View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Institute of Advanced Engineering and Science

International Journal of Electrical and Computer Engineering (IJECE)
Vol. 5, No. 2, April 2015, pp. 289~296
ISSN: 2088-8708

The Extended Dijkstra's-based Load Balancing for OpenFlow Network

Widhi Yahya[1], Achmad Basuki[2], Jehn-Ruey Jiang[3]
[1,2]Department of Electrical Engineering, University of Brawijaya, Malang, Indonesia
[1,3]Department of Computer Science and Information Engineering, National Central University, Taoyuan City, Taiwan

Article Info

Article history:

Received Dec 26, 2014
Revised Feb 9, 2015
Accepted Feb 20, 2015

Keyword:

Dijkstra's algorithm
Load-balancing
Shortest path
Software defined networking

ABSTRACT

This paper proposes load-balancing algorithm on the basis of the Extended Dijkstra's shortest path algorithm for Software Defined Networking (SDN). The Extended Dijkstra's algorithm considers not only the edge weights, but also the node weights to find the nearest server for a requesting client. The proposed algorithm also considers the link load in order to avoid congestion. We use Pyretic to implement the proposed algorithm and compare it with related ones under the Abilene network topology with the Mininet emulation tool. As shown by the comparisons, the proposed algorithm outperforms the others in term of the network end-to-end latency, and throughput.

Copyright © 2015 Institute of Advanced Engineering and Science.
All rights reserved.

Corresponding Author:

Widhi Yahya,
Department of Electrical Engineering,
University of Brawijaya,
Jl. Veteran, Malang 65145,East Java, Indonesia.
Email: widhi.yahya.ub@gmail.com

1. INTRODUCTION

The emergence of the SDN technology brings many new network applications realized by programming the SDN controller. Typical examples include load-balancing, multimedia multicast, intrusion detection, and so on. Load-balancing is an important concept in networking. The purpose of the load-balancing application is to distribute the loads into multiple servers in order to get the best performance [1]. Online services, such as e-commerce, e-government, web sites, and social networks, often use multiple servers to achieve reliability and high availability. Those service systems can use a load balancer in the front-end to map requests from clients to servers. Using hardware load balancers can be a solution, but it suffers from the expensive cost, and the rigid policy imposed by the vendor. The SDN technology enables users to design their own flexible and cost-efficient load balancer that is suitable for their system.

Some load-balancing method using SDN technology are recently proposed. LABERIO [1] and LOBUS [2] consider the path and link utilization as a method to optimize the system throughput. The other method is the service-based load-balancing algorithm [3]. Service-based means consider the flow that related to specific services. The purpose of the service-based load-balancing algorithm is specified resources (switches, links, and servers) with particular services to increase reliability and availability of the system. Both methods consider the path as the important thing in order to reach best performance, but maintaining the path must be clearly defined. In this paper, we proposed load-balancing algorithm that is extended by shortest path algorithm as a method to choose the best path.

In 2014, J.R. Jiang, et.al, extends the well-known Dijkstra's shortest path algorithm [4] to consider not only the edge weights, but also the node weights for a graph derived from the underlying SDN topology. As shown by the simulation results in [4], the extended Dijkstra's algorithm outperforms the Dijkstra's algorithm and the non-weighted Dijkstra's algorithm under the Abilene network [5] in terms of end-to-end

latency. This is because the extended Dijkstra's algorithm takes edge weights as transmission delays over edges and takes node weights as process delays over nodes, while the other two algorithms consider only edge weights or no weights.

In this paper, we propose a load-balancing algorithm for servers that are located and spread in wide area SDN networks. In the SDN, the controller has a global view of the network topology. This information can be processed to get the nearest servers by using a shortest path algorithm. In the proposed algorithm, we can find the nearest servers from the clients by using the extended Dijkstra's algorithm that can adapt to the topology changes. We also consider the link load for congestion control. We use Pyretic [6] to implement the proposed algorithm and compare it with the round-robin, the randomized load-balancing algorithm and the LABERIO, under the Abilene network [5] topology with the Mininet [7] emulation tool. As shown by the comparisons, the proposed algorithms outperform the others in term of the network end-to-end latency, and the throughput.

The remainder of this paper is organized as follows. In Section 2, we introduce some preliminary knowledge, including the SDN concept, load-balancing in SDN, and the extended Dijkstra's algorithm. Section 3 describes the load-balancing concept and the proposed algorithms. Section 4 shows the simulation results and observations. Finally, this paper is concluded with Section 5.

2. PRELIMINARIES

2.1 SDN Load-Balancing

SDN advocates the separation of control and data planes (or layers), where underlying switching hardware devices (called *switches*) are controlled via software entities (called *applications*) that runs on external, decoupled automated control plane devices (called *controllers*) [8]. OpenFlow is one of the first open protocols defined between the control plane device, the controller, and the dataplane device, the *switch*, of the SDN architecture [8]. An OpenFlow switch consists of one or more *flow tables* and/or *group tables*, as shown in Figure 2. An OpenFlow controller can update, add and delete flow entries in flow table both reactively and proactively. Each flow table in the switch contains a set of flow entries, each of which consists of *match fields, counters,* and *set of instructions*, as shown in Figure 3.

The SDN enables to design our own protocol on top of SDN switches. It is abolishing rigidity of nowadays network. The intelligent of network administrator or researcher can easily put on the network devices such as load-balancing methods.N. Handigol, et.al, [2] proposed the Plug-n-Serve system implementing a load-balancing algorithm, called LOBUS (LOad-Balancing over UnStructure networks), using OpenFlow for unstructured networks. LOBUS maintains the network topology and link status, and greedily chooses the client-server pair that yields the lowest total response time for each newly arriving request. M. Koerner and O. Kao [3] developed a load-balancing algorithm for handling multiple services (called LBMS) by the SDN technology. It uses the FlowVisor, an SDN device to achieve network virtualization, to coordinate multiple controllers, each of which handles requestsdestined for different services. In 2013, H. Long, et.al, [1] proposed a load-balancing algorithm, named LABERIO (LoAd-BalancEd Routing wIth OpenFlow), to minimize latency and response time and to maximize the network throughput by better utilizing available resources.

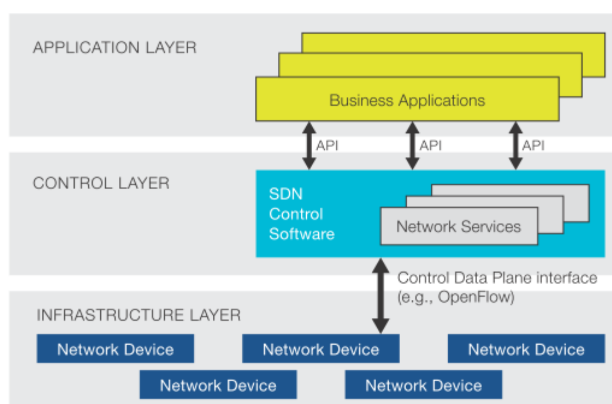

Figure 1. The illustration of the SDN architecture [8]

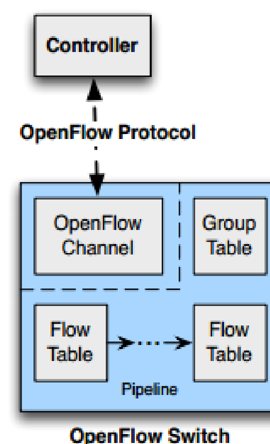

Figure 2. The Open Flow controller and the switch [9]

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Figure 3. The flow table entry of the OpenFlow switch [10]

The algorithm uses ToR (Top of Rack) Switch-to-ToR Switch Paths Table (S2SPT) and Load Allocation Table (LAT). However, maintaining S2SPT becomes a problem for the LABERIO, if the network topology changes. So, the LABERIO is not suitable in wide area network network that has dynamic topology.

2.2 The Extended Dijkstra's Algorithm

Given a weighted, directed graph $G=(V, E)$ and a single source node s, the classical Dijkstra's algorithm can return a shortest path from the source node s to every other node, where V is the set of nodes and E is the set of edges, each of which is associated with a non-negative *weight* (or *length*) [11]. In the original Dijkstra's algorithm, nodes are associated with no weight. The paper [4] shows how to extend the original algorithm to consider both the edge weights and the node weights.

Figure 4 shows the extended Dijkstra's algorithm, whose input is a given graph $G=(V, E)$, the edge weight setting *ew*, the node weight setting *nw*, and the single source node s. The extended algorithm uses $d[u]$ to store the *distance* of the current shortest path from the source node s to the destination node u, and uses $p[u]$ to store the *previous* node preceding u on the current shortest path. Initially, $d[s]=0$, $d[u]=\infty$ for $u\in V$, $u\neq s$, and $p[u]$=null for $u\in V$.

Extended Dijkstra's Algorithm

Input: $G=(V, E)$, *ew*, *nw*, *s*
Output: $d[|V|]$, $p[|V|]$

1: $d[s]\leftarrow 0$; $d[u]\leftarrow\infty$, for each $u\neq s$, $u\in V$
2: **insert** u with key $d[u]$ into the priority queue Q, for each $u\in V$
3: **while** $(Q\neq\emptyset)$
4: $u\leftarrow$Extract-Min(Q)
5: **for** each v adjacent to u
6: **if** $d[v]>d[u]+ew[u,v]+nw[u]$ **then**
7: $d[v]\leftarrow d[u]+ew[u,v]+nw[u]$
8: $p[v]\leftarrow d[u]$

Figure 4. The extended Dijkstra's algorithm [4]

Note that the extended Dijkstra's algorithm is similar to the original Dijkstra's algorithm. The difference is that the extended version adds the node weight in line 6 and line 7 of the algorithm. The original Dijkstra's algorithm cannot achieve the same result just by adding node weights into edge weights. This is because the node weight should be considered only at the outgoing edge of an intermediate node on the path. Adding node weights into edge weights implies that an extra node weight of the destination node is added into the total weight of every shortest path, making the algorithm return the wrong result.

The extended Dijkstra's algorithm is very useful in deriving the best routing path to send a packet from a specific source node to another node (i.e., the destination node) for the SDN environment in which significant latency occurs when the packet goes through intermediate nodes and edges (or links). Below, we show how to define the edge weights and node weights so that the extended Dijkstra's algorithm can be applied to derive routing path for some specific SDN environment.

Assume that we can derive from the SDN topology a graph $G=(V, E)$, which is weighted, directed, and connected. For a node $v\in V$ and an edge $e\in E$, let *Flow*(v) and *Flow*(e) denote the set of all the flows passing through v and e, respectively, let *Capability*(v) be the *capability* of v (i.e., the number of bits that v can process per second), and let *Bandwidth*(e) be the bandwidth of e (i.e., the number of bits that e can transmit per second). The node weight $nw[v]$ of v is defined according to Eq. (1), and the edge weight $ew[e]$ of e is defined according to Eq. (2).

$$nw[v] = \frac{\sum_{f \in Flow(v)} Bits(f)}{Capability(v)} \quad (1)$$

where Bits(f) stands for the number of f's bits processed by node v per second.

$$ew[e] = \frac{\sum_{f \in Flow(e)} Bits(f)}{Bandwidth(e)} \quad (2)$$

where Bits(f) stands for the number of f's bits passing through edge e per second.

Note that we can easily obtain the number of a flow's bits processed by a node or passing through an edge with the help of the "counters field" of the OpenFlow switches' flow tables. Also note that the numerators in Eq. (1) and Eq. (2) are of the unit of "bits", and the denominators are of the unit of "bits per second". Therefore, the node weight nw[v] and the edge weight ew[e] are of the unit of "seconds". When we accumulate all the node weights and all the edge weights along a path, we can obtain the end-to-end latency from one end to the other end of the path.

3. PROPOSED LOAD-BALANCING ALGORITHM

The load balancer is placed in the front-end of the online services systems to map the request from the clients to the servers. In the Linux virtual server (LVS) use Network Address Translation(NAT) to direct traffic from the Internet to a variable number of servers on the second layer, which in turn provide the necessary services. Service requests arriving at the LVS cluster are addressed to a virtual IP address or VIP. VIP is can be a public IP address that associates with a fully-qualified domain name and which is assigned to multiple servers [12].

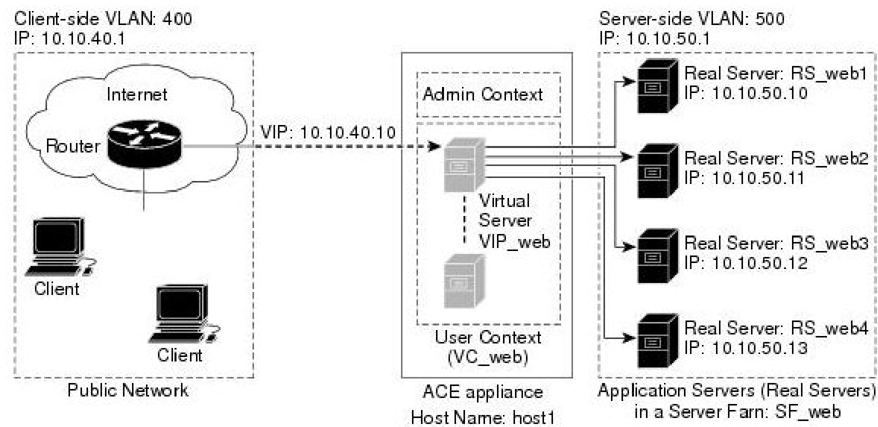

Figure 5. Example Server Load-balancing Setup [13]

Figure 5 explains the server load-balancing architecture by using VIP. In this paper, we use VIP address as IP public that will be accessed from the clients. Figure 5 is an example of a single point load balancer architecture. In this paper, the load-balancing algorithm was implemented on the Abilene topology that every switches in the topology act as a load balancer.

Using naive algorithms, such as the round robin algorithm and the randomized algorithm, in wide-area-network load-balancing has the possibility to forward a request to the farthest server. This is not efficient because the request and the replied data will go across the network and consume a lot of bandwidth of the whole network. In SDN, the controller has the global information of the whole network, and can decide to forward the request to the nearest server by finding the shortest path with the extended Dijkstra's algorithm from the client to a server, where the shortest path means the path with the smallest summation of node weights and edge weights.

Figure 6 shows the proposed load-balancing algorithm. The basic idea is to forward each request to the nearest server with the *link load* (utilization of the link between the server and the switch) lower than a pre-specified threshold θ. However, if all the servers have link loads larger than the threshold, the algorithm still chooses the nearest server. In this way, we can prevent congestion on the servers.

Proposed Load-Balancing Algorithm
Input:sw_{src},S_{dst}, θ
Output: s, $s \in S_{dst}$
$P \leftarrow eDijkstra(sw_{src}, S_{dst})$
forevery $p_i \in P$
if$p_i.server.kl > \theta$**then** *move* p_i *from P to Q*
if$P = \varnothing$**then**
  $s \leftarrow min(P).server$
else
  $s \leftarrow min(Q).server$
return$s$

Figure 6. Proposed Load-balancing Algorithm

We assume server s_i is attached to switch sw_i and a switch is attached with at most one server. Later on, we use s_i and sw_i exchangabely for convenience. Given the source switch sw_{src} to which the request client is attached, the set S_{dst} of servers, and a prespecified threshold θ, the proposed algorithm will return the best server for load-balancing.

The link load $kl_i$of serve s_i (the utilization of the link $<s_i, sw_i>$ between the server s_i and the switch sw_i) is defined as follows:

$$kl_i = \frac{current\ traffic\ of\ link < s_i, sw_i >}{maximum\ bandwidth\ of\ < s_i, sw_i >} \quad (3)$$

Since the proposed algorithm is based on the extended Dijkstra's algorithm [4], we also take the same mechanisms to obtain the node weights and the edge weights. Note that $eDijkstra(sw_{src}, S_{dst})$ will use the extended Dijkstra's algorithm to return a set P of shortest paths from the source switch sw_{src}, to every server in the server set the S_{dst},. Also note that $p_i.server$ stands for the server associated with the path p_i, and hence $p_i.server.kl$ stands for the link load of the server associated with the path p_i. Furthermore, the function $min(P)$ will return the shortest one among all shortest paths in P.

4. SIMULATION

4.1. Simulation Setting

According to the Abilene core topology, we set up an OpenFlow controller and 11 Open Flow switches as nodes, each of which was linked to the controller logically, as shown in Figure 7. For load-balancing testing, we assumed there are two web servers that placed and spread in two different locations in the Abilene network that will be accessed from clients.

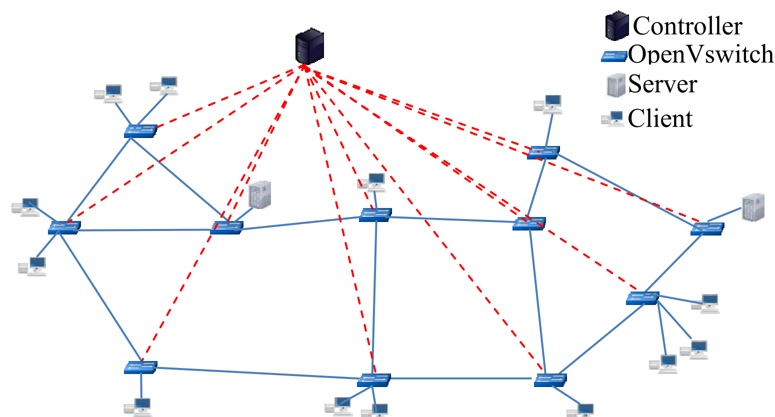

Figure 7. Topology Used in Simulation

The POX was used as OpenFlow controller. The load-balancing algorithm was implemented using Pyretic. The VIP address was used as IP public that will be accessed from the Client.

Table 1. Simulation and Settings

Parameter	Setting
Bandwidth on edges	1Gbps
Number of server	2
Number of switches	11
Number of edges	25
Controller	POX 2.0 supporting Pyretic
Openflow Switch	OpenVSwitch 1.0
Testing tool	Iperf, Netperf
Testing time	30second

In the Figure 7, there are 2 servers and 12 clients in the Abilene topology. In this simulation testing, we generated Transmission Control Protocol (TCP) data stream from the clients to the servers using Iperf. To extend more information we also used the Netperf to generate request from clients. For every testing, we defined the number of clients are 4, 8, and 12 clients. This simulation ran on AMD Phenom(tm) 9650 Quad-Core Processor and 8GB of RAM. To prove the algorithm's reliability, we compared it with the round robin and randomized algorithms which have no consider the shortest path.

4.2. Simulation Result

In this experiment the request can happen in every switch in the topology that describe in the simulation settings. The proposed algorithm chooses the best path (nearest server) for a requesting client by using the extended Dijkstra's algorithm. It is more convenient than maintaining the S2SPT and the LAT like in the LABERIO. The LABERIO was designed for a data center that used term of Top-of-Rack switch to maintain the paths. Since the different topology, in this experiment we use all switch to maintain the path for the LABERIO. As shown in the Figure 8, the proposed algorithm outperforms the LABERIO in the term of latency. It is quite similar because both algorithms consider the link capacity, but the superiority of the proposed algorithm isalsoconsider the node capacity. We also compare the proposed algorithm with naive algorithm, such as round-robin and randomized algorithm that do not consider the nearest server. The simulation results show that the proposed algorithm is better than the two naive algorithms in the term of end-to-end latency, as shown in Figure 8. The network end-to-end latency was measured by using the ping tool to send 30 packets whose packet size is 65507 bytes from clients to the servers for 30 seconds. The superiority is because the proposed algorithm considers the shortest paths (nearest server) and also the congestion control. On the contrary, it is possible for the naive algorithms to forward requests to the far server to go through some switches. In the real network, a high performance IP routers and switches add approximately 200 microseconds of latency to the link due to packet processing. It means, if the request are deflected to the farthest server to go through a lot of switches, the latency will increase significantly.

The throughput measurement and comparison was conducted to evaluate the capability of the proposed algorithm. Throughput is the rate of successful message delivery over a communication channel. In Figure 9, shows the proposed algorithm has higher throughput than the round robin,the randomized algorithm, and the LABERIO. The round robin and randomized algorithm may deflect request to the far server and even the link is congested. It causes the throughput is lower than the proposed algorithm.Consider the node capacity derive the proposed has better throughput then the LABERIO.

As shown in the Figure 10, we use standard deviation to measure the server load variation. Standard deviation measures the amount of variation or dispersion from the average. The load information is provided by the Iperf server program. In this experiment, the proposed algorithm shows good result (smallest) in the server's load variation.

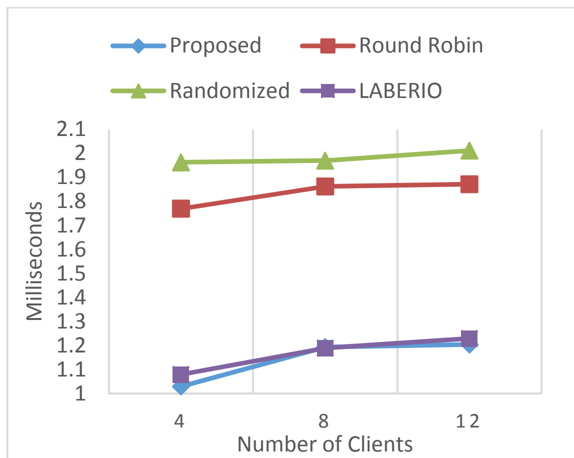

Figure 8. End-to-End Latency

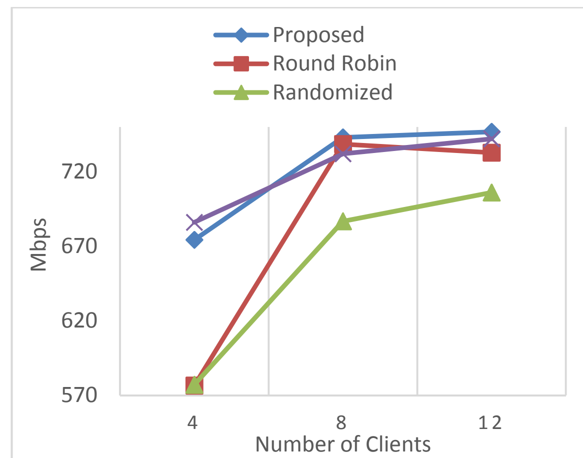

Figure 9. Throughput

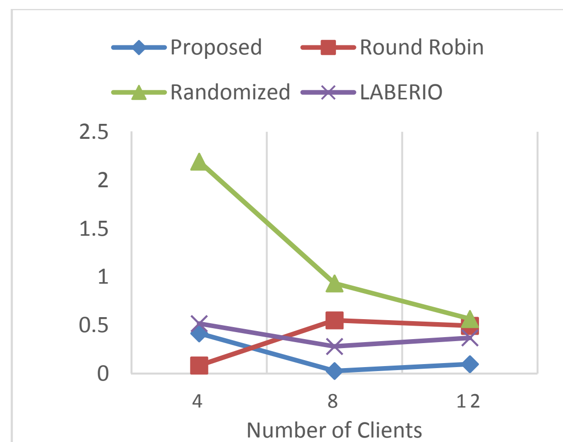

Figure 10. Standard Deviation Load of Servers

5. CONCLUSION

This paper proposes a load-balancing algorithm that takes advantage of the extended Dijkstra's shortest path algorithm. The extended Dijkstra's shortest path algorithm was used to find the nearest server for a requesting client. The extended Dijkstra's algorithmconsiders not only the edge weights but also the node weights for a graph derived from the underlying SDN topology. We use Pyretic to implement the proposed load-balancing algorithm under the Abilene network topology with the Mininet emulation tool. The simulation results show that the proposed load-balancing algorithm outperforms others in terms of the network end-to-end latency, and throughput.

REFERENCES

[1] H. Long, Y. Shen*, M. Guo, and F. Tang. "LABERIO: Dynamic load-balanced routing in OpenFlow-enabled networks". *IEEE 27th International Conference on Advanced Information Networking and Applications*. 2013.

[2] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari. "Plug-n-Serve: Load-balancing web traffic using Open Flow". *Demo at ACM SIGCOMM*, Aug. 2009.

[3] M. Koerner and O. Kao. "Multiple Service Load-Balancing with Open Flow". *IEEE 13th International Conference on High Performance Switching and Routing*. 2012.

[4] J.R. Jiang, H.W. Huang, J.H. Liao, and S.Y. Chen. "*Extending Dijkstra's Shortest Path Algorithm for Software Defined Networking*". in Proc. of the 16th APNOMS. 2014.

[5] Abilene Network, http://en.wikipedia.org/wiki/Abilene_Network-#cite_note-line-1, last accessed on March 4, 2014.

[6] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN Programming with Pyretic". *Technical Reprot of USENIX*, available athttp://*www.usenix.org*, 2013.

[7] Mininet Website, http://mininet.org/, last accessed on May 2014.
[8] Open Network Foundation (ONF) Website (SDN white paper), https://www.opennetworking.org/sdn-resources/sdn-definition, last accessed on January 2014.
[9] Open Networking Foundation. “OpenFlow Switch Specification version 1.4.0”. October 14, 2013.
[10] N. McKeown, et. al. “Open Flow: Enabling Innovation in Campus Networks”. *ACM SIGCOMM Computer Communication*. 2008.
[11] E. Dijkstra, “A note on two problems in connexion with graphs”. *Numerische mathematik*. vol. 1, no.1, 1959, pp. 269-271.
[12] Red Hat, Inc. “Linux Virtual Server Administration: RHEL5: Linux Virtual Server (LVS)”. 2007.Doc
[13] Wiki Cisco, http://docwiki.cisco.com/wiki/Cisco_ACE_4700_Series_Appliance_Quick_Start_Guide,_Release_A3(1.0)_--_Configuring_Server_Load_Balancing, last accessed on june 2014.

BIOGRAPHIES OF AUTHORS

Widhi Yahya received the Master Science degree in Department of Computer Science and Information Engineering, National Central University, Taiwan in 2014 as an International Dual Degree Master student between University of Brawijaya and National Central University. He got Bachelor degree in Department of Informatics Engineering, University of Brawijaya, Indonesia. His research interest area is in the computer science and information technology areas, especially in network programming.

Achmad Basuki is a senior lecturer in Department of Electrical Engineering, University of Brawijaya, Indonesia. He is expert in computer networking research area, especially in IP multicast,P2P, CDN, data center, and networking. He got Ph. D. degree from KEIO University, Japan. He presently work in PTIK, University of Brawijaya, Indonesia as a Chairman of the PTIK UB.

Jehn-Ruey Jiang received his Ph. D. degree in Computer Science in 1995 from National Tsing-Hua University, Taiwan, R.O.C. He joined Chung-Yuan Christian University as an Associate Professor in 1995. He joined Hsuan-Chuang University in 1998 and became a full Professor in 2004. He is currently with the Department of Computer Science and Information Engineering, National Central University, and co-leads the Adaptive Computing and Networking (ACN) Laboratory, which aims at developing adaptive mechanisms for collaborative computing entities to make proper adjustments according to their current understandings about the computing environments or resources, in order to efficiently perform given tasks.