on in Information Networking (ICOIN 2016), pp. 420-423, 2016.

[15] L. Xie, Z. Zhao, Y. Zhou, G. Wang, Q. Ying and H. Zhang, "An Adaptive Scheme for Data Forwarding in Software Defined Network", IEEE. 2014 Sixth International Conference on Wireless Communications and Signal Processing (WCSP), pp. 1-5.

[16] S. Basu, A. Mukherjee and S. Klivansky, "Time series models for internet traffic", In IEEE INFOCOM'96, pp. 611-620, 1996.


## APPENDIX

***Traffic predictor (Added)***

```
Public boolean TrafficPredictor(port, currentspeed, maxspeed)
   Make list of fifty current speed samples. If the list is not set, return
currentspeed;
   Expected speed = getArima(current speed list);
   If (expected speed > max speed)
      Return true; Return false; }
Public List<Port> getCongestionExpectedPort(){
   For(about all existing ports in topology);
      Get current speed, maxspeed, port number from each port statistics using
DeviceService's getPortstatistics;
      If(TrafficPredictor(port,currentspeed,maxspeed) == true)
         Return the list of ports;}
```

***LinkService (Modified)***

```
Public Iterable<Link> getInActiveLinks(){
   Return  list of links its state is inactive; }
Public boolean RecognizeLinkisCongested(Link link){
   If (link is connected to getCongestionExpectedPort()'s port list)
      Return true;
   Else{ Return false; }
Public Iterable<Link> getCongestionExpectedLinks(){
   for(all existing links){
         if(RecognizeLinkisCongested(link)==true);
           return link list which is connected to congested port;}
Public Iterable<Link> getAllCongestionLinks(){
   Return list of links which is already congested + lists of links which is
expected to be congested;}
```

***DeviceService (Modified)***

```
Public Iterable<Device> getAllCongestionDevices(){
   Return  list of devices which has congested link + list of devices expected
to be congested;}
Public Iterable<Device> getCongestionDevices(){
   For(all existing devices)
     If( RecognizeDeviceCongestion(deviceId)==true){
      Return list of devices;}
Public Iterable<Device> getCongestionExpectedDevices(){
      For(all existing devices)
     If (RecognizeDeviceisExpectedCongestion (deviceId)==true){
      Return list of devices;}
Public boolean RecognizeDeviceisExpectedCongestion(DeviceId deviceId){
   For(all links of this device)
     If ( RecognizeLinkisCongested(links)==true)
      Return true; Break; return false; }
Public boolean RecognizeDeviceCongestion(DeviceId deviceId){
      For(all links of this device)
     If ( link state is inactive
      Return true;
      Break; return false; }
```

***ReactiveForwarding (Modified)***

```
Flow Time out = DEFAULT_TIMEOUT;
phase or already has congestion
For(all links which are members of selected path){
  If( link is in getAllCongestionLinks){
   FlowTimeout = DEFAULT_TIMEOUT / 2;}}
InstallRule;
```

***MeterStatsCollector (Modified)***

```
Public void run(Timeout timeout){
   adaptvieInterval = DEFAULT;
if(this device is in  getAllCongestionDevices){
    adaptiveInterval = DEFAULT / 2;}
timeout.getTimer().newTimeout(adaptiveInterval); }
```