applied sciences

MDPI

Article

SmartX Multi-View Visibility Framework with Flow-Centric Visibility for SDN-Enabled Multisite Cloud Playground

Muhammad Usman, Muhammad Ahmad Rathore and JongWon Kim *

School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology, Gwangju 61005, Korea; musman@gist.ac.kr (M.U.); ahmadrathore@gist.ac.kr (M.A.R.)

* Correspondence: jongwon@gist.ac.kr; Tel.: +82-62-715-2219

Received: 5 April 2019; Accepted: 14 May 2019; Published: 17 May 2019

Abstract: Modern information communication technologies (ICT) infrastructures are getting complicated to cope with the various demands needed to accommodate the emerging technology paradigms such as cloud, software-defined networking (SDN), and internet of things (IoT). Visibility is essential for the effective operation of such modern ICT infrastructures to easily pinpoint server faults, network bottlenecks, and application performance troubles. Even though many conventional monitoring solutions are now available, multi-layer visibility is still limited when operating such complicated infrastructures. To address this particular limitation, a futuristic multi-layer visibility framework denoted as SmartX multi-view visibility framework (MVF), is proposed for unifying the visibility of underlay, physical and virtual resources, flow, and workload layers. To unify multi-layer visibility, this paper presents a comprehensive extension of SmartX MVF with flow-centric visibility for simultaneously monitoring physical-virtual resources, flows classification, and visualization to eventually assist secured operation of SDN-enabled multisite cloud infrastructure. Flow-centric visibility design has five main components (1) a lightweight network packets-precise flows visibility collection component, (2) a visibility data aggregation and tagging component, (3) a multi-layer visibility data integration component, (4) a non-learning-based network packets flows classification component, and (5) a visualization component. Furthermore, a prototype implementation of SmartX MVF with flow-centric visibility is deployed in an SDN-enabled multisite cloud playground to verify the proposed multi-view visibility of fine-grained flow-aware physical-virtual resources.

Keywords: software-defined networking; cloud computing; future internet testbed; flow-centric visibility; interactive visualization; monitoring technology

1. Introduction

Recently, software-defined networking (SDN) [1] has captivated the interest of both the industry and research community since it proposes to decouple data-plane functionality from control plane functionality. Data-plane functionality for programmable packet forwarding is built into switching fabric, while the control plane functionality of managing network devices is placed in a logically centralized software controller(s). This separation simplifies network management tasks and minimizes the associated complexities of distributed network configurations. Because of the cost-effectiveness of SDN approach, it enables developers to perform various experiments, which were too costly or difficult with legacy network devices. Industry is adapting to vendor-independent interface protocols, like OpenFlow, for managing network devices [2,3]. Cloud computing [4] is another big emerging paradigm, which attracts the interest of service providers and consumers 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. What is more, cloud computing can offer distributed intercontinental connectivity to its developers [5].

Various research works have concentrated on establishing research testbeds to provide advanced experimental networking facilities for evaluating emerging software-defined technologies. Aligned with future internet testbed projects like GENI [6] and FIRE [7], we launched OF@TEIN Playground [8] to serve as SDN-enabled multisite clouds to facilitate miniaturized academic experiments. OF@TEIN Playground is composed of ten distributed sites, spread over nine countries (i.e., Korea, Malaysia, Thailand, Indonesia, Philippines, Pakistan, Taiwan, Vietnam, and India).

At the playground, multi-tenant and multisite underlay infrastructure asserts the main challenges when dealing with heterogeneous resource monitoring, flows identification, and deployed workload performance measurements. Playground resources that are distributed at multiple geographical sites, utilize multiple planes for different connectivity purposes where associated flows monitoring is very challenging. Also, playground spreads over heterogeneous underlay networks, where exploitation is easy due to lack of integrated flow monitoring support tools for the tiny-size operator team. In order to address these challenges, we are developing a multi-layered visibility (underlay, physical, and virtual resource-layers, flow-layer, and workload-layer) solution denoted as SmartX multi-view visibility framework (MVF) [9], but flow-layer visibility with the desired level of flow-centric visibility is missing and should be added. Visibility is defined as the state of being able to see or be seen. As compared to the monitoring that solely focuses on measurements data, the term visibility referred here covers both measurements data and data for tracing. Basically, flow-layer should be extended to be called flow-centric visibility that can smoothly integrate several layers of visibilities together and provide certain level of flow information such as flow type and statistics, flow class identification, flow physical-virtual topology, etc.

In this paper, we list flow-centric visibility requirements for SDN-enabled multi-site clouds and then extend SmartX MVF to process multi-layer visibility data from diverse measurement points over the OF@TEIN Playground. The proposed extension of framework attempts to deliver an integrated design for both flow-centric visibility and visualization workflow. As compared to the original SmartX MVF design that focused on defining base stages, in this paper, we mainly focus on the technical details about visibility collection from distributed SmartX boxes and multi-layer visibility integration with supporting visualizations. Flow-centric visibility adopts the following approaches. First, we enable lightweight, packet-precise, visibility collections from distributed sites. Then, visibility data aggregation, tagging, and integration uses an analytics engine for smooth processing of playground visibility data. Finally, network packet flows are classified using key attributes from multiple layers of visibilities and results are disseminated in the form of interactive visualizations to the playground users. We also identify various tools and provide the software implementation for the proposed solution. In addition, we present the real testbed-scale verifications to highlight the potential of the proposed flow-centric visibility for delivering several types of visibility. In summary, the following key contributions are made in this work.

- We list flow-centric visibility requirements of unified visibility framework (SmartX MVF) for SDN-enabled multi-site clouds. These requirements are clearly explained in order to effectively provide a clear mapping with flow-centric visibility design choices.
- By leveraging SmartX MVF, we propose a unique integrated design for flow-centric visibility. We provide the prototype implementation of flow-centric visibility for the specific case of OF@TEIN Playground by focusing on the physical-virtual topology of the playground.
- The proposed flow-centric visibility is verified as capable of enabling visibility data collection from distributed sites and enables interactive visualization of classified visibility data to support the sustained operation of playground. Also, the results highlight the proposed design potential to process and describe complicated flow information in a simplified presentable manner.

The rest of this paper is organized as follows. In Section 2, we provide an overview of OF@TEIN Playground, list flow-centric visibility requirements, and briefly summarize related work. In Section 3, we provide an integrated design and prototype implementation of flow-centric visibility, which is followed by verification results in Section 4. Finally, in Section 5, we conclude the paper.

2. Background and Requirements of Flow-Centric Visibility

In this section, we introduce the key terminologies related to playground visibility for the specific case of OF@TEIN Playground. That is, OF@TEIN is an SDN-enabled multisite cloud playground that dynamically provides dedicated resources to the developers for performing various miniaturized experiments. Visibility is essential for the effective operation of OF@TEIN Playground in order to understand and troubleshoot server and network issues before they affect the developers. Now, we briefly discuss the flow-centric visibility requirements with a brief survey of related work.

2.1. OF@TEIN Playground as an SDN-Enabled Multisite Cloud Playground

Aligned with Future Internet testbeds, we established OF@TEIN Playground, which is an SDN-enabled multisite cloud playground [8,10]. OF@TEIN Playground connects ten international sites spread over nine Asian countries to dynamically provide dedicated resources for performing various miniaturized experiments to the developers. In order to automatically build, operate, and utilize multisite playground, we have a SmartX playground tower (playground tower), which provides a logical space in a centralized location by following the concept of monitor and control tower. Playground tower systematically covers various functional requirements of operating multisite playground by employing several entities as depicted in Figure 1. First, ID center is responsible for OpenStack-based keystone authentication and provides OpenStack Horizon Web UI. The provisioning center (pro center) is responsible for remote installation and configuration of multi-site playground resources. The visibility center (vis center) covers playground visibility and provides panoramic visualization support. The security center (sec center) and SDN controllers take care of the security challenges of keeping the playground safe and inter-connects playground sites by using virtual extensible local area network (VXLAN) tunnels.

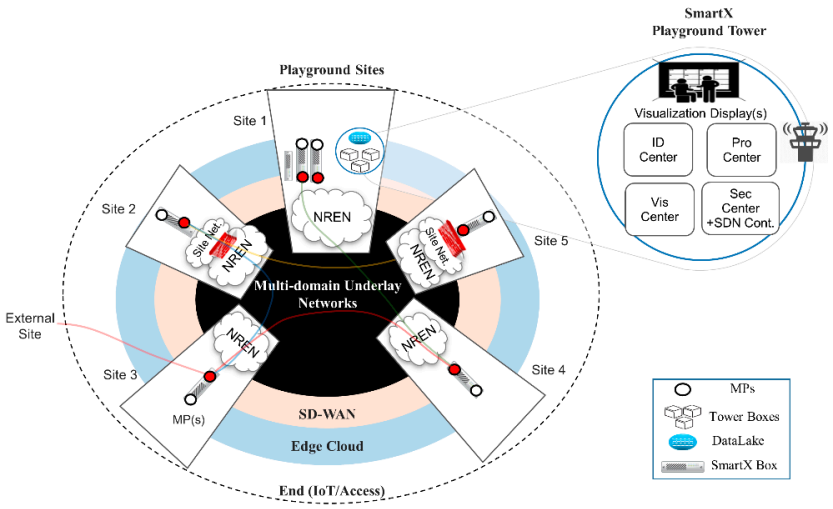


Figure 1. OF@TEIN playground as a software-defined networking (SDN)-enabled multisite cloud.

At OF@TEIN playground, distributed SmartX boxes [10] hosts the virtual machine (VM) instances of OpenStack cloud [11] and controls virtual networking overlay among multiple sites. Playground developers request OpenStack VM instances and virtual local area network (VLAN) isolated overlay virtual networking from distributed SmartX boxes to perform various diversified networking experiments [12]. Distributed SmartX boxes support multi-tenancy by hosting any number

of dedicated virtual entities for each tenant. By using five virtual switches at each SmartX box, we implement required virtual networking features. Three virtual switches: br-int (integration bridge), brvlan (VLAN tagging bridge), and br-ex (external bridge) are used by the cloud portion of SmartX box. Two virtual switches: br-dev (developer's switch) and br-cap (operator switch) are used to enable SDN capability at the playground. Developer switch is sliced based on flowspace allocation under FlowVisor OpenFlow controller and utilized by several developers simultaneously. Operator switch provides output ports for the developer switch, which is mapped into several VXLAN-based tunnel ports for inter-site connections. Additionally, the underlay network for the playground is spread across several research and education (R&E) networks under distinct administrative domains. For instance, an interconnection between any two sites of the playground involves from three to seven heterogeneous underlay networks.

2.2. Flow-Centric Visibility Requirement for Unified Monitoring

We proposed SmartX MVF to deal with the multiple layers of visibility with associated visualization support for SDN-enabled multisite cloud [13]. Among multiple visibility layers, as shown in Figure 2, flow-layer is becoming crucial because it can help us to smoothly integrate several layers of visibilities together and verify packets that are flowing through the playground. A network flow is the sequence of packets that belong to a certain network session between two endpoints. Typically, flows are identified by a five-tuple consisting of source IP address, source port, destination IP address, destination port, and transport layer protocol. Flow information is useful for understanding network behavior by denoting source address and destination address to understand flow origin, port information to characterize application traffic utilization, device interface identification to indicate traffic utilization for specific network devices, and matched packets and bytes to show the amount of traffic flow. This view of the flow-centric visibility can be used to instantly highlight congested links and identify the source of the flow and the associated application-level conversations. To enable a practical, sustainable version of flow-centric visibility to effectively collect, integrate, and visualize visibility data of multiple layers together, the following key requirements should be fulfilled.

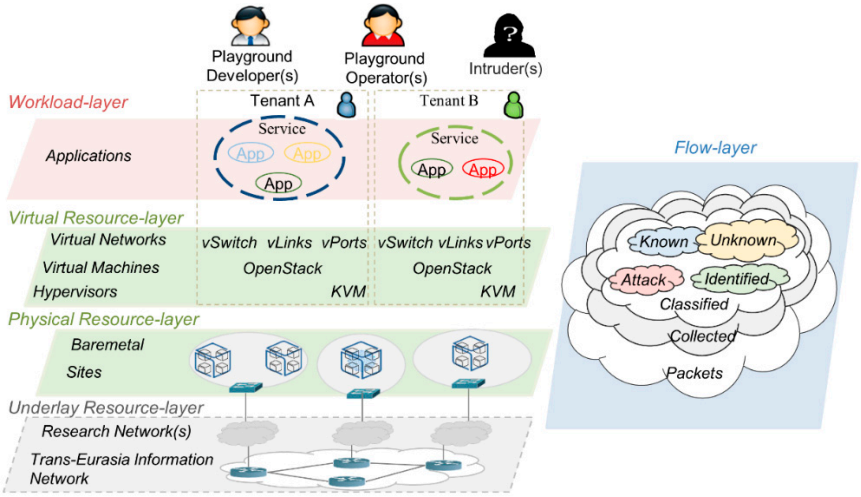


Figure 2. Identified multiple layers of visibilities for SmartX multi-view visibility framework (MVF) that should be integrated through flow-layer.

2.2.1. Network Packet-Precise Collection

In computer networks, traffic measurement is the process of measuring the amount and type of traffic on a particular network [14]. At the playground, each SmartX box has a number of dedicated networking planes for carrying different types of network traffic such as control traffic, data traffic, etc. For flexibility, playground is kept open but can be misused by either playground users or outside

intruders to exploit playground resources. Therefore, flow-centric visibility should inspect each packet that is coming in or going out from the playground by using the light-weight approach. Also, visibility agent of SmartX box for network packet measurement should be able to relaunch itself in the case of system restart or failure.

2.2.2. Visibility Data Aggregation and Tagging

Data aggregation is a process in which information is expressed in summary form for purposes such as statistical analysis and data reduction [15]. Visibility data aggregation goal is to gather particular information about individual groups based on specific variables and cater missing data issues when data is coming from distributed geographical locations. Visibility data should be aggregated for all data instances over a specified time period and filter out unnecessary data to overcome any additional metrics burden. Raw visibility data should be kept at DataLake [16] for analysis and aggregated data should be sent to the next component for further processing.

SmartX MVF is designed to offer unified monitoring by integrating multiple layers of visibilities together. As of now, due to the diverse nature of workloads, workload-layer visibility is missing from the SmartX MVF. As a workaround, visibility data tagging should be supported to incorporate additional information from both playground operators and developers into collected data for catering missing information issues. Data tagging allows playground users to organize information efficiently by associating pieces of information with tags. Examples of this data are playground user tenant and application information.

2.2.3. Multi-Layer Visibility Data Integration

Due to the growth in collected metrics from multiple layers of visibilities, the amount of visibility data generated is huge. Data integration is the process of combining data from different sources into a single unified view [17]. In typical scenarios of data integration, a user sends a request to the server for the data. The server gets the required data from internal or external datastores, consolidates it into a single cohesive data set, and sends it back to the user for use. Another key benefit of data integration is that it produces a summarized output that is more suitable for subsequent analysis. Flow-centric visibility requires scalable tools to analyze and integrate metrics from multiple layers together. Identification of key metrics that can facilitate the linking of these multiple layers is also required.

2.2.4. Network Packets Flow Classification

Network packets flow classification is a process that categorizes computer network traffic according to various parameters (e.g., based on protocol or port number) into a number of traffic classes such as identified and attack [18]. Then each traffic class can be treated differently to differentiate the services implied for the playground user. Network packet flow classification for playground should offer sufficient information from various dimensions such as underlay resources, physical resources, virtual resources, etc.

2.2.5. Visibility Data Visualization

Classified visibility data should be accessible to playground users through APIs and support interactive visualization for further exploration of visibility data. Visualization should be innovative to offer a single unified view for SDN-enabled multisite clouds by covering various physical and virtual resource components and their associated flows of OF@TEIN playground.

2.3. Related Work

In relation to the work presented in this paper, we describe the first group of monitoring approaches designed for the SDN and cloud environments. In the last few years, we have witnessed a growing interest in monitoring cloud resources. Authors in [19] discussed the key properties of a cloud monitoring system and identified open issues and future directions in the field of cloud monitoring. Nagios

core [20] is an integral monitoring tool, capable of monitoring systems, network, and infrastructure. Although Nagios was initially perceived outside the cloud, its most current versions can be easily installed in a cloud environment and it is capable of monitoring both physical and virtual resources. Global monitoring system (GMonE) is a general-purpose monitoring approach that covers the different aspects of the cloud, as the main cloud service models, client-side or service provider-side, and virtual or physical side monitoring [21]. Flexible monitoring solution at the edge (FMonE) is proposed to allow monitoring workflows that conform to the specific demands of edge computing systems [22]. Distributed architecture for resource management and monitoring in clouds (DARGOS) is a decentralized cloud monitoring tool for multitenant clouds and designed to efficiently disseminate monitoring information providing aggregate and filter functionalities [23]. To cover diverse network conditions, robust and agile defense (RAD) presented a reactive system for monitoring failures, anomalies, and attacks for high availability and reliability [24]. New Relic [25] is a famous commercial software as a service (SaaS) solution that can be deployed to monitor data in a cloud. Likewise, Dynatrace [26] also commercializes a solution for application performance monitoring, focusing on application monitoring inside a platform as a service (PaaS) cloud. PayLess also proposes a low-cost SDN monitoring framework by utilizing Floodlight controller's API [27]. For OF@TEIN Playground, tapping-based flow-layer visibility is proposed as a collection tool in [28]. Finally, improving network measurements in terms of accuracy, security, and tight time-scales are approached in [29–31].

Several works have been published to provide flow classification and tagging mechanisms. Authors studied a comparison of flow classification and tagging methods in the distributed visibility point (i.e., single OpenFlow switch), which focuses on improving localized flow handling in that point [32]. MultiClassifier combines deep packet inspection (DPI) and machine learning (ML) to quickly classify the collected flows based on application information with an acceptable accuracy rate [33]. Authors studied a number of clustering algorithms and applied them to network traffic classification with a different set of traffic characteristics (e.g., packet size, byte counts, inter-arrival, and connection duration) for each cluster [34]. Similarly, other works have been presented to improve the network traffic classification such as the introduction of flows statistical properties and supervised traffic classification [18,35,36]. Most of these works mainly focus on flow clustering or classification by using machine-learning techniques and lack comprehensive details for lightweight packet-precise visibility collection and flexible integration of visibility data that is collected from multiple measurement points of the system and network infrastructure.

3. SmartX MVF with Flow-Centric Visibility: Design and Implementation

In this section, we explain the design approaches of flow-centric visibility to satisfy the requirements that are listed in Section 2. The general architecture of the SmartX MVF with flow-centric visibility is depicted in Figure 3. SmartX MVF previously leveraged a minimalist flow-centric visibility approach proposed in [28]. That approach utilized tapping-based flow-layer visibility for collection. But it lacked the support in capturing network packets from SmartX boxes because each network packet is captured and sent to user-space (system memory allocated to run applications) for the processing. This work opts a lightweight approach to overcome the overhead of capturing and processing all the packets in user-space. Also, in a previous approach, visibility integration support was limited because only a data plane network traffic was considered, while the current approach is extensive in terms of multi-layer visibility integration. Moreover, visualization support with the previous approach was limited and desktop-oriented while work presented in this paper offers interactive web-based visualization support. By leveraging a number of existing open-source technologies, flow-centric visibility was implemented for the SDN-enabled multi-site cloud playground infrastructure particularities. We explain here each of the main components of the proposed solution.

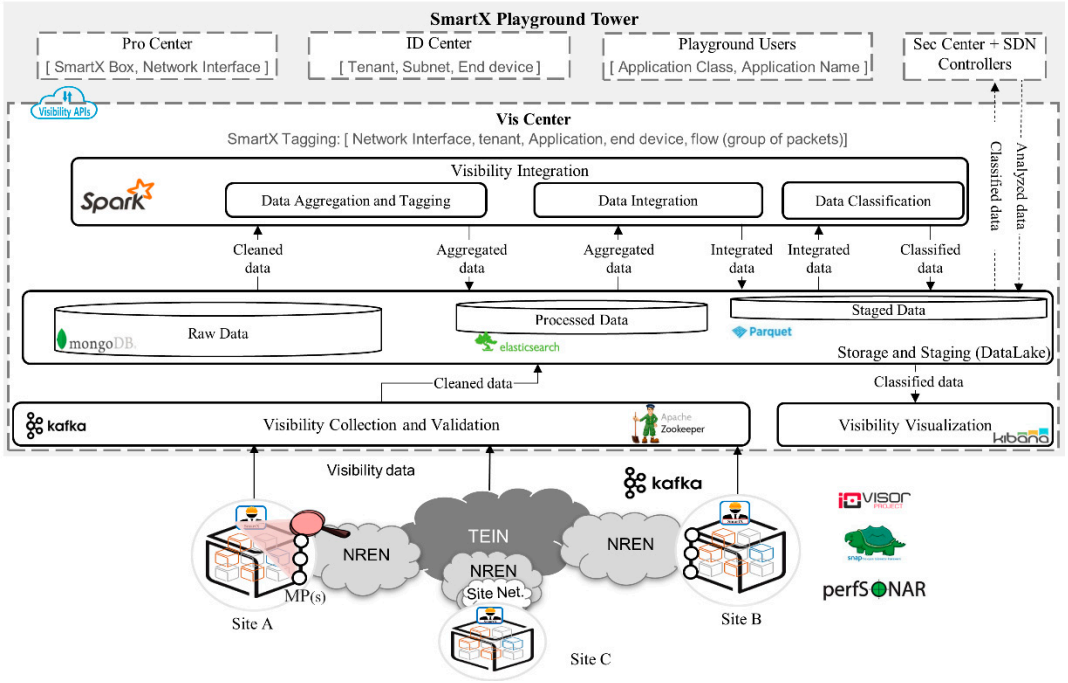


Figure 3. The overall architecture of SmartX MVF with the extension of flow-centric visibility.

3.1. Network Packet-Precise Collection

As mentioned in the requirements section, we required a special visibility agent that should collect the network packets from each network interface of SmartX box. IO Visor [37] is an open-source project designed to accelerate the innovation, development, and sharing of virtualized kernel input/output (I/O) services for networking, security, and tracing. It extends the networking capabilities based on extended Berkeley packet filter (eBPF) [38], which exists in the Linux upstream kernel version 4.x or later. eBPF is a small virtual machine that runs programs inserted from the user-space and attached to specific hooks in the kernel. It is used on Linux to filter packets and avoid expensive copies to user-space, for instance with tcpdump. IO Visor extends the capability by providing a BPF compiler collection (BCC) tool for executing eBPF instruction from other high-level languages, such as Python. By executing IO visor-based packet tracing, we can either trace the entire packet payload or check only specific packet headers by leveraging the eBPF feature.

By employing eBPF and IO Visor, we developed a visibility agent that collects information about each packet with a small number of CPU cycles because it directly accesses raw packets, copied from NIC to receive ring buffer. Considering SmartX box network interface types, we created two functions: ip_filter and vxlan_filter. Ip_filter collects packet header information such as five-tuples (IP source, IP destination, protocol, TCP/UDP source port, TCP/UDP destination port), TCP window size, and associated number of bytes. Vxlan_filter collects packets header information such as nine-tuples (host IP source, host IP destination, host protocol, guest IP source, guest IP destination, VLAN ID, guest protocol, guest TCP/UDP source port, guest TCP/UDP destination port), guest TCP window size, and associated number of bytes. The overall design of visibility agent is shown in Figure 4.

Before IO visor-based packet tracing and collection can be enabled in the SmartX box, the kernel should be upgraded to the kernel version 4.4.x and BCC tools/libraries should be installed. To ensure the automated recovery of visibility agent, it is created as a Linux system service. Our implementation of IO visor-based tracing code collects packet data from all the network interfaces of SmartX box by attaching our user-space program into Linux networking socket and then starts filtering packet data based on the filter definition (e.g., 0x0800 is IP packet) inside our program. Kernel networking I/O sends matched packets in bytecode to a user-space program. The second implementation is a user-space tracing program written in Python. The program is hooked into a Linux networking socket through

python socket programming. It acquires all raw packets from that socket and translates them into byte arrays. Finally by scanning, it extracts five-tuples or nine-tuples information from the whole byte array. A timestamp field is added to the extracted header fields of the packet and sent to DataLake by using the data delivery method specified at the visibility agent startup.

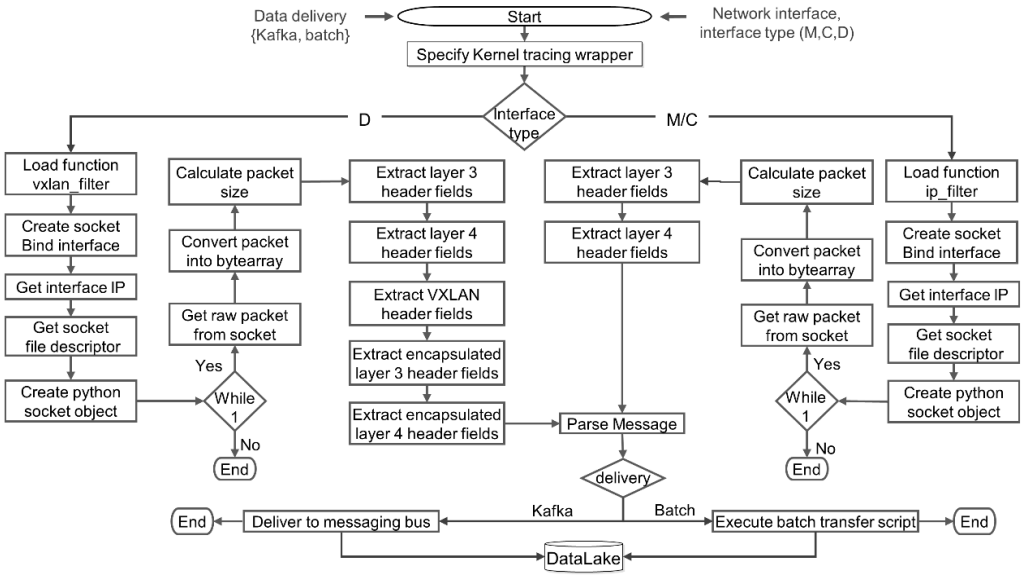


Figure 4. Design of network packets flow collection component.

3.2. Visibility Data Aggregation and Tagging

To satisfy visibility data aggregation and tagging requirements, flow-centric visibility has a special component. Under the title of SmartX tagging, additional information is collected from playground users and system generated information is extracted from pro and ID centers. Figure 5 shows the various tags that are acquired and incorporated into visibility data. Visibility data aggregation requirement is also facilitated by this component. Visibility data is aggregated based on the statistical values at five-minute intervals by using average function against the key identifiers such as SmartX box ID. In addition, visibility data aggregation also helps in overcoming the visibility data processing dilemma of distributed playground resources locality and data timestamp issues. For systematic processing of visibility data, SmartX MVF employs the Apache Spark analytics engine [39]. Core advantages of Spark are its speed of operation, native support for selected datastores of DataLake, and extensive API support.

SmartX Box NIC tags

SmartX Box ID	SmartXBoxCHULA
SmartX Box Name	SmartX-Box-CHULA
Management Interface	eth0
Control Interface	eth1
Data Interface	eth2
Active	Y

Application class and name tags

App ID	21
App Class ID	8
App Class name	Database
App Name	Memcached

User provided Application tags

Protocol	6
App ID	21
App port	11211
Tenant Name	chula

OpenStack tenant tags

Tenant ID	e51505e035de4b33b6c6321a200c9eba
Tenant Name	chula
VLAN ID	101
Data subnet	192.168.1.0/24

OpenStack VM instance tags

Tenant ID	e51505e035de4b33b6c6321a200c9eba
Tenant Name	chula
SmartX Box	SmartX-Box-GIST1
VM ID	8270603f-0a01-4c52-939a-e76c0ea625a5
VM Name	gist1-vm1-ex1
Floating IP address	x.x.x.x
Data plane IP address	192.168.115.10

Figure 5. Automatically acquired SmartX tags from the playground tower and playground users.

The design of visibility data aggregation and tagging component is shown in Figure 6. There are four main functions that facilitate flexible processing of multi-layer visibility data. An aggregate

underlay data function loads visibility data from the MongoDB datastore [13] of DataLake. Query and filter specify conditions for matching data and pushes the queries to the MongoDB to bring only matching data in the Spark executor program. On the fetched data, grouping function average is applied to reduce the volume of data. Before saving the resultant data frame to the Elasticsearch [13] and Parquet [40] datastores, a timestamp field is added. A dataset is a distributed collection of data and a data frame is a dataset organized into named columns. Elasticsearch datastore retains processed visibility data for long-term needs and Parquet datastore keeps the lastly processed visibility data and is consumed by a subsequent component of visibility integration stage. Besides raw data filtering and aggregation steps, aggregate physical data function also incorporates SmartX tags to the system level visibility metrics. Aggregate packets data function deals with flow-layer visibility data from each SmartX box. A SmartX box has two data files. The first file contains management and control planes network packets and the second file contains data plane network packets. Aggregate packet data function gets all the files from the disk and creates a combined data frame. SmartX tags are then incorporated into this data according to the matching identifiers in the data frame. Additional statistical fields are extracted (e.g., min/max/avg/total data bytes) for more detailed information about playground network traffic.

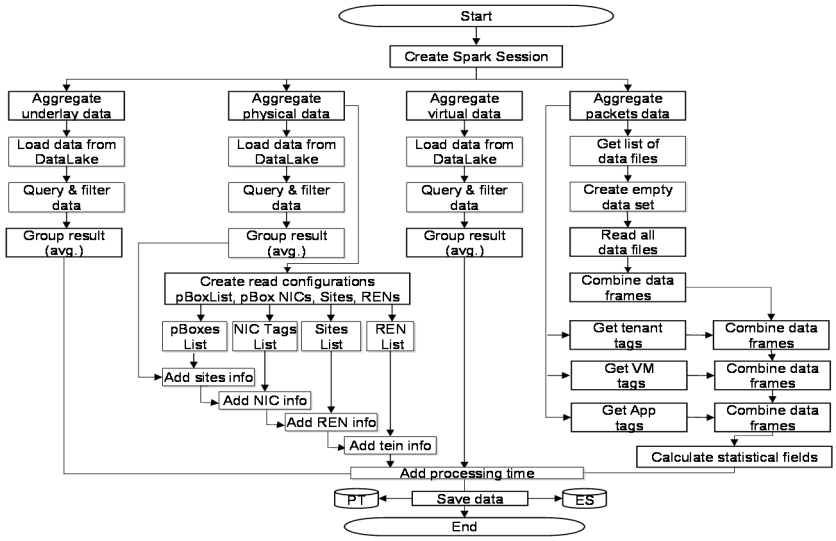


Figure 6. Design of visibility data aggregation and tagging component.

Apache Spark version 2.2.0 and Java version 1.8.0_201 are installed on vis center. SmartX tags acquisition is implemented by shell scripting and OpenStack cloud platform python APIs and execute at various centers of playground tower. Visibility data tagging and aggregation component is developed using the Scala programming language. By using simple build tool (sbt) [41], a thin jar file is created and deployed at the vis center. To repeatedly execute this component, we developed a shell script and following command specifies execution parameters when running this component.

$SPARK_HOME/bin/spark-submit –class smartx.multiview.flowcentric.Main –master local[*] –driver-memory 24g multi-view-flowcentric-tag-aggregate_2.11-0.1.jar "x.x.x.x" &&

This command utilizes spark-submit tool to execute data processing jobs on the local system with all available CPU cores and 24G of memory. Other important parameters are the class that specifies the name of the main Scala class and the name of the executable jar file.

3.3. Multi-Layer Visibility Data Integration

Visibility data integration is one of the most critical components of flow-centric visibility designed to satisfy multi-layer visibilities unification requirements. To integrate multiple layers, first, we identify key identifiers for each collection of visibility data. A list of selected key identifiers is given in Figure 7. Design of the visibility data integration component is shown in Figure 8. An integrate multi-layer

visibility function creates five data frames and gets aggregated visibility data from the Parquet datastore of DataLake. By selecting the flow-layer data frame as a starting point, we integrate it with all the other data frames. A lookup is created to avoid identical column names exceptions. At the end of execution, only two data frames are left and stored in both Parquet and Elasticsearch datastores.

Visibility data integration leverages Apache Spark SQL. Spark SQL is a Spark module for structured data processing. Unlike the basic Spark resilient distributed dataset (RDD) API, the interfaces provided by Spark SQL provide Spark with more information about the structure of both the data and the computation being performed. There are several ways to interact with Spark SQL, including dataset API. For visibility data integration, a separate project is created and implemented in Scala. A jar file is created and deployed at vis center. By using Spark-submit this jar file is executed at five-minute intervals.

Visibility Data	Identifier#1	Identifier#2	Identifier#3
Underlay Resource-layer	Source Box ID		
Physical Resource-layer	SmartX Box ID		
Virtual Resource-layer	SmartX Box ID	VM ID	IP Address
Flow-layer	MP Box	VLAN ID	IP Address's
SmartX Box NIC Tags	SmartX Box ID		
Tenant Tags	Tenant Name	VLAN ID	
VM Instance Tags	SmartX Box ID	Tenant Name	VM ID
Application Class Tags	App ID	App Class ID	
Application Tags	App ID	Tenant Name	

Figure 7. Selected list of key identifiers for visibility data integration.

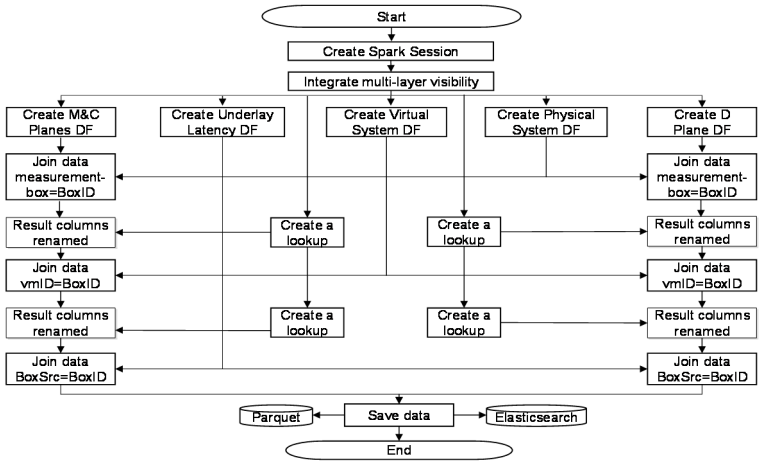


Figure 8. Design of multi-layer visibility data integration component.

3.4. Network Packets Flow Classification

As visibility data is already integrated by following flow-layer characteristics, grouping of processed data into various classes becomes a relatively simplified job. For the classification of network packets flows, in this initial version of flow-centric visibility, we define four classes (attack, identified, known, and unknown). Attack class contains network packet flows that belong to previously attacked IP addresses. These IP's can be obtained from the internet, as known attack databases. Identified class contains network packets flow from playground user tenant and already notified applications to the playground operators via SmartX tagging mechanism. Known class contains network packets flow from the playground user tenant, but the playground operator does not know the application. Unknown class contains network packets flow from unknown IP addresses and unknown applications (i.e., both user tenant and application are not known). After this initial classification, further analysis can be performed at the sec center by considering other metrics.

The design of network packets flow classification component is shown in Figure 9. Classify network packet flows is the main function that loads already integrated visibility data into a Spark data frame. A pre-downloaded data file with attack IP addresses is loaded, and contents are validated before further execution. After data validation, both integrated visibility data frame and attacker IP addresses data frame are joined together on source and destination IP addresses. The resultant data frame is stored to the Elasticsearch and Parquet datastores. Find identified flows analyzes the tenant name and application name fields in the integrated data frame, if both the fields have values and these data records are not in the attacked flows data frame, then it is classified into the identified class. Find known flows then analyzes tenant name and application name fields and if the tenant name is present and application name is absent, then it is classified into the known class. Find unknown flows analyzes tenant name and application name fields and if both tenant name and application name are absent, then it is classified into the unknown class.

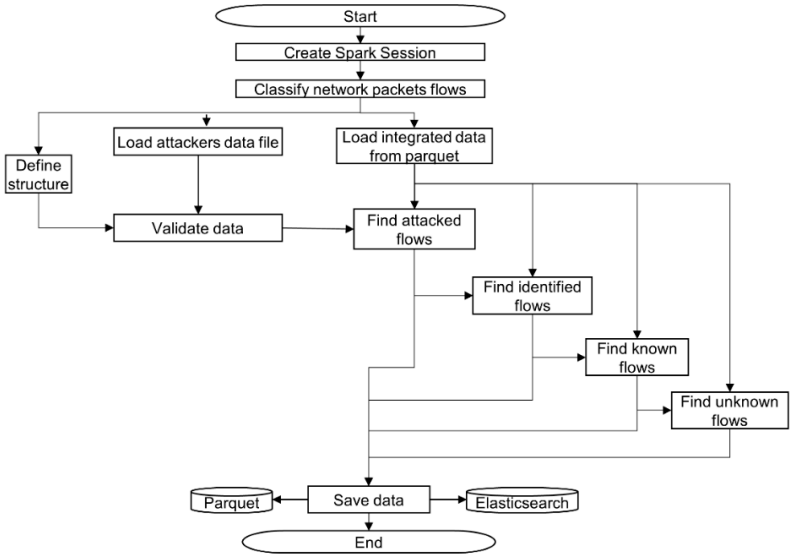


Figure 9. Design of network packets flow classification component.

Visibility classification also leverages Apache Spark SQL and implemented as a separate project in Scala. Again, a jar file is created and deployed at the vis center. By using Spark-submit, this jar file is executed at regular time intervals of five minutes.

3.5. Visibility Data Visualization

To address the playground multi-layer visibility data visualization requirement, we design multi-belt onion-ring style visualization as shown in Figure 10. Primarily there are five main belts for each layer of visibility and then they are further divided into three sub-belts to allow incorporation of additional information for display in a particular belt. Also, the color-coded areas that are separated with white lines correspond to distributed sites that host physical or virtual resources.

In order to realize the multi-belt onion-ring visualization, we leverage open-source visualization library called psd3. That is, psd3 is primarily based on D3.js and supports multi-level pie charts. In order to visualize flow-centric visibility data, we choose Kibana [42] visualization software due to available support for a large number of feature-rich graphs. Visibility data is accessible via REST API. For example, following call fetches visibility data in JSON format from identified class for demo tenant.

curl -X GET "x.x.x.x:9200/dp-classified-identify-1/_search?size=5&pretty=true" -H 'Content-Type: application/json' -d '{"query" : {"term" : { "tenantname" : "demo" }}}'

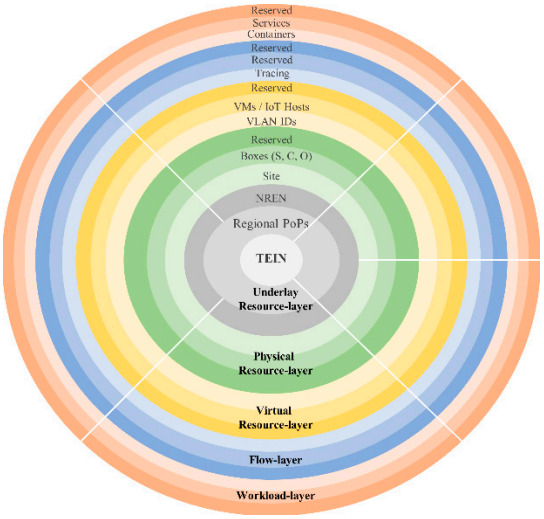


Figure 10. Multi-belt onion-ring visualization to show multi-layer visibility data.

4. SmartX MVF with Flow-Centric Visibility: Verification

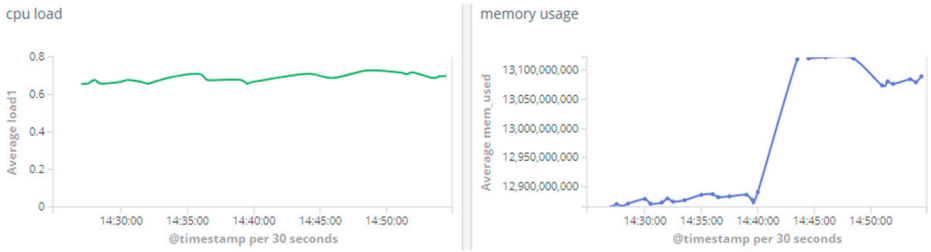
This section verifies proposed extension of SmartX MVF by demonstrating that the implemented prototype meets the above-listed requirements. A prototype of the proposed solution can be found on GitHub (https://github.com/K-OpenNet/OpenStack-MultiView.git). To setup the verification environment, we have considered reference architecture shown in Figure 1. We envision a cloud architecture where a user had resources in different geographical regions and were inter-connected by overlay-based SDN networks. These resources can be physical or virtual servers and run applications from users in different locations. Since regions are distinct from each other, there are bandwidth and latency restrictions between them. A central playground tower is located at the highest level and all the other regions communicate with it to push the visibility data. From the playground tower, operators need to monitor and make decisions about cloud infrastructure. For example, thanks to the visibility data, we can detect overloaded servers in a given region or a server that went down. There were several issues that should be validated from the viewpoint of overhead and usability through detailed experiments for this type of environment. The first issue was visibility overhead which is covered in Section 4.2. We needed to verify a successful classification of network packets flow into various classes with visualization support for assisting secured operation in such environments and covered in Section 4.3. Finally, we have a general discussion about verification results and future directions in Section 4.4.

4.1. Verification Setup

An initial-stage deployment of the proposed solution utilized server-based hardware for vis center with the following specs: Intel(R) Xeon(R) CPU E5-2690 v2@3.00GHz, memory DDR3 12x8GB, HDD 5.5TB, and four network interfaces of 1Gbit/s. We mainly utilized an IBM M4 server for our SmartX box. In general, the 2U server has 12 CPU cores with 2.30 GHz Intel processors, 32 GB memory, and 300 GB of hard-disk with RAID 0 array configuration. It has one dedicated physical interface for IPMI-based remote access management through CLI (command-line interface) and web UI (user interface). Also, it has four onboard 1G network interfaces. Both SmartX box and vis center are utilizing Ubuntu 16.04.4 LTS release with a minimal kernel version of 4.4.0. Each playground developer is assigned a dedicated tenant to perform various networking experiments. By choosing the client-server virtual machine deployment model, they generate various types of TCP/UDP traffic during their experiments. For the verification of the proposed solution, we leveraged existing developer tenants and running applications to collect visibility data. We utilized six tenants with a total of twenty VMs within ten SmartX boxes. Iperf version three is used for network traffic generation.

4.2. Visibility Overhead Results

The visibility agent for network packets collection should be lightweight and nonintrusive. The aim of this first experiment is to ensure the requirement of non-intrusiveness of visibility agent. We monitored the performance of SmartX box in two situations: without execution of the visibility agent and with the execution of visibility agent. We recorded the average CPU load and memory utilization before and after the execution of visibility agent as shown in Figure 11. On the x-axis we have time and the y-axis shows resource usage. Visibility agent execution is started after 14:40:00. The CPU load almost remains the same at 0.7 as shown in Figure 11a. Memory utilization is initially increased by 217 MB and later stayed around 177 MB as shown in Figure 11b. These graphs indicate that visibility agent is lightweight and incurs low impact on SmartX box performance, even when processing network packets from multiple network interfaces.



(**a**) (**b**)

Figure 11. Visibility data collection overhead for packet-precise network flows. (**a**) CPU load and (**b**) memory utilization.

To verify the persistent measurement and collection of data via the proposed solution, as shown in Figure 12a, we reported the total number of data instances collected per day from playground distributed sites to the DataLake in one month. The highest number of data instances, around 47 million, were recorded on the twenty-ninth and the lowest number of data instances, just above 20 million, were recorded on the eighteenth. We effectually collected, validated, and stored almost 25 million data instances per day. In addition, we calculated the number of data instances that are collected from each layer of the visibilities for an hour (underlay resource layer = 54, physical resource layer = 420, virtual resource layer = 840, and flow layer = 52 million). In order to measure the execution time of implemented visibility integration component, we simulated it by increasing the number of network packets. In this experiment, we started the Spark analytics engine in standalone mode with a different number of CPU cores and memory allocation as shown in Figure 12b. We begin the tests using 25,726 network packets with a total size of 3.4 MB, where execution took a total of 3.7 s over 4 CPU cores and 8 GB of memory to aggregate network packets and tag them. Next, we increased the number of network packets (number = 450,000 and dataset size on disk = 61 MB). Now, execution took 5.8 s. Also, a similar trend was observed accordingly to the increased number of network packets (number = 5 million and dataset size on disk = 630.30 MB), where 7.4 s were taken by the execution. When we further increase the number of packets (number = 10 million and dataset size on disk = 1.3 GB), 13.5 s was needed for completing the execution. Based on these results, we can see execution efficiency improved with the increase in number of assigned resources to the analytics engine.

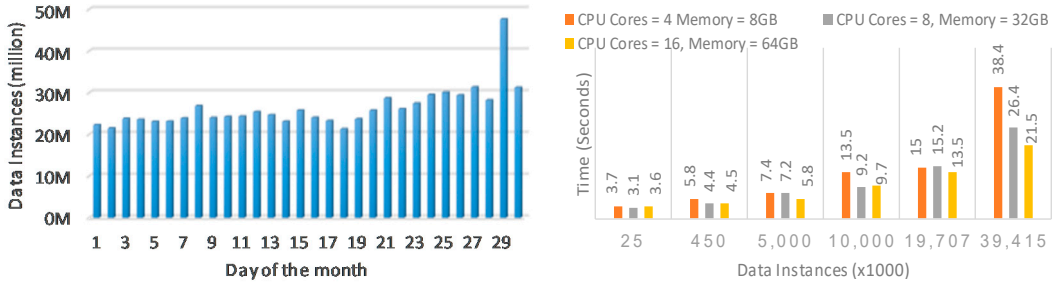


Figure 12. Flow-centric visibility verification results. (**a**) The number of visibility data instances collected for a month and (**b**) execution time of visibility integration component.

4.3. Network Packets Flow Classification and Visualization Results

Integrated network packets flow classification is verified by conducting various experiments. For this purpose, we selected three tenants (i.e., demo, ncku, and uscm) to generate network traffic by using various TCP/UDP application ports. As shown in Table 1, four cases are verified by generating traffic that is associated with various classes to ensure implemented non-learning-based classification is working as expected. We understand that assuming the server port always identifies an application is not fully correct. However, we assume the ports used in this work belong to the expected applications. Furthermore, we also do not consider traffic of the selected applications on other than the standard server ports e.g., HTTP traffic is on port 80 but not on port 81.

Table 1. Verification cases of network packets flow classification for three different tenants.

	Demo	**Uscm**	**Ncku**
Case I	Identified (9092)	Identified (11,211)	Identified (6343)
Case II	Identified (53)	Attack (27,017)	Identified (161)
Case III	Attack (443)	Identified (8086)	Known (1250)
Case IV	Known (8000)	Known (7000)	Attack (11,211)

Figure 13a shows the result for network packets flows that are classified as identified. In this pie chart, the innermost ring shows the TEIN PoPs (point-of-presence) (TEINHK and TEINSG) from where network packets have passed. The second ring shows the site names (e.g., VN-HUST) from where identified network packets flows are collected and third ring shows the tenant names (e.g., demo). The fourth ring shows the OpenStack VM instance names (e.g., USCM-myren-vm) from where network packets flow is initiated or terminated. The outermost ring shows the application names (e.g., InfluxDB). We can see all the applications, which were listed as identified in Table 1, are also classified as identified by the system and properly tagged with corresponding tenant and application names. For example, port 9092 corresponded to the Kafka and belonged to the demo tenant, and port 8086 corresponded to the InfluxDB and belonged to the uscm tenant. Figure 13b shows the result for network packet flow that were classified as known. In this pie chart, the innermost ring shows the TEIN PoPs. The second ring shows the site names and third ring shows the tenant names. The outermost ring shows the application ports (7000, 8000, and 1250). Please note these snapshots were taken from the real system and can easily incorporate further rings to display more information as well as show the tooltip with corresponding and associated ring's actual values from the data. Figure 13c shows the result for network packets flows that are classified as unknown. In this pie chart, the innermost ring shows the TEIN PoPs. The second ring shows the SmartX boxes names (e.g., SmartX-Box-GIST1) from where unknown network packets were collected and the third ring shows the IP addresses (e.g., 6.2.0.120). The outermost ring shows the application port numbers (e.g., 1285) from where network packets are originated. Interestingly, we did not generate any traffic for unknown class and still got packets classified into this class. Upon further investigation, ONOS SDN controller application

was found to be using this port number and IP addresses. Figure 13d shows the result for network packets flows that are classified as an attack. In this pie chart, the innermost ring shows the TEIN PoPs. The second ring shows the national research and education network names (e.g., TWAREN) and third ring shows the tenant names. The outermost ring shows the OpenStack VM instance IP addresses (e.g., 192.168.1.124). We can see all the IP addresses, which were listed as the attacker, were also classified as attacked by the system. For example, before performing experiments for case II, we have added 192.168.116.6 and 192.168.116.35 in attackers list and they were successfully detected as attack class.

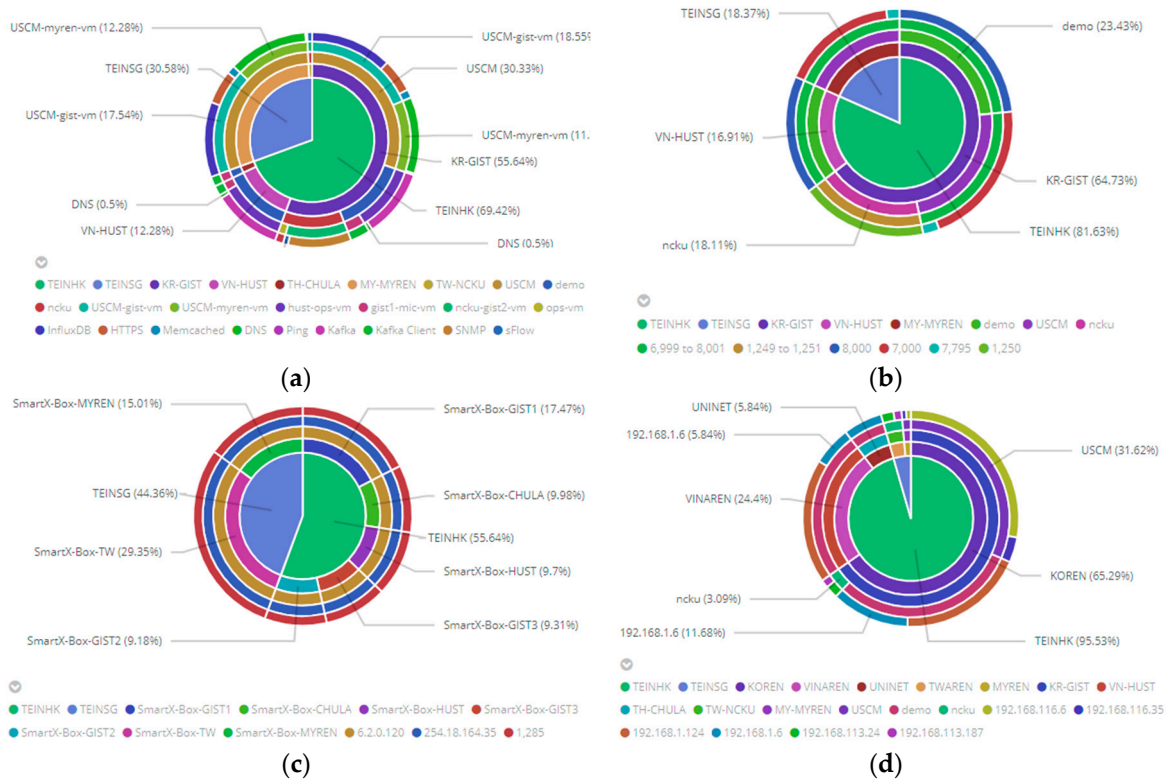


Figure 13. Network packets flow classification results. (**a**) Identified class, (**b**) known class, (**c**) unknown class and (**d**) attack class.

Additionally, for faster troubleshooting of OF@TEIN playground, we can generate various visualizations from aggregated, integrated, and classified visibility data and arrange them into interactive dashboards. After integration, a single data instance has eighty-three metrics available for further analysis while flow-centric visibility that was proposed in [28] had seven metrics available for further analysis. To show some of these metrics, we have created a dashboard that presents four charts as shown in Figure 14. These charts were automatically refreshed after every thirty seconds and express last thirty minutes' visibility data. The first chart on top-left shows the top applications with the highest amount of data bytes. The next chart shows the five-tuple-based flow information with corresponding application. The third chart shows top flows statistics such as the number of packets and data bytes. The last chart shows system resources (CPU, memory, and network interfaces) utilization per site. In addition, we have uploaded a demonstration video of our preliminary version of onion-ring visualization at https://www.youtube.com/watch?v=iJ7lah4v1y8.

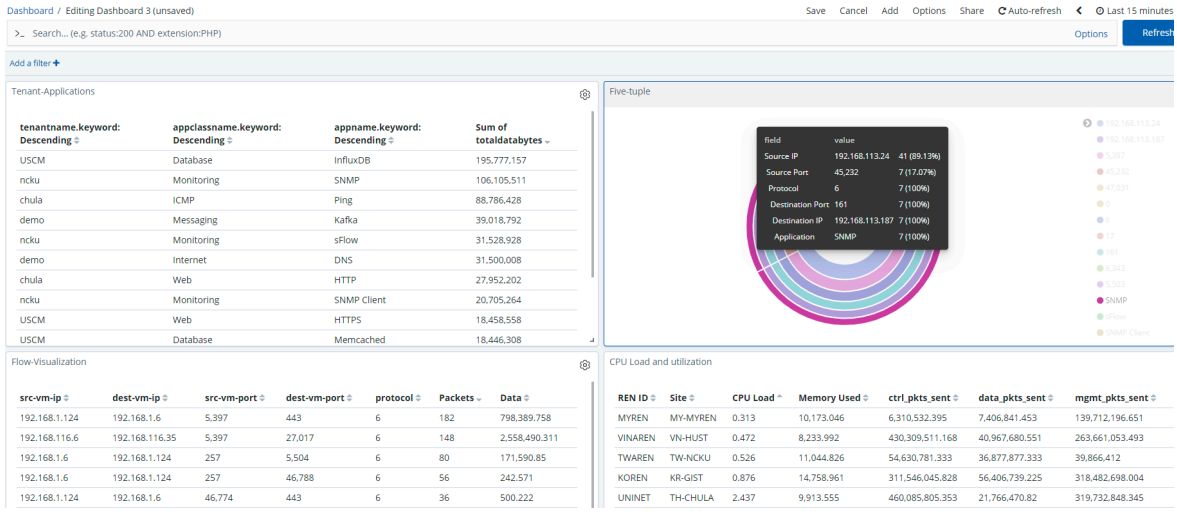


Figure 14. Kibana dashboard to show visualization results from classified visibility data.

4.4. Discussion

To prove the viability of our approaches, in this section, we have verified our design and prototype implementation. Flow-centric visibility leverages popular open-source tools such as IO Visor, Apache Spark, Parquet, and Kibana. The features of the used tools may limit the proposed solution. IO visor-based network packets measurements can be augmented by other tools (e.g., sFlow [43]) to collect the sampled packets flows in resource-constrained sites of the playground. The proposed solution does not solve all the operational problems in the playground, but it can expand with other tools or methods to support new features. For example, comprehensive security strategy involves protecting the system and network from external and internal misuse and upon detection of anomalies using flow-centric visibility, operators can baseline normal behaviors and trigger for suspicious events [44,45]. Integrating further tools like IDS and SDN controllers in flow-centric visibility workflow is a topic for upcoming study. Moreover, as a further study, it will be vital to support flow-centric visibility with learning-based approaches for automated classification of network packets flow and detecting various system and network operational issues.

5. Conclusions

In this study, we proposed a comprehensive extension to visibility and visualization framework called SmartX MVF for flow-centric visibilities (e.g., aggregated packets flow, integrated packets flow, or classified packets flow) on SDN-enabled multisite clouds. Such a comprehensive system has not been considered by previous studies on unified visibility. After flow-centric visibility extension, SmartX MVF offers fine-grained flow information for physical-virtual resources system performance and network utilization. The lightweight network packet-precise flows collection component in the proposed solution first collects network packets and then send them to the vis center. The data aggregation and tagging component generates summarized output by applying time-based data aggregation and adds extra information to the collected visibility data. Aggregated data are integrated together by using key identifiers from multiple layers of visibilities via the data integration component. The data classification component classifies the integrated visibility data into four classes. Finally, this classified visibility data is readily accessible via APIs and interactive dashboards that are created in Kibana. We instantiated flow-centric visibility in the playground that is built with SDN controllers (OpenDayLight [12] and ONOS [46]) and multisite OpenStack cloud to investigate the properties of the proposed solution. We then conducted a series of experiments for system verification. Our experimentation results show that our approaches are both feasible and practical, and can successfully augment an existing SDN-enabled multisite cloud infrastructure in terms of robustness and agility.

Author Contributions: This work is completed by M.U., M.A.R., and J.K., M.U. designed and developed the proposed system in this work. M.A.R. helped to implement and experiment proposed system. M.U. and M.A.R. focused on writing the paper. J.K. guided this whole project as a corresponding author and reviewed this manuscript.

Funding: This research was supported by Agency Defense Development funded project "Study on QoS method in tactical communications network" (UD170050ED).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kim, H.; Feamster, N. Improving network management with software defined networking. *IEEE Commun. Mag.* **2013**, *51*, 114–119. [CrossRef]
2. Hu, F.; Hao, Q.; Bao, K. A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 2181–2206. [CrossRef]
3. Nunes, B.A.A.; Mendonca, M.; Nguyen, X.N.; Obraczka, K.; Turletti, T. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 1617–1634. [CrossRef]
4. Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A.D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.A.; Rabkin, A.; Stoica, I.; Zaharia, M. A view of cloud computing. *Commun. ACM* **2010**, *53*, 50–58. [CrossRef]
5. Zhang, Q.; Cheng, L.; Boutaba, R. Cloud computing: State-of-the-art and research challenges. *J. Int. Serv. Appl.* **2010**, *1*, 7–18. [CrossRef]
6. Berman, M.; Chase, J.S.; Landweber, L.; Nakao, A.; Ott, M.; Raychaudhuri, D.; Ricci, R.; Seskar, I. GENI: A federated testbed for innovative network experiments. *Comput. Netw.* **2014**, *61*, 5–23. [CrossRef]
7. SmartFIRE-NITlab-Network. Available online: https://nitlab.inf.uth.gr/NITlab/index.php/projects/40-projects/current/444-smartfire.html (accessed on 30 March 2019).
8. Risdianto, A.C.; Kim, J. Prototyping Media Distribution Experiments over OF@TEIN SDN-enabled Testbed. *Proc. Asia-Pac. Adv. Netw.* **2014**, *38*, 12–18. [CrossRef]
9. Usman, M.; Risdianto, A.C.; Han, J.; Kang, M.; Kim, J. SmartX multiview visibility framework leveraging open-source software for SDN-cloud playground. In Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft), Bologna, Italy, 3–7 July 2017; pp. 1–4.
10. Risdianto, A.C.; Shin, J.; Kim, J. Building and Operating Distributed SDN-Cloud Testbed with Hyper-Convergent SmartX Boxes. In *Cloud Computing*; Zhang, Y., Peng, L., Youn, C.H., Eds.; Springer International Publishing: Cham, Switzerland, 2016; Volume 167, pp. 224–233.
11. Sefraoui, O.; Aissaoui, M.; Eleuldj, M. OpenStack: Toward an Open-source Solution for Cloud Computing. *Int. J. Comput. Appl.* **2012**, *55*, 38–42. [CrossRef]
12. Risdianto, A.C.; Kim, N.L.; Shin, J.; Bae, J.; Usman, M.; Ling, T.C.; Panwaree, P.; Thet, P.M.; Aswakul, C.; Thanh, N.H.; et al. OF@TEIN: A Community Effort towards Open/Shared SDN-Cloud Virtual Playground. *Proc. Asia-Pac. Adv. Netw.* **2015**, *40*, 22–28. [CrossRef]
13. Usman, M.; Risdianto, A.C.; Han, J.; Kim, J. Interactive Visualization of SDN-Enabled Multisite Cloud Playgrounds Leveraging SmartX MultiView Visibility Framework. *Comput. J.* **2018**. [CrossRef]
14. Mohan, V.R.; Reddy, Y.R.J.; Kalpana, K. Active and Passive Network Measurements: A Survey. *Int. J. Comput. Sci. Inf. Technol.* **2011**, *2*, 1372–1385.
15. Data Aggregation. Available online: https://www.ibm.com/support/knowledgecenter/en/SSBNJ7_1.4.2/dataView/Concepts/ctnpm_dv_use_data_aggreg.html (accessed on 31 March 2019).
16. DataLake. Available online: https://martinfowler.com/bliki/DataLake.html (accessed on 31 March 2019).
17. Data Integration. Available online: https://www.ibm.com/analytics/data-integration (accessed on 31 March 2019).
18. Vlăduţu, A.; Comăneci, D.; Dobre, C. Internet traffic classification based on flows' statistical properties with machine learning. *Int. J. Netw. Manag.* **2017**, *27*, e1929. [CrossRef]
19. Aceto, G.; Botta, A.; De Donato, W.; Pescapè, A. Cloud monitoring: A survey. *Comput. Netw.* **2013**, *57*, 2093–2115. [CrossRef]
20. Nagios. Available online: https://www.nagios.org/ (accessed on 31 March 2019).
21. Montes, J.; Sánchez, A.; Memishi, B.; Pérez, M.S.; Antoniu, G. GMonE: A complete approach to cloud monitoring. *Future Gener. Comput. Syst.* **2013**, *29*, 2026–2040. [CrossRef]

22. Brandón, Á.; Pérez, M.S.; Montes, J.; Sanchez, A. FMonE: A Flexible Monitoring Solution at the Edge. *Wirel. Commun. Mob. Comput.* **2018**, *2018*, 2068278. Available online: https://www.hindawi.com/journals/wcmc/2018/2068278/ (accessed on 30 March 2019). [CrossRef]
23. Povedano-Molina, J.; Lopez-Vega, J.M.; Lopez-Soler, J.M.; Corradi, A.; Foschini, L. DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant Clouds. *Future Gener. Comput. Syst.* **2013**, *29*, 2041–2056. [CrossRef]
24. Kim, M.; Park, Y.; Kotalwar, R. Robust and Agile System against Fault and Anomaly Traffic in Software Defined Networks. *Appl. Sci.* **2017**, *7*, 266. [CrossRef]
25. New Relic. Available online: https://newrelic.com/ (accessed on 31 March 2019).
26. Dynatrace. Available online: https://www.dynatrace.com/ (accessed on 31 March 2019).
27. Chowdhury, S.R.; Bari, M.F.; Ahmed, R.; Boutaba, R. PayLess: A low cost network monitoring framework for Software Defined Networks. In Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 5–9 May 2014; pp. 1–9.
28. Risdianto, A.C.; Kim, J. Flow-centric Visibility Tools for OF@TEIN OpenFlow-based SDN Testbed. In Proceedings of the 10th International Conference on Future Internet, Seoul, Korea, 8–10 June 2015; pp. 46–50.
29. Van Adrichem, N.L.; Doerr, C.; Kuipers, F.A. OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks. In Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, Poland, 5–9 May 2014; pp. 1–8.
30. Rasley, J.; Stephens, B.; Dixon, C.; Rozner, E.; Felter, W.; Agarwal, K.; Carter, J.; Fonseca, R. Planck: Millisecond-scale monitoring and control for commodity networks. In Proceedings of the 2014 ACM conference on SIGCOMM, Chicago, IL, USA, 17–22 August 2014; pp. 407–418.
31. Yoon, C.; Lee, S.; Kang, H.; Park, T.; Shin, S.; Yegneswaran, V.; Porras, P.; Gu, G. Flow Wars: Systemizing the Attack Surface and Defenses in Software-Defined Networks. *IEEE ACM Trans. Netw.* **2017**, *25*, 3514–3530. [CrossRef]
32. Farhadi, H.; Nakao, A. Application layer flow classification in SDN. In Proceedings of the 15th Asia-Pacific Network Operations and Management Symposium (APNOMS), Hiroshima, Japan, 25–27 September 2013; pp. 1–3.
33. Li, Y.; Li, J. MultiClassifier: A combination of DPI and ML for application-layer classification in SDN. In Proceedings of the 2nd International Conference on Systems and Informatics (ICSAI 2014), Shanghai, China, 15–17 November 2014; pp. 682–686.
34. McGregor, A.; Hall, M.; Lorier, P.; Brunskill, J. Flow Clustering Using Machine Learning Techniques. In *Passive and Active Network Measurement*; Barakat, C., Pratt, I., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3015, pp. 205–214.
35. Bagui, S.; Fang, X.; Kalaimannan, E.; Bagui, S.C.; Sheehan, J. Comparison of machine-learning algorithms for classification of VPN network traffic flow using time-related features. *J. Cyber Secur. Technol.* **2017**, *1*, 108–126. [CrossRef]
36. Promrit, N.; Mingkhwan, A. Traffic Flow Classification and Visualization for Network Forensic Analysis. In Proceedings of the 29th International Conference on Advanced Information Networking and Applications (2015 IEEE), Gwangiu, Korea, 24–27 March 2015; pp. 358–364.
37. Nam, T.; Kim, J. Open-source IO visor eBPF-based packet tracing on multiple network interfaces of Linux boxes. In Proceedings of the 2017 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea, 16–18 October 2017; pp. 324–326.
38. Learn eBPF Tracing: Tutorial and Examples. Available online: http://www.brendangregg.com/blog/2019-01-01/learn-ebpf-tracing.html (accessed on 31 March 2019).
39. Li, Y.; Jiang, Y.; Gu, J.; Lu, M.; Yu, M.; Armstrong, E.M.; Huang, T.; Moroni, D.; McGibbney, L.J.; Frank, G.; et al. A Cloud-Based Framework for Large-Scale Log Mining through Apache Spark and Elasticsearch. *Appl. Sci.* **2019**, *9*, 1114. [CrossRef]
40. Apache Parquet. Available online: https://parquet.apache.org/documentation/latest/ (accessed on 31 March 2019).
41. Getting Started with sbt. Available online: https://www.scala-sbt.org/1.0/docs/Getting-Started.html (accessed on 31 March2019).

42. Kibana User Guide. Available online: https://www.elastic.co/guide/en/kibana/current/introduction.html (accessed on 31 March 2019).
43. sFlow Overview. Available online: https://sflow.org/about/index.php (accessed on 31 March 2019).
44. Shin, J.S.; Usman, M.; Kim, J. Conceptual Verification of Multi-Level Visibility Points for SmartX MultiView Security. In Proceedings of the 2018 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea, 17–19 October 2018; pp. 1449–1451.
45. Usman, M.; Risdianto, A.C.; Han, J.; Kim, J. Inter-correlation of Resource-/Flow-Level Visibility for APM Over OF@TEIN SDN-Enabled Multi-site Cloud. In *Quality, Reliability, Security and Robustness in Heterogeneous Networks*; Lee, J.H., Pack, S., Eds.; Springer International Publishing: Cham, Switzerland, 2017; Volume 199, pp. 478–484.
46. Risdianto, A.C.; Tsai, P.W.; Ling, T.C.; Yang, C.S.; Kim, J. Enhanced Onos Sdn Controllers Deployment for Federated Multi-Domain Sdn-Cloud with Sd-Routing-Exchange. *Malays. J. Comput. Sci.* **2017**, *30*, 134–153. [CrossRef]