NI 기술 문서

Machine Learning을 활용한 OpenStack 환경 내 VNF Deployment 방법

문서 ID	NI-Deployment-01
문서 버전	v2.0
작성일	2020.10.21
작성자	포항공과대학교 김희곤

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
2020.09.28	v1.0	초안 작성	김희곤
2020.10.02	v1.1	그림 갱신	김희곤
2020.10.10	v1.2	오타 수정 및 내용 보완	김희곤
2020.11.14	v1.3	오타 수정	김희곤
2020.11.22	v2.0	참고문헌 추가	김희곤

목 차

요 약 문

오늘날 5G 네트워크 시대가 도래하면서, 급변하는 서비스 요구사항을 만족시키기 위해 유연하고 민첩한 네트워크 구축 및 관리가 요구되고 있다. 네트워크 기능 가상화 (NFV, Network Function Virtualization)는 이를 실현하기 위한 기술 중 하나로, 전용 하드웨어를 통해 제공되는 네트워크 기능을 소프트웨어 형태로 구현하여 상용 서버에서 가상 네트워크 기능 (VNF, Virtual Network Function)으로 운영하는 것이다. NFV는 네트워크 기능을 네트워크에 유연하게 적용할 수 있는 장점이 있지만, 한편으로는 수많은 가상 자원을 생성하기 때문에 네트워크 관리를 복잡하게 만드는 원인이 된다. 복잡한 네트워크를 사람이 직접 관리하는 것은 어려운 일이기 때문에, 최근에는 네트워크 관리에 인공지능 (AI, Artificial Intelligence)을 적용하는 연구가 주목을 받고 있다.

NFV 환경의 VNF-deployment은 VNF 라이프 사이클 (Life-cycle) 관리 기능 중 하나로, VNF 인스턴스 (Instance) 개수를 추가/제거하는 것을 의미한다. 본 문서에서는 네트워크 환경과 트래픽 요구조건에 따라 네트워크 관리 비용(Opex)를 줄이는 것을 목적으로 하여 최적으로 VNF 인스턴스 수를 조절하는 것을 다룬다. VNF Deployment 기능을 이용하면 네트워크 관리자는 네트워크 상황에 대응하여 최적의 관리가 가능하지만 빠르게 변화하는 네트워크 상황에 맞춘 빠른 VNF Deployment 판단이 요구되기도 한다. 네트워크 운용비용은 많은 요소들이 들어가서 복잡한 계산식으로 얻을 수 있으며 VNF Deployment가 네트워크 운용비용에 미치는 영향 역시 복잡한 수식으로 표현이 되어 빠른 시간 내에 최적의 Deployment 판단을 내리기 어렵다. 이러한 문제를 해결하기 위해서 기계 학습을 사용하여 VNF Deployment 에 대한 판단을 내리는 것을 제안이 되었다. 본 문서에서는 NFV 환경의 VNF Deployment 문제를 정의하며, 기계 학습이 어떻게 해당 문제를 해결하며, OpenStack 내에서 해당 모듈을 이용하는 방법에 대해서 서술한다.

1 NFV 환경에서의 VNF Deployment 방법

1.1 VNF Deployment 개요

NFV 환경의 VNF Deployment 기능은 네트워크 변화에 대응하여 필요한 Virtual Network Function (VNF) 의 인스턴스를 동적으로 추가/제거할 수 있는 기능이다. 네트워크 관리자는 VNF Deployment 기능을 통해 유저가 요구하는 서비스를 공급할 수 있으며, 네트워크 자원을 효율적으로 사용할 수 있다.

네트워크 서비스는 일련의 VNF을 이용하여 제공이 되기 때문에, 유저가 요구하는 네트워크 서비스가 달라지면 네트워크에 설치되어야 하는 VNF도 달라지게 된다. 네트워크 관리자는 이러한 변화에 대응하여 필요한 VNF를 네트워크의 서버에 Deploy한다. 네트워크는 한정된 자원을 가지고 유저의 요구를 만족시킨다. 따라서 효율적인 네트워크 관리는 관리자의 중요한 역할 중 하나이다. 네트워크 관리자는 유저가 더 이상 필요로 하지 않는 VNF를 제거하여 네트워크 자원을 확보할 수 있으며, 필요 이상의 VNF 인스턴스를 설치하지 않는 것으로 최적의 네트워크 효용을 가질 수 있다. 따라서 VNF Deployment 기능은 VNF를 네트워크 서버에 설치 및 제거하는 기본적인 기능 외에도 효율적인 VNF 관리 역할을 포함하고 있다. 본 기술 문서에서 VNF Deployment 문제는 특정 타입의 VNF의 인스턴스를 네트워크의 어느 서버에 설치하며, 그 수는 정하는 것으로 정의된다. 해당 문제는 VNF 인스턴스의 개수를 조절하는 것으로 Scale-in/out이라 구분할 수도 있다.

전통적인 네트워크 환경에서 VNF는 오버 프로비저닝(Over-provisioning) 되었다. 최대한 만흔 종류의 VNF를 넉넉하게 서버에 설치하는 것으로 유저의 요구가 변화하였을 때, 문제 없이 네트워크 서비스를 제공하고자 하였다. 하지만 해당 방법은 네트워크 자원을 과사용하는 문제가 있어 관리와 운용에 비용이 많이 들었다. 해당 문제를 해결하기 위해 임계값 기반(Threshold-based auto-scaling)을 이용하여 서버에 설치되는 VNF 의 종류와 인스턴스 수를 결정하기도 하였다. 하지만 해당 방법은 네트워크에 대한 이해와 VNF 종류 등에 따른 전문적 지식을 요구한다는 단점이 있다. 또한 보통의 경우, 네트워크는 다수의 유저가 존재하며 각각의 유저들이 서로 다른 서비스를 요구하며, 해당 서비스들의 제약조건이 다르기 때문에 모든 경우에 대해 완벽한 임계값을 설정한다는 것은 거의 불가능에 가깝다. 따라서 이러한 문제를 해결하기 위해 동적 프로그래밍을 통해 관리식을 세우거나 휴리스틱 모델을 만들어 네트워크 관리 문제를 해결하고자 하였다. 하지만 동적프로그래밍은 계산 소요시간 비용이 많이 들어 실시간으로 변화는 네트워크에 적용이 힘들며 휴리스틱 모델은 임계값 기반이나 동적 프로그래밍에 비해 관리가 정확하지 않은 문제가 있었다. 최근에는 이러한 문제를 해결하기 위해 기계학습을 적용하고자 하는 시도가 진행되고 있다. 기계학습 모델은 주어진 데이터를 스스로 학습할 수 있기 때문에, 네트워크 관리자는 복잡한 네트워크의 이해없이도 효율적인 네트워크 관리가 가능해 진다. 본 문서에서는 기계학습을 이용한 VNF Deployment 방법에 대해 서술한다.

2 Machine Learning 기반 VNF Deployment 방법

2.1 VNF Deployment 환경 정의

VNF Deployment 기능을 구현하기 위해 네트워크 환경을 정의해야 한다. 본 문서에서 서술하는 기계학습 기반 VNF Deployment 기능은 <네트워크 데이터>, <VNF 데이터>, <서비스 정보 데이터> 총 세 개의 데이터 셋을 정의하여 사용한다. 해당 데이터 셋을 바탕으로 동적 프로그래밍 기법 중 하나인 Integer Linear Programming (ILP)를 이용하여 최적의 VNF Deployment 정책을 생성하며, 해당 정책을 Ground-Truth 라벨 데이터로 사용하여 기계 학습 모델이 ILP 모델을 학습하도록 하였다.

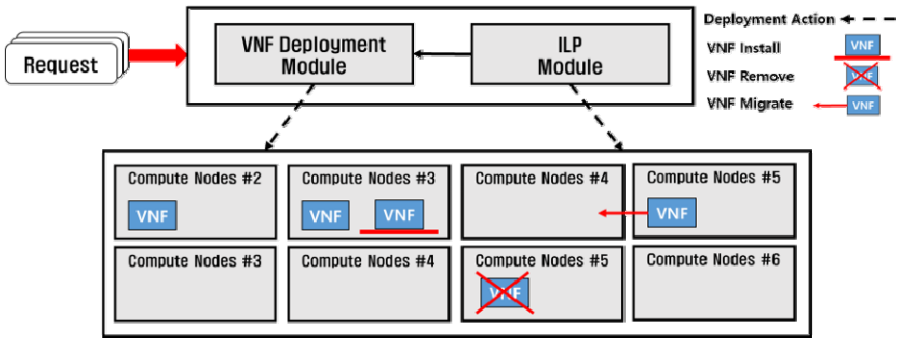


그림 1 ILP와 기계 학습 모델에 따른 VNF의 Install, Remove, Migration

네트워크 데이터(수식 1)와 VNF 데이터(수식 2)는 아래 수식과 같이 정의되어 사용되었다. 네트워크는 노드와 링크로 구성되어 있으며, 노드는 가용 CPU core의 수, 가용 대역폭 정보를 가지고 있으며, 링크는 딜레이와 가용 대역폭 정보를 가지고 있다. 트래픽 데이터는 트래픽이 시작되는 노드, 도달하는 노드 정보를 가지고 있으며 해당 트래픽의 서비스 타입과 함께, 서비스 지속시간, 서비스 지연으로 인한 비용, 최대 허용 지연시간, 요구 대역폭 정보를 가지고 있다. 네트워크 데이터와 VNF 데이터는 테스트베드에서 수집되어 사용된다. 서비스 정보 리스트 데이터는 수식3과 같이 표현된다. 서비스 정보 리스트는 유저가 새로운 서비스를 요구할 때 마다 현재 시점에서 네트워크에 요구되는 서비스들의 정보를 나타낸다. 서비스 정보 리스트가 생성될 때마다 물리 네트워크와 VNF의 정보를 수집하여 데이터 간의 동기가 이루어지도록 한다.

기계 학습이 사용할 라벨링 데이터를 만들기 위해 Integer Linear Programming (ILP)를 사용한다. ILP는 VNF의 관리 비용을 낮추는 것을 목표로 하며, VNF의 설치 비용, 에너지 비용, 트래픽 전달 비용, 서비스 딜레이로 인한 비용 등을 최소화하는 것을 목표로 한다(수식 4). 알고리즘은 ILP가 생성한 라벨 데이터를 학습하여 ILP의 모델을 학습하도록 하였다.

$G = (N, E)$

N : 네트워크 노드

E: 링크

$E_{ij} \in E$: 노드 i,j 의 링크

노드 데이터 $D_s = (\ c_s, D_{V_s}\ )$

c_s : 가용 가능한 CPU core 수,

V_s: 노드 s에 설치된 VNF

D_{V_s} : 노드 s에 설치된 VNF 데이터

L : 링크 데이터

$L_{ij} = (m_{ij}, d_{ij}, b_{ij})$

m_{ij} : 링크 E_{ij}의 최대 허용 대역폭

d_{ij} : 링크 E_{ij}의 지연시간

b_{ij} : 링크 E_{ij}의 가용 대역폭

C:링크 연결 데이터

$$C_{ij} = \begin{cases} 1, & \text{노드 } i\text{와 } j\text{가 같거나, } i\text{와 } j\text{간의 링크가 존재할 경우} \\ 0, & \text{그외의 경우.} \end{cases}$$

수식 1. 네트워크 데이터 정의

T : VNF 종류 집합

$t \in T$: VNF 종류

$V_{st} \in V_s$: 노드 s 에 설치된 t 종류의 VNF

D_{st}: 노드 s에 설치된 t 종류의 VNF 데이터

$$D_{st} = \begin{cases} (I_{st}, \tau_t, \kappa_{st}), & V_s\text{가 } V_{st}\text{ 을 가지고 있을때} \\ 0, & \text{그외의 경우.} \end{cases}$$

I_{st} : VNF instance 수

κ_{st}: 노드 s 에 설치된 t 종류 VNF의 현재 사용 대역폭

τ_t: t 종류 VNF의 최대 대역폭

ε_t: t 종류 VNF가 요구하는 CPU core의 수

δ_t: t 종류 VNF의 프로세싱 딜레이

$\digamma_t$: t 종류 VNF의 설치 비용

$$D_{V_s} = \bigcup_{t \in T} D_{st}$$

ψ : 서비스 집합

$v \in \psi$: 서비스

$v = (w_v, u_v, d_v, \phi_v, p_v, \beta_v, \gamma_v)$

w_v : v서비스 시작 노드

u_v: v 서비스 도착 노드

d_v : v 서비스 실행 시간

ϕ_v:v 서비스 종류

p_v: v 서비스 딜레이에 따른 패널티 비용

β_v : v 서비스 요구 대역폭

γ_v : v 서비스 딜레이 조건

수식 2. VNF 및 트래픽 데이터 정의

π : 서비스 정보 리스트

ν_{new} :가장 최근에 생성된 서비스

μ : 서비스 리스트 정보 아이디

$\hat{a}$: 서비스 생성 시간

$$\pi_{\mu+1} = \{\nu_{new}\} \cup \pi_{\mu} \cap \{x : d_{\nu_x} > \hat{a}_{\nu_{new}} - \hat{a}_{\nu_x}\}$$

수식 3. 서비스 정보 리스트

에너지 비용:

$$E = \sum_{n \in N} \sum_{t \in T} I_{st} (e_{te} + (e_{pk} - e_{te}) \frac{\xi_t}{C_{spec}}) \lambda_{energy}$$

e_{te} : idle 에너지 비용
e_{pk} : peak 에너지 비용
C_{spec} : CPU Cores 수
λ_{energy} : 에너지 소모 비용

트래픽 포워딩 비용:

$$W_{s_1 s_2}^{\nu t_1 t_2} = \begin{cases} 1 & \text{서비스 } \nu \text{에 대해서 } \nu_{s_1 t_1} \text{와 } \nu_{s_1 t_1} \text{간의 트래픽이 존재할때} \\ 0. & \text{그외의 경우.} \end{cases}$$

$$J_{s_1 s_2}^{\mu t_1 t_2} = \sum_{\nu \in \pi_\mu} \sum_{t_1 t_2 \in T} W_{s_1 s_2}^{\nu t_1 t_2} \beta_\nu$$

$$T = \sum_{s_1 \in N} \sum_{s_2 \in \vartheta(s_1), s_2 < s_1} (J_{s_1 s_2}^{\mu t_1 t_2} - J_{s_1 s_2}^{(\mu-1) t_1 t_2}) \lambda_{transit}$$

$\vartheta(s_1)$: 노드 s_1의 이웃 노드

$\lambda_{transit}$: 패킷 포워딩 비용

서비스 지연 비용:

$\widehat{\phi_\nu}$: ϕ_ν에 포함되는 VNF 들

$$S = \sum_{\nu \in \pi_\mu} \max (\sum_{t \in \widehat{\phi_\nu}} \delta_t + \sum_{s_1 \in N} \sum_{s_2 \in \vartheta(s_1), s_2 < s_1} \sum_{t_1 t_2 \in T} W_{s_1 s_2}^{\nu t_1 t_2} d_{s_1 s_2} - \gamma_\nu, 0)$$

리소스 분할 비용:

$$F = \sum_{s \in N, V_s \neq 0} (C_{spec} - \sum_{t \in T} I_{st} \xi_{st}) \lambda_{core} + \sum_{s_1 \in N} \sum_{s_2 \in \vartheta(s_1)} \frac{\max(J_{s_1 s_2}^{\mu t_1 t_2}, 0)}{J_{s_1 s_2}^{\mu t_1 t_2}} (m_{s_1 s_2} - J_{s_1 s_2}^{\mu t_1 t_2}) \lambda_{band}$$

λ_{core} : CPU core 개별 비용
λ_{band} : 개별 대역폭 사용 비용

전체 ILP 목적식:

Minimizing ($\acute{a}$E+$\acute{b}$T+$\acute{c}$S+$\acute{d}$F)

$\acute{a}$, $\acute{b}$, $\acute{c}$, $\acute{d}$: Weighting factor

수식 4 Integer Linear Programming 조건

2.2 Graph Neural Network 기반 VNF Deployment 모듈 구현

VNF Deployment 기능을 위해 정의된 데이터들은 그래프 형태로 변환한 뒤 Graph Neural Network (GNN) 모델에 적용한다. 그래프 데이터는 세 개의 행렬로 구성이 되며 각각 노드 행렬, 엣지 행렬, 연결 행렬로 표현된다. GNN 모델로는 Edge-conditioned Filtered Graph Convolutional Neural Network (그림 2) 를 사용하였다. 해당 학습 모델은 엣지의 정보를 이용하여 필터를 생성하고, 생성된 필터와 노드 정보를 연산하고 인접 노드에 전파를 하는 구조의 모델이다.

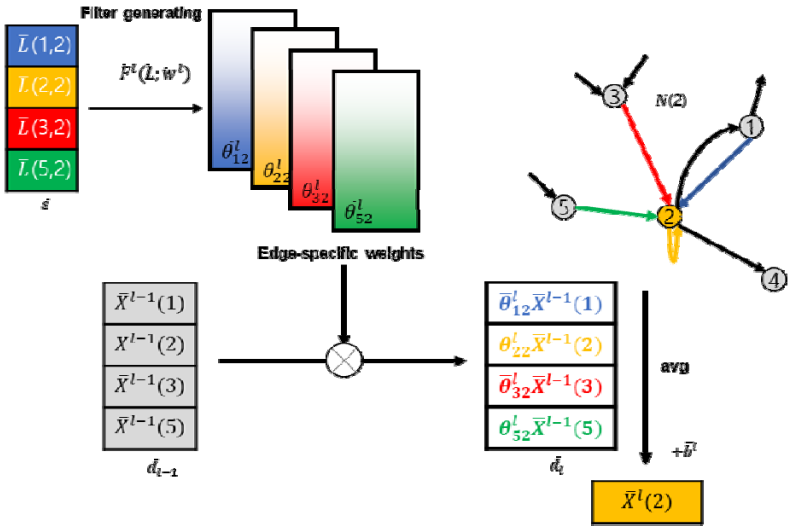


그림 2 Edge-conditioned Filtered Graph Convolutional Neural Network

GNN으로 네트워크 데이터를 학습하는 시간, 서비스 정보 리스트는 Feed Forward Neural Network (FNN) 모델을 사용하여 학습된다. 학습된 네트워크 데이터와 서비스 정보 리스트 데이터는 Concatenate 레이어를 통해 합쳐진 뒤, 두 개의 FNN 레이어와 Fully Connected Network (FCN) 레이어를 지나 결과 값을 생성한다. 전체 모델의 구조는 그림3에 표현되어 있다.

모델의 상세 하이퍼 파라미터의 값은 표1에 제시되어 있다. Learning rate는 0.01이며 Optimizer로 Adam을 사용하고 있다. Optimizer의 경우 성능과 학습속도의 교환을 위해 RMSProp, 또는 Adadelta, SGD와 같은 다른 Optimizer를 사용할 수도 있다. Batch Size는 1024이며 데이터의 크기에 따라 감소 혹은 증가 될 수 있다. FNN 및 GNN의 레이어 크기는 현재 30, 15, 200, 200으로 설정되어 있으며, 학습에 사용되는 데이터의 피쳐의 크기와 서비스 정보 리스트의 크기에 따라 변경될 수 가 있다.

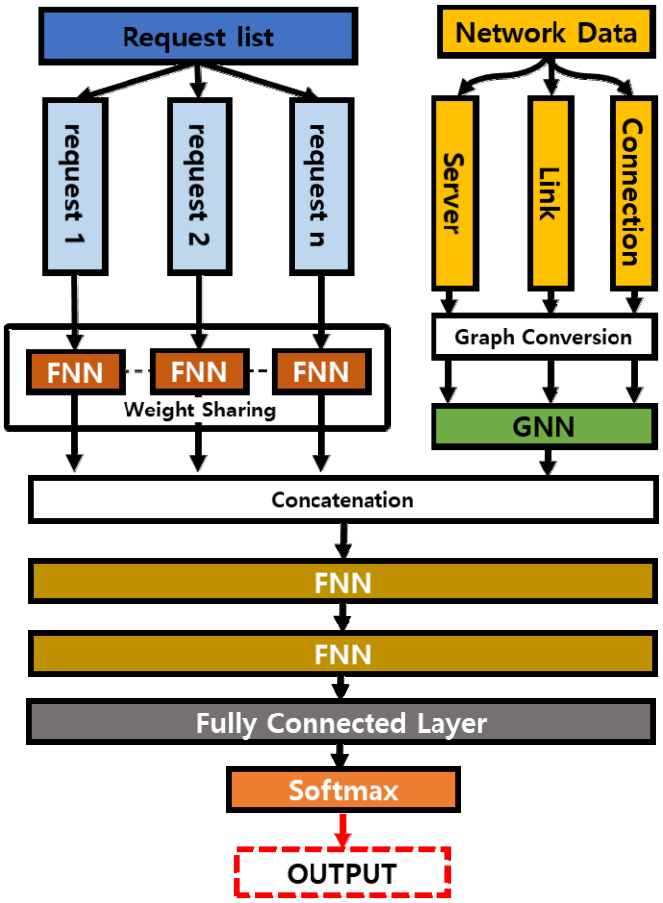


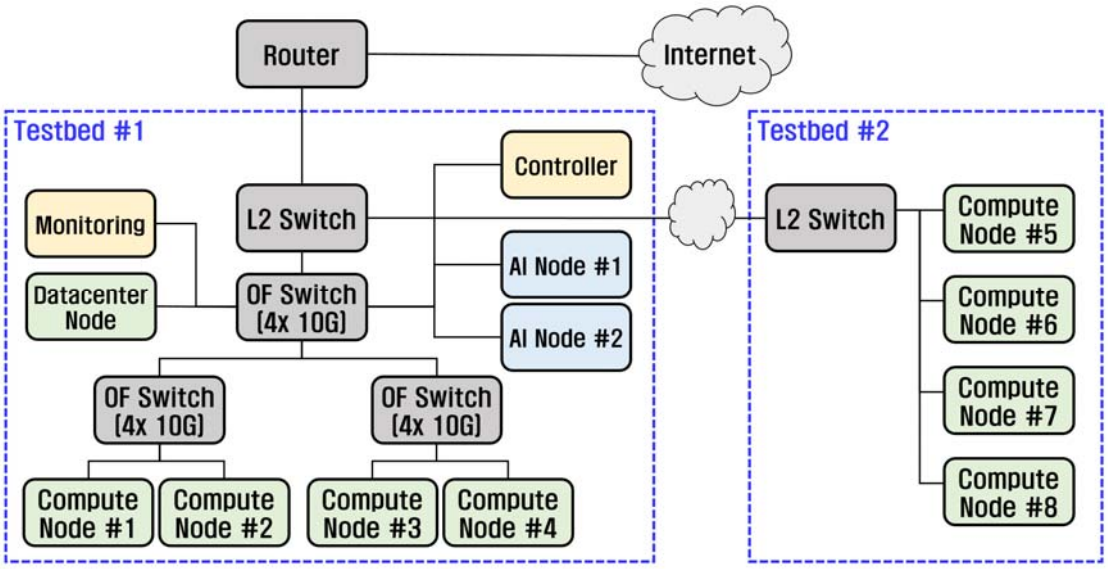
그림 3 VNF Deployment 학습 모델 구조

Hyper-parameter	
Learning rate	0.01
Optimizer	Adam
Batch size	1024
Layer output size	
FNN (request)	30
GNN	15
FNN1 (after concatenation)	200
FNN2 (after concatenation)	200

표 1 VNF Deployment 학습 모델의 하이퍼 파라미터 값

2.3 성능 검증

DQN 기반 Auto-scaling 방법의 성능 검증을 위해 (그림 6)과 같이 OpenStack 기반으로 테스트베드를 구축하였다. 테스트베드는 OpenStack 환경을 관리하는 컨트롤러 노드 (Controller Node)와 VNF 인스턴스가 설치될 수 있는 8개의 컴퓨트 노드 (Compute Node)로 구성된다. 테스트베드 내에 VNF 인스턴스를 설치할 때에는 임의의 컴퓨트 노드를 선택하여 VNF가 설치 된 VM을 생성하며, 각 VM에는 하나의 VNF만 동작한다. 본 문서의 OpenStack 테스트베드 구축을 위해 사용한 OpenStack 설치 스크립트 파일은 Github를 통해 공개되어 있다 [1].



(그림 6) OpenStack 기반 테스트베드

성능 검증은 랜덤한 서비스 정보 리스트를 생성하여 진행하였다. 성능 검증 대상이 되는 VNF들은 Firewall, Flow monitor, DPI, IDS, Proxy이 있다. 이 때, Firewall VNF는 Iptables [2], Flow monitor는 ntopng [3], DPI는 nDPI [4], IDS는 Suricata [5], 그리고 Proxy 기능을 제공하는 VNF는 HAProxy [6]을 사용한다. 본 문서의 VNF Deployment 기능은 서비스 정보 리스트에 해당하는 Service Function Chain (SFC)를 구성한 뒤, ILP와 기계 학습이 해당 SFC에 대해 VNF를 배치하고 트래픽을 발생시켜 VNF를 통해 패킷이 처리되고 전달되는 소요시간을 End-to-End latency로 비교하였다. End-to-end latency는 클라이언트 VM에서 Apache Bench (AB)로 HTTP 요청 메시지를 생성하여 트래픽 요구사항에 해당하는 경로를 통해 목적지인 웹 서버 VM으로 전달하는데, 요청 메시지에 대한 모든 응답 메시지가 클라이언트 VM으로 도달하기까지 소요된 시간을 측정하였다. 실험 결과로 ILP 대비 기계학습 모델의 배치 성능을 평가하였을 때, 모든 트래픽 요구사항에서 계산된 VNF Processing time (예측) 정확도 값이 80% 이상이고, 3회분 평균 91%을 달성하였다.

$$\text{VNF Processing time (예측) 정확도} = 1 - \frac{|[ILP\text{배치 지연 시간}] - [ML\text{배치 지연 시간}]|}{MAX([ILP\text{배치 지연 시간}], [ML\text{배치 지연 시간}])}$$

순번	ILP	ML	VNF Processing Time
1회	39.704 [ms]	32.654 [ms]	82%
2회	32.615 [ms]	32.847 [ms]	99%
3회	159.228 [ms]	176.465 [ms]	90%
평균	77.182 [ms]	80.655[ms]	91%

표 8 3회분 End-to-End Latency 비교

3 Machine Learning 기반 VNF Deployment 모듈 사용법

3.1 환경 구축

본 문서에서 다루는 VNF Deployment 모델은 OpenStack 환경에서 동작할 수 있는 형태로 구현되었다. 앞서 서술한 것처럼, 구현 된 VNF Deployment 모듈은 네트워크 자원 정보를 수집하고 Deployment 결정을 반영하기 위해 OpenStack 테스트베드 내 존재하는 각 VNF 인스턴스들의 데이터를 활용하고 해당 VNF 인스턴스들을 추가 또는 제거할수 있어야한다. 이를 위해 OpenStack 기반 환경의 설치가 필요하며, Deployment 모듈과 상호 연동되는 NFVO와 모니터링 모듈의 설치가 선행되어야 한다. OpenStack 환경 구축을 위한 스크립트 파일 및 설명서 [1]과 NFVO 및 모니터링 모듈 설치를 위한 코드, 설명서 [7]은 Github를 통해 공개되어 있기 때문에 본 문서에서 해당 요소들의 설치 과정은 생략한다. OpenStack 환경이 구축되고, NFVO와 모니터링 모듈이 설치한 후에는 Deployment 모듈을 설치해서 실행할 수 있다. 본 문서에서 모듈 설치와 실행은 Ubuntu 16.04에서 수행하였다.

Deployment 모듈은 Python으로 작성되었으며, Python v3.5.2 이상에서 실행하는 것을 권장한다. Deployment 모듈은 Tesnorflow-gpu v1.5.0과 Keras v2.2.0 라이브러리를 사용하고 있으며, 해당 버전을 사용하는 것을 권장한다. Tensorflow-gpu의 경우 CUDA v10.1 와 CUDNN v7.5 설치가 필요하며, 학습 속도의 저하를 감수할시, 일반 Tensorflow도 사용이 가능하다.

Deployment 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 Deployment 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 Deployment 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
sudo pip3 install virtualenv

# virtualenv를 통한 가상 환경 생성 후 활성화
virtualenv deployment
source deployment/bin/activate

# 가상 환경 비활성화
deactivate
```

3.2 VNF Deployment 모듈 설치 및 실행

앞선 과정에서 pip3 설치까지 완료하고 가상 환경까지 활성화한 후에 Deployment 모듈 설치를 진행한다. Github [8]에 공개 되어있는 Deployment 모듈을 내려 받고, 실행에 필요한 패키지를 설치한다. Deployment 모듈은 Integer Linear Programming 관련 패키지가 필요하며 해당 패키지는 requirements.txt 파일에 정리되어 있으며, 이를 사용하여 패키지를 설치한다. 패키지를 설치 한 후에는 Python 명령어를 통해 Deployment 모듈을 실행한다. 아래 명령어를 순서대로 입력하면 Deployment 모듈 내려 받기, 필수 패키지 설치, 모듈 실행까지 수행할 수 있다. 참고로, Deployment 모듈은 spektral 패키지를 이용하여 GNN을 구현하였기 때문에 해당 패키지를 설치하여야 한다. 또한, 해당 패키지는 keras와 tensorflow 버전에 민감하기 때문에 정확한 버전을 설치할 필요가 있다.

```
# Auto-scaling 모듈 다운로드 후 필수 패키지 설치
git clone https://github.com/dpnm-ni/ni-vnf-deployment-module
cd ni-vnf-deployment-module
sudo pip3 install -r requirements.txt

# Auto-scaling 모듈 실행
python3 -m server
```

Deployment 모듈 실행 명령어를 입력하면 Flask 기반으로 웹 서버가 동작한다. Deployment 모듈은 네트워크 자원을 실시간으로 수집하면서 VNF Deployment 요청을 처리하기 위해 웹 서버로 동작하며 메시지를 처리한다. Auto-scaling 모듈에서 설정되어있는 기본 웹 서버 포트는 8004이며, 포트 변경을 원할 경우에는 내려 받은 Deployment 모듈 폴더의 server/__main__.py 파일에서 (그림 11)과 같이 port라고 명시 된 부분을 수정한다.

```
def main():
    app = connexion.App(__name__, specification_dir='./swagger/')
    app.app.json_encoder = encoder.JSONEncoder
    app.add_api('swagger.yaml', arguments={'title': 'NI VNF Sub-Module Service'})
    app.run(port=8888)
```

(그림 11) Deployment 모듈 server/__main__.py 파일 - Port 설정 가능

Deployment 모듈은 VNF를 Deploy하는 과정에서 VNF 인스턴스의 생성 및 제거, SFC의 갱신을 위해 NFVO 및 모니터링 모듈과 상호작용한다. 이를 위해 설정 파일에 각 모듈이 동작하는 Host의 IP 주소와 포트 번호를 미리 입력해야 한다. 또한, VNF Deployment 과정에서 필요로 하는 파라미터 (Parameter)를 미리 정의할 필요가 있다. 내려 받은 Deployment 모듈 폴더 내에서 config/config.yaml을 열면, 모니터링 모듈에 해당하는 ni_mon, NFVO에 해당하는

ni_nfvo의 host 정보를 각각 수정하고 저장한다. 또한, 그 외에 Github 페이지 [A]에 있는 문서를 참조하여 (그림 12)의 파라미터들을 설정한다.

```
ni_mon:
  host: http://141.223.82.66:8383
ni_nfvo:
  host: http://141.223.82.66:8181
```

(그림 12) config/config.yaml 파일 설정

3.3 VNF Deployment 모듈 사용법

Deployment 모듈이 정상적으로 실행되고, 설정에 오류가 없을 경우 Auto-scaling 모듈은 그림 14처럼 제공되는 Web UI를 확인할 수 있다. 접속 경로는 “http://<Auto-scaling 모듈이 실행되는 Host IP>:<모듈 포트 번호>/ui” 이다.

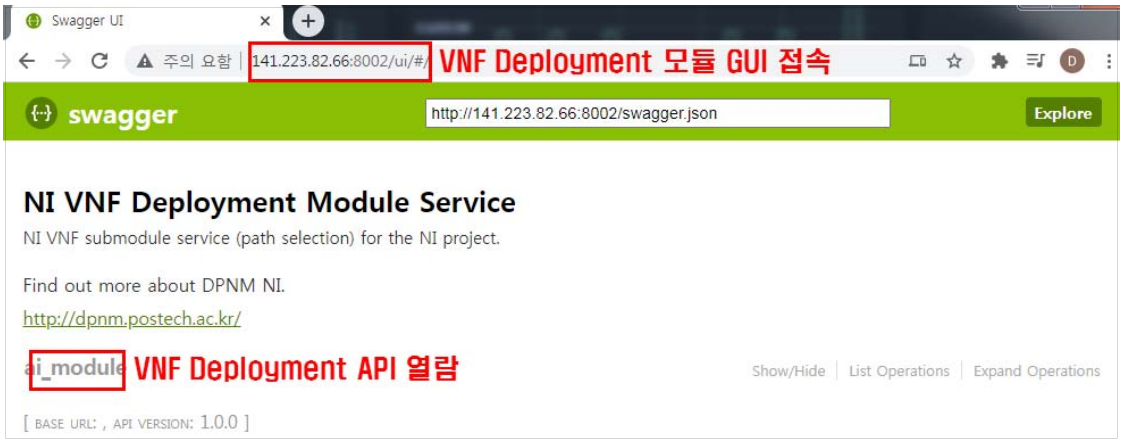


그림 11 VNF Deployment 모듈 GUI

VNF Deployment 모듈은 총 7가지의 기능을 제공하며 해당 기능은 HTTP 메시지를 통해 요청 받는다. 모델이 제공하는 기능은 그림15와 같다.

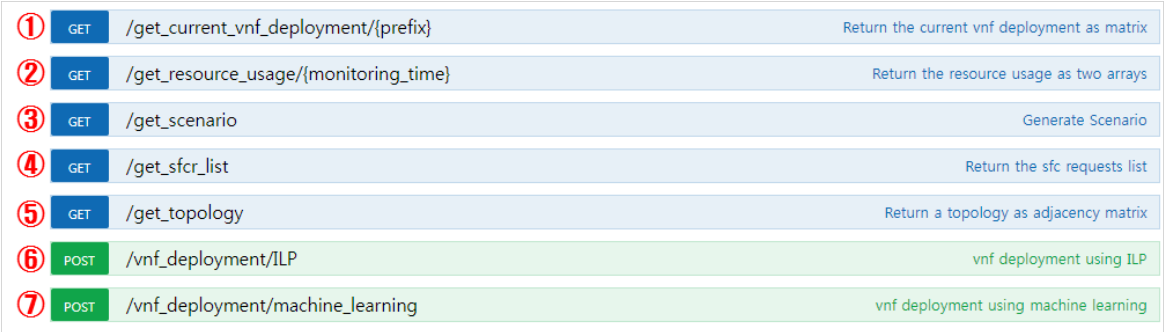


그림 12 VNF Deployment 모듈 기능

모듈은 ①현재 설치된 VNF 정보 요청, ②네트워크 자원 정보 요청, ③시나리오 생성, ④생성

된 SFC 정보 요청, ⑤토폴로지 구성 정보 요청, ⑥ILP을 이용한 VNF 배치, ⑦학습 모델을 이용한 VNF 배치 기능을 제공한다. ③시나리오 생성 기능은 VNF Deployment 모듈을 테스트하기 위해 랜덤 트래픽을 생성하는 것으로 해당 기능은 실행하게 되면 ILP 입력 데이터(solver_src/logs/traf-testbed-8-00002), ML 입력 데이터(X1_sfc1_ni.pickle), VNF 생성 준비 데이터(traf.txt), 총 세 개의 파일이 생성된다. VNF를 직접적으로 Deploy 하는 기능은 ⑥ILP을 이용한 VNF 배치 기능과 ⑦학습 모델을 이용한 VNF 배치 기능이다. 각 기능을 실행하면 VNF 배치를 그림16과 같이 출력하여 보여주며 Openstack 환경에서 VNF가 실제로 배치된 것을 그림17과 같이 확인 할 수 있다.

그림 13 VNF 인스턴스 배치 결정

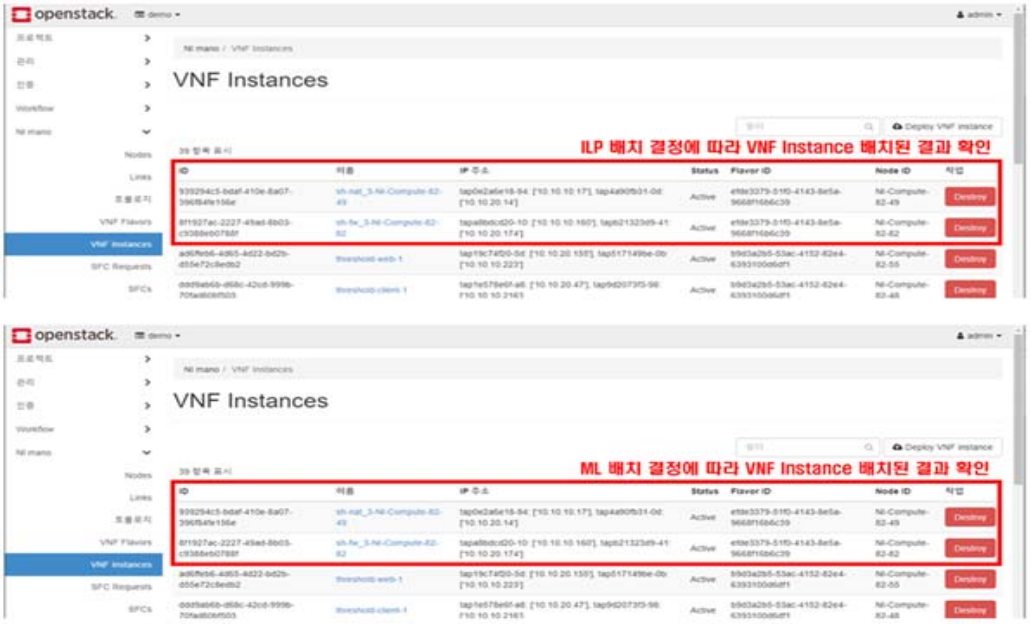


그림 14 Openstack 내 VNF 인스턴스 배치 결과 확인

4 결론

본 문서에서는 NFV 환경의 VNF Deployment에 적용할 수 있는 기계 학습 기반 VNF Deployment 방법에 대해 서술하였다. 본 문서에서 서술한 방법은 네트워크 운용 비용을 다각적인 측면에서 계산하였으며, 네트워크 운용 비용이 최소가 되도록 VNF의 인스턴스 수를 조절하였다. VNF 인스턴스는 네트워크의 노드에서 추가/제거 되었으며, 본 문서의 VNF Deployment 방법은 OpenStack 환경에서 동작할 수 있는 모듈 형태로 구현되었다.

향후 연구로는 현재 모델에서 사용한 GNN (Graph Neural Network)을 효과적으로 활용하기 위하여 다양한 네트워크 토폴로지 데이터 및 토폴로지에 장애가 발생한 경우의 데이터를 수집하여 학습을 할 예정이다. 해당 데이터 수집 및 학습이 수행된다면 현재 데이터 셋으로 학습한 모델보다 훨씬 범용성이 있고 정확도가 높은 모델이 만들어 질 것으로 예상된다.

5 참고문헌

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

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

[3] 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.

[4] 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.

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

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

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

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