Regular paper | Journal of the KIECS. pp. 1087-1094, vol. 18, no. 6, Dec. 31. 2023, t. 122, pISSN 1975-8170 | eISSN 2288-2189
http://dx.doi.org/10.13067/JKIECS.2023.18.6.1087

SDN 환경에서 서버 상태를 고려한 단계적 가중치 기반의 부하 분산 기법 연구

이재영* · 권태욱**

Study of Load Balancing Technique Based on Step-By-Step Weight Considering Server Status in SDN Environment

Jae-Young Lee* · Tae-Wook Kwon**

요 약

빅데이터, 클라우드, IoT, AI 등 기술의 발전으로 인해 높은 데이터 처리율이 요구되고 있으며 네트워크의 유연성과 확장성에 대한 중요성이 증가하고 있다. 하지만 기존 네트워크 체계는 벤더와 장비에 종속되어 앞선 요구를 충족하기에는 한계가 존재한다. 이에 소프트웨어 중심의 유연한 네트워크를 구성할 수 있는 SDN 기술이 주목받고 있으며 특히 SDN을 기반의 부하 분산 방식은 방대한 트래픽을 효율적으로 처리하여 네트워크 성능을 최적화할 수 있다. 기존 SDN 환경에서 부하 분산 연구들은 서버와 컨트롤러 간 불필요한 트래픽이 발생하거나 서버가 임계치에 도달한 후에야 부하 분산이 이루어지는 제한사항이 존재한다. 본 논문에서는 이를 해결하기 위해 서버 부하에 따라 단계적으로 서버에 가중치를 부여하는 방식을 통해 불필요한 트래픽을 최소화하고 서버가 과부화 되기 전에 적절한 부하 분산이 이루어질 수 있는 방식을 제안한다.

ABSTRACT

Due to the development of technologies, such as big data, cloud, IoT, and AI, The high data throughput is required, and the importance of network flexibility and scalability is increasing. However, existing network systems are dependent on vendors and equipment, and thus have limitations in meeting the foregoing needs. Accordingly, SDN technology that can configure a software-centered flexible network is attracting attention. In particular, a load balancing method based on SDN can efficiently process massive traffic and optimize network performance. In the existing load balancing studies in SDN environment have limitation in that unnecessary traffic occurs between servers and controllers or performing load balancing only after the server reaches an overload state. In order to solve this problem, this paper proposes a method that minimizes unnecessary traffic and appropriate load balancing can be performed before the server becomes overloaded through a method of assigning weights to servers in stages according to server load.

키워드

Software Defined Network, Load Balancing, Weight
소프트웨어 정의 네트워크, 부하 분산, 가중치

* 국방대학교 관리대학원(2j0115@naver.com)
** 교신저자 : 국방대학교 컴퓨터공학과
· 접 수 일 : 2023. 09. 18
· 수정완료일 : 2023. 10. 30
· 게재확정일 : 2023. 12. 27
· Received : Sep. 18, 2023, Revised : Oct. 30, 2023, Accepted : Dec. 27, 2023
· Corresponding Author : Tae-Wook Kwon
Dept. Computer engineering, Korea National Defense University
Email : 2j0115@naver.com

Ⅰ. 서 론

빅데이터, 사물인터넷(IoT, Internet of Things), 인공지능 (AI, Artificial Intelligence) 등 제 4차 산업혁명 기술이 다양한 분야에서 적용되면서 엄청난 양의 데이터가 생성되고 있다. 세계적인 통계조사 전문기관인 Statista에 의하면, 향후 2025년까지 글로벌 데이터 양이 약 181제타바이트까지 증가할 것으로 예상이 된다[1]. 2017년에 생성된 글로벌 데이터양이 26제타바이트였으므로 2025년에는 2017년 대비 약 600% 증가가 예상되는 것이다.

디지털 데이터의 급격한 증가와 그 활용 가치가 증대되면서 디지털 데이터를 저장하고 관리하는 데이터센터의 역할이 더욱 중요해지고 있다. 급변하는 환경에 대응하기 위해서 데이터센터는 유연성과 확장성을 갖춰진 네트워크 환경이 필요하다. 하지만 기존 네트워크 체계는 벤더와 하드웨어에 종속되어 유연성과 확장성 측면에서 한계가 존재하였다. 이에 네트워크 동작과 기능을 소프트웨어 기반으로 구성할 수 있는 SDN 기술이 등장하였다[2].

SDN 기술은 소프트웨어 기반의 중앙집중적 네트워크를 구현함으로써 다양한 사용자 요구사항에 유연하게 대응할 수 있으며 네트워크 확장에도 유리하다[3]. 이러한 특징 때문에 SDN 기반의 부하 분산 정책은 방대한 데이터를 관리하고 다양한 정책이 적용되는 데이터센터에서 효율적일 것이라 기대된다[4].

앞선 SDN 기반의 부하 분산 기법 연구들은 효율적인 부하 분산을 위해 다양한 방식을 제안했으나 서버와 컨트롤러 간 불필요한 트래픽이 발생하거나, 서버가 임계치에 도달하고 나서야 부하 분산 정책이 적용되어 적절한 부하 분산이 이루어지지 못하였다.

이에 본 논문에서는 서버와 컨트롤러 간 불필요한 트래픽을 최소화하기 위해서 서버는 정해진 부하 단계에 도달한 경우에만 컨트롤러에게 상태를 보고한다. 또한 컨트롤러는 부하 단계에 따라 단계적으로 서버에 가중치를 부여함으로써 서버가 과부화 되기 전에 적절한 부하 분산이 이루어지도록 하는 방식을 제안한다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 SDN의 개념과 SDN 환경에서 서버 상태 기반의 부하 분산 연구에 대해 논하고 Ⅲ장에서는 본 논문에서 제안하는 단계적 가중치 기반의 부하 분산 기법을 소개한다. 그 다음 Ⅳ장에서 제안한 기법의 실험 결과를 분석하고 마지막으로 Ⅴ장에서 결론으로 마무리한다.

Ⅱ. 관련 연구

2.1 SDN

SDN[5]의 핵심 특징은 제어와 포워딩의 분리다. 이에 따라 SDN은 전송계층(Data Plane), 제어계층(Control Plane), 응용 계층(Application Layer)으로 구성되는 삼계층 구조로 표현되며 이러한 계층 구조를 시각적으로 나타내면 그림 1과 같다.

Control Plane은 하나의 중앙집중적인 컨트롤러로 구현되며 망 동작을 제어하는 네트워크 정책이 적용되는 영역이고 Application Plane은 사용자가 원하는 정책을 적용하도록 지원하는 영역이다. Data Plane은 네트워크 정책에 따라 패킷을 포워딩하는 전송 장치의 역할만 수행한다.

전송 장치에 대한 컨트롤러의 제어는 OpenFlow[6] 같은 개방형 사우스바운드 API(Southbound API)에 의해 이루어지고, 서비스에 필요한 다양한 응용의 개발을 지원하기 위한 노스바운드 API(Northbound API)가 적용된다.

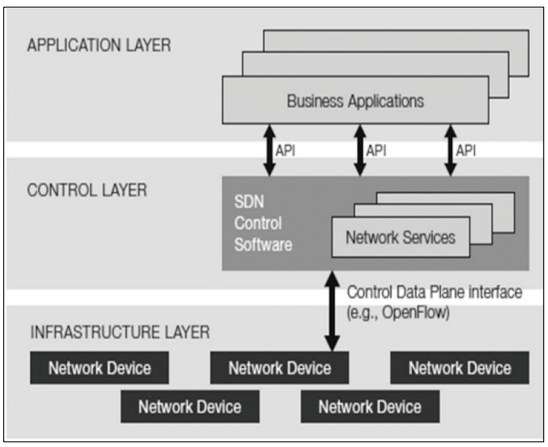

그림 1. SDN 계층 구조
Fig. 1 Layer structure of SDN

2.2 SDN 환경에서 부하 분산 연구동향

J. Kim[7]의 연구에서는 시스템에 유입되는 패킷형

태와 특정 이슈에 영향을 받아 편중된 데이터를 고려하기 위해 미들박스를 활용하여 데이터 유형을 분류하고 그 중 폭증하는 데이터를 응답시간이 빠른 서버를 통해 우선 처리하는 부하 분산 개념을 제시했다.

P. Deepalakshmi[8]의 연구에서는 네트워크에서 부하 분산을 판단하는 매개변수로 서버의 CPU, Memory, 연결 가능 세션 수를 지정하고, 매개변수별 중요도에 따라 가중치를 부여하여 서버의 부하를 판단하였다. 컨트롤러가 서버의 부하 상태를 주기적으로 모니터링하고 서버의 부하를 3가지 단계로 나누어 그에 따라 부하 분산하는 기법을 제안하였다.

J. Lee[9]의 연구에서는 서버의 부하 측정을 위해 발생하는 트래픽 교환의 최소화하는 것을 목표로 하였다. 이를 위해 컨트롤러는 서버의 부하 상태를 주기적으로 확인하지 않고 서버의 특정 임계치 도달 시에만 서버가 컨트롤러로 패킷을 전송하도록 설정하였다. 컨트롤러는 서버로부터 수신된 정보를 확인하여 등록된 서버의 상태정보를 수정하고 부하가 임계치에 도달한 서버를 제외한 다음 정상 서버를 대상으로만 요청을 할당하는 방식을 제안했다.

K. Lee[10]의 연구에서는 임계치에 도달한 서버를 포워딩 목록에서 제외하여 가용 능력이 낭비되는 것을 개선하고자 하였다. 네트워크 동작 중 특정 서버의 부하가 특정 임계치를 초과하거나 내려가는 경우 전체 서버의 CPU와 Memory 사용률을 확인하고 이를 기준으로 서버마다 가중치를 부여하여 클라이언트 요청을 할당하는 기법을 제안하였다.

기존 연구들은 서버의 부하 측정을 위해 주기적으로 트래픽을 주고 받으면서 서버와 컨트롤러 간 불필요한 트래픽이 발생하였다. 또한 임계치에 도달한 서버를 포워딩에서 제외함으로써 자원의 낭비가 발생하거나 서버가 과부하 상태가 되고 나서야 부하 분산이 이루어지는 개선점이 존재한다.

III. 제안 사항

3.1 제안하는 기법의 동작 방법

제안기법은 서버와 컨트롤러 간 서버의 부하측정을 위해 발생하는 트래픽을 최소화하고 서버가 과부화 상태가 되기 전에 단계적으로 부하 분산을 하여 서버가 과부화 상태가 되는 횟수를 줄이는 것을 목표로 한다.

서버와 컨트롤러는 서버가 설정된 두 단계의 임계값을 각각 초과하거나 내려가는 경우에만 패킷을 주고받으며 서버 부하 관련 이벤트가 발생하기 이전까지는 라운드로빈 방식을 적용한다. 서버 부하 관련 이벤트가 발생한 이후부터는 컨트롤러는 서버 부하 관련 패킷 접수하면 전체 서버를 대상으로 부하 상태를 확인하고 단계적으로 서버에 가중치를 부여하는 방식을 통해 사용자의 요청사항을 할당한다. 제안기법의 시스템 설계와 알고리즘은 각각 그림 2, 그림 3와 같다.

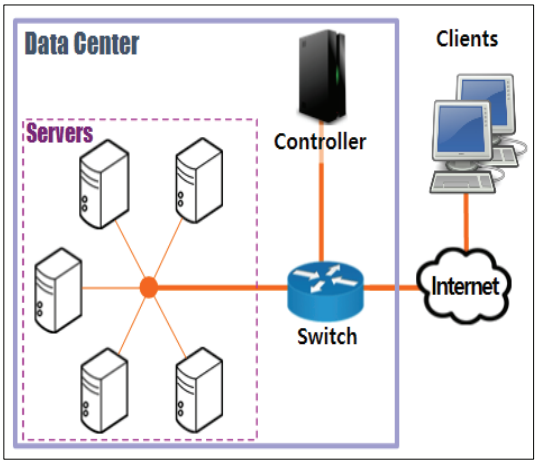

그림 2. 시스템 설계
Fig. 2 System design

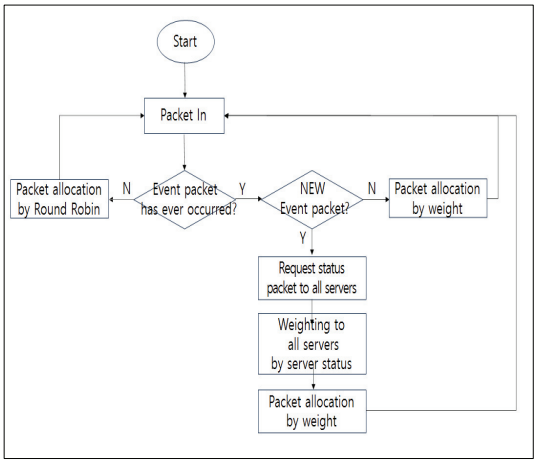

그림 3. 제안하는 기법 알고리즘
Fig. 3 Algorithm of proposed protocol

3.2 Controller의 동작

시스템이 시작되면 서버들로부터 서버의 기본 정보가 담긴 패킷(Control Packet : Basic infor)를 받게 되고 이를 기반으로 서버 목록을 생성한다. 두 단계의 임계값 Threshold 1과 Threshold 2(이때, Threshold 2가 Threshold 1보다 크게 설정한다.)에 대한 정보가 담긴 패킷을 서버로 전송한다.

컨트롤러는 서버 부하 관련 이벤트가 발생하기 이전까지 라운드로빈 방식을 적용하여 부하 분산을 실시한다. 이후 특정 서버로부터 부하 상태에 관한 이벤트 패킷을 받게 되면 컨트롤러는 전체 서버를 대상으로 부하 상태 확인을 위한 패킷을 요청하고, 서버들로부터 패킷을 응답받으면 서버들에게 부하 상태 기반의 가중치를 부여하여 클라이언트의 요청을 할당한다.

3.3 Server의 동작

시스템이 시작되면 서버는 컨트롤러로 기본 정보가 담긴 Control Packet(Basic Infor)를 보내 컨트롤러의 서버 목록에 등록하고 할당된 클라이언트의 요청 패킷을 처리한다. 서버의 부하가 컨트롤러로부터 전달받은 두 단계의 임계값(Threshold 1, Threshold 2)을 각각 초과하거나 이하로 내려가는 이벤트가 발생 시 Control Packet을 전송한다.

컨트롤러로부터 서버 부하 상태 확인을 위한 요청 패킷을 받게 되면 자신의 CPU와 메모리의 사용률을 기반으로 부하 상태를 판단한 뒤 응답 패킷을 컨트롤러로 전송한다. 컨트롤러가 할당한 가중치에 따라 클라이언트 요청 패킷을 처리한다.

이때, 서버의 부하 판단은 식 (1), (2), (3), (4)를 따른다.

$$SL_i = \alpha \times C_i + \beta \times M_i \quad \cdots (1)$$

if ($TH_1 < SL_i < TH_2$) then

server state msg = 1 $\cdots (2)$

else if ($TH_2 < SL_i$) then

server state msg = 2 $\cdots (3)$

else

server state msg = 0 $\cdots (4)$

i : 서버 번호

SL_i : 각 서버의 자원에 대한 부하 상태

C_i : i번 서버의 CPU 사용률

M_i : i번 서버의 메모리 사용률

α, β : 부하 판단 요소별 가중치,

TH_1, TH_2 : 임계값 Threshold 1, Threshold 2

식 (1)에 따라 각 서버는 CPU 사용률과 메모리 사용률에 요소별 가중치를 적용하여 부하를 판단한다. SL_i이 TH_1보다 크고 TH_2보다 작아지면 식(2)처럼 서버는 컨트롤러에게 상태 보고 값으로 1을 전송하고, SL_i이 TH_2보다 커질 경우에는 식(3)처럼 컨트롤러에게 상태 보고 값으로 2를 전송한다. 마지막으로 SL_i이 TH_1보다 작아지면 식(4)처럼 컨트롤러에게 상태 보고 값으로 0을 전송한다.

Ⅳ. 실험 및 분석 결과

4.1 시스템 구현 및 실험 환경

서버 부하 상태를 고려한 단계적 가중치 부하 분산의 효과를 확인하기 위해 지속적인 패킷을 생성하는 클라이언트와 서버들의 부하 상태 확인하고 부하 분산 정책을 적용하는 컨트롤러, 수신받은 부하 분산 정책을 기반으로 클라이언트의 요청을 할당하는 스위치, 클라이언트의 요청을 수신하고 처리하는 서버를 Riverbed Modeler로 구축하였다.

제안하는 부하 분산 기법을 가상 네트워크 환경에서 구현하여 효과를 평가하였다. 실험 환경은 CPU Intel(R) Core(TM) i7-8750H @ 2.20GHz, OS Windows 10, RAM 16GB, Riverbed Modeler 18.10.0과 같다.

4.2 실험방법

제안기법의 효율성을 검증하기 위해 서버별 부하율과 서버별 임계값(Threshold 2)에 도달한 횟수를 대조 기법(라운드로빈)과 비교한다. 본 논문에서는 서버들의 성능은 동일하게 설정하고 부하 측정을 위한 CPU와 메모리의 가중치를 동일한 비율로 설정하였으며, 서버별 임계치(Threshold 1, Threshold 2)는 각각 50%, 70%로 설정하였다.

우선 대조기법과 제안기법의 부하 분산 효율을 비교하기 위해 서버 클러스터의 평균 부하율이 85%이하 수준으로 유지되도록 클라이언트 요청 패킷을 지

속적으로 생성한다. 그런 다음 각 서버의 부하율을 측정하여 서버별 최대 부하율의 평균과 표준편차 값을 구하고 제안기법과 대조기법을 비교한다.

다음으로 제안기법과 대조기법의 서버별 임계값(Threshold 2)에 도달하는 횟수를 비교하여 제안기법을 통해 단계적으로 부하 분산이 이루어지는지 확인한다.

4.3 서버별 부하율 측정

제안기법과 대조기법를 적용하였을 때 서버별 부하율을 측정한 결과는 그림 4, 그림 5, 그리고 표 1, 표 2과 같다. 제안기법과 대조기법의 최대 부하율 평균은 각각 82%, 87.19%으로 제안기법이 5.19%가량 향상된 성능을 확인할 수 있으며, 서버별 표준편차(Std dev)의 평균값은 제안기법이 16.46%, 대조기법이 19.53%로, 제안기법이 대조기법보다 3.07% 가량 균등하게 부하 분산이 이루어진 것을 볼 수 있다.

이처럼 제안기법은 두 단계의 임계값 기준으로 서버의 부하 상태를 단계적으로 확인하고 전체 서버별 부하 상태에 따른 가중치를 부여하여 사용자 요청을 할당하기 때문에 서버의 부하를 고려치 않고 순서대로 시간 단위로 서버에 요청을 할당하는 대조기법보다 균등한 부하 분산을 달성할 수 있다.

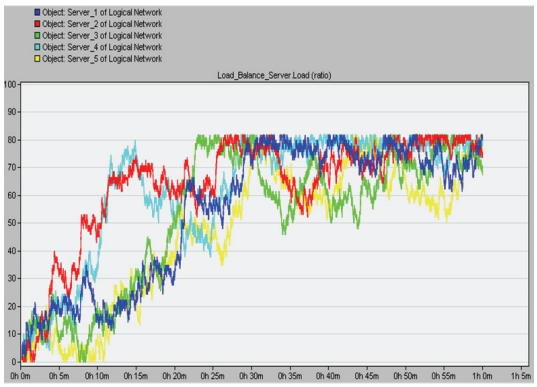

그림 4. 제안기법의 서버별 부하율 측정
Fig. 4 Measurement of load ratio for each server of the proposed method

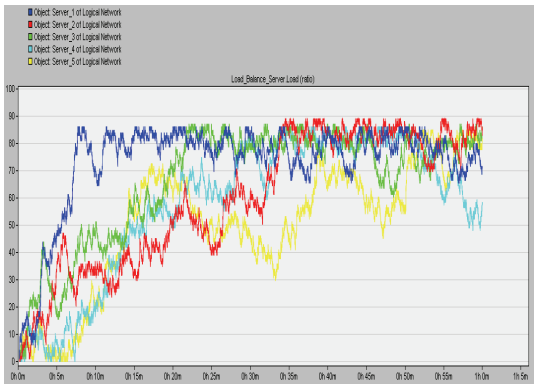

그림 5. 대조기법의 서버별 부하율 측정
Fig. 5 Measurement of load ratio for each server of the Round Robin

표 1. 제안기법의 서버별 부하율 측정 결과
Table 1. Load for server of the proposed method

Object	Minimum	Average	Maximum	Std dev
server1	0.0813	0.6314	0.8202	0.1631
server2	0.0803	0.6408	0.8194	0.1692
server3	0.0811	0.6177	0.8216	0.1611
server4	0.0813	0.6523	0.8197	0.1684
server5	0.0809	0.6003	0.8191	0.1616
Total avg	0.0809	0.6285	0.8200	0.1646

표 2. 대조기법의 서버별 부하율 측정 결과
Table 2. Load for server of the contrast method

Object	Minimum	Average	Maximum	Std dev
server1	0.0844	0.6831	0.8674	0.1992
server2	0.0816	0.6411	0.8954	0.1952
server3	0.0831	0.6203	0.8760	0.1968
server4	0.0813	0.6632	0.8643	0.1932
server5	0.0832	0.6143	0.8564	0.1921
Total avg	0.0827	0.6444	0.8719	0.1953

4.4 서버별 임계값 도달 횟수 측정

각 서버가 Threshold 2의 임계점에 도달하였을 때, 컨트롤러로 부하 상태 보고 패킷을 전송한 횟수를 측정한 결과는 그림 5, 그림 6, 그림 7, 표 3과 같다. 패킷 전송횟수는 대조기법이 평균 3131번, 제안기법이 평균 2866번으로 제안기법이 대조기법보다 8.5% 가량 적게 부하 상태 보고 패킷을 전송하였다. 부하 상태 보고 패킷을 적게 전송한 만큼 서버 자원들이 Threshold 2에 적게 도달하였으므로 서버 자원의 과부화 정도가 상대적으로 개선되었다고 판단할 수 있다.

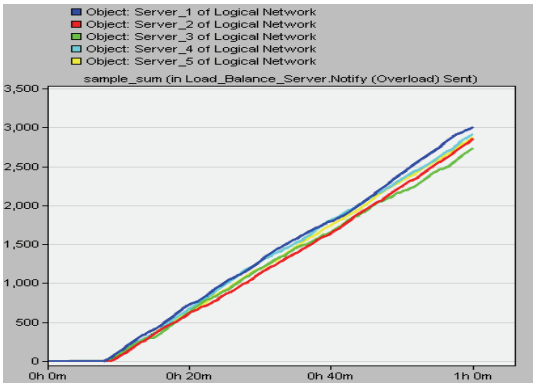

그림 6. 서버별 Overload 전송 측정(제안기법)
Fig. 6 Measurement of overload transmission by server(Proposed Method)

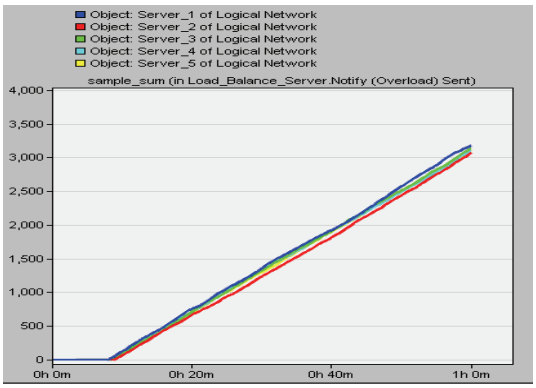

그림 7. 서버별 Overload 전송 측정(대조기법)
Fig. 7 Measurement of overload transmission by server(R.R)

표 3. 대조기법과 제안기법의 서버별 Overload
Table 3. Load for server of the contrast method

Object	Round Robin	Proposed Method
server1	3192	2998
server2	3065	2827
server3	3162	2756
server4	3121	2908
server5	3117	2843
Total avg	3131	2866

V. 결론

본 논문에서는 SDN 환경에서 서버가 두 단계의 임계점을 기준으로 부하 상태를 컨트롤러에게 보고하면 컨트롤러는 전체 서버를 대상으로 현재 부하 상태를 확인하고 가중치를 부여하여 사용자 요청을 할당하는 기법을 제안하였다. 대조기법과 성능을 비교한 결과, 제안기법은 서버의 최대 부하율과 임계값 도달 횟수를 낮추어 서버 부하율을 개선 시키고, 좀 더 균등한 부하 분산을 달성하여 자원을 효율적으로 활용하는데 효과적임을 확인하였다.

제안기법은 SDN 환경에서 구현되었기 때문에 대규모 서버 클러스터 환경에서 네트워크 관리자가 자원의 가용 능력을 파악하여 요구사항에 따라 유연하게 임계값을 부여하기 용이하다. 이러한 과정을 거치면 제안기법의 효과는 더욱 커질 것으로 기대되며 앞서 언급한 기존 네트워크의 확장성과 유연성 측면에서 보이는 한계를 극복하기 위한 하나의 선택지가 될 것이다.

References

[1] Statista “Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025,” *Report*, May 2022.
[2] B. Yoon, “Future Networking Technology of SDN,” *Electronics and Telecommunications Trends*, vol. 27, no. 2, 2012, pp. 129-136.
[3] D. Min, “Market Trends of SDN/NFV Supply and Demand,” *Electronics and Telecommunications Trends*, vol. 31, no. 2, 2016, pp. 28-40.
[4] IDC, “Worldwide Data center Software-Defined Networking Forecast, 2019-2023,” *Report*, Nov. 2019.
[5] S. Kang, Y. Kim, and S. Yang, “SDN Core Technology and Evolution Prospect Analysis,” *Information & Communications mag*, vol. 30, no. 3, 2013, pp. 3.8.
[6] Y. Seo and M. Lee, *Understanding OpenFlow using open source Introduction to SDN. Seoul*: YOUNGJIN.COM, 2014.
[7] J. Kim and T. Kwon, “Efficient Load Balancing Technique Considering Data Generation Form and Server Response Time in SDN,” *J. of the Korea Institute of Electronic Communication Sciences*, vol. 15, no. 4, 2020, pp. 679-686.
[8] P. Deepalakshmi, "D-Serv LB: Dynamic server

load balancing algorithm," *International Journal of Communication Systems,* vol. 32, issue 1, 2019, pp. 293-310.
[9] J. Lee and T. Kwon, "Efficient Load Balancing Technique through server load threshold Alert in SDN," *J. of The Korea Institute of Electronic Communication Sciences,* vol. 16, no. 5, 2021, pp. 817-814.
[10] K. Lee and T. Kwon, "Server state-based weighted load balancing techniques in SDN environments," *J. of The Korea Institute of Electronic Communication Sciences,* vol. 17, no. 6, 2022, pp. 1039-1046.

저자 소개

이재영(Jae-Young Lee)

2016년 부경대학교 전자공학과 졸업(공학사)
2022년~현재 국방대학교 대학원 컴퓨터공학과

※ 관심분야 : Networking, SDN

권태욱(Tae-Wook Kwon)

1986년 육군사관학교 전자공학과 졸업(공학사)
1995년 美 해군대학원 컴퓨터공학과 졸업(공학석사)
2001년 연세대학교 대학원 컴퓨터공학과 졸업(공학박사)
2007년~현재 국방대학교 컴퓨터공학과 교수

※ 관심분야 : Next Generation Networking, Content Centric Networking, Software Defined Networking, Network Function Virtualization, U-Sensor Networking, VR, RFID