K-ONE 기술 문서 #18

LoRa, Wi-Fi를 포함하는 이기종 통신 인터페이스에 대응 가능한 K-Cluster 기반의 IoT-Cloud Hub 설계 및 활용방안

Document No. K-ONE #18
Version 0.4
Date 2017. 05. 14.
Author(s) 윤희범, 남택호

K-ONE

ONK
OpenNetworkingKorea

■ 문서의 연혁

버전	날짜	작성 내용	비고
0.1	2017-05-08	기술문서 아웃라인 작성	
0.2	2015-05-12	초안 작성- 남택호	
0.3	2015-05-13	초안 작성- 윤희범	
0.4	2015-05-14	참고문헌 수정	

본 문서는 2015년도 정부(미래창조과학부)의 재원으로 정보통신 기술진흥센터의 지원을 받아 수행된 연구임 (No. B0190-15-2012, 글로벌 SDN/NFV 공개소프트웨어 핵심 모듈/기능 개발)

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. B0190-15-2012, Global SDN/NFV OpenSource Software Core Module/Function Development)

기술문서 요약

K-ONE 컨소시엄의 SDN/NFV/Cloud 오픈소스 프로젝트 공동 개발/실증 환경인 Edge-Cloud 모델에 대응하는 클러스터형 테스트베드인 K-Cluster를 중심으로 LoRa, Wi-Fi를 포함하는 이기종 통신 인터페이스에 대응 가능한 K-Cluster 기반의 IoT-Cloud Hub 요소에 대해 설계 및 구축 과정을 설명한다. 그리고 IoT-Cloud Hub의 통제를 받는 LoRa, Wifi 통신 인터페이스에 기반의 소규모 IoT 환경을 구축하고 SDN, IO Visor 중심의 IoT-Cloud Hub 소프트웨어를 설계한다. 상기 구축한 환경에서 동작하는 SmartX IoT-Cloud 서비스에 대해 설명하고, 이를 활용하여 현재까지 K-Cluster와 연계되어 구현된 IoT-Cloud Hub의 기능을 검증하며, 향후 활용 방안에 대해 기술한다.

Contents

K-ONE #18. LoRa, Wi-Fi를 포함하는 이기종 통신 인터페이스에 대응 가능한 K-Cluster 기반의 IoT-Cloud Hub 설계 및 활용 방안

그림 목차

표 목차

K-ONE #18. LoRa, Wi-Fi를 포함하는 이기종 통신 인터페이스에 대응 가능한 K-Cluster 기반의 IoT-Cloud Hub 설계 및 활용방안

1. 서론 및 IoT-Cloud Hub 정의

1.1. 목적

o 본 기술문서는 Edge-Cloud 모델에 대응하는 클러스터형 테스트베드인 K-Cluster를 중심으로 LoRa, Wi-Fi를 포함하는 이기종 통신 인터페이스에 대응 가능한 IoT-Cloud Hub의 설계 및 구성 방안을 제시한다. 따라서 다양한 이슈를 반영하는 SDN/NFV/Cloud 기술의 연구개발을 위해 제작된 K-Cluster에 대해 설명하고 K-Cluster를 활용한 하나의 Use Case로써 IoT-Cloud Hub 요소를 K-Cluster에 추가한다. 또한 단일 사이트로 형성된 소규모 IoT 환경을 구축하여 위 환경에서 동작하는 IoT-Cloud Hub의 소프트웨어 요소에 대해 설명하며, 향후 개발될 요소에 대해 기술한다.

o IoT-Cloud Hub는 정확하게 소프트웨어를 지칭하는 명칭이며 물리적인 하드웨어 구성에 국한되지 않고, IoT-Cloud Hub는 3G/4G,5G, WiFi, LoRa 등과 같은 다양한 IoT 통신 인터페이스에 모두 대응되는 Multi-Access를 추구한다. IoT-Cloud Hub는 단일 사이트에 배치된 하나의 논리적인 Edge-Cloud인 K-Cluster의 한 요소로써 동작가능하며, IoT 네트워크를 구성하는 네트워크 장비에 따라 제공하는 소프트웨어 구현이 상이할 수 있다. 본 기술문서에는 WiFi, LoRa 통신 인터페이스에 대응되는 IoT-Cloud Hub 모델을 제시한다.

o 최근 부상하는 사물인터넷(IoT: Internet of Things) 관련 서비스들은 분산된 IoT 단말들로부터 수집된 대량의 데이터를 저장/처리하기 위해 클라우드 인프라와 연계되고 있다[1]. 데이터 그 자체만으로는 가치를 온전히 살릴 수 없기 때문에 사물 인터넷 시대에는 빠르게 생성되는 많은 양의 데이터를 안전하게 전송하고 실시간으로 수집, 관리, 분석 할 필요성이 있다[2]. 즉, 다수의 IoT 기기와 클라우드간의 안정적인 데이터 전달과 이에 대한 지속적인 모니터링이 필요하다. 그러나 전통적인 방식의 네트워크 인프라에서 IoT-Cloud 서비스를 도입할 경우 실무자들은 네트워크 장비를 구매하고, 물리적으로 연결하고, 필요한 서비스를 설정하고, 기존 인프라와의 호환성을 점검하고 안정화 작업도 거쳐야 한다. 이런 과정은 서비스를 추가할 때마다 반복된다. 사용자 특성이나 사업 환경이 시시각각 달라질 수 있는 IoT 시대에 구축 절차와 물리적 시간 소요가 거듭된다면 안정적인 서비스 제공에 문제를 야기할 수 있다. 가령 IoT-Cloud 서비스를 이용할 때 문제가 발생할 경우 해당 문제가 물리적인 네트워크 장비 설정에서 발생한 것인지, IoT-Cloud 서비스 내에서 발생한 문제인지 파악하기가 어렵다. 또 물리적인 네트워크 장비의 문제일 경우 정확하게 어떠한 장비에서 발생한 문제인지, 장비 설정을 바꾼 후 네트워크에 어떠한 영향을 미칠지 알 수 없을뿐더러, 원활한 서비스를 제공 할 수 없다.

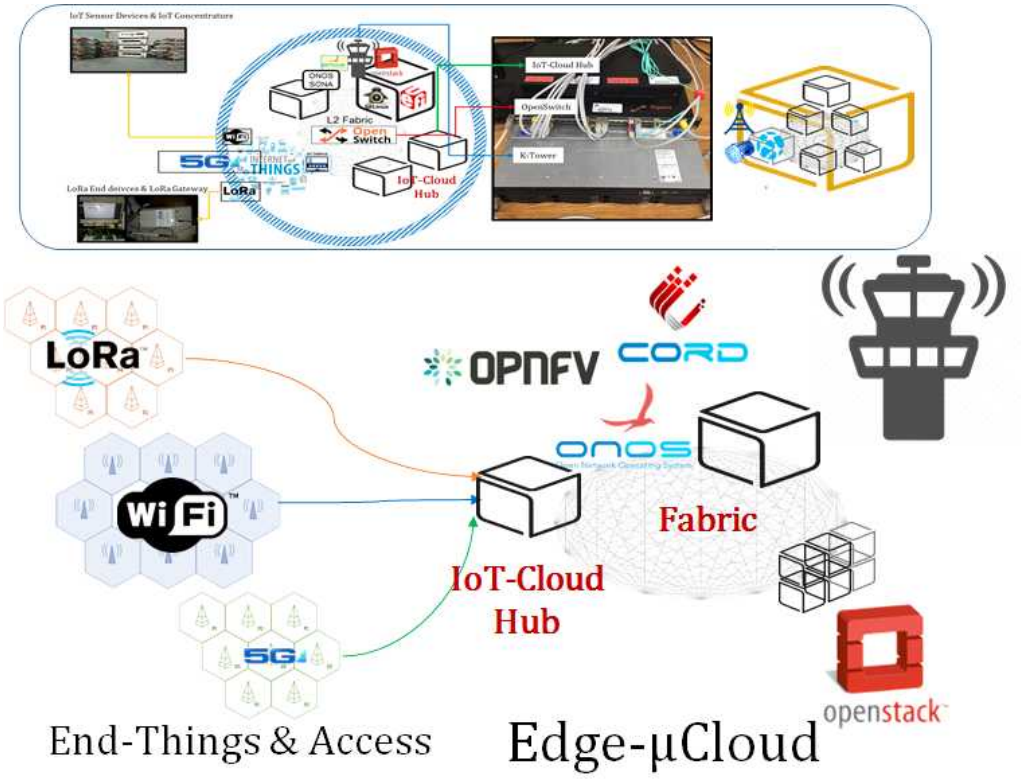
o 소프트웨어-정의-네트워킹(SDN: Software-Defined Networking)을 이용하면 IoT-Cloud 간 여러 전송 경로들에 대한 손쉬운 관제가 가능하다. 관리자는 SDN 제어기를 통해 다수의 네트워크 장치들에 대한 트래픽 관리, 경로설정 등을 한 번에 할 수 있어 IoT-Cloud 서비스를 효율적으로 지원할 수 있기 때문이다.

1.2. IoT-Cloud Hub의 정의 및 개요

o IoT-Cloud Hub는 K-Cluster의 구성요소로써 WiFi, LoRa를 포함하는 통신 인터페이스에 대응 가능하고 이를 바탕으로 IoT 기기들과 IoT-Cloud Hub 산하에 존재하는 IoT Network를 관제하는 소프트웨어로 정의한다.

o Edge-Cloud 와 연관된 다양한 SDI 실증 및 연구개발을 위한 테스트베드인 K-Cluster를 활용하여 IoT-Cloud에 대응되는 서비스를 개발 및 실증할 필요가 있을 때, IoT-Cloud Hub는 IoT와 Edge-Cloud를 유연하고 안정적으로 연결 할 수 있도록 지원하고 IoT-Cloud Hub 산하에 있는 IoT 장비, 네트워크 장비를 손쉽게 관리할 수 있어 IoT-Cloud 서비스에 대한 개발 및 실증을 효율적으로 진행할 수 있다.

o 부상하는 IoT-Cloud 서비스들을 효율적으로 지원하기 위해서는 IoT-Cloud 간의 안정적이고 지속적인 데이터 전송이 필수적이다. 본 논문에서는 초융합형 SmartX Box에 기반한 K-ONE Playground 상의 K-Cluster를 활용해 IoT-Cloud 서비스를 위한 다양한 모니터링 데이터를 보다 안정되게 수집/전송함에 있어서 오픈소스 메시징 시스템인 Kafka를 활용하고 ONOS(Open Network Operation System) SDN 제어기에 기반한 SDN 응용을 적용함으로써 실제적인 IoT-SDN-Cloud 환경에서의 안정적인 IoT-Cloud 서비스에 대응하는 모니터링 데이터 전송을 위한 유연한 수집 경로 설정이 가능함을 확인한다.



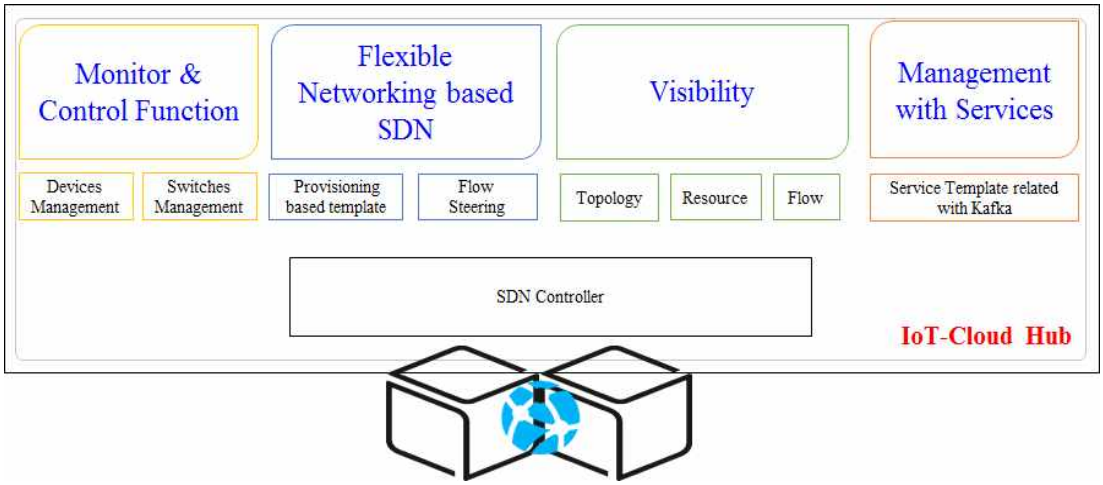
<그림 1: Multi-Access Edge Cloud for IoT-Cloud Services>

o 즉, <그림 1>과 같이 K-Cluster를 IoT 환경과 연계하기 위해서 IoT-Cloud Hub를 교두보로 활용한다. 구체적으로 IoT 환경의 다양한 센서 및 엑츄에이터들 과의 통신에 대응하기 위한 3G/4G, WiFi, LoRa IoT등과 같은 무선 인터페이스를 통해 IoT를 K-Cluster에 물리적으로 연계할 수 있다. 이를 통해 IoT 환경에서 발생하는 다양한 소량의 데이터를 전달받아 K-Cluster로 전달하는 진입로의 역할을 수행한다. 이와 동시에 연결된 IoT 환경들을 Cloud 입장에서 관제하는 대장 역할을 수행하면서 메쉬 형태로 구성되는 IoT와 Cloud 간의 네트워킹을 지원해야 한다. 즉 IoT-Cloud Hub라는 중간자를 통해 IoT와 K-Cluster가 담당하는 μCloud 간의 이질적인 네트워크 연결을 극복하면서 매끄럽게 연계하는 것이 중요하다.

o 상기 IoT-Cloud에 대응하는 실증을 지원하기 위해 IoT-Cloud Hub는 다음과 같은 요구사항을 만족해야 한다.

- IoT-Cloud Hub는 DevOps 자동화 도구를 활용하여 소프트웨어 기반으로 IoT 기기에 대한 전반적인 자동화된 관제 (Monitor & Control)가 되어야 함.
- IoT-Cloud Hub는 K-Cluster의 한 요소로써, IoT와 K-Cluster 사이의 유연한 네트워킹을 제공해야 함.
- IoT 환경에 대한 자동화된 관제와 SDN을 활용해 IoT 네트워킹을 조율하고, 가시성을 제공해야함.

- IoT-Cloud Hub는 IoT 산하 네트워크에 대해 SDN을 기반으로 WiFi, LoRa와 같은 통신 인터페이스를 지원해야 한다.



<그림 2: IoT-Cloud Hub 기능>

o 상기 기술한 요구사항을 만족하기 위해 IoT-Cloud Hub는 <그림 2>와 같은 기능들을 제공해야 한다. 우선적으로 Monitor& Control Function 기능은 IoT-Cloud Hub 산하 IoT 네트워크에 대해 지속적인 모니터링 및 장비들을 컨트롤한다. SDN 컨트롤러의 도움을 받아 IoT 기기와 네트워크 장비의 하드웨어/소프트웨어 정보들을 토대로 어떤 기기들이 얼마나, 어떻게 토폴로지를 구성하고 있는지 알 수 있고 이에 따라 해당 장비에 대한 개별 컨트롤이 가능하다.

o SDN 컨트롤러의 도움을 받아 IoT-Cloud Hub는 단순히 IoT-Cloud의 교두보 역할 뿐 아니라 유기적이고 안정적인 데이터 전송이 이루어지도록 IoT-Cloud 서비스에 대한 서비스 단위의 데이터 경로를 만들어 상이한 서비스들 간 데이터 경로를 구분한다. 이를 위해 초기 IoT-Cloud 서비스가 실행 될 경우, 서비스 개발자는 일종의 사용서와 같은 템플릿을 IoT-Cloud Hub로 전달한다. 그리고 IoT-Cloud Hub는 템플릿을 바탕으로 관리하고 있는 토폴로지에서 적당한 Path를 찾아 SDN을 이용해 서비스 전용 데이터 경로를 설정한다. 뿐만 아니라 IoT-Cloud Hub는 주기적으로 트래픽 및 예기치 않은 링크의 단절과 같은 상황을 고려해 동적으로 데이터 경로를 바꿔 IoT-Cloud 서비스가 네트워크 장애 없이 원활하게 동작하도록 한다.

o IoT-Cloud Hub 산하 관리되는 WiFi, LoRa IoT 네트워크에 대한 전반적인 Visibility를 제공하여 Operator 입장에서 원활한 데이터 전송이 진행되는지 즉, 교두보 역할을 제대로 하고 있는지 손쉽게 확인할 수 있도록 한다. 모니터링 및 컨트롤 되는 IoT 장비와 네트워크 장비를 비롯하여 관리되는 토폴로지 수준부터 Container, VM 과 같은 가상 자원들의 상태 정보, 네트워크와 관련해 설치된

Flow 정보들까지 다양한 수준의 Visibility를 제공한다.

o 서비스 단위 수준으로 데이터 경로를 구분해 맞춤형 네트워킹을 제공하기 위해서 서비스 개발자는 IoT-Cloud Hub로 서비스를 운용하기 위한 템플릿을 전달한다. 구축한 IoT-Cloud Hub는 SmartX IoT-Cloud Service를 대상으로 프로토타입 개발을 진행 중이기에 템플릿 역시 SmartX IoT-Cloud 서비스를 대상으로 작성된다. SmartX IoT-Cloud 서비스는 기본적으로 Apache Kafka를 사용하기 때문에 템플릿은 기본적으로 서비스를 운용하고자 할 때 사용하려고하는 Kafka component와 IP가 mapping되어 있다. IoT-Cloud Hub는 템플릿을 파싱하여 서비스에 맞는 네트워크 환경을 설정하여 원활한 서비스를 지원한다.

2. 관린 기술 배경

o 2절은 IoT-Cloud Hub를 구축하기 위해 활용한 오픈소스 프로젝트들에 대해 간략하게 기술하며, LoRaWAN에 대해 설명한다.

2.1. LoRaWAN

o LoRa는 사물인터넷 통신의 요구 특성에 따라 LoRa라는 무선 변조 기술을 이용한다. LoRa는 장거리에 걸쳐 저전력의 신호 송수신이 가능하게 하는 RF 인터페이스/Physical layer를 포함하는 기술이다.

o LoRa는 Chirp Spread Spectrum 변조 방식을 사용하며, 제한된 Channel Bandwidth에서 낮은 데이터 전송률 대신 신호 송수신 거리를 확보한다. Chirp Spread Spectrum 변조 방식은 시간이 지남에 따라 선형적으로 주파수가 증가/감소하는 Chirp 펄스로 주파수를 변조, 광대역에 걸쳐 정보를 인코딩한다. 보내고자 하는 원 신호에 확산 코드를 곱하면 신호의 에너지는 낮고 넓게 펴지며, 노이즈는 확산되지 않은 채로 수신된다.

o 수신기는 원 신호로의 복원을 위해 다시 역확산 코드를 곱하게 되며, 노이즈에는 확산되는 효과가 발생하여 수신기로부터 무시됨에 따라 Coding Gain이 발생한다. 또한, 서로 직교하는 확산 대역 코드를 사용하여 동시에 여러 단말이 통신 할 수 있는 Coexistance 측면의 이점이 있다.

o LoRaWAN[3]은 물리 계층에 대한 신호 변조 기술인 LoRa의 상위 MAC 계층과 통신 규약에 대한 정의를 포함하는 기술이다. LoRaWAN 프로토콜 클래스와 각 클래스별 통신 규격 및 MAC 메시지 포맷 등이 표준 문서에 정의되어 있다.

o 넓은 신호 송수신 범위와 낮은 저력 소비량을 극대화 하면서 수백만 개의 무선 센서 노드를 게이트웨이에 연결 할 수 있기 때문에 인프라 구축비용이 낮고 실외

환경에서도 안정적으로 작동함에 따라 광범위한 저속 무선 모니터링 및 제어 설계에 매우 적합하다고 설명되고 있다.

o SDI 환경에서의 차세대 네트워크 구조에서는 LoRaWAN 구성요소의 Network Server와 Application Server의 가상화 및 SDN 연동 구조를 통해 기능 향상이 이루어질 수 있다.

2.2. ONOS

o SDN(Software-defined Networking)이란 네트워크 제어 기능(control plane)이 물리적 네트워크와 분리되어 있는 네트워크 구조를 말한다. 하드웨어와 분리된 제어 기능은 제어기(controller)로 불리는 스위치나 라우터가 아닌 별도의 장치에서 구현된다. 이 제어 기능은 SDN의 두 가지 요소와 상호작용하는데 하나는 응용이고 다른 하나는 하드웨어에 구현된 추상화 계층이다. 제어 기능을 통해서 응용은 네트워크의 다양한 정보를 얻을 수 있고, 반대로 네트워크 역시 응용 요구 사항 등의 정보를 얻을 수 있다. 이러한 핵심적인 역할 때문에, 제어 기능은 종종 네트워크 운영체제(Network OS)라 호칭되고 있다. 하드웨어와 소프트웨어를 분리하여, 소프트웨어는 자연스럽게 네트워크 전체를 관장하게 되는 중앙 집중형으로 구성되고, 전체 네트워크가 중앙의 제어 기능을 통해 단일화하여 통제 받음으로써, 효율적인 통신이 이루어지고, 일괄적인 정책 적용을 통해 빠르면서도 안전한 네트워크 운용이 이루어지게 된다. Openflow는 SDN을 위한 인터페이스 표준 기술로 정의되며, ONF(Open Networking Foundation)[4]를 중심으로 연구/개발된 가장 대표적인 SDN을 위한 프로토콜이다. OpenFlow는 제어 평면과 데이터 평면을 분리하고, 이들을 통신 할 수 있도록 하는 통신 규약이다. OpenFlow enabled 장비는 기존과 다르게 SDN 제어기에 의한 제어가 가능하고 고정적이던 기존의 네트워크에서 벗어나 정책이나 사용 목적에 따라 관리 변경이 가능하므로 유연하게 활용할 수 있는 장점이 있다[5].

o ONOS(Open Network Operating System)[6]는 ON.Lab에서 개발하였으며, 제어기는 Floodlight에 기반이며 분산형 SDN 제어기 중 가장 대표적이다. 중앙 집중형식의 SDN 구조는 단일 SDN 제어기가 중앙에서 SDN을 구성하는 모든 네트워크 장치들을 제어/관리해 제어 평면의 병목 현상문제를 야기 할 수 있다. 이를 극복하기 위한 방안으로 분산형 SDN 제어기가 나왔고, 중앙 집중식 SDN 제어기에서 야기되는 병목 현상 문제를 완화 할 수 있으므로, 최근 각광 받고 있다. 그림 1은 ONOS의 구조를 개념적으로 나타낸다. 그림 1에서 볼 수 있듯이, 최 하단에 위치한 네트워크 스위칭 장비들을 분산형 구조를 갖는 ONOS 제어기가 제어하는 구조이다.

2.3. Docker, Apache Kafka

o Docker[7]는 Linux 기반의 Container 기술을 활용한 오픈소스 소프트웨어로서, 가상 머신과 유사한 특징을 갖지만 보다 경량화된 형태로 활용 가능하다. IoT-Cloud 환경에서는 다수의 기기에 필요한 소프트웨어의 배포 및 관리가 요구되므로 가상화 기술의 도입이 필연적인데, Docker Container는 별도의 게스트 OS없이 호스트의 커널을 공유하는 까닭에, 한정적인 자원을 갖고 있는 IoT 환경에서도 유용하게 활용 가능하다. Docker Hub라는 공용 저장소를 활용하여 손쉽게 원하는 이미지를 주고 받을 수 있는 장점도 갖고 있다.

o Apache Kafka[8]는 대용량의 실시간 로그처리에 특화되어 설계된 메시징 시스템으로써 publish-subscribe 모델을 기반으로 동작하며 크게 producer, consumer, broker로 구성된다. 또한 분산 시스템을 기본으로 설계되었기 때문에, 기종 메시징 시스템에 비해 분산 및 복제 구성을 손쉽게 할 수 있고 파일 시스템에 메시지를 저장하기 때문에 별도의 설정을 하지 않아도 데이터의 영속성이 보장된다. Producer에서 전달한 데이터는 정해진 topic에 따라 분류되며 여러 대의 broker로 구성될 경우 topic은 몇 개의 partitions으로 구성되며 각 partitions들은 여러 개의 사본을 가질 수 있다. 때문에 일부 broker에 장애가 생기더라도 다른 broker에서 원하는 topic에서 필요한 데이터를 전달 받을 수 있다.

o 기존 범용 메시징 시스템(ActiveMQ, RabbitMQ)과 비교하였을 때 다양한 기능을 지원하지는 않지만, 수많은 IoT 기기에서 생성되는 데이터에 대응하여 많은 양의 데이터를 안정적으로 빠르게 전달하는 측면에서는 매우 우수하다.

2.4. IO Visor

o IO Visor는 리눅스 재단의 협력 오픈소스 프로젝트로써 확장성 및 유연성을 겸비한 고성능의 프로그래밍 가능한 데이터 평면을 통해 기존보다 발전된 형태의 네트워킹을 포함하는 커널 내부의 입출력 가상화를 목적으로 한다. 대표적으로 패킷 트레이싱을 통한 보안 측면에 활용할 수 있으며, 세부적으로는 유저 영역과 커널 영역에 걸쳐 동작하는 구조적 특징으로 인해 패킷뿐만 아니라 사용자 입출력, 동작중인 프로세스 등을 트레이싱 하도록 지원할 수 있다 [9].

o IO Visor 프로젝트의 핵심 기술 중에 하나인 BCC(Berkeley Packet Filter Complier Collection) [10]는 네트워크 모니터링을 목적으로 패킷을 필터링하고 분석하기 위한 도구, eBPF(extended Berkeley Packet Filter) [11] 프로그램이 커널 내부에서 동작할 수 있도록 컴파일 해주는 역할을 담당한다. 즉 eBPF 기반의 리눅스 입출력, 네트워킹, 트레이싱을 쉽게 수행할 수 있도록 제공하는 도구들의 집

합으로 볼 수 있다.

o 상기 언급한 BCC, eBPF를 활용하여 K-Cluster에 연계된 IoT-Cloud Hub 내 물리적인 또는 논리적인 네트워크 인터페이스와 오픈소스 가상 스위치 OVS(Open vSwitch)의 브릿지를 대상으로 패킷 수준의 가시성을 제공하기 위한 기능을 구현한다.

3. K-ONE Playground와 IoT-Cloud Hub

o 3절은 IoT-Cloud Hub를 사용하는 전체적인 인프라 환경에 대해 기술한다. IoT-Cloud Hub는 K-Cluster의 한 요소로써 사용되기에 3.1절은 K-Cluster에 대해 기술한다. 또한 K-Cluster를 활용해 운용중인 K-ONE Playground을 3.2절에서 기술하며 IoT-Cloud Hub 산하에 IoT-Cloud 서비스에 대한 연구 및 실증을 위해 운용 중인 SmartX-mini Playground를 3.3절에 소개한다. 3.4절은 K-Cluster와 SmartX-mini Playground를 연결해주는 IoT-Cloud Hub에 대해 기술한다. 마지막으로 3.5절은 IoT-Cloud Hub의 검증을 위해 활용한 연구실에서 운용 중인 SmartX IoT-Cloud Service에 대해 간략하게 기술한다.

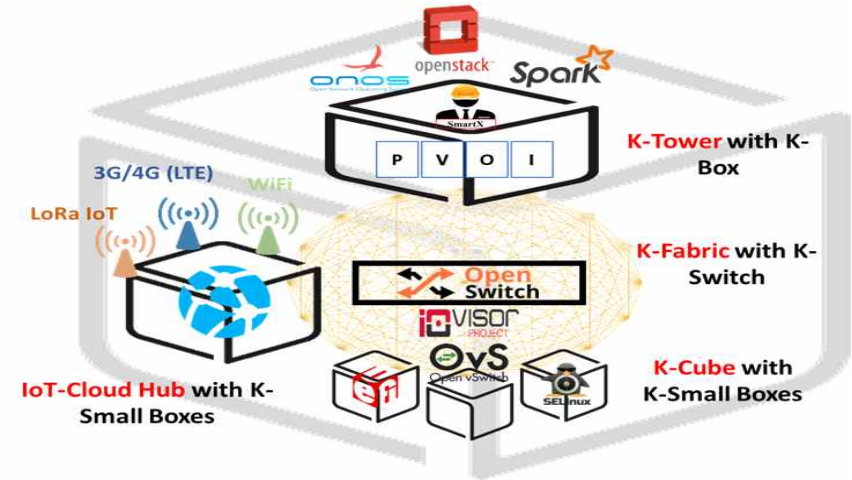
3.1. K-Cluster

o K-Cluster 및 K-ONE Playground의 자세한 설명은 'K-ONE 기술문서 16#_멀티사이트 클라우드 실증환경에 대응하는 K-Cluster 중심의 K-ONE Playground 설계 및 활용 방안'을 참조하면 되고, 본 기술문서에는 간략하게 기술한다[12].

o K-Cluster는 소프트웨어 정의 인프라 패러다임에 대응하는 다양한 SDN/NFV/Cloud 실증을 제공하기 위한, Edge-Cloud 모델에 대응하는 경제적인 소규모 클러스터 형태의 테스트베드로 정의한다. K-Cluster의 예상 활용처는 대규모의 ICT 서비스를 제공하는데 필요한 안정적이고 고성능의 인프라를 구성하는데 쓰이는 것이 아니라, Edge-Cloud와 연관된 다양한 SDI 실증을 및 연구개발을 위한 적정 규모의 테스트베드를 확보하기 어려운 연구자, 개발자를 위한 소규모 테스트베드 환경을 기본 활용처로 상정한다. 물론 K-Cluster도 안정적이고 고성능의 자원을 제공하는 것이 주요 요구사항 중 하나로 실제 Production 환경에 적용하는데 부족함이 없으나, 요구사항의 우선순위 측면에서 다양한 Edge-Cloud 기술을 하나의 테스트베드에 수용하기 위한 자원 측면의 유연성과 구축 경제성을 설계 시 가장 높은 우선순위로 한다.

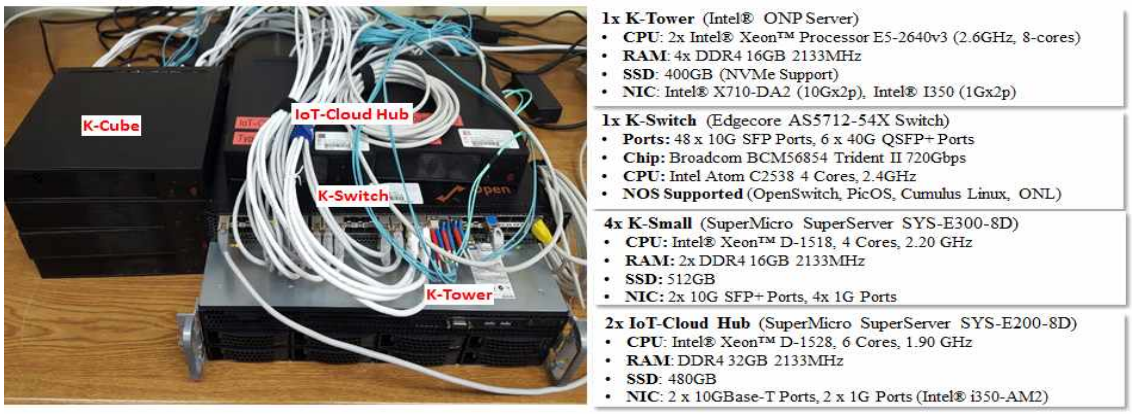
o 먼저 K-Tower는 K-Cluster 전체를 DevOps(개발운영병행체제) 관점에서 관리/운영의 자동화를 위한 단일 또는 복수의 K-Box들로 구성된 관제(monitor &

control)을 전담한다. K-Tower는 내부적으로 오픈소스 DevOps 도구를 활용하여 자동화를 지원하는 Provisioning 센터, Visibility 센터, Orchestration센터, 그리고 Intelligence 센터를 위한 지원역할을 담당하는 소프트웨어 적인 기능들을 포함한다. K-Cluster의 하단부에 위치한 K-Cube는 다수의 동종 K-Small 박스 자원들로 구성된 K-Cluster 내부의 클러스터로써 계산/저장/네트워킹을 담당하는 공유된 자원집합을 지칭한다. K-Fabric은 K-Cluster의 중앙에 위치하여 물리/가상 네트워킹 기능들을 활용하여 패브릭 형태의 밀결합된 네트워킹을 구성한다. 이를 통해 K-Clsuter 구성 요소들인 K-Tower, K-Cube, IoT-Data Hub 간의 빠르고(fast) 유연하며(flexible) 안정된(reliable) 연결성을 제공한다.



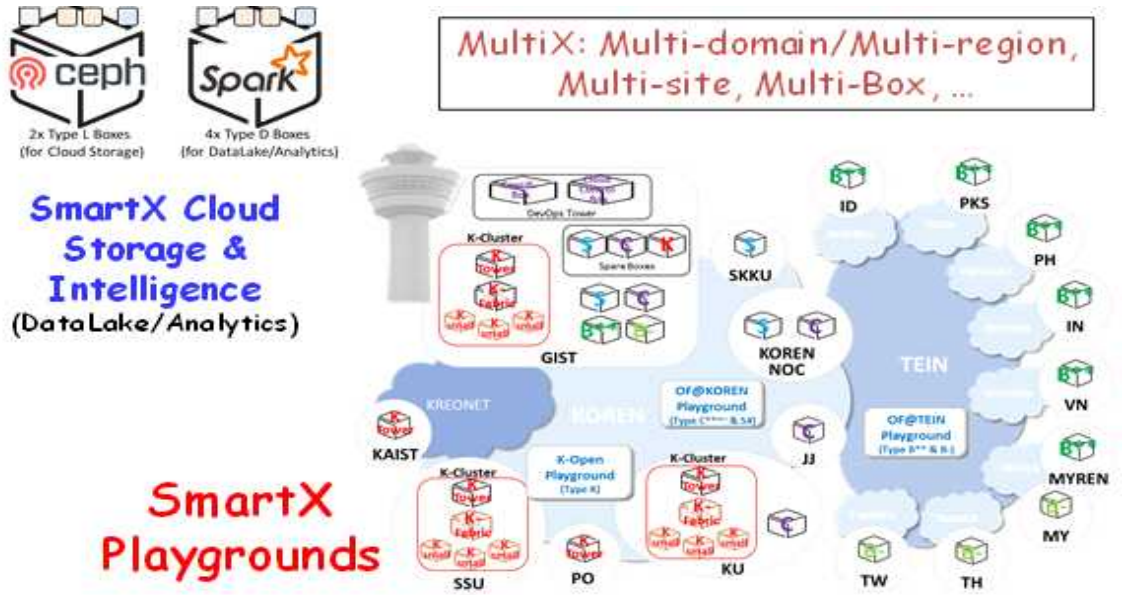
<그림 3: K-Cluster 구성요소>

o K-Tower, K-Fabric, K-Cube, IoT-Data Hub등으로 구성된 K-Cluster 모델을 활용하면 에지 클라우드에서 요구하는 SDN/NFV/Cloud 통합과 FastData/ BigData/ HPC 기반의 다양한 서비스를 저렴한 상용 화이트박스들을 활용하여 유연하게 대응할 수 있다. 또한 IoT-Cloud Hub를 통한 IoT-Cloud 서비스들 역시 K-Cluster와 연계해 IoT-Edge-Cloud 실증 환경을 구성할 수 있다.



<그림 4: K-Cluster Hardware 구성>

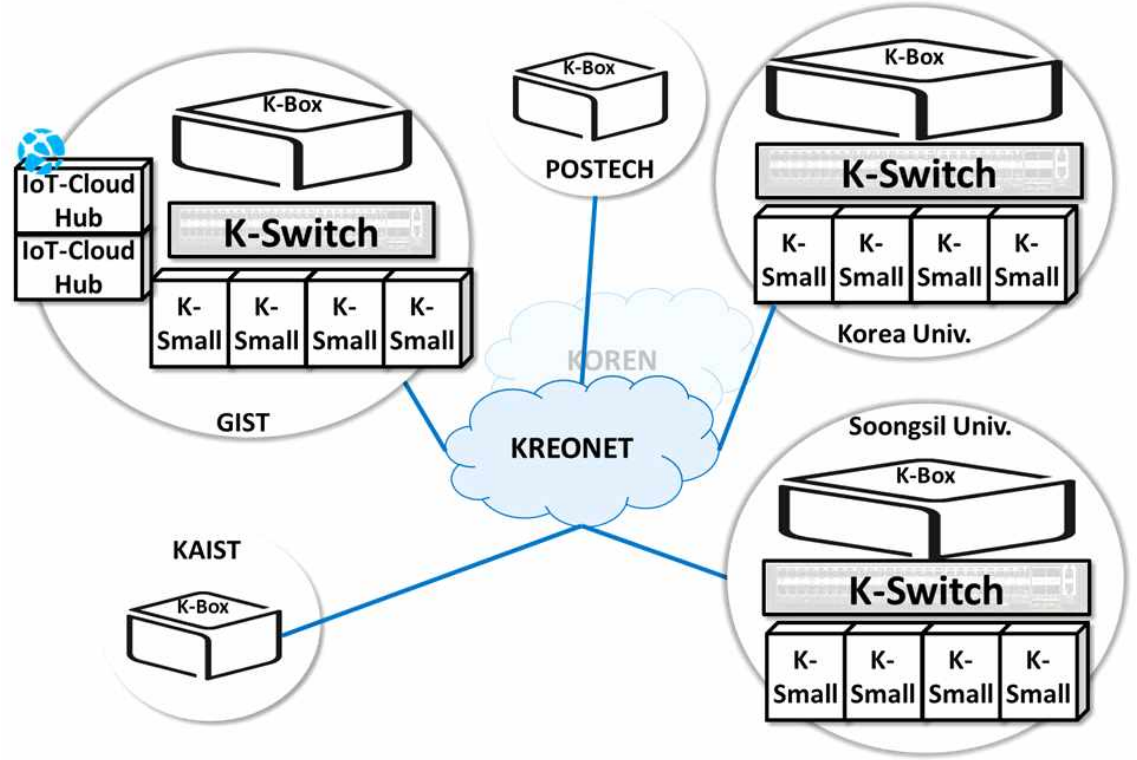
3.2. K-ONE Playground



<그림 5: 소프트웨어 정의 인프라 패러다임의 MultiX Challenges>

o 고립된 하나의 박스, 하나의 클러스터, 하나의 사이트를 고도화 하는 전략은 기술적으로 이미 많이 성숙되어 자원의 용량 및 성능 측면에서 큰 향상을 기대하는 것은 어려우며, 근본적으로 자원 활용의 유연성/확장성 측면의 커다란 단점을 내포하고 있다. 게다가 다수의 멀티 자원요소들 간의 연결성을 제공하는 클러스터링, 네트워킹 기술이 고도화 됨에 따라 다수의 요소를 연결함으로써 발생하는 병목현상, 성능저하 보다 자원의 클러스터링을 통한 유연하고 확장성 있는 활용이 큰 가치를 갖게 되었다. 따라서 최근에는 <그림 5>의 우측 상단과 같이 멀티박스를 넘어서 멀티리전, 멀티사이트, 심지어 멀티도메인과 같이 다양한 멀티 이슈에 대응하는 MultiX가 ICT 인프라의 연구 키워드의 하나로 부상하고 있다.

o K-ONE 컨소시엄 기관을 대상으로 멀티사이트 SDN/NFV/Cloud 통합 실증환경인 K-ONE Playground을 <그림 5>과 같이 구성하였으며, 이를 위해 아래와 같은 노력을 수행하였다.



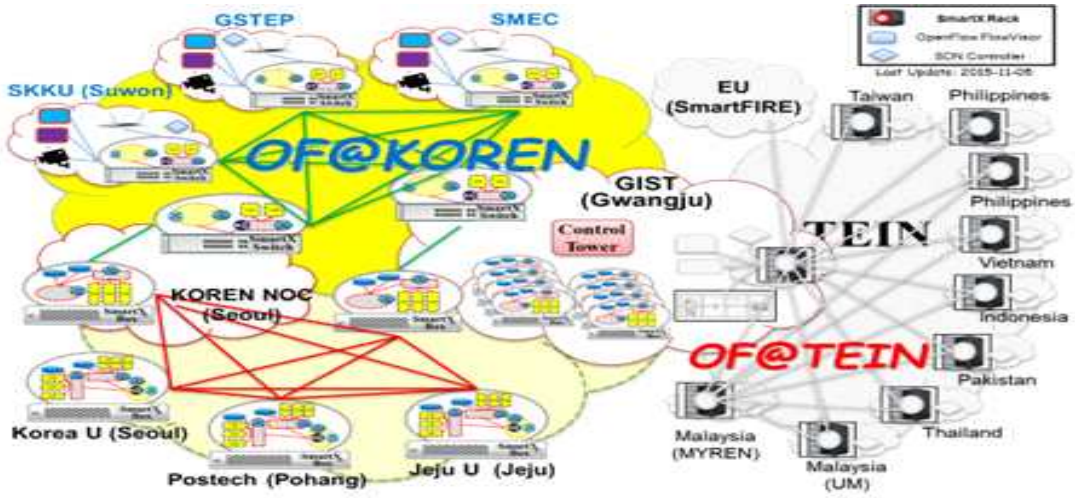
<그림 6: K-ONE Playground의 구축 현황도>

o 우선 K-ONE Playground의 하드웨어 배치 측면에서는, 1차년도에 K-Box들을 전체 컨소시엄 기관 사이트들에 한 대씩 배포하였으며, 2차년도에는 NFV, Cloud를 연구주제로 수행하는 GIST, 숭실대학교, 고려대학교를 대상으로 K-Switch와 K-Small 들을 추가로 배포함으로써 앞서 제시한 K-Cluster 하드웨어 시제품을 세 사이트에 구성하였다.

o K-ONE 컨소시엄의 다섯 참여기관 사이트를 하나의 Playground로 구성하기 위하여 각 사이트의 여건에 맞추어 KREONET/KOREN 연구망을 들이는 작업을 진행하였다. 결과적으로 1-2차년도에 걸친 노력으로 <그림 6>과 같이 각 사이트에 배포한 K-Box/K-Cluster는 KREONET/KOREN 연구망으로 모두 연동하였다. 특히 K-Cluster가 배포된 GIST, 숭실대학교, 고려대학교는 별도의 KREONET Public IP 대역을 각각 할당받아 사이트별로 독립적인 실증환경을 구축하였다.

3.3. SmartX-mini Playground

o IoT와 Cloud가 접목되는 IoT-Cloud 서비스를 효율적으로 체험하려면, 계산/저장/네트워킹 자원을 융합하여 단일 Box로 통합한 초융합형(hyper-converged) SmartX Box 들을 기반으로 소위 SmartX Playground로 불리는 실증형 공용개발환경을 구축하여 활용하면 효과적이다[8]. 차세대 기술로 손꼽히고 있는 IoT,



<그림 7: SmartX Playground 구성>

Cloud, SDN의 기술들을 실증할 때, 시뮬레이션 기반의 실험보다 실제 인프라 상에서 실증 실험을 하는 것이 더 효과적이고 직관적일 수 있다. 실제 실험 환경을 구성하기 위해 국내에서는 2012년~2013년 내에 국내5개와 국외 7개 사이트에 한국형 융합 지원박스(서버)로써 SmartX Box를 <그림 7>와 같이 설계하였고, SmartX Playground라는 이름의 미래인터넷 테스트베드 운용을 시작하였다[13]. Playground는 의미 그대로 사용자들이 하고 싶은 다양한 실증을 자유롭게 할 수 있는 환경을 제공하는 테스트베드를 의미한다. SmartX Box는 계산/저장/네트워킹 자원을 하나의 박스 단위로 확보해 다양한 서비스를 실증할 수 있는 초융합형 자원 박스로 정의한다.

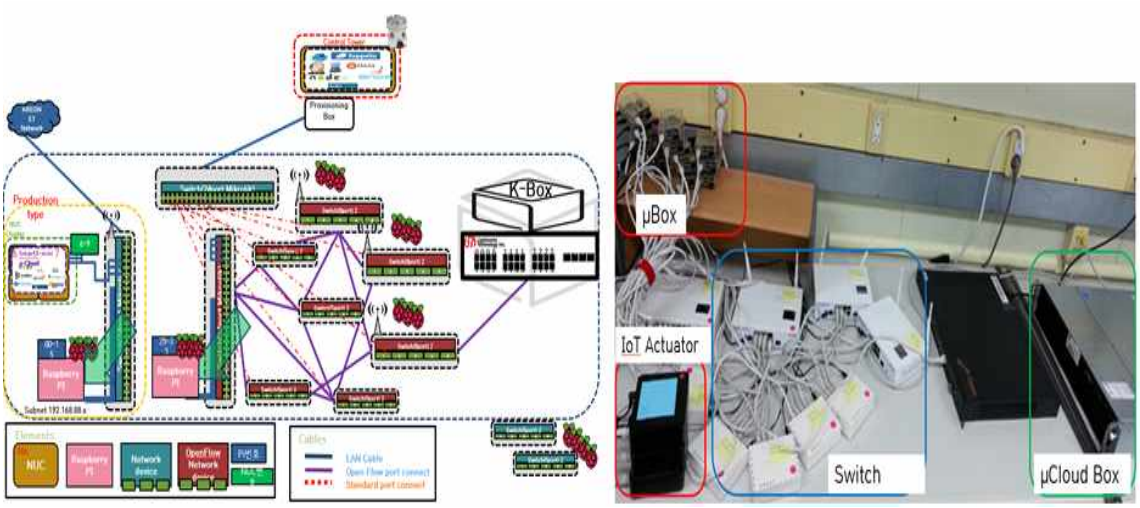
o SmartX-mini Playground는 SmartX Playground의 참조 모델로써, IoT-SDN-Cloud 환경을 구성하고 실험을 실증하기 위한 단일 사이트내의 테스트베드이다. IoT 기기를 다루는 경우에 기존의 고가 서버 자원을 사용하는 경우 자원의 낭비로 이어질 수 있기 때문에 보다 저렴한 비용으로 구축할 수 있는 환경이 필요하다. 이에 SmartX-mini Playground는 이러한 요구사항을 충족시키기 위해 상대적으로 저렴한 하드웨어를 활용하여 구축하였다.

Device	name	brand	aim	OS
μBox	Raspberry Pi2	Raspberry Pi Foundation	IoT device	hypriot OS
IoT Actuator	NUC	Intel	IoT Gateway	Ubuntu 14.0.4 LTS
DevOps Tower	NUC	Intel	DevOps, SDN Controller	Ubuntu 14.0.4 LTS
μCloud Box	ONP Server	Intel	μCloud	Ubuntu 14.0.4 LTS
Switch	CRS	Mikrotik	OpenFlow switch	Router OS MIPSBE

	125-24G-1s-2HnD			6.34
Switch	CRS 109-8G-1s-2HnD	Mikrotik	OpenFlow switch	Router OS MIPSBE 6.34
Switch	RB 750GL	Mikrotik	OpenFlow switch	Router OS MIPSBE 6.34

<표 1: SmartX-mini Playground 하드웨어 사양>

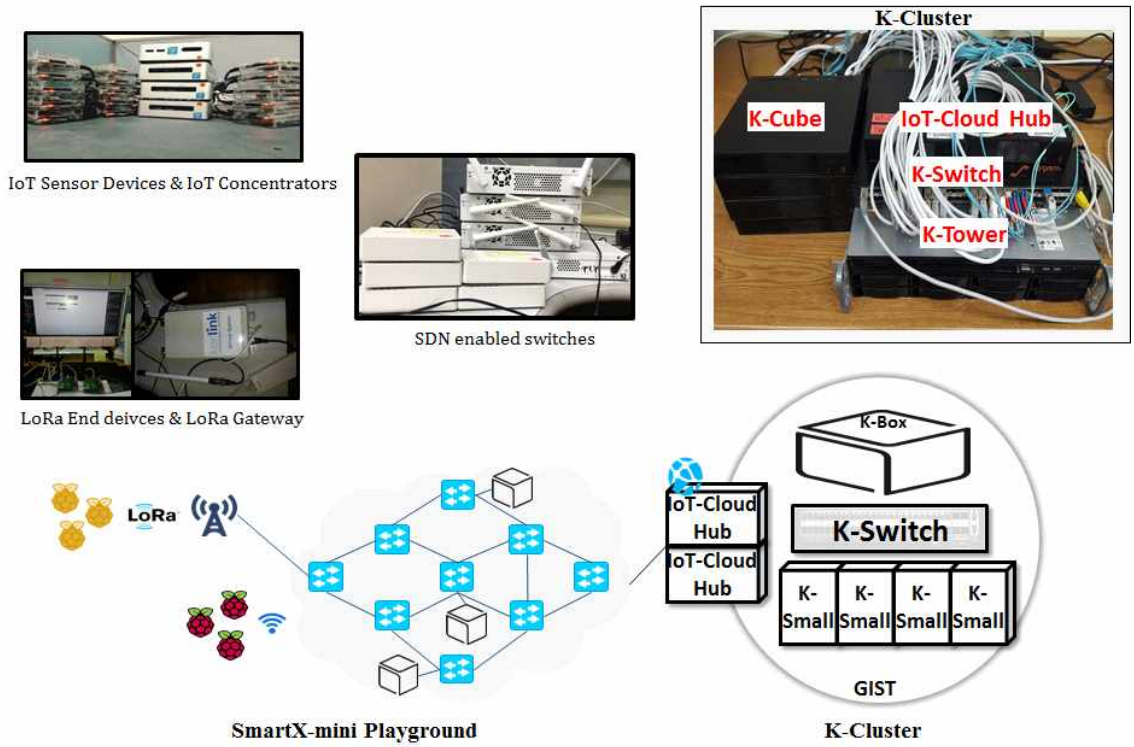
o IoT-Cloud 서비스 실증을 위한 공용개발환경을 제공하는 SmartX-mini Playground는 IoT 종단(end)에 적합한 장치인 μBox, 소규모로 꾸며진 Cloud를 대표하는 장치인 μCloud Box, IoT Gateway를 대표하는 IoT-Actuator, OpenFlow SDN을 지원하는 스위치, 그리고 전체 공용개발환경을 과제하기 위한 DevOps(개발운영병행체제) Tower를 포함한다. SmartX-mini Playground는 소규모의 저렴한 장비들을 이용해 IoT-SDN-Cloud 환경에서의 다양한 실험을 실증하기 위한 테스트베드로써 사용된 하드웨어는 <표1> 와 같다. 전체적인 테스트베드 구성은 <그림 8>와 같다. μBox 하드웨어는 Raspberry Pi 2를 사용하고, IoT Actuator 및 DevOps Tower의 하드웨어로는 Intel 사의 소형 PC인 NUC을 활용한다. 또한 μCloud Box는 Intel ONP Server를 이용, SDN 스위치는 Mikrotik 사의 스위치를 사용한다. μBox는 13대, IoT Actuator는 3대, 9대의 switch, 한 대의 μCloud Box를 이용한다.



<그림 8: SmartX-mini Playground 구성>

3.4. K-Cluster를 활용한 IoT-Cloud 대응형 테스트 베드 구성

o <그림 9>와 같이 K-Cluster와 SmartX-mini Playground를 활용해 IoT-Cloud Hub를 실증하기 위한 테스트베드를 구성한다. 기존 구축되어 운용 중이던 SmartX-mini Playground에 LoRaWAN을 위한 게이트웨이를 추가하고 K-Cluster의 요소인 IoT-Cloud Hub가 SmartX-mini Playground와 연결된다.



<그림 9: IoT-Cloud Hub 실증을 위한 K-Cluster+SmartX-mini Playground 환경>

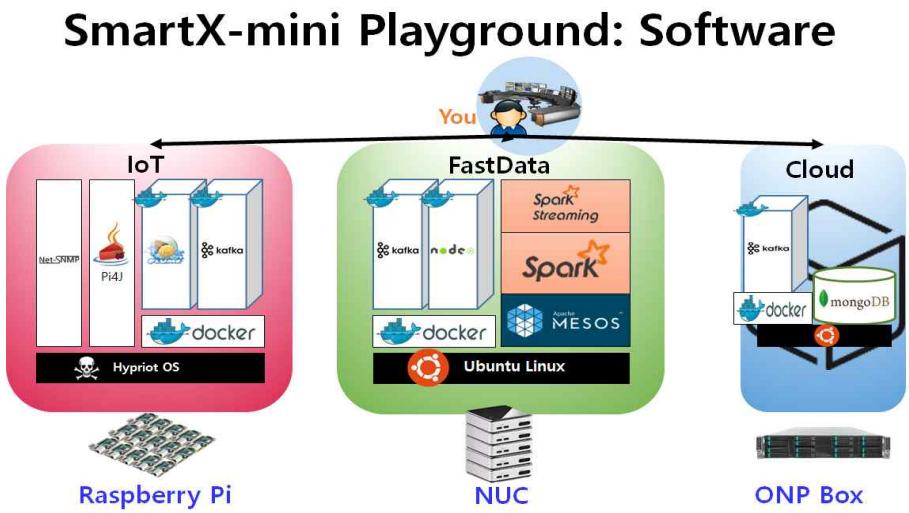
o LoRaWAN Gateway로는 Kerlink사의 Gateway를 사용하며, LoRaWAN end device는 IMST사 제품인 IM880b Model을 사용한다. SmartX-mini Playground는 SDN-enabled switch인 MikroTik 스위치로 <그림9>와 같은 토폴로지를 나타내며, 토폴로지 중앙에는 IoT-Actuator가 위치한다. 또한 K-Cluster가 연결되어 기존의 μCloud Box와 DevOps Tower의 역할을 대신하도록 한다. 즉, IoT device가 생산해낸 데이터는 IoT-Cloud Hub의 제어 아래 SDN Network를 통해 안정적으로 Cloud로 모이게 되며, 원활한 IoT-Cloud 서비스가 운용되게 된다.

3.5. SmartX IoT-Cloud Service: IoT-Cloud 자원 모니터링 서비스

o IoT-Cloud 자원 모니터링 서비스는 Apache Kafka 기반으로 다량의 데이터를 신속하게 전달하기에 적합한 서비스이지만, 물리적인 경로 상에서 발생하는 문제에 대한 대응에는 취약점을 갖고 있다. 때문에 해당 서비스에 ONOS SDN 제어기에 기반한 SDN 응용을 활용해 유연한 데이터 경로 설정을 적용함으로써 SDN 네트워크를 통한 보다 더 유연하고 안정적인 IoT-Cloud 서비스 관리/제어 가능성을 확인할 수 있다.

o 다수의 IoT 기기들을 제어하기 위해서는 기본적으로 모든 IoT 기기들의 상태를 알아야 할 필요가 있기에 이러한 환경을 효율적으로 다루기 위한 지속적이고 신뢰할 수 있는 운영 데이터 수집이 요구된다. 따라서 SmartX-mini Playground에서도 IoT-Cloud를 위한 자원 모니터링 서비스를 지원함으로써, 환경의 운영에 필요

한 데이터를 모니터링하고 분석할 필요가 있다. 이때, 상대적으로 자원이 부족한 IoT 기기의 특성상 자원의 효율적인 활용과 실시간 처리 역시 다뤄져야 할 문제이다. 이를 해결하기 위해 하이퍼바이저(Hypervisor)에 비해 적은 자원을 요구하는 컨테이너(Container) 기술 중 Docker를 활용하여 다수의 IoT 기기에 원하는 서비스를 손쉽게 배포하고 운영하고자 한다.



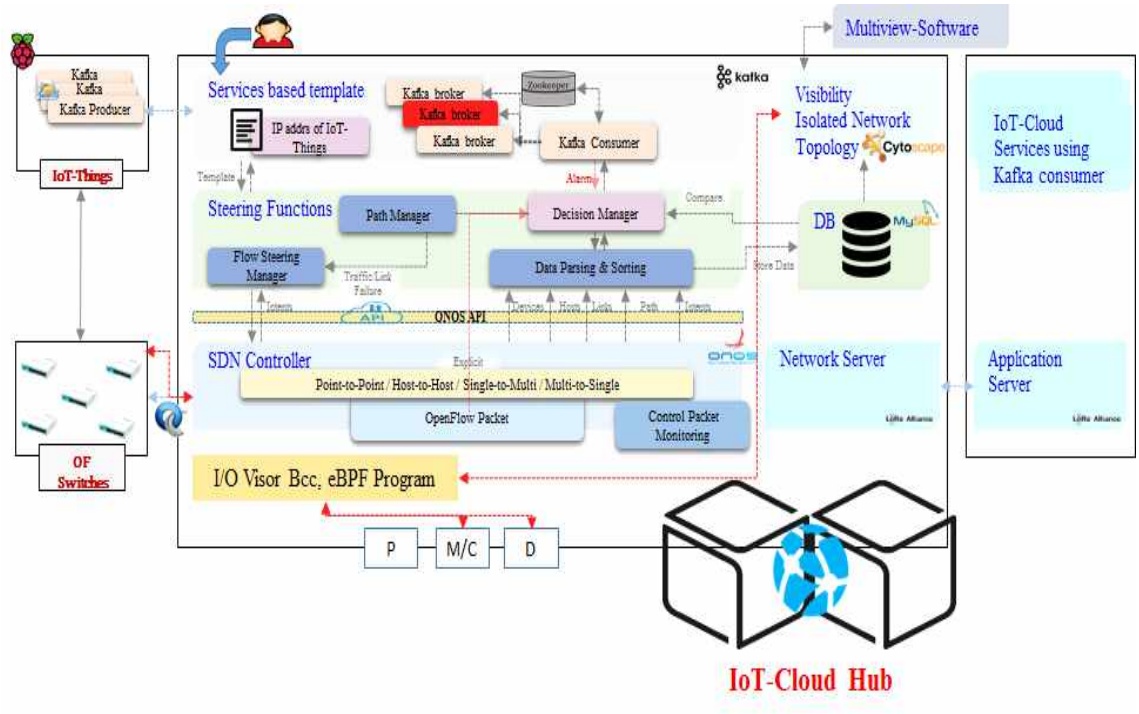
<그림 10: SmartX-mini Playground Software>

o <그림 10>은 SmartX-mini Playground에서 해당 서비스를 지원하기 위하여 μBox, IoT Actuator, μCloud Box에 설치된 소프트웨어 구성을 나타낸다. μBox들은 Apache Flume을 통해 주기적으로 μBox에서 동작하는 SNMP 데몬으로부터 필요한 자원 상태 정보를 수집한다. 수집된 정보는 종류에 따라 Kafka Broker 내의 topic에 대응되므로, 이를 Kafka producer API를 활용해 IoT actuator에 위치한 Kafka broker로 전달한다[11]. 각각의 μBox로부터 수집하는 자원 상태 정보는 Raspberry Pi2의 CPU, Memory, I/O 등에 대한 16개의 항목을 포함한다. 이 때, 각 broker들은 각각의 topic에 대해 1개 이상의 partition을 갖고 있으며, 손실을 방지하기 위하여 각 partition은 2개의 사본이 존재한다. 각 partition에 저장된 정보들은 파일 시스템 내에 저장되므로 일시적인 장애로 인해 발생한 지연에 대응 가능하다.

4. IoT-Cloud Hub 소프트웨어 구성

o 4절은 IoT-Cloud Hub의 소프트웨어 구성에 대한 설명과 동작 방식에 대해 기술한다. 4.1절은 전체적인 IoT-Cloud Hub의 디자인에 대해 설명하며 4.2절은 SDN 부분에 대한 기능 및 구현 동작에 대해, 4.3절은 Io Visor 중심의 기능 및 구현 동작에 대해 기술한다.

4.1. IoT-Cloud Hub 설계



<그림 11: IoT-Cloud Hub 소프트웨어 구조>

o <그림 11>은 IoT-Cloud Hub의 소프트웨어 구조를 나타낸다. IoT-Cloud Hub의 대표적인 역할로 전체적인 대표 기능으로는 SDN 기반의 WiFi/LoRa 이기종 IoT 네트워크 데이터 경로 제어 및 IoT 디바이스, SDN-enabled 스위치 관리, IO Visor 기반의 패킷 트레이싱 프로그램, 마지막으로 기능들에 대한 Visibility를 제공해야한다. IoT-Cloud Hub의 소프트웨어 기능들에 대해 상세히 설명하기 앞서, IoT-Cloud Hub가 동작하는 일련의 시나리오는 서비스 개발자가 템플릿을 통해 사용하고자 하는 물리 박스에 대해 명세를 한 뒤, IoT-Cloud Hub는 그 템플릿을 기반으로 토폴로지에 해당하는 물리 박스가 있는지 우선 검사한 후, 물리 박스가 존재하면 서비스를 지원하기 위해 종단간 데이터 경로를 설정한다. 이를 위해 IoT-Cloud Hub는 산하에 관리되고 있는 IoT 네트워크의 전반적인 물리 박스들을 관리할 수 있어야 한다. IoT 네트워크의 토폴로지 뿐만 아니라, 연결되어 있는 물리 컴퓨팅 자원 및 IoT 디바이스에 대해서도 관리를 지원한다. 이후 서비스 별 종단간 데이터 경로는 SDN에 기반하여 플로우 룰을 내림으로써 구분한다. 이후, 과도한 트래픽 혹은 물리적 링크 전달과 같은 상황에서도 원활히 서비스를 지원하기 위해 주기적으로 검사하여 데이터 경로의 신뢰성을 높인다. 또한 IO Visor를 활용해 패킷 수준의 정밀적인 트레이싱을 통해 패킷 수준의 Visibility를 제공한다.

o K-Cluster와 SmartX-mini Playground가 결합된 테스트베드에서 상기 설명한 SmartX IoT-Cloud 서비스인 자원 모니터링 서비스를 실행시키고자 할 때, 서비스 개발자 입장에서 고려할 것은 사용할 물리적 자원에 대한 명세이다. 자원 모니터링 서비스를 포함한 모든 SmartX IoT-Cloud 서비스는 기본적으로 Kafka Messaging 시스템을 사용하기에 Kafka Messaging 시스템과 SDN을 유기적으로 연동해 전체적인 IoT-Cloud 서비스 별 데이터 경로를 시스템 수준에서 서비스 수준까지 분류할 수 있도록 한다. 템플릿은 JSON 형식으로 작성되며 서비스 단위로 구성된다. <그림 12>와 같이 Service 객체를 배열로 가지며, 내부 요소로 version, name, zookeeper, broker, consumer, producer를 가진다. version은 상기 템플릿이 적용 가능한 버전을 의미하며 현재는 초기 프로토타입으로 0.1만을 인식한다. name은 SmartX IoT-Cloud 서비스를 지칭하며, zookeeper, broker, consumer, producer는 Kafka Messaging 시스템을 사용할 때 쓰이는 요소로써 각 사용하고자 하는 머신의 IP 정보와 포트 번호를 포함한다.

```
{
  "Service": [{
    "Version": "0.1",
    "Name": "resource_monitor",
    "Zookeeper": [{"IP":"192.168.88.92","Port":"2181"}],
    "Broker": [{"IP": "192.168.88.127","Port": "9092"},{"IP": "192.168.88.93","Port": "9092"}],
    "Consumer":[{"IP": "192.168.88.131"},{"IP": "192.168.88.133"}],
    "Producer":[{"IP": "192.168.88.130"}]
},
{
    "Version": "0.1",
    "Name": "smart_energy",
    "Zookeeper": [],
    "Broker": [],
    "Consumer":[],
    "Producer":[]
}
]}
```

<그림 12: 서비스 별 데이터 경로 구분을 위한 Kafka-SDN Template>

o SmartX IoT-Cloud 서비스 개발자는 상기 JSON 포맷의 템플릿을 IoT-Cloud Hub 내의 특정 디렉토리에 저장하면, IoT-Cloud Hub는 주기적으로 검사하여 템플릿이 존재할 경우 템플릿을 불러들여와 파싱하게 된다. IoT-Cloud Hub는 Java 기반으로 작성되어 있으며, <그림 13>은 템플릿을 읽어 파싱하는 코드를 나타낸다.

```
public static void JSON_Parsing(String template) throws ParseException {
    org.json.simple.parser.JSONParser jsonParser = new org.json.simple.parser.JSONParser();
    JSONObject jsonObject = (JSONObject) jsonParser.parse(template);
    JSONArray InfoArray = (JSONArray) jsonObject.get("Service");
    /*
    org.json.simple.parser.JSONParser jsonParser = new org.json.simple.parser.JSONParser();
    JSONObject jsonObject = (JSONObject) jsonParser.parse(buffer);
    JSONArray InfoArray = (JSONArray) jsonObject.get("Devices");
    */

    ResourcePool resource = ResourcePool.getInstance();
    ResourcePool.template_INFO_list[] tlist = resource.getTemplate_INFO_lists();

   for (int i=0;i<InfoArray.size();i++){
        JSONObject Object = (JSONObject) InfoArray.get(i);

        System.out.println("Version: "+Object.get("Version"));
        System.out.println("Name: "+Object.get("Name"));
        System.out.println("Zookeeper: "+Object.get("Zookeeper"));
        System.out.println("Broker: "+Object.get("Broker"));
        System.out.println("Consumer: "+Object.get("Consumer"));
        System.out.println("Producer: "+Object.get("Producer"));

       tlist[i].version = Object.get("Version").toString();
       tlist[i].service = Object.get("Name").toString();
```

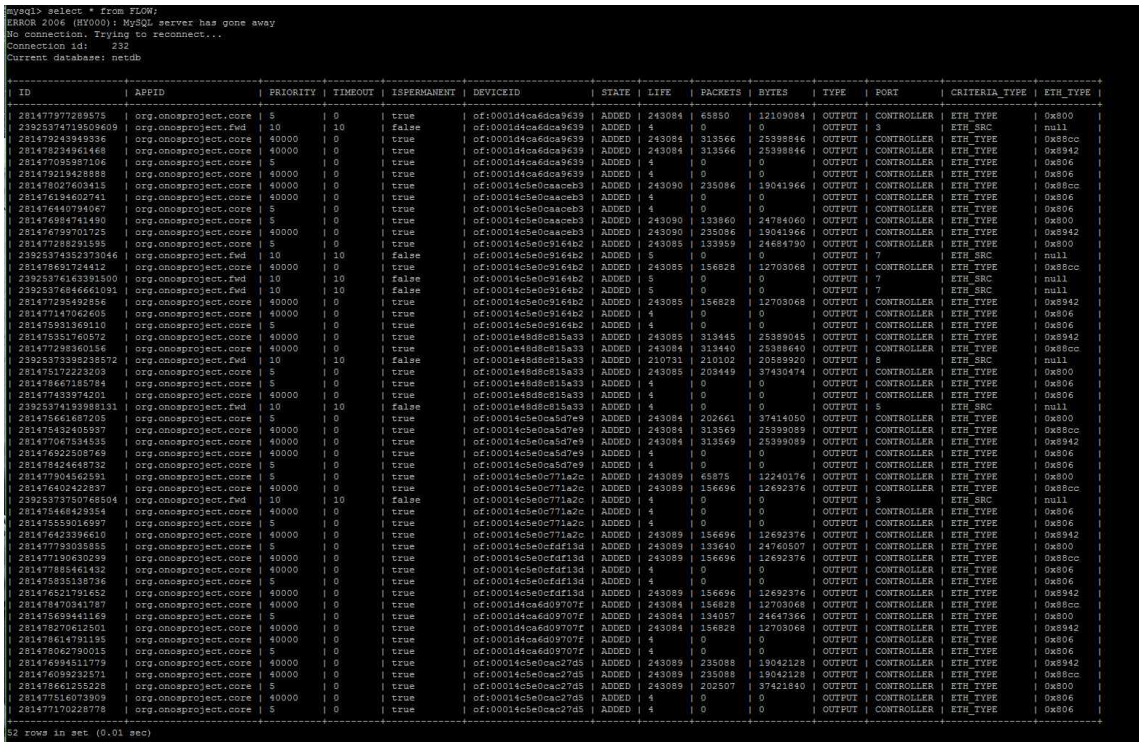
<그림 13: 자바 기반 템플릿 파싱 코드>

o IoT-Cloud Hub는 또한 산하 IoT 네트워크를 관리하기 위해 네트워크 장비, 토폴로지 그리고 IoT 기기들을 관제해야 한다. 이를 위해 ONOS SDN 컨트롤러의 도움을 받게 되는데, REST API를 통해 해당 정보들을 MySQL 데이터베이스에 저장하게 된다. 관리를 위해 ONOS 컨트롤러로 호출하는 REST API는 Device, Host, Intent, Flow, Path, Link로 총 6번의 API 호출을 통해 전달 받은 데이터를 저장한다. ONOS SDN의 REST API는 JSON 포맷으로 전달받아 템플릿과 마찬가지로 파싱한 뒤, MySQL 데이터베이스에 저장한다.

```
public static ResourcePool.flow_Info_list[] GET_ONOS_INFO_Flow(ResourcePool.flow_Info_list[] list) throws IOException, ParseException{
    String USERNAME= "karaf";
    String PASSWORD = "karaf";
    String DEVICE_API_URL= "http://"+Controller_IP+"/onos/v1/flows";
    URL onos = null;

    String buffer = URL_REQUEST(USERNAME,PASSWORD,DEVICE_API_URL,onos);
    org.json.simple.parser.JSONParser jsonParser = new org.json.simple.parser.JSONParser();
    JSONObject jsonObject = (JSONObject) jsonParser.parse(buffer);
    JSONArray InfoArray = (JSONArray) jsonObject.get("flows");

    for (int i=0;i<InfoArray.size();i++){
        JSONObject Object = (JSONObject) InfoArray.get(i);
        list[i].id = Object.get("id").toString();
        list[i].appId = Object.get("appId").toString();
        list[i].priority = Object.get("priority").toString();
        list[i].timeout = Object.get("timeout").toString();
        list[i].isPermanent = Object.get("isPermanent").toString();
        list[i].deviceId = Object.get("deviceId").toString();
        list[i].state = Object.get("state").toString();
        list[i].life = Object.get("life").toString();
        list[i].packets = Object.get("packets").toString();
        list[i].bytes = Object.get("bytes").toString();

        JSONObject Object2 = (JSONObject) Object.get("treatment");
        JSONArray InfoArray2 = (JSONArray) Object2.get("instructions");
        for (int j=0;j<InfoArray2.size();j++){
            JSONObject Object3 = (JSONObject) InfoArray2.get(j);
            list[i].type = Object3.get("type").toString();
            list[i].port = Object3.get("port").toString();
        }
        Object2 = (JSONObject) Object.get("selector");
        InfoArray2 = (JSONArray) Object2.get("criteria");
        for (int j=0;j<InfoArray2.size();j++){
            JSONObject Object3 = (JSONObject) InfoArray2.get(j);
            if (Object3.get("ethType")!=null)
                list[i].ethType = Object3.get("ethType").toString();
            list[i].ctype = Object3.get("type").toString();
        }
    }
    return list;
}
```

<그림 14: ONOS SDN 컨트롤러 REST API 호출 및 파싱 코드>

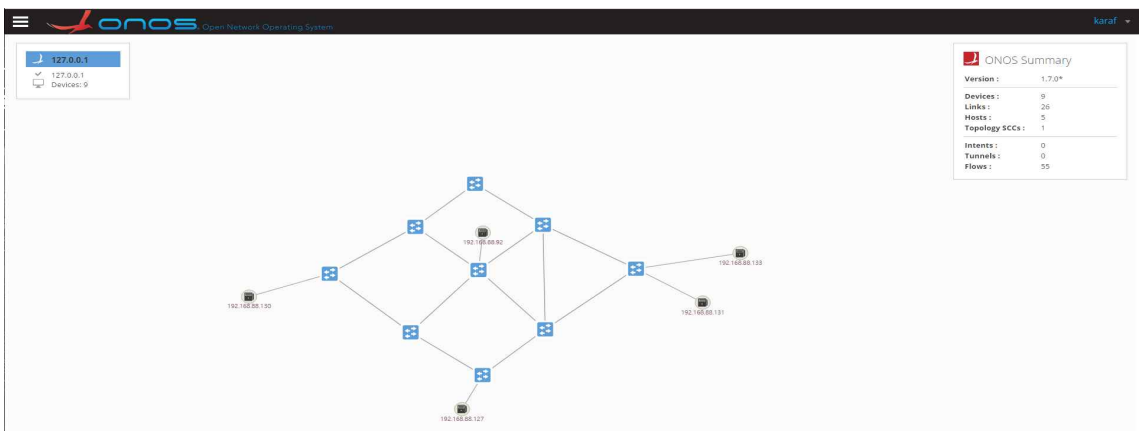
o <그림 14>는 ONOS SDN 컨트롤러부터 Flow 정보를 얻기 위해 REST API를 호출하고, 전달 받은 JSON 포맷의 데이터를 파싱하는 코드이며, <그림 15>는 MySQL 데이터베이스에 저장된 FLOW 테이블을 나타낸다. 데이터베이스에는 총 6개의 FLOW, LINK, SWITCH, HOST, INTENT, SWITCH_PORT 테이블이 존재한다.



```
mysql> select * from FLOW;
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    232
Current database: netdb

+------------------+----------------------+----------+---------+--------------+--------------------+---------+--------+---------+---------+--------+------------+---------------+----------+
| ID               | APPID                | PRIORITY | TIMEOUT | ISPERMANENT  | DEVICEID           | STATE   | LIFE   | PACKETS | BYTES   | TYPE   | PORT       | CRITERIA_TYPE | ETH_TYPE |
+------------------+----------------------+----------+---------+--------------+--------------------+---------+--------+---------+---------+--------+------------+---------------+----------+
| 281477977289575  | org.onosproject.core | 5        | 0       | true         | of:0001d4ca6dca9639 | ADDED | 243084 | 65850   | 12109084 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x800    |
| 2392537471950960 | org.onosproject.fwd  | 10       | 10      | false        | of:0001d4ca6dca9639 | ADDED | 4      | 0       | 0       | OUTPUT | 3          | ETH_SRC       | null     |
| 281479243949336  | org.onosproject.core | 40000    | 0       | true         | of:0001d4ca6dca9639 | ADDED | 243084 | 313566  | 25398846 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x88cc   |
| 281478234961468  | org.onosproject.core | 40000    | 0       | true         | of:0001d4ca6dca9639 | ADDED | 243084 | 313566  | 25398846 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x8942   |
| 281477095987106  | org.onosproject.core | 5        | 0       | true         | of:0001d4ca6dca9639 | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281479219428888  | org.onosproject.core | 40000    | 0       | true         | of:0001d4ca6dca9639 | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281478027603415  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0caaceb3 | ADDED | 243090 | 235086  | 19041966 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x88cc   |
| 281476194602741  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0caaceb3 | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281476440794067  | org.onosproject.core | 5        | 0       | true         | of:00014c5e0caaceb3 | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281476894741490  | org.onosproject.core | 5        | 0       | true         | of:00014c5e0caaceb3 | ADDED | 243090 | 133860  | 24784060 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x800    |
| 281476799701725  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0caaceb3 | ADDED | 243090 | 235086  | 19041966 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x8942   |
| 281477288291595  | org.onosproject.core | 5        | 0       | true         | of:00014c5e0c9164b2 | ADDED | 243085 | 133959  | 24684790 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x800    |
| 2392537435237304 | org.onosproject.fwd  | 10       | 10      | false        | of:00014c5e0c9164b2 | ADDED | 5      | 0       | 0       | OUTPUT | 7          | ETH_SRC       | null     |
| 281478691724412  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0c9164b2 | ADDED | 243085 | 156828  | 12703068 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x88cc   |
| 2392537616339150 | org.onosproject.fwd  | 10       | 10      | false        | of:00014c5e0c9164b2 | ADDED | 5      | 0       | 0       | OUTPUT | 7          | ETH_SRC       | null     |
| 2392537684661091 | org.onosproject.fwd  | 10       | 10      | false        | of:00014c5e0c9164b2 | ADDED | 5      | 0       | 0       | OUTPUT | 7          | ETH_SRC       | null     |
| 281477295492856  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0c9164b2 | ADDED | 243085 | 156828  | 12703068 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x8942   |
| 281477147062605  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0c9164b2 | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281475931369110  | org.onosproject.core | 5        | 0       | true         | of:00014c5e0c9164b2 | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281475351760572  | org.onosproject.core | 40000    | 0       | true         | of:0001e48d8c815a33 | ADDED | 243085 | 313445  | 25389045 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x8942   |
| 281477298360156  | org.onosproject.core | 40000    | 0       | true         | of:0001e48d8c815a33 | ADDED | 243084 | 313440  | 25388640 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x88cc   |
| 2392537398238572 | org.onosproject.fwd  | 10       | 10      | false        | of:0001e48d8c815a33 | ADDED | 210731 | 210102  | 20589920 | OUTPUT | 8          | ETH_SRC       | null     |
| 281475172223203  | org.onosproject.core | 5        | 0       | true         | of:0001e48d8c815a33 | ADDED | 243085 | 203449  | 37430474 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x800    |
| 281478667185784  | org.onosproject.core | 5        | 0       | true         | of:0001e48d8c815a33 | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281477433974201  | org.onosproject.core | 40000    | 0       | true         | of:0001e48d8c815a33 | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 2392537419398813 | org.onosproject.fwd  | 10       | 10      | false        | of:0001e48d8c815a33 | ADDED | 4      | 0       | 0       | OUTPUT | 5          | ETH_SRC       | null     |
| 281475661687205  | org.onosproject.core | 5        | 0       | true         | of:00014c5e0ca5d7e9 | ADDED | 243084 | 202661  | 37414050 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x800    |
| 281475432405937  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0ca5d7e9 | ADDED | 243084 | 313569  | 25399089 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x88cc   |
| 281477067534535  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0ca5d7e9 | ADDED | 243084 | 313569  | 25399089 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x8942   |
| 281476922508769  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0ca5d7e9 | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281478424648732  | org.onosproject.core | 5        | 0       | true         | of:00014c5e0ca5d7e9 | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281477904562591  | org.onosproject.core | 5        | 0       | true         | of:00014c5e0c771a2c | ADDED | 243089 | 65875   | 12240176 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x800    |
| 281476402422837  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0c771a2c | ADDED | 243089 | 156696  | 12692376 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x88cc   |
| 2392537375076850 | org.onosproject.fwd  | 10       | 10      | false        | of:00014c5e0c771a2c | ADDED | 4      | 0       | 0       | OUTPUT | 3          | ETH_SRC       | null     |
| 281475468429354  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0c771a2c | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281475559016997  | org.onosproject.core | 5        | 0       | true         | of:00014c5e0c771a2c | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281476423396610  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0c771a2c | ADDED | 243089 | 156696  | 12692376 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x8942   |
| 281477793035855  | org.onosproject.core | 5        | 0       | true         | of:00014c5e0cfdf13d | ADDED | 243089 | 133640  | 24760507 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x800    |
| 281477190630299  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0cfdf13d | ADDED | 243089 | 156696  | 12692376 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x88cc   |
| 281477885461432  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0cfdf13d | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281475835138736  | org.onosproject.core | 5        | 0       | true         | of:00014c5e0cfdf13d | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281476521791652  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0cfdf13d | ADDED | 243089 | 156696  | 12692376 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x8942   |
| 281478470341787  | org.onosproject.core | 40000    | 0       | true         | of:0001d4ca6d09707f | ADDED | 243084 | 156828  | 12703068 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x88cc   |
| 281475699441169  | org.onosproject.core | 5        | 0       | true         | of:0001d4ca6d09707f | ADDED | 243084 | 134057  | 24647366 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x800    |
| 281478270612501  | org.onosproject.core | 40000    | 0       | true         | of:0001d4ca6d09707f | ADDED | 243084 | 156828  | 12703068 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x8942   |
| 281478614791185  | org.onosproject.core | 40000    | 0       | true         | of:0001d4ca6d09707f | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281478062790015  | org.onosproject.core | 5        | 0       | true         | of:0001d4ca6d09707f | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281476994511779  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0cac27d5 | ADDED | 243089 | 235088  | 19042128 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x8942   |
| 281476099232571  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0cac27d5 | ADDED | 243089 | 235088  | 19042128 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x88cc   |
| 281478661255228  | org.onosproject.core | 5        | 0       | true         | of:00014c5e0cac27d5 | ADDED | 243089 | 202507  | 37421840 | OUTPUT | CONTROLLER | ETH_TYPE      | 0x800    |
| 281477516073909  | org.onosproject.core | 40000    | 0       | true         | of:00014c5e0cac27d5 | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
| 281477170228778  | org.onosproject.core | 5        | 0       | true         | of:00014c5e0cac27d5 | ADDED | 4      | 0       | 0       | OUTPUT | CONTROLLER | ETH_TYPE      | 0x806    |
+------------------+----------------------+----------+---------+--------------+--------------------+---------+--------+---------+---------+--------+------------+---------------+----------+
52 rows in set (0.01 sec)
```

<그림 15: MySQL 데이터베이스 FLOW 테이블>

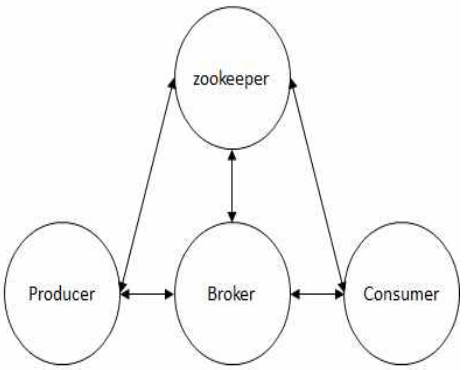
o IoT-Cloud Hub는 파싱하여 저장된 템플릿 정보와 MySQL 데이터베이스에 저장된 정보를 기반으로 템플릿에 명시된 IoT 기기가 실제로 존재하는지 비교한다. 명시된 IoT 기기가 있을 경우, IoT-Cloud Hub는 해당 IoT 기기들을 대상으로 Path를 만들게 된다.



<그림 16: ONOS SDN 토폴로지>

o <그림 16>은 <그림12>의 템플릿에 명시된 IoT 기기들이 모두 존재하는 상황을 ONOS SDN 토폴로지로 나타낸 경우이다. Path는 Host-to-Host로 종단 간 Source와 Destination의 정보를 포함해 ONOS SDN 컨트롤러한테 REST API를 호출하면 가능한 Path 목록을 반환한다. <그림 17>은 <그림 12> 템플릿에 명시된 것에 따라 SmartX IoT-Cloud 서비스인 Resource Monitoring 서비스에 대해서 Kafka 요소인 Producer, Broker, Zookeeper, Consumer에 대해 Path를 만들어낸 것을 나타낸다. SmartX IoT-Cloud 서비스는 <그림 17>의 오른쪽 다이어그램과 같이 각 Kafka 요소들 간에 연결성이 보장되기만 하면 원활한 서비스 진행이 가능하다. 따라서 Producer <-> Zookeeper, Zookeeper <-> Broker, Broker <-> Consumer, Zookeeper <-> Consumer, Consumer <-> Broker 간 연결성을 보장해야 하며 이에 따라 Path는 <그림 17>의 왼쪽과 같이 나타내게 된다.


```
++++++++++++Available Paths for Service resource_monitor ++++++++++
Producer <-> Broker
{"port":"0","host":"B8:27:EB:E1:C6:D3\/None"}%{"port":"5","device":"of:0001d4ca6d09707f"}
{"port":"3","device":"of:0001d4ca6d09707f"}%{"port":"1","device":"of:00014c5e0caaceb3"}
{"port":"2","device":"of:00014c5e0caaceb3"}%{"port":"1","device":"of:00014c5e0c771a2c"}
{"port":"3","device":"of:00014c5e0c771a2c"}%{"port":"0","host":"B8:AE:ED:79:CA:C9\/None"}
Producer <-> Broker
{"port":"0","host":"B8:27:EB:E1:C6:D3\/None"}%{"port":"5","device":"of:0001d4ca6d09707f"}
{"port":"3","device":"of:0001d4ca6d09707f"}%{"port":"1","device":"of:00014c5e0caaceb3"}
{"port":"3","device":"of:00014c5e0caaceb3"}%{"port":"8","device":"of:0001e48d8c815a33"}
{"port":"5","device":"of:0001e48d8c815a33"}%{"port":"0","host":"B8:AE:ED:76:80:C0\/None"}
Zookeeper <-> Consumer
{"port":"0","host":"0C:C4:7A:CA:3B:F3\/None"}%{"port":"7","device":"of:00014c5e0c9164b2"}
{"port":"1","device":"of:00014c5e0c9164b2"}%{"port":"7","device":"of:00014c5e0ca5d7e9"}
{"port":"1","device":"of:00014c5e0ca5d7e9"}%{"port":"4","device":"of:0001e48d8c815a33"}
{"port":"5","device":"of:0001e48d8c815a33"}%{"port":"0","host":"B8:AE:ED:76:80:C0\/None"}
Broker <-> Consumer
{"port":"0","host":"B8:AE:ED:79:CA:C9\/None"}%{"port":"3","device":"of:00014c5e0c771a2c"}
{"port":"2","device":"of:00014c5e0c771a2c"}%{"port":"3","device":"of:0001d4ca6dca9639"}
{"port":"6","device":"of:0001d4ca6dca9639"}%{"port":"17","device":"of:00014c5e0c9164b2"}
{"port":"7","device":"of:00014c5e0c9164b2"}%{"port":"0","host":"0C:C4:7A:CA:3B:F3\/None"}
Broker <-> Zookeeper
{"port":"0","host":"B8:AE:ED:79:CA:C9\/None"}%{"port":"3","device":"of:00014c5e0c771a2c"}
{"port":"2","device":"of:00014c5e0c771a2c"}%{"port":"3","device":"of:0001d4ca6dca9639"}
{"port":"4","device":"of:0001d4ca6dca9639"}%{"port":"3","device":"of:0001e48d8c815a33"}
{"port":"5","device":"of:0001e48d8c815a33"}%{"port":"0","host":"B8:AE:ED:76:80:C0\/None"}
```

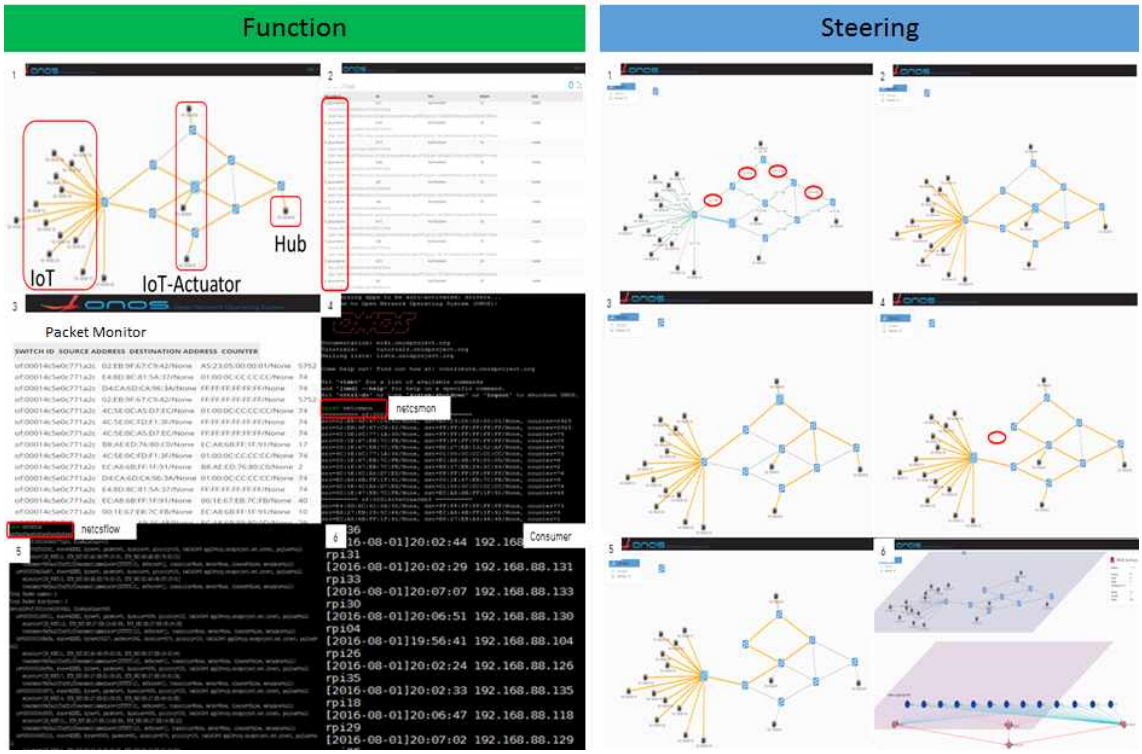



<그림 17: SmartX IoT-Cloud 서비스를 지원하기 위한 Path 목록>

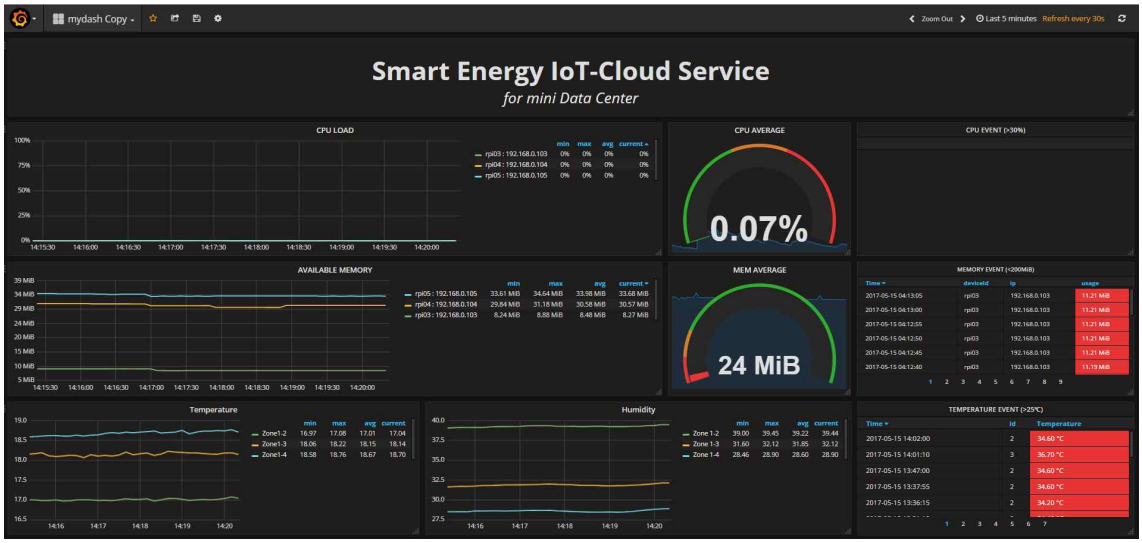
o 생성 가능한 Path 목록이 나오면 Path에 대해 플로우 룰을 내려줌으로써, 원활한 서비스 진행이 이루어지도록 한다. 플로우 룰은 ONOS의 Intent 프레임워크를 이용하여 내릴 수 있다. Intent는 Host-to-Host Intent, Multiple-to-Single Intent, Single-to-Multiple Intent, Point-to-Point Intent를 Path에 맞게 적용하며 같은 우선순위로 설정한다.

o IoT-Cloud Hub는 SmartX IoT-Cloud 서비스를 원활하게 동작하도록 지원하기 위해 서비스 별 데이터 경로를 자동으로 생성한다. 또한 주기적으로 검사하여 물리적 단절이 발생했을 경우, Path 리스트 항목에서 해당 링크를 검출 하여 해당 Path로 지나가는 플로우 룰을 재조정한다. 재조정은 Path에 대한 Intent를 지우고, 우회 경로에 대해 새로운 Intent를 설치하여 서비스를 지원한다. 또한 특정 링크에 대해 서비스 별 플로우에 대한 트래픽이 미리 정한 임계치를 넘을 경우, 사용

가능한 다른 Path로 데이터 경로를 바꿔주는 역할을 한다. 트래픽 임계치는 현재 IoT-Cloud Hub의 검증을 위해 랜덤 값을 넣어 실증을 했고, 현재는 서비스를 진행하는데 필요한 평균 트래픽을 계산 후, 대체할 예정이다. <그림 18>은 상기 설명한 기능들에 대해 나타낸다. 오른쪽의 Steering 쪽은 자동적으로 Intent가 설치된 모습, Link Failure 발생해 대체 경로로 Intent가 바뀐 경우를 보여준다. 오른쪽의 Function은 Packet Monitor로 플로우에 대해 Source, Destination 주소를 나타낸다.



<그림 18: ONOS를 활용한 SmartX IoT-Cloud 서비스 Dynamic Flow Steering>



<그림 19: SmartX IoT-Cloud 서비스 리소스 모니터링 결과>

o <그림 19>는 IoT-Cloud Hub의 성능을 검증하기 위해 사용한 SmartX IoT-Cloud 서비스인 자원 모니터링 서비스의 UI 화면을 나타낸다. 오픈 소스 시각화 툴인 Grafana를 활용한 것으로 그림 19를 자세히 보면 물리적 링크 단절과 같은 경우에도 IoT-Cloud Hub의 네트워크 관제로 인해 서비스가 영향을 받지 않고 정상적으로 데이터 값이 들어오는 것을 확인 할 수 있다.

4.2. IO Visor: 패킷 수준의 가시성 제공

o 구현을 위한 환경을 구축하기 위해 IO Visor BCC 사용을 위해 권장되는 Ubuntu 16.04.2 LTS 운영체제와 Linux Kernel 4.5.1 버전을 활용한다. 구축된 환경에 IO Visor 설치를 위해서 사전에 작성한 스크립트 파일을 이용할 수 있다. 스크립트 파일은 하위 커널 버전의 업그레이드를 포함하고 있으며, 이를 활용하면 따로 커널 버전을 갱신하지 않아도 IO Visor 설치를 진행할 수 있다. 설치 스크립트 파일은 (https://github.com/TaekhoNam/iovisor/tree/master/install)에서 확인할 수 있다.

o ‘step1.sh’ 스크립트 파일은 IO Visor 설치에 앞서 호환 가능한 커널 버전으로 업그레이드 하기 위한 코드가 포함되어 있다. ‘step2.sh’ 스크립트 파일은 debian/ubuntu 운영체제 환경에서 IO Visor를 설치하기 위한 스크립트 코드를 포함하고 있다.

```
# 2016-10-26
# Kernel Configuration
CONFIG_BPF=y
CONFIG_BPF_SYSCALL=y
CONFIG_NET_CLS_BPF=m
CONFIG_NET_ACT_BPF=m
CONFIG_BPF_JIT=y
CONFIG_HAVE_BPF_JIT=y
CONFIG_BPF_EVENTS=y

# Ubuntu Trusty - Binary
VER=4.5.1-040501
PREFIX=http://kernel.ubuntu.com/~kernel-ppa/mainline/v4.5.1-wily/
REL=201604121331
wget ${PREFIX}/linux-headers-${VER}-generic_${VER}.${REL}_amd64.deb
wget ${PREFIX}/linux-headers-${VER}_${VER}.${REL}_all.deb
wget ${PREFIX}/linux-image-${VER}-generic_${VER}.${REL}_amd64.deb
sudo dpkg -i linux-*${VER}.${REL}*.deb
sudo reboot
```

<그림: 20 커널 업그레이드를 위한 스크립트 코드>

```
# Packages
# Ubuntu Trusty - Binary
# Signed Packages
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys D4284CDD
echo "deb https://repo.iovisor.org/apt trusty main" | sudo tee /etc/apt/sources.list.d/iovisor.list
sudo apt-get update
sudo apt-get install binutils bcc bcc-tools libbcc-examples python-bcc -y

# (Optional) Install pyroute2 for additional networking features
git clone https://github.com/svinota/pyroute2
cd pyroute2; sudo make install
cd ..


# Source
# Ubuntu - Source
# Install build dependencies
VER=trusty
echo "deb http://llvm.org/apt/$VER/ llvm-toolchain-$VER-3.7 main
deb-src http://llvm.org/apt/$VER/ llvm-toolchain-$VER-3.7 main" | \
  sudo tee /etc/apt/sources.list.d/llvm.list
wget -O - http://llvm.org/apt/llvm-snapshot.gpg.key | sudo apt-key add -
sudo apt-get update

sudo apt-get -y install bison build-essential cmake flex git libedit-dev \
  libllvm3.7 llvm-3.7-dev libclang-3.7-dev python zlib1g-dev libelf-dev -y

sudo apt-get -y install luajit luajit-5.1-dev

# Install and complie BCC
git clone https://github.com/iovisor/bcc.git
mkdir bcc/build; cd bcc/build
cmake .. -DCMAKE_INSTALL_PREFIX=/usr
make
sudo make install
```

<그림 21: debian/ubuntu 계열의 운영체제에 IO Visor 설치를 위한 스크립트>

o 정상적으로 IO Visor 설치를 진행하기 위해서는 제시된 일련의 과정을 수행할 것을 권장한다. Github 저장소를 호스트로 다운로드하고, 커널 버전을 업그레이드하기 위해서 'step1.sh' 스크립트 파일을 실행시킨다. 만약, 권한에 오류가 발생한

다면 chmod 명령어를 통해서 실행 가능하도록 권한을 설정해준다. 이 때, 설정 파일의 적용을 위해서 호스트가 자동으로 재부팅된다. 재부팅이 완료된 후에 'step2.sh' 파일을 실행시키면 자동으로 IO Visor 설치가 완료된다. 설치가 정상적으로 진행되었다면, 'bcc' 디렉터리가 새로 생기며, 예제 코드를 통해서 문제없이 동작하는지 확인할 수 있다.

o IO Visor 예제 코드가 정상적으로 동작한다면, 이후에는 'packet_tracing.py'를 활용해서 네트워크 인터페이스로 출입하는 http 형식의 패킷을 추적할 수 있다. Python으로 작성된 응용프로그램이 C 언어로 작성된 eBPF를 호출하고 패킷을 트레이싱한다. 트레이싱 된 정보는 터미널 창을 통해서 실시간으로 확인할 수 있다.

```
15  from __future__ import print_function
16  from bcc import BPF
17
18  import sys
19  import socket
20  import os
21  import argparse
22
23  # initialize BPF - load source code from http-parse-simple.c
24  bpf = BPF(src_file = "http-parse-simple.c",debug = 0)
```

<그림 22: BPF import 및 호출>

```
84  #TCP HEADER
85  #https://www.rfc-editor.org/rfc/rfc793.txt
86  #  12              13              14              15
87  #  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
88  # +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
89  # |  Data |           |U|A|P|R|S|F|                               |
90  # | Offset| Reserved  |R|C|S|S|Y|I|            Window             |
91  # |       |           |G|K|H|T|N|N|                               |
92  # +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
93  #
94  #Data Offset: This indicates where the data begins.
95  #The TCP header is an integral number of 32 bits long.
96  #value to multiply * 4 byte
97  #e.g. DataOffset = 5 ; TCP Header Length = 5 * 4 byte = 20 byte
98
99  #calculate tcp header length
100 tcp_header_length = packet_bytearray[ETH_HLEN + ip_header_length + 12]  #load Byte
101 tcp_header_length = tcp_header_length & 0xF0                            #mask bit 4..7
102 tcp_header_length = tcp_header_length >> 2                              #SHR 4 ; SHL 2 -> SHR 2
103
104 #calculate payload offset
105 payload_offset = ETH_HLEN + ip_header_length + tcp_header_length
106
107 #print first line of the HTTP GET/POST request
108 #line ends with 0xOD 0xOA (\r\n)
109 #(if we want to print all the header print until \r\n\r\n)
```

<그림 23: TCP 헤더구조로부터 패킷 정보 추적>

```
124 #save information about action eg. HTTP, GET ... (refer http-parse-simple.c file)
125 action = ""
126 for i in range (payload_offset-1,len(packet_bytearray)-1):
127   if (packet_bytearray[i]== 0x0A):
128     if (packet_bytearray[i-1] == 0x0D):
129       break
130   action += chr(packet_bytearray[i])
131
132 if (dstPort == "22"):
133       action = "SSH packet passing"
134
135
136
137 #print information including ipVer/srcAddr/dstAddr:port/action
138 print("%3s%20s%20s%9s%5s" % (str(int(ipversion, 2)), srcAddr, dstAddr, dstPort, "     : " + action))
139
140 #---------------------------------------------------------------------------
141
```

<그림 24: 트레이싱 된 데이터를 터미널 CLI 환경으로 출력>

o 관리자 입장에서 네트워크 모니터링을 위해서 패킷 수준의 트레이싱을 진행할 경우 CLI 환경에서 패킷 정보를 일일이 확인하는 데에는 어려움이 따른다. 이에 따라서, 오픈소스 schemeless 데이터베이스 시스템인 InfluxDB [14]를 활용해서 추적된 데이터를 저장한다. 그리고 오픈소스 시각화 도구인 Grafana [15]를 활용하여, InfluxDB에 저장된 패킷 데이터에 접근하여 시각화한다. 이를 통해서 네트워크 관리자는 시각화된 정보를 토대로 K-Cluster, IoT-Cloud Hub 내외로 출입하는 패킷에 대한 전반적인 상황을 파악할 수 있다.

o 기존 'packet_tracing.py'를 개선하여 InfluxDB와 연동을 진행한다. 연동된 버전의 소스코드는 'packet_tracing_with_InlfuxDB.py'에서 확인할 수 있다. 데이터를 Json 형태로 가공하여, InluxDB에 저장한다.

```
#InfluxDBClient-----------------------------------------------

if (srcAddr == "192.168.0.20" or dstAddr == "192.168.0.20"):
  addr = "192.168.0.20"
  host_name = "container#2"
  count_c1 = count_c1 + 1

json_body = [
  {
    "measurement": "packets2",
    "tags": {
      "host": host_name,
      "ipAddr": addr
    },
    "fields": {
       "value": count_c1
    }
  }
]
```

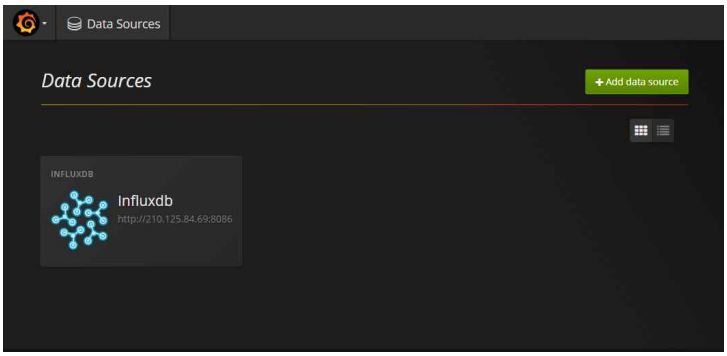
<그림 25: InfluxDB 저장을 위한 Json 가공 예시>

o Json 타입으로 가공한 데이터를 InfluxDB에 저장하기 위해서 InfluxDB에 연결을 진행한다. 연결된 이후에 데이터 저장을 위한 데이터베이스를 생성하고, 가공된 데이터를 저장한다. 저장된 데이터에 접근하거나, 정상적으로 저장되었는지 확인하기 위해서는 클라이언트 객체에 쿼리 연산을 통해서 수행할 수 있다.

```
163    client = InfluxDBClient('localhost', 8086, 'root', 'root', 'example')
164    client.create_database('example')
165    client.write_points(json_body)
166    result = client.query('select value from host1;')
```

<그림 26: InfluxDB 접근 및 데이터 저장, 조회 소스코드 예제>

o IO Visor를 통해서 트레이싱 된 패킷 데이터를 Json 형태로 가공하여 InfluxDB에 정상적으로 저장했다면, 최종적으로 이를 시각화할 작업만 남았다. 오픈소스 시각화 도구인 Grafana를 통해 InfluxDB에 접근할 수 있으며, 웹 기반의 화면에서 쿼리 연산의 결과를 차트, 도표 형식으로 볼 수 있다.



<그림 27: Grafana 내 InfluxDB 연동 선택 화면>

<그림 28: Grafana를 통한 패킷 트레이싱 데이터 시각화 예시>

5. 결론

o 본 기술문서를 통해 Edge-Cloud 모델에 대응하는 클러스터형 테스트베드인 K-Cluster를 중심으로 LoRa, Wi-Fi를 포함하는 이기종 통신 인터페이스에 대응 가능한 K-Cluster 기반의 IoT-Cloud Hub 요소에 대해 설계 및 구축 과정을 설명하였다. 그리고 설계에 따라 구축한 IoT-Cloud Hub 및 실증을 위한 테스트베드와 결과에 대해 상세히 기술하였다.

o IoT-Cloud Hub의 전체적인 큰 기능인 Data Collect, Data Validate, Control Data Path, Manage Things & Switches 중 SDN을 활용해 데이터 경로를 컨트롤하고 산하 IoT 네트워크 장비와 IoT 기기들에 대해 관리하는 기능들에 대해서만 기술하였다. Data Collect 및 Data Validate 부분은 추후 구현을 할 예정이며, 위 기술한 4가지 대표 기능들을 통해 궁극적으로 Multi-Access를 지원하는 IoT-Cloud Hub를 완성 & K-Cluster의 IoT와의 연결을 담당하는 교두보 역할을 더 완벽하게 하도록 보완할 계획이다.

o 본 기술문서에서 기술한 프로토타입의 IoT-Cloud Hub를 사용하면 SmartX IoT-Cloud 서비스에 대해서 SDN에 기반한 동적인 데이터 경로 설정을 통해 원활한 서비스 운용이 가능함을 확인할 수 있다.

References

[1] K.S. Yeo, M.C. Chain, T.C. W, Ng, and A.T. Do, "Internet of Things: Trends, challenges and applications," in *Proc. IEEE ISIS,* Mar. 2014.

[2] Yeonkeun Kim and Seungwon Shin, "Design Secure IoT (Internet of Things) Environments with SDN and NFV," *INFORMATION AND COMMUNICATIONS MAGZAINE ISSN*, pp. 27-33, vol.32, no. 7 Jul. 2015.

[3] LoRaWAN Specification, https://www.lora-alliance.org/portals/0/specs /LoRaWAN%20Specification%201R0.pdf

[4] ONF, https://www.opennetworking.org/about/onf-overview

[5] N. McKeown, et. al., "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOM Computer Communication Review*, pp. 69-74, Apr. 2008.

[6] ONOS, http://onosproject.org/

[7] Docker, https://www.docker.com/

[8] Kafka, http://kafka.apache.org/

[9] IO Visor Project, https://www.iovisor.org/.

[10] BCC (Berkeley Packet Filter Complier Collection), https://www.iovisor.org/technology/bcc/.

[11] eBPF (extended Berkeley Packet Filter), https://www.iovisor.org/technology/ebpf

[12] K-ONE 기술문서 #16 '멀티사이트 K-ONE Playground 활용을 위한 K-DevOps Tower의 구축 및 활용 방법

[13] JongWon Kim, et. al., "OF@TEIN: An OpenFlow-enabled SDN testbed over international SmartX Rack Sites," in *Proc. APAN-Networking Research Workshop*, Aug. 2013.

[14] InfluxDB, https://www.influxdata.com/.

[15] Grafana, https://grafana.com/.

K-ONE 기술 문서

- K-ONE 컨소시엄의 확인과 허가 없이 이 문서를 무단 수정하여 배포하는 것을 금지합니다.

- 이 문서의 기술적인 내용은 프로젝트의 진행과 함께 별도의 예고 없이 변경될 수 있습니다.

- 본 문서와 관련된 문의 사항은 아래의 정보를 참조하시길 바랍니다.
  (Homepage: http://opennetworking.kr/projects/k-one-collaboration-project/wiki, E-mail: k1@opennetworking.kr)

작성기관: K-ONE Consortium
작성년월: 2017/05