소프트웨어 정의 네트워크에서 Q-learning기반 네트워크 혼잡방지 메커니즘

김선혁O, 이다은, Ashis Talukder, 홍충선*
경희대학교 컴퓨터공학과
{kshyuk0605, daeunlee, ashis, cshong}@khu.ac.kr

Network Congestion Prevention Mechanism based on Q-learning in Software-Defined Networking

Seonhyeok KimO, Daeun Lee, Ashis Talukder, ChoongSeon Hong*
Department of Computer Science and Engineering, Kyung Hee University

요 약

Software Defined Networking(SDN)은 기존 네트워크 운영 자동화와 중앙관리의 어려움, 벤더 의존성, 개별 처리로 인한 네트워크 복잡성 증가로 인한 문제를 해결하기 위해 대두되고 있다. 하지만 SDN 컨트롤러 중 하나인 OpenDaylight의 패킷 포워딩 프로토콜은 Dijkstra 알고리즘을 이용하여 최단 Flow Path를 선정하게 되는데, 이는 급작스런 트래픽 발생에 따른 대역폭 오버헤드를 고려하지 못하여 네트워크 경로상의 혼잡을 야기한다. 따라서 본 논문에서는 SDN 네트워크 환경에서 효율적인 패킷 포워딩을 위해 Q-learning기반 네트워크 혼잡방지 메커니즘을 제안한다. 제안하는 메커니즘에서는 대역폭의 임계값을 설정해 트래픽으로 인한 혼잡 발생 시 컨트롤러에서 Q-learning 알고리즘을 통해 경로를 재선정하고 Flow Table을 변경함으로써 트래픽 발생에 따른 네트워크 혼잡문제를 효과적으로 개선 한다.

1. 서 론

클라우드, 모바일, 사물인터넷 등과 같은 IT기술의 발전과 더불어 인터넷 사용량이 급격하게 증가되면서 기존 네트워크 운영 자동화와 중앙관리의 어려움, 벤더 의존성, 개별 처리로 인한 네트워크 복잡성 증가와 같은 구조적인 문제점이 지속적으로 발생하고 있다. 이를 해결하기 위해 미래인터넷과 관련하여 다양한 기술들이 개발되고 있으며, 그 중에서 소프트웨어 프로그래밍을 통해 네트워크 경로 설정, 제어 및 관리를 유연하게 처리 할 수 있는 SDN(Software Defined Networking)이 각광받고 있다[1][2]. SDN은 기존 네트워크 장비에서 Data plane과 Control Plane을 분리하여 소프트웨어 프로그래밍을 통한 네트워크 경로 설정 및 제어, 관리를 가능하게 한다. SDN의 아키텍처는 bottom-up방식의 3개의 layer로 구성되어 있으며 각 layer의 내용은 다음과 같다. 첫 번째로 infrastructure layer는 네트워크 상태를 수집하고 패킷 전송을 책임진다. 두 번째 control layer는 인프라 계층의 장치들을 제어하고 API를 통해 다양한 서비스 액세스 포인트를 제공할 수 있다. 마지막으로 application layer에서는 사용자의 요구사항을 만족시키기 위해 SDN 응용프로그램을 제공한다[2]. 이러한 장점들로 인하여 유연하고 확장성이 뛰어난 네트워크 환경을 구축 할 수 있다. 하지만 SDN 컨트롤러 중 하나인 OpenDaylight의 Dijkstra 알고리즘을 이용하는 패킷 포워딩 방법은 각 경로의 대역폭 변화를 고려하지 못해 네트워크 혼잡을 야기하는 문제가 있다.

본 논문에서는 이러한 문제를 해결하기 위해 OpenDaylight의 라우팅 프로토콜인 Dijkstra 알고리즘을 이용하는 최단 경로 선정 방식의 문제점을 파악한다. 그리고 대역폭의 임계값을 설정하여 트래픽 발생에 따른 대역폭의 오버헤드 발생 시 강화학습의 일종인 Q-learning 알고리즘을 적용하여 네트워크 혼잡문제를 최소화 하는 방안을 제안한다.

본 논문의 구성은 다음과 같다. 1장에서는 서론으로 본 논문의 전반적인 내용을 다루고, 2장에서는 본 논문에서의 연구와 관련된 연구를 서술할 것이다. 3장에서는 기존 패킷 포워딩 방법의 문제점을 분석하고, 4장에서는 네트워크 혼잡문제를 최소화하는 방안에 대해 제안한다. 5장에서는 이에 대한 시뮬레이션 및 테스트를 통해 성능 향상 결과를 보인다. 마지막으로 6장에서는 결론을 통해 연구 결과 및 향후 연구에 대한 내용으로 논문을 마무리 한다.

2. 관련연구

2.1. OpenDaylight

OpenDaylight는 SDN/NFV를 위한 오픈소스 소프트웨어 플랫폼으로 Cisco, IBM, Dell 등 여러 IT회사들이 SDN/NFV 발전의 가속화와 품질향상을 위해 OpenDaylight 프로젝트에 참여하고 있다. OpenDaylight는 OpenFlow, OpFlex 등의 다양한 네트워크 장비 제어 프

이 논문은 2015년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (B0190-15-2013, 유무선 통합 네트워크에서 접속 방식에 독립적인 차세대 네트워킹 기술 개발).
본 연구는 한국정보화진흥원(NIA)의 미래네트워크연구시험망(KOREN) 사업 지원과제의 연구결과로 수행되었음 (15-951-00-001).
*Dr. CS Hong is the corresponding author

로토콜을 지원하며, 표준화된 모델을 사용함으로써 플랫폼간의 이식성이 우수한 장점을 가지고 있다[3]. 기존 다른 컨트롤러와는 다르게 SAL(Service Abstraction Layer)이 컨트롤러와 네트워크 장비 사이의 프로토콜을 결정해 줌으로써 다양한 프로토콜 지원이 가능하다. 또한 모듈화와 확장성을 위해 OSGi(Open Services Gateway Initiative framework) framework를 사용하여 별도의 재부팅 없이 동적으로 번들(Application 또는 Component)을 추가 및 삭제 할 수 있다.

2.2. Q-learning

강화 학습(Reinforcement Learning)은 그림 1과 같이 환경(environment)을 탐색하는 에이전트(agent)가 현재의 상태(state)를 인식하여 특정 액션(action)을 취하고 에이전트(agent)는 환경(environment)으로부터 보상(reward)을 얻게 되는데, 이를 통해 에이전트(agent)의 보상(reward)을 최대화하는 방법을 찾는 과정을 의미한다. Q-learning은 이러한 강화학습 기술의 일종으로 Markov decision process에 기반을 두어 최적의 액션(action) 선택 방법을 찾기 위해 사용된다[3]. 결국 Q-learning을 통해 보상(reward)을 기반으로 출발점과 도착점 사이의 최단 경로를 구할 수 있다.

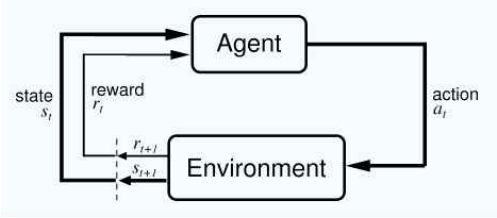

그림 1. 강화학습

3. SDN에서 기존 패킷 포워딩 방법의 문제점 분석

기존 SDN 컨트롤러중 하나인 OpenDaylight를 살펴보면 Dijkstra 알고리즘을 이용하는 라우팅 모듈을 사용하고 있다. OpenDaylight에서는 Dijkstra SPF(Shortest Path First) 모듈을 이용하여 출발지와 도착지 사이의 최단경로를 구하고 Southbound API인 OpenFlow 프로토콜을 통해 OVS(OpenVSwitch)의 Flow Table을 변경시킨다. 그리고 호스트에서 트래픽 발생 시 Flow Table에 미리 정해진 경로를 따라 트래픽이 전달된다[4]. 하지만 기존 OpenDaylight의 Dijkstra 알고리즘을 이용하는 패킷 포워딩 방법은 각 링크의 대역폭 변화를 고려하지 않고 라우팅 경로를 선정한다. 따라서 급작스런 트래픽 발생 시 특정 링크에서 트래픽증가로 인한 대역폭의 오버헤드를 고려하지 못해 네트워크상의 혼잡이 발생하게 되고 패킷 전송률이 떨어지게 된다. 즉, 처음 연결된 경로에 트래픽이 증가해 혼잡이 발생하더라도 이미 최적의 경로라고 결정된 경로로만 패킷을 포워딩 하여 네트워크 성능저하를 야기한다[5].

4. Q-learning을 통한 네트워크 혼잡방지 메커니즘

본 장에서는 급작스런 트래픽 발생 시 대역폭의 오버헤드를 고려하여 네트워크 혼잡 문제를 최소화하기 위한 방법에 대하여 제안한다. 제안하는 방법은 컨트롤러에서 대역폭의 임계값을 설정하고 네트워크 경로의 대역폭을 탐지한다. 대역폭 탐지 중 대역폭의 임계값을 초과하는 트래픽이 발생하면 컨트롤러에서 Q-learning 경로설정 알고리즘을 통해 트래픽 상황을 예측하고 네트워크 혼잡을 방지 할 수 있는 최적의 경로를 탐색한다. 알고리즘을 통해 변경된 경로는 OpenFlow를 통해 OVS에 전달되고 Flow Table이 변경된다. 따라서 변경된 경로를 따라 트래픽 플로우가 분산되어 네트워크 혼잡을 방지할 수 있다.

알고리즘 1은 이에 대한 Q-learning기반 경로설정 알고리즘의 수도코드이다. 수도코드의 구조를 보면 처음에는 초기화 단계이다. CB(CurrentBandwidth)를 800Mbit, gamma는 0.7, Iteration은 10으로 초기화하고, Q-Matrix값을 0으로 Reward는 TI(TopologyInformation)를 이용하여 초기화한다[3]. 다음으로 R-Matrix는 Reward를 이용하여 설정되며 컨트롤러는 네트워크 링크의 대역폭을 주기적으로 측정한다. 네트워크 링크를 측정하는 도중 예상되는 대역폭인 PB(PredictedBandwidth)가 현재 가용한 대역폭의 80%를 초과하면 해당 링크에 오버헤드가 발생 할 수 있으므로 R-Matrix 값을 변경한다. R-Matrix값은 다음 노드가 목적지 노드인 경우 30의 가중치를 주고, 다음 노드가 목적지 노드가 아닌 경우 20의 가중치를 준다. 변경된 R-Matrix값을 통해 Reward 값이 바뀌게 되고, 이를 Q-learning 알고리즘에 적용하여 새로운 최적의 경로를 구한다. 마지막으로 컨트롤러에서 재선정된 최적의 경로를 OpenFlow를 통해 OVS의 Flow Table에 적용함으로써 네트워크 혼잡문제를 최소화 할 수 있다.

알고리즘 1. Q-learning기반 경로설정 알고리즘

```
Input: Edge, Vertex, G=(V, E), PB, CB, Reward
       gamma, Iteration, Q-Matrix, R-Matrix, TI
Output: Q-Matrix, route
Initialization
    CB ← 800Mbit, gamma ← 0.7, Iteration ← 10
    Q-Matrix ← 0, Reward ← TI

while
    R-Matrix ← Reward
    for all e ∈ G.edges()
        if (PB > CB * 0.8) then
            e ⇒ Vsrc, Vdst
            if (Vdst == Goal state) then
                R[Vsrc][Vdst] value is 30
            else
                Find V adjacent Vsrc
                R[Vsrc][V] value is 20
    while(Iteration)
        Q(state, action) = R(state, action) + gamma
                         * Max[Q(next state, all actions)]
    Update Q-Matrix
    Update route
endwhile
```

5. 성능평가

앞 장에서 언급한 Q-learning기반 네트워크 혼잡방지 메커니즘의 타당성을 검증하기 위해 그림 2와 같은 토폴로지를 구성하고 네트워크 혼잡에 따른 데이터 전송 시간을 측정하여 시뮬레이션 한다. 시뮬레이션은 mininet를

통해 가상네트워크를 구성하여 진행 되었으며, OpenDaylight SDN 컨트롤러를 사용하여 네트워크 상황을 모니터링하고 제어한다.

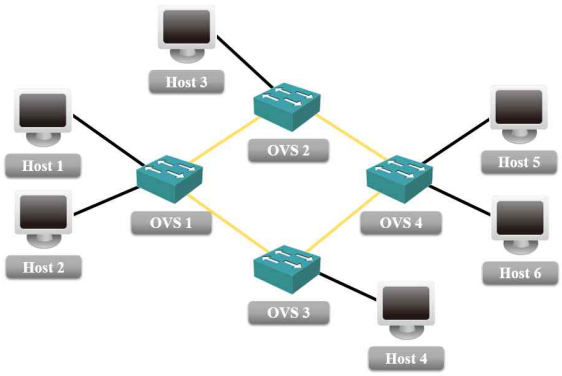

그림 2. 네트워크 토폴로지

기존 OpenDaylight 패킷 포워딩 방법과 본 논문에서 제안하는 방법을 적용한 패킷 포워딩 방법을 비교하기 위해 Host1에서 Host5로 데이터를 전송하였고, OVS1->OVS2->OVS4의 경로로 패킷이 전달됨을 확인하였다. 데이터 전송 도중 Host3에서 Host6으로 새롭게 데이터를 전송하여 OVS2와 OVS4 링크에서 오버헤드를 발생시켰고, 이 때 OVS의 상태 변화와 데이터 전송 시간을 측정하였다. 그림 3은 OpenDaylight에서 데이터 전송 시 OVS의 상태를 보여주는 컨트롤러 화면이다. 그림 3에서 보이는 것처럼 OVS2와 OVS4링크 사이에서 혼잡이 발생하였을 때 OVS3으로 경로를 재선정하여 패킷 포워딩이 되는 것을 확인 할 수 있다. 그림 4에서는 데이터 전송에 따른 전송시간 비교 결과를 보여준다. 시뮬레이션 결과 제안하는 방법이 기존 방법과 비교하여 좋은 성능을 보이며, 대규모의 파일 전송 시 더욱 효율적임을 확인할 수 있다.

Node	DL TYPE	NW Dst	Actions	Byte Count	Packet Count
ovs1	IPv4	10.0.0.5	OUTPUT = OFI1	1033781267	107191
ovs2	IPv4	10.0.0.6	OUTPUT = OFI2	1721687513	154896
ovs2	IPv4	10.0.0.5	OUTPUT = OFI2	685894215	45938
ovs3	IPv4	10.0.0.5	OUTPUT = OFI3	1033781267	107191
ovs4	IPv4	10.0.0.6	SET_DL_DST = d1:74:d0:47:10:e4 OUTPUT = OFI2	1721687513	154896
ovs4	IPv4	10.0.0.5	SET_DL_DST = e5:00:15:d1:88:ea OUTPUT = OFI1	1719675482	153129

그림 3. OpenDaylight에서 데이터 전송 결과

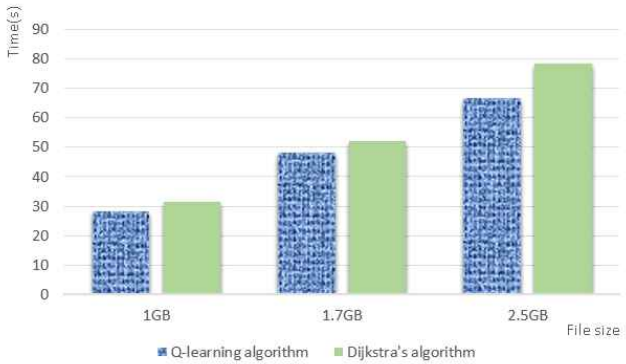

그림 4. 데이터 전송 평균 시간 비교

6. 결론 및 향후 연구

본 논문에서는 기존 OpenDaylight 컨트롤러에서 대역폭의 변화를 고려하지 않는 패킷 포워딩 방법의 문제점에 대해 살펴보았다. 그리고 이 문제를 해결하기 위해 Q-learning기반의 네트워크 혼잡방지 메커니즘을 제시하였다. 제안하는 메커니즘은 앞서 언급한 것처럼 대역폭의 임계값을 설정하고 트래픽 오버헤드로 인한 혼잡 발생 시 컨트롤러에서 이를 탐지한다. 그리고 Q-learning 알고리즘을 이용하여 최적의 경로를 재선정하였고, OpenFlow를 통해 OVS의 Flow Table을 변경함으로써 네트워크 혼잡문제를 해결하였다. 실험결과 데이터 전송 시 제안하는 메커니즘이 기존 패킷 포워딩 방법에 비해 더 효율적임을 확인 할 수 있었고, 이는 SDN환경에서 제안하는 메커니즘이 네트워크의 혼잡을 최소화할 수 있음을 의미한다.

하지만 제안하는 메커니즘은 대역폭 크기, 고정된 트래픽 발생 등 제한된 환경에서만 적용이 가능하다는 한계가 있다. 따라서 향후 연구에서는 실제 네트워크와 유사한 환경을 구축하고 다양한 시뮬레이션 및 테스트를 통해 좀 더 효율적인 알고리즘과 네트워크 혼잡 판단 기준을 세워야 할 것이다.

참 고 문 헌

[1] D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, S. uhling, “Software-Defined Networking: A Comprehensive Survey,” proceedings of the IEEE, Vol. 103, pp. 14-76, January 2015.

[2] B.A.A. Nunes, M. Mendonca, N. Xuan-Nam, K. Obraczka, T. Turletti, “A survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks”, Communications Surveys & Tutorials, IEEE, Vol. 16, pp. 1617-1634, February 2014.

[3] F. Farahnakian, M. Ebrahimi, M. Daneshtalab, P. Liljeberg, J. Plosila, “Q-learning based Congestion-aware Routing Algorithm for On-Chip Network”, Networked Embedded Systems for Enterprise Applications(NESEA), pp. 1-7, December 2011.

[4] Z.K. Khattak, M. Awais, A. Iqbal, “Performance evaluation of OpenDaylight SDN controller”, Parallel and Distributed Systems (ICPADS), 2014 20th IEEE Conference on, pp.671-676, December2014.

[5] J. Jehn-Ruey, H. Hsin-Wen, L. ji-Hau, C. Szu-Yuan, “Extending Dijkstra’s shortest path algorithm for software defined networking”, Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific, pp.17-19, September2014.