K-ONE Technical Document #17

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

Document No.	**K-ONE #17**
Version	**1.0**
Date	**2017-05-15**
Author(s)	**Usman, Muhammad** **JungSu Han**

K-ONE

ONK
OpenNetworkingKorea

□ History

Version	Date	Author(s)	Contents
Draft - 0.1	2017. 05. 12	JungSu Han	Chapter 1
0.2	2016. 04. 13	Usman, Muhammad	Chapter 2
0.3	2016. 05. 14	Usman, Muhammad	Chapter 3
0.4	2016. 05. 15	Usman, Muhammad	Chapter 4

본 문서는 **2015**년도 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)

Summary

In this document, we present preliminary design of ′SmartX MultiView Visibility Framework′ for SDN-enabled Multi-site Cloud. In Chapter 1, we provide an overview of OF@TEIN Playground and significance of visibility and visualization in operation. In chapter 2, we provide detailed explanation of each step of ′SmartX MultiView Visibility Framework′. In chapter 3, we provide the initial-stage implementation of ′SmartX MultiView Visibility Software′ to enable resource-layer visibility monitoring and visualization. Finally, we show how to deploy ′SmartX MultiView Software′ to SDN-enabled multi-site cloud playground.

Contents

K-ONE 기술 문서 #17 Design of SmartX MultiView Visibility Software for SDN-enabled Multi-site Cloud

1. Introduction 8

1.1. Introduction 8

2. Design of SmartX MultiView Visibility Framework 11

2.1. Visibility Collection and Validation 11

2.2. Visibility DataLake and Analytics 11

2.3. Visibility Integration 12

2.4. Visibility Staging and Visualization 12

3. Implementation of SmartX MultiView Visibility Software 13

3.1. Visibility Collection and Validation 13

3.2. Visibility DataLake 15

3.3. Visibility Integration 16

3.4. Visibility Staging and Visualization 16

4. Deployment of SmartX MultiView Software 19

List of Figures

Figure 1: OF@TEIN playground as multi-site SDN-Cloud spread across multi-domains. 9

Figure 2: Design of 'SmartX MultiView Visibility Framework' 11

Figure 3: 'Intel Snap Telemetry Framework' simplified workflow illustration. 13

Figure 4: Configuration file used during creation of task. 14

Figure 5: 'Apache Kafka' workflow illustration. 15

Figure 6: Visibility integration model using tree structure. 16

Figure 7: p+v topological visualization of playground resources. 20

Figure 8: Grafana dashboard to present performance graphs. 20

List of Tables

Table 1: Resource-layer Visibility and Visualization Use Cases 19

K-ONE #17. Design of SmartX MultiView Visibility Software for SDN-enabled Multi-site Cloud

1. Introduction

1.1. Introduction

In this chapter, we briefly discuss about software-defined networking (SDN) and cloud technologies, background of OF@TEIN/OF@KOREN Playgrounds, operational challenges, and our proposed solution.

Recently, Software Defined Networking (SDN) has attracted the interest from both industry and research community. SDN proposes to decouple data plane from control plane. Data plane functionality of packet forwarding is built into switching fabric, while control plane functionality of managing network devices is placed in logically centralized software called controller. This separation simplifies the network management tasks and reduces the associated complexities of distributed configurations. Due to cost effectiveness of SDN approach, it enables researchers to perform experiments which were too expensive or difficult to perform previously. Additionally, industry is also adopting to vendor independent network management protocols like OpenFlow for managing their networks.

Cloud computing is another emerging paradigm which attracted the interest of large community of service providers and consumers due to its pay per use service model. It removes the upfront cost of purchasing physical hardware and hides network connectivity complexities from consumer's side. Cloud computing has tendency to deliver on-demand IT resources by dynamically scaling its provisioning service. Moreover, cloud computing offers connectivity on fly to its consumers due to its availability over Internet.

At present, a lot of research is going on to establish Future Internet testbeds to provide experimental networking facilities to effectively test and evaluate aforementioned technologies before their actual deployment in real production environments. Aligned with Future Internet testbed projects like GENI and Fire, we launched OF@TEIN and OF@KOREN projects. These playgrounds provide an open SDN-enabled multi-site cloud infrastructure to facilitate advanced academic research and experimentation. OF@TEIN is composed of distributed and multi-domain sites spread over 14 international sites, across 9 countries (i.e. Korea, Malaysia, Thailand, Indonesia, Philippines, Pakistan, Taiwan, Vietnam, and India) as depicted in Fig 1.

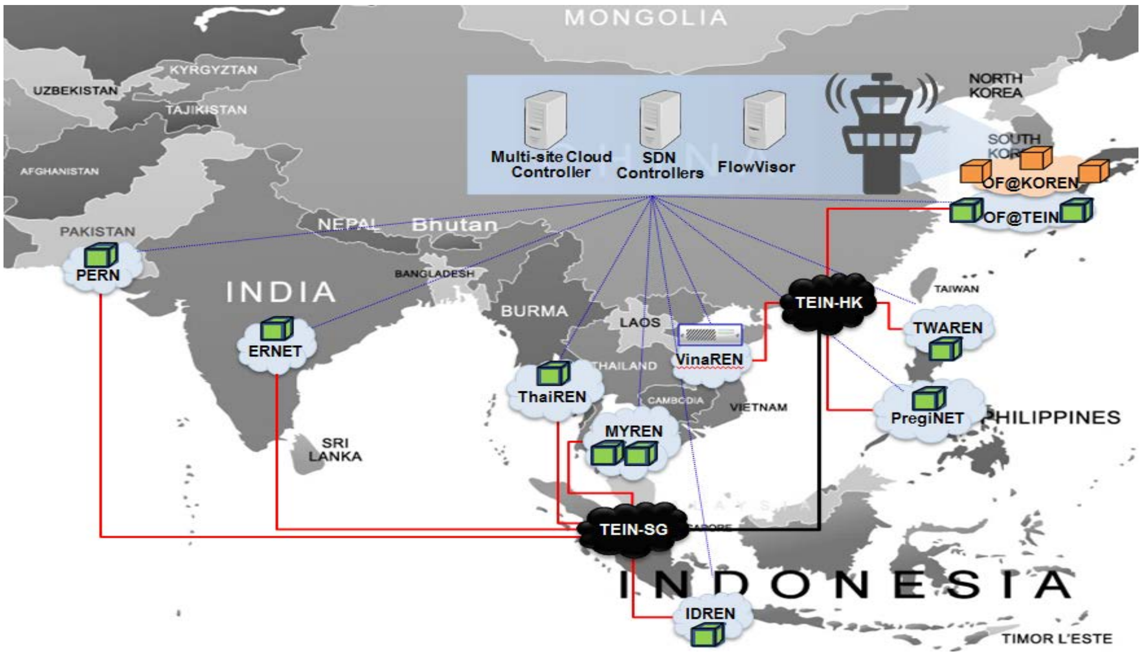


Figure 1: OF@TEIN playground as multi-site SDN-Cloud spread across multi-domains.

At these playgrounds, distributed SmartX Boxes hosts the OpenStack instances and control overlay virtual networks across different geographical locations. Playground users, on-demand requests OpenStack instances (i.e. managed as separate tenants) and overlay virtual networks (i.e. isolated from other users using VLAN's) in SmartX Boxes to perform experimentations. The underlay infrastructure of OF@TEIN Playground spreads across several research and education networks (REN's) with different administrative domains.

In emerging SDI-enabled testbeds, like OF@TEIN Playground, visibility and visualization are one of major operation challenges, since multi-site resource islands with complicated end-to-end connectivity is demanding various operation tools for multiple tenants at sophisticated level. That is, with the support of visibility and visualization, playground operators gain timely insights of the operational states of their playground resources and associated flows. This collected visibility data plays a vital role in maintaining the health level of playground resources, tied with agile outage detection and recovery.

Based on the operation experience with OF@TEIN Playground since year 2012, we seriously felt the need for well-designed visibility and visualization tools, comprised of concrete steps to collect, validate, integrate, store, and visualize the operational states of overall playground.

However, due to the unique deployment style of single-box-virtualized multi-site OF@TEIN Playground, there is no directly matching open-source visibility and visualization tool available.

By uniquely integrating various visibility related open-source upstream software, we propose an integrated design for multi-layered visibility workflow, denoted as 'SmartX MultiView Visibility Framework'. That is, the proposed framework attempts to enable an effective design for visibility and visualization workflow, associated with the identification of open-source software to be leveraged, the software implementation of integration framework, and the playground-based verification and visualizations.

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

2. Design of SmartX MultiView Visibility Framework

This chapter discusses overall design of 'SmartX MultiView Visibility Framework' by focusing on four core steps: Visibility Collection and Validation, Visibility DataLake and Analytics, Visibility Integration, and Visibility Staging and Visualization as depicted in Fig. 2.

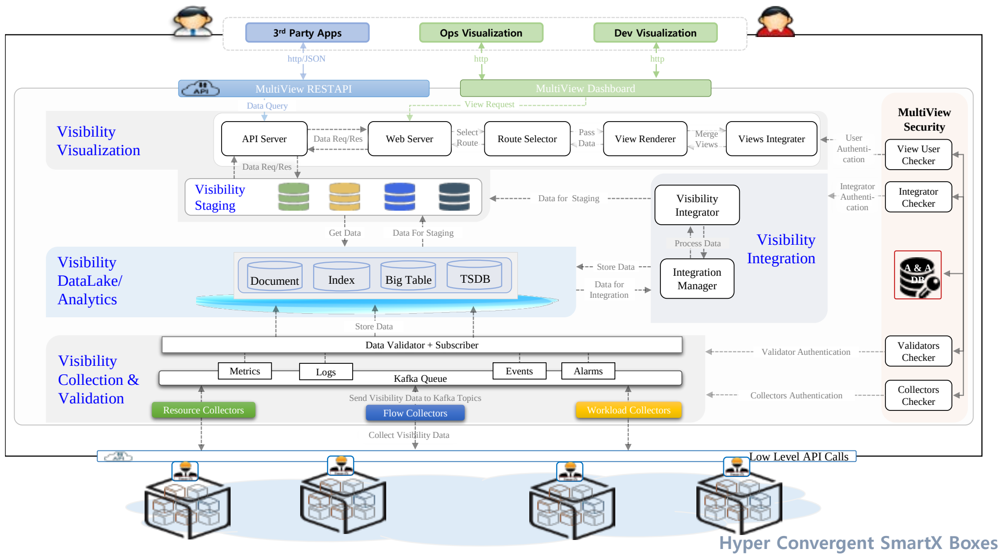


Figure 2: Design of 'SmartX MultiView Visibility Framework'

2.1. Visibility Collection and Validation

Identification of targeted sources from where data is required to be measured is initial step in visibility collection and validation. This source identification and associated parameters selection process must be performed very carefully since afterwards, analysis will be dependent on the consistent quality of data collections. Collected visibility data is required to be reliably moved to 'SmartX Visibility Center' (that is a placeholder for visibility and visualization tools) for further processing. It is followed by visibility validation step, which verifies the contents of visibility data before taking any further actions. Subsequently, validated visibility data is passed through initial formatting and assigned with specific tags.

2.2. Visibility DataLake and Analytics

Visibility data should be preserved either momentarily or permanently before getting utilized. For this, we adopt the concept of 'DataLake' in storing vast amount of structured, semi-structured, or un-structured data in its native format. Also, the associated commoditytype storage hardware must handle diverse types and sizes of data in a fault-tolerant manner. Thus, multiple dedicated 'DataStores' can be deployed over DataLake to store specified visibility data in different formats. It is possible to store playground configurations, logs, and time series data in separate DataStores. Also, associated data schema, to define the collection of data objects as tables/views adopts schema-on-read strategy, since it can provide flexibility in dealing with operational complexity. Visibility Analytics is responsible for providing tools for BigData as well as FastData processing to automatically extract meaningful information e.g. anomalies detection from visibility data.

2.3. Visibility Integration

Collected visibility data is not ready for visibility integration (e.g., data fusion with analytics) because data is collected from multiple isolated sources at distributed locations. Therefore, visibility integration that fuses collected visibility data is essential to identify key patterns and anomaly detection. The main difficulty is how to identify key parameters to be integrated. Also, it is very hard to isolate the impact of selected parameters on the overall operation of playground resources. Thus, visibility integration must be empowered by 'BigData/FastData' capability. It is necessary to construct analysis models to cater visibility integration difficulties. Also, visibility integration should handle large-volume time-sensitive visibility data from multiple distributed sources.

2.4. Visibility Staging and Visualization

Recall that the main objective of DataLake is to retain raw (but validated) data in its native format, leaving visibility applications to specifically query data matching their requirements. Thus, visibility staging is introduced to hold ready-to-use visibility data collections to facilitate visibility application requests. Staged visibility data may originate directly from visibility collection/validation, from DataLake queries, or from visibility integration. This ready-to-use data is useful for interactive visualization since it removes data preparation overhead. Finally, visibility visualization enables the visual rendering to observe the operational status of playground resources and associated flows. Visibility visualization must cover different viewing requirements ranging from simple bar/line charts to specialized graphical views. Also, for remote playground operators and users, visibility visualization should provide open access to visibility data and web-based visualization UIs.

3. Implementation of SmartX MultiView Visibility Software

In this chapter, implementation details of 'SmartX MultiView Visibility Software' to enable resource-layer visibility are discussed by leveraging SmartX 'MultiView Visibility Framework', which we presented in previous section.

3.1. Visibility Collection and Validation

We used 'Intel Snap Telemetry Framework' [4] for visibility collection from different resources. That is an open-source telemetry framework written in Glolang for consumption of data center telemetry data. This framework offers three types of plugins: collectors, processors, and publishers. 'Snap' workflow starts with data collection at all layers by using collector plugins, then visibility data is transformed with basic filters using processor plugins, and finally visibility data is published to one or more platforms for consumption using publisher plugins. Fig. 3 shows simplified workflow of 'Intel Snap Telemetry Framework'.

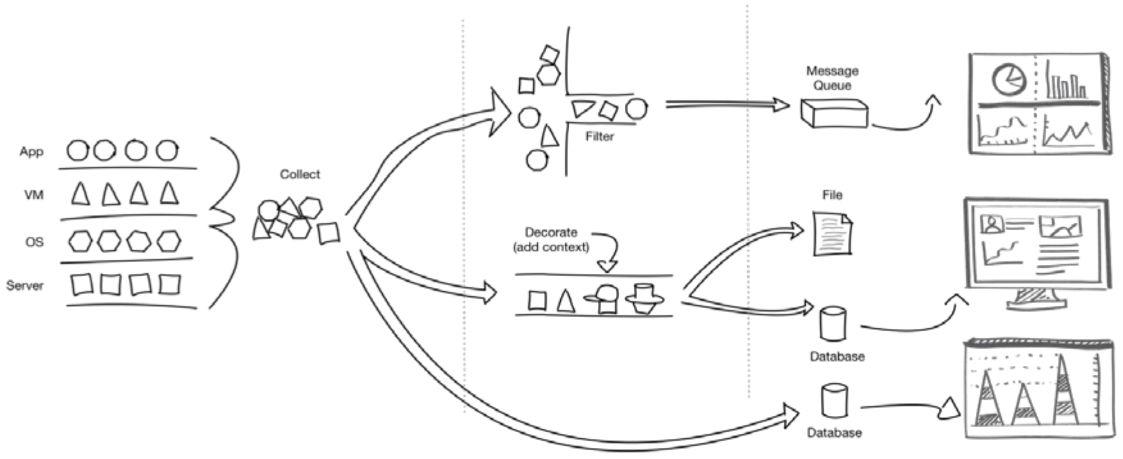
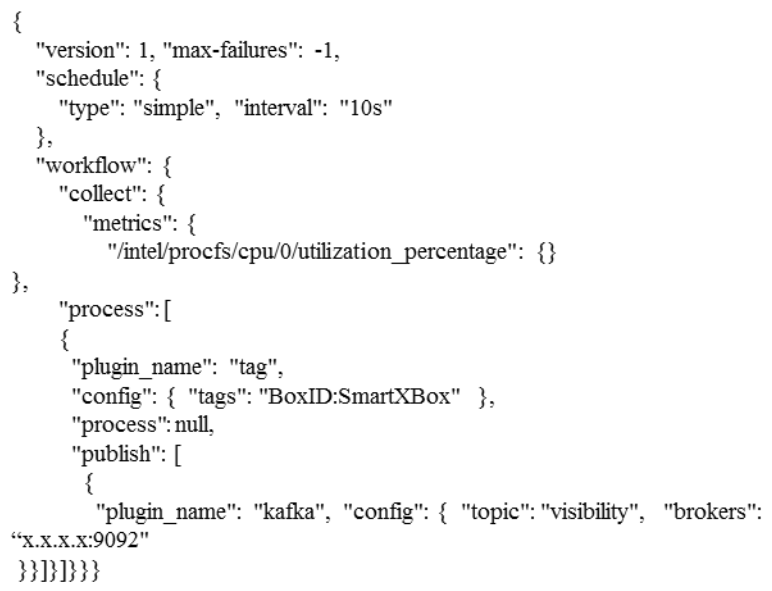


Figure 3: 'Intel Snap Telemetry Framework' simplified workflow illustration.

To collect visibility data from SmartX Boxes, first, we need to get latest release of 'Intel Snap Telemetry Framework' from GitHub (https://github.com/intelsdi-x/snap/releases). Next, we need to startup 'Snap' daemon. After that, we start collector plugins, processor plugins and publisher plugins simultaneously. A configuration file is used to control which metrics to be measured. By using this template file, a Snap task is initiated to load plugins and start collection. We also execute a management script for Snap collector itself to recover in case of failures. A task is workflow, which uses this configuration file for starting complete process as shown in Fig. 3.



```
{
  "version": 1, "max-failures":  -1,
  "schedule": {
    "type": "simple",  "interval":  "10s"
  },
  "workflow": {
    "collect": {
      "metrics": {
        "/intel/procfs/cpu/0/utilization_percentage":  {}
},
    "process": [
    {
     "plugin_name":  "tag",
     "config": {  "tags": "BoxID:SmartXBox"   },
     "process": null,
     "publish": [
      {
       "plugin_name":  "kafka",  "config": {  "topic": "visibility",   "brokers":
“x.x.x.x:9092"
 }}]}]}}}
```

Figure 4: Configuration file used during creation of task.

Collected (and to-be-validated) visibility data is moved to ‘SmartX Visibility Center’ through ‘Apache Kafka’ [5] and placed in a queue provided by Kafka broker. Kafka is an open-source message broker 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. Kafka has three components namely Producers (writes data to Kafka topics), Kafka cluster (stores data in partition identified by name “topic”), and consumers (end applications which consumes stored data). In Fig. 4 overall workflow of Kafka is presented. First of all, we need to deploy ‘Apache Kafka’ in ‘SmartX Visibility Center’ to enable data movement functionality. Kafka uses ‘Apache ZooKeeper’ [6] for log management and synchronization. So, first we need to get ZooKeeper version 3.4.6 and start zookeeper server (bin/zookeeper-server-start.sh config/zookeeper.properties). To deploy ‘Apache Kafka’, we need to get latest version (0.10.0.1) from (https://kafka.apache.org/downloads). After downloading Kafka distribution, we start Kafka broker (bin/kafka-server-start.sh config/server.properties) and create required topics for storing visibility data. By using Snap publisher plugin for Kafka, visibility data is delivered to Kafka topic by using Kafka publisher API.

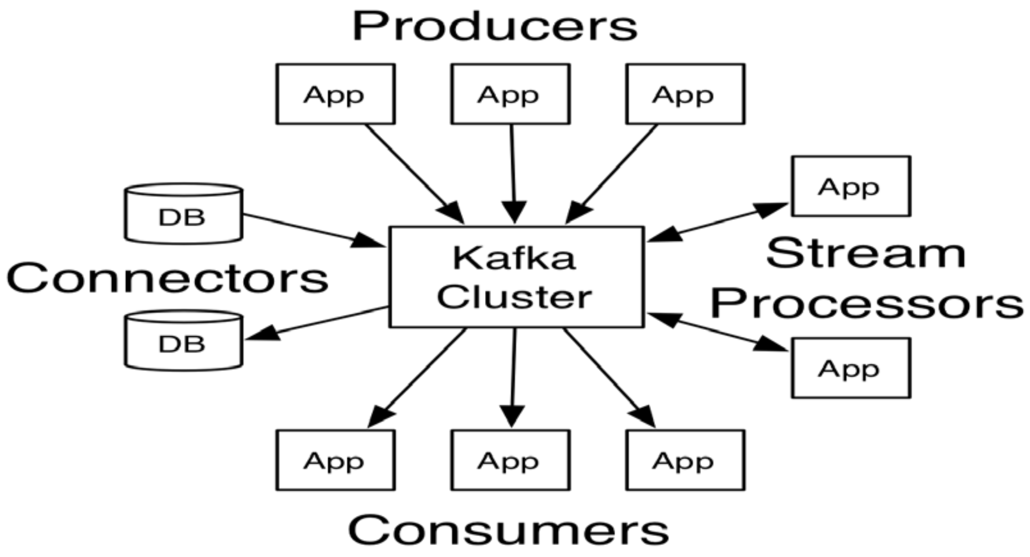


Figure 5: 'Apache Kafka' workflow illustration.

3.2. Visibility DataLake

We used MongoDB, Elasticsearch, and InfluxDB DataStores inside DataLake to provide data preservation functionalities. Currently, DataLake is designed to be deployed at 'SmartX Visibility Center'. Installation procedures for each of these DataStores are given below:

```
# MongoDB 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 Installation
# wget https://download.elastic.co/elasticsearch/elasticsearch/elasticsearch-1.7.2.deb
# sudo dpkg –i elasticsearch-1.7.2.deb
# sudo update-rc.d elasticsearch defaults
```

```
# InfluxDB Installation
# wget https://dl.influxdata.com/influxdb/releases/influxdb_1.0.2_amd64.deb
# sudo dpkg -i influxdb_1.0.2_amd64.deb
```

3.3. Visibility Integration

Visibility Integration is performed via tree structure where each node represents attribute, branches represent values of attributes and leafs represents end result. Different colors at leaf nodes represent special status messages to facilitate the playground users for faster troubleshooting by providing exact visual information. Let's, follow a particular path in tree to understand how decisions are made for enabling p+v topological visibility integration. Process starts by identifying type of resource. In case of physical resource, physical interconnects are checked. If we follow the "Data" branch and result of attribute accessible is "yes" then tunnel is up data plane is working fine. In case, data plane is not accessible then interface status is checked and based on interface status attribute next decision is made. In case, interface is up then routes are checked and based on result of routes checking appropriate decision is made like "tunnel issue" or "route issue". This implementation of Visibility Integration is covered via Java and JavaScript languages.

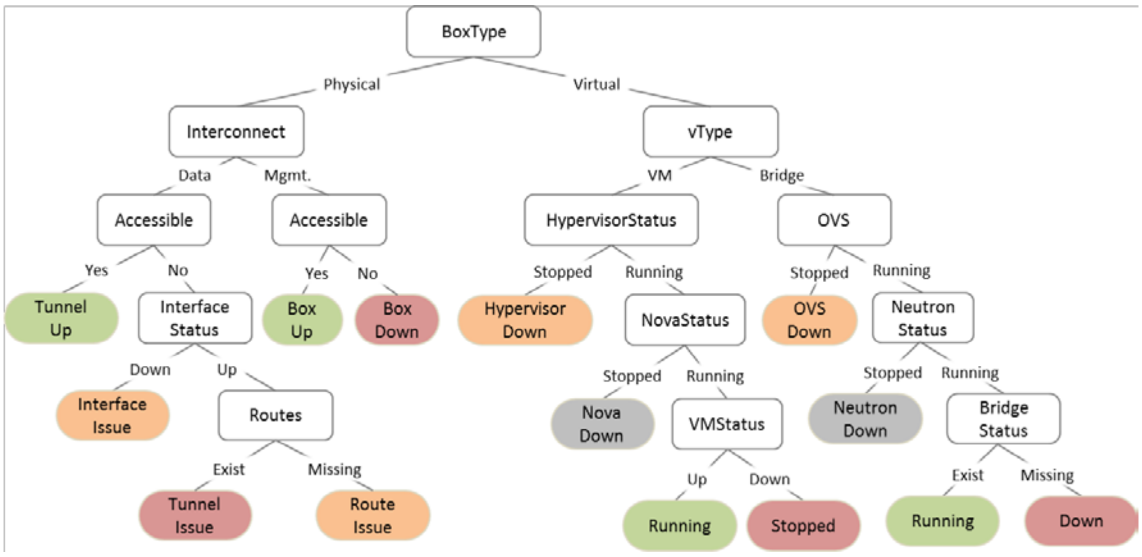


Figure 6: Visibility integration model using tree structure.

3.4. Visibility Staging and Visualization

In initial-stage implementation, same DataStores are used for staging visibility data but with different collections (i.e. tables), e.g. resourcelevel-ppath collection contains all the data for network links while resourcelevel-ppath-rt collection contains only latest staged data for links information.

For visualization implementation, we used topological visualization technique as baseline. In topological visualization implementation main challenge is the collection of visibility information from network and showing dynamically on GUI using graph-based technique with nodes (representing resources) and edges (representing interconnects). So, we considered Vis.js [7] library for graph-based drawings. Vis.js offers a simple and easy to use graph-based library for interactive network graphs development. Vis.js can also handle large amounts of dynamic data, and interaction with the DataStores. We downloaded Vis.js library using "npm install vis" and used in our JavaScript code for creating topology. We used Node.js [8] to create a server for web application, since Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Furthermore, Node.js provides number of plugins, to help users to easily build complex applications. We installed Node.js server version 4.x using commands:

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

We used Grafana [9] for performance metrics dashboard creation. Grafana provides a powerful and elegant way to create, explore and share dashboard and data. Grafana offers support for number of backhand DataStores. So, we deployed Grafana for creating custom dashboard and graphs. We deployed Grafana version 3.1.1 using Debian package:

```
# wget https://grafanarel.s3.amazonaws.com/builds/grafana_3.1.1-1470047149_amd64.deb
# sudo apt-get install -y adduser libfontconfig
# sudo dpkg –i grafana_3.1.1-1470047149_amd64.deb
```

Snippet of JavaScript code for server.js is shown below, which is used for dependencies inclusion, routes redirection, and data API calls.

```
/** Module dependencies. */
var express = require('express');
var ArticleProvider = require('./resource-mongodb').ArticleProvider;
var RouteProvider = require('./route-mongodb').RouteProvider;
var BoxProvider = require('./multiview-hierarchical-mongodb').BoxProvider;
var UserProvider = require('./user-mongodb').UserProvider;
var http = require("http");
var app = module.exports = express.createServer();
var client = require('socket.io').listen(8080).sockets;
// Configuration
app.configure(function(){
app.set('views', __dirname + '/views');
app.set('view engine', 'jade');
app.use(express.bodyParser());
app.use(express.methodOverride());
app.use(require('stylus').middleware({ src: __dirname + '/public' }));
app.use(app.router);
app.use(express.static(__dirname + '/public'));
});
app.configure('development', function(){
});
app.configure('production', function(){
app.use(express.errorHandler());
});
//Define Application Routes
var resourceProvider = new ResourceProvider();
app.get('/resourcecentricviewops', function(req, res){
var boxList = null;
var switchList = null;
var instanceList = null;
var serviceList = 0;
var ovsBridgeStatus = null;
var pPathStatus = null;
resourceProvider.getpBoxList( function(error,boxobj)
{
boxList = boxobj;
console.log( boxList);
showView();
})
```

4. Deployment of SmartX MultiView Software

This chapter provides the deployment results of 'SmartX MultiView Visibility' software, by focusing on Resource-layer visibility in SDN-enabled Multi-site Playgrounds. Initial implementation of SmartX MultiView Visibility software is available at: https://github.com/K-OpenNet/OpenStack-MultiView. Two shell scripts Install_Dependencies_vCenter.sh and Install_Dependencies_vCenter.sh are used to deploy the required components in both 'SmartX Box' and 'SmartX Visibility Center'. For collecting the visibility information, initial topology is created via Create_MultiView_Database.sh, which in turn relies on MultiView_Configuration_Database.js file. A list of use cases currently covered by 'SmartX MultiView Visibility' software for Resource-layer visibility and visualization are listed in Table 1.

Table 1: Resource-layer visibility and visualization use cases.

Categories	**Use Case**	**Severity**
Infrastructure Services	Hypervisor service checking	High
	OpenvSwitch service checking	High
	Compute service checking	High
	Networking service checking	High
Topology	Topology verification	High
	Resources GeoMap View	Low
Inter-connection	Data Plane checking	Low
	Management Plane checking	High
	Tunnel verification	Low
	Links checking	Low
Measurements	CPU, Memory, Network, Processes, Latency, etc.	Low

In Fig. 7., p+v topological visualization of SDN-enabled multi-site playgrounds to illustrate numerous operational states exhibited by different type of resources at any particular time are provided. To get detailed information about any particular 'SmartX Box', resource performance graphs are created and merged together in single dashboard via Grafana as shown in Fig. 8.

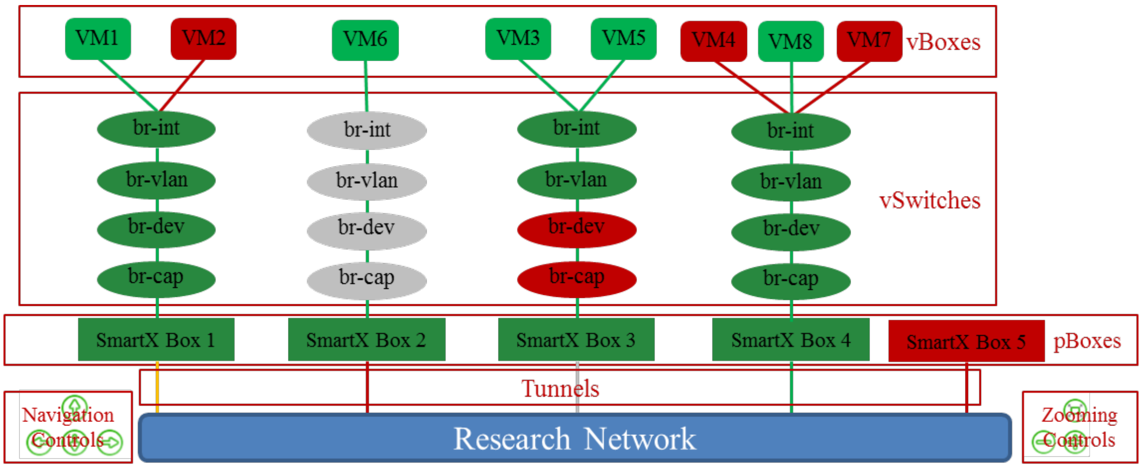


Figure 7: p+v topological visualization of playground resources.

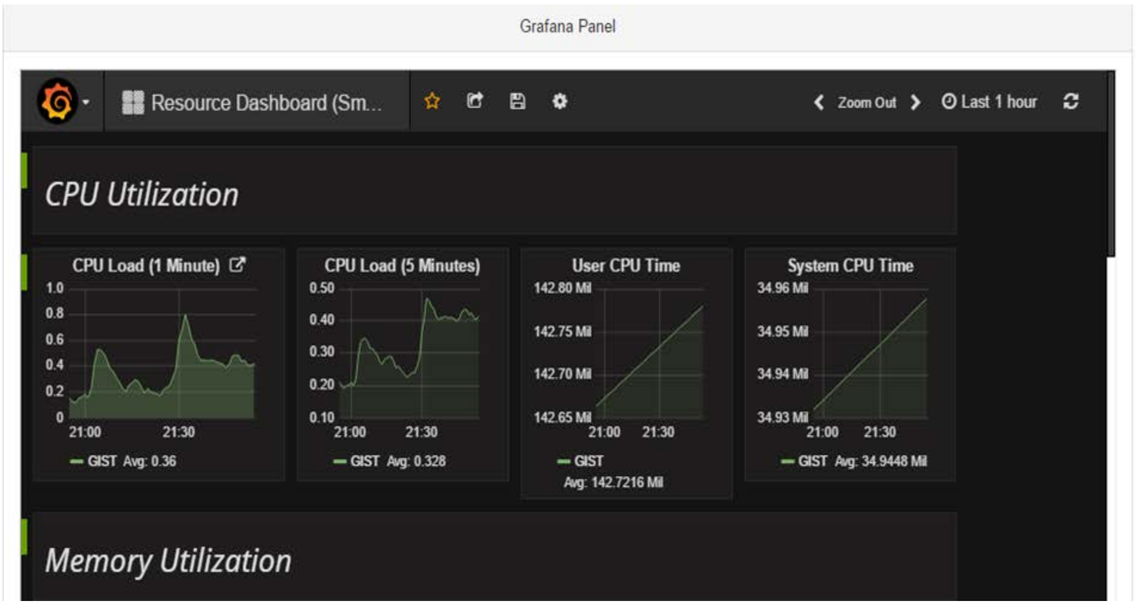


Figure 8: Grafana dashboard to present performance graphs.

References

[1] Elasticsearch, https://www.elastic.com/.

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

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

[4] Intel Snap Framework, http://sanp-telemetry.io/.

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

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

[7] Visjs, http://visjs.org/.

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

[9] Grafana, http://grafana.org/.

K-ONE Technical Document

Written by K-ONE Consortium

Written in 2017/05