NI 기술 문서

딥러닝 기반 Attack&Intrusion Detection 모델

문서 ID	NI-Attack&IntrusionDetection-01
문서 버전	v1.3
작성일	2022.11.29
작성자	포항공과대학교 홍지범

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
2022.11.08	v1.0	초안 작성	홍지범
2022.11.09	v1.1	오타 수정	홍지범
2022.11.28	v1.2	내용 수정 및 정리	홍지범
2022.11.29	v1.3	오타 및 그림 수정	홍지범

목 차

요 약 문

현대의 네트워크는 급변하는 서비스 요구사항을 만족시키기 위해 보다 유연하고 민첩한 네트워크 구축 및 관리가 요구되고 있다. 네트워크 기능 가상화 (NFV, Network Function Virtualization)는 이러한 요구사항을 실현하기 위한 기술 중 하나로, 기존 전용 하드웨어를 통해 제공되는 네트워크 기능을 소프트웨어 형태로 구현하여 상용 서버에서 가상 네트워크 기능 (VNF, Virtual Network Function)으로 운영하는 것이다. NFV 개념이 제안된 이후 통신 사업자와 서비스 제공 업체는 해당 기술을 활용하여 네트워크 구축 및 서비스를 보다 효율적으로 제공하려고 노력하고 있다. NFV 기술은 네트워크 기능을 소프트웨어화 및 가상화하여 네트워크에 유연하게 적용할 수 있는 장점이 있지만, 수많은 가상 자원을 생성하기 때문에 네트워크가 더욱 복잡해져 새로운 관리 문제를 발생시킨다. 물리 및 가상 네트워크의 관리 문제가 발생할 경우 네트워크 관리자 (사람)가 매번 직접 수작업으로 해결하는 것은 한계가 존재하기 때문에, 최근에는 사람의 개입 없이 인공지능 (AI, Artificial Intelligence)을 적용하여 네트워크를 관리하려는 연구들이 활발하게 이루어지고 있다.

Attack&Intrusion Detection은 NFV 환경에서 운용되는 VNF들의 라이프사이클 (Life-cycle) 관리 기능 중 하나로, NFV 환경에서 운용되는 물리 및 가상 네트워크에 네트워크 공격 (Network Attack) 및 침입 행위 (Intrusion)와 같은 비정상적인 트래픽을 탐지하는 것이다. 따라서 본 문서의 Attack&Intrusion Detection은 DoS (Denial of Service)와 같은 다양한 네트워크 공격이나 Bruteforce와 같은 침입 행위들을 비정상 트래픽으로 정의하고, 이를 탐지하는 것을 목표로 한다. 그리고 상기 네트워크 공격 및 침입 트래픽을 탐지하기 위한 딥러닝 (Deep Learning) 기반 모델을 기술하고, 제안하는 모델의 성능을 검증하고 분석한다.

1 서론

1.1 개요

현대의 네트워크는 소프트웨어 정의 네트워킹 (SDN, Software-Defined Networking) 및 네트워크 기능 가상화 (NFV, Network Function Virtualization) 등 새로운 기술을 기반으로 네트워크를 설계, 구축 및 운영하고 있다. 통신 사업자와 서비스 제공 업체는 기존의 폐쇄된 네트워크 기능을 소프트웨어화한 VNF (Virtual Network Functions)로 대체하여 CAPEX 및 OPEX를 줄이는 것이 가능하다 [1]. 이에 따라 가상 네트워크에서 작동하는 VNF (또는 CNF, Cloud-native Network Function) 및 서비스의 수가 크게 증가하고 있지만 반대로 이로 인해 물리 및 가상 네트워크를 더욱 복잡해졌다. 따라서 이로 인해 성능 저하 문제, 서비스 오류 또는 시스템 과부하 문제들이 발생한다. 이러한 네트워크 관리 문제를 해결하기 위해 오케스트레이션, 성능, 보안 등과 관련된 몇 가지 요구사항 [2]이 정의되었으며, 대표적으로 자원 관리, 장애 관리 등을 예로 들 수 있다.

기존 네트워크 관리는 네트워크 관리자가 수동으로 네트워크 관리를 직접 수행했지만 네트워크 및 서비스 요구사항이 다양해지고 복잡해짐에 따라 서비스의 배포/변경에 더 많은 시간이 소요된다. 그리고 세부 네트워크 구성에 대한 전문적인 인력이 필요하게 되었다. 그러나 소규모 네트워크가 아닌 대규모 네트워크, 혹은 복잡한 종속성을 가지고 있는 네트워크의 경우 네트워크 관리자가 이를 모두 해결하는 것은 상당한 시간과 비용을 소모하기 때문에 기존과는 다른 접근 방식이 필요하게 되었다. 이에 최근 인공지능 기술 (머신러닝 또는 딥러닝)을 적용한 가상 네트워크 관리에 대한 연구가 주목을 받고 있다. 이러한 연구들은 시스템 혹은 네트워크 관리자가 네트워크의 실시간 상태를 바탕으로 네트워크를 관리하기 위한 정책(Policy)이나 의사결정(Decision)에 도움을 주거나, 나아가 사람의 개입 없이 네트워크 상태를 이해하고 네트워크를 최적으로 관리한다 [3]. 이러한 연구들과 더불어 ETSI ZSM (Zero-touch network and Service Management) [4] 이나 TM Forum의 Autonomous Networks Project [5]와 같이 표준화 작업도 활발하게 진행되고 있다.

1.2 Attack&Intrusion Detection

Attack&Intrusion Detection은 보안 및 장애 관리 기술에 속하며, 본 기술문서와 연관된 과제 (인공지능 기반 가상 네트워크 관리기술 개발)에서는 NFV 환경에서 물리 및 가상 자원으로 유입되는 네트워크 공격 또는 침입 트래픽을 탐지하는 기능을 의미한다. 여기서 자원에 대한 정의는 물리 서버 및 네트워크와 가상 머신 (Virtual Machine, VM)이나 컨테이너 (Container)에서 동작하는 VNF 및 CNF, 그리고 가상 네트워크를 포함한다. 그리고 네트워크 공격 및 침입 트래픽은 DoS/DDoS와 같은 서비스 거부 공격과 공격을 준비하기 위한 port scan과 같은 공격 등 다양한 원인에 의해 발생한다. 기존 비정상 트래픽 분류 및 공격 탐지

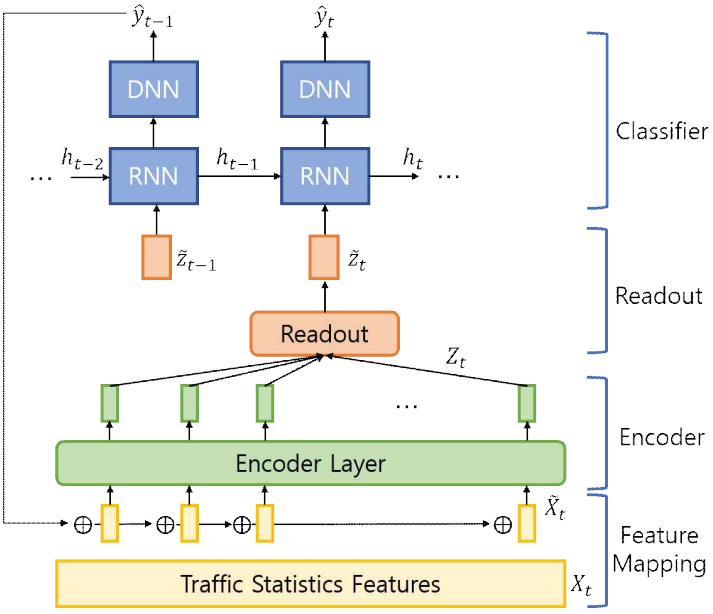
방법은 잘 알려진 포트 번호를 이용한 포트 기반 탐지 방법과 패킷의 페이로드 (Payload)에 포함된 signature를 이용한 페이로드 기반 탐지 방법을 사용하여 공격 및 침입 트래픽을 탐지하였다. 그러나 이와 같은 기존 방법들은 네트워크 공격 방법이 광범위하게 다양화되어 현재 그 한계가 존재한다 [6].

따라서 본 문서에서는 이러한 NFV 환경에서 운용되는 물리 및 가상 자원을 보호하기 위해 네트워크 공격 및 침입 트래픽을 탐지하는 방법과 그 모델에 대해 서술한다. 제안하는 방법은 머신러닝 및 딥러닝 알고리즘을 기반으로 지도학습을 이용하여 학습된 탐지 모델을 사용한다. 그리고 제안된 모델을 네트워크 공격 및 침입 관련 공개 데이터셋을 이용해 그 성능을 검증한다.

2 딥러닝 기반 Attack&Intrusion Detection 방법

2.1 딥러닝 기반 Attack&Intrusion Detection 모델

본 문서에서 제안하는 딥러닝 기반 Attack&Intrusion Detection 모델은 (그림 1)과 같이 설계되어 있다. 해당 모델은 네트워크 트래픽 통계 정보 (Network Traffic Statistics)를 입력으로 받아 네트워크 공격이나 침입 등 비정상 트래픽을 정상 트래픽과 비교하여 분류한다. 이러한 탐지(분류) 문제는 출력 $\hat{y}^t$를 생성하는 매개변수 함수 $f(X_t;\theta)$로 볼 수 있다 (X는 입력값, t는 시간 인덱스, θ는 매개변수를 의미한다). 이때, 이진분류 (binary classification)의 경우 출력은 정상 트래픽과 비정상 트래픽을 나타내는 0과 1로 구분되며, 다중 분류 (multi-class classification)의 경우에는 클래스별 숫자값으로 구분된다.



(그림 1) 딥러닝 기반 Attack&Intrusion Detection 모델

Attack&Intrusion Detection 모델이 사용하는 딥러닝 알고리즘은 네트워크 트래픽의 시계열 특성을 보다 잘 학습할 수 있도록 현재 시간 인덱스의 입력과 이전 시간 인덱스의 입력값을 모두 고려한다. 따라서 현재 시간 t에 대한 입력은 $\overset{t}{\underset{t-l+1}{X}} = [X_{t-l+1}, X_{t-l+2}, \cdots, X_t]$이며, 여기서 l은 윈도우 사이즈 (학습 시 지정한 입력 시퀀스 길이)를 의미한다. 제안된 모델은 이와 같은 입력 시퀀스를 기반으로 학습을 진행하며, 학습 및 분류는 (그림 1)과 같이 총 4개의 계층을 통해 진행된다.

먼저, Feature Mapping 계층은 활성화 함수 (activation function)이 없는 fully connected

layer로 구성되어 입력 벡터 X_t를 $\widetilde{X_t} \in R^D$로 매핑한다. 여기서 D는 입력값의 차원(dimension)을 의미한다. 그리고 입력값으로 활용되는 데이터는 네트워크 트래픽 플로우(flow)의 통계 정보를 포함하는 각 feature 값들로 구성된다. 따라서 이를 수식화하면 아래 수식과 같이 정의된다.

$$\widetilde{X_t} = feature\_mapping(X_t; \theta_{mapping})$$

다음으로, Encoder layer는 feature mapping layer에서 처리되어 Encoder layer로 입력된 전체 데이터를 축소시킨 벡터로 변환한다. 이 과정에서 활용되는 알고리즘은 RNN (Recurrent Neural Network) 계열의 알고리즘인 기본 RNN과 Bi-directional RNN (Bi-RNN) [6] 이나 Encoder/Decoder 구조를 지니는 Transformer [7] 이다. 이를 통해 전체 데이터를 축소하여 입력 벡터의 크기를 줄임으로써 학습이 보다 용이하게 이루어질 수 있도록 한다. Encoder layer에서 활용되는 알고리즘은 사용자가 학습 시 Encoder layer의 매개변수 지정을 통해 3개 알고리즘 중 1개를 선택하여 학습을 진행할 수 있다. 위와 같은 Encoder layer의 학습 과정을 수식화하면 아래 수식과 같이 정의된다. 여기서 Z_t는 Encoder layer를 거쳐 출력된 데이터를 나타낸다.

$$Z_t = Encoder(\widetilde{X_t}; \theta_{encoder})$$

그리고 Readout layer는 데이터를 하나의 벡터로 축약하는 역할을 수행한다. 해당 계층에서 사용되는 readout 함수는 min/max/self-attention 3개 함수가 존재한다. 해당 함수들은 Encoder layer에서와 마찬가지로 매개변수 설정을 통해 3개 함수 중 1개 함수를 사용자가 선택할 수 있다. Encoder layer에서 RNN이나 Bi-RNN 알고리즘을 사용하는 경우 max/min 함수와 같이 일반적으로 많이 사용되는 pooling 방법을 사용하고, Transformer를 사용하는 경우 self-attention을 통해 가중치 합 (weighted sum)을 활용하여 Encoder layer에서 출력된 벡터 집합을 축약한다. Readout layer의 출력은 $\widetilde{Z_t}$로 표현되며, 이에 대한 수식은 아래와 같이 정의된다.

$$\widetilde{z_t} = Readout(Z_t; \theta_{readout})$$

마지막 classifier layer는 RNN과 DNN이 결합된 형태로 구성된다. 해당 layer는 시간 단계에 따라 먼저 RNN 모델은 입력 시퀀스 $[\widetilde{z}_{t-l+1}, \widetilde{z}_{t-l+2}, \cdots, \widetilde{z}_t]$을 받아 hidden state $[h_{t-l+1}, h_{t-l+2}, \cdots, h_t]$를 출력한다. 그 후, DNN은 RNN의 hidden state h_t를 입력으로 받아 탐지 결과 $\widetilde{y}_t = DNN(h_t)$를 출력한다. 해당 과정의 순차적인 식은 다음과 같이 정의된다:

$$\overset{t}{\underset{t-l+1}{h}} = RNN(\overset{t}{\underset{t-l+1}{\widetilde{z}}}; \theta_{RNN}),$$

$$\hat{y_t} = DNN(h_t; \theta_{DNN})$$

그리고 classifier layer에서 분류를 위한 목적함수 J는 다음과 같이 주어진다:

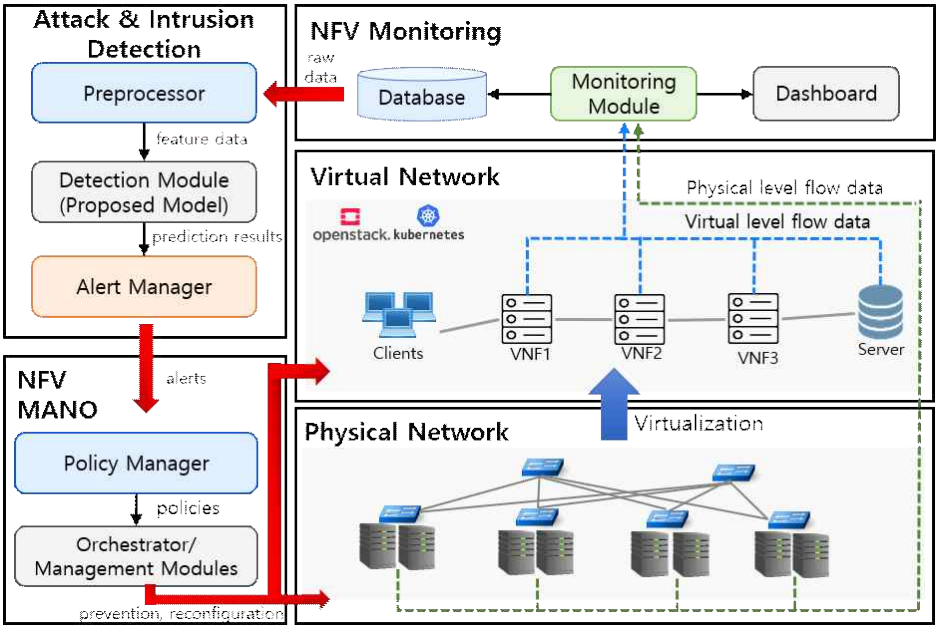
$$J(y_t, \hat{y_t}) = -y_t \log(\hat{y_t}) - (1 - y_t) \log(1 - \hat{y_t})$$

또한, 제안하는 모델의 성능 향상을 위해 이전 시간 단계의 예측 결과를 사용한다. 네트워크 공격 및 침입과 같은 비정상적인 이벤트는 이전 시간 인덱스에서 발생한 이벤트와 높은 상관관계가 있다 [4], [33]. 따라서 제안하는 모델은 선행 시간 인덱스에서의 예측 결과를 현재 시간 인덱스의 트래픽 분류에 활용하며, 현재 입력 X_t와 함께 이전 예측 결과 $\hat{y}_{t-1}$를 사용하도록 모델의 입력을 다음과 같이 수정한다:

$$X_t = \hat{y}_{t-1} \oplus X_t$$

2.2 Attack&Intrusion Detection 시스템 설계

상기 설명한 Attack&Intrusion Detection 모델을 실제 네트워크 환경에서 구현 및 적용하기 위한 네트워크 공격 및 침입 탐지 시스템의 설계는 (그림 2)와 같다. 제안된 설계는 본 연구 과제를 기반으로 NFV 환경에서 운용이 가능한 물리 네트워크와 가상 네트워크를 타겟으로 한다.



(그림 2) 네트워크 공격 및 침입 탐지 시스템 구조도

NFV 환경은 서버 및 스위치를 활용하여 물리 네트워크를 구성하고, 이후 OpenStack/Kubernetes와 같은 VIM (Virtual Infrastructure Manager)를 이용하여 가상 네트워크

를 구축한다. 이러한 가상 네트워크에서 풀/CNF 인스턴스들을 동작시켜 서비스를 운용한다. NFV 환경에서 모니터링 기능은 물리 및 가상 서버의 CPU, 메모리와 같은 자원 사용 정보와 네트워크 트래픽 정보, syslog와 같은 로그 정보를 포함한다. 이때 Attack&Intrusion Detection 모델에 사용되는 네트워크 트래픽 정보는 플로우 단위로 수집한 통계정보를 포함하며, 수집된 데이터는 모니터링 모듈로 전달되어 데이터를 대시보드를 통해 실시간 모니터링 정보를 시각화하여 제공하거나, 데이터베이스에 시계열 형태의 데이터로 저장된다.

물리 및 가상 네트워크에 대한 모니터링 모듈을 통해 수집된 트래픽 통계 데이터는 데이터베이스에 저장된 데이터 형태로 공격 및 침입 탐지 모듈에 전달된다. 이 raw 데이터는 전처리기를 통해 모듈화된 탐지 모델에서 사용할 수 있는 형식으로 변환된다. 다음으로, 공격 및 침입 탐지 모듈은 전처리된 데이터를 입력으로 사용하고 탐지 결과를 출력한다. 트래픽 데이터는 정상 또는 네트워크 공격/침입으로 분류한다. 이러한 탐지 결과를 기반으로 Alert Manager는 정책 관리자에게 네트워크 공격 또는 침입 트래픽이 발생했음을 알린다.

이후 전체 네트워크 환경에 대한 관리 정책을 결정하는 Policy Manager는 공격 대상 주소에 대한 보안 규칙을 업데이트하거나 탐지 결과에 따라 서비스 품질 또는 보안 수준을 유지하기 위한 새로운 관리 정책을 생성한다. 마지막으로 정책 관리자는 관리 정책을 Orchestrator로 전송하여 네트워크에 정책을 반영한다. 필요한 경우 Orchestrator는 이때 Auto-Scaling 또는 Live Migration과 같은 다른 관리 모듈과 함께 네트워크를 관리한다. 제안된 방법을 통해 네트워크 공격 및 탐지를 위한 통합적인 closed-loop automation으로 구성된 시스템을 기대할 수 있다.

3 성능 검증

3.1 성능 검증 방법

본 연구에서 제안하는 공격 및 침입 탐지 모델의 성능을 평가하기 위해, 네트워크 공격 및 침입과 관련된 공개 데이터셋을 사용하여 성능을 검증한다. 성능 검증을 위한 데이터셋은 NSL-KDD [7], UNSW-NB15 [8] 및 CICIDS 2017 [9]의 3가지 데이터셋이 사용된다. NSL-KDD는 IDS에서 수집된 인터넷 트래픽 데이터를 포함하고 있으며, 4가지 공격 종류로 구성된다. UNSW-NB15는 NSL-KDD에 포함되지 않은 low footprint와 같은 공격 시나리오가 포함되어 있으며, 총 9개의 서로 다른 공격 종류로 구성된다. 그리고 CICIDS 2017는 실제 데이터와 유사한 benign (normal) 및 최신 네트워크 공격이 포함되어 있으며, 총 14개의 공격 종류로 구성되어 있다. 상기 3가지 데이터셋에 대한 공격 종류 및 데이터 수 등의 상세 정보는 (그림 3)과 같다.

Datasets		NSL-KDD [5]	UNSW-NB15 [6]	CICIDS 2017 [7]
No. Features		41	47	78
No. Records	Total	Normal (77,054) Abnormal (71,463)	Normal (93,000) Abnormal (164,673)	Normal (2,273,098) Abnormal (557,646)
	Train	Normal (67,343) Abnormal (58,630) - DoS (45,927) - Probe (11,656) - R2L (995) - U2R (52)	Normal (37,000) Abnormal (45,332) - Worms (44) - Analysis (677) - Backdoor (583) - DoS (4,089) - Exploits (11,132) - Fuzzers (6,062) - Generic (18,871) - Reconnaissance (3,496) - Shell Code (378)	Normal (2,273,098) Abnormal (557,646) - FTP-Patator (7,938) - SSH-Patator (5,897) - DoS GoldenEye (10,293) - DoS Hulk (231,073) - DoS Slowhttptest (5,499) - DoS slowloris (5,796) - Heartbleed (11) - XSS (652) - SQL (21) - Brute (1,507) - Infiltration (36) - DDoS (128,027) - PortScan (158,930) - Botnet (1,966)
	Test	Normal (9,711) Abnormal (12,833) - DoS (7,458) - Probe (2,421) - R2L (2,754) - U2R (200)	Normal (56,000) Abnormal (119,341) - Worms (130) - Analysis (2,000) - Backdoor (1,746) - DoS (12,264) - Exploits (33,393) - Fuzzers (18,184) - Generic (40,000) - Reconnaissance (10,491) - Shell Code (1,133)	

(그림 3) 성능 검증에 사용된 공개 데이터셋의 세부 정보

본 연구의 실험은 모델의 성능을 두 가지 방식으로 평가한다. 먼저 이진 분류를 통해 정상 트래픽 데이터와 비정상 트래픽 데이터를 잘 분류할 수 있는지 평가한다. 그리고 분류 문제에서 최근 일반적으로 사용되는 머신러닝 알고리즘 (SVM, Random Forest, XGBoost, DNN)으로 구성된 baseline 모델과 성능을 비교한다. baseline 모델은 Python, Keras, Scikit-learn을 사용하여 구현된다. 다음으로, 보다 상세한 분류 정보를 제공하기 위해 다중 클래스 분류를 수행하여 성능을 분석한다.

NSL-KDD 및 UNSW-NB15의 경우 모델의 성능을 평가하기 위해 학습에 활용하기 위한 training set과 모델의 성능을 평가하기 위한 test set이 제공된다. 따라서 다른 연구와 정확한 성능 비교를 위해 제공된 데이터셋을 사용한다. 해당 데이터셋들은 학습 시 training set을 training set과 validation set으로 나누어 모델의 학습 과정에서의 성능을 1차적으로 검증한다. 하지만 CICIDS 2017의 경우 training set과 test set를 제공하지 않는다. 또한 해당 데이터셋은 4:1의 비율로 정상 및 비정상 데이터의 불균형이 다른 데이터셋에 비해 심하기 때문에 전처리 과정에서 정상 트래픽 데이터에 대해 under-sampling을 수행하여 data imbalance 문제를 약 2:1 정도로 완화시킨다. 그 후, 데이터셋을 75%의 training set과 25%의 test set으로 나눈다. 이때, training set의 경우 NSL-KDD 및 UNSW-NB15와 같은 방법으로 training set과 validation set으로 나누어 학습을 진행한다.

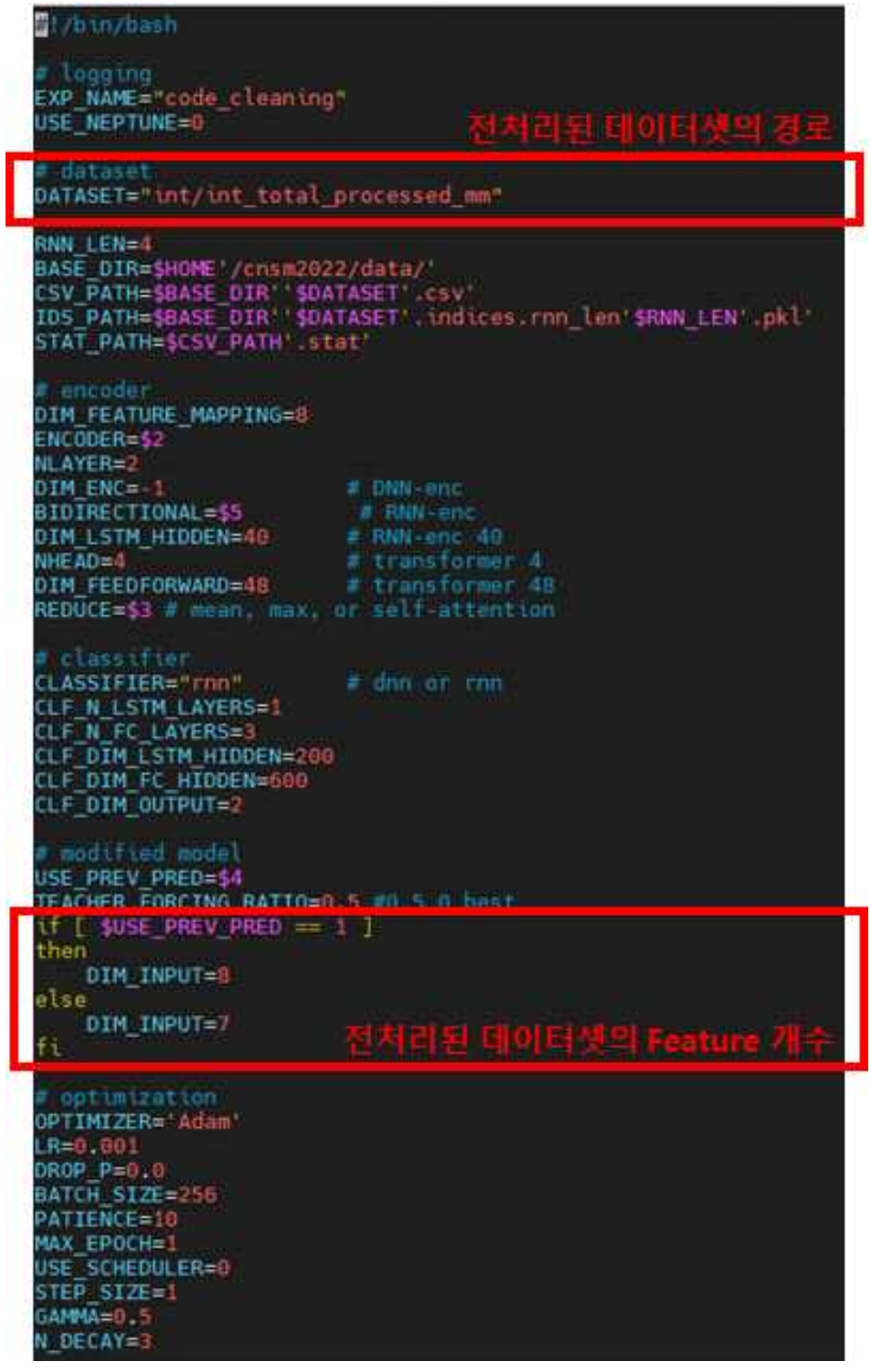
```
#!/bin/bash

DATASET=$1
DATA_DIR=$HOME"/cnsm2022/data/"
RNN_LEN=4
TRAIN_RATIO=0.6
VALID_RATIO=0.2
TEST_RATIO=0.2
```

(그림 4) 데이터셋 전처리 과정 중 train/validation/test 데이터셋 configuration

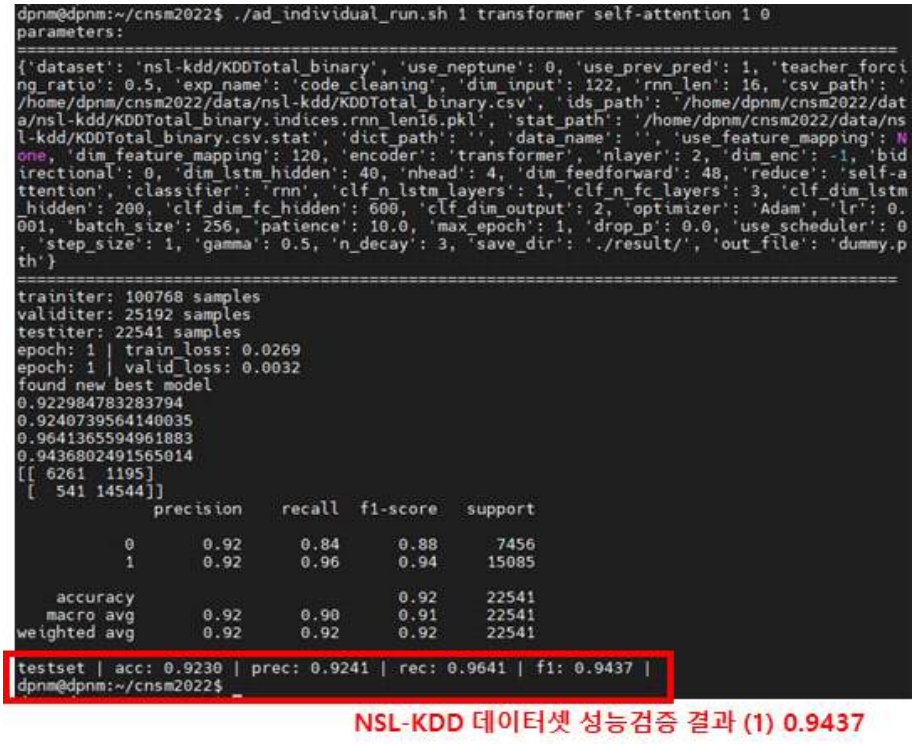
하지만 모든 데이터셋에서 이진 분류에서도 각 레이블 간의 데이터 불균형이 여전히 존재하고, 다중 클래스 분류에서는 그림 1과 같이 불균형이 크게 존재한다. 따라서 정확도(accuracy) 지표를 사용할 경우 데이터의 불균형으로 인해 부정확한 성능 평가가 발생할 가능성이 존재하므로 보다 정확한 성능 평가를 위해 precision과 recall 값의 조화 평균으로 데이터 불균형을 고려하는 지표인 F1 score를 성능 평가에 사용한다. 실험 결과는 실험을 10회 반복하여 F1-score의 평균을 계산한다. 전체적인 configuration 및 실험 과정은 (그림 4~7)과 같다.

```
dpnm@dpnm:~/cnsm2022$ ./ad_split_index.sh int/int_total_processed_mm
/home/dpnm/cnsm2022/data/
int/int_total_processed_mm
0.9
0.001
0.099
1.0
0    43584
1     1341
Name: classification, dtype: int64
0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44925 entries, 0 to 44924
Data columns (total 8 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   duration         44925 non-null  float64
 1   hop_latency      44925 non-null  float64
 2   egress_time      44925 non-null  float64
 3   flow_latency     44925 non-null  float64
 4   sink_time        44925 non-null  float64
 5   port_tx_util     44925 non-null  float64
 6   queue_occupancy  44925 non-null  float64
 7   classification   44925 non-null  int64
dtypes: float64(7), int64(1)
memory usage: 2.7 MB
```

(그림 5) 데이터셋 전처리 과정 중 train/validation/test 데이터셋 분할



(그림 6) 탐지 모델의 학습 파라미터 설정 configuration



(그림 7) 학습 파라미터 설정 후 실험 결과

3.2 성능 검증 결과

Method	Model		NSL-KDD	UNSW-NB15	CICIDS 2017
	Encoder	Readout			
Baseline	SVM		0.6402	0.8391	0.9286
	Random Forest		0.6527	0.8681	0.9858
	XGBoost		0.6977	0.8650	0.9899
	DNN		0.5453	0.8472	0.9198
	R. Vinayakumar et. al [24]		0.8070	0.8200	0.9390
Proposed (w/o previous prediction)	RNN	Max / Mean / Self-attention	0.9253 / 0.9238 / **0.9338**	0.9188 / 0.9167 / **0.9193**	0.9825 / **0.9838** / 0.9812
	Bi-RNN		0.9285 / **0.9364** / 0.9363	0.9206 / 0.9186 / **0.9224**	0.9836 / 0.9841 / **0.9843**
	Transformer		0.9223 / **0.9328** / 0.9267	**0.9215** / 0.9197 / 0.9192	0.9889 / 0.9880 / **0.9890**
Proposed (w/ previous prediction)	RNN	Max / Mean / Self-attention	0.9461/ 0.9403 / **0.9561**	0.9274 / 0.9282 / **0.9287**	**0.9897** / 0.9888 / 0.9894
	Bi-RNN		**0.9552** / 0.9428 / 0.9440	0.9279 / 0.9296 / **0.9377**	0.9913 / 0.9905 / **0.9917**
	Transformer		**0.9563** / 0.9450 / 0.9551	0.9324 / 0.9316 / **0.9346**	0.9952 / **0.9966** / 0.9955

(그림 8) 이진 분류 성능 비교 결과

(그림 8)은 3개 공개 데이터셋에 대한 각 모델의 성능을 설명한다. 먼저, NSL-KDD 데이터셋의 경우 baseline 모델은 0.53~0.69로 낮은 정확도 (F1-score)를 보였다. 관련 연구의 경우 약 0.8070로 가장 높은 성능을 보였다. 반면 본 연구의 모델은 previous prediction 기능을 사용하지 않은 경우, 0.92에서 0.93으로 상대적으로 높은 성능을 보였다. encoder 및 readout layer의 하이퍼파라미터 설정에 따라 F1-score 값이 다소 차이가 있지만, 제안하는 모델은

encoder layer에서 Bi-RNN을 사용하고 readout layer에서 mean 함수를 사용할 때 0.9338의 가장 높은 F1-score를 보였다. 또한, previous prediction 기능을 사용하는 경우 해당 기능을 사용하지 않은 결과에 비해 성능이 향상되었다. 특히, 제안된 모델은 encoder layer에서 Transformer를 사용하고 readout layer에서 max 함수를 사용할 때 0.9563의 가장 높은 성능을 보였다.

다음으로 UNSW-NB15 데이터셋의 경우 NSL-KDD와 달리 baseline 모델의 성능도 최대 0.8681의 성능을 보였다. baseline 모델 중 Random Forest 모델과 XGBoost 모델은 가장 높은 성능을 보였지만 분류 정확도 (F1-score) 0.9를 달성하지 못하였다. 반면 제안하는 모델은 NSL-KDD보다 약간 낮은 성능을 보였으나 baseline 모델의 성능보다 평균적으로 우수하였다 (약 0.91~0.92). previous prediction 기능을 사용하지 않은 경우 encoder 및 readout layer configuration 중 Bi-RNN과 self-attention을 사용한 결과가 0.9224로 가장 높은 성능을 보였으며, 해당 기능을 사용할 경우 기존보다 개선된 성능을 보였다. 특히 Bi-RNN과 self-attention을 적용한 모델이 0.9377로 가장 성능이 우수하였다.

마지막으로 CICIDS 2017 데이터셋은 대부분의 모델이 높은 분류 정확도를 보였다. baseline 알고리즘의 경우 정확도가 약 0.9 이상이었고, 특히 Random Forest, XGBoost와 같은 DT 계열의 알고리즘은 0.98 이상의 높은 성능을, DNN 모델은 약 0.92의 성능을 보였다. 본 연구의 모델은 previous prediction 기능을 사용하지 않았을 때 0.98 이상의 분류 정확도를 보였고, 이러한 결과는 DT를 기반으로 한 baseline 모델 (Random Forest, XGBoost)과 유사하였다. 또한, previous prediction을 사용했을 경우 약 0.99 이상의 F1-score를 보였다. 특히, encoder layer에서 Transformer를 사용하고 readout layer에서 mean 함수를 사용할 때 0.9966으로 가장 높은 분류 성능을 보였다. 따라서 성능 측정 결과, 제안하는 모델이 previous prediction 기능을 사용할 경우 가장 성능이 우수하였다.

다중 클래스 분류 실험은 정상 트래픽 데이터의 포함 유무에 따른 분류 성능을 측정하였다. 먼저 (그림 9)는 4개의 공격 클래스를 가지는 NSL-KDD 데이터셋에 대한 previous prediction 기능과 함께 제안된 모델로서 공격 유형에 따른 다중 클래스 분류 결과를 보여준다. 정상 트래픽 데이터와 비정상 트래픽 데이터를 함께 분류했을 때 정상 트래픽에 대한 분류 정확도는 약 0.8로 나타났다. 비정상 트래픽 데이터의 경우 DoS는 약 0.9, Probe는 약 0.82를 보인 반면, 데이터 수가 적은 클래스인 R2L (Root to Local)과 U2R (User to Root) 공격은 약 0.2의 낮은 분류 정확도를 보였다.

Training setting	Multi-class classification (w/ normal data)			Multi-class classification (w/o normal data)		
Label (No. Record in test set)	RNN (Max / Mean / Self-attention)	Bi-RNN (Max / Mean / Self-attention)	Transformer (Max / Mean / Self-attention)	RNN (Max / Mean / Self-attention)	Bi-RNN (Max / Mean / Self-attention)	Transformer (Max / Mean / Self-attention)
DoS (7,456)	0.894 / **0.901** / 0.894	0.893 / 0.882 / **0.902**	0.897 / 0.889 / **0.899**	0.912 / 0.906 / **0.914**	0.920 / 0.912 / **0.937**	0.897 / 0.902 / **0.905**
Probe (2,421)	0.787 / **0.811** / 0.805	**0.834** / 0.816 / **0.834**	**0.796** / 0.772 / 0.763	**0.832** / 0.820 / 0.807	0.760 / **0.834** / 0.822	**0.813** / 0.810 / 0.808
R2L (2,754)	0.233 / 0.182 / **0.290**	0.200 / 0.185 / **0.224**	0.180 / **0.215** / 0.197	**0.737** / 0.722 / 0.724	**0.736** / 0.720 / 0.703	0.692 / **0.703** / 0.696
U2R (200)	**0.172** / 0.153 / 0.105	0.070 / **0.126** / 0.080	0.160 / 0.144 / **0.218**	0.190 / **0.205** / 0.193	0.172 / 0.208 / **0.216**	0.272 / 0.256 / **0.286**
Normal (9,710)	**0.812** / 0.810 / 0.808	0.802 / 0.804 / **0.826**	0.802 / 0.793 / **0.807**	-	-	-

(그림 9) 다중 클래스 분류 성능 비교 결과 (NSL-KDD)

또한 정상 트래픽 데이터와 비정상 트래픽 데이터에 대해 이진 분류를 먼저 수행하는 시나리오를 가정하고 비정상 트래픽 데이터에 대해 다중 클래스 분류를 수행하였다. 그 결과 DoS 및 Probe 공격의 평균 성능이 약간 향상되었다. 그리고 R2L 공격의 분류 정확도는 약 0.2에서 0.7로 크게 향상되었으며, 전반적으로 정상 트래픽 데이터를 포함하지 않았을 때 더 우수한 성능을 보였다.

다음으로, (그림 10)은 UNSW-NB15 데이터셋에 대한 다중 클래스 분류 결과를 나타낸다. 정상 트래픽 데이터를 포함하여 분류를 수행한 경우, 정상 클래스의 분류 정확도는 약 0.93이었다. 여러 공격 클래스 중 Generic 클래스가 0.976으로 가장 높은 분류 정확도를 보였고, Exploits, Fuzzers, Reconnaissance 클래스가 약 0.7 정도의 정확도를 보였다. 하지만 제안하는 모델은 학습 데이터가 매우 적은 3가지 공격 클래스 (training dataset에서 Worms: 44개, Analysis: 677개, Backdoor: 583개 데이터 인스턴스)를 잘 분류하지 못했다.

Training setting	Multi-class classification (w/ normal data)			Multi-class classification (w/o normal data)		
Label (No. Record In test set)	RNN (Max / Mean / Self-attention)	Bi-RNN (Max / Mean / Self-attention)	Transformer (Max / Mean / Self-attention)	RNN (Max / Mean / Self-attention)	Bi-RNN (Max / Mean / Self-attention)	Transformer (Max / Mean / Self-attention)
DoS (12,264)	**0.434** / 0.389 / 0.411	**0.435** / 0.425 / 0.393	**0.421** / 0.412 / 0.400	0.602 / 0.642 / **0.657**	0.593 / 0.610 / **0.621**	**0.632** / 0.595 / 0.620
Exploits (33,393)	**0.701** / 0.694 / 0.690	0.655 / 0.650 / **0.682**	**0.683** / 0.679 / 0.676	0.732 / 0.728 / **0.740**	**0.738** / 0.712 / 0.703	**0.781** / 0.770 / 0.752
Fuzzers (18,184)	**0.660** / 0.644 / 0.623	0.605 / **0.640** / 0.638	**0.646** / 0.622 / 0.610	0.730 / 0.740 / **0.751**	0.720/ **0.742** / 0.725	0.725 / 0.740 / **0.761**
Generic (40,000)	**0.966** / 0.950 / 0.953	0.933 / **0.954 / 0.954**	0.942 / 0.952 / **0.976**	0.928 / 0.947 / **0.950**	**0.953** / 0.942 / 0.942	**0.942** / 0.935 / 0.933
Reconnaissance (10,491)	**0.596** / 0.576 / 0.553	0.602 / **0.630** / 0.594	0.622 / **0.643 / 0.640**	0.626 / 0.630 / **0.672**	**0.688** / 0.662 / 0.654	0.660 / **0.687** / 0.658
Shell code (1,133)	0.025 / **0.032 / 0.032**	0.014 / 0.020 / **0.052**	0.215 / **0.301** / 0.260	0.120 / 0.132 / **0.155**	0.127 / 0.136 / **0.209**	0.200 / 0.252 / **0.335**
Normal (55,989)	0.930 / 0.923 / **0.937**	0.926 / **0.938 / 0.938**	**0.945** / 0.932 / 0.937	-	-	-

(그림 10) 다중 클래스 분류 성능 비교 결과 (UNSW-NB15)

그리고 정상 트래픽 데이터를 포함하지 않았을 때, NSL-KDD 데이터셋의 결과와 유사하게 평균적으로 성능이 향상되었다. 데이터셋에 존재하는 여러 공격 클래스 중 Generic의 경우 평균 정확도가 비슷했지만, DoS, Exploits, Fuzzers, Reconnaissance, Shell code의 경우 평균 성능 (F1 score)이 약 0.1에서 0.2 정도 향상된 것을 확인하였다. 하지만 Shell code는 약 0.3의 낮은 성능을 보였다. 이러한 결과는 학습 데이터가 매우 적을 때의 NSL-KDD 데이터셋 실험 결과와 유사하였다.

Training setting	Multi-class classification (w/ normal data)			Multi-class classification (w/o normal data)		
Label (No. Record In test set)	RNN (Max / Mean / Self-attention)	Bi-RNN (Max / Mean / Self-attention)	Transformer (Max / Mean / Self-attention)	RNN (Max / Mean / Self-attention)	Bi-RNN (Max / Mean / Self-attention)	Transformer (Max / Mean / Self-attention)
FTP-Patator (1,966)	0.968 / 0.964 / **0.988**	**0.986** / 0.978 / 0.982	**0.988** / 0.986 / **0.988**	0.991 / **0.996** / **0.996**	0.996 / 0.992 / **1.0**	0.998 / **1.0** / 0.998
SSH-Patator (1,451)	0.954 / 0.958 / **0.960**	0.960 / 0.962 / **0.964**	**0.974** / 0.970 / 0.972	0.982 / 0.984 / **0.992**	0.986 / **0.992** / 0.990	0.992 / **0.996** / **0.996**
DoS GoldenEye (2,569)	0.982 / **0.990** / 0.985	**0.990** / 0.988 / **0.990**	0.990 / 0.988 / **0.991**	0.984 / **0.986** / 0.984	0.988 / 0.988 / **0.990**	0.992 / **0.996** / 0.988
DoS Hulk (57,727)	**0.998** / **0.998** / 0.996	**1.0** / **1.0** / 0.998	**1.0** / 0.998 / **1.0**	**1.0** / **1.0** / **1.0**	**1.0** / **1.0** / **1.0**	**1.0** / **1.0** / **1.0**
DoS Slowhttptest (1,347)	0.976 / **0.978** / 0.976	0.982 / 0.976 / **0.986**	0.986 / 0.978 / **0.987**	0.940 / 0.966 / **0.977**	0.934 / 0.972 / **0.982**	**0.980** / 0.972 / 0.962
DoS Slowloris (1,403)	0.948 / 0.951 / **0.954**	0.960 / **0.966** / 0.964	**0.976** / 0.974 / 0.974	0.938 / 0.958 / **0.964**	0.915 / 0.966 / **0.978**	**0.976** / 0.972 / 0.963
XSS (167)	**0.158** / 0.082 / 0.102	**0.118** / 0.048 / 0.002	0 / **0.096** / 0.038	0.200 / 0.236 / **0.272**	0.318 / 0.200 / 0.196	0.69 / 0.384 / **0.385**
Brute force (349)	0.458 / **0.518** / 0.480	0.568 / 0.570 / **0.604**	0.69 / 0.676 / **0.703**	0.740 / **0.758** / 0.718	0.742 / **0.840** / 0.805	**0.896** / 0.854 / 0.879
DDoS (32,008)	0.998 / **1.0** / 0.998	**1.0** / **1.0** / **1.0**	**1.0** / **1.0** / **1.0**	**1.0** / **1.0** / **1.0**	**1.0** / **1.0** / **1.0**	**1.0** / **1.0** / 0.996
PortScan (39,918)	0.972 / 0.966 / **0.978**	0.976 / 0.972 / **0.979**	**0.984** / 0.978 / 0.983	**1.0** / **1.0** / **1.0**	**1.0** / **1.0** / **1.0**	**1.0** / **1.0** / 0.998
Botnet (480)	**0.636** / 0.602 / 0.578	0.624 / 0.586 / **0.648**	0.730 / 0.720 / **0.738**	0.908 / **0.926** / 0.918	**0.948** / 0.922 / **0.948**	0.928 / **0.994** / 0.935
Normal (248,816)	0.990 / 0.990 / **0.991**	0.990 / **0.992** / **0.992**	**1.0** / 0.992 / 0.994	-	-	-

(그림 11) 다중 클래스 분류 성능 비교 결과 (CICIDS 2017)

마지막으로, CICIDS 2017 데이터셋의 다중 클래스 분류 결과는 그림 7과 같다. CICIDS 2017의 경우 이진 분류에서 0.99의 성능을 보였기 때문에, 여러 클래스 (FTP-Patator, SSH-Patator, DoS 관련 공격 클래스)에서는 탐지 정확도가 F1 Score 기준 1에 가까운 결과를 보였다. XSS (Cross-Site Scripting) 공격은 약 0.1의 낮은 성능을 보였지만 정상 트래픽 데이터를 포함하지 않았을 때에는 그 성능이 약 0.38까지 향상되었다. 그리고 brute force 공격과 botnet 공격은 평균 약 0.65 정도의 분류 정확도를 보였으나, 해당 클래스들의 분류 성능도 정상 데이터를 제외했을 때 약 0.2 이상 개선되었다 (Brute force: 0.896, Botnet: 0.994).

4 결론

네트워크 및 서비스가 발전하면서 다양한 공격 및 침입에 대한 위협이 증가하였다. 본 논문에서는 네트워크 공격 및 침입과 관련된 비정상 트래픽을 탐지하기 위해 딥러닝을 기반으로 하는 탐지 모델을 제안한다. 제안하는 모델은 3가지 서로 다른 공개 데이터셋에서 정상 데이터와 비정상 데이터의 이진 분류 및 Root-Cause Analysis를 위한 다중 클래스 분류를 수행한다. 실험 결과는 이진 분류에서의 F1-score를 기준으로 볼 때, NSL-KDD에서는 0.956, UNSW-NB15에서는 0.938, 그리고 CICIDS 2017에서는 0.997의 결과값을 가지며, 이는 baseline 모델 및 관련 연구와 비교했을 때 보다 향상되고 균일한 성능을 보인다.

향후 연구에서는 제안하는 모델 [10]의 성능을 향상시키고, 네트워크 공격 및 침입과 관련된 보다 다양한 데이터셋을 이용하여 성능을 검증할 예정이다. 또한, 본 과제의 실제 테스트베드 환경 [11]으로 적용을 통해 실제 환경에서 생성된 정상 및 비정상 트래픽 데이터를 사용하여 모델의 성능을 검증할 예정이다.

5 참고문헌

[1] O. Sefraoui, M. Aissaoui, and M. Eleuldj, “Open-stack: toward an open-source solution for cloud computing,“ in International Journal of Computer Applications, vol. 55, no. 3, pp. 38-42, Oct. 2012.

[2] IRTF Network Function Virtualization Research Group, “Network Functions Virtualisation - Update White Paper,“ [Online]. Available at https://portal.etsi.org/NFV/NFV_White_Paper2.pdf.

[3] J. Hong, S. Park, J. -H. Yoo and J. W. -K. Hong, “Machine Learning based SLA-Aware VNF Anomaly Detection for Virtual Network Management,“ 2020 16th International Conference on Network and Service Management (CNSM), pp. 1-7, 2020.

[4] ETSI, “zero-touch network & service management (ZSM)”, [Online], Available at: https://www.etsi.org/technologies/zero-touch-network-service-management

[5] TM Forum, “Autonomous Networks Project”, [Online], Available at: https://www.tmforum.org/collaboration/autonomous-networks-project/

[6] Kwon, D. et al., “A survey of deep learning-based network anomaly detection“, Cluster Computing vol.22, pp. 949-961, Sep. 2019.

[7] NSL-KDD Dataset, 2009. Available at: https://www.unb.ca/cic/datasets/nsl.html

[8] UNSW-NB15 Dataset, 2015. Available at: https://research.unsw.edu.au/projects/unsw-nb15-dataset

[9] Intrusion Detection Evaluation Dataset (CIC-IDS 2017), 2017. Available at: https://www.unb.ca/cic/datasets/ids-2017.html

[10] Attack & Intrusion Detection Model, DPNM NI Project Github, [Online], Available at: https://github.com/dpnm-ni/AttackandIntrusion-Detection

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