NI 기술 문서

Deep Q-network를 활용한 OpenStack 환경 내 최적 SFC 구성 방법

문서 ID	NI-SFC-02
문서 버전	v1.1
작성일	2021.07.19
작성자	포항공과대학교 이도영

본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 정보통신·방송 연구개발사업의 일환으로 수행하였음. [IITP-2018-0-00749, 인공지능 기반 가상 네트워크 관리기술 개발]

- 문서 변경 기록

날짜	버전	변 경 내 역 설 명	작성자
2021.06.28	v1.0	초안 작성	이도영
2021.07.06	v1.1	오타 수정 및 내용 보완	이도영
2021.07.19	v.1.2	오타 수정 및 내용 보완	이도영

목 차

요 약 문

오늘날 5G 네트워크 시대가 도래하면서, 급변하는 서비스 요구사항을 만족시키기 위해 유연하고 민첩하게 네트워크를 구축하고 관리하는 것이 요구되고 있다. 소프트웨어 정의 네트워킹 (SDN, Software-Defined Networking)과 네트워크 기능 가상화 (NFV, Network Function Virtualization)는 네트워크를 소프트웨어 기반으로 전환하여 유연한 네트워크 관리를 가능케 하는 핵심 기술들이다. 특히, NFV는 네트워크 기능들을 소프트웨어 형태로 가상화해 상용 서버에서 운영하며, 동적으로 네트워크 기능을 관리할 수 있는 장점이 있다. 하지만, 한편으로는 물리 네트워크 자원 뿐 아니라 수많은 가상 네트워크 및 자원들로 인해 네트워크 관리를 복잡하게 만드는 원인이 된다. 이를 해결하기 위해 최근에는 인공지능 기술을 도입하여 복잡한 NFV 환경 및 기술들을 관리하는 연구가 주목받고 있다. 특히, 서비스 펑션 체이닝 (SFC, Service Function Chaining)은 필수 NFV 기술 중 하나로, 효율적인 SFC를 구성하는 것이 요구된다.

본 문서에서는 인공지능 기술 중 하나인 강화학습 (RL, Reinforcement Learning)을 활용해 최적 SFC를 구성하는 방법을 서술한다. 구체적으로는, 심층 강화학습 알고리즘 중 하나인 Deep Q-network (DQN)를 통해 SFC 구성 문제를 정의하고 해결한다. 검증을 위해 OpenStack을 활용해 테스트베드로 구축한 후, 제안하는 강화학습 기반 SFC 구성 알고리즘을 통해 테스트베드 내 동작하는 VNF 인스턴스들을 선택하거나 새로운 VNF 인스턴스를 생성하여 SFC를 구성한다. 또한, 제안하는 DQN 기반 SFC 구성 알고리즘을 OpenStack에서 사용할 수 있도록 모듈 형태로 개발하였으며, 이를 사용하기 위한 상세한 설명도 포함한다.

1 OpenStack 내 Service Function Chaining 생성 방법

1.1 Service Function Chaining 개요

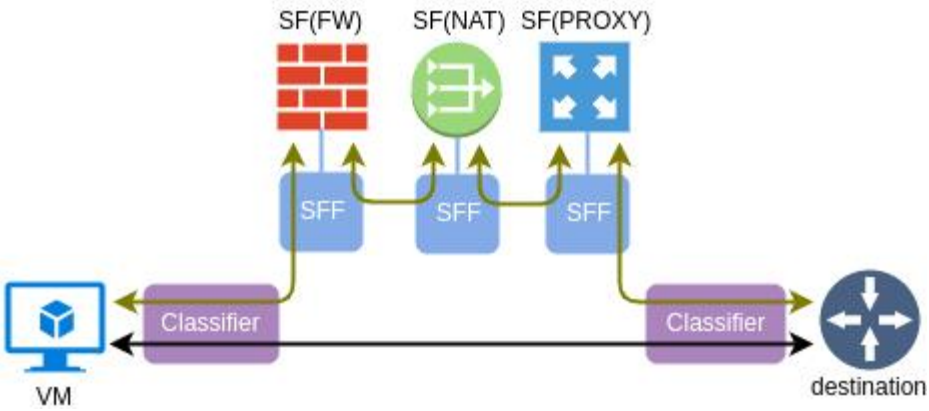
네트워크 기능 가상화 (NFV, Network Function Virtualization)는 전용 하드웨어를 통해 제공되는 네트워크 기능을 소프트웨어 형태로 가상화하여 상용 서버에서 운영하는 기술이다. NFV는 서비스 요구사항에 따라 동적으로 네트워크 기능을 VNF (Virtual Network Function) 형태로 배치할 수 있지만, 한편으로는 수많은 가상 네트워크와 자원들을 생성하여 네트워크 관리를 복잡하게 만드는 원인이 된다. 복잡해지는 네트워크 관리 문제를 해결하기 위해 최근에는 인공지능 기술을 네트워크 관리에 접목하려는 연구가 주목받고 있다. 오늘날에는 다양한 인공지능 기술들을 VNF 배치 (VNF Deployment), 서비스 펑션 체이닝 (SFC, Service Function Chaining), 오토 스케일링 (Auto-Scaling), 마이그레이션 (Migration) 등 전반적인 VNF 관리 기술에 적용하기 위한 연구가 진행되고 있다. 그중에서도 SFC는 네트워크 트래픽에 일련의 네트워크 기능을 적용할 수 있도록 트래픽 경로를 제어해 VNF를 경유시키는 기술로써, QoS를 보장하는 최적의 SFC를 구성하는 것이 요구된다.

1.2 OpenStack Networking-SFC

NFV 환경의 VNF 라이프 사이클 (Life-cycle) 관리 기능 중 필수 기능인 SFC는 네트워크에서 흐르는 트래픽에 일련의 네트워크 기능을 적용하는 기술이다. 본 문서에서는 NFV 환경을 OpenStack으로 구축한 후, SFC를 구성하는 방법에 대해 서술한다. OpenStack에서는 SFC 구성을 위해 크게 OpenStack Networking-SFC 플러그인을 활용하는 방법과 OpenStack Tacker를 활용하는 방법 두 가지가 존재한다. 본 문서에서는 OpenStack Networking-SFC 플러그인을 통해 SFC를 구성하는 방법에 대해 다룬다.

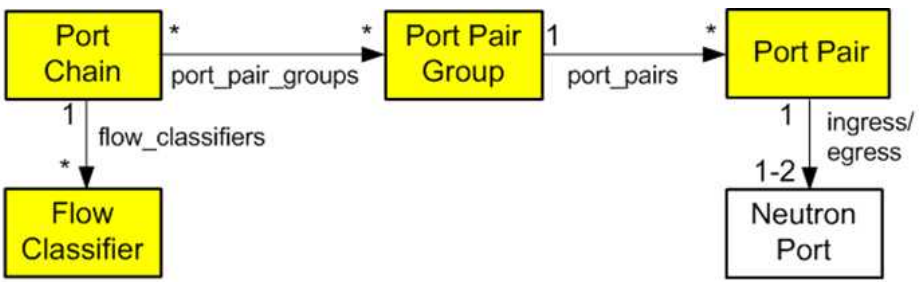
OpenStack Networking-SFC는 OpenStack VNF 인스턴스들의 네트워크 포트 (Neutron Port)를 논리적으로 연결 지은 후, 특정 네트워크 트래픽을 해당 포트들로 통과시키는 방식으로 SFC를 구현하는 OpenStack의 공식 프로젝트이다 [1]. OpenStack Networking-SFC를 활용하여 SFC를 생성할 경우, (그림 1)과 같이 주요 구성요소인 Classifier, Service Function (SF), Service Function Forwarder (SFF)들을 포함한다. 이들 중 Classifier는 특정 네트워크 트래픽을 식별하여 대응하는 SFC로 트래픽을 우회시키는 역할을 수행한다. OpenStack 기반 테스트베드에서 Classifier는 OpenStack 컨트롤러 또는 컴퓨트 노드에 위치하는 가상 스위치인 Open vSwitch (OVS)에서 동작하며, 특정 네트워크 패킷들을 매치시켜 태깅 (tagging)하기 위한 플로우 룰로 구현된다. 본 문서에서 서술하는 OpenStack 테스트베드는 Rocky 버전으로 설치되었으며, 해당 버전의 Networking-SFC는 식별된 플로우의 패킷들에 MPLS (Multi-Protocol Label Switching) 값을 태깅하는 방식으로 동작한다. SF는 입력 받는 패킷에 대해 특정 네트워크 기능 (예, 방화벽, NAT 등)을 수행하는 VNF 인스턴스를 의미한다. VNF 인스턴스는 가상 머신 (VM, Virtual Machine)이나 컨테이너 (Container) 형태로 생성될 수 있

으며, 본 문서에서는 VM의 경우만 고려한다. SFF는 OpenStack 컨트롤러 또는 컴퓨트 노드에 위치하여 VM을 연결하는 OVS 가상 스위치에 해당되며, 해당 스위치는 수신 패킷의 MPLS 값을 확인 후 패킷이 대응하는 SFC를 통과하도록 다음 위치 (VM 또는 다른 OVS)로 전달한다.



(그림 1) SFC 구성의 예

SFC 생성을 위한 Networking-SFC의 구성 요소를 보여주는 (그림 2)에서, Neutron Port는 SF에 해당하는 VM의 네트워크 포트를 의미한다. Neutron은 각 네트워크 포트들에게 ID를 부여하고 OpenStack의 자원으로 관리한다. Port Pair는 SF에 이러한 포트가 1개 이상 존재할 때 패킷의 입력 및 출력 포트를 각기 정의한다. Port Pair Group은 동일한 유형의 SF 인스턴스 (예, 방화벽 기능 목적의 VM)가 다수 존재할 때 이들의 Port Pair를 그룹 단위로 묶어서 로드밸런싱 (Load-balancing) 할 수 있게 한다. Flow Classifier는 SFC의 Classifier 역할을 수행하고, OpenStack 내부에서 SFC의 적용 시작 지점을 지정하며 네트워크 플로우를 식별하기 위한 매치 필드를 설정한다. Port Chain은 Flow Classifier와 Port Pair Group을 연관지어 OpenStack 내부에서 동작하는 SFC를 생성한다.



(그림 2) Networking-SFC 구성요소

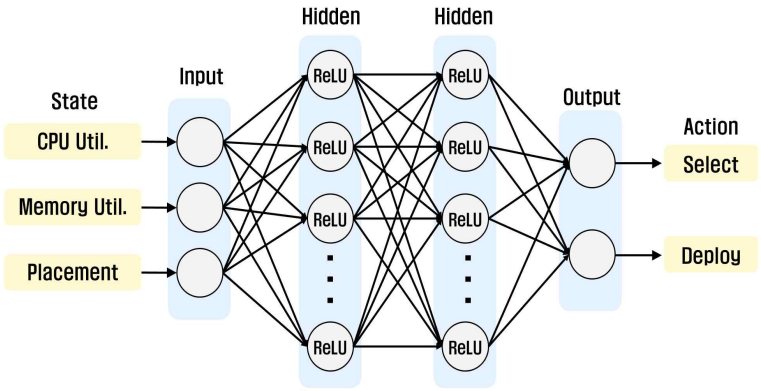
본 문서에서는 단일 VM에 단일 VNF를 설치한 후, Networking-SFC 플러그인을 활용하여 이들 VM들 간의 포트를 연결해 SFC를 생성한다.

2 Deep Q-network 기반 SFC 구성 방법

2.1 최적 SFC 구성을 위한 강화학습 문제 정의

강화학습은 인공지능 기술 중 하나로, 최대의 누적 보상을 주는 행동 (Action)들을 수행할 수 있도록 시행착오를 거쳐 최적의 정책 (Policy)을 찾는 학습 방법이다. 일반적으로 강화학습은 에이전트 (Agent)와 환경 (Environment)으로 구성되며, 에이전트는 정책에 따라 현재 상태 (State)에서 특정 행동을 수행하게 된다. 이를 통해 다음 상태로 이동하게 되며, 동시에 보상을 얻게 된다. 강화학습의 목적은 각 상태에서 다양한 행동을 수행하며 보상을 최대화하는 정책을 찾는 것이기 때문에 상태와 행동, 그에 대한 보상의 정의가 필요하다.

본 문서에서 서술할 DQN (Deep Q-network) 기반 SFC 구성 방법은 클라우드 컴퓨팅 환경의 제한된 가용 자원을 고려하여 서비스 성능 목표를 만족하는 SFC를 구성한다. DQN은 심층 강화학습 방법 중 하나로, 인공 신경망 (Neural network)을 통해 수많은 상태 (State)가 존재하는 최적화 문제를 설계할 수 있다. (그림 3)은 SFC 구성을 위한 DQN 모델을 나타낸다. DQN 모델은 SFC를 구성하는 VNF의 종류와 순서가 명시된 SFC 요청이 있을 때, VNF 인스턴스들을 선택 또는 생성하는 과정을 반복하여 SFC를 구성한다. 즉, 이미 배치된 VNF 인스턴스를 활용하여 성능 목표를 달성할 수 있다면, 해당 인스턴스로 SFC를 구성하기 때문에 VNF 생성 비용이 들지 않아 제한된 자원 환경 속에서 효율적으로 SFC를 생성한다. DQN 모델은 2개의 은닉층을 가지며, 활성화 함수 (Activation function)로 ReLU (Rectified Linear Unit)를 사용한다.



(그림 3) Deep Q-network (DQN) 기반 SFC 구성 모델

DQN 모델의 입력 계층에서는 SFC를 구성할 VNF 인스턴스들의 평균 CPU 사용률 (CPU Util.), 메모리 사용률 (Memory Util.), 배치 상태 (Placement)로 구성된 현재 상태를 입력값으로 받는다. CPU와 메모리는 VNF 인스턴스에서 패킷 처리에 사용되는 자원들이며, 배치 위치는 SFC에서 처리되는 패킷들의 경로로 인한 전파 지연 시간 (Propagation delay)에 영향을 미친다. VNF 인스턴스들의 배치 상태는 이전 순서에 선택된 VNF 인스턴스로부터 다음 순서의

VNF 인스턴스들 사이의 평균 거리를 Hop으로 계산한 값이다. 따라서 DQN 모델의 상태는 선택 대상이 되는 인스턴스들의 패킷 처리 성능과 위치를 고려하여 SFC를 구성한다.

제안하는 모델의 출력 계층에서는 SFC를 구성할 VNF를 기존에 존재하는 VNF 인스턴스들 중 선택할지 (Select), 또는 새로운 VNF 인스턴스를 생성할지 (Deploy)에 대한 행동 값 (Action value)을 출력한다. SFC 구성을 위한 VNF 인스턴스를 선택하는 행동을 결정했을 경우, CPU와 메모리 사용률이 작고, 이전 순서에 선택된 VNF 인스턴스와 가까운 인스턴스를 선택한다. 반면, VNF 인스턴스를 생성하는 행동을 결정했을 때는 이전 순서에 선택된 VNF 인스턴스와 가까운 서버들 중, 가용 자원이 충분한 위치를 선택하여 새로운 인스턴스를 생성한다. SFC를 구성할 때, 기존에 존재하는 VNF 인스턴스들 중 하나를 선택하는 것과 새로운 VNF 인스턴스를 생성하는 것은 SFC 성능과 구성 비용 측면에서 장단점이 존재하기 때문에 이를 고려한 행동 결정이 필요하다. 따라서 본 문서의 DQN 모델은 수식 (1)과 같은 보상 모델을 사용한다.

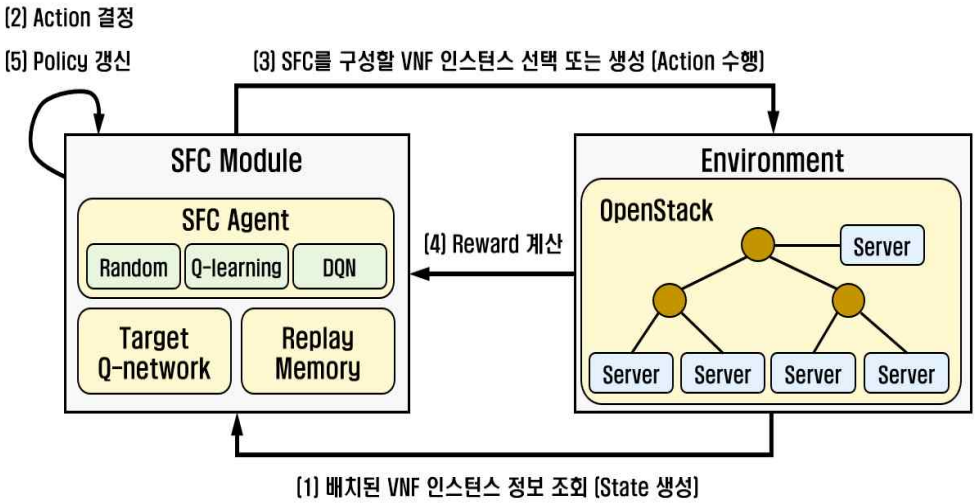
$$Reward = -\ln(1 + resTime) \times Cost_{deploy} \qquad (1)$$

DQN 모델은 SFC 구성을 위한 최적 정책 (Policy)을 학습하기 위해 행동을 결정할 때마다 그에 대한 보상 (Reward)을 얻는다. 제안하는 방법의 목표는 적은 자원과 비용으로 서비스 성능 요구사항을 만족하는 SFC를 구성하는 것이다. 네트워크 서비스에서 성능을 측정하기 위한 주요 지표 중 하나는 응답 시간이기 때문에 수식 (1)에서는 응답 시간 (resTime)을 고려한다. 응답 시간은 사용자가 생성한 패킷이 VNF 인스턴스에 의해 처리된 후, 그에 대한 응답을 사용자가 받기까지 소요된 시간이다. 제안하는 방법에서 DQN 모델의 행동은 다음 순서의 VNF 인스턴스를 선택하는 것이기 때문에, 보상 계산을 위한 응답 시간은 이전 순서에 선택된 VNF 인스턴스로부터 새롭게 선택 또는 생성된 VNF 인스턴스로 Ping 메시지를 전달하여 millisecond 단위로 측정한다. 또한, 인스턴스 생성에 요구되는 비용은 가중치 ($Cost_{deploy}$)를 통해 측정된 서비스 성능, 즉 응답 시간을 보정한다. 예를 들어, VNF 인스턴스를 선택했을 때와 생성했을 때 동일한 응답 시간이 측정되었을 경우, 인스턴스를 생성했을 때 받는 보상은 가중치($Cost_{deploy}$)로 1.2를 곱하여 감소시킨다. 이와 같은 보상 모델을 통하여 SFC 구성을 위한 서비스 성능과 구성 비용을 함께 고려할 수 있다. 마지막으로, 본 문서의 DQN 모델 학습 과정 중, 특정 상황에서 수행할 행동을 선택할 때는 ε-greedy 알고리즘을 활용한다. 마지막으로, 본 문서의 DQN 기반 SFC 구성과 관련된 내용은 참고문헌 [2]에 자세히 서술되어 있다.

2.2 Deep Q-network 기반 SFC 구성 모듈 구현

본 문서의 DQN 기반 SFC 구성 기능은 OpenStack 환경에서 실제 동작할 수 있는 모듈 (Module) 형태로 구현되었다. (그림 4)는 SFC 경로 선택 모듈 (이하 SFC 모듈)의 구조도를 보인다. OpenStack 환경에는 다수의 VNF 인스턴스들이 설치되어 있고, Networking-SFC 플러그

인을 통해 이들을 연결시켜 SFC를 생성할 수 있다. 이 때, SFC에 포함되는 VNF들의 종류와 순서는 달라질 수 있기 때문에 SFC 요청 (Request)에 SFC를 구성하는 VNF 종류와 순서가 명시된다.



(그림 4) SFC 모듈 구조도

SFC 모듈은 SFC 구성 요청 메시지를 받았을 때, 해당 SFC를 구성하기 위해 환경 (Environment)에 배치된 VNF 인스턴스 정보를 조회한다. 그리고, SFC 구성을 위한 행동을 결정하여 VNF 인스턴스를 SFC에 포함시키고 행동에 대한 보상을 받는다. SFC 구성이 완료될 때까지 VNF 인스턴스 선택 또는 생성을 반복하며, 주기적으로 SFC 구성 정책을 갱신한다. 또한, SFC 모듈이 DQN 알고리즘을 사용할 때, 안정적인 학습을 위해 [3]에서 제안한 Replay memory와 Target Q-network를 활용한다. Replay memory와 Target Q-network는 강화학습 알고리즘의 학습 과정에서 순차적인 행동과 그에 대한 보상 값으로 정책을 갱신할 경우 각 행동들의 상관 관계 (Correlation)로 인해 최적 정책을 찾지 못하는 문제를 해결한다. 또한, 구현된 SFC 모듈은 DQN 기반 SFC 구성 기능 뿐 아니라, Random 알고리즘, Q-learning을 통한 SFC 구성 기능을 제공한다 [4, 5]. 구현된 SFC 모듈은 GitHub을 통해 공개되어있다 [6].

3 Deep Q-network 기반 SFC 구성 모듈 사용법

3.1 환경 구축

본 문서에서 다루는 DQN 기반 SFC 구성 방법은 OpenStack 환경에서 동작할 수 있는 형태로 구현되었다. 앞서 서술한 것처럼, 구현된 SFC 모듈은 SFC 생성을 위해 OpenStack 테스트베드 내 존재하는 각 VNF 인스턴스들의 데이터를 활용하고, SFC 모듈에서 DQN, Q-learning 또는 Random 알고리즘을 통해 결정된 SFC 경로는 실제 OpenStack 테스트베드 내 설치되어야 한다. 이를 위해 OpenStack 기반 환경 뿐만 아니라, SFC 모듈과 상호 연동되는 NFVO와 모니터링 모듈의 설치가 선행되어야 한다. OpenStack 환경 구축을 위한 스크립트 파일 및 설명서 [7]와 NFVO 및 모니터링 모듈 설치를 위한 코드, 설명서 [8]는 Github를 통해 공개되어 있기 때문에 본 문서에서 해당 요소들의 설치 과정은 생략한다. OpenStack 환경이 구축되고, NFVO와 모니터링 모듈을 설치한 후에는 SFC 모듈을 설치해서 실행할 수 있다. 본 문서에서 모듈 설치와 실행은 Ubuntu 16.04에서 수행하였다.

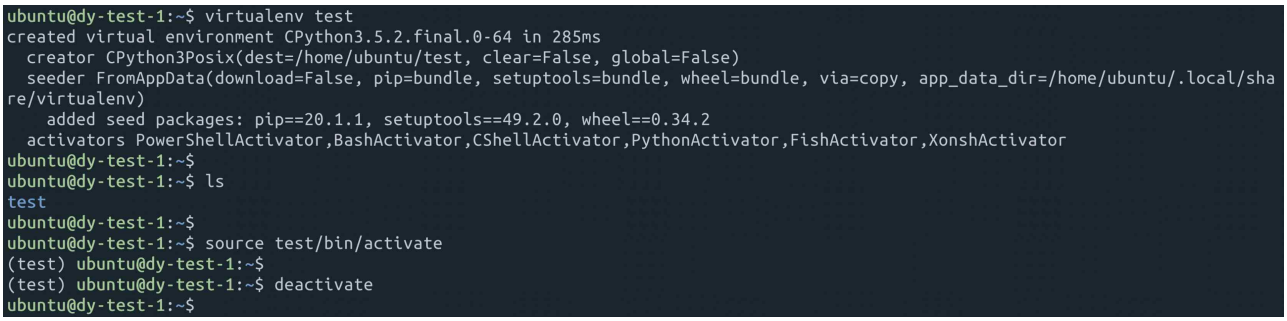
SFC 모듈은 Python으로 작성되었으며, Python v3.6.5 이상에서 실행하는 것을 권장한다. SFC 모듈을 설치하고 실행시키기 위해서는 관련 패키지들을 미리 포함하고 설치해야 한다. Python 환경에서는 패키지의 버전 문제에 따른 호환 문제가 쉽게 발생할 수 있으므로, 다른 기능들과 충돌을 막기 위해, 가상 환경에서 SFC 모듈을 설치한다. 가상 환경 생성은 virtualenv를 사용한다. 아래 명령어들을 통해 Python3 패키지 설치 도구인 pip3를 먼저 설치한 후, 이를 통해 virtualenv까지 설치할 수 있다. virtualenv를 설치한 후에는 임의의 가상 환경 이름을 갖는 독립된 환경을 구축할 수 있다. 가상 환경에서 포함되고 설치되는 Python 관련 패키지들은 다른 환경과는 서로 간섭이 되지 않아 SFC 모듈을 위한 패키지를 안정적으로 불러오고 설치할 수 있다.

```
# Python 패키지 설치 도구 pip3 설치 후 가상환경 구성을 위한 virtualenv 설치
sudo apt-get update
sudo apt-get install python3-pip
pip3 install virtualenv

# virtualenv를 통한 가상 환경 생성 후 활성화
# test 대신 원하는 환경 명 지정 가능
virtualenv test
source test/bin/activate

# 가상 환경 비활성화
deactivate
```

virtualenv 명령어를 통해 가상 환경을 생성한 후에 활성화하면 CLI 계정 명 왼쪽에 현재 활성화된 가상 환경 명이 표시된다. (그림 5)은 test라는 가상 환경을 생성하고 활성화한 후에, 비활성화까지 진행한 화면을 보인다.



(그림 5) virtualenv를 통한 가상 환경 생성

3.2 SFC 구성 모듈 설치 및 실행

앞선 과정에서 pip3 설치까지 완료하고 가상 환경까지 활성화한 후에 SFC 모듈 설치를 진행한다. Github [6]에 공개 되어있는 SFC 모듈을 내려 받은 후, 모듈 실행에 필요한 패키지를 설치한다. SFC 모듈을 위한 패키지는 requirements.txt 파일에 정리되어 있으며, 이를 사용하여 패키지를 설치한다. 패키지를 설치한 후에는 Python 명령어를 통해 SFC 모듈을 실행한다. 아래 명령어를 순서대로 입력하면 SFC 모듈 내려 받기, 필수 패키지 설치, 모듈 실행까지 수행할 수 있다.

```
# SFC 모듈 다운로드 후 필수 패키지 설치
git clone https://github.com/dpnm-ni/ni-sfc-path-module-public
cd ni-sfc-path-module-public
pip3 install -r requirements.txt

# SFC 모듈 실행
python3 -m server
```

SFC 모듈 실행 명령어를 입력하면 Flask 기반으로 웹 서버가 동작한다. SFC 모듈은 SFC 요청 메시지를 통해 DQN, Q-learning, Random 알고리즘을 활용하여 SFC 경로를 결정하기 때문에 웹 서버로 동작하면서 메시지를 처리한다. SFC 모듈에서 사용하는 기본 웹 서버 포트는 8001이며, 포트 변경을 원할 경우에는 내려 받은 SFC 모듈 폴더의 server/__main__.py 파일에서 (그림 6)과 같이 port라고 명시된 부분을 수정한다.

```python
def main():
    app = connexion.App(__name__, specification_dir='./swagger/')
    app.app.json_encoder = encoder.JSONEncoder
    app.add_api('swagger.yaml', arguments={'title': 'NI SFC Sub-Module Service'})
    app.run(port=8001)
```

(그림 6) SFC 모듈 server/__main__.py 파일 - Port 설정 가능

SFC 모듈은 SFC 구성을 위해 NFVO 및 모니터링 모듈과 상호작용한다. 이를 위해 설정 파일에 각 모듈이 동작하는 Host의 IP 주소와 포트 번호를 미리 입력해야 한다. 또한, DQN 기반 SFC 구성을 위해 필요로 하는 파라미터 (Parameter)를 미리 정의할 필요가 있다. 내려받은 SFC 모듈 폴더 내에서 config/config.yaml을 열면 (그림 7)처럼 각 파라미터 값을 설정할 수 있다. 모니터링 모듈에 해당하는 ni_mon, NFVO에 해당하는 ni_nfvo의 host 정보를 각각 수정하고 저장한다. 또한, SFC 모듈 폴더 내 README.md 문서를 참조하여 파라미터 값들을 설정한다.

```yaml
ni_mon:
  host: http://<ni_mon_ip>:<ni_mon_port>      # Configure here to interact with a monitoring module
ni_nfvo:
  host: http://<ni_nfvo_ip>:<ni_nfvo_port>    # Configure here to interact with an NFVO module
instance:
  monitor: <IP of a monitoring instance>      # IP of traffic generator (i.e., VM instance that has a public IP address)
  id: <ssh_id of a monitoring instance>       # SSH ID of new VNF instance
  password: <ssh_id of a monitoring instance> # SSH ID of new VNF instance
image:                                        # Image IDs used by OpenStack
  firewall: <OpenStack Image ID>
  flowmonitor: <OpenStack Image ID>
  dpi: <OpenStack Image ID>
  ids: <OpenStack Image ID>
  proxy: <OpenStack Image ID>
```

(그림 7) SFC 모듈 config/config.yaml 파일 설정

DQN을 수행하기 위해서는 사용자에 의해 하이퍼파라미터 값들이 설정되어야 한다. 또한, OpenStack 환경에서는 SFC를 구성하는 각 VNF 인스턴스들이 1개 이상의 네트워크 포트를 가질 수 있기 때문에 어떤 포트를 활용하여 SFC를 생성할 것인지 미리 명시해야 한다. (그림 8)처럼 DQN을 위한 하이퍼파라미터 값과 SFC 생성에 사용할 네트워크 포트 식별을 위한 OpenStack Network ID는 내려받은 SFC 모듈 폴더 내에서 sfc_using_dqn.py 파일을 열어 수정한다. OpenStack Network ID는 OpenStack Dashboard 또는 OpenStack CLI를 통해 확인 가능하며, 해당 네트워크에 속한 포트들을 식별하여 SFC 생성에 활용한다.

```
# Parameters
# OpenStack Parameters
openstack_network_id = "9cdee37a-fdcd-45f5-860b-253fc62ce578" # Insert OpenStack Network ID to be used for creating SFC

# <Important!!!!> parameters for Reinforcement Learning (DQN in this codes)
learning_rate = 0.01            # Learning rate
gamma         = 0.98            # Discount factor
buffer_limit  = 10000           # Maximum Buffer size
batch_size    = 16              # Batch size for mini-batch sampling
num_neurons = 64                # Number of neurons in each hidden layer
epsilon = 0.99                  # epsilon value of e-greedy algorithm
required_mem_size = 24          # Minimum number triggering sampling
print_interval = 24             # Number of iteration to print result during DQN
```

(그림 8) DQN의 Parameter 및 OpenStack Network ID 설정

설정을 마무리하고 실행 명령어를 입력하여, 정상적으로 SFC 모듈이 실행되면 (그림 9)와 같은 출력 화면을 볼 수 있다. 본 문서에서는 기본 포트 8001을 변경 없이 사용한다.

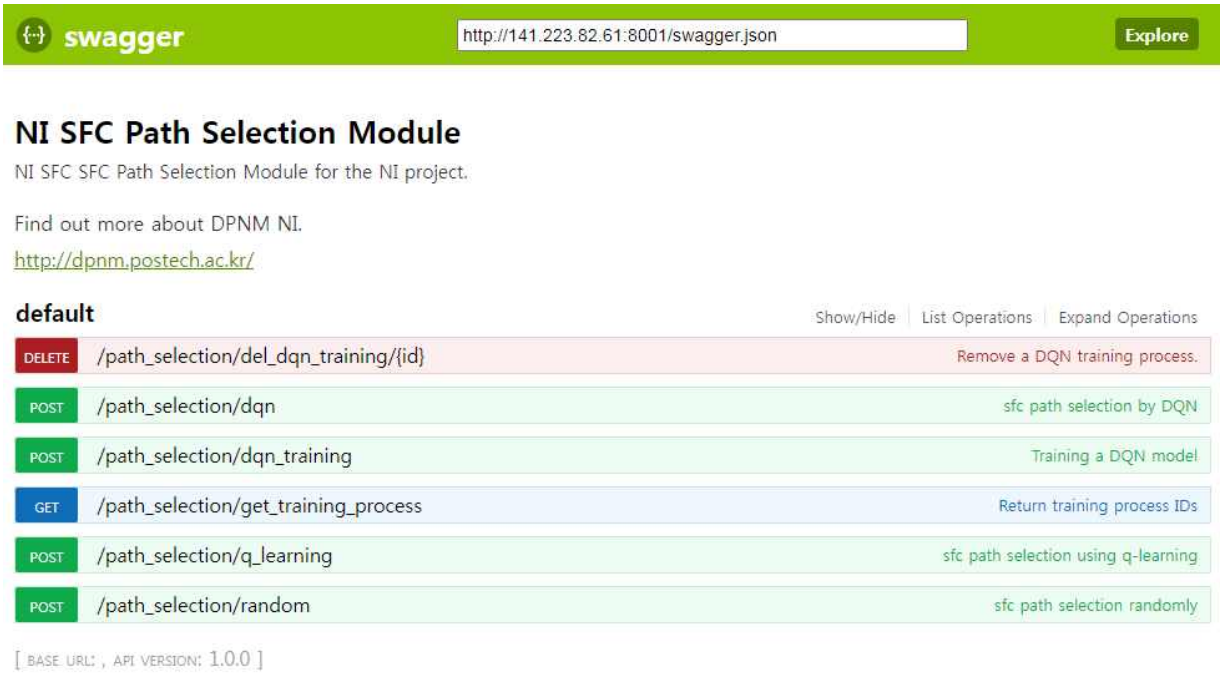
```
(test) ubuntu@dy-test-1:~/ni-sfc-path-module-public$ python3 -m server
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8001/ (Press CTRL+C to quit)
```

(그림 9) SFC 모듈 실행 화면

3.3 SFC 모듈 사용법

SFC 모듈이 정상적으로 실행되고, 설정에 오류가 없을 경우, SFC 모듈은 (그림 10)처럼 제공되는 Web UI를 아래 경로를 통해 확인할 수 있다.

http://<SFC 모듈이 실행되는 Host IP>:<모듈 포트 번호>/ui



(그림 10) SFC 모듈 실행 화면

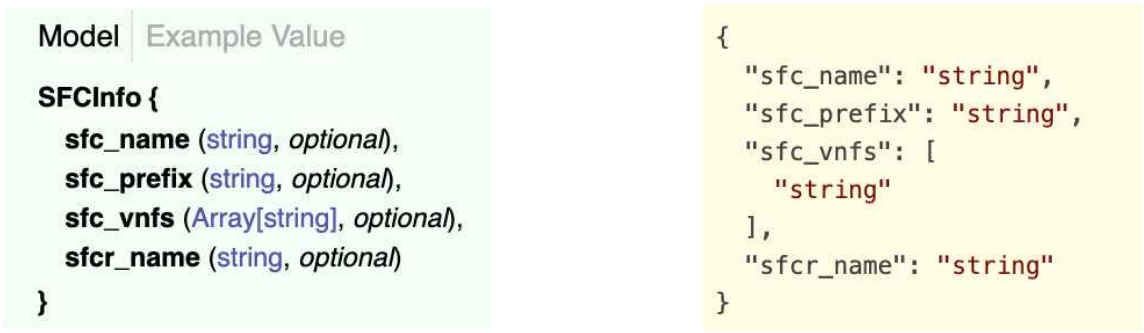
SFC 모듈은 HTTP POST/GET/DELETE 메시지를 통해 SFC 구성과 관련된 기능들을 실행시킨다. SFC 구성을 위한 메시지는 Web UI를 통해 직접 SFC 모듈로 요청하거나, HTTP 메시지를 생성하여 SFC 모듈로 전달하는 방법 두 가지가 존재한다. 본 문서에서는 SFC 모듈의 Web UI를 통해 SFC를 구성하는 방법에 대해 서술한다. SFC 모듈은 HTTP 메시지를 수신하는 경로를 다음과 같이 제공한다.

(1) DQN Training 프로세스 종료: http://<SFC 모듈 Host>:<포트>/path_selection/del_dqn_training/{id}
(2) DQN 기반 SFC 구성: http://<SFC 모듈 Host>:<포트>/path_selection/random
(3) DQN Training 프로세스 실행: http://<SFC 모듈 Host>:<포트>/path_selection/dqn_training
(4) DQN Training 프로세스 조회: http://<SFC 모듈 Host>:<포트>/path_selection/get_training_process
(5) Q-learning 기반 SFC 구성: http://<SFC 모듈 Host>:<포트>/path_selection/q_learning
(6) Random 기반 SFC 구성: http://<SFC 모듈 Host>:<포트>/path_selection/random

DQN을 활용하여 SFC를 구성할 때에는 SFC를 구성하는 VNF 인스턴스들의 선택 또는 생성을 결정할 DQN 모델이 필요하다. 이를 위해, (3)은 DQN 모델을 생성하고 학습시키는 프로세스를 실행하는 기능을 제공한다. 또한, DQN 모델의 학습을 충분히 수행한 후에는 (1)을 통해 학습을 중단시킬 수 있으며, 현재 학습이 되고 있는 DQN 모델들은 (4)를 통해 조회할 수 있다. 학습이 중단될 때, DQN 모델은 폴더 내 /dqn_models에 생성된다. DQN 모델이 준비된 후에는 (2)를 통해 특정 DQN 모델을 사용해 SFC를 구성할 수 있다. 이 외에도 (5), (6)을 통해 Q-learning 또는 Random 알고리즘을 통해 SFC를 구성할 수도 있다.

(2), (5), (6)은 HTTP POST 메시지의 Body에 명시된 데이터를 활용하여 SFC를 구성하는 기능들이다. (그림 11)처럼 SFC 요청 메시지의 Body에는 SFCInfo라고 정의된 모델이 JSON으로

표현된다. SFCInfo 모델은 총 4개의 string 형태의 데이터로 구성된다. (그림 11)에서 왼쪽은 SFCInfo 모델의 정의를 보이며, 오른쪽은 SFC 요청 메시지에 각 데이터들이 포함될 때 어떤 형식으로 기입되어야 하는지 보이는 예시이다. 요청 메시지에 명시할 때는 각 데이터의 이름과 값들을 “” 내에 기입한다.



(그림 11) SFC 생성 요청 메시지 Body 모델 및 예

SFCInfo에 명시되는 데이터의 각 의미는 아래와 같다.

(1) sfc_name: OpenStack에서 생성되는 SFC 이름이자 DQN 모델 이름
(2) sfc_prefix: 생성되는 SFC에 포함되는 각 VNF 인스턴스 이름을 식별할 접두사
(3) sfc_vnfs: 생성되는 SFC에 포함되는 Classifier, VNF 인스턴스들의 이름
(4) sfcr_name: OpenStack에서 생성되는 Classifier 이름

SFC 모듈을 사용하지 않더라도 OpenStack Networking-SFC 플러그인은 기본적으로 OpenStack Dashboard 또는 OpenStack CLI를 통해 SFC를 생성할 수 있는 API들을 제공하고 있기 때문에 이를 활용해 SFC를 생성할 수 있다. 하지만 해당 방법들은 SFC 생성을 위해 SFC를 구성하는 각 VNF 인스턴스의 네트워크 포트 ID를 미리 알고 있어야 하며, 생성된 SFC로 흘러야 하는 트래픽을 분류하는 Classifier의 네트워크 포트 IP 주소를 수동으로 설정해야 한다는 불편함이 존재한다. 따라서 본 문서의 SFC 모듈은 사람이 기억하고 이해하기 힘든 네트워크 포트 ID 또는 IP보다는 Classifier 역할을 수행하는 VM과 SFC를 구성하는 각 VNF 인스턴스의 이름을 입력하면, NFVO 및 모니터링 모듈과 상호작용하여 자동으로 SFC를 생성해준다.

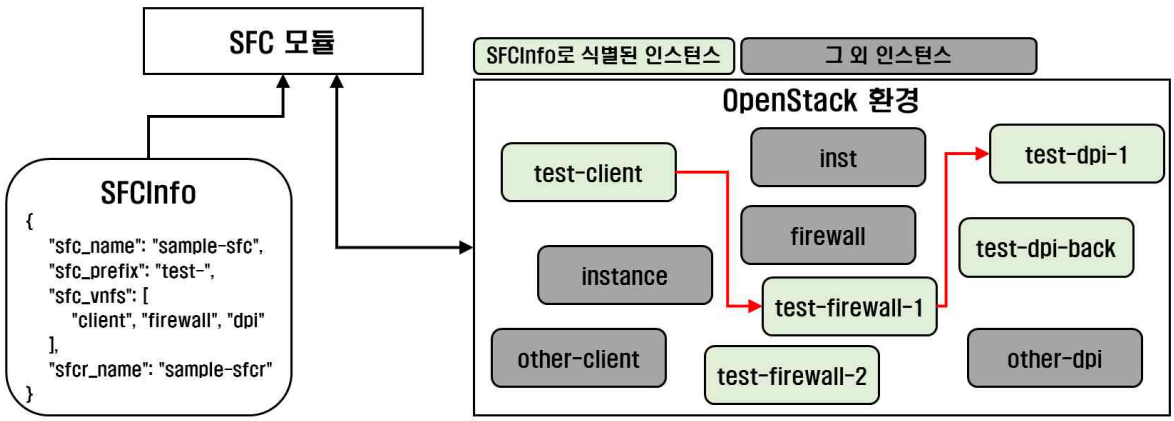
(예시)는 SFC 생성 요청 메시지에 명시되는 각 데이터의 예를 보인다. (예시)에서는 Firewall, DPI 기능들이 순서대로 연결되는 SFC의 생성을 요청하는 메시지이다. sfc_name과 sfcr_name에는 SFC를 생성한 후에 이를 식별하기 위한 SFC 이름과 SFCR (Classifier)의 이름을 사용자가 임의로 입력한다. 만약 DQN 모델 학습 기능을 실행시켰을 경우, 생성되는 DQN 모델의 이름은 sfc_name에 기입한 이름으로 할당된다. 또한, DQN 기반 SFC 구성 기능을 실행했을 때는 SFC 모듈 내 /dqn_models 폴더에서 sfc_name과 동일한 이름을 갖는 모델을 활용하여 SFC를 구성한다. sfc_prefix는 OpenStack 내 존재하는 다양한 인스턴스 중 SFC 생성

에 활용할 수 있는 인스턴스를 식별하는 접두사 역할을 한다. (예시)처럼 sfc_prefix를 test-로 입력하면, OpenStack 환경 내 존재하는 인스턴스 중 이름에 test-라는 접두사를 가진 인스턴스들만 고려하여 SFC 생성에 활용한다. sfc_vnfs는 SFC를 구성하는 Classifier와 VNF 인스턴스들의 이름을 식별하기 위한 데이터이다. sfc_vnfs의 첫 번째 요소는 Classifier 역할을 수행하는 인스턴스의 이름이며, 이 후 데이터들은 SFC를 구성하는 각 VNF 인스턴스들의 이름을 순서대로 나열된다. 따라서, (예시)에 의해 SFC 생성 요청이 SFC 모듈에 도착하면, 이름이 test-client, test-firewall, test-dpi로 시작되는 인스턴스들을 활용하여 SFC를 생성한다. 다만, Classifier는 특정 Classifier을 선택하며, VNF 인스턴스들은 여러 인스턴스들 중 한 개씩을 SFC 구성 알고리즘에 따라 선택 또는 생성하여 SFC를 구성한다.

```
{
  “sfc_name“: “sample-sfc“,
  “sfc_prefix“: “test-“,
  “sfc_vnfs“: [
    “client“, “firewall“, “dpi“
  ],
  “sfcr_name“: “sample-sfcr“
}
```

(예시) SFC 생성 요청 메시지 Body 예

(그림 12)은 SFC 모듈의 (예시)와 같은 Body를 포함한 SFC 생성 요청 메시지에 의한 SFC 생성 과정을 도식화한 것이다. SFCInfo에 명시된 sfc_prefix와 sfc_vnfs들을 활용하여 SFC 생성에 활용할 수 있는 인스턴스들을 우선 식별한다. 그 후에는 SFC 모듈에서 DQN, Q-learning, DQN 알고리즘 중 하나를 적용하여 SFC 경로를 결정한 후 이를 OpenStack 환경에 실제 SFC로 생성한다. (그림 12)에서 빨간 선으로 연결 된 것이 SFC 모듈에서 결정 된 SFC 경로에 따라 생성된 실제 SFC이다.



(그림 12) SFC 모듈의 SFC 생성 요청 메시지에 의한 SFC 생성

(그림 13)은 SFC 모듈을 통해 SFC 구성을 요청했을 때, SFC 모듈이 출력하는 응답 메시지이다. 정상적으로 SFC가 생성되었을 경우, 응답 코드 (Response code) 200과 함께, 응답 메시지 Body에 생성된 SFC 정보를 받을 수 있다. 그 중, SFC 구성에 사용된 Classifier와 VNF 인스턴스들의 이름은 sfc_path 항목에 나열된다. 또한, SFC 생성과 관련하여 OpenStack에서 Networking-SFC 플러그인을 통해 생성된 sfc_id와 sfcr_id가 명시된다. sfc_id와 sfcr_id는 본 문서에서 자세히 서술하지 않는다. 만약, 응답 메시지로 오류를 받았을 경우는 NFVO 및 모니터링 모듈에 문제가 발생해서 정상적으로 OpenStack 환경에 SFC를 생성하지 못했거나, SFC 생성 요청 메시지에 잘못 된 형태로 SFCInfo 값이 입력 된 경우가 대부분이니 해당 부분을 확인할 필요가 있다.

(그림 13) SFC 모듈의 SFC 생성 요청 메시지에 대한 응답 메시지

4 결론

본 문서에서는 심층 강화학습 알고리즘인 DQN을 활용하여 OpenStack 환경에서의 최적 SFC 구성 방법을 서술하였다. DQN 기반 SFC 구성 방법은 SFC 구성 요청사항을 바탕으로 SFC의 성능 목표인 서비스 시간을 만족시킴과 동시에, 구성 비용을 최소화하는 SFC를 구성한다. 하지만, DQN 기반 SFC 구성 방법이 현재 배치된 VNF 인스턴스들의 자원 사용률과 위치를 고려하여 짧은 서비스 시간을 보장하는 SFC를 구성할 수는 있지만, DQN은 VNF 인스턴스와 관련된 데이터만 사용해 학습을 수행하기 때문에, 네트워크 링크 상태, 혼잡 상황 등 네트워크 환경에 대응한 SFC 구성에는 한계가 존재한다. 따라서, 향후 연구로는 동적으로 변하는 네트워크 환경을 고려한 SFC 구성 모델을 설계할 예정이다. 또한, 본 문서의 DQN 모델은 간단한 인공 신경망으로 구성되었지만, 향후 개선될 모델에서는 순환 신경망 (RNN, Recurrent Neural Network)을 활용하여, VNF 인스턴스들의 패킷 처리 성능의 추세를 SFC 구성에 고려할 것이다.

5 참고문헌

[1] OpenStack networking-sfc, [web] https://docs.openstack.org/networking-sfc/ocata/

[2] 이도영, 유재형, 홍원기, "Deep Q-Networks 기반 Service Function Chaining 구성 연구", KNOM Conference 2021, Pohang, Korea, Apr. 29-30, 2021.

[3] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. nature, 518(7540), 529-533.

[4] 이도영, 유재형, 홍원기, "Q-Learning 기반 VNF 자원 인식 Service Function Chaining", KNOM Conference 2020, On-line KNOM Conference Venue, Korea, May 15, 2020

[5] Doyoung Lee, Jae-Hyoung Yoo, James Won-Ki Hong, "Q-learning Based Service Function Chaining Using VNF Resource-aware Reward Model", 21st Asia-Pacific Network Operations and Management Symposium (APNOMS 2020), Daegu, Korea, Sep. 23-25, 2020.

[6] DPNM NI Project Github, [web] https://github.com/dpnm-ni/ni-sfc-path-module-public

[7] DPNM NI Project Github, [web] https://github.com/dpnm-ni/ni-testbed-public

[8] DPNM NI Project Github, [web] https://github.com/dpnm-ni/ni-mano