NI 기술 문서

Q-learning을 활용한 OpenStack 환경 내 최적 SFC 경로 선택 방법

문서 ID	NI-SFC-01
문서 버전	v1.4
작성일	2020.09.02
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
2020.07.31	v1.0	초안 작성	이도영
2020.08.03	v1.1	그림 갱신	이도영
2020.08.07	v1.2	오타 수정	이도영
2020.08.29	v1.3	오타 수정	이도영
2020.09.02	v1.4	오타 수정	이도영

목 차

요 약 문

오늘 날 5G 네트워크 시대가 도래 하면서, 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 가상화하여 상용 서버에서 운영하며, 동적으로 네트워크 기능을 관리할 수 있는 장점이 있다. 하지만, 한편으로는 물리 네트워크 자원 뿐 아니라 수 많은 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 생성하는 것이 요구된다.

본 문서에서는 인공지능 기술 중 하나인 강화학습 (RL, Reinforcement Learning)을 활용해 최적 SFC 경로를 선택하는 방법을 서술한다. OpenStack을 활용해 테스트베드로 구축한 후, 제안하는 강화학습 기반 SFC 경로 선택 알고리즘을 통해 테스트베드 내 동작하는 VNF 인스턴스들을 선택하여 SFC를 생성한다. 또한, 제안하는 강화학습 기반 SFC 경로 선택 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

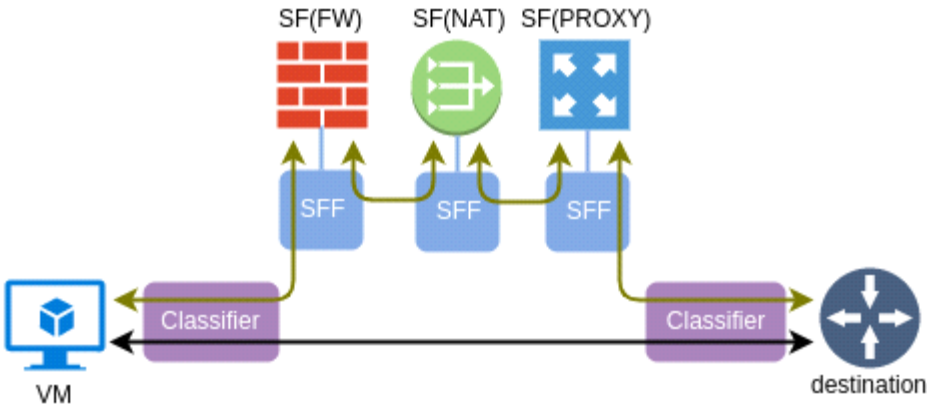
네트워크 기능 가상화 (NFV, Network Function Virtualization)는 전용 하드웨어를 통해 제공되는 네트워크 기능을 소프트웨어 형태로 가상화하여 상용 서버에서 운영하는 기술이다. NFV는 서비스 요구사항에 따라 동적으로 네트워크 기능을 VNF (Virtual Network Function) 형태로 배치할 수 있지만, 한편으로는 수 많은 가상 네트워크와 자원들을 생성하여 네트워크 관리를 복잡하게 만드는 원인이 된다. 복잡해지는 네트워크 관리 문제를 해결하기 위해 최근에는 인공지능 기술을 네트워크 관리에 접목하려는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

1.2 OpenStack Networking-SFC

NFV 환경에서 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, SFC 생성을 하는 방법에 대해 서술한다. OpenStack에서는 SFC 생성을 위해 크게 OpenStack Networking-SFC 플러그인을 활용하는 방법과 OpenStack Tacker를 활용하는 방법 두 가지가 존재한다. 본 문서에서는 OpenStack Networking-SFC 플러그인을 통해 SFC를 생성하는 방법에 대해 다룬다.

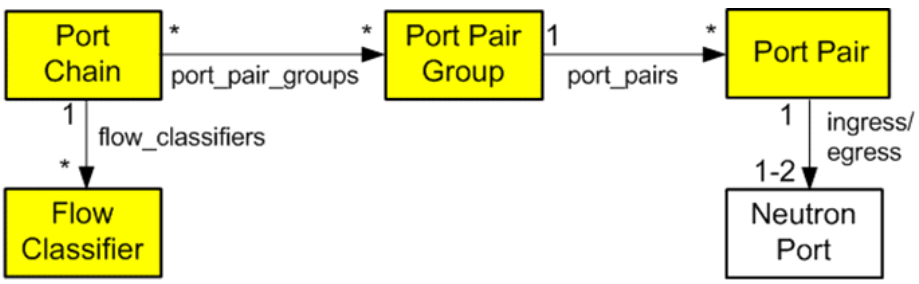
OpenStack Networking-SFC는 OpenStack VNF 인스턴스들의 네트워크 포트 (Neutron Port)를 논리적으로 연결 지은 후, 특정 네트워크 트래픽을 해당 포트들로 통과시키는 방식으로 SFC를 구현하는 OpenStack의 공식 프로젝트이다 [1]. OpenStack Networking-SFC를 활용하여 SFC를 생성할 경우, (그림 1)과 같이 주요 구성요소인 Classifier, Service Function (SF), Service Function Forwarder (SFF)들을 포함한다. 이들 중 Classifier는 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 생성 될 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 해당되는 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 로드 밸런싱 (Load-balancing) 할 수 있게 한다. Flow Classifier는 SFC의 Classifier 역할을 수행한다. 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.



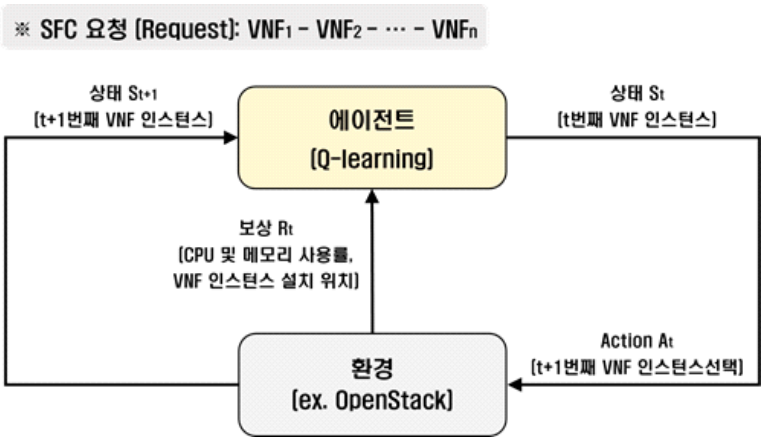
(그림 2) Networking-SFC 구성요소

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

2 Q-learning 기반 SFC 경로 선택 방법

2.1 최적 SFC 경로 선택을 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.



(그림 3) 강화학습 기반 SFC 경로 선택 모델

본 문서에서는 SFC 경로 선택 방법을 (그림 3)과 같은 강화학습 문제로 정의한다. SFC를 구성하는 각 VNF 인스턴스를 패킷이 통과해야 하는 상태로 정의하고, 다수의 VNF 인스턴스 중 어떤 것을 다음 인스턴스로 선택할지를 행동으로 정의한다. 그리고 SFC를 구성하는 VNF 인스턴스들을 순서대로 올바르게 선택했을 때 보상을 준다. 다만, SFC 경로 선택 문제에서는 단순히 VNF들을 순서대로 선택한 것만으로 보상을 주는 것이 아니라, 최적의 성능을 제공하는 인스턴스들을 선택해 SFC 경로를 구성 했는지 여부도 보상으로 고려해야 한다. 따라서 본 문서의 강화학습 기반 SFC 경로 선택 문제는 VNF 인스턴스들의 현재 CPU 사용량 및 메모리 사용량, 그리고 각 인스턴스가 설치 된 물리 서버의 위치를 고려하여 보상으로 책정한다. CPU와 메모리를 고려하는 이유는 해당 자원들이 충분치 않으면 패킷 처리가 지연 (Delay)되거나 이로 인한 패킷 손실 (Packet loss)을 발생 시키는 등, 패킷 처리 성능에 영향을 크게 미치는 요소들이기 때문이다 [2, 3]. 또한, VNF가 설치 된 물리 서버 위치는 SFC 경로를 지나는 패킷 전달 시간과 관련이 있는 요소이다. SFC를 구성하는 VNF들이 같은 물리 서버에 위치해 있을 경우, 패킷들은 다른 물리 서버로 전달 될 필요 없이 동일한 물리 서버 내에서만 이동하기 때문에 VNF 간 패킷 전달 시간이 감소한다.

다양한 강화학습 알고리즘 중, 본 문서에서는 Q-learning을 활용하여 최적 SFC 경로 선택

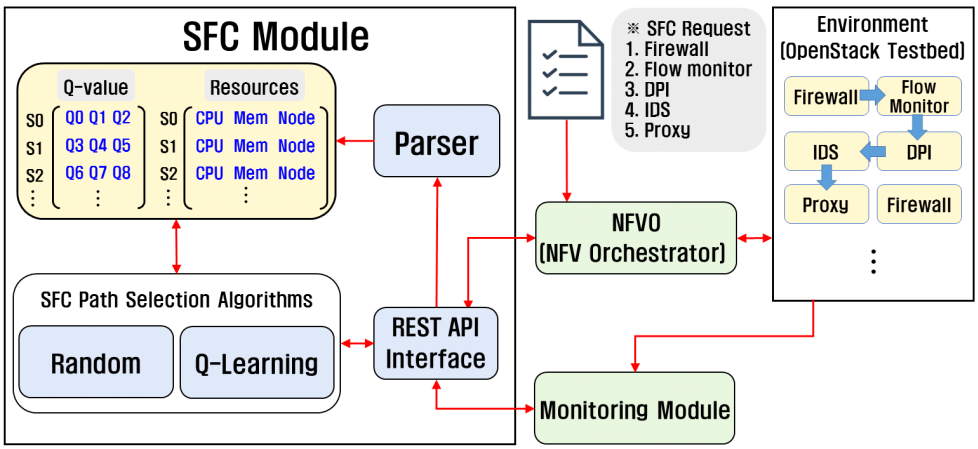
문제에 적용한다. Q-learning은 특정 상태에서 행동을 수행할 때 얻을 수 있는 보상을 예측하는 지표인 Q-value를 반복적으로 학습하는 방법이다. 이 때, 학습한 Q-value를 특정 상태에서 어떤 행동을 수행할지 결정하는 정책으로 사용 한다 (예를 들어, 특정 상태에서 Q-value가 가장 큰 행동을 선택). Q-learning의 Q-value 갱신 과정은 수식 (1)과 같다.

$$Q(S_t, A_t) = Q(S_t, A_t) + \eta(R + \gamma \max(Q(S_{t+1}, A) - Q(S_t, A_t)) \qquad (1)$$

SFC 경로 선택 문제에서 S는 패킷이 지나가는 SFC의 VNF 인스턴스를 의미하며, A는 다음 VNF로 이동하는 행동을 의미한다. R은 특정 상태에서 선택한 행동으로 인해 받을 수 있는 보상, 즉 선택된 다음 VNF의 자원 사용량과 물리 서버 위치에 따라 책정 된 보상 값을 의미한다. 이 외에 η은 학습률 (Learning rate), γ은 할인율 (Discount factor)을 의미한다. 본 문서의 Q-learning 기반 SFC 경로 선택 문제 정의는 참고문헌 [4, 5]에 자세히 서술되어 있다.

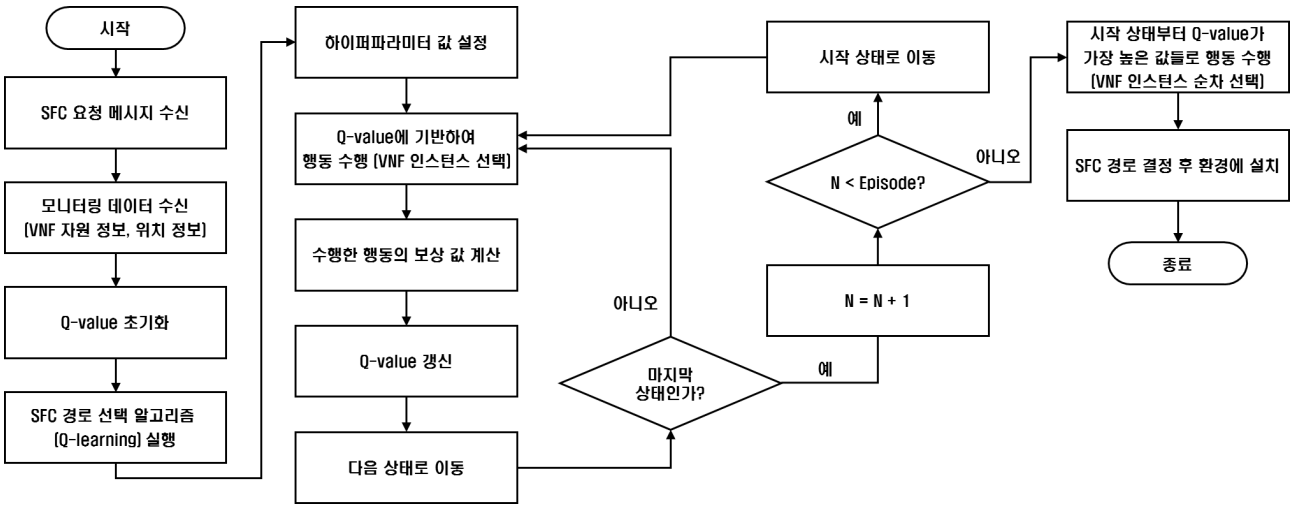
2.2 Q-learning 기반 SFC 경로 선택 모듈 구현

본 문서의 Q-learning 기반 SFC 경로 선택 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 플러그인을 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. NFVO (NFV Orchestrator)는 SFC 모듈 또는 OpenStack 환경과 상호작용하며 SFC를 생성하는 기능을 제공한다. 새로운 SFC 생성을 위한 SFC 요청이 NFVO에 도착하면, NFVO는 해당 요청 메시지를 REST API를 통해 SFC 모듈로 전달한다. 모니터링 모듈 (Monitoring Module)은 SFC 생성을 위해 SFC 모듈에서 요구하는 데이터를 제공한다.



(그림 4) SFC 모듈 구조도

SFC 요청 메시지는 SFC를 생성하는 VNF의 종류 및 순서 정보 뿐 아니라, 어떤 알고리즘을 통해 SFC를 생성할지 명시한다. 구현한 SFC 모듈은 Random과 Q-learning, 두 가지의 알고리즘을 제공한다. Random 알고리즘으로 SFC를 생성할 때에는 학습을 위한 데이터는 필요하지 않으며, 현재 OpenStack 환경에 설치되어 있는 VNF 인스턴스들 중, SFC에 포함되어야 하는 VNF 종류에 해당하는 인스턴스들 정보를 가져온다. 가져온 데이터를 바탕으로 SFC 모듈은 SFC 요청에 명시된 VNF 종류와 순서만을 고려하여 임의로 각 VNF 인스턴스들을 한 개씩 선택하여 SFC 경로를 결정한다. 반면, Q-learning 알고리즘을 적용해 SFC를 생성하면, 학습을 위해 각 VNF 인스턴스들의 자원 정보 (CPU, 메모리 사용량)와 설치 위치 (VNF 인스턴스가 실행되고 있는 물리 서버)를 추가로 가져온 후, SFC 모듈에게 전달한다. Q-learning은 Random 알고리즘과는 달리 학습을 통해 최적의 SFC 경로를 결정한다. SFC 모듈에서 각 알고리즘을 통해 결정 된 경로는 REST API를 통해 NFVO로 전달되며, NFVO는 해당 SFC 경로를 실제로 OpenStack 환경에 SFC로 생성한다.



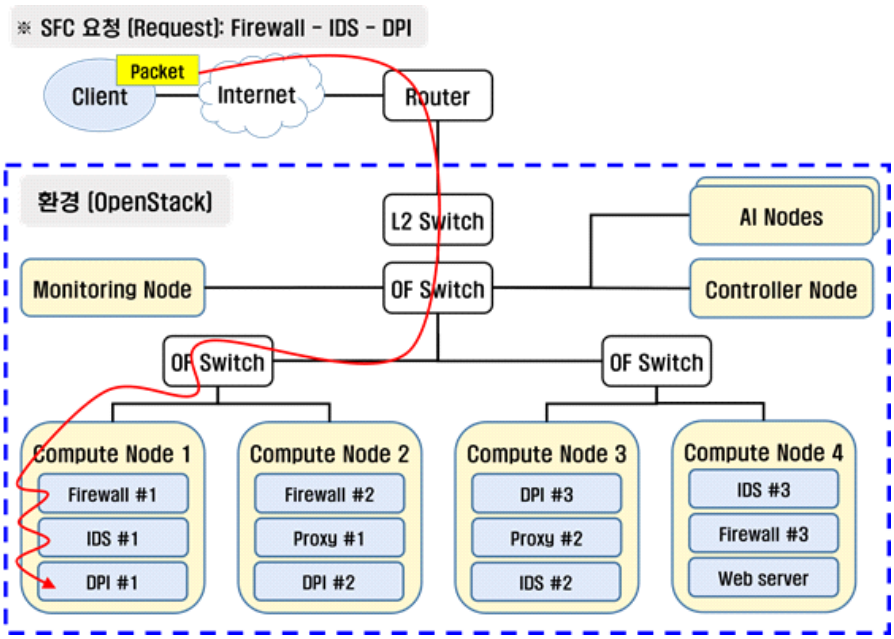
(그림 5) SFC 모듈의 Q-learning 기반 SFC 생성 기능 실행 순서도

Random 알고리즘과는 달리 Q-learning 알고리즘은 최적의 SFC 경로를 선택하기 위한 학습 과정이 필요하다. (그림 5)는 SFC 요청이 발생했을 때, Q-learning을 활용하여 SFC 경로를 결정하는 기능이 실행되는 순서도이다. SFC를 구성하는 VNF의 종류 및 순서가 명시된 SFC 요청 메시지를 수신하면, Q-learning에 필요한 모니터링 데이터를 모니터링 모듈에 요청하여 받아온다. VNF 인스턴스 자원 사용률, 설치 위치가 포함된 데이터를 수신하면 이를 활용하여 Q-value 표를 생성하고 Q-value를 초기화한다. 그 다음 순서로는 Q-learning을 수행하는데, 학습률, 할인율, 에피소드 횟수 등의 하이퍼파라미터 (Hyperparameter) 값들을 설정한다. 하이퍼파라미터 값 설정 후 Q-learning은 현재 상태에서 Q-value에 기반 하여 특정 행동을 수행하는데, 일반적으로 가장 높은 Q-value를 가지는 행동을 수행한다. 하지만, 항상 Q-value 값에 따라서만 행동을 수행하면 초기에 잘못된 Q-value가 설정되어 있을 경우, 최적의 정책을 찾는데 적합하지 않을 수 있다. 따라서 기본적으로는 Q-value에 기반 하여 행동을 수행하

되, ε -greedy 알고리즘 등을 활용하여 일정 확률로 임의의 행동을 선택하도록 설정할 수 있다. 현재 상태에서 특정 행동을 수행한 후에는 그에 대한 보상 값을 계산하여 Q-value를 갱신한다. 그리고 다음 상태로 이동하게 되는데, 이동한 다음 상태가 마지막 상태 즉, SFC를 구성하는 마지막 VNF 인스턴스가 된다면 하나의 SFC 경로를 찾은 것이다. 마지막 상태가 아닌 경우에는 SFC 경로를 찾을 때까지 Q-value에 기반하여 마지막 상태에 도달할 때까지 행동을 수행해나간다. SFC 경로를 찾을 때마다 현재 수행한 학습 횟수 N이 미리 설정해 놓은 에피소드 값보다 작은 지 판별한다. 아직 학습 횟수 N이 에피소드 값에 이르지 못하면, 시작 상태로 돌아가서 SFC 경로를 찾는 과정을 반복한다. 이 후, 에피소드 값만큼 학습을 반복하면, 최종 갱신 된 Q-value를 활용하여 시작 상태에서 가장 높은 Q-value를 가진 행동들을 수행하면서 (VNF 인스턴스들을 선택하면서) SFC 경로를 결정한다. 본 문서에서 구현한 SFC 모듈은 Github를 통해 공개되어 있다 [6].

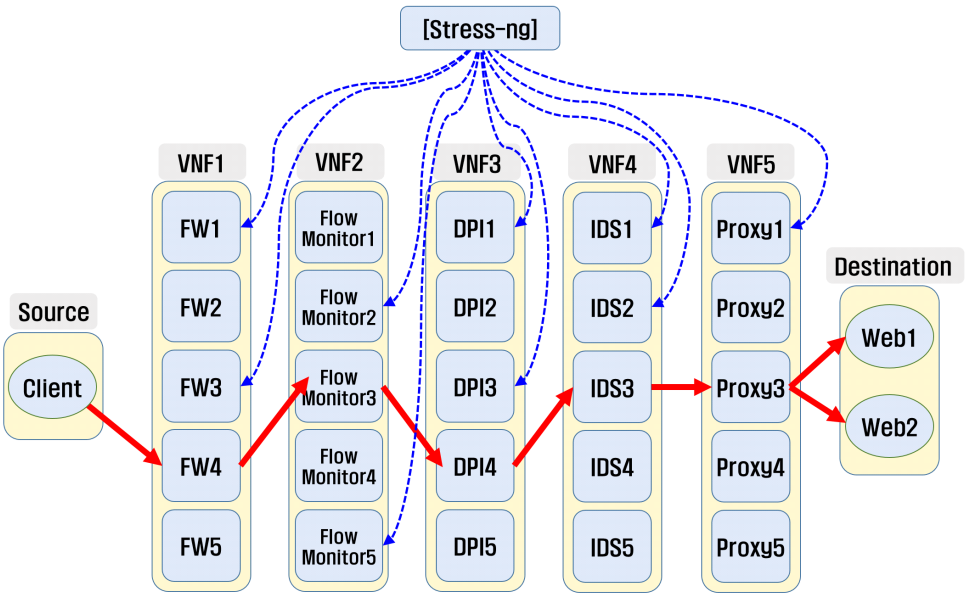
2.3 성능 검증

Q-learning 기반 SFC 경로 선택 방법의 성능 검증을 위해 (그림 6)과 같이 OpenStack 기반으로 테스트베드를 구축하였다. 테스트베드는 OpenStack 환경을 관리하는 컨트롤러 노드 (Controller Node)와 VNF 인스턴스가 설치 될 수 있는 4개의 컴퓨트 노드 (Compute Node)로 구성된다. 테스트베드 내에 VNF 인스턴스를 설치 할 때에는 임의의 컴퓨터 노드를 선택하여 VNF가 설치 된 VM을 생성하며, 각 VM에는 하나의 VNF만 동작한다. 본 문서의 성능 검증에서는 모든 VNF 인스턴스들을 미리 테스트베드에 생성해 놓은 상태에서 SFC를 구성하는 인스턴스들을 선택해 SFC를 구성한다. 본 문서의 OpenStack 테스트베드 구축을 위해 사용한 OpenStack 설치 스크립트 파일은 Github를 통해 공개되어 있다 [7].



(그림 6) OpenStack 기반 테스트베드

성능 측정 시나리오는 (그림 7)의 클라이언트 (Client)에서 다수의 HTTP 요청 메시지 (HTTP Request Message)를 발생시켜 테스트베드 내 SFC 경로를 경유해 웹 서버로 전달 한 후, 요청 메시지에 대한 모든 응답 메시지 (Response Message)를 받는 데 소요되는 시간을 측정하였다. 본 문서에서는 이를 서비스 시간 (Service time)으로 정의한다. 이 때, HTTP 요청 메시지 생성은 Apache 웹 서버 상태를 측정하는 성능 측정 도구인 AB (Apache HTTP Server Benchmarking tool)를 사용하였다. 웹 서버로 HTTP 요청 메시지가 전달되기 전에 경유하는 SFC는 5개의 VNF (Firewall, Flow monitor, DPI, IDS, Proxy)로 구성된다. 방화벽 기능을 하는 Firewall VNF는 Iptables [8]를 사용하였으며, Flow monitor는 ntopng [9], DPI는 nDPI [10], IDS는 Suricata [11], 그리고 Proxy 기능을 제공하는 VNF는 HAProxy [12]를 설치하였다. 마지막으로, HTTP 요청 메시지에 대한 응답을 위해 두 개의 Apache 웹서버를 설치하고, SFC의 마지막 VNF인 HAProxy가 Round-robin으로 각 웹 서버로 패킷을 전달하도록 설정하였다. 각 VNF 종류 별로 5개의 인스턴스를 생성하여 총 25개 VNF 인스턴스가 테스트베드 내에 존재한다. 각 VNF 인스턴스에 할당 된 자원은 vCPU 2개, RAM 1024MB, 디스크 20GB이다. 모든 VNF 인스턴스 생성을 마친 후에 생성할 수 있는 SFC 경로 개수는 3125 (5^5)개이다.



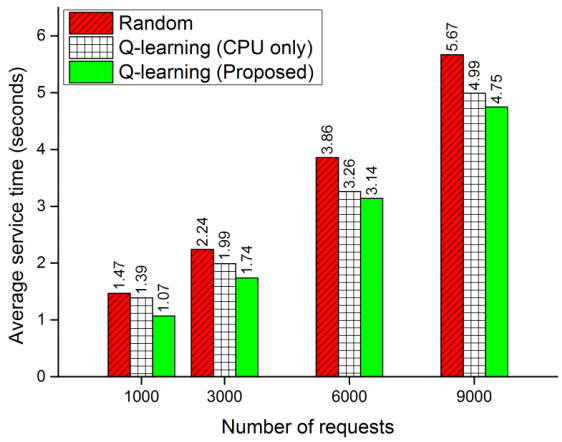
(그림 7) 성능 검증 시나리오를 위한 VNF 인스턴스 배치

본 문서의 Q-learning 기반 SFC 경로 선택의 성능 검증을 위해, 동일한 성능 측정 시나리오에서 Random 알고리즘으로 SFC를 구성한 경우와 비교하였다. 이 때, Stress-ng 도구를 사용하여 임의의 VNF 인스턴스에서 임의로 CPU 및 메모리를 사용하도록 설정하였다. Random 알고리즘으로 SFC 경로를 선택하는 경우에는 단순히 SFC를 구성하는 VNF들의 순서만을 고려하여 경로를 선택하지만, Q-learning 알고리즘을 활용하면 현재 측정된 VNF의 자원 사용량 (CPU, 메모리), 그리고 VNF 설치 위치를 보상으로 활용하여 학습 후 경로를 선택한다.

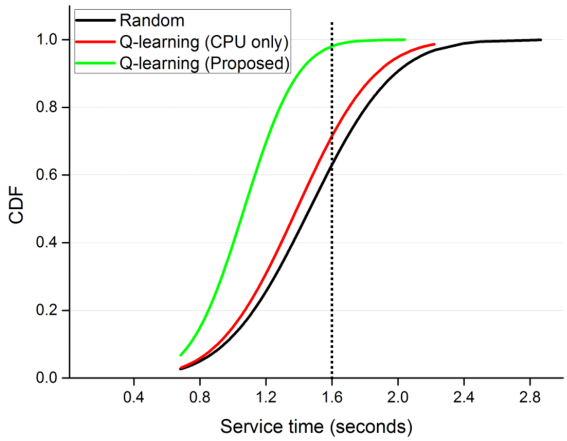
표 1. Q-learning 초기 변수 정보

변수 종류	값
η (학습률)	0.10
γ (할인율)	0.60
ϵ (탐험 확률)	0.90
Episode (에피소드)	3000

Q-learning 학습에 사용한 초기 변수 값은 [표 1]과 같다. 학습률 η의 초기 값은 0.10으로 고정하여 사용하였으며, 미래의 보상을 얼마나 가중치를 두고 환산할지 결정하는 할인율 γ 값은 0.90로 설정하였다. 또한, ε-greedy 알고리즘의 초기 ϵ 값은 0.90로 설정하여, 초반 학습 과정에서는 대부분의 경우 Q-value에 따른 행동 선택보다는 임의의 행동을 선택하도록 유도하였다. ϵ 값은 학습을 반복할수록 일정 비율로 줄어들게 된다. 마지막으로, 학습 반복 횟수를 의미하는 에피소드를 3000으로 설정하였다.



(그림 8) 평균 서비스 시간 측정 결과



(그림 9) CDF로 표현한 각 SFC별 서비스 시간

모든 설정을 마치고 나서, 서로 다른 방법으로 세 개의 SFC를 생성하였다. 첫 번째 SFC는 Random 알고리즘을 통해 생성하였으며, 두 번째 SFC는 Q-learning을 활용하되 보상 값을 계산할 때 오직 VNF 인스턴스들의 CPU 사용량만을 고려하였다. 세 번째 SFC는 Q-learning을 활용하며 앞서 서술한 VNF 인스턴스들의 CPU 및 메모리 사용량과 설치 위치를 보상 값으로 고려하였다. 이렇게 생성한 각 SFC 경로를 통해 HTTP 요청 메시지를 1,000개, 3,000개, 6,000개, 그리고 9,000개를 발생시킨 후, 모든 응답 메시지가 클라이언트로 도착하는 시간 (Total Response Time), 즉 서비스 시간을 측정한 결과는 (그림 8)과 같다. 전반적으로, Random 알고리즘을 통해 SFC를 구성하는 경우에는 Q-learning 기반으로 생성한 SFC에 비해 응답 메시지를 모두 받을 때까지 더 긴 시간이 필요하였다. 또한, HTTP 요청 메시지 개수가 증가할수록 각 SFC에서의 서비스 시간도 증가하였지만, Q-learning으로 생성한 SFC가 Random한 경우보다는 더 짧은 서비스 시간을 보장한다는 것을 확인하였다. 또한, CPU 사용량만을 보상으로 고려한 경우보다, 본 문서에서 서술한 CPU 및 메모리 사용량, 설치 위치를 보상으로

고려했을 때, 서비스 시간 측면에서 더 좋은 성능을 보이는 SFC 경로를 선택할 수 있음을 확인하였다. 또한, 평균적인 서비스 시간 측정 뿐 아니라, 1000개의 HTTP 요청 메시지를 전송했을 때 각 SFC을 통해 측정되는 응답 시간을 누적 분포 함수 (Cumulative Distribution Function, CDF)로 (그림 9)와 같이 확인하였다. 이에 따르면 본 문서에서 다루는 Q-learning 기반 SFC 경로 선택 방법으로 SFC를 구성할 경우, 1.6초 기준으로 95%의 HTTP 응답 메시지를 받을 수 있음을 확인하였다. 반면, Random 알고리즘으로 구성한 SFC에서는 동일한 시간 기준으로 65%, CPU 사용량만을 고려한 Q-learning의 경우 70%의 응답 메시지만을 수신할 수 있었다. 결과적으로, 본 문서에서 다루는 Q-learning 기반 SFC 경로 선택 방법은 평균 서비스 시간 뿐 아니라, 각 요청 메시지를 안정되게 처리할 수 있는 경로를 결정한다는 사실을 확인하였다.

3 Q-learning 기반 SFC 경로 선택 모듈 사용법

3.1 환경 구축

본 문서에서 다루는 Q-learning 기반 SFC 경로 선택 방법은 OpenStack 환경에서 동작할 수 있는 형태로 구현되었다. 앞서 서술한 것처럼, 구현 된 SFC 모듈은 SFC 생성을 위해 OpenStack 테스트베드 내 존재하는 각 VNF 인스턴스들의 데이터를 활용하고, SFC 모듈에서 Q-learning 또는 Random 알고리즘을 통해 결정 된 SFC 경로는 실제 OpenStack 테스트베드 내 설치되어야 한다. 이를 위해 OpenStack 기반 환경 뿐 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 [13]는 Github를 통해 공개되어 있기 때문에 본 문서에서 해당 요소들의 설치 과정은 생략한다. OpenStack 환경이 구축되고, NFVO와 모니터링 모듈울 설치한 후에는 SFC 모듈을 설치해서 실행할 수 있다. 본 문서에서 모듈 설치와 실행은 Ubuntu 16.04에서 수행하였다.

SFC 모듈은 Python으로 작성되었으며, Python v3.5.2 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 명을 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 10)은 test라는 가상 환경을 생성하고 활성화 한 후에, 비활성화까지 진행한 화면을 보이고 있다.

```
ubuntu@dy-test-1:~$ virtualenv test
created virtual environment CPython3.5.2.final.0-64 in 285ms
  creator CPython3Posix(dest=/home/ubuntu/test, clear=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=/home/ubuntu/.local/sha
re/virtualenv)
    added seed packages: pip==20.1.1, setuptools==49.2.0, wheel==0.34.2
  activators PowerShellActivator,BashActivator,CShellActivator,PythonActivator,FishActivator,XonshActivator
ubuntu@dy-test-1:~$
ubuntu@dy-test-1:~$ ls
test
ubuntu@dy-test-1:~$
ubuntu@dy-test-1:~$ source test/bin/activate
(test) ubuntu@dy-test-1:~$
(test) ubuntu@dy-test-1:~$ deactivate
ubuntu@dy-test-1:~$
```

(그림 10) virtualenv를 통한 가상 환경 생성

3.2 SFC 경로 선택 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 설치 한 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 받으면 Random 또는 Q-learning 알고리즘을 활용하여 SFC 경로를 결정하기 때문에 웹 서버로 동작하면서 메시지를 처리한다. SFC 모듈에서 설정되어 있는 기본 웹 서버 포트는 8001이며, 포트 변경을 원할 경우에는 내려 받은 SFC 모듈 폴더의 server/__main__.py 파일에서 (그림 11)과 같이 port라고 명시 된 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 11) SFC 모듈 server/__main__.py 파일 - Port 설정 가능

SFC 모듈은 SFC 경로 설정을 위해 NFVO 및 모니터링 모듈과 상호작용한다. 이를 위해 설정 파일에 각 모듈이 동작하는 Host의 IP 주소와 포트 번호를 미리 입력해야 한다. 내려 받은 SFC 모듈 폴더 내에서 config/config.yaml을 열면, 모니터링 모듈에 해당하는 ni_mon, NFVO에 해당하는 ni_nfvo의 host 정보를 각각 수정하고 저장한다.

```yaml
ni_mon:
  host: http://<ni_mon_ip>:<ni_mon_port>
ni_nfvo:
  host: http://<ni_nfvo_ip>:<ni_nfvo_port>
```

(그림 12) SFC 모듈 config/config.yaml 파일 설정

2장에서 설명한 것처럼 Q-learning을 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. Q-learning을 위한 하이퍼파라미터 값과 SFC 생성에 사용할 네트워크 포트 식별을 위한 OpenStack Network ID는 내려 받은 SFC 모듈 폴더 내에서 sfc_path_selection.py 파일을 열어 수정한다. OpenStack Network ID는 OpenStack Dashboard 또는 OpenStack CLI를 통해 확인 가능하며, 해당 네트워크에 속한 포트들을 식별하여 SFC 생성에 활용한다.

```python
# Parameters
# OpenStack Parameters
openstack_network_id = "" # Insert OpenStack Network ID to be used for creating SFC

# <Important!!!!> parameters for Reinforcement Learning (Q-learning in this codes)
learning_rate = 0.10          # Learning rate
discount_factor = 0.60        # Discount factor
initial_epsilon = 0.90        # epsilon value of -greedy algorithm
num_episode = 3000            # Number of iteration for Q-learning
```

(그림 13) Q-learning의 Parameter 및 OpenStack Network ID 설정

설정을 마무리하고 실행 명령어를 입력하여, 정상적으로 SFC 모듈이 실행되면 (그림 14)과 같은 출력 화면을 볼 수 있다. 본 문서에서는 기본 포트 8001을 변경 없이 사용한다.

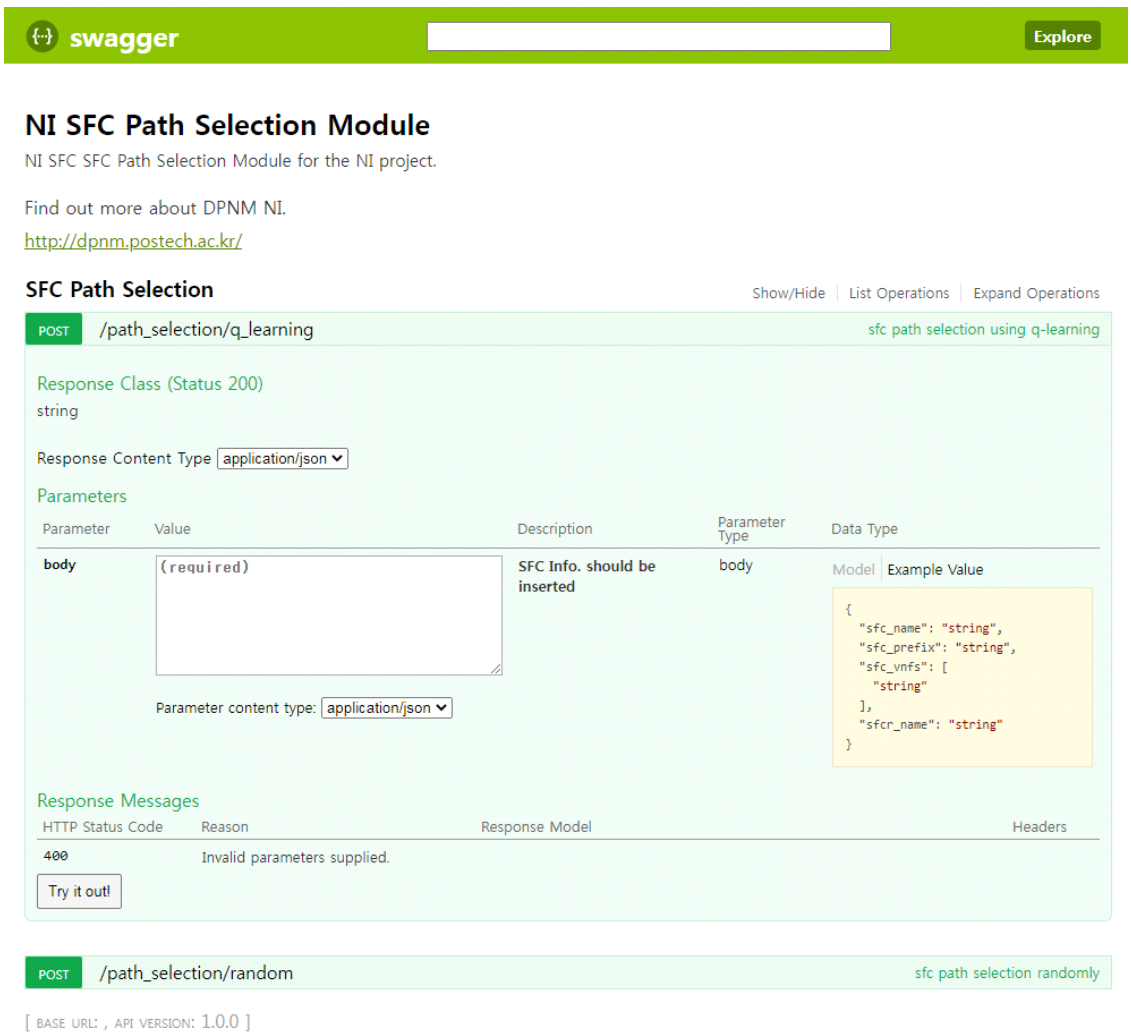
```
(test) ubuntu@dy-test-1:~/ni-sfc-path-module-public$ python3 -m server
 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running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

(그림 14) SFC 모듈 실행 화면

3.3 SFC 경로 선택 모듈 사용법

SFC 모듈이 정상적으로 실행되고, 설정에 오류가 없을 경우 SFC 모듈은 (그림 15)처럼 제공되는 Web UI를 아래 경로를 통해 확인할 수 있다.

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

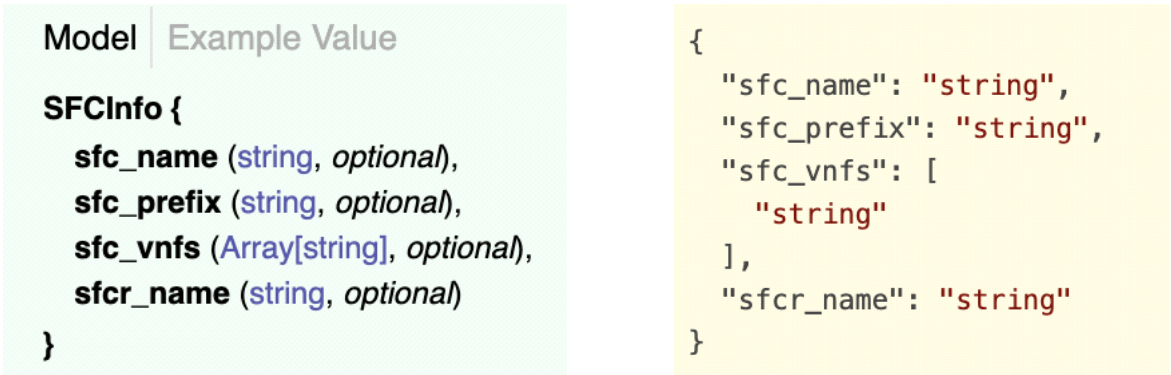


(그림 15) SFC 모듈 실행 화면

SFC 모듈은 SFC 생성 요청 메시지를 HTTP POST로 받는다. SFC 생성 요청 메시지는 Web UI를 통해 직접 SFC 모듈로 요청하거나, HTTP POST 메시지를 생성하여 SFC 모듈로 전달하는 방법 두 가지가 존재한다. 본 문서에서는 SFC 모듈의 Web UI를 통해 SFC 생성 요청 메시지를 전달하는 방법에 대해 서술한다. SFC 모듈은 HTTP POST로 요청 메시지를 받기 때문에 해당 메시지를 수신 할 경로를 다음과 같이 제공한다.

1) Q-learning 기반 SFC 생성: http://<SFC 모듈 Host>:<포트>/path_selection/q_learning
2) Random 기반 SFC 생성: http://<SFC 모듈 Host>:<포트>/path_selection/random

SFC를 생성할 때 사용할 알고리즘을 URL을 통해 식별한 후에는 HTTP POST 메시지의 Body에 명시 된 데이터를 활용하여 SFC를 생성한다. (그림 16)처럼 SFC 요청 메시지의 Body에는 SFCInfo라고 정의 된 모델이 JSON으로 표현된다. SFCInfo 모델은 총 4개의 string 형태의 데이터로 구성된다. (그림 16)에서 왼 쪽은 SFCInfo 모델의 정의를 보이며, 오른 쪽은 SFC 요청 메시지에 각 데이터들이 포함될 때 어떤 형식으로 기입되어야 하는지 보이는 예시이다. 요청 메시지에 명시할 때는 각 데이터의 이름과 값들을 “” 내에 기입한다.



(그림 16) SFC 생성 요청 메시지 Body 모델 및 예

SFCInfo에 명시되는 데이터의 각 의미는 아래와 같다.

1) sfc_name: OpenStack에서 생성되는 SFC 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

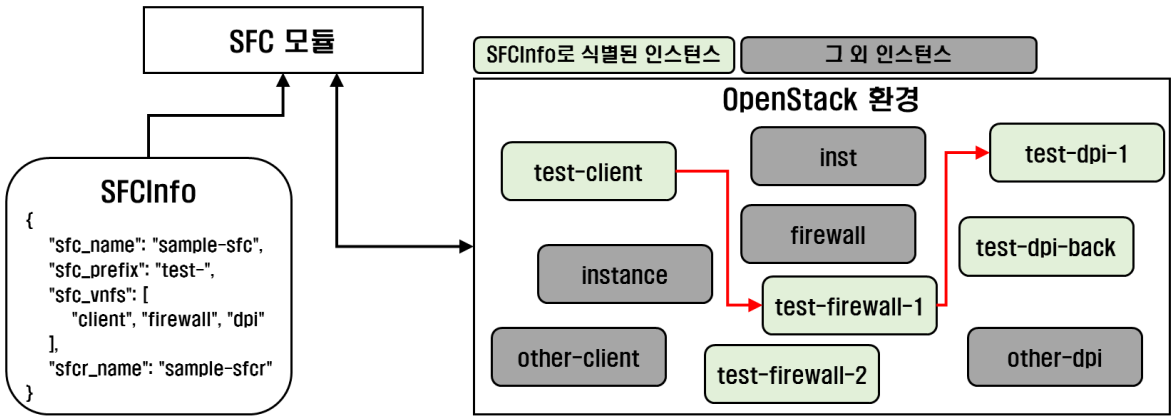
SFC로 흘러야 하는 트래픽을 분류하는 Classifier의 네트워크 포트 IP 주소를 수동으로 설정해야 한다는 불편함이 존재한다. 따라서 본 문서의 SFC 모듈은 사람이 기억하고 이해하기 힘든 네트워크 포트 ID 또는 IP보다는 Classifier 역할을 수행하는 VM과 SFC를 구성하는 각 VNF 인스턴스의 이름을 입력하면, NFVO 및 모니터링 모듈과 상호작용하여 자동으로 SFC를 생성해준다.

(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. sfc_prefix는 OpenStack 내 존재하는 다양한 인스턴스 중 SFC 생성에 활용 할 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 후의 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 Random 알고리즘이나 Q-learning 알고리즘에 의해 선택되지 않고 특정 Classifier을 선택하며, VNF 인스턴스들은 여러 인스턴스들 중에서 한 개씩을 Random 또는 Q-learning을 통해 선택해 SFC를 생성한다.

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

(예시) SFC 생성 요청 메시지 Body 예

(그림 17)은 SFC 모듈의 (예시)와 같은 Body를 포함한 SFC 생성 요청 메시지에 의한 SFC 생성 과정을 도식화한 것이다. SFCInfo에 명시 된 sfc_prefix와 sfc_vnfs들을 활용하여 SFC 생성에 활용 할 수 있는 인스턴스들을 우선 식별한다. 그 후에는 SFC 모듈에서 Random 또는 Q-learning 알고리즘을 적용하여 SFC 경로를 결정한 후 이를 OpenStack 환경에 실제 SFC로 생성한다. (그림 17)에서 빨간 선으로 연결 된 것이 SFC 모듈에서 결정 된 SFC 경로에 따라 생성된 실제 SFC이다.



(그림 17) SFC 모듈의 SFC 생성 요청 메시지에 의한 SFC 생성

(그림 18)은 Random 알고리즘을 통해 SFC 생성을 요청했을 때, SFC 모듈이 출력하는 응답 메시지이다. 정상적으로 SFC가 생성되었을 경우, 정상적인 응답 메시지에 기입되는 응답 코드 (Response code) 200과 함께, 응답 메시지 Body에 생성 된 SFC와 관련 된 정보를 받을 수 있다. 그 중, SFC 구성에 사용 된 Classifier와 VNF 인스턴스들의 이름은 sfc_path 항목에 나열된다. 또한, SFC 생성과 관련하여 OpenStack에서 Networking-SFC 플러그인을 통해 생성 된 sfc_id와 sfcr_id가 명시된다. sfc_id와 sfcr_id는 본 문서에서 자세히 서술하지 않는다. 만약, 응답 메시지로 오류를 받았을 경우에는 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 각 부분을 확인할 필요가 있다.

Response Body


```
{
  "sfc_id": "770fa0fc-51a3-4662-a76c-c657c4992dd6",
  "sfc_path": [
    "test-client-1",
    "test-firewall-2",
    "test-dpi-1"
  ],
  "sfcr_id": "7df52c88-5c8e-4c3d-acda-495ccca298a6"
}
```


Response Code

```
200
```

Response Headers


```
{
  "content-length": "190",
  "content-type": "application/json",
  "date": "Tue, 11 Aug 2020 08:06:55 GMT",
  "server": "Werkzeug/0.16.1 Python/3.6.9"
}
```


(그림 18) SFC 모듈의 SFC 생성 요청 메시지에 대한 응답 메시지

4 결론

본 문서에서는 강화학습 알고리즘 중 하나인 Q-learning을 활용해 SFC 경로를 찾는 방법을 서술하였다. Q-learning 기반 SFC 경로 선택 방법은 SFC를 구성하는 VNF의 현재 CPU 사용량과 메모리 사용량, 배치된 물리 서버 위치를 고려하여 보상을 책정하고, 이를 기반으로 짧은 서비스 시간을 제공하는 SFC 경로를 선택한다. 본 문서의 Q-learning 기반 SFC 경로 선택 방법은 임의로 SFC 경로를 선택하는 것보다는 응답 시간 측면에서 좋은 경로를 선택할 수 있도록 돕지만, 현재 자원 상태와 VNF 설치 위치 같은 제한된 정보만을 보상으로 고려한 경로 선택이기 때문에 네트워크 혼잡 등 동적으로 네트워크 상태가 변할 때 대응하기 어렵다는 문제도 있다. 따라서 향후에는 단순한 Q-learning이 아닌, 신경망을 도입한 DQN (Deep Q Network) 등의 기법을 활용하여 NFV 환경에서 고려할 수 있는 다양한 경우들을 상태로 정의한 후, SFC 생성 연구에 활용할 예정이다.

5 참고문헌

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

[2] Gallenmüller, S., Emmerich, P., Wohlfart, F., Raumer, D., & Carle, G. (2015, May). Comparison of frameworks for high-performance packet IO. In 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS) (pp. 29-38). IEEE.

[3] Dobrescu, Mihai, Norbert Egi, Katerina Argyraki, Byung-Gon Chun, Kevin Fall, Gianluca Iannaccone, Allan Knies, Maziar Manesh, and Sylvia Ratnasamy. "RouteBricks: exploiting parallelism to scale software routers." In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pp. 15-28. 2009.

[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] D. Coulson, “Network security iptables,” 2003.

[9] L. Deri, M. Martinelli, and A. Cardigliano, “Realtime high-speed network traffic monitoring using ntopng,” in 28th Large Installation System Administration Conference (LISA14), 2014, pp. 78–88.

[10] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, “ndpi: Open-source high-speed deep packet inspection,” in 2014 International Wireless Communications and Mobile Computing Conference (IWCMC). IEEE, 2014, pp. 617–622.

[11] Suricata: Open Source IDS/IPS/NSM engine. [Online]. Available: https://suricata-ids.org/

[12] W. Tarreau et al., “Haproxy-the reliable, high-performance tcp/http load balancer,” 2012.

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