1. KOREN Playground 활용 네트워크 P+M / C / D

IoT, SDN, NFV, Cloud 등의 최신 기술들이 혼재된 ICT 인프라 상에는 다양한 종류의 네트워크 트래픽이 발생한다. 각 인프라 기술 별로 트래픽의 구분 방식은 약간씩 상이하나 크게 다음과 같은 형태로 구분한다. 인프라 관리를 위해 운영자가 발생시키는 관리(Management) 트래픽, 인프라 상의 소프트웨어 구성 요소 간 대화를 위한 제어(Control) 트래픽, 서비스를 제공하는 과정에서 발생하는 막대한 데이터를 전송하기 위한 데이터(Data) 트래픽으로 구분한다. 트래픽 별로 별도의 네트워크를 운영하는 것이 언뜻 보기에는 불필요한 일인 것처럼 보이나 사실 많은 이점이 있다. 트래픽이 한 네트워크에서 혼재되면서 발생하는 보안 위협이 감소되고, 한 네트워크가 마비되었을 때 다른 네트워크를 통해 대체 가능하여 안정적인 Playground 운영 및 이를 위한 네트워크 troubleshooting이 용이하고, 고성능의 실험을 위한 안정적인 대역폭을 확보할 수 있다. 따라서 이런 장점으로 인해 IoT, SDN, NFV, Cloud로 대표되는 최신 ICT 인프라 기술들도 트래픽 별로 별도의 네트워크를 구성하는 것을 권장하고 있다. 따라서 GIST NetCS 연구실에서도 KOREN Playground에서 최신 인프라 기술들을 적용하고 안정적으로 운영하기 위하여 P+M / C / D 네트워크로 디자인했고, 실제로 해당 네트워크를 구축한 후 그 위에 Playground을 운영한다. 따라서 다양한 사이트들이 연결된 Playground에 대해 제대로 이해하기 위해서는 각 네트워크에 대해 제대로 이해할 필요가 있다.

P+M (Power + Management) 네트워크: Playground의 관리를 위해 운영자가 사용하는 관리 전용 네트워크

- 운영자가 각 박스의 설정을 위해 접속하거나(ssh), 필요한 소프트웨어 설치 시에 본 네트워크를 사용한다.
- 이에 더하여 원격지에 위치한 박스들을 제대로 관리하기 위해서는 Out-of-Band 네트워크를 통해 원격 박스의 전원 수준까지 제어하는 것이 필요하고, 근래에 판매되는 모든 서버들은 원격지에서 전원 제어를 가능케 하는 IPMI (Intelligent Platform Management Interface), WOL (Wake On Lan) 등의 인터페이스를 Out-of-Band 인터페이스 형태 (BMC, Baseboard Management Controller)로 구비하고 있다.
- KOREN Playground에서도 원격지의 모든 박스에 전원 제어를 위한 별도의 인터페이스로 Power 네트워크를 설정해 사용하고 있으며(GIST, 제주대, KOREN NOC), 이를 위한 IP 주소는 Management 네트워크와 동일한 네트워크 대역을 사용한다.
- 이 네트워크는 원격에서 운영자가 접속 가능해야 하며, 필요 소프트웨어를 외부에서 내려 받기 때문에 Public IP 주소 대역을 할당한다. 다만 관리 차원의 트래픽은 대용량의 데이터 트래픽을 발생시키지 않으므로 1Gbps 수준의 네트워크로 연결한다.

C (Control) 네트워크: Playground 내부에서 제어를 위한 메시지들이 전송되는 네트워크

- 실례로 Type-C 장비에 설치되는 OpenStack의 경우 Nova, Neutron, Keystone 등의

OpenStack 서비스 간 메시지가 전송되는 네트워크.

- Playground 내부의 제어 트래픽을 위한 네트워크이므로 외부에서의 간섭을 막기 위해 보통 Private 주소 대역으로 설정한다. 또한 고 대역폭을 요구하지 않으므로 1Gbps 네트워크로 구성한다.

D (Data) 네트워크: Playground 내부에서 서비스를 위한 박스 간 실제 데이터들이 전송되는 네트워크

- Apache Spark을 활용한 Analytics을 위한 데이터 트래픽들에 활용된다.
- 실제 서비스를 위한 데이터가 전송되는 네트워크이므로 고 대역폭이 요구되어, 10Gbps 대역폭의 네트워크로 구성하는 것을 기본으로 한다.

2. Type-O 내부 네트워크 구성

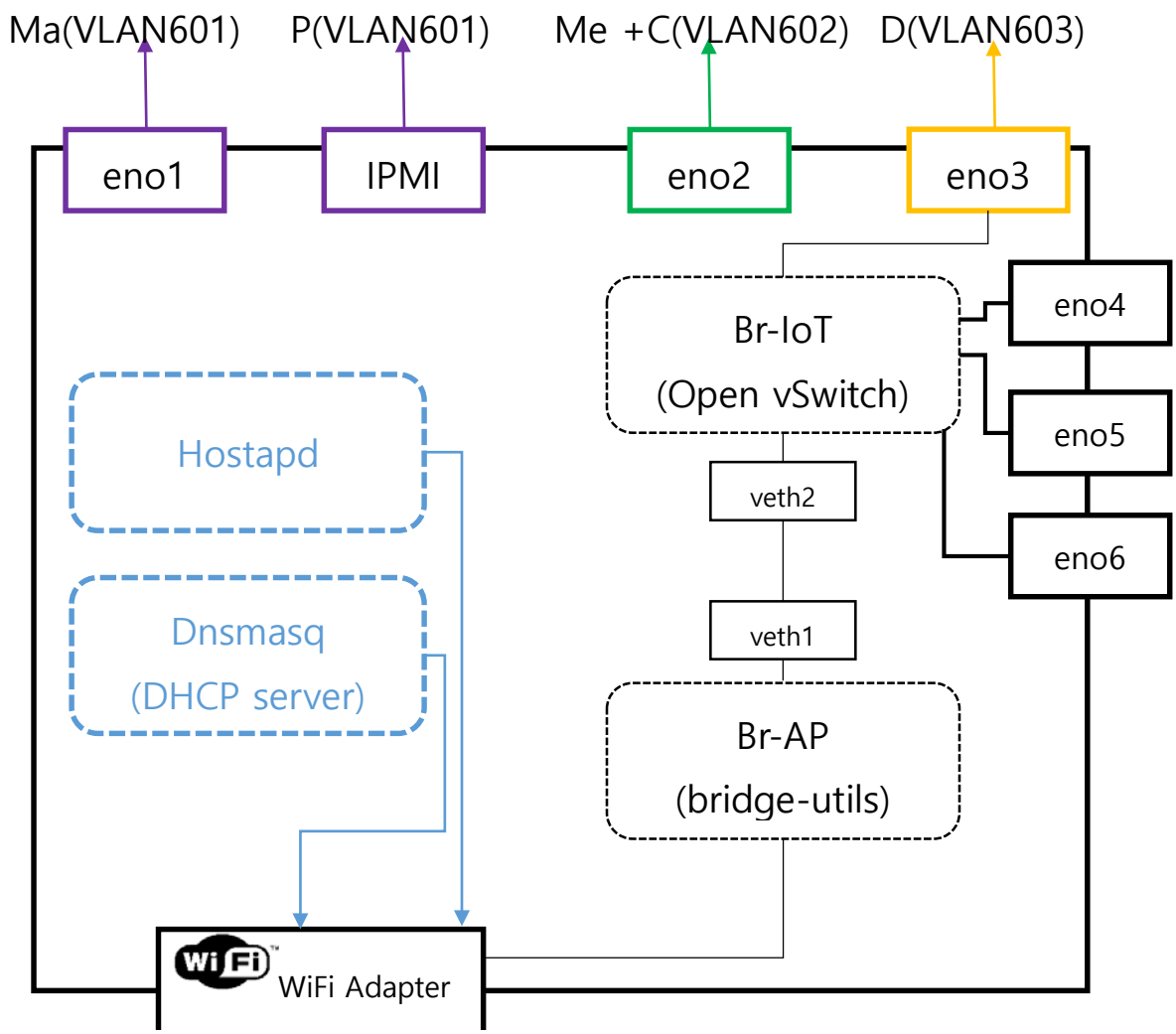

Figure 1. Type-O 내부 네트워크 구성

Type-O의 내부 네트워크 구성을 나타내는 그림은 다음과 같다. 주요 구성 Software로는 Hostapd, dnsmasq, Bridge-utils(리눅스 Bridge), Open vSwitch(이하 OVS) 네 개이다. 각각의 역할은 다음과 같다.

- Hostapd: 사용자의 설정에 따라 무선 AP(Access Point)를 만들고, 이에 대한 접근 권한을 설정하기 위한 Daemon이다. hostapd.conf 파일의 설정에 따라서 AP를 생성해주고 그에 대한 SSID 및 Password를 설정할 수 있는 기능을 제공해 준다.
- dnsmasq: DNS forwarder, DHCP(Dynamic Host Configuration Protocol) 서버의 역할을 하는 소프트웨어. Ubuntu, Debian을 포함한 리눅스 배포 판에 선 탑재되어 있는 소프트웨어로서, /etc/dnsmasq/dnsmasq.conf 파일의 설정에 따라서 DNS forwarder와 DHCP 서버의 역할을 수행하게 된다.
- OVS: NIC(Network Interface Card) 하드웨어의 가상화를 제공하고, 컴퓨터 네트워크에 사용되는 여러 프로토콜과 표준을 지원하는 목적으로 제작된 Open Source 프로젝트. 현재 Type-O에서는 Task 3-1, 인프라 슬라이싱을 위해 SDN 컨트롤러에 IoT 데이터를 전송하기 위한 용도로 사용한다.
- Bridge-utils(리눅스 Bridge): 리눅스 Ethernet Bridge 설정을 위한 기능을 제공하는 패키지이다. 리눅스 Bridge를 통해서 리눅스의 여러가지 Ethernet 장치들이 연결될 수 있다. Type-O의 리눅스 Bridge는 WiFi의 Authentication 이슈를 해결하기 위해 추가했다.

그림 1을 보면 veth1, veth2를 활용해서 Br-AP와 Br-IoT를 Patching 했다. 다음과 같은 구성을 한 이유는 Hostapd의 문제에서 비롯 되었다. 무선 인터페이스가 OVS Bridge에 직접적으로 연결되고, Hostapd로 무선 AP를 동작시키는 경우에 문제가 발생했기 때문이다. AP에 접근하려 하는 WiFi 디바이스들이 비밀번호를 정확하게 입력을 해도 비밀번호가 틀렸다고 나오는 문제가 있었다. 해당 문제는 Hostapd 자체의 문제이고 여러 커뮤니티에서도 제기된 현상이었다. 이런 이유로 무선 인터페이스를 리눅스 Bridge에 연결하고 리눅스 Bridge와 OVS Bridge를 Patching해서 연결하는 구주로 설계하여 해당 문제를 해결했다. 비밀번호를 가지고 AP에 접근이 가능함을 확인했다.

3. Type-O 자동 설치 및 Recover Script

다음과 같은 내부 네트워크 구성을 자동으로 진행해 주는 Shell Script를 작성해서 각 사이트에 배포된 Type-O 안에 저장해 두었다. Type-O 박스에 대한 재 작업이 필요한 경우 해당 Script를 통해서 네트워크 구성을 Recover해서 사용할 수 있도록 했다. Script의 문장과 각 문장의 역할은 다음과 같다.

- 패키지 설치 부분

```
apt-get purge -y hostapd bridge-utils openvswitch-switch
apt-get update && apt-get install -y hostapd dnsmasq bridge-utils openvswitch-switch
#Type 0의 동작을 위해 필요한 유틸들(Hostapd, Linux Bridge Util, OpenvSwitch)
#를 최신 버전으로 업데이트하기 위해서 지우고 다시 설치하는 과정을 수행
```

Figure 2. 패키지 설치 부분

내부 네트워크 구성을 위해 필요한 소프트웨어를 설치하는 부분이다. Hostapd, Bridge-utils, dnsmasq, OVS를 설치한다. 기존에 있던 패키지를 삭제하고 새로 설치하는 과정을 통해서 최신 버전으로 설치한다.

- Patching Interface 재생성 부분

```
if [    -n "ifconfig -a | grep veth1"   ]
then
        ifconfig veth1 down&&ifconfig veth2 down
        ip link delete veth1 type veth
fi
#기존에 연결되어 있는 Veth1, Veth2를 지워주는 과정

ip link add name veth1 type veth peer name veth2
ifconfig veth1 up && ifconfig veth2 up
#Open vSwitch Bridge(Br-IoT)와 리눅스 Bridge(Br-AP)의 patching에 사용되는 Veth1, Veth2를 생성
```

Figure 3. Patching Interface 재생성

Br-IoT, Br-AP Patching을 위해서 기존에 있던 Interface를 삭제하고 새로 생성해주는 동작을 한다. Patching을 위해서 가상 Ethernet Interface를 생성한다. 이름은 veth1, veth2이고, veth1는 리눅스 Bridge(Br-AP)에 연결되고, veth2는 OVS Bridge(Br-IoT)에 연결된다.

- Br-IoT 생성 및 Interface 연결 부분

```
ovs-vsctl add-br br-IoT
ovs-vsctl add-port br-IoT veth2
#Patching에 사용되는 Veth2를 Br-IoT에 연결
ovs-vsctl add-port br-IoT eno3
ovs-vsctl add-port br-IoT eno7
ovs-vsctl add-port br-IoT eno4
ovs-vsctl add-port br-IoT eno5
ovs-vsctl add-port br-IoT eno6
ovs-vsctl set-controller br-IoT tcp:                :6633
#Task 3-1 인프라 슬라이싱을 위해 해당 Bridge를 ONOS Controller에 연결
echo -e "₩novs switch setting₩n"
ovs-vsctl show
#Open vSwitch로 만든 Bridge인 Br-IoT를 생성해주는 과정
#Br-IoT에 연결될 D(Data Plane) Interface들을 연결해주는 과정을 수행
```

Figure 4. Br-IoT 생성 및 Interface 연결

IoT 데이터를 모으고, SDN Controller의 제어를 통해서 데이터를 전송하기 위해 사용되는 OVS Bridge인 Br-IoT를 생성하는 과정이다. Br-IoT를 생성하고 IoT 데이터를 전송할 Interface들을 연결한다. Eno3 Inteface는 D(Data Plane)을 통해서 다른 사이트로 데이터를 보내기 위해서 사용하는 Interface로 사용한다. Eno3을 제외한 나머지 유선 인터페이스들(eno4, eno5, eno6)은 유선으로 Type-O에 연결되는 IoT 데이터를 받기 위해서 사용된다.

- Br-AP 생성 및 무선 Interface 연결 부분

```
brctl addbr br-AP
#리눅스 Bridge 생성
iw dev wlx485d601fbca4 set 4addr on
#<wlx485d601fbca4>는 무선 인터페이스의 장치 이름으로 사이트마다 다름
#해당 설정을 통해서 무선 인터페이스가 IPv4 주소를 가질 수 있도록 함
brctl addif br-AP veth1 && sudo brctl addif br-AP wlx485d601fbca4
echo -e "\nlinux bridge seting\n"
#무선 인터페이스와 Veth1를 Br-AP에 연결
```

Figure 5. Br-AP 생성 및 무선 Interface 연결

Bridge-utils 패키지의 명령어인 brctl 명령어를 사용해서 리눅스 Brdige인 Br-AP를 생성하고 Interface를 연결하는 부분이다. Br-AP에는 무선 인터페이스가 연결되게 되고, Patching Interface인 veth1이 연결된다. "iw dev <무선 Interface 장치명>" 명령어는 무선 Interface가 IPv4의 주소를 가질 수 있도록 설정해 주는 부분이다. 해당 명령어를 수행하지 않으면 무선 Interface는 Br-AP에 연결되지 않는다.

- dnsamsq.conf에 설정 값 삽입 부분

```
sed -i 's/no-resolv/no-resolv/g' /etc/dnsmasq.conf
sed -i 's/dhcp-range=interface:wlx485d601fbca4,                    ,12h/dhcp-range=wlx485d601fbca4
,                          ,12h/g' /etc/dnsmasq.conf
sed -i 's/server=8.8.8.8/server=8.8.8.8/g' /etc/dnsmasq.conf
#dnsmasq(DHCP 서버)의 설정 파일
#DHCP IP pool은 Data Plane으로 여러 사이트에서 동시에 통신하는 상황을 고려 겹치지 않도록 설정 완료
```

Figure 6. dnsmasq.conf 설정 값 삽입

DHCP 서버로 동작하는 dnsmasq의 설정 값을 입력하는 동작을 하게 된다. WiFi로 연결되는 IoT 장치들이 IP 주소를 받을 수 있도록 동작하는 역할을 한다. DHCP 서버로 동작할 Interface와 DHCP IP Pool을 지정해서 WiFi로 연결되는 장치들이 IP를 받을 수 있도록 한다. IP Pool은 각 사이트에 연결되는 IoT 장치들의 IP가 겹치지 않도록 사이트 별로 20개씩 설정하여 배포 했다.

- Hostapd.conf에 설정 값 삽입 부분

```
echo -e "interface=wlx485d601fbca4
#bridge=br-AP
driver=nl80211
ssid=type0_GIST
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=3
wpa_passphrase=
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP"> ~/hostapd.conf
#무선 AP Daemon Hostapd 설정 파일
#Interface의 이름은 위에서 사이트 마다 다름
#AP의 SSID 역시 사이트마다 다르게 설정 완료
```

Figure 7. hostapd.conf 설정 값 삽입

AP Deamon인 hostapd의 설정 값을 입력하는 부분으로 동작하게 된다. 각 주요 항목 별 의미는 다음과 같다.

- interface 어떤 interface 를 AP 모드로 할지를 정한다.
- driver : 브리지모드로 할게 아니면 nl80211 로 설정한다.
- ssid : AP 의 세션이름
- hw_mode : 무선랜 타입 (a/b/g..)
- channel : 무선랜에서 사용할 채널.
- macaddr_acl : 맥인증을 하려고 할때 사용한다. 사용하지 않기 때문에 0 으로 하였다.

Interface의 이름과 ssid의 이름은 각 사이트 별로 장치에 따라 이름이 다르기 때문에 해당 부분은 사이트에 따라 다르다.