ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

소프트웨어 정의 네트워크상의 미들박스 성능을 고려한 동적 서비스 체이닝 방안☆

Dynamic Service Chaining Method Considering Performance of Middlebox Over SDN

오 현 석[1] 김 남 기[1*] 최 윤 호[2*]
Hyeongseok Oh Namgi Kim Yoon-Ho Choi

요 약

기존 동적 라우팅은 최소 링크 비용을 기준으로 최적의 라우팅 경로를 설정하고 이에 따라 유입 혹은 유출 되는 플로우를 단말로 전송한다. 하지만 이 경우에는 보안 기능을 담당하는 미들 박스를 우회하게 되고 이에 따라 해당 네트워크는 보안 위협을 직면하게 되는 문제가 발생한다. 따라서 최적의 라우팅 경로 설정 시 각 플로우의 서비스 유형을 고려하여 해당 플로우를 보안 미들 박스를 거쳐 단말로 전송하기 위한 라우팅 방안인 동적 서비스 체이닝이 필요하다. 본 논문에서는 SDN의 동적 플로우 트래픽 제어 기능과 보안 서비스 기능을 동시에 고려한 동적 라우팅 방안에 대해 제안한다.

☞ 주제어 : 소프트웨어 정의 네트워크; 동적 라우팅; 미들박스; 서비스 체이닝

ABSTRACT

The conventional dynamic routing methods in Software Defined Networks (SDN) set the optimal routing path based on the minimum link cost, and thereby transmits the incoming or outgoing flows to the terminal. However, in this case, flows can bypass the middlebox that is responsible for security service and thus, thus the network can face a threat. That is, while determining the best route for each flow, it is necessary to consider a dynamic service chaining, which routes a flow via a security middlebox. Therefore, int this paper, we propose a new dynamic routing method that considers the dynamic flow routing method combined with the security service functions over the SDN.

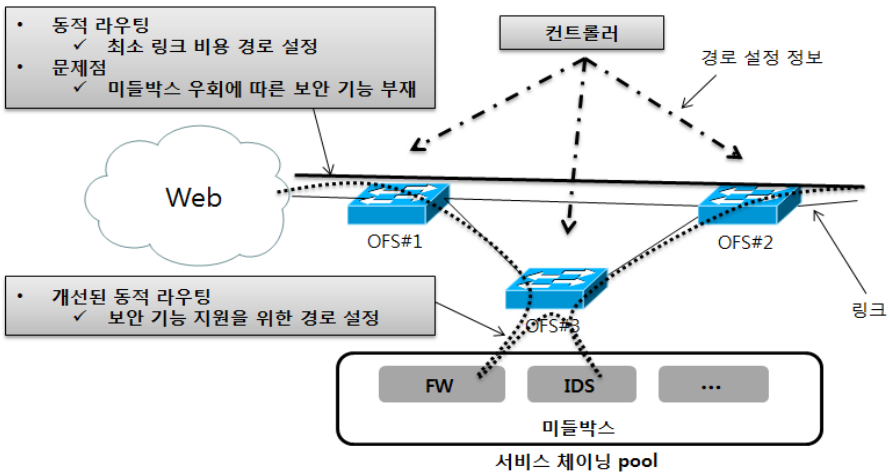
☞ keyword : Software Defined Networks; Dynamic Routing; Middlebox; Service Chaining

1. 서 론

Software Defined Networking (SDN) 기술의 등장으로 기존의 전송 평면과 제어 평면이 개별 네트워크 장비에 통합되어 분산 패킷 처리하는 환경에서, 전송 평면과 제어 평면을 논리적으로 분리하여 중앙 집중화된 제어 평면에 프로그래머블 기능과 개방형 API(e.g., Openflow[1])를 제공함에 따라 특정 벤더에 종속적이지 않고 다양한 가상의 네트워크 서비스를 중앙에서 필요에 따라 실시간으로 직접 운영 혹은 관리하는 것이 가능하게 되었다. 이와 함께, 범용 서버 성능의 고속화 및 대용량화와 운영체제 가상화 기술의 발달로 인해서 통신 사업자들은 Network Functions Virtualization (NFV) 기술을 통해 매우 빠른 서비스 전환에 따른 장비 운용 및 관리 비용 부담을 절감하고 공간 및 전원 소비를 줄임으로써 Return of Investment (ROI) 와 전체적으로는 Total Cost Ownership (TCO)를 최소화할 수 있게 되었다.

Open Networking Foundation (ONF)는 2009년 12월31일, 공개 API인 OpenFlow를 사용하여 SDN 스위치와 SDN 컨트롤러 간에 플로우 경로 제어 정보를 교환하기 위한 통신 규약을 정의하여 배포하였다[2]. SDN 컨트롤러는 임의의 플로우가 스위치에 진입하면 데이터 통신망 링크를 통과하는 비용 즉, 링크 비용의 합이 가장 작은

[1] Department of Computer Science, Kyonggi University, Suwon, 443-760, Korea.
[2] School of Computer Science and Engineering, Pusan National University, Busan, 609-735, Korea.
* Corresponding author (ngkim@kyonggi.ac.kr, yhchoi@pusan.ac.kr)
[Received 30 May 2015, Reviewed 2 June 2015, Accepted 4 November 2015]
☆ 본 연구는 연구재단 논문연구과제(NRF-2013R1A1A1005991)와 2014학년도 부산대학교 신임교수연구 정착금 지원으로 이루어졌음



(그림 1) 링크 비용을 고려한 동적 라우팅 경로와 링크 비용과 서비스 체이닝을 고려한 동적 라우팅 경로의 경로 설정 과정 비교

(Figure 1) Routing Comparison of Dynamic Routing based on Link Cost and Dynamic Routing based on Link Cost and Service Chaining

경로를 예약함으로써 해당 플로우에 대한 최적의 경로를 정적 혹은 동적 라우팅 알고리즘을 통해 설정한다[3].

ETSI 표준 문서에 따르면 SDN과 NFV는 상호 보완적인 기술로서 SDN은 네트워크를 추상화하여 가상 네트워크를 효율적이며 신속한 혁신을 제공한다. NFV는 다양한 네트워크 기능을 수행하는 방화벽, 침입 탐지 시스템 등의 미들박스를 가상화하여 유연하고 확장 가능한 네트워크 환경을 제공할 것으로 기대된다[4]. 하지만, SDN과 NFV 기술을 함께 사용하여 특정 플로우에 대한 최적의 경로를 결정하기 위해서는 기존의 라우팅 방안을 해당 플로우의 서비스 유형을 고려하여 개선할 필요가 있다.

그림 1에서 도시화한 바와 같이, 기존의 동적 라우팅의 경우 최소 링크 비용을 기준으로 최적의 라우팅 경로를 설정함에 따라 유입 혹은 유출 되는 플로우를 OFS#1과 OFS#2을 거쳐 단말로 전송한다. 여기서, OFS는 OpenFlow Switch, FW는 Firewall 그리고, IDS는 Intrusion Detection System을 의미한다. 하지만 이 경우, 보안 기능을 담당하는 미들박스를 우회하게 되고 이에 따라 해당 네트워크는 보안 위협을 직면하게 되는 문제가 발생한다. 따라서 최적의 라우팅 경로 설정 시 각 플로우의 서비스 유형을 고려하여 해당 플로우를 OFS#1, OFS#1과 OFS#3을 거쳐 단말로 전송하기 위한 서비스 유형을 고려한 라우팅 방안, 이하 '서비스 체이닝', 이 필요하다[5].

본 논문에서는 SDN의 동적 플로우 트래픽 제어 기능과 NFV의 가상화 네트워킹 서비스 기능을 동시에 고려한 동적 라우팅 방안에 대해 제안한다. 다양한 성능의 가상화 미들박스가 존재하는 네트워크상에서 플로우 서비스 유형의 변환에 따른 동적 서비스 체이닝 방안의 부재로 인한 플로우의 도착시간 지연, 패킷 손실을 경감시키기 위하여 네트워크에서의 링크 비용과 미들박스의 성능을 고려한 플로우 기반의 동적 라우팅 방안에 대해 제안한다. 이를 통해, 플로우의 도착시간 지연, 패킷 손실을 경감시키는 방안을 제안하고자 한다. 본 논문의 기여사항을 요약하면 다음과 같다.

- 데이터 센터 내에 존재하는 다양한 미들 박스를 고려한 서비스 체이닝 방안을 제안한다.
- 데이터 센터 망내 미들 박스를 경유하는 플로우 증가에 따른 효율적인 라우팅 경로 설정 방안을 제안한다.
- 기존 방안과의 비교 실험을 통해 제안하는 방안의 효율성을 검증한다.

본 논문은 다음과 같이 구성되었다. 2장에서는 관련 연구에 대하여 서술하며, 3장에서는 제안된 알고리즘에 대해 기술한다. 4장에서는 제안한 알고리즘의 효율성을 검증한 결과를 기술한다. 마지막으로, 5장에서는 논문의 중요성에 대하여 요약한다.

(표 1) SDN 플로우 관리 방안 비교
(Table 1) Comparison on SDN Flow Management Methods

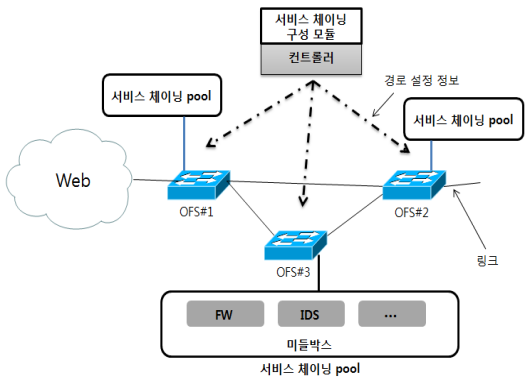
구분	OSPF	ECMP	Hedera	제안
적용 대상	패킷	플로우	대용량 플로우	플로우
라우팅 metric	링크 비용	링크 비용 + 로드 밸런싱	링크 비용 + 스위치 중복 이용 현황	링크 비용 + 미들박스 현황
대상 네트워크	모든 네트워크	모든 네트워크	Fat-tree 네트워크	Fat-tree 네트워크
라우팅 유형	정적	동적	동적	동적

2. 관련 연구

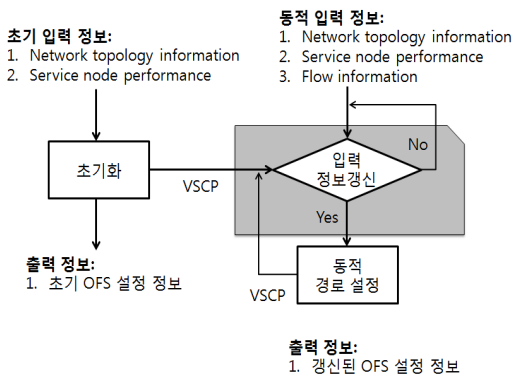
SDN에 대한 관심이 높아지면서 플로우 기반의 라우팅 기법들에 대한 연구가 이루어졌다. 출발지와 목적지 간의 최단 링크 비용을 계산하여 최단 경로를 찾는 Open Shortest Path First (OSPF) 방법은 동일 비용을 가지는 경로가 다수 존재할 때도 하나의 경로로 다수의 플로우들이 흐르게 되어 쉽게 특정 링크의 대역폭이 고갈될 수 있는 문제점이 존재한다. 이를 해소하기 위한 방안으로 Openflow를 활용하여 네트워크 내의 동일한 비용을 가지는 경로들에 한해서 플로우를 분배하여 할당해 주는 Equal Cost Multi Path (ECMP)가 제안되었다[6].

그러나 ECMP는 동일 비용 경로의 수가 적을 경우 OSPF와 마찬가지로 특정 경로의 대역폭이 고갈되는 문제가 발생할 수 있다. Fat-tree 내에서 이러한 문제점을 해결하는 방안으로 Hedera는 전체 링크 용량의 10% 이하 크기의 플로우를 Mice 플로우, 이상의 플로우를 Elephant 플로우로 구분하여 Mice 플로우에 대해서는 ECMP로 경로를 설정하고 Elephant 플로우에 대해서는 스위치의 할당된 플로우의 수를 최소화 하는 방향으로 경로를 결정한다[7, 8]. 이를 통해, Hedra의 경우 대용량 플루우를 라우팅하는 과정에서 효율적인 특성을 갖는다. 하지만 플로우의 크기를 처음부터 알고 있어야 하는 단점이 있어 비현실적이다[9].

앞서 설명한 방안 중 OSPF와 ECMP의 경우는 모든 네트워크 모델에 쉽게 적용할 수 있는 장점이 있다. 그러나 네트워크 구조에 따라 성능이 항상 보장되지 않고 특히 현재와 같이 다양한 성능의 장비들이 혼재되어있는 네트워크에서는 더욱 낮은 성능을 보일 수 있다. 이에 비해 Hedera는 대용량 플로우의 경로 설정과정에서 뛰어난 라우팅 성능을 나태내지만 실현 가능성이 낮은 전제조건과 Fat-tree라는 데이터 센터 내의 특정 네트워크에 적용이 가능하다는 단점이 있다. 또한 앞서 서론에서 언급한 바와 같이 SDN과 NFV 기술을 함께 사용하여 특정 플로우에 대한 최적의 경로를 결정하기 위해서는 기존의 라우팅 방안을 해당 플로우의 서비스 유형을 고려하여 개선할 필요가 있다.



(그림 2) 동적 서비스 체이닝 구성을 위한 네트워크 구성도
(Figure 2) Network Topology For Dynamic Service Chaining



(그림 3) 제안한 동적 서비스 체이닝 개요
(Figure 3) Overview of the Proposed Dynamic Service Chaining

본 연구에서는 특정 플로우가 OFS와 연결된 미들박스를 경유하여 목적지에 도착하기 위한 최적의 경로를 결정하는 과정에서 서비스 체인 경로를 간편하게 관리하고 동적으로 각 플로우의 서비스 체이닝 경로를 계산한다.

이를 통해, 특정 경로 내 플로우 증가 시 효율적으로 라우팅 경로를 설정하기 위한 방안을 제안한다. 표 1에서 플로우 관리를 위한 기존의 방안과 제안하는 방안을 비교 기술한다. 즉, 본 논문에서 제안하는 방안은 ECMP와 같이 플로우 분석에 기반하고 있지만 기존과 달리 링크 비용과 미들박스 현황을 고려한 라우팅 방안으로서, Hedera와 같이 Fat-tree 네트워크를 대상 네트워크로 하여 SDN에서의 보안 강화를 위한 동적 라우팅과 이를 통한 플로우 관리가 가능하다.

3. 제안하는 방안

3.1 시스템 구성 및 동작 개요

그림 2에서 서비스 체이닝 구성을 위한 네트워크 구성도를 도시화한다. 각 OFS에는 보안 서비스를 지원하기 위한 서비스 노드인 미들박스가 가상화 솔루션으로 구성된다. 서비스 체이닝을 위한 경로 설정 정보는 SDN 컨트롤러의 서비스 체이닝 구성 모듈에서 구성한다. SDN 컨트롤러는 각 OFS로 경로 설정 정보 전달 시 경로 설정을 위한 action값에 해당 정보를 반영함으로써 OFS가 미들박스로 특정 플로우를 포워딩한다. 또한 제안하는 서비스 구성 방안은 중복 경로를 최소화할 수 있어 현재 주목받고 있는 Fat-tree 구조의 네트워크에서 동작 가능하도록 설계한다.

3.2 서비스 체이닝 개요

동적 서비스 체이닝 구성을 위한 동작은 그림 3에서 기술한 바와 같이 초기화 단계와 플로우 경로 재설정 단계로 구성된다.

초기화 단계에서는 관리자가 NFV 기반 가상화 서비스 체이닝 풀을 포함한 관리 대상 네트워크의 토폴로지 정보, 미들박스의 플로우 처리 성능을 바탕으로 한 서비스 체인 구성 정보를 입력한다. 구체적으로는 인접 행렬의 형태로 저장된 네트워크 토폴로지와 미들박스의 플로우 처리 성능을 바탕으로 관리자는 서비스 타입(예: 목적지 포트 번호)에 따라 설정된 순차적인 서비스 체이닝 번호인 Virtual Service Chaining Path(VSCP)를 생성한다. 예를 들어 임의의 플로우가 그림 2의 OFS#1에서 출발하여 OFS#3에 있는 vFW과 vIDS를 통과하여 OFS#2로 전달되는 경우, 해당 플로우에 대한 VCSP는 vFW가 service node #1, vIDS가 service node #2인 경우 '1->2'.로 설정된다.

플로우 경로 재설정 단계에서는 서비스 체인 설정 후 동적으로 수집되는 네트워크 토폴로지 정보, 미들박스 성능, 플로우 정보(예: 미들 박스별 트래픽 처리량)를 바탕으로 서비스 체인 경로를 재계산한다.

다음 장에서 동적 경로 설정 알고리즘을 기술하기 위해서 다음의 변수를 정의한다.

- **topo**: topology
- **src**: source
- **dst**: destination
- **prevPath**: previous path
- **snlist**: service node list
- **sc_num**: service chain number
- **sc_paths**: service chain paths
- **alertList**: alert list of flows for path reconfiguration

3.3 동적 경로 설정

알고리즘 1은 Depth First Search (DFS)를 사용하여 주어진 출발지와 목적지를 연결하는 모든 경로를 탐색하여 저장하는 과정을 기술한다. 탐색 시 루프를 방지하고 불필요한 검색을 막기 위한 조건문을 확인한 뒤(2 – 3행), 탐색 경로 갱신과 (4 – 7행) 탐색트리에서 다음 탐색(11 – 12행)을 진행한다. 이 유사코드는 단순히 링크와 스위치와의 관계를 사용하여 스위치의 리스트로 구성된 모든 경로를 저장한다.

```
1. FindAllPath(topo,src,before,now,dst,prevPath,hop,limit,l
   en):
2. if hop > limit: break
3. if prevPath is None and now is not dst: break
4. if prevPath exists:
5.       m_path <- prevPath
6. else:
7.       m_path.add(now)
8. if now == dst:
9.       append global.path[src][dst] and break
10. for i in range(1, topo[0].length):
11.       if topo[now][i] != Inf and i != before:
12.             alen = len + topo[now][i]
13. FindAllPath(topo,now,i,dst,path,hop+1,limit,alen)
```

(알고리즘 1) 출발지-목적지의 모든 경로 탐색
(Algorithm 1) Source-Destination Path Search

```
1. GetServiceChainPath(vscp, sc_num, topo, snlist):
2. sub_topo = topo
3. for i in range(len(vscp)):
4.       searchlist[i] = getAllServiceNode(vscp[i], snlist)
5. for i in range(len(vscp) - 1):
6.       for src in searchlist[i]:
7.             for dst in searchlist[i+1]:
8.                   sub_topo[src][dst] = path_length(src, dst)
9. global.sc_path[sc_num] =
10.                  dfs_chain_path_search(sub_topo)
```

(알고리즘 2) VSCP에 대한 모든 경로 탐색
(Algorithm 2) Patch Search for VSCP

입력받은 네트워크 경로 내의 모든 호스트의 쌍과 호스트, 진입 스위치 쌍의 경로를 위 방안을 사용하여 저장을 하게 되고, 저장된 경로들은 추후 출발지와 목적지의 입력을 통하여 빠르게 경로를 얻을 수 있다. 이렇게 얻은 경로를 바탕으로 서비스 체인 경로를 재계산한다.

알고리즘 2는 관리자가 설정한 VSCP들에 대하여 모든 가능한 물리적인 경로를 탐색하는 과정으로, VSCP 내의 각 서비스 체인 단계의 타입에 해당하는 미들박스를 분류하고 (2 – 4행), 각 서비스 체인 단계 사이의 최단 경로 길이를 이용하여 인접행렬을 생성한다 (5 – 8행). 생성된 인접 행렬을 사용하여 알고리즘 1과 동일한 방법으로 DFS를 통한 서비스 체인 경로를 계산한다. 가장 최적의 경로의 집합을 쉽게 얻도록 경로의 지연시간은 각 링크의 지연시간들에 미들박스 처리율의 역수를 더한 값으로 설정한다. 모든 서비스 체인의 경로가 설정이 되면 해당 데이터들을 저장한 뒤, 차후 출발지와 목적지, 서비스 체인 번호의 입력만으로 빠르게 경로를 획득한다.

초기화 과정을 통한 네트워크 분석 및 경로 탐색 과정이 끝난 후에, 컨트롤러는 입력 플로우에 대하여 경로 할당을 수행한다. 관리자가 설정한 값 (특정 IP 또는 포트)을 사용하여 사전에 계산된 경로 집합을 빠르게 획득하여 해당 플로우 경로에 대하여 Round-Robin식으로 경로를 선택한다.

알고리즘 3은 이러한 플로우 경로 할당에 대한 유사코드로 플로우 정보를 통한 해당 서비스 체인 경로의 집합을 얻고(2행), 해당 경로 중 하나를 Round-robin 방식으로 선택하여(3 – 5행), 최종 선택된 경로를 스위치에 설정한다 (6행).

```
1. FlowAllocation(flowInfo):
2.       sc_paths = getServiceChainPath(flowInfo)
3.       idx = hash(flowInfo)
4.       selected_path = sc_paths[rr_val[idx]]
5.       rr_val[idx] += 1
6.       __install_path(selected_path)
```

(알고리즘 3) 플로우 경로 할당
(Algorithm 3) Flow Patch Allocation

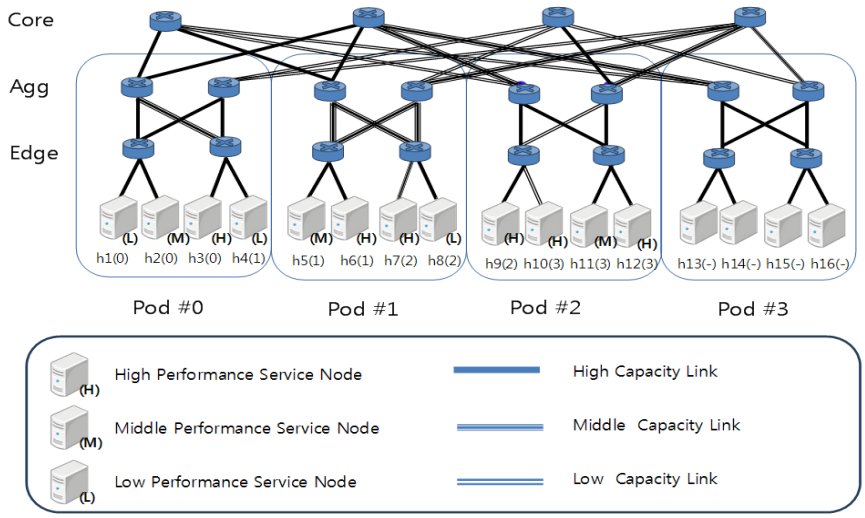
```
1. SearchOverheadPath(trafficInfo):
2. while all switch status data update:
3.       if link bandwidth > allocated bandwidth:
4.             flows = getFlow(link)
5.             alertList.insert(flows)
6.             call flowReconfiguration(alertList)
```

(알고리즘 4) 과부하 경로 탐색
(Algorithm 4) Overloaded Path Search

이러한 경로 설정 이외에도 컨트롤러는 주기적으로 경로별 남아있는 용량을 측정한다. 이를 위해서 컨트롤러는 주기적으로 OFS내 플로우 상태 정보를 요청하는 메시지를 전송하고 각 OFS는 상태 응답 메시지를 전송한다. 알고리즘 4에서 기술한 바와 같이 수집된 상태 응답 메시지는 컨트롤러 내의 토폴로지의 대역폭 정보를 가진 인접행렬내의 값과 비교함으로써 링크 내의 과부하 발생 여부를 체크한다.

```
1. FlowReconfiguration(alertList):
2. sort(alertList, 'decrease')
3. complete = False
4. den = {} //initialization
5. for i in range(len(alertList)):
6.       anotherPath = getOtherPath(alertList[i])
7.       if anotherPath.minBandwidth >= alertList[i].need:
8.             __install_flow_path(anotherPath, alertList[i])
9.             complete = True
10.            break
11.      den[alertList[i]][anotherPath] =
12.  alertList[i].need - anotherPath.minBandwidth
13.
14. if not complete:
15.       flow, path = getmin(den)
16.       __install_flow_path(path, now)
```

(알고리즘 5) 경로 재할당
(Algorithm 5) Patch Reallocation



(그림 4) 제안한 방안의 성능 검증을 위한 네트워크 토폴로지
(Figure 4) Network Topology for Experimental Evaluation of the Proposed Method

위 과정을 통하여 과부하가 발생된 링크내 플로우는 경로 재설정 대상이 되는데, 이때 재설정 대상 플로우중 가장 큰 부하를 갖는 플로우부터 시작하여 내림차순으로 다른 경로로 옮길 수 있는지 탐색하게 된다. 이때 대역폭 인접 행렬의 값을 이용하여 해당 서비스 체인의 경로에 대한 할당 가능한 용량을 계산하고 첫 번째로 적합한 경로를 선택하여 경로를 할당 한다.

알고리즘 5는 경로 재 할당을 위한 유사코드로 플로우를 내림차순으로 정렬하고 (2행), 플로우의 요구 대역폭에 대하여 적합한 대역폭을 갖는 경로를 찾아 할당하며 (5 - 10행), 만약 없을 경우 요구 대역폭과 할당 가능 대역폭의 차이를 기록한다 (11 - 12행). 만약 모든 탐색 후에 플로우 재설정을 못한 경우 계산된 요구와 가능의 격차가 가장 적은 경로에 플로우를 할당한다 (14 - 16행).

4. 성능 검증

제안한 서비스 체이닝 방안을 검증하기 위하여 POX 컨트롤러[10]와 Mininet[11]을 사용하여 그림 4와 같은 네트워크 토폴로지를 갖는 가상 네트워크를 구축하고 성능을 비교 분석하였다.

성능 검증을 위해 사용한 토폴로지는 서버와 스위치의 집합인 4개의 Pod를 포함한 Fat-Tree로 특정 링크와 특정 노드의 성능이 다른 링크와 노드보다 높은 경우를 가진다. 각각의 Edge Switch는 두 개의 미들박스가 연결된다. 체인을 위한 미들 박스는 Pod #0 ~ #2이며 각Pod에는 미들박스의 서비스 타입이 균일하지 않게 분포되어있다. 마지막에 위치한 Pod #3는 최종 서비스를 제공하는 미들 박스로 실험을 위하여 파이썬의HTTPServer를 CPU 부하가 발생하는 코드를 추가한 뒤 구동시켰다. 또한 각 Core switch 상단에 서비스 요청 단말을 2개씩 두어 Http Request에 대한 응답을 통하여 성능 측정이 가능하도록 하였다.

(표 2) 노드 또는 링크 설정
(Table 2) Node and Link Setup

Link			Service Node	
			성능	CPU
High capacity	capacity	4Mbps	High	1/100
	loss	0 %		
	delay	1 ms		
Middle capacity	capacity	3Mbps	Middle	1/2000
	loss	0 %		
	delay	1 ms		
Low capacity	capacity	2Mbps	Low	1/40000
	loss	0%		
	delay	1 ms		

(표 3) VSCP 설정
(Table 3) VCSP Setup

Service Chain ID	Virtual Service Chain Path (VSCP)
0	0 -> 2 -> 3
1	1 -> 3
2	2 -> 3
3	1 -> 2 -> 3

4.1 실험 환경

본 실험에서 사용된 링크와 미들 박스는 고/중간/저 성능 링크와 고/중간/저 성능 서비스로 분류를 하였는데 각 항목들에 대한 구체적인 설정 값은 표 2와 같다. 링크는 손실율과 딜레이는 동일하고 처리 용량이 다르며 미들 박스는 오직 처리 용량만을 다르게 설정하였고, 이때의 처리용량은 지정된 것이 아닌 다른 미들 박스에 비해 얼마나 높은지 상대적인 비율로 나타내었다.

또한 사용된 VSCP의 종류는 표 3과 같이 총 4가지로 구성하였으며, 표 4와 같은 시스템 구성을 사용하여 본 실험을 진행하였다.

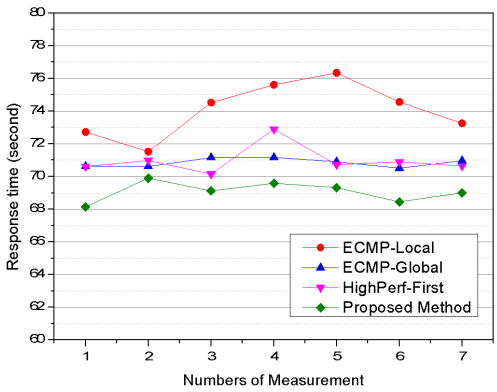
(표 4) 실험 시스템 사양
(Table 4) System Specification for Experiment

Elements	Specification
CPU	Intel i5-4210U
RAM	8GB
DISK	120 GB SSD
OS	Linux Mint 17
POX	0.90
Mininet	2.0.1
OpenVSwitch	1.3

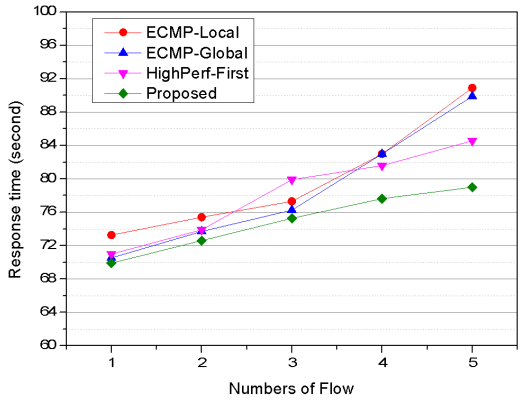
4.2 성능 분석 결과

제안한 방안의 성능 검증을 위하여, ECMP 기반의 정적인 경로 설정 방안과 미들 박스의 성능을 우선적으로 고려하여 경로를 설정하는 방안의 성능을 분석하였다.

비교 대상으로 사용된 ECMP 기반의 정적 경로 설정 방안은 크게 링크만을 고려하는 방안과 미들 박스의 성능만을 고려하는 방안으로 나누어진다. 링크만 고려하는 경우는 경로 계산시의 처리 시간의 향상을 위하여 서비스 체인 단계와 단계 사이의 최솟값만을 고려하여 경로를 생성하는 방안 (ECMP-Local)과 전역적인 최적화를 추가적으로 진행하는 방안 (ECMP-Global)을 비교 대상으로 선정하였다. 이에 반해 호스트 고려 방안 (HighPerf-First)은 서비스 체인 단계를 고려하여 동일 서비스 중 가장 성능이 좋은 미들 박스를 선택한다.

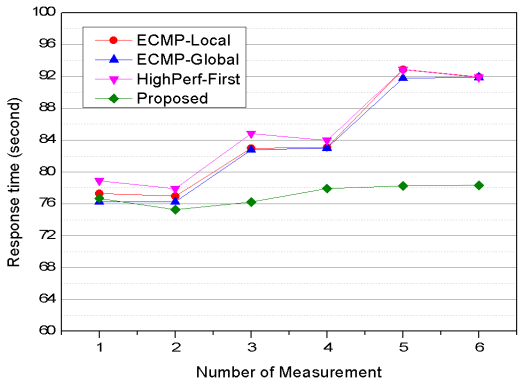


(그림 5) 단일 플로우에 대한 응답시간
(Figure 5) Response Time per Flow



(그림 6) 플로우 수의 증가에 따른 평균 응답시간
(Figure 6) Average Response Time for the Increase of the Number of Flows

검증 과정에서는 단일 플로우 진입의 경우와 다수의 균일한 플로우를 서비스 체인 구조 내에 보내는 경우, 일정 플로우를 서비스 체인 구조 내에 보내는 경우에 대한 평균 응답시간을 측정함으로써 각 서비스 또는 링크 별 부하를 측정하였다.



(그림 7) 단일 플로우 트래픽 양의 증가에 따른 평균 응답시간
(Figure 7) Average Response Time for the Increase of the Single Flow Traffic

먼저 단일 플로우에 대한 응답 시간을 측정하는 실험을 진행하였다. 해당 실험은 알고리즘이 얼마나 적합한 경로를 선택하여 플로우를 할당하는지를 확인하기 위한 실험으로, Apache Benchmark를 이용하여 서비스 제공자에게 1MB의 텍스트 문서를 첫 번째 서비스 체인을 통하여 Http Request를 전송함에 따른 응답시간을 측정하였다.

그림 5와 같이 ECMP-Local과 ECMP-Global는 모두 링크와 스위치만을 고려한 최단 경로 중 가장 첫 번째 경로를 선택하는 방법이다. 하지만, ECMP-Local의 경우 첫 번째 서비스 체인 단계를 결정할 때 목적지가 다음 단계의 경로에 영향을 주므로 전체적인 최적 경로라고 할 수 없다. 따라서, 전체적인 최적 경로를 계산하는 ECMP-Global의 응답시간이 조금 더 빠른 것을 볼 수 있다. HighPerf-First의 경우 가장 성능이 좋은 미들 박스를 선택하는 과정에서 경로의 길이가 길어짐에 따라 최적의 성능을 보이지 않았다. 이에 비하여 제안한 방안은 성능과 경로를 함께 고려하여 경로를 계산하므로 가장 낮은 응답시간을 보임을 확인할 수 있었다.

다음으로 관리 네트워크 내에 동시에 다수의 플로우가 진입했을 경우 각 플로우 별 평균 응답시간을 측정하였다. 그림 6에서와 같이 플로우 개수의 증가에 따른 각 플로우 응답시간이 증가하는 것을 확인할 수 있다. 제안 방안의 경우 실험에 사용된 토폴로지 상에서 현재 상황에 적합한 사용 가능한 경로를 계산하여 할당하므로, 다른 방안에 비하여 평균 응답시간 증가율이 최대 58% 감소하는 것을 확인할 수 있다.

마지막으로 일정 플로우를 진입 시킨 후에 하나의 플로우 양을 증가시키는 실험을 진행하였다. 구체적으로는 네트워크 내에 3개의 플로우를 진입 시킨 후에 요청 데이터 크기를 700KB 증가 시키면서 3회 반복 실험을 진행하였다. 그림 7에 나타난 바와 같이, 기존 방안에서는 신규 플로우 수가 증가함에 따라 동적으로 플루우 양을 조정하는 부분이 존재하지 않아 플로우 양의 증가에 따른 지연시간이 큰 폭으로 늘어나는 것을 확인할 수 있다. 그러나 제안한 방안의 경우 주기적인 플로우 및 네트워크 정보 수집을 통하여 지연 발생 시 해당 플로우의 재조정으로 인해 평균 응답 시간의 증가율이 기존 대비 최대 88%가량 감소하는 것을 확인할 수 있었다.

5. 결 론

본 논문에서는 초기 경로의 처리 용량 계산 시, 각 미들 박스의 처리 용량을 고려한 경로를 생성한 뒤, 계산된 경로에게 경로 할당 및 재 할당 과정을 반복하여 적합한 경로를 할당시키는 방안을 제안하였다. 실험을 통해 단일 플로우의 할당을 통하여 기존 세 방안 보다 실제적으로 빠른 응답시간을 가지는 경로를 선택함을 확인할 수 있었다. 또한, 동시에 다중의 플로우가 진입하였을 경우 플로우의 개수 증가에도 평균 응답시간 증가율이 최대 58%가량 감소하는 것을 확인할 수 있었으며, 기존 플로우의 용량 증가에도 평균 응답시간 증가율이 최대 88%가 감소하는 것을 확인할 수 있었다. 향후 다양한 네트워크 구조 내에서 많은 플로우가 진입할 경우에 대한 방안을 모색할 예정이다.

참 고 문 헌 (Reference)

[1] Nick McKeown, Tom Anderson, Hari Balakrishnan, Scott Shenker and Jonathan Turner, “OpenFlow: Enabling innovation in campus networks,” ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, April 2008. http://dx.doi.org/10.1145/1355734.1355746

[2] Open Networking Foundation, “OpenFlow Switch Specification Version 1.0.0,” White Paper, pp. 1-42, Dec. 2009. http://dx.doi.org/10.1145/1355734.1355746

[3] Wolfgang Braun and Michael Menth, ″Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices,″ Future Internet, vol. 6, pp. 302-336, 2014. http://dx.doi.org/10.3390/fi6020302

[4] Open Networking Foundation, “OpenFlow-enabled SDN and Network Functions Virtualization,” ONF Solution Brief, pp. 1-12, Feb. 2014.
http://dx.doi.org/10.3390/fi6020302

[5] Nicolai Leymann, Deutsche Telekom AG, ″Flexible Service Chaining: Requirements and Architectures,″ ERLEBEN WAS VERBINDET, 2013.
http://www.ewsdn.eu/files/Presentations/EWSDN%202013/IS2_2_Flexible_service_chaining.pdf

[6] OpenFlow, “Openflow Multipath Proposal,” 2011.
http://archive.openflow.org/wk/index.php/Multipath_Proposal

[7] Mohammad Al-Fares, Sivasankar Radhakrishnan, Barath Raghavan, Nelson Huang and Amin Vahdat, “Hedera: Dynamic Flow Scheduling for Data Center Networks,” Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, pp. 281-296, 2010.
http://dl.acm.org/citation.cfm?id=1855730

[8] Mohammad Al-Fares, Alexander Loukissas and Amin Vahdat, “A scalable, commodity data center network architecture,” Proceedings of the ACM SIGCOMM 2008 conference on Data communication, pp. 63-74, Aug. 2008.
http://dx.doi.org/10.1145/1402958.1402967

[9] Niels L. M. van Adrichem, Christian Doerr and Fernando A. Kuipers, “OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks,” Proc. of the IEEE/IFIP Network Operations and Management Symposium (IEEE/IFIP NOMS 2014), May 2014.
http://dx.doi.org/10.1109/noms.2014.6838228

[10] nox, “About POX,”
http://www.noxrepo.org/pox/about-pox/

[11] Mininet, “An Instant Virtual Network on your Laptop,”

◐ 저 자 소 개 ◑

오 현 석 (Hyeongseok Oh)
2013.2 경기대 컴퓨터과학과(공학사)
2015.2 경기대 컴퓨터과학과(공학석사)
관심분야: SDN, OpenFlow, NFV
E-mail : mytemp_id@naver.com

김 남 기 (Namgi Kim)
1997.2 서강대 컴퓨터학과(공학사)
2000.2 KAIST 전산학과(공학석사)
2005.2 KAIST 전산학과(공학박사)
2005.3～2007.2 삼성전자 통신연구소 책임연구원
2007.3～현재 경기대학교 컴퓨터과학과 교수
관심분야 : 통신시스템, 네트워크
E-mail : ngkim@kgu.ac.kr

최 윤 호 (Yoon-Ho Choi)
2002 경북대학교 전기전자컴퓨터공학부(공학사)
2008 서울대학교 대학원 전기컴퓨터공학부(공학박사)
2009 펜실베니아주립대학교 컴퓨터과학과 박사후연구원
2010～2011 삼성전자 책임연구원
2012～2014.8 경기대학교 융합보안학과 교수
2014.9～현재 부산대학교 정보컴퓨터공학부 교수
관심분야 : SW 보안, 모바일 보안, 지능형 자동차 IT 보안, 빅데이터 보안, SDN 보안 etc.
E-mail : yhchoi@pusan.ac.kr