스마트네트워크서비스

최종 과제 3: SDN 기반 정책 제어 실습

1. 과제 개요

본 과제는 Software Defined Networking(SDN)의 핵심 동작인 **Packet-In 처리**,

Learning Switch 구현, 및 **특정 트래픽 제어(Firewall 정책 적용)** 를 실습하는 것을 목표로 한다.

Ryu 컨트롤러 스켈레톤 코드와 Mininet 토폴로지를 제공하며,

학생들은 스켈레톤 코드의 TODO 부분을 완성하여 **정책 기반 패킷 제어 기능**을 구현

---

2. 실습 환경

필수 구성 요소

- Ubuntu 20.04 / 22.04
- Mininet
- Ryu SDN Controller
- Python 3.x
- Wireshark(선택)

실행 방법

1. Ryu 컨트롤러 실행

```
$ ryu-manager simple_firewall_qos_skeleton.py
```

1. Mininet 토폴로지 실행

```
$ sudo python3 sdn_topology.py
```

1. Mininet CLI에서 ping 테스트

```
mininet> pingall
mininet> ping h1 h3
mininet> ping h2 h3
mininet> ping h3 h4
```

---

3. 제공 자료

아래 2개 파일이 제공된다.

파일명	설명
`sdn_topology.py`	Mininet용 네트워크 토폴로지
`simple_firewall_qos_skeleton.py`	Ryu 컨트롤러 스켈레톤 코드 (TODO 포함)

---

4. 과제 요구사항 (필수)

1) Learning Switch 기능 구현

- MAC 학습 테이블 구축
- 목적 MAC 주소가 학습된 경우 지정 포트로 포워딩
- 학습되지 않은 경우 FLOOD

2) IPv4 헤더 파싱 기능 구현

- Packet 객체에서 IPv4를 검사
- src_ip, dst_ip 값 추출
- 디버깅 로그에 출력

3) 트래픽 차단 정책(Firewall Rule) 구현

- 예: `('10.0.0.1', '10.0.0.3')` 또는 본인이 설정한 조합
- 차단 기준에 부합하는 경우
  - IPv4 match 기반 flow entry 설치
  - DROP 동작 수행
- 차단 동작이 ping 테스트에서 확인되어야 함

4) 보고서 작성

보고서에는 다음 내용을 필수 포함해야 한다.

(1) 실험 환경 및 토폴로지 설명

- 토폴로지 그림 직접 작성(또는 제공 코드 기반 그림)
- 각 호스트/스위치 역할 설명

(2) TODO 부분 완료 코드(핵심 부분 발췌 가능)

- 구현한 핵심 코드 또는 수정 사항 기술

(3) 테스트 결과

- `pingall` 결과
- 차단 정책이 적용된 ping 결과(예: h1 → h3 통신 실패)
- 정상 통신되는 다른 ping 결과
- Flow Table 출력:

```
sudo ovs-ofctl dump-flows s1
sudo ovs-ofctl dump-flows s2
```

(4) 정책 설계 설명

- 어떤 IP 쌍을 차단했는지
- 왜 해당 조합을 선택했는지
- 실제 네트워크 환경에서의 활용 가능성

(5) 결론 및 개선점

- 구현하면서 어려웠던 점
- 개선 가능한 기능 제안(QoS 정책, 로깅 추가 등)

---

5. 제출 방법

- 제출 파일:
  - 보고서(PDF) 1부

    - 수정한 Ryu 코드(.py)
    - 테스트 스크린샷 포함
- 제출 경로: LMS(과제 제출 란)
- 제출 기한: **YYYY-MM-DD (월) 23:59**

---

6. 평가 기준 (루브릭)

항목	설명	배점
SDN 기능 구현 정확성	Learning switch, Packet-In, Flow 설치 정확성	30점
Firewall 정책 구현	IP 기반 차단 정책 동작 여부	25점
분석 및 설명	보고서 구성, 실험 분석, 아키텍처 설명	25점
창의성 및 확장성	추가 기능, 정책 설계의 타당성	10점
보고서 문서 품질	캡처·표·그림 활용, 문서 완성도	10점
총점		**100점**

---

7. 주의 사항

- 제공 스켈레톤 코드의 TODO는 반드시 직접 구현해야 한다.
- 다른 학생 코드와 유사도가 과도하게 높을 경우 감점 또는 0점 처리될 수 있다.
- ChatGPT 등 생성형 AI의 결과를 그대로 제출할 경우 불이익이 발생할 수 있다.
  (참고용 사용은 허용하나 구현은 본인이 직접 해야 함)

---

1. 학생용 Mininet 토폴로지 (`sdn_topology.py`)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
스마트네트워크서비스 - SDN 실습용 토폴로지 (학생용 버전)
- 스위치 3개 (s1 - s2 - s3)
- 호스트 4개 (h1, h2, h3, h4)
- 외부 Ryu 컨트롤러 연결

TODO(보고서용):
```

```
  1) 이 토폴로지 구조를 직접 그림으로 그려오기
  2) 각 호스트/스위치의 역할을 정리해오기
"""

from mininet.net import Mininet
from mininet.node import RemoteController, OVSSwitch
from mininet.link import TCLink
from mininet.log import setLogLevel, info
from mininet.cli import CLI


def create_topology():
    net = Mininet(controller=RemoteController,
                  link=TCLink,
                  switch=OVSSwitch,
                  autoSetMacs=True,
                  autoStaticArp=True)

    info('*** Adding controller\n')
    # TODO: 필요하면 컨트롤러 IP/포트를 변경해보고, 왜 그런지 보고서에 설명해 보
기
    c0 = net.addController('c0',
                           controller=RemoteController,
                           ip='127.0.0.1',
                           port=6633)

    info('*** Adding switches\n')
    s1 = net.addSwitch('s1', protocols='OpenFlow13')
    s2 = net.addSwitch('s2', protocols='OpenFlow13')
    s3 = net.addSwitch('s3', protocols='OpenFlow13')

    info('*** Adding hosts\n')
    # TODO: 이 IP들이 같은 서브넷(10.0.0.0/24)에 속하는지 직접 확인해 보기
    h1 = net.addHost('h1', ip='10.0.0.1/24')
    h2 = net.addHost('h2', ip='10.0.0.2/24')
    h3 = net.addHost('h3', ip='10.0.0.3/24')
    h4 = net.addHost('h4', ip='10.0.0.4/24')
```

```
    info('*** Creating links\n')
    # host - switch
    net.addLink(h1, s1, bw=100)
    net.addLink(h2, s2, bw=100)
    net.addLink(h3, s2, bw=100)
    net.addLink(h4, s3, bw=100)

    # switch - switch (선형 토폴로지)
    net.addLink(s1, s2, bw=100)
    net.addLink(s2, s3, bw=100)

    info('*** Starting network\n')
    net.build()
    c0.start()
    s1.start([c0])
    s2.start([c0])
    s3.start([c0])

    info('*** Running CLI\n')
    CLI(net)

    info('*** Stopping network\n')
    net.stop()


if __name__ == '__main__':
    setLogLevel('info')
    create_topology()
```

2. 학생용 Ryu 컨트롤러 (`simple_firewall_student.py`)

- **Learning switch / IPv4 파싱 / Firewall rule 부분은 TODO**

  이 상태로 실행하면 단순 Flood 허브처럼 동작하고, **Firewall은 전혀 동작하지 않음**

```
# -*- coding: utf-8 -*-

"""
```

```
스마트네트워크서비스 - SDN 실습용 Ryu 컨트롤러 (학생용 버전)

목표:
  - 기본 learning switch 기능 구현
  - 특정 IP 쌍에 대한 트래픽을 차단하는 간단한 firewall 정책 구현

※ 본 파일은 스켈레톤 코드입니다.
   TODO 부분을 직접 작성하지 않으면 firewall 기능이 동작하지 않습니다.
"""

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCH
ER, set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet, ethernet, ipv4
from ryu.lib.packet import ether_types


class SimpleFirewallStudent(app_manager.RyuApp):
    # OpenFlow 1.3 사용
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleFirewallStudent, self).__init__(*args, **kwargs)
        # dpid(스위치 ID) 별 MAC 학습 테이블
        # 예: self.mac_to_port[dpid][mac] = port_no
        self.mac_to_port = {}

        # TODO: 차단할 IP 쌍을 정의할 것

        self.block_pairs = set()

    def add_flow(self, datapath, priority, match, actions, buffer_id=None):
        """스위치에 flow entry를 추가하는 헬퍼 함수"""
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
```

```
    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                         actions)]
    if buffer_id:
        mod = parser.OFPFlowMod(datapath=datapath,
                                buffer_id=buffer_id,
                                priority=priority,
                                match=match,
                                instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=datapath,
                                priority=priority,
                                match=match,
                                instructions=inst)
    datapath.send_msg(mod)

@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    """
    스위치가 컨트롤러에 처음 연결될 때 호출됨.
    - table-miss 엔트리 설정 (알 수 없는 패킷은 컨트롤러로 보냄)
    """
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    # TODO: 아래 table-miss 엔트리


@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    """
    스위치에서 컨트롤러로 올라온 Packet-In 이벤트 처리 함수
    """
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
```

```
        in_port = msg.match['in_port']
        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})

        pkt = packet.Packet(msg.data)
        eth = pkt.get_protocol(ethernet.ethernet)

        # LLDP 등은 무시
        if eth.ethertype == ether_types.ETH_TYPE_LLDP:
            return

        dst = eth.dst
        src = eth.src

        # ============================
        # 1) MAC 학습 (Learning Switch)
        # ============================
        # TODO: learning switch의 핵심 코드 한 줄을 작성하시오.
        #  - 의미: 해당 dpid에서 src MAC은 in_port를 통해 들어왔다고 학습
        #
        #
        #
        # 한 줄을 직접 작성
        # -------------------------------
        # 여기에 코드 작성
        # -------------------------------

        # ============================
        # 2) IPv4 헤더 파싱
        # ============================
        ip4 = pkt.get_protocol(ipv4.ipv4)
        src_ip = None
        dst_ip = None

        # TODO: IPv4 패킷인 경우, src_ip와 dst_ip를 추출하는 코드를 작성하시오.
        # 힌트: ip4.src, ip4.dst
        # -------------------------------
        # if ip4:
```

```
        #     src_ip = ...
        #     dst_ip = ...
        # --------------------------------

        # 디버깅용 로그 출력
        self.logger.info("dpid=%s in_port=%s src=%s dst=%s src_ip=%s dst_i
p=%s",
                         dpid, in_port, src, dst, src_ip, dst_ip)

        # ============================
        # 3) Firewall 정책 적용
        # ============================
        # TODO: block_pairs에 포함된 (src_ip, dst_ip) 조합이면
        #      해당 트래픽을 DROP하는 flow entry를 설치하시오.
        #
        #
        # --------------------------------
        # 여기에 코드 작성
        # --------------------------------

        # ============================
        # 4) 기본 포워딩 (학습 스위치 동작)
        # ============================
        # MAC 학습이 완료되면, dst MAC이 어느 포트에 있는지 알 수 있다.
        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            # 아직 모르면 flood
            out_port = ofproto.OFPP_FLOOD

        actions = [parser.OFPActionOutput(out_port)]

        # Flow 설치 (성능 향상을 위해)
        if msg.buffer_id != ofproto.OFP_NO_BUFFER:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=sr
c)
            self.add_flow(datapath, priority=10, match=match,
                          actions=actions, buffer_id=msg.buffer_id)
```

```
            return
        else:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=sr
c)
            self.add_flow(datapath, priority=10, match=match, actions=actions)

        # 실제 패킷 전송
        out = parser.OFPPacketOut(datapath=datapath,
                                  buffer_id=msg.buffer_id,
                                  in_port=in_port,
                                  actions=actions,
                                  data=msg.data)
        datapath.send_msg(out)
```
