K-ONE Technical Document #25

Design of SmartX MultiView Visibility Software for SDN-enabled Multi-site Cloud

Document No. K-ONE #25

Version 1.0

Date 2018-02-26

Author(s) Usman, Muhammad

K-ONE

ONK
OpenNetworkingKorea

History

Version	Date	Author(s)	Contents
Draft - 0.1	2018. 01. 05	Usman, Muhammad	Chapter 1
0.2	2018. 01. 20	Usman, Muhammad	Chapter 2
0.3	2018. 02. 05	Usman, Muhammad	Chapter 3
0.4	2018. 02. 23	Usman, Muhammad	Chapter 4

본 문서는 2017년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No. 2015-0-00575, 글로벌 SDN/NFV 공개소프트웨어 핵심 모듈/기능 개발)

This work was supported by Institute for Information & communications Technology Promotion(IITP) grants funded by the Korea government(MSIT) (No. 2015-0-00575, Global SDN/NFV Open-Source Software Core Module/Function Development)

K-ONE

ONK
OpenNetworkingKorea

Summary

In this document, we present the preliminary design of SmartX MultiView Visibility Framework for SDN-enabled multi-site clouds. In Chapter 1, we provide an overview of OF@TEIN Playground, and playground visibility and visualization requirements. In Chapter 2, we briefly discuss about each stage of SmartX MultiView Visibility Framework. In Chapter 3, we provide initial-stage implementation of SmartX MultiView Visibility Software for enabling flow-layer visibility with associated visualizations in OF@TEIN Playground. Finally, we provide how to deploy and verify SmartX MultiView Software into SDN-enabled multi-site cloud playground for delivering flow-layer visibility and visualizations.

Contents

K-ONE 기술 문서 #25 Design of SmartX MultiView Visibility Software for SDN-enabled Multi-site Cloud

List of Figures

List of Tables

K-ONE #25. Design of SmartX MultiView Visibility Software for SDN-enabled Multi-site Cloud

1. Introduction

In this chapter, we will briefly discuss about emerging technologies, OF@TEIN Playground, and associated operational challenges.

Recently, Software-Defined Networking (SDN) has captivated the interest of both industry and research community since it proposes to decouple data-plane functionality from control-plane functionality. Data-plane functionality of packet forwarding is built into switching fabric, while the control-plane functionality of managing network devices is placed in logically-centralized software controller(s). This separation simplifies the network management tasks and minimizes the associated complexities of distributed configurations. Because of the cost-effectiveness of SDN approach, it enables researchers to perform various experiments, which were too costly or difficult with legacy network devices. Cloud computing is another emerging paradigm, which attracts the interest of its service providers and consumer communities with its pay-per-use service model since it removes the upfront cost of purchasing physical hardware and hides network connectivity complexities. That is, cloud computing delivers on-demand IT resources by dynamically scaling its provisioning power.

Lately, various research work has concentrated on establishing Future Internet testbeds to provide advanced experimental networking facilities for evaluating emerging software-defined technologies. Aligned with Future Internet testbed projects like GENI and FIRE, we launched OF@TEIN Playground. Mainly, serving as SDN-enabled multi-site clouds to facilitate academic miniaturized experiments. OF@TEIN Playground is composed of 10 distributed sites, spread over nine countries (i.e., Korea, Malaysia, Thailand, Indonesia, Philippines, Pakistan, Taiwan, Vietnam, and India) as depicted in Fig 1.

At OF@TEIN Playground, distributed 'SmartX Boxes' hosts the VM instances of OpenStack cloud and controls overlay virtual networks among multiple sites. Playground developer, requests OpenStack VM instances (i.e., managed as a separate tenant) and VLAN-isolated overlay virtual networking in SmartX Boxes. SmartX Boxes support multi-tenancy by hosting any number of dedicated virtual resources (e.g., virtual machines, virtual switches) for each tenant.

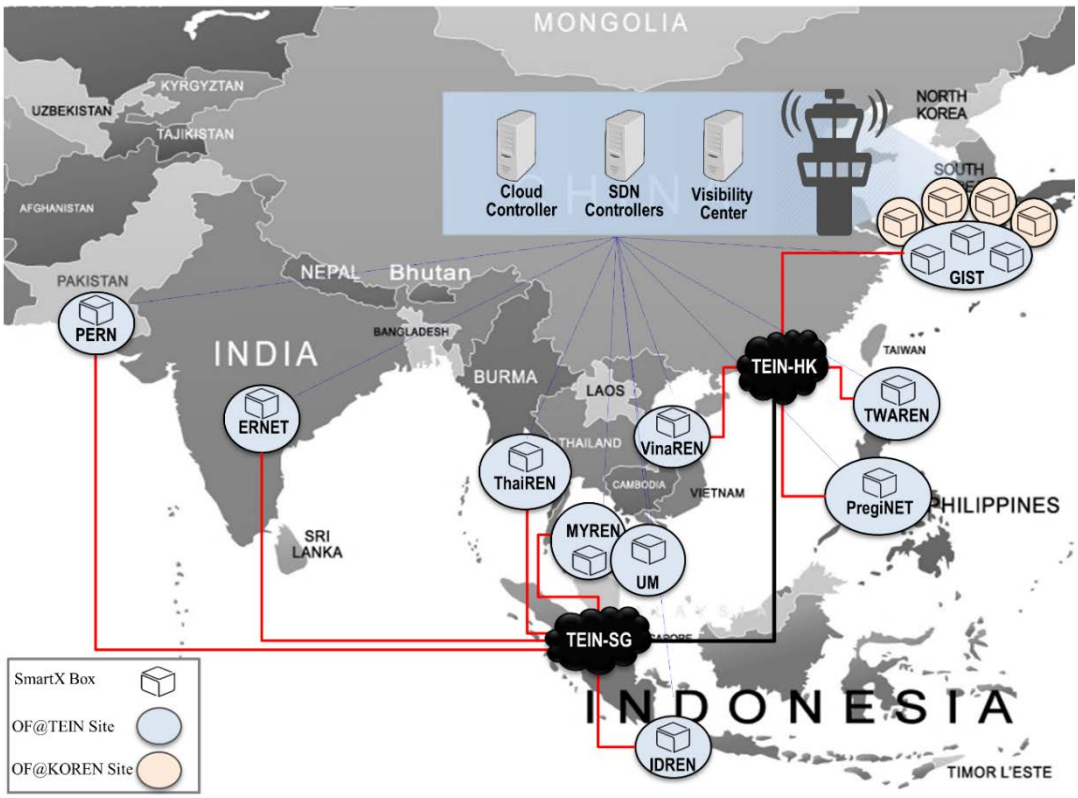

Figure 1: OF@TEIN playground as multi-site SDN-enabled Cloud.

Visibility measurement and associated visualization are essential for the operation monitoring of OF@TEIN Playground, so that operators can gain timely insights into the operational status of playground resources and related flows. Particularly, an effective flow monitoring is necessary to understand what is happening in the playground infrastructure.

Thus, we worked on a multi-layer visibility framework, denoted as 'SmartX Multi-View Visibility Framework (MVF)' and verify its real-world feasibility by targeting OF@TEIN Playground for multi-site SDN-enabled cloud testbed. Particularly, the given framework attempts to deliver an effective design for multi-layer visibility workflow in given testbed.

The rest of this document is organized as follows. In Chapter 2, we present the design of SmartX MultiView Visibility Framework, which is followed by initial-stage implementation details in Chapter 3. Finally, an initial-stage deployment and verification of SmartX MultiView Software at OF@TEIN Playground is given in Chapter 4.

2. Design of SmartX MultiView Visibility Framework

In this chapter, we will discuss the overall design of SmartX MultiView Visibility Framework by focusing on four main stages: Visibility Collection and Validation, Visibility DataLake and Analytics, Visibility Integration, and Visibility Staging and Visualization as shown in Fig. 2.

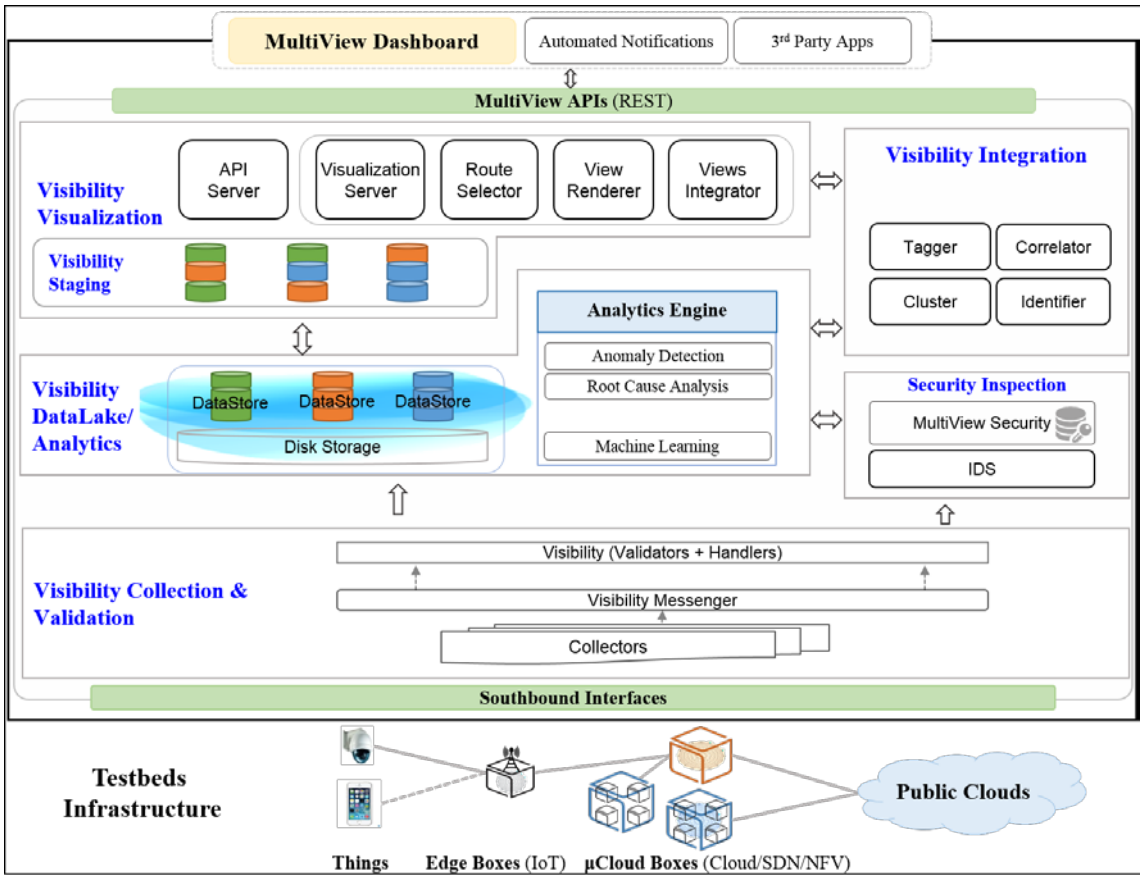

Figure 2: Design of SmartX MultiView Visibility Framework.

2.1. Visibility Collection and Validation

This stage consists of four main modules, including; 'Visibility Collectors', 'Visibility Messenger', 'Visibility Validators' and 'Visibility Handler'. Visibility Collector collects metrics from the distributed SmartX Boxes from one of the visibility layers. According to the playground visibility requirements, we divide collectors into two main categories: local and remote, which are differentiated from each other due to deployment location, data collection volume, and measurement interval. Remote collector should collect measured

metrics from SmartX Boxes locally, and send it to SmartX Visibility Center. On the contrary, local collectors should reside into SmartX Visibility Center for remote collection of metric data from the SmartX Box. For reliably moving collected data from distributed SmartX Boxes to SmartX Visibility Center, Visibility Messenger is designed. That is followed, by Visibility Validator, which verifies collected data contents. Ultimately, Visibility Consumer should apply initial formatting before data storage.

2.2. Visibility DataLake and Analytics

An efficient and fault-tolerant preservation of diverse multi-layer data is quite challenging because collected data possesses numerous types and data format. Considering these requirements, we opted DataLake approach to provide large-scale storage repository for preserving visibility data, either structured, semi-structured or unstructured with varying characteristics. During Visibility DataLake design, we considered two deployment models (i.e., centralized and distributed) for allowing fault-tolerance and flexibility during playground visibility processing. A centralized model is easy to manage but offers a single point of failure while a distributed model is more flexible, yet hard to manage. For enabling efficient data management and fast query processing, throughout framework design, we endorsed multiple 'NoSQL DataStores' in Visibility DataLake to match wide-range of playground visibility use cases. The analytics engine is foundational to perform playground visibility analysis for improving the testbed operations. Particularly, 'Visibility integration' stage extensively leverages analytics engine for delivering insights into how workloads, flows, and physical as well as virtual resources are performing across the entire playground. A single analytics engine deployable on a single host or cluster depending on the availability of resources should be chosen.

2.3. Visibility Integration

Visibility integration is a core stage to transform collected metrics data into meaningful information, leading towards automated detections of playground operational glitches. Visibility integration should cope with several challenges for timely analysis of the multi-layer metrics data because collection stage gathers numerous metrics to support individual visibility layer in detail, but timely processing of all the metrics is unrealistic. Therefore, visibility integration should filter the data for identifying highly dependent metrics with

appropriate data range required for integration. Usually, filtered data is still large and identical, which should be further reduced using data aggregation methods like average. For example, flow-layer visibility data collected over the certain time mostly contain a large number of measurements, but among those only limited number of measurements are unique and significant enough to represent the whole time range.

2.4. Visibility Staging and Visualization

Massive visibility data preservation offers a more accurate picture of the playground, but making it readily available to the users is unfeasible. To cater this challenge, framework design introduced Visibility Staging, a model for holding processed or recent data, resulting in less processing cost and faster data access.

This stage should fetch data from the Visibility DataLake, and process it for rendering interactive visualizations. To function accurately, visibility visualization introduces following modules: 'API Server', 'Visualization Server', 'Route Selector', 'View Renderer' and 'Views Integrator'. Firstly, Visualization Server should host and serve the visualization requests. Secondly, Route Selector should process views request and empower inter-module communications. Next, API Server that should provide data access from the Visibility DataLake. Finally, a View Renderer should translate data into visualizations in assistance by Views Integrator for third-party visualization tools integration.

3. Implementation of SmartX MultiView Visibility Software

In this chapter, we provide the prototype implementation details of framework for flow-layer visibility and interactive visualizations of OF@TEIN Playground under the title of SmartX MultiView Visibility Software.

3.1. Visibility Collection and Validation

We leveraged sFlow [1] and 'IO Visor Project' [2] for flow-layer visibility measurement from OF@TEIN playground. sFlow is an industry standard technology for monitoring high speed switched networks. It gives completed visibility into the use of networks for enabling performance optimization. sFlow is a sampling technology that meets the key requirements for a network traffic monitoring. sFlow sends measured data to a collector, named sFlow-RT [3]. The simplified workflow of sFlow with sFlow-RT is shown in Fig. 3.

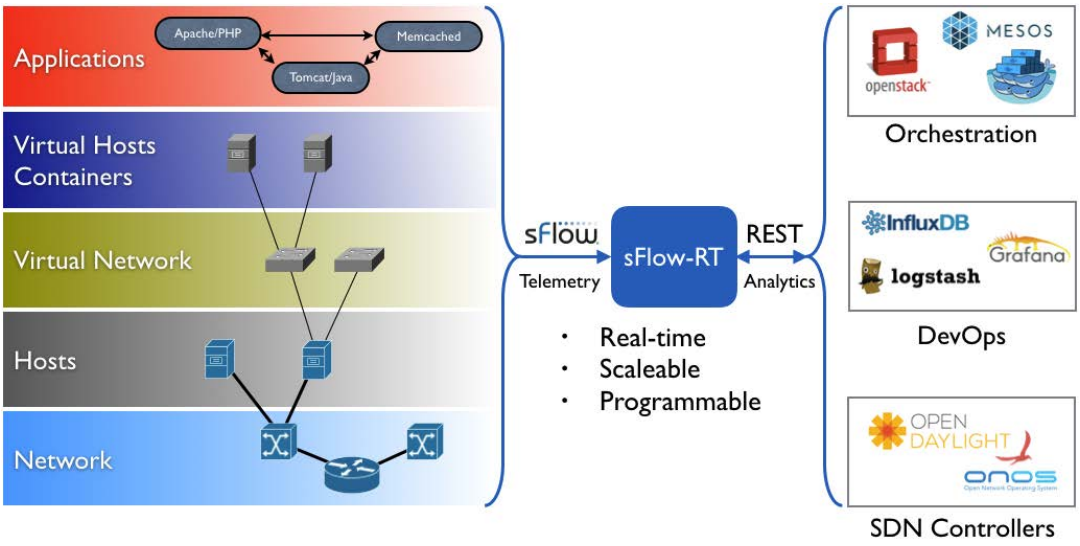

Figure 3: sFlow simplified workflow illustration.

To collect sampled network traffic, we have to enable sFlow agent in the specific virtual switch instance (e.g., operator bridge brcap) inside SmartX Box. The steps to enable sFlow in OVS are given below:

```
sudo ovs-vsctl -- --id=@sflow create sflow agent=${AGENT_IP}
target=₩"${COLLECTOR_IP}:${COLLECTOR_PORT}₩" header=${HEADER_BYTES}
sampling=${SAMPLING_N} polling=${POLLING_SECS} -- set bridge $BRIDGE sflow=@sflow
```

IO Visor is an open-source project to accelerate the innovation, development and sharing of in-kernel IO services for tracing, analytics, monitoring, security, and networking functions. We utilized IO Visor for flexible tracing of network packets at kernel-level. IO Visor Project architecture consists of single or multiple IO Visor Engines built on eBPF technology and development tools. The IO Visor Engine has a set of plugins that provides functionality in different areas such as networking, tracing, and security as shown in Fig. 4.

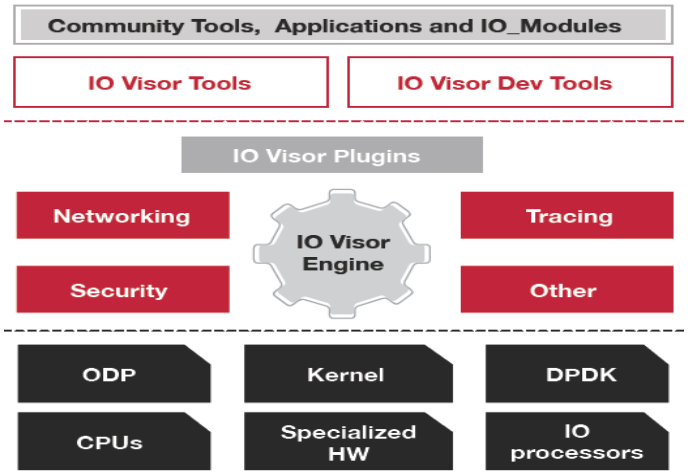

Figure 4: IO Visor Engine Plugins for providing functionality in different areas.

To collect network packets from SmartX Boxes at kernel-level, we need to fetch and install the latest release of IO Visor from GitHub (https://github.com/iovisor/bcc). The steps to install IO Visor on Ubuntu 16.04 are given below:

```
# Trusty and older
# 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 cmake3 flex git libedit-dev libllvm3.7 llvm-3.7-dev libclang-3.7-dev python zlib1g-dev libelf-dev
# Install and compile BCC
# git clone https://github.com/iovisor/bcc.git
```

mkdir bcc/build; cd bcc/build

cmake .. -DCMAKE_INSTALL_PREFIX=/usr

make

sudo make install

A snapshot of custom developed IO Visor tracing program is given below:

```
ISDATA: ;
        u32  tcp_header_length = 0;
        u32  ip_header_length = 0;
        u32  payload_offset = 0;
        u32  payload_length = 0;

        struct tcp_t *tcp = cursor_advance(cursor, sizeof(*tcp));

        /*
        calculate ip header length
        value to multiply * 4
        e.g. ip->hlen = 5 ; IP Header Length = 5 x 4 byte = 20 byte

        The minimum value for this field is 5, which indicates a length of 5 x 32 bits(4 bytes) = 20 bytes
        */
        ip_header_length = ip->hlen << 2:    //SHL 2 -> *4 multiply

        /*
        calculate tcp header length
        value to multiply *4
        e.g. tcp->offset = 5 ; TCP Header Length = 5 x 4 byte = 20 byte

        The minimum value for this field is 5, which indicates a length of 5 x 32 bits(4 bytes) = 20 bytes
        */
        tcp_header_length = tcp->offset << 2: //SHL 2 -> *4 multiply

        //calculate patload offset and length
        payload_offset = ETH_HLEN + ip_header_length + tcp_header_length;
        payload_length = ip->tlen - ip_header_length - tcp_header_length;

        if(payload_length < 7) {
                goto DROP;
        }

        /*
        load first 7 byte of payload into p (payload_array)
        direct access to skb not allowed
        load_byte(): read binary data from socket buffer(skb)
        */
        unsigned long p[7];
        int i = 0;
        int j = 0;
        for (i = payload_offset ; i < (payload_offset + 7) ; i++) {
                p[j] = load_byte(skb , i);
                j++;
        }

        goto KEEP;

//keep the packet and send it to userspace retruning -1
KEEP:
        return -1;

//drop the packet returning 0
DROP:
        return 0;
```

Figure 5: IO Visor Program to trace network packets from Linux Kernel.

Next, we should download and start the sFlow-RT collector in SmartX Visibility Center for collecting sampled traffic data as following:

wget http://www.inmon.com/products/sFlow-RT/sflow-rt.tar.gz

tar –xvzf sflow-rt.tar.gz

cd sflow-rt

./sflow-rt/start.sh

Collected (and to-be-validated) visibility data is moved to SmartX Visibility Center through 'Apache Kafka' [4] and placed in a queue. Kafka is an open-source messaging project developed by 'Apache Software Foundation' and written in Scala. Kafka provides a unified, high throughput, low-latency platform for handling real-time data feeds. Fig. 4 shows, overall workflow of Kafka Messenger:

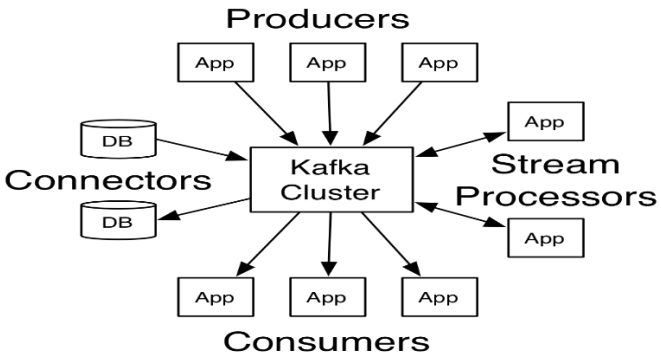

Figure 6: Apache Kafka workflow illustration.

Kafka uses 'Apache ZooKeeper' [5] for log management and synchronization. To move data from distributed SmartX Boxes, we should deploy both ZooKeeper and Kafka in SmartX Visibility Center. Firstly, we should get ZooKeeper (version 3.4.6) from (https://zookeeper.apache.org/releases.html#download) and start ZooKeeper service (bin/zookeeper-server-start.sh config/zookeeper.properties). After that, we should get Kafka (version 0.10.0.1) from (https://kafka.apache.org/downloads), and start Kafka broker (bin/kafka-server-start.sh config/server.properties).

3.2. Visibility DataLake

To store playground visibility data in different formats, Visibility DataLake is setup at SmartX Visibility Center. It is equipped with two NoSQL DataStores: MongoDB [6] and Elasticsearch [7]. Installation steps for both DataStores are given below:

 # MongoDB 3.2 installation

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927

echo"deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.2 multiverse"| sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list

sudo apt-get update

sudo apt-get install –y mongodb-org

 # Elasticsearch 5.6 Installation

wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -

echo "deb https://artifacts.elastic.co/packages/5.x/apt stable main" | sudo tee -a /etc/apt/sources.list.d/elastic-5.x.list

```
# sudo apt update
# sudo apt install elasticsearch
```

3.3. Visibility Integration

'Apache Spark' [8] is adopted as the main analytics engine for flow-layer visibility analysis. In order, to meet various use case integration demands, visibility integrator offers several implementations. At the moment, we have a tree-based data integration and analysis tool for matching visualization application requirements. Also, an implementation for flow-centric visibility is in progress, leveraging Apache Spark SQL libraries for unifying multi-layer metrics data. Apache Spark version 2.2.0 can be installed and executed as following:

```
# wget https://www.apache.org/dyn/closer.lua/spark/spark-2.2.0/spark-2.2.0-bin-hadoop2.7.tgz
# tar –xvzf spark-2.2.0-bin-hadoop2.7.tgz
# cd spark-2.2.0-bin-hadoop.2.7
# ./spark-shell --master local[*] --driver-memory 16g --jars /home/netcs/mysql-connector-java-5.1.16.jar,/home/netcs/elasticsearch-hadoop-6.0.1.jar --conf spark.es.nodes="x.x.x.x"
```

3.4. Visibility Staging and Visualization

Integrated and inspected visibility data is staged to already deployed DataStores but in separate tables for data isolation from historic one. Interactive visualization of playground visibility is following web-based model, so that playground operator and developers can access it globally. The core view of interactive visualization integrates standard graphs and specialized graph for meeting diverse visualization requirements of playground visibility. To visualize flow-layer metric data, we have chosen Kibana [9] due to feature-rich graphs. Also, we considered Vis.js [10] library because it offers a simple and easy to use graph-based library for interactive network graphs development. We used Node.js [11] to create a server for web application. We installed Node.js server and Kibana using below commands:

```
# curl -sL https://deb.nodesource.com/setup_4.x |sudo -E bash -
# sudo apt-get install –y nodejs
# sudo apt-get install apt-transport-https kibana
```

4. Deployment and Verification of SmartX MultiView Software

This chapter provides the deployment results of SmartX MultiView Visibility software, by focusing on flow-layer visibility of OF@TEIN Playground. Implementation of SmartX MultiView Visibility software is also available at: https://github.com/K-OpenNet/OpenStack-MultiView. A shell script Install_Dependencies_vCenter.sh is used to deploy the basic required software in SmartX Visibility Center.

For defining a flow that should identify all the traffic from each tenant is an initial difficulty of flow-layer visibility. So first, a flow is defined by a 5-tuple of 'Source IP', 'Destination IP', 'Source Port', 'Destination Port', and 'VLAN ID', which is extracted from the Ethernet frame header, IP packet header, and transport layer header of network packets. Next, a custom developed python API is executed as following to create the above flow:

```
# python sFlowRestAPI.py
```

After that, we need to start IO Visor program in SmartX Boxes to trace network packets. Also, to identify where a packet is collected in the network, a visibility tag for measurement point is added. In order, to discriminate the flows per tenant, we have a pre-arranged 'VLAN ID' association with each tenant. Then, to discriminate different types of traffic, we rely on the source and destination ports of transport layer (e.g., TCP/UDP protocols). After that flow-layer visibility statistically measures metrics such as: number of flows, type of flows, and flows data volume. IO Visor program is started by using below command:

```
# sudo python packet_tracing_with_kafka_temp.py
```

After that, sampled collected visibility data can be visualized through sFlow-RT UI, deployed in SmartX Visibility Center as shown in Fig. 7. For example, it is showing total sampled traffic in bits per seconds from playground infrastructure. Likewise, visibility data collected from IO Visor can be visualized through Kibana UI as shown in Fig. 8. For example, it is showing total number of packets, collected from each SmartX Box and then grouped based on tenant VLAN ID.

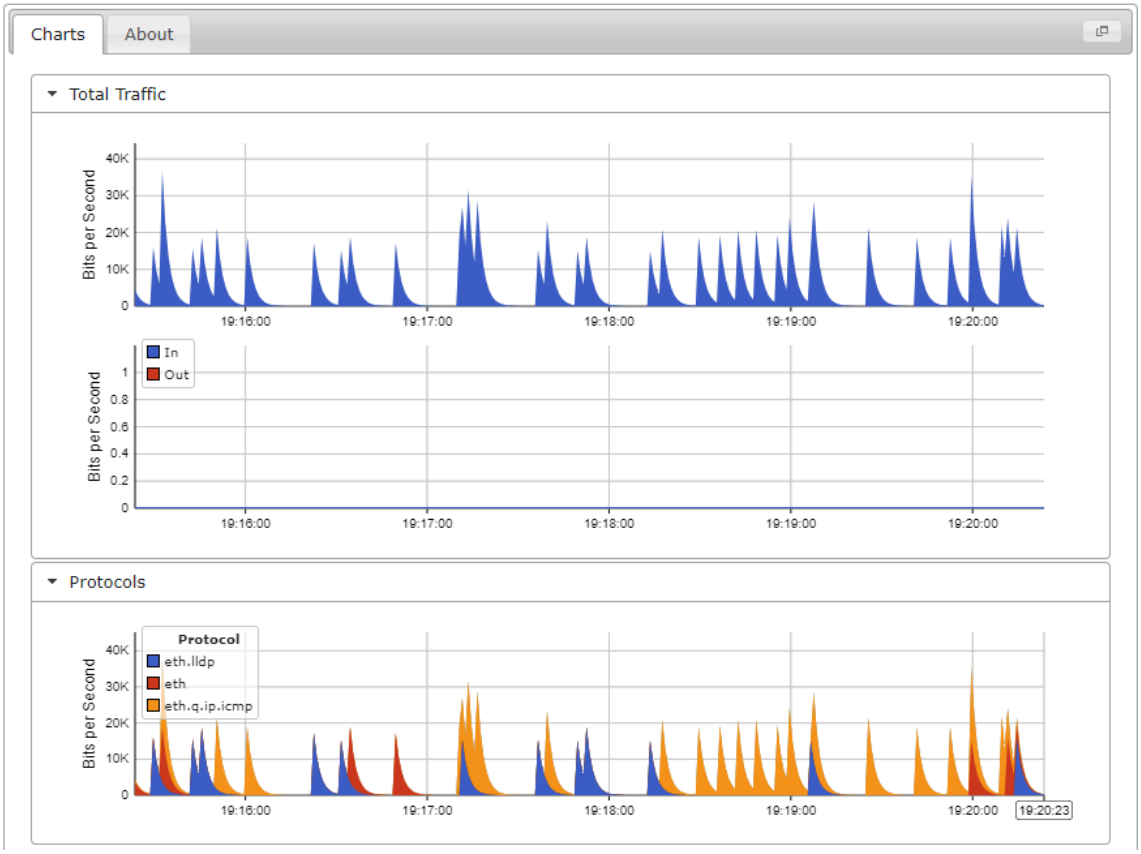

Figure 7: Flow-layer visualization for playground flows.

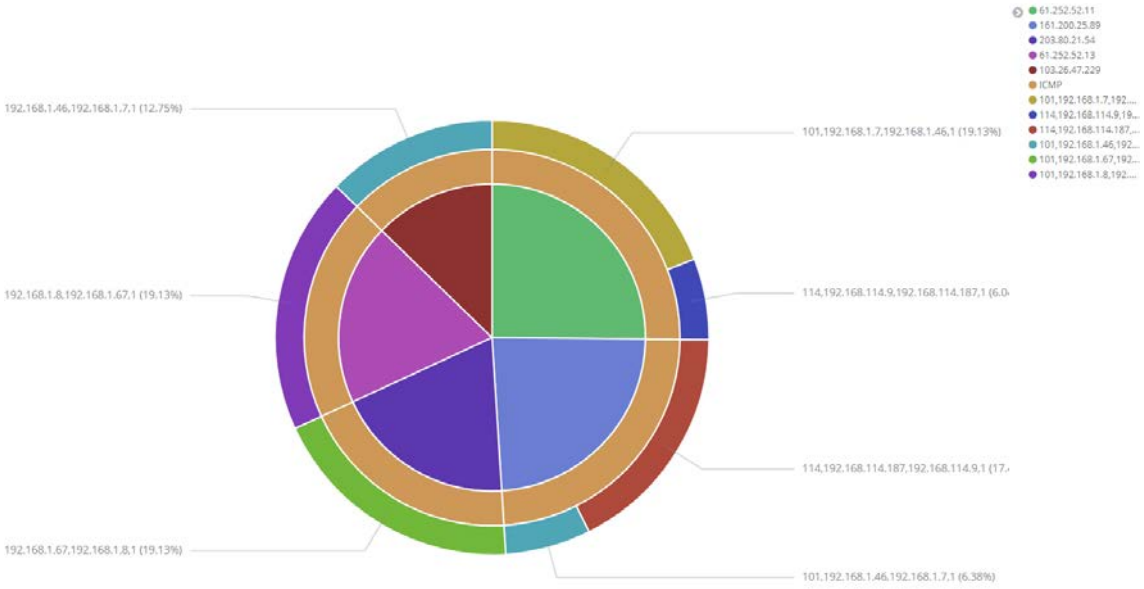

Figure 8: Flow-layer visualization to give insights into each flow.

References

[1] sFlow, http://www.sflow.org/index.php.

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

[3] sFlow-RT, http://www.sflow-rt.com/.

[4] Apache Kafka, http://kafka.apache.org/documentation.html/.

[5] Apache ZooKeeper, https://zookeeper.apache.org/.

[6] MongoDB, https://www.mongodb.com/.

[7] Elasticsearch, https://www.elastic.co/.

[8] Apache Spark, https://spark.apache.org/.

[9] Kibana, https://www.elastic.co/products/kibana.

[10] Vis.js, http://visjs.org/.

[11] Node.js, https://nodejs.org/en/.

K-ONE Technical Document

○ The technical contents of this document can be changed without a notification due to the progress of the project.

○ Refer website below for getting more information of this document.

(Homepage: http://opennetworking.kr/projects/k-one-collaboration-project/wiki, E-mail: k1@opennetworking.kr)

Written by K-ONE Consortium

Written in 2017/05