논문 21-46-12-19 The Journal of Korean Institute of Communications and Information Sciences '21-12 Vol.46 No.12
https://doi.org/10.7840/kics.2021.46.12.2251

SDN에서의 보안 시스템 부하 감소 기법

이 예 원◆, 이 채 우°

Security System Load Reduction in SDN

Ye-won Lee◆, Chae-woo Lee°

요 약

자신의 환경에 적합한 네트워크를 소프트웨어적으로 구현할 수 있는 SDN(Software Defined Networking)이 각광받고 있다. SDN은 중앙 집중형 구조를 가지므로 네트워크 관리가 용이하며 기존 장비에 통합되어 있던 기능들을 분리하여 소프트웨어적으로 스위치를 제어할 수 있는 장점이 있다. 또한 네트워크 보안성 증대를 위해 SDN 환경 내에 IDS(Intrusion Detection System), IPS(Intrusion Prevention System) 등의 다양한 보안 시스템이 도입되었다. 하지만 모든 패킷을 검사하여 보안 시스템의 부하가 높아졌고 이를 해결하고자 다양한 패킷 샘플링 기반 기법들이 제시되었지만 여전히 보안 시스템의 부하가 높거나 샘플링 기법들로 인해 네트워크의 보안성이 떨어지는 문제가 있었다. 따라서 본 논문에서는 관리테이블 기반의 패킷 관리 기법을 통해 Flooding 공격 발생 시 보안 시스템의 과부하와 보안성을 개선한 시스템을 제안한다. Backlog Queue 값을 기준으로 패킷 검사를 실행하는 기존 시스템과 달리 패킷 간의 시간차와 패킷 발생 빈도를 기반으로 관리 테이블을 생성하므로 보안 시스템을 각 패킷마다 유동적으로 실행하여 네트워크의 보안성을 높일 수 있다. 또한 이상 패킷인 경우에는 일정 시간동안 해당 패킷을 컨트롤러에서 차단하여 보안 시스템의 부하를 줄일 수 있으며 제안하는 기법의 성능 평가 결과 기존 시스템과 비교하여 보안 시스템의 부하가 약 25% 개선됨을 확인하였다.

키워드 : 소프트웨어 정의 네트워크, 침입탐지시스템, 플러딩
Key Words : SDN, IDS, Flooding

ABSTRACT

The recent trend of SDN has been demonstrated for security systems such as Intrusion Detection System and Intrusion Prevention System have been introduced in the SDN environment to increase network security. However, the load on the security system increased by examining all packets, and various packet sampling-based techniques were proposed to solve this problem, but there were still problems with the security system being high or the network security was poor due to the sampling techniques. In this paper, we propose a system improving security and reducing system loads through management table-based packet management in case of flooding attack. The existing system is inspected by backlog queue value but the proposed security system can be flexibly executed for each packet creating a management table. This table is based on the time difference between packets and the frequency of occurrence of packets. The load on the security system can be reduced by blocking the corresponding packet by the controller for a certain period of time when abnormal packet comes. As a result of the performance evaluation of the proposed method, it is confirmed that the load of the security system is improved by about 25% compared to the existing system.

◆ First Author : Ajou University Department of IT Convergence, dldpdnjs1566@naver.com, 학생회원
° Corresponding Author : Ajou University Department of IT Convergence, cwlee@ajou.ac.kr, 정회원
논문번호 : 202106-131-B-RN, Received June 10, 2021; Revised August 30, 2021; Accepted September 24, 2021

Ⅰ. 서 론

기존에는 음성, 방송, 데이터, 차량 등에서 필요한 네트워크 서비스의 성능 향상을 위해서 하드웨어 기술 위주로 발전해왔다. 하지만 최근 사물인터넷, 클라우드, 빅데이터 등의 기술들로 인한 급격한 트래픽 상승을 정적인 구조의 기존 네트워크로는 쉽게 따라갈 수 없다. 이를 위해서 네트워크를 유연하게 유지보수할 수 있는 소프트웨어 정의 네트워크인 SDN(Software Defined Networking)이 개발되었다. SDN은 사용자가 원하는 대로 자신의 환경에 적합한 네트워크를 구축하여 최적화된 환경을 만들 수 있으며 네트워크 전체가 아닌 필요한 기능만 설계할 수 있다. 또한 기존 네트워크에서는 각 스위치를 개별적으로 설정해야 하지만 SDN은 중앙 집중형 구조를 가지므로 컨트롤러의 알고리즘이나 설정 변경 시 전체의 스위치에 한 번에 반영되어 네트워크 관리가 용이하다[1,2].

SDN은 Infrastructure Layer, Control Layer, Application Layer로 분리되어 데이터 전송, 운영체제, 네트워크 지능화 기능을 각각 담당하며 컨트롤러에서 이를 관제하는 구조를 가진다. 이때 각 Layer 간에는 API를 이용하여 통신을 하게 되며 가장 대표적인 프로토콜은 OpenFlow가 있다. 이를 통해 네트워크 트래픽 전달 동작을 컨트롤러에서 제어하여 사용자의 요구 사항에 따라 여러 개의 네트워크를 관리할 수 있는 장점이 있다. 하지만 SDN은 네트워크의 구조가 바뀌면서 효율성이 높아진 것이지 기존에 존재하는 네트워크 공격들에 대한 방어책이 되지 못한다. 또한 기존의 네트워크와 달리 중앙 집중형 구조를 가지므로 보안상 약점이 늘어날 수도 있다. 이를 해결하기 위해 SDN 내에 IDS(Intrusion Detection System), IPS(Intrusion Prevention System) 등의 다양한 보안 시스템을 적용하는 연구가 진행되었다.

초기에는 SDN 컨트롤러로 들어오는 모든 패킷을 검사하여 보안성을 높이는 데 중점을 두었다. 하지만 보안 시스템에 부하가 높아지게 되었고 이를 해결하기 위해 공격이 지속되는 경우에는 일정 시간동안 모든 패킷을 차단하거나 Backlog Queue가 임계치에 도달하기 전까지 패킷을 검사하지 않는 등의 패킷 샘플링 기반 기법들이 제안되었다[3-6]. 이를 통해 보안 시스템의 부하는 줄일 수 있었지만 Backlog Queue가 일정 수준이 되지 않으면 컨트롤러로 들어오는 모든 패킷을 검사하지 않는 문제가 있었다. 또한 Backlog Queue가 일정 수준이 되지 않는 상태에서 이상 패킷이 컨트롤러로 들어오면 목적지로 패킷을 전송하여 네트워크에 문제가 발생할 수 있다. 이를 해결하기 위해 본 논문에서는 Flooding 공격 발생 시 보안 시스템의 부하를 줄이면서 보안성을 높이기 위한 관리 테이블 알고리즘을 제안한다. 패킷의 메타데이터를 확인하여 각 패킷 간의 시간차와 횟수 기반의 관리 테이블을 생성하며 이를 기준으로 보안 시스템을 실행하여 유동적으로 각 패킷을 검사할 수 있으므로 네트워크의 보안성을 개선할 수 있다. 또한 이상 패킷으로 판별된 경우에는 일정 시간 해당 패킷을 컨트롤러에서 차단하므로 보안 시스템의 부하율도 줄일 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 SDN과 IDS의 이론적 배경을 설명한다. 3장에서는 시스템 모델을 구성하고 이상 패킷 감지 알고리즘과 패킷 카운팅 알고리즘을 제안한다. 4장에서는 기존 IDS의 패킷 처리율, 이상 패킷 관리 테이블을 적용했을 때 IDS의 패킷 처리율, 이상 패킷 관리 테이블과 카운팅 테이블을 적용했을 때 IDS의 패킷 처리율을 비교하여 시스템 성능을 확인한다. 마지막으로 5장에서는 결론으로 끝을 맺는다.

Ⅱ. 이론적 배경

2.1 관련 연구

[3]에서는 목적지 주소별 패킷 샘플링을 이용한 SYN Flooding 공격 방어기법을 제시하였다. SYN Flooding 패킷이 Backlog Queue의 70% 이상을 점유하면 패킷 모니터링을 시작하며 이 패킷이 이상 패킷으로 판별되면 일정 시간동안 해당 패킷을 차단하였다. 하지만 Backlog Queue가 임계치에 도달하기 전에 이상 패킷이 컨트롤러로 들어온다면 목적지로 패킷이 전달될 가능성이 크므로 네트워크의 보안성이 떨어지는 문제가 있다.

[4]에서는 SDN에서 Flow를 기반으로 침입을 탐지한 후 IDS로 전송하는 기법을 제시하였으며 Flow를 기반으로 공격을 탐지하므로 보안 시스템의 오버헤드를 낮출 수 있었다. 해당 논문에서는 대략적인 시스템은 제안되어 있지만 IDS에서 패킷을 검사한 후 어떻게 이 데이터를 처리하고 정상 패킷 감지 절차와 비정상 패킷 감지 절차 등이 기술되어 있지 않다. 또한 시스템의 정량적인 평가도 포함되어 있지 않아서 시스템 성능을 파악할 수 없다.

[5]에서는 OpenFlow와 sFlow를 이용한 공격 방어기법을 제시하였으며 공격 패킷이 임의로 정한 임계치를 초과하면 네트워크 전체의 패킷을 일정 시간동

안 차단한다. 하지만 패킷 차단 시간 동안 정상 패킷도 목적지로 전달되지 못하고 공격 패킷과 정상 패킷을 판별하는 기준과 임계값이 명확하지 않다.

[6]에서는 L7 스위치 기반 DDoS 감지 기법을 제안하였다. TCP Connection, 세션 테이블 관리 등의 기능을 수행하는 L7 스위치를 통해 패킷을 검사하여 이상 패킷을 차단하였지만 스위치의 성능이 높아진 만큼 하드웨어 비용이 높은 문제가 있다.

위와 같이 SDN 환경에서 보안 시스템의 부하를 줄이기 위한 기법들이 제시되었지만 보안성이 떨어지거나 정상 패킷의 흐름에도 영향을 끼쳤으며 하드웨어 기반의 보안 시스템의 경우에는 비용이 높은 문제가 있다.

2.2 SDN

2.2.1 정의

기존 네트워크 장비는 그림 1과 같이 데이터를 전송하는 Data Plane 영역, Operating System 기능을 하는 Control Plane 영역, 네트워크 지능화 기능을 담당하는 Application Plane 영역이 하나로 되어있다. 이로 인해 필요 이상의 기능들이 강제로 포함되어 다른 장비들과의 호환성이 맞지 않을 수 있으며 스위치 설정 변경 시 각 스위치마다 별도로 설정을 해야 하는 문제가 있다. 따라서 그림 2와 같이 Data Plane 영역, Control Plane 영역, Application Plane 영역을 분리한 뒤 네트워크를 중앙 집중형으로 구성하여 빠르고 안전한 네트워크를 운용할 수 있는 소프트웨어 정의 네트워크인 SDN(Software Defined Networking)이 개발되었다[7-9]. SDN은 Openflow 프로토콜을 통하여 해당 프로토콜을 지원하는 스위치들을 일괄적으로 제어할 수 있다. 또한 사용자의 필요에 따라 자신의

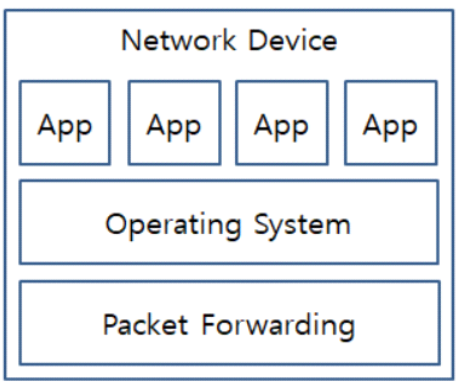

그림 1. 기존 네트워크 장비 구조
Fig. 1. Existing Network Equipment Structure

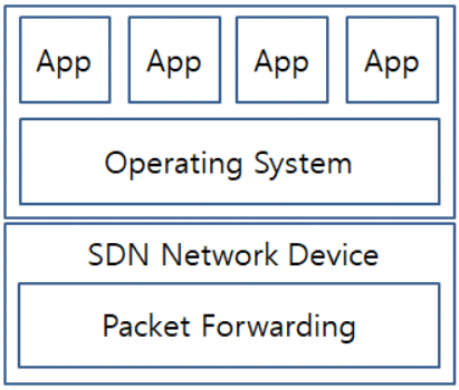

그림 2. SDN 구조
Fig. 2. SDN Structure

환경에 맞게 네트워크를 구축할 수 있는 개방형 구조를 가지므로 필요한 기능만을 이용하여 최적화된 네트워크를 구축할 수 있다.

2.2.2 Hierarchy

SDN은 그림 3과 같이 Infrastructure Layer, Control Layer, Application Layer로 분리되어 있다. Infrastructure Layer와 Control Layer 간의 통신을 위해서 Southbound API를 이용하며 주로 대표적인 프로토콜인 OpenFlow를 이용한다. 또한 Control Layer와 Application Layer 간의 통신을 위해서는 Northbound API를 이용하며 표준 프로토콜이 없으므로 각 컨트롤러마다 자체 구현하는 것이 일반적이다[10,11].

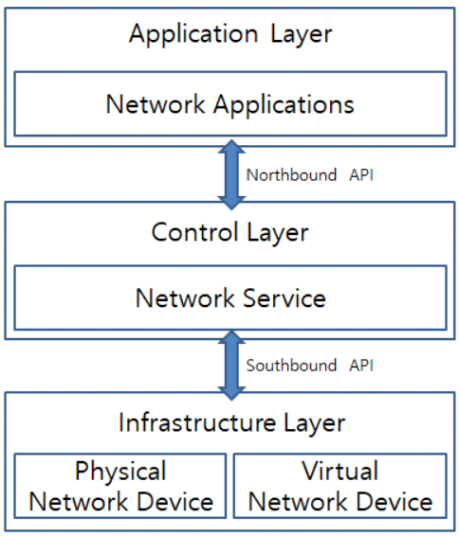

그림 3. SDN 계층
Fig. 3. SDN Hierarchy

SDN의 Infrastructure Layer는 트래픽을 전송하는 영역으로 Flow Table에 따라 동작한다. Flow Table은 어떤 패킷을 처리할지 정의하는 Rule, 어떻게 패킷을 처리할지 정의하는 Action, 패킷 전송 및 매칭 상태를 알려주는 Stats로 구성된다. Control Layer는 데이터 영역으로 무슨 패킷이 어떤 Flow로 흐르는지 제어하는 영역이며 패킷을 포워딩한다. 그리고 Application Layer는 SDN 운영을 위한 Tool, 모니터링 등의 기능을 수행한다.

2.2.3 컨트롤러

SDN 컨트롤러는 NOX, POX, ONOS, OpenMul 등 다양한 Open Source가 있으며 각각의 특징은 표 1과 같다[12-14]. NOX는 C++을 기반으로 한 최초의 SDN 컨트롤러이며 학교나 연구 기관을 대상으로 개발되어 조작은 간단하지만 기능이 제한적이다. POX는 NOX와 같은 연구기관에서 개발되었으며 NOX와 동일한 연구 목적의 Python 기반의 제어기다. ONOS는 Java 기반의 컨트롤러로 확장성이 높고 서비스 제공업체들이 많이 사용하여 자료가 다양하지만 각 컨트롤러에 배치하는 스위치의 트래픽 량을 조절하기 어려운 문제가 있다. OpenMul은 국내 최초의 SDN 컨트롤러로 다른 컨트롤러에 비해 자료가 적은 단점이 있지만 C언어를 기반으로 하여 동일한 하드웨어에서 최상의 처리량과 이식성을 가진다. 또한 GUI와 CLI를 통해 네트워크를 구성할 수 있어서 아키텍처 가시성이 좋으며 FirmFlow와 FlexPlug 기능이 지원되어 다른 어플리케이션이나 서비스에 의해 네트워크가 영향을 받지 않는다. 따라서 본 시스템에서는 동일한 환경에서 최상의 성능을 가지는 OpenMul 컨트롤러를 이용하여 SDN 환경을 구성한다.

2.2.4 구조

표 1. SDN 컨트롤러 비교
Table 1. SDN Controller Comparison

	Language	Advantages	Weakness
NOX	C++	Easy to Operate	Limited Function
POX	Python	Easy to Operate	Limited Function
ONOS	Java	Lots of Data	Hard to Control Traffic
OpenMul	C	High Throughput/ Portability	Not Much Data

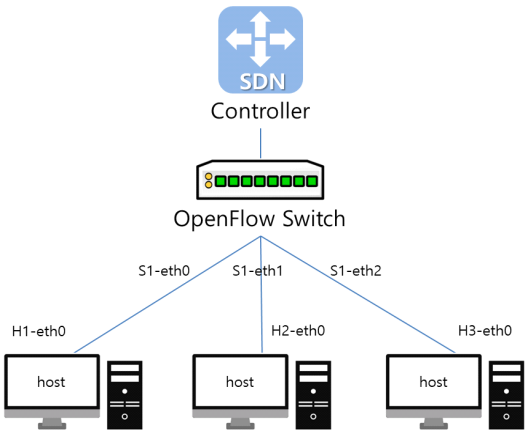

그림 4. SDN 아키텍처
Fig. 4. SDN Architecture

SDN의 기본 구조는 그림 4와 같이 구성되며 컨트롤러와 스위치가 연결되고 스위치는 각 호스트와 연결된다. 하나의 컨트롤러에는 여러 개의 스위치를, 각 스위치에서는 여러 개의 호스트를 연결할 수 있으며 컨트롤러 배치 구조는 Centralized, Distributed, Multilayer가 있다. 본 시스템에서는 현재 가장 많이 사용되는 방식으로 장애가 발생하면 Standby 컨트롤러에서 이전 컨트롤러에서 하던 일을 넘겨받아서 처리하는 Centralized 구조를 이용한다[15].

2.3 IDS

2.3.1 정의

네트워크 환경에서 권한이 없는 사용자가 접근하거나 공격 패킷이 오는 경우에 이를 감지할 수 있는 기능이 필요하다. 따라서 네트워크로 들어오는 패킷을 검사하여 감시 카메라의 역할을 수행하는 침입 탐지 시스템인 IDS(Intrusion Detection System)가 개발되었다[16-18]. 이때 패킷 데이터를 분석하여 출발지 IP, 출발지 Port, 도착지 IP, 도착지 Port, 공격 종류 등의 정보를 통해 공격 패킷인지 아닌지 판별한다. IDS의 주요 기능은 네트워크의 실시간 감시, 공격 감지 및 로그 생성, 침입 분석 등이 있다.

2.3.2 구조

IDS는 침입자나 다른 곳에서 공격 패킷을 보내는 경우에 시스템 자원에 영향을 끼치기 전에 이를 발견하는 것이 목적이며 IDS는 공격 감지만 가능하기 때문에 트래픽을 차단하거나 별도의 알림을 주는 기능이 없다. 따라서 일반적으로는 그림 5와 같이 라우터 다음 단에 IDS나 Firewall 등을 배치하여 공격을 감

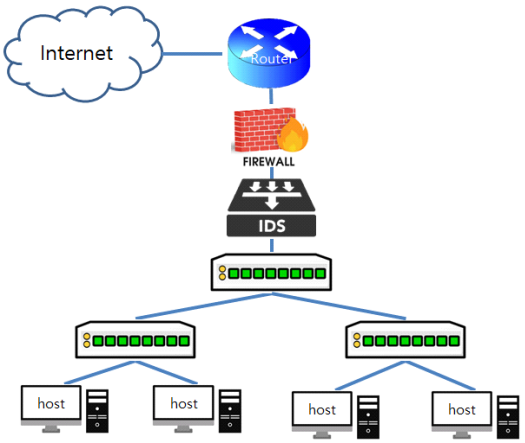

그림 5. 일반적인 IDS 위치
Fig. 5. Typical IDS Location

지하고 이상 패킷인 경우 트래픽을 차단하는 등 설정된 루틴에 따라 처리한다.

SDN에서는 IDS 환경을 구성하기 위해 패킷 검사를 수행하는 Host를 생성하여 Flow를 기반으로 검사를 실행하거나 Snort 등의 Open Source 기반의 IDS 시스템을 적용할 수 있다. Flow 기반의 검사는 패킷 검사를 위한 시스템 구축이 어렵지만 패킷 처리 속도가 빠르다. Open Source 기반의 IDS는 사용은 간단하지만 Flow 기반의 검사보다 패킷 처리 속도가 느리므로 본 시스템에서는 Flow 기반의 검사를 이용한다. 이때 SDN의 구조상 모든 패킷은 컨트롤러와 컨트롤러 아래에 위치한 스위치를 거치므로 패킷 검사를 위한 Host는 스위치에 연결되어 있다. 이를 통해 SDN 내의 모든 패킷을 검사할 수 있으며 검사 결과를 컨트롤러에서 확인하여 설정한 루틴에 따라 이상 패킷과 정상 패킷을 처리할 수 있다.

Ⅲ. 시스템 모델

3.1 시스템 구성

본 논문에서는 SDN에서의 효율적인 이상 패킷 감지 기법의 구성을 그림 6과 같이 제안한다. 이때 리눅스 환경에서 가상 호스트, 가상 스위치, 가상 라우터 등을 생성하고 네트워크 토폴로지 구성이 가능한 네트워크 Emulator인 Mininet을 이용한다[19-21].

제안하는 시스템의 네트워크 토폴로지는 컨트롤러 1개, 스위치 1개, Attacker Host 2개, Server Host 1개, IDS Host 1개로 하며 각 Host는 스위치에 연결된다. 이때 Attacker1 Host는 10.0.0.1, Attacker2 Host는

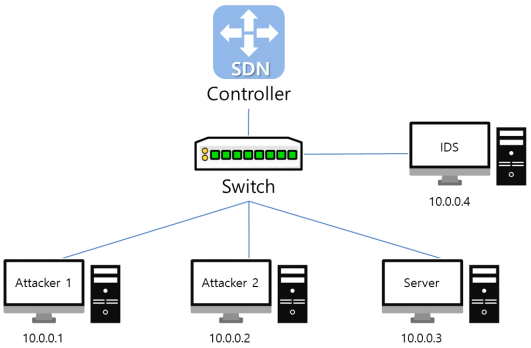

그림 6. 시스템 구성도
Fig. 6. System Architecture

10.0.0.2, Server Host는 10.0.0.3, IDS Host는 10.0.0.4 IP를 가진다. 또한 컨트롤러에는 IDS Host에서 얻은 검사 결과를 기반으로 이상 패킷 관리 테이블과 패킷 카운팅 테이블을 구성한다. 이상 패킷 관리 테이블은 패킷을 일정시간 차단하기 위해 관리하며 패킷 카운팅 테이블은 이상 패킷 관리 테이블에 등록하여 패킷을 차단하기 전에 일정 횟수 이상 공격이 반복 되는지 확인한다.

3.2 시스템 흐름도

SDN에서의 효율적인 이상 패킷 감지 기법의 Flow Chart는 그림 7과 같다. Packet In 이벤트를 통해 컨트롤러에 패킷이 들어오며 만약 패킷 데이터가 관리 테이블에 있다면 패킷을 Drop 하고 관리 테이블에 없다면 스위치의 출력 포트를 조작하여 IDS Host로 패킷 데이터를 전송하여 정상 패킷인지, 비정상 패킷인지 검사한다. 패킷 검사 결과가 비정상이라면 관리 테이블에 추가한 후 패킷을 Drop 하며 패킷 검사 결과가 정상이라면 목적지 Host로 전달한다.

3.3 이상 패킷 테이블 관리 알고리즘

3.3.1 이상 패킷 관리 테이블

이상 패킷 관리 테이블을 구성하기 위한 의사 코드는 표 2와 같다. Packet In 이벤트가 발생하면 우선 관리 테이블에 있는지 확인하고 이상 패킷 관리 테이블에 있다면 그림 8과 같이 패킷을 Drop 한 후 이벤트를 종료한다. 만약 관리 테이블에 해당 패킷 정보가 없다면 in_port가 IDS_PORT인지 확인한다. 이때 패킷 전달 절차는 그림 9와 같다.

IDS에서는 검사한 패킷이 비정상 패킷일 경우에는 공격 종류를 판별하여 우선순위에 따라 패킷 차단 시간을 설정하며 검사 결과를 컨트롤러로 전송한다. 이

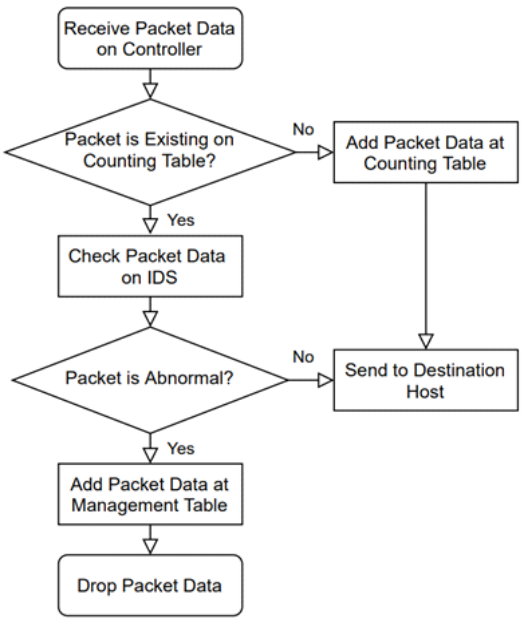

그림 7. 시스템 흐름도
Fig. 7. System Flow Chart

표 2. 이상 패킷 관리 테이블 Pseudo-Code
Table 2. Abnormal Packet Management Table Pseudo-Cod

```
1:  func packet_in_event
2:      if pkt_in is existing
                on abnormal_packet_table
3:          drop packet
4:      end if
5:
6:      lrn_port = forwarding_table[dst_host]
7:      if in_port is IDS_PORT
8:          if packet is abnormal
9:              drop packet
10:             add packet in
                    abnormal_packet_table
11:         else
12:             send packet out to lrn_port
13:         end if
14:     end if
15: end func
16:
17: func timer
18:     if abnomal_packet_table is not null
19:         if time_data is 0
20:             remove packet in
                    abnormal_packet_table
21:         else
22:             decrease time_data
23:         end if
24:     end if
25: end func
```

때 컨트롤러에서는 IDS의 검사 결과를 바탕으로 패킷이 정상이면 그림 10과 같이 패킷을 목적지로 전달하고 정상이 아닌 경우에는 패킷을 Drop한 뒤 이상 패킷 관리 테이블에 추가한다.

in_port가 IDS_PORT인 경우에는 컨트롤러에서는 IDS 검사 결과를 받은 것으로 판단하며 IDS에게 전

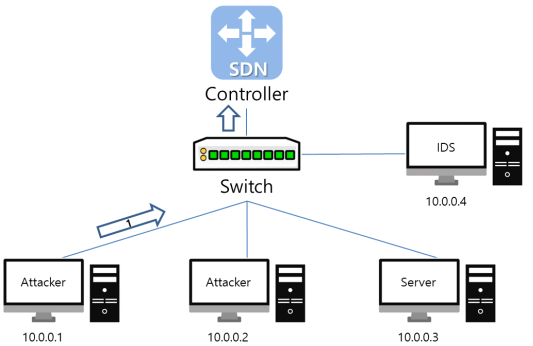

그림 8. 이상 패킷 관리 테이블에 있을 때의 비정상 패킷 전달 절차
Fig. 8. The Forwarding Process When Abnormal Packet In Management Table

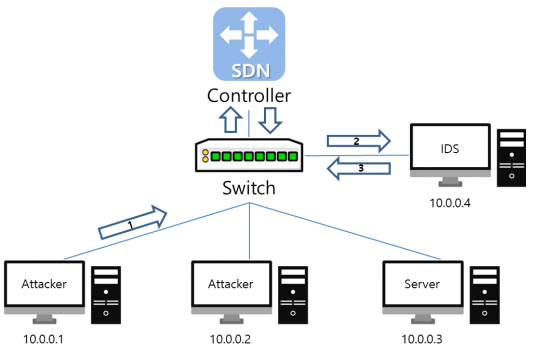

그림 9. 이상 패킷 관리 테이블에 없을 때의 비정상 패킷 전달 절차
Fig. 9. The Forwarding Process When Abnormal Packet Not In Management Table

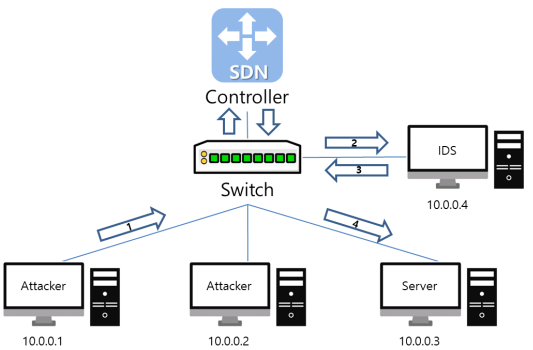

그림 10. 정상 패킷 전달 절차
Fig. 10. The Forwarding Process When Normal Packet

달받은 결과가 비정상 패킷이면 패킷을 Drop한 후 이상 패킷 관리 테이블에 추가한다. 이때 이상 패킷 관리 테이블은 데이터 탐색이 효율적인 Hash Table을 이용하여 구성하며 Table의 Key는 패킷 정보, Data는 패킷 차단 시간으로 설정한다. 또한 Hash Table의 패킷 차단 시간은 1초마다 줄어들며 차단 시간이 0일 경우에는 테이블에서 삭제한다. 그리고 IDS에게 전달받은 결과가 정상 패킷인 경우에는 목적지 주소로 패킷을 전달한다.

기존 IDS의 처리 패킷 합은 수식 (1)과 같으며 S1으로, 이상 패킷 관리 테이블을 적용한 시스템은 수식 (2)와 같으며 S2로 표현한다. 이때 N은 전송 패킷 수, $P^{i}_{A\rightarrow B}$는 i번째 A에서 B로 가는 패킷 수이다. 또한 (3)번과 같이 함수를 구성할 수 있다.

$$S_1 = \sum_{i=1}^{N} P^{i}_{h1\rightarrow h2} + \sum_{i=1}^{N} P^{i}_{c_o\rightarrow IDS} \quad (1)$$

$$S_2 = \sum_{i=1}^{N} P^{i}_{h1\rightarrow h2} + \sum_{i=1}^{N} Alg_2(P^{i}_{c_o\rightarrow IDS}) \quad (2)$$

$$Alg_2(P) = \begin{cases} P\ (P\ does\ not\ exist\ in\ IDS\ table) \\ 0\ (P\ exists\ in\ IDS\ table) \end{cases} \quad (3)$$

3.3.2 패킷 카운팅 테이블

패킷 카운팅 테이블을 구성하기 위한 의사 코드는 표 3과 같다. Packet In 이벤트가 발생한 뒤 in_port가 IDS_PORT라면 이상 패킷 관리 테이블 알고리즘을 실행하며 IDS_PORT가 아니고 패킷 카운팅 테이블에 해당 패킷 데이터가 없는 경우에는 그림 11과 같이 패킷카운팅 테이블에 해당 패킷 정보를 저장한다. 이때 테이블을 작성하기 위해 컨트롤러로 들어온 패킷의 메타데이터를 이용하며 패킷의 출발지 IP/Port, 목적지 IP/Port와 타이머 정보를 기록한 후 목적지 Host로 패킷을 전송한다. 이후 동일한 Host에서 일정 시간 내에 패킷이 다시 전송된다면 스위치의 출력 포트를 조작하여 목적지 Host가 아닌 IDS Host로 보내어 패킷을 검사한다. 그리고 동일한 Host에서 일정 시간 이후에 패킷이 다시 컨트롤러로 들어왔다면 패킷 카운팅 테이블에서 해당 패킷 정보를 삭제하고 IDS Host가 아닌 목적지 Host로 패킷을 전송한다. 따라서 컨트롤러로 들어오는 모든 패킷을 IDS로 보내지 않으므로 IDS의 부하를 줄일 수 있다. 본 논문에서는 SYN Flooding 발생 시 공격 패킷이 연달아 전송

표 3. 패킷 카운팅 테이블 Pseudo-Code
Table 3. Packet Counting Table Pseudo-Code

```
1:  func packet_in_event
2:      lrn_port = forwarding_table[dst_host]
3:      if in_port is not IDS_PORT
4:          if count_table[inport] is 0
5:              count_table[inport] = current_time
6:              send packet out to lrn_port
7:          else
8:              if timer interval
                    is under 3 second
9:                  send packet out
                        to IDS_PORT
10:             else
11:                 send packet out to lrn_port
12:             end if
13:         end if
14:     else if in_port is IDS_PORT
15:         process Alg2
16:     end if
17: end func
```

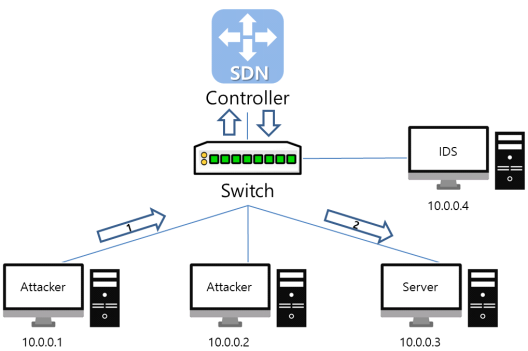

그림 11. 카운팅 테이블에 없을 경우 패킷 전달 절차
Fig. 11. The Forwarding Process When Not In Counting Table

되는 특성을 고려하여 패킷 카운팅 테이블의 시간을 3초로 설정하였다.

$$S_3 = \sum_{i=1}^{N} P^{i}_{h1\rightarrow h2} + \sum_{i=1}^{N} Alg_3(P^{i}_{c_o\rightarrow IDS}) \quad (4)$$

$$Alg_3(P) = \begin{cases} P\ (P\ does\ not\ exist\ in\ counting\ table) \\ 0\ (P\ exists\ in\ count\ table) \end{cases} \quad (5)$$

이상 패킷 관리 테이블과 카운팅 테이블을 적용한 시스템은 수식 (4)와 같으며 S3으로 나타낼 수 있다. 이때 N은 전송 패킷 수, $P^{i}_{A\rightarrow B}$는 i번째 A에서 B로 가는 패킷 수이다. 또한 (5)번과 같이 함수를 구성할 수 있다.

Ⅳ. 성능평가

4.1 실험 환경

본 연구 결과의 성능을 평가하기 위해 그림 12와 같이 Mininet에서 OpenMUL SDN 컨트롤러 기반 환경을 구성하여 두 가지 항목을 확인하였다. 첫 번째는 동일한 패킷을 임의로 생성하여 이상 패킷 관리 테이블과 카운팅 테이블이 제안하는 알고리즘에 따라 동작하는지 분석하였다. 두 번째는 Backlog Queue 기반의 시스템과 제안하는 알고리즘의 성능을 비교하였다. 이때 성능을 평가하기 위한 실험 환경은 표 4와 같다.

실험 환경은 VMWare에서 그림 13과 같이 Mininet을 실행하고 sudo mn 명령어를 통해 네트워크 토폴로지를 구성할 수 있으며 스위치 타입은 'ovsk', Host는 한 개의 스위치에 4개가 연결되도록 'single,4' 옵션을 이용한다. 이때 Remote 컨트롤러

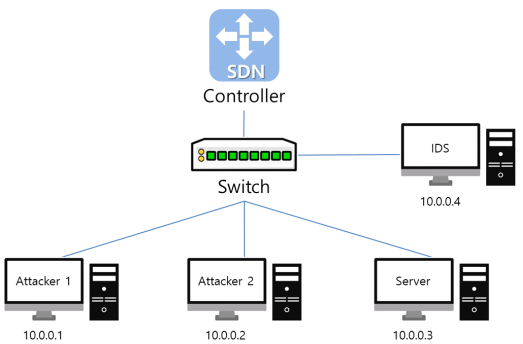

그림 12. 실험 환경
Fig. 12. Experimental Environment

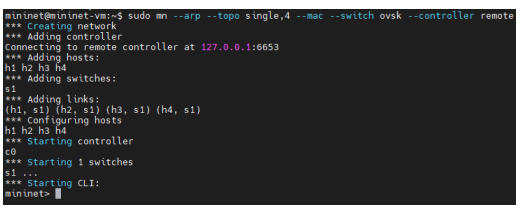

그림 13. Mininet에서의 SDN Network 생성
Fig. 13. Creating An SDN Network In Mininet

표 4. 성능 평가 실험 환경
Table 4. Performance Evaluation Experiment Environment

Name	Version
Operating System	Ubuntu 14.04.4
Emulator	Mininet 2.2.2
Controller	OpenMUL
Switch	OpenSwitch

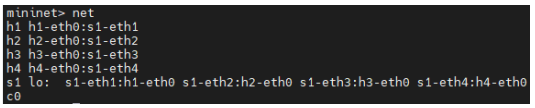

그림 14. Mininet 연결 구성 목록
Fig. 14. Mininet Connection List

는 127.0.0.1 주소에 6653 포트로 설정된다.

Mininet을 이용하지 않을 경우에는 컨트롤러, 스위치, 각 Host 별로 가상 머신을 생성해야 하지만 Mininet을 이용하면 하나의 가상 머신에서 테스트 환경 구축이 가능하다. Mininet을 실행한 뒤 연결 목록을 확인하면 그림 14와 같다.

4.2 알고리즘 검증

제안하는 알고리즘에 따라 이상 패킷 관리 테이블과 카운팅 테이블이 동작하는지 확인하기 위해 0.1초 간격으로 1000개의 동일한 패킷을 생성하였다. 이때 패킷은 Attacker Host에서 Server Host로 전송하며 IDS에서 판별한 우선순위에 따라 패킷 차단 시간을 설정함을 확인하였다. Priority 1이면 10초, Priority 2라면 20초, Priority 3이면 30초, Priority 4라면 40초간 해당 패킷을 컨트롤러에서 Drop하여 IDS 및 목적지로 전송하지 않는다. 또한 패킷 차단 시간이 종료된 후에 패킷이 다시 컨트롤러로 들어온다면 IDS에서 패킷을 검사한 후 이상 패킷 관리 테이블에 해당 패킷을 재등록하여 일정 시간동안 패킷을 Drop 하는 것을 확인하였다.

표 5. 우선순위에 따른 IDS의 패킷 검사 횟수
Table 5. Number of Packet Checks in IDS according to Priority

Priority	IDS Packet Inspection Count
None	1000
1	10
2	5
3	4
4	3

표 5의 결과와 같이 1000개의 패킷을 전송받은 경우 알고리즘을 적용하지 않으면 IDS에서 1000번의 검사를 실행하며 알고리즘을 적용했을 때는 Priority 1일 경우 10번, Priority 2일 경우 5번, Priority 3일 경우 4번, Priority 4일 경우 3번을 IDS에서 패킷을 검사하는 것을 확인하였다.

4.3 성능 비교

성능 비교를 위한 대조군으로 본 시스템과 동일하게 Flooding 공격 발생 시 SDN의 보안 시스템의 부하를 줄이기 위한 시스템을 선정하였으며 이 시스템은 Backlog Queue를 기반으로 일정 시간동안 패킷을 차단한다. 이를 위해 SYN Flooding 패킷 5000개를 임의로 생성하여 제안하는 시스템과 성능을 비교하였으며 Backlog Queue 기반의 시스템은 SYN 패킷이 16개가 넘으면 패킷 검사를 시작하며 이상 패킷일 경우에는 일정 시간 해당 패킷을 차단한다. 이 시스템을 적용한 상태에서 5000개의 SYN Flooding이 발생할 경우 약 1104개의 패킷이 IDS에서 검출되어 보안 시스템의 부하가 약 78% 개선되었다. 제안하는 기법의 검증도 위와 동일하게 SYN Flooding 패킷 5000개를 임의로 생성하여 확인하였다. 이때 약 827개의 패킷이 IDS에서 검출되었으며 알고리즘을 적용하지 않았을 때와 비교하여 약 83%, Backlog Queue 기반의 시스템과 비교하여 보안시스템의 부하가 약 25% 개선됨을 보았다. 실험 결과는 그림 15와 같다.

V. 결 론

본 논문에서는 SDN에서의 패킷 샘플링 기반 이상 패킷 감지 기법을 제안하였다. 하드웨어 기반의 시스템이 아니므로 시스템 구성비용이 저렴한 장점이 있으며 Flooding 공격 발생 시 컨트롤러에 들어오는 패킷의 메타데이터를 확인하여 각 패킷 간의 시간차와 횟수 기반의 관리 테이블을 생성하였다. 이를 기반으로 각 패킷별로 패킷 검사를 유동적으로 실행할 수 있었으며 이상 패킷인 경우에는 컨트롤러에서 일정 시간동안 패킷을 차단하여 보안 시스템으로 전달하지 않아 보안 시스템의 부하를 줄어드는 것을 보았다. 또한 패킷 데이터별로 테이블을 관리하여 패킷 검사를 실행하므로 네트워크의 보안성도 개선되었다. 성능 평가 결과 모든 패킷을 검사하는 경우 대비 보안 시스템의 부하가 83% 감소하였으며 기존의 Backlog Queue 기반 시스템 대비 25% 줄어드는 것을 확인하였다. 본 시스템을 이용하여 IoT 서비스망이나 WAN, 데이터 센터 등 SDN을 이용하는 모든 환경에서 적용이 가능할 것으로 예상하며 추후에는 AI와 접목하여 이상 패킷을 보다 효율적으로 판별할 수 있을 것으로 사료된다.

References

[1] B. C. Lee, J. D. Park, and B. S. Lee, “Technology trends of SDN, NFV, and cloud,” *Electron. and Telecommun. Trends*, vol. 30, no. 1, pp. 87-93, Feb. 2015.

[2] S. I. Lee, J. H. Yi, M. K. Shin, H. J. Kim, and S. W. Sohn, “Trend and forecast on standardization of SDN and NFV for smart internet,” *Electron. and Telecommun. Trends*, vol. 29, no. 2, pp. 79-86, Apr. 2014.

[3] G. H. Bang, D. J. Choi, and S. W. Bang, “A protection method using destination address packet sampling for SYN flooding attack in SDN environments,” *J. Korea Multimedia*

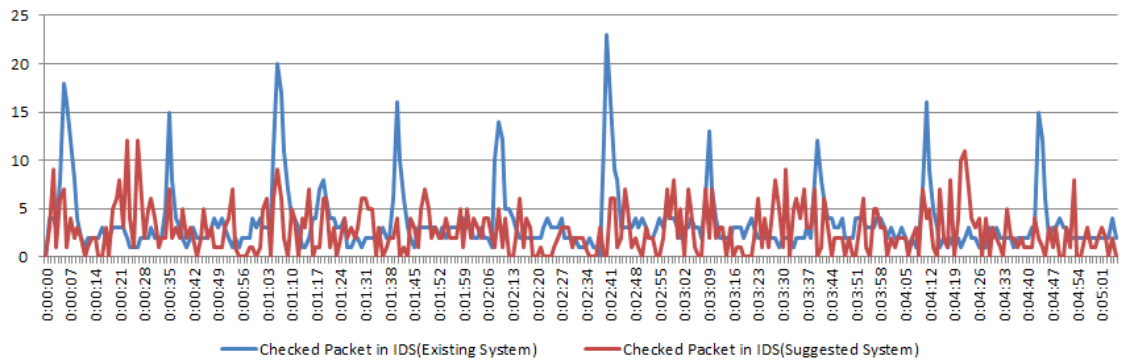

그림 15. IDS 패킷 검사 횟수
Fig. 15. IDS Packet Inspection Count

Soc., vol. 18, no. 1, pp. 35-41, Jan. 2015.

[4] D. S. Lee, "Improving detection capability of flow-based IDS in SDN," M.S. Thesis, Korea Advanced Institute of Science and Technology, Daejeon, Republic of Korea, 29 pages, 2015.

[5] M. Nugraha, I. Paramita, A. Musa, D. Choi, and B. Cho, "Utilizing OpenFlow and sFlow to detect and mitigate SYN flooding attack," *J. Korea Multimedia Soc.*, vol. 17, no. 8, pp. 988-994, Aug. 2014.

[6] J. H. Park and G. H. Kim, "Implementation and validation of the web DDoS shelter system(WDSS)," *J. Korea Info. Process. Soc.*, vol. 4, no. 4, pp. 135-140, Apr. 2015.

[7] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "SDN security: A survey. In future networks and services (SDN4FNS)," *2013 IEEE SDN*, pp. 1-7, Nov. 2013.

[8] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Commun. Surv. & Tuts.*, vol. 18, no. 1, pp. 623-654, Jul. 2016.

[9] N. G. Kang and T. W. Kwon, "Unauthorized software blocking techniques in software defined network(SDN) environments," *J. KIISC*, vol. 29, no. 2, pp. 393-399, Apr. 2019.

[10] S. G. Kim, "A study on the detection technique of DDoS attacks on the software-defined networks," *J. Korea Inst. Info., Electron., and Commun. Technol.*, vol. 13, no. 1, pp. 81-87, Feb. 2020.

[11] N. J. Choi, "Evolution of mobile network architecture: Cloud, software defined networking, and network functions virtualization," *Telecommun. Rev.*, vol. 25, no. 2, pp. 245-260, Apr. 2015.

[12] NOX/POX, NOXRepo(2015), Retrieved Jul., 10, 2021, from https://noxrepo.github.io/pox-doc/html/.

[13] G. Lee, D. Y. Lee, S. Y. Jeong, and W. K. Hong, "ONOS: Open source SDN distributed controller for Telcos," *J. KICS*, vol. 34, no. 12, pp. 10-19, Nov. 2017.

[14] D. Saikia and N. Malik, *An introduction to OpenMUL SDN suite*(2014), Retrieved Jul., 10, 2021, from www.openmul.org.

[15] Y. S. Seo and M. J. Lee, *SDN Introduction(2014)*, Retrieved Jun., 30, 2014, from Youngjin.com

[16] Z. Zarringhalami and M. K. Rafsanjani, "A survey on intrusion detection systems in computer networks," *J. Appl. Math. & Info.*, vol. 30, no. 5/6, pp. 847-864, Sep. 2012.

[17] S. Maza and M. Touahria, "Feature selection algorithms in intrusion detection system: A survey," *KSII Trans. Internet and Info. Syst.*, vol. 12, no. 10, pp. 5079-5099, Oct. 2018.

[18] P. Kabiri and Ali A. Ghorbani, "Research on intrusion detection and response: A survey," *Int. J. Netw. Secur.*, vol. 1, no. 2, pp. 84-102, Sep. 2005.

[19] A. Cortes, "Simulation of software defined networks with open network operating system and mininet," *Int. J. Comput. Sci. & Info. Technol.*, vol. 10, no. 5, pp. 21-32, Oct. 2018.

[20] I. Z. Bholebawa and U. D. Dalal, "Design and performance analysis of openflow-enabled network topologies using mininet," *Int. J. Comput. and Commun. Eng.*, vol. 5, no. 6, pp. 419-429, Nov. 2016.

[21] P. B. Patil, K. S. Bhagat, D. K. Kirange, and S. D. Patil, "Software defined networks using mininet," *Int. J. Recent Technol. and Eng.*, vol. 9, no. 1, pp. 843-849, May 2020.

이 예 원 (Ye-won Lee)

2018년 2월 : 금오공과대학교 전자공학부 졸업
2021년 8월 : 아주대학교 IT융합대학원 석사
<관심분야> 임베디드, 네트워크, 무선통신

[ORCID:0000-0001-6745-8436]

이 채 우 (Chae-woo Lee)

1985년 2월 : 서울대학교 제어계측공학과 졸업
1988년 8월 : 한국과학기술원 전기및전자과 졸업
1995년 8월 : University of Iowa. ECE dept. 졸업

1985년 : 금성통신
1988년~1999년 : KT
1999년~2001년 : Lucent Technologies
2002년~현재 : 아주대학교 전자공학과 교수