Received: 1 October 2019 | Revised: 9 June 2020 | Accepted: 10 July 2020

DOI: 10.1002/ett.4101

SPECIAL ISSUE ARTICLE

WILEY

Maintaining SmartX multi-view visibility for OF@TEIN+ distributed cloud-native edge boxes

Muhammad Ahmad Rathore[1] | **Muhammad Usman**[1,2] | **JongWon Kim**[1]

[1]School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology, Gwangju, Korea

[2]Institute of Computer Science (ICS), Masaryk University (MUNI), Brno, Czech Republic

Correspondence
JongWon Kim, School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology, 123 Cheomdangwagi-ro, Buk-gu, Gwangju 61005, South Korea.
Email:jongwon@gist.ac.kr

Funding information
Cybersecurity project, Institute of Information & Communications Technology Planning & Evaluation (IITP) grant, Korea government (MSIT), Grant/Award Number: 2017-0-00421, Cyber Security Defense Cycle Mechanism for New Security Threats); K-ONE project, Institute of Information & communications Technology Planning & Evaluation (IITP) grant, Korea government (MSIT), Grant/Award Number: 2015-0-00575, Global SDN/NFV OpenSource Software Core Module/Function Development).

Abstract

Maintaining knowledge of infrastructure has always been a key element to ensure the performance of complex distributed systems, being the first step to control quality of service, identify key performance bottlenecks, or make decisions about resource allocation and job scheduling, to name a few. In cloud-native edge boxes, edge computing leverages cloud-native applications to process/analyze some of their computing/storage own close to the location where the data is generated by a large number of heterogeneous devices. Some of the differences between this approach and more conventional architecture, like cloud are that in a distributed environment, these resources/applications have varying running conditions depending on resource availability, quality of network connection, and being geo-distributed. These aforementioned challenges in operations, however, contribute to a requirement toward identifying efficient methods for monitoring operational environment with continuity in order to maintain and sustain services. In this article, we present setup and configuration of distributed resources with cloud-native edge capabilities and having centralized connectivity in a specialized multi-site cloud testbed. Second, we propose a solution to persistently maintain multi-view (ie, resource-layer and flow layer) visibility data (monitoring) collection under varying situations and environments. Third, we present verification results on a prototype implementation and interactive visualization support. This solution allows maintaining monitoring visibility data that conform to the specific demands of cloud-native edge computing resources. We evaluate the research by maintaining visibility data of multi-site cloud at 14 research institutes for several months and we demonstrate that it fulfills the requirements we previously enumerated.

1 | INTRODUCTION

Maintaining a stable monitoring state has been a key element in ensuring the sustainable performance of complex distributed systems, as the first step in managing quality of service, detect failures, or making decisions about resource allocation, to name a few. Edge cloud[1] is a broad distribution of storage/-compute resources around geographic locations with cloud-like capabilities located at the infrastructure edge. Cloud-native application is a methodology of building

Abbreviations: MVF, SmartX multi-view visibility framework; OF@TEIN, Open Flow@ Trans-Eurasia information network

and running applications build on cloud having common characteristics.[2] In a cloud-native environment, edge computing moves applications workload from a centralized location to remote locations, thus improve latency as well as reduce the bandwidth constraints over the network and storage capacity at the center. Also, edge computing leveraged the benefits of cloud-native by keeping what is great about the cloud-native model (Microservices architectures, Containerized applications, and dynamically orchestrated), yet applying it in the harsh physical and environment of IoT devices. Aligned with Future Internet testbeds like GENI (the Global Environment for Networking Innovation), in 2017, we launched, OF@TEIN+ playground (ie, SDN-enabled multisite cloud), as an open/federated playground for future networks, proposed to further enhance and expand OF@TEIN playground.[3-5] OF@TEIN+ multisite cloud (denoted as OF@TEIN+ Playground) connects around 14 sites located in 10 countries (ie, Korea, Malaysia, Thailand, Indonesia, Laos, Cambodia, Vietnam, Myanmar, Bhutan, and India) and interconnected via OF@TEIN+ network. OF@TEIN+ playground maintains hyper-converged box-style resources named "SmartX Micro-Box." These distributed boxes are configured to support interconnected cloud-native and edge computing with virtualized/containerized IoT-SDN-Cloud functionalities.

Although many conventional monitoring solutions are now available, maintaining multi-layer visibility (ie, resource-layer and flow-layer) is still limited when operating such complicated infrastructures. To address this particular limitation, "SmartX Multi-View Visibility Framework" (MVF) provides multi-layered visibility (underlay, physical, and virtual resource-layers and flow layer).[6,7] In comparison to the monitoring that mainly focuses on measurement data, the term *Multi-View visibility* referred here covers both measurement data and data for tracing.[6] SmartX MVF leveraged monitoring enables monitoring of dynamic status metrics for both physical and virtualized resources and associated flows[7] of the playground for OF@TEIN+ playground.[8]

Maintaining persistent visibility[1]is one of the major operational challenges where SmartX MVF and related tools fall short. Especially due to the complexity of heterogeneous underlay networks with distributed resources, visibility is further restricted and this playground faces a variety of reliability-related issues such as network failures, limited resource availability, and flaws in software being triggered by environmental changes. A comparison of network availability among ASI@CONNECT members shows many Research and Education Networks (REN) falling well behind International standards, due to lengthy outages on the TEIN links and re-routing of traffic.[9] Also, SmartX MVF is limited to the collection/processing/storage of visibility data at the center and originally designed for the outdated OF@TEIN playground. However, in the upgraded OF@TEIN+ playground, the number of sites has much expanded in the existing setup. Due to the burden of large data volume generated and transferred by multiple distributed and physical/virtual devices, we experience overload on available network and storage capacity. The additional need for maintaining visibility, data aggregation for analysis, and time-series visualization deemed SmartX MVF infeasible at a certain point. Therefore, requirements of applying multi-view visibility using SmartX MVF need to be improved with the existing environment. In response to the mentioned challenges, a solution is therefore needed that can monitor changes of operational environment and maintain long-term multi-view visibility and visualization, offering a persistent solution with reliable transfer and efficient storage in the complex distributed infrastructure.

In this article, we propose to establish multi-site Playground sites which consist of distributed hyper-converged box-style resources named "SmartX Micro-Box." These resources are configured to support cloud-native edge computing having software stack that is, microservices oriented with virtualized/containerized IoT-SDN-Cloud functionalities. Second, we propose *Persistent Visibility* for distributed cloud-native edge boxes that enable collection of multi-view visibility data from distributed Micro-boxes at a centralized location, ensuring reliable transfer of visibility data and resilience in running applications. Also, persistence [2]in visibility data includes recovery of data as well as timely recovery of monitoring functionalities.[3]Visibility data collected, parsed, and validated from distributed resources is reliably transferred at a centralized location named "SmartX Visibility Center" and stored in "Visibility DataLake" waiting for visibility data integration for analysis and subsequent interactive visualizations.[11] Finally, to effectively verify the operational status of physical, virtual, and container types of resources, we have leveraged multi-belt onion-ring style visualization for playground.[12] In summary, the key contributions of this article are:

- We propose setup and configuration of unique cloud-native edge boxes to support Cloud-native computing with IoT-Gateway and edge computing capabilities for the specialized environment of OF@TEIN+ playground.

[1]Visibility is defined as the state of being able to see or be seen.
[2]Persistence in native cloud computing, services are running without interruption and failure.[10]
[3]Functionalities: The applications running in the Box-style resource.

- We provide prototype implementation for maintaining persistent multi-view visibility by:
  – Monitoring running processes with timely recovery and alerts generating capabilities (ie, within 30s processes recovery).
  – Visibility collection for over a month with minimal visibility data loss and recovery during short-term network connectivity outages (ie, 99.9% visibility collection).
  – Over 80% network and storage overhead during visibility collection.
- Persistent multi-view visibility through Onion-ring visualization, a time-series based visualization, and a historical summarized report to verify sustainable playground operation.

The remainder of this article is organized as follows: Section 2 elaborates on the requirements together with overall approach and challenges entailed by the inclusion of persistence visibility for edge clouds. The design and the context offered are described in Section 3 followed by relevant implementation in Section 4. Verification results are provided in Section 5. Finally, in Section 6, we conclude the article.

2 | REQUIREMENTS AND APPROACH FOR MAINTAINING PERSISTENT VISIBILITY

In this section, we list requirements for persistent SmartX MVF for distributed (multi-site) edge clouds. Next, we mention the approach for maintaining persistent visibility by leveraging SmartX MVF. At the end of this section, we briefly highlight the related work.

2.1 | Requirements of persistent visibility

In distributed edge, clouds under multi-site underlay infrastructure, the key issues of visibility collection in playground resources include heterogeneous resource monitoring and operational continuity. Such monitoring infrastructure is required that should persist the monitoring despite having associated issues such as failure or delays in the hardware, and software and network connectivity. Thus, persistence visibility in this environment is expected to identify and react timely to undesirable system states such as connection loss. While centralized control could reduce complexity of management and operating costs, centralized management approaches have several limitations in current network environments, despite being widespread.[13] One of these constraints relates to computational scalability, as an increase in the amount of managed network devices results in a proportional rise in the computational load and network traffic needed on the management station. Based on the distributed edge cloud challenges described in Section 1, on existing monitoring solutions, we have established following requirements for maintaining SmartX Multi-View.

R1) Distributed cloud-native edge: OF@TEIN+ playground with cloud-native edge supports an open-source software stack to be containerized, dynamically orchestrated, and micro-services oriented. This requires realization of functionalities with open APIs enabled by hyper-converged (compute /storage/networking) resources.

R2) Resilient measurement: Resilience means a system can stay responsive when a failure occurs. It applies to both the running functionalities and its underlying infrastructure. In case of failure, these functionalities should be able to recover within a specific time, while during temporary network delay/loss visibility data should be retained and be able to recover fully at the centralized collection.

R3) Efficient and timely measurement: Underlay heterogeneous networks, due to their automated management and increased complexity, requires (i) efficient monitoring and (ii) timely measurement of visibility data in terms of comparatively low network bandwidth and storage consumption, with the raw visibility data. Reducing visibility data in terms of bandwidth and storage consumption is required to reduce overhead on available resources. The requirement for near real-time inclined toward minimizing excessive delay to the distribution of metrics; where excessive is anything over one second.

R4) Verification: Visibility data should be visualized in a historical format to verify persistence of SmartX Multi-View visibility operation over time. Visibility persistence coverage includes metrics measurements, and visibility data summarizing and storage.

2.2 | Proposed approach for persistent visibility

The proposed approach for maintaining persistence deals with multiple layers and stages of OF@TEIN+ Playground.[6] As depicted in Figure 1, resource-layer visibility is responsible for monitoring and visualization of playground physical/virtual resources and interconnects.[14] Flow-layer visibility is in charge of offering various levels of flow-centric information by using flow collection, aggregation, and tagging schemes to support flexible networking (ie, Steering/mapping).[15] Workload-layer visibility which has limited coverage includes the monitoring and visualization of interconnected functions and their positioning for containerized functionalities.[16] Furthermore, proposed solution consists of four main stages, that is, "Measurement Collection and staging" stage collects, validates, and process visibility data from selected metrics. "Visibility transfer" stage, ensure reliable transfer of data. "Visibility Storage and Staging" is responsible for storage of cleaned visibility data at multiple data stores. Whereas, "Visibility Integration" stage integrates collected visibility data for summarization. Finally, "Visualization" stage access processed visibility data for transforming into graphical outputs according to the various visualization requirements.

Table 1 shows the metrics collected for multi-view visibility. While maintaining persistent visibility, it is assumed that playground servers are up and running. Measurement of visibility metrics directly from underlay resources is not supported due to insufficient privileges. As a workaround overlay restricted active monitoring approach is applied. During visibility functionalities failure, they are restarted within the defined interval and reported to the Visibility Center

The summarized solution for maintaining persistent visibility is given below:

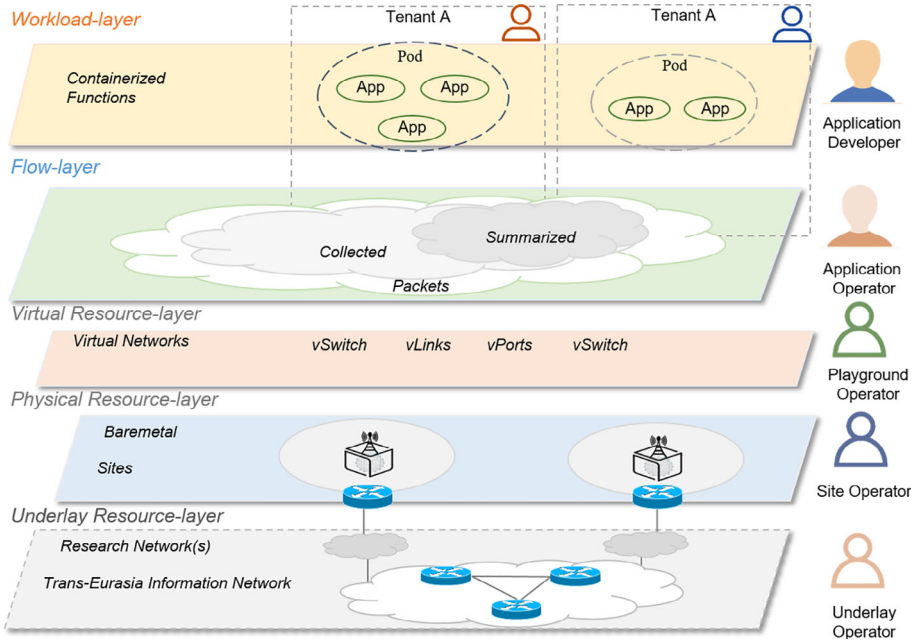


FIGURE 1 Concept of multi-layer visibility for unified monitoring

TABLE 1 Metrics collected for multi-view visibility

Goal	Visibility-Layer	Measured Metrics
Active monitoring (connectivity status)	Resource-layer	Ping (Between Site-to-site and site-to-visibility centre)
Active Monitoring (performance of network traffic)	Resource-layer	Latency (Between Site-to-site)
Active Monitoring (Network capacity)	Resource-layer	Bandwidth (tcp, udp)
Overlay network traffic as passive monitoring	Flow-layer	Packet tracing
Summarized Flows	Flow-layer	Flows (from Packet) tracing
Traffic on Interface	Resource-layer	Bytes received/sec, Bytes sent/sec
System performance statistics	Resource-layer	CPU, Load, memory, disk
Agent-based monitoring	Workload-layer	Applications running status and alerts

- Proposed solution for preparing and configuring cloud-native edge boxes in OF@TEIN+ playground capable of maintaining persistent multi-view visibility.
- For resilient measurements during either process failure or network loss, data collection from underlay resources should be recoverable without loss of monitored data. To achieve resilience against functionalities failure an Agent-based solution is proposed. Agents deployed at distributed and centralized resources are responsible for keeping functionalities in running state. While for reliable transfer of visibility data during network connectivity loss, a shared Messaging-interface is proposed. Messaging service provides the capabilities for retaining data temporarily and guaranteeing once delivery to the Visibility Center.
- To reduce network and storage overhead we proposed a solution for summarized packet collection to reduce more than 80% of bandwidth and storage overhead. Next, to achieve near real-time and resource-efficient visibility, a low polling interval is proposed for monitoring of visibility layers.
- Multi-view visibility data is collected/parsed/aggregated at the Visibility Center from distributed Micro-Boxes. The stored visibility data in a time-series database supports historical retrieval that enables summarized reports send as Daily Visibility report to operators and administrators. Next, we proposed visualization schemes for verification of persistent operations of playground such as multi-belt onion-ring visualization and time-series visualization.

2.3 | Related work

The rapid development of communication, information technology, and the Internet of Things provides new opportunities for many applications to optimize services and scheduling resources.[17] A number of efforts have been considered on using the edge, but most of them focus on achieving better performance by optimizing task scheduling by minimizing network overhead without considering the nodes itself.[1] The Internet of Things contains a large number of terminal equipment. Most of them have only minimal computing, storage, and communication capabilities, hence they need to rely on cloud platforms to enhance node information processing capabilities. A fine-grained approach to improve the effectiveness and efficiency of resource processing is discussed with a focus on maintaining the consistency of the original content.[18] A dynamic reconfiguration scheme considers both service stability and cost of service invocation for the cloud-edge environment.[19]

Most works have focused only on cloud management and monitoring in the literature. GMonE proposes a unified monitoring concept along with an implementation of cloud monitoring system.[10,20] However, GMonE does not detect any failures and anomalies in the cloud infrastructure. OPTIMIS Toolkit, as an example of service-oriented monitoring solution,[21,22] provides monitoring data about physical and virtual resources of cloud infrastructures and services to assist the processes of self-management.[23] However, OPTIMIS but cannot be applied to federated settings where resources from heterogeneous infrastructures are provided to customers. Continuous Monitoring (CoMo)[24] is another solution for supporting monitoring over multiple infrastructures. CoMo provides a unified data interface but is limited to passive network measurement. Flexible monitoring solution at the edge (FMonE)[25] is a monitoring framework that flexibly manages highly customizing monitoring agents and their back-ends.

To monitor distributed systems and take appropriate decisions, many researchers have concentrated on developing and using third-party open-source tools or use customized tools to log application metrics, such as Zabbix, [4]Nagios, [5] or Cacti.[6]However, Nagios is limited in terms of scalability and dependencies on third-party tools for resource monitoring. Zabbix is cumbersome in configuration and limited in auto-discovery. Cacti, on the other hand, lacks application monitoring with limited historical data and few supported plugins. Nagios and Cacti having better functionality but improper to the requirements of cloud computing.[1]

For multi-layer visibility, SmartX MVF is a useful framework.[7] However, minimal consideration was given at the requirements needed to support resource-constrained nodes reachable only over unreliable or bandwidth-limited network connections or thought about the needs of applications that demand very high bandwidth, low latency, or widespread compute capacity across many sites. In comparison, we adopted an approach to provide persistent operations for OF@TEIN+ playground developers and operators to effectively operate and maintain the playground. Through unique visibility support of onion-ring and time-series based visualization, deep insights into current operations are provided.

[4]https://www.zabbix.com/
[5]http://nagios.org
[6]https://www.cacti.net/

3 | PERSISTENT SMARTX MULTI-VIEW VISIBILITY: DESIGN

In this section, we discuss the proposed design of cloud-native edge Boxes and identify key terminologies. Afterward, we discuss detailed components design as functionalities responsible for maintaining multi-view visibility from measurement phase at the Micro-Box up-to-the storage phase at the Visibility-Center.

3.1 | OF@TEIN+ distributed cloud-native edge boxes

OF@TEIN playground launched to serve as SDN-enabled multi-site clouds for miniaturized academic experiments is a miniaturized, software-defined playground with hyper-converged box-style resources. It is an overlay-interconnected, multi-site playground over heterogeneous (3 ∼7 inter-connected) underlay networks. Built for cloud-native edge boxes, OF@TEIN+ playground leverages "SmartX Playground Tower," a logical space in centralized location meant to automatically build, operate, and utilize OF@TEIN+ playground. Following the concept of "monitor & control" tower,[26] playground tower leads the operation of multi-site playground by employing several entities such as Visibility Center. OF@TEIN+ maintain multiple types of distributed physical and virtual resources that are dynamically provided to the developers to learn operational and development issues and perform various experiments. For visualization, SmartX multi-view provides multi-layer and multi-stage support. As shown in Figure 2, underlay backbone is provided and managed by TEIN. Then we have a layer for NREN followed by different sites containing various types of resources. The Site is a physical location containing server boxes with multiple virtual resources. SD-Access controls virtual networking among multiple wired and wireless devices in the OF@TEIN+ playground. Unlike cloud-based infrastructure where we use virtual machines and allocate fixed resources to the users, in cloud-native, we switch to the containers to deploy applications over these edge cloud boxes.

In the given environment, we placed cloud-native edge boxes termed "SmartX Micro-Box" aka Micro-Box, distributed at multi-site edge locations of OF@TEIN+ playground. These Micro-Boxes are commodity server-based hyper-converged resources (compute /storage/networking) to allow experiments (Cloud/Software-defined networking/Network Function Virtualization) over OF@TEIN+ playground. Supported by cloud-native (containerized) functionalities, Micro-Box is prepared as Kubernetes-orchestrated workers with SDN-coordinated special connectivity to other Micro-Boxes. The tenant awareness of deployed containerized application provides isolation. Currently, only namespace isolation is provided. For network connectivity, we use three network interfaces as shown in Figure 3. Micro-Box is configured with two public IPs, configured as control interface and data interface. For remotely checking the power status of Micro-Box and for booting purpose, Intelligent Platform Management Interface (IPMI) port is used. Besides, for Pods and container communication, we use private IP addressing schemes. For persistent connectivity, multi-access interface is enabled with Wi-Fi and two wired interfaces. In terms of visibility measurements among multiple distributed sites, Micro-Boxes are configured in mesh-style networking, where visibility collection is reported to the centralized Visibility Center at the SmartX Playground Tower. As depicted in Figure 3, the applications running in the Micro-Box are termed as "Functionalities," whereas the

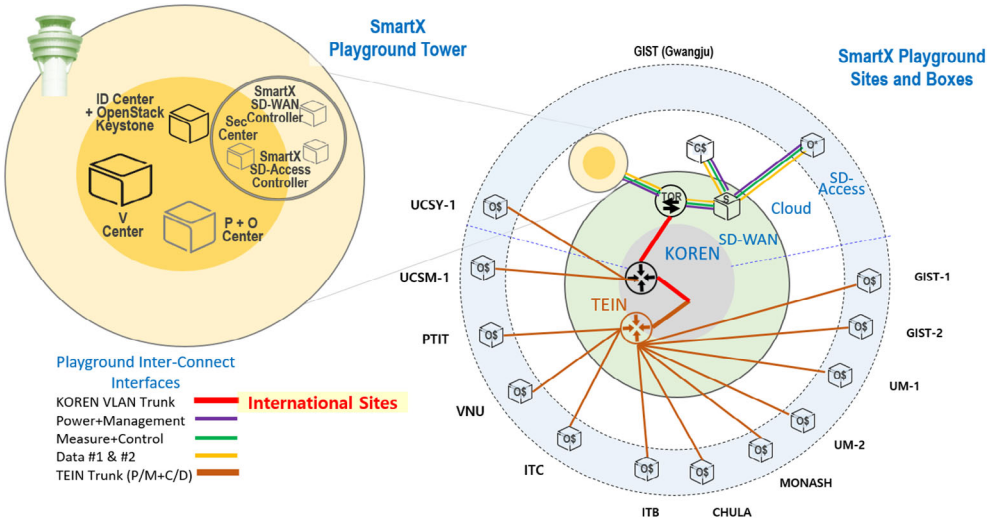


FIGURE 2 OF@TEIN+ playground with SmartX Playground Tower for SmartX Micro-Boxes Operation

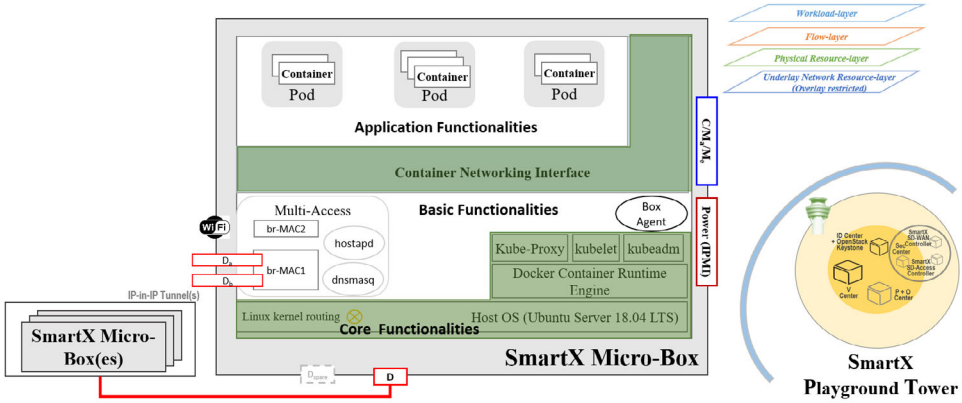


FIGURE 3 Design of SmartX Micro-Box software ($) Coordinated with SmartX Playground Tower

corresponding applications at the Tower Centers are termed as "Companion functionalities." The software version in the Micro-Box is named as "$ (Dollar)" and the "Core functionalities" consist of software functionality that manages the Micro-Box operations such as OS, kernel, Kubernetes, etc. "Basic functionalities" are additional development at the bare metal and constitutes monitoring of connection status/health with connected Micro-Boxes in the playground, ensuring collection consistency and service reliability. The "Application functionalities" consist of applications in the form factor of containers orchestrated through Kubernetes. Maintaining persistent visibility is crucial for the effective operation of OF@TEIN+ playground to ensure reliable operations with timely knowledge of issues concerning playground.

As shown in Figure 4, measurement data consists of active and passive monitoring. Active monitoring periodically tracks the connection status among playground sites to offer enough insights to the playground operators, who need to immediately recover problematic resources. By injecting probing packets into specific network paths, active monitoring measures the network performance in terms of delay, loss, jitter, and packet/frame loss. Second, passive measurement involved packet precise tracing of all incoming and outgoing packets from any particular site for analyzing passing traffic.

Furthermore, to ensure reliable and uninterrupted operation, we collected regular connection status of Micro-Box with the Visibility Center called "liveliness status of Micro-Box," as well as connection status with other Micro-Boxes called "interconnection liveliness." Reporting is action taken by Micro-box to an assigned task (eg, sending collected measurements to the center, whereas response is the action taken at Visibility Center to an event generated from Micro-box). Measurements are transferred through a shared messaging interface ensuring reliable communication and resistance against losses. To ensure persistent execution of functionalities, Agents monitors and starts stopped services. Box-Agent asynchronously communicates with Center-Agent with status messages. Persistent storage stores data for real-time and temporal analysis.

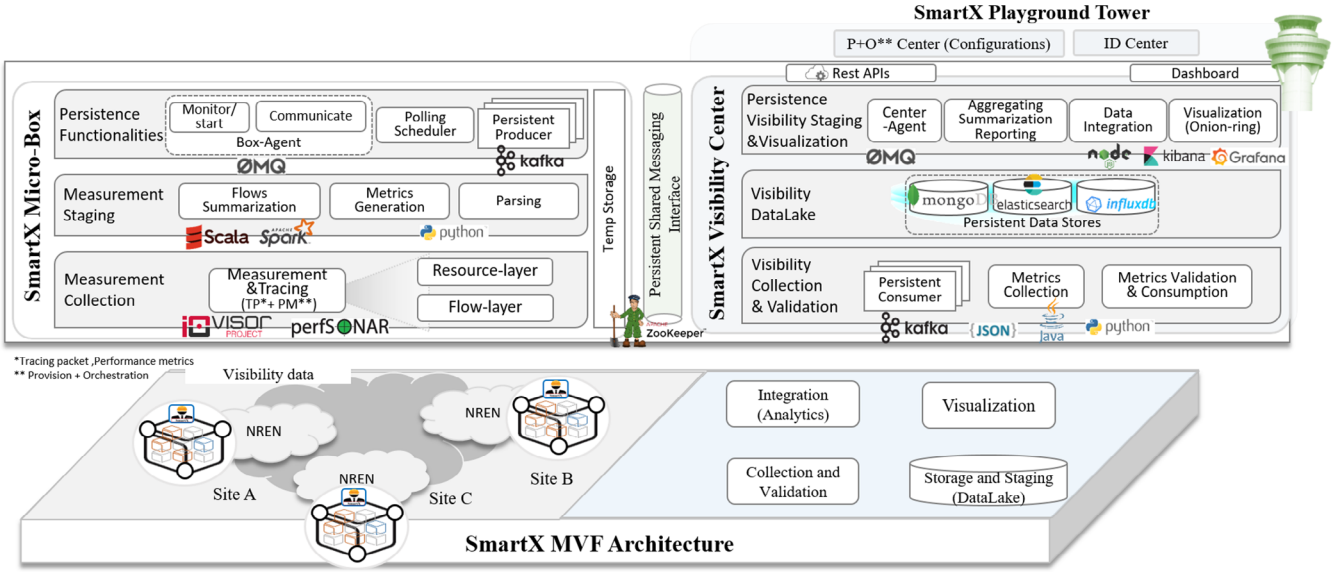


FIGURE 4 Design of maintaining persistent visibility at Micro-Box, Interconnection, and SmartX Visibility Center

3.2 | Multi-agent and shared messaging interface for maintaining functionalities and visibility data

As stated in the requirements R2, functionalities need to run continuously, have woke-up capabilities, and generate alerts at a centralized location. We have implemented lightweight visibility-assisting agents that monitor, wake up, and send alerts on running functionalities in the distributed Micro-Box and companion functionalities at the Visibility Center. Agents are designed in the given playground environment to process in a reactive way to the changes in functionalities and communicate with other Agents in a request-response manner. For distinction, agents running at Micro-Box are termed as "Box-Agent" and agent running in the Visibility Center is termed as "Center-Agent." Both these agents communicate in an asynchronous socket-based connection. For persistence measurements, the visibility agent executes at a high polling interval. Center-Agent performs additional monitoring of distributed Box-Agents at regular intervals and sends an alert. Since measurements for visibility have to be collected at regular time intervals, agents can actively wake-up stopped functionalities. Afterward status of stopped functionalities is reported as alert to Visibility Center, together with Micro-Box name, total downtime by using a shared messaging interface. This status enquiring involves back and forth messaging between the agents. Each Box-Agent is designed to periodically inquire about the status of running functionalities as well as respond to incoming messages from Center-Agent. The process flow for Multi-Agents is shown in Figure 5. Event-based alerts are sent through a reliable shared messaging interface and stored in the Visibility Center. Agents are designed to work in a distributed environment and enabled flexible support for interactive messaging. Considering multiple socket connections opened during communication between agent, network overhead and scalability is a challenge. To tackle these issues, Agents are enabled with request and reply based asynchronous communication. This allows agents to respond to another agent's request. For scalability, we have maintained configuration based input for monitoring to handle increase in functionalities and/or Micro-Box. The persistent capability of monitoring collection functionalities leverages publish and subscribe based distributed shared messaging interface capable of high throughput, fault-tolerance, and overhead less metadata management.

At the shared messaging interface, producer and consumer are responsible for reliable transfer and receiving of visibility data. However, Producer and consumer are prone to data loss when the connection between Micro-Box and other entities is lost due to power failure or network issues. As depicted in Figure 6, to counter these losses, we configured the shared messaging interface to retain the measurement data by enabling temporary storage and configuring specific parameters for a shared messaging interface for persistent sending and collection. When tuning the shared messaging interface for optimal performance, it is important to set the measurements according to the polling intervals of monitoring measurement, instead of keeping the default or maximum values. Since by default the producer (ie, sending entity) does not wait and sends the buffer any time data is available. However, during connection loss between Micro-Box and Visibility Center, data need to be retained until recovery. Therefore, we increase the buffer size to a certain value that retains a certain amount of measurement for a specific duration. During connectivity loss, reporting of visibility data is stopped. However, functionalities continue the measurement process. On sensing the connection loss, the shared messaging interface enables temporary collection at the local storage. After the revival of connection, the locally stored visibility data

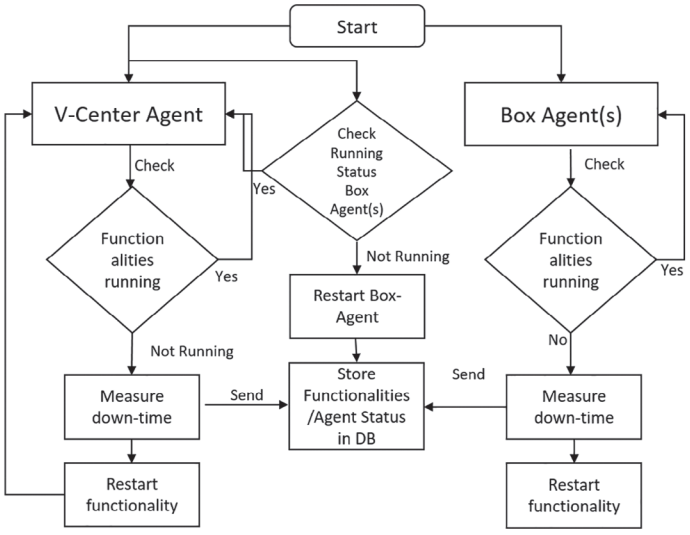


FIGURE 5 Multi-Agents process flow

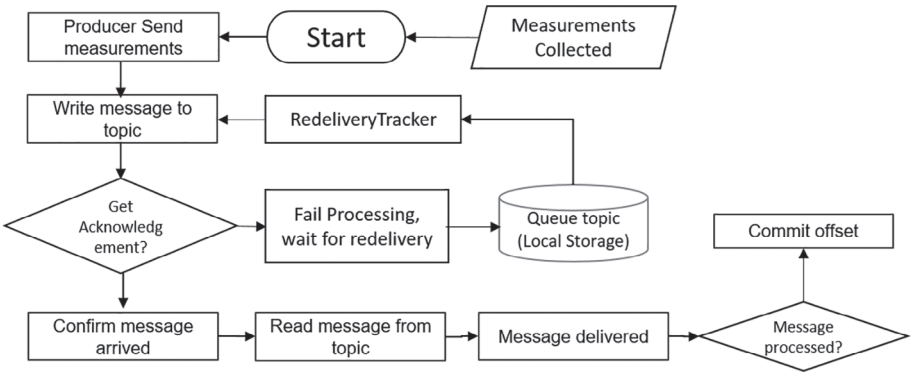


FIGURE 6 Maintaining SmartX multi-view visibility at interconnection and SmartX Tower

collected is sent back to the tower from local storage by estimating the disconnected period and time. Since the shared messaging interface is accessed simultaneously by multiple entities for writing and reading visibility data, so to perform essential coordination between multiple entities, we enabled automated management for metadata and synchronization issues.

3.3 | Flow summarization and optimal polling for maintaining network and storage load

In the initial design of SmartX MVF, aggregation from packet tracing is carried out at the Visibility Center. However, an increase in the network and storage load in terms of bandwidth and storage size is observed during the visibility data transfer and collection phase at the Data Lake.

To minimize this overhead, we have modified SmartX MVF to implement flows summarization process resulting from aggregated packet tracing, at the Micro-Box with the integration capabilities at the Visibility Center. Flow-centric visibility is an upright method to understand what traffic is traversing the network, instead of looking at network traffic from the packet level. The key idea of flow-centric visibility for packet precise collection is to store information about flows and the corresponding statistics instead of individual packet information at centralized Data Lake.[27]

Algorithm 1. Packet Tracing for Base Collection

Require: Basic IP Packet ▷ p = packet buffer
Ensure: 6-tuples of header form IP-only packet as tracing criteria ▷ 6-tuple packet

procedure BASE TRACING(p)
  if Packet in the Interface does not match the given IP ($p.ethtype \neq IP$) **then**
    Drop the packet p
  end if
  Extract source address ($srcaddr \leftarrow p.ip.srcaddr$)
  Extract destination address $destaddr \leftarrow p.ip.destaddr$
  Extract packet length$length \leftarrow p.ip.totallength$
  if $p.ip.nextproto \neq TCP$ **then**
    if $p.ip.nextproto \neq UDP$ **then**
      Drop the packet p
    end if
  end if
  Extract protocol ($protocol \leftarrow p.ip.nextproto$)
  Extract destination port ($dstport \leftarrow p.ip.tcp.dstport$)
  Extract source port ($srcport \leftarrow p.ip.tcp.srcport$)
  return Headers
end procedure

Algorithm 2. Flowcentric tracing packet Summarization

Require: Load network packets (P) data from disk as $Np_1 \ldots Np_N$
Ensure: Sum of Aggregated packet)

```
1:  procedure TRACING SUMMARIZATION(p)
2:      Load network packets data from disk as N ← list(P)
3:      for A given time-window do
4:          Combine all data Packets together
5:      end for
6:      validate collected visibility data
7:      for For each packet as P ← 1 to N do
8:          Calculate count of protocols
9:          Calculate min/max/average tcp window size
10:         Calculate min/max/avg databytes
11:         Calculate max collection time as flow duration
12:     end for
13:     Add processing time to data frame
14:     Save result to a file format
15:     return P_Agg                          ▷ Aggregated packet for a time Frame
16: end procedure
```

For packet tracing and flow-centric packet Summarization, algorithm 1 illustrates the method of implementation of our IO Visor-based tracing algorithm. We developed tracing based on a different criterion to trace basic IP packet and TCP specific information. It gives us TCP-related traced data to fetch concerned metrics of the packet based on the TCP behavior.

Algorithm 2 describes the implementation for the summarized flows from packet tracing. Visibility data is aggregated, tagged and prepared for multi-layer visibility data integration component.

As shown in Figure 7, multiple measurements as visibility data are collected from distributed locations and stored at centralized storage. For synchronized collection of measurements at a centralized database, uniform time is set on the distributed Micro-Boxes. A customized script named "scheduler," collects and sends the measurements at regular time intervals with the same proximity over all Micro-Boxes.

Second, during visibility collection poorly planned probing can result in network performance degradation. The associated visibility metrics selection with fine-grained polling intervals affects the persistent quality of this collection. For persistent playground visibility, selection of metrics and frequency of polling interval are two important factors to consider. However, we found that metrics selection is dynamic for each use case; likewise, the polling interval depends on the monitoring and system sensitivity demands. Thus, based on these observations, we select different metrics and polling intervals for satisfying various demands of playground visibility. A template file with configuration parameters is used

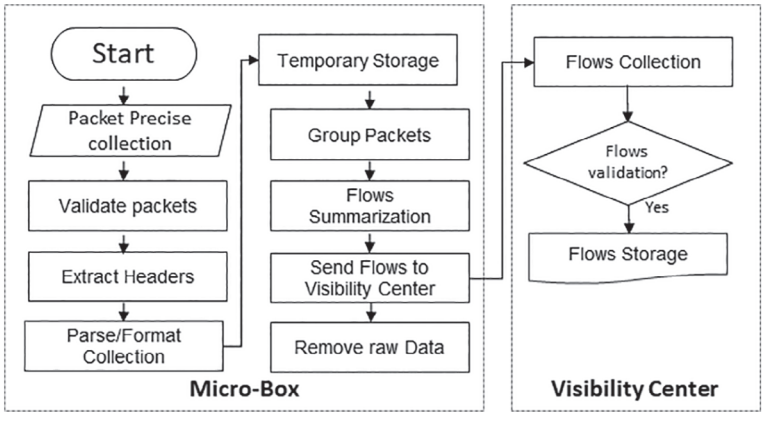


FIGURE 7 Design for flows summarization

with specific metrics that need to be measured. In addition, customized tools are implemented to measure, visibility of physical and virtual resources flows and connected Micro-Boxes.

3.4 | Maintaining SmartX multi-view visibility at the tower

Multi-view visibility data from Micro-Boxes gets collected in the DataLake of Visibility Center. For verification of persistent visibility monitoring tasks, we organize and summarize visibility data as either collection or average for a day. After collecting visibility data along with summarization, we configure a visualization tool to retrieve that information and create graphs/bars to display persistent visibility along with missed or delayed collection due to resource unavailability. Furthermore, visibility collection provides useful troubleshooting points for operators over the running resources hence facilitate in sustainable operation.

At the Tower raw visibility data from resource-layer is kept at a DataLake for analysis and aggregation is sent to next stage for further processing. Table 2 describes Raw visibility data collected from Micro-boxes comprising of metrics (eg, cpu, disk) from resource layer. These metrics from multiview-layer are parsed and integrated into respective databases. Furthermore, to verify Multi-view visibility and to match unique single-box-containerized deployment style of OF@TEIN+ playground, we leverage multi-belt onion-ring style visualization, adopting more systematic and fine-grained visualization with a clear separation between multiple layers and multiple sites.

In addition, for visualization of summarized persistent visibility, a customized tool generates aggregated visibility data for one-day. This compiled data is formatted as an HTML report and disseminated to the playground operators as shown in Figure 8. This daily visibility report aggregates collection of multi-view visibility for a 24-hours period as sum/average/percentage of collections and measurement. Our solution provides a historical view of persistent multi-view visibility with customized visualization. The stored visibility data in a time-series database supports near real-time and historical retrieval through visualization and verify persistent visibility. Also, playground operators can verify uniformity among micro-boxes by comparing measurements on the same time-line.

TABLE 2 Raw format multi-view visibility data for parsing metrics

Raw Visibility Data for cpu/disk		
[{“values”:[0,0],	[{“values”:[63.9020263657957],	[{“values”:[0,0.30004865136956],
“dstypes”:[“derive”,“derive”],	“dstypes”:[“derive”],	“dstypes”:[“derive”,“derive”],
“dsnames”:[“io_time”,“weighted_io_time”],	“dsnames”:[“value”],	“dsnames”:[“read”,“write”],
“time”:1589692165.948,	“time”:1589692165.948,	“time”:1589692165.948,
“interval”:10.000,	“interval”:10.000,	“interval”:10.000,
“host”:“smartx-microbox-gist-2”,	“host”:“smartx-microbox-gist-2”,	“host”:“smartx-microbox-gist-2”,
“plugin”:“disk”,	“plugin”:“cpu”,	“plugin”:“disk”,
“plugin_instance”:“loop0”,	“plugin_instance”:“1”,	“plugin_instance”:“nvme0n1”,
“type”:“disk_io_time”,	“type”:“cpu”,	“type”:“disk_time”,
“type_instance”:“”}]	“type_instance”:“idle”}]	“type_instance”:“”}]

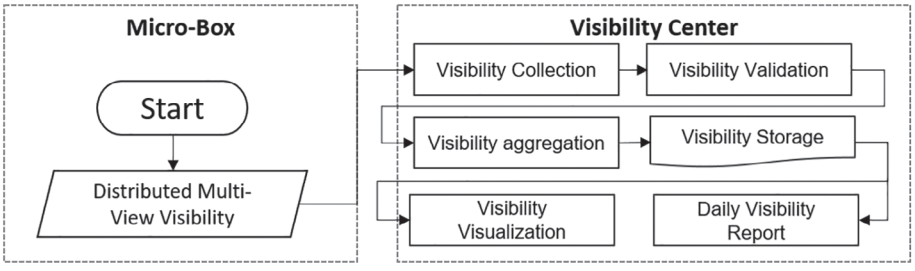
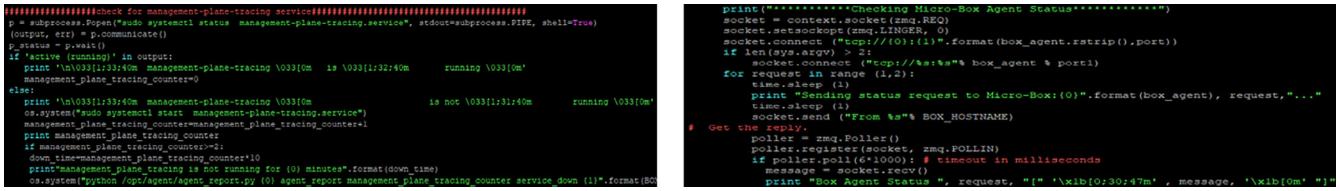


FIGURE 8 Visualization for persistent visibility



```
################check for management-plane-tracing service#################################
p = subprocess.Popen("sudo systemctl status  management-plane-tracing.service", stdout=subprocess.PIPE, shell=True)
(output, err) = p.communicate()
p_status = p.wait()
if 'active (running)' in output:
  print '\n\033[1;33;40m  management-plane-tracing \033[0m  is \033[1;32;40m       running \033[0m'
  management_plane_tracing_counter=0
else:
  print '\n\033[1;33;40m  management-plane-tracing \033[0m                    is not \033[1;31;40m       running \033[0m'
  os.system("sudo systemctl start  management-plane-tracing.service")
  management_plane_tracing_counter=management_plane_tracing_counter+1
  print management_plane_tracing_counter
  if management_plane_tracing_counter>2:
   down_time=management_plane_tracing_counter*10
   print"management_plane_tracing is not running for {0} minutes".format(down_time)
   os.system("python /opt/agent/agent_report.py {0} agent_report management_plane_tracing_counter service_down {1}".format(BOX
```

```
    print("***********Checking Micro-Box Agent Status***********")
    socket = context.socket(zmq.REQ)
    socket.setsockopt(zmq.LINGER, 0)
    socket.connect ("tcp://{0}:{1}".format(box_agent.rstrip(),port))
    if len(sys.argv) > 2:
        socket.connect ("tcp://%s:%s"% box_agent % port1)
    for request in renge (1,2):
        time.sleep (1)
        print "Sending status request to Micro-Box:{0}".format(box_agent), request,"..."
        time.sleep (1)
        socket.send ("From %s"% BOX_HOSTNAME)
#  Get the reply.
        poller = zmq.Poller()
        poller.register(socket, zmq.POLLIN)
        if poller.poll(4*1000): # timeout in milliseconds
            message = socket.recv()
            print "Box Agent Status ", request, "[" '\x1b[0;30;47m' , message, '\x1b[0m' "]"
```

FIGURE 9 Code snippet for agents communication and functionalities checking

4 | PERSISTENT SMARTX MULTI-VIEW VISIBILITY: IMPLEMENTATION

In this section, we describe the implementation of OF@TEIN+ distributed cloud-native edge boxes followed by the role of multi-agents and a shared messaging bus to enable persistence visibility. Next collection of visibility data is discussed. Later we explained how visibility data is maintained at the SmartX Tower.

4.1 | OF@TEIN+ distributed cloud-native edge boxes

As part of core functionality, each Micro-Box is installed with Ubuntu 18.04 server edition. For connectivity network interface "eno1" provides control and management functionality while eno2 is used as a data interface. Both these interfaces have public IP from the REN network. Eno3 and eno4 are used for wired IoT devices. Basic functionalities include various tools. As shown in Figure 9, for measurements of selected multi-layer metrics, we select and deploy plugin-based open-source tools like IO Visor and Collectd at each Box. Leveraging PerfSONAR, overlay restricted active monitoring approach (eg, performing site-to-site latency and TCP/UDP bandwidth measurements) is implemented to offer enough insights to the playground operators and respective network administrators, who can immediately identify and recover resource issues. Furthermore, to aid the operators on the visibility of resource availability, we have measured connection status between the boxes and with the center. When performing active/passive monitoring over the shared underlay networks, a high frequency of measurements can cause congestion and overhead, so we selected lower polling frequency. For automating deployment and management of containerized applications, *Application functionalities* include container orchestration through Kubernetes. While Kubernetes Master is provisioned in in P+O Center at SmartX Tower. For Flow-layer measurement, we applied summarized collection using Apache Spark and Scala based-implementation. Collection for multi-view visibility is sent as JSON format through Kafka-based implementation. In addition, for fault tolerance, Apache Zookeeper is used to manage configuration for distributed synchronization. At the tower, we estimate the aggregated collection of measurement data using a Java-based customized program, to provide useful knowledge about the consistency of running visibility functionalities and identify the missing resources as a basis of troubleshooting. To estimate the robustness of measurement at the Micro-Boxes, at the Tower total number of collected visibility data is compared with the expected collections in terms of percentage gain/loss provides the numbers. To ensure continuous operation of functionalities, Box-Agent and Center-Agent are implemented using python and zeroMQ. Finally, visibility data is collected in MongoDB or Elasticsearch and visualization support is provided using a NodeJS-based implementation.

4.2 | Multi-agent and shared messaging interface for maintaining functionalities and visibility data

As discussed in the design section, multi-agents in the playground are implemented as either Box-Agent or Center-Agent. A common goal of multi-agent is to keep the functionalities in running condition by periodically inquiring about the status of functionalities. Functionalities that fail to run are reported as an alert in the DataLake. Center-Agent is also capable of communicating with each other in a request-response manner. Center-Agent performs the additional task of periodically sending requests to Box-Agent and waits for a reply. Center-Agent can autonomously wake-up Box-Agent that fails to respond or stopped running. For two-way communication, zeroMQ is implemented as an efficient, embeddable that can handle I/O asynchronously and background threads. It supports a variety of messaging patterns as well as wrappers for multiple languages. Also, during messaging, it queues messages automatically when needed by pushing messages to the

FIGURE 10 Automatically acquired flow summarization metrics from playground boxes

Packet header Tags

Measurement_boxname	smartx-microbox-gist-2
src_host	xxx.xx.xxx.xx
dest_host	xxx.xx.xxx.xx
src_host_port	52034
dest_host_port	9092
protocol	6
net_plane	0

Flows summarization Tags

protocol_count	757
min_tcp_window_size	2425
max_tcp_window_size	2426
avg_tcp_window_size	2426
std_dev_tcp_window_size	0.04
min_data_bytes	67
max_data_bytes	11636
avg_databytes	1510.18
std_dev_databytes	985.06
total_data_bytes	1143203
flow_duration	290.016
Processing_time	2020-01-14T11:00:31.136+09:00

TABLE 3 Polling settings for multi-agents

Agent functionalities	Running schedule	Language/Library
Functionalities Status	30 seconds	Python
Connect and check Box-Agents Status	30 Seconds	Python/ZeroMQ
Send status report	Event based	Python/Kafka
Tests Configurations	-	YAML
Store status report	Event based	MongoDB

receiver before queuing them. Communication between Box-Agents and Center-Agent support arbitrary transports: TCP, multicast, and based on REQ/REP pattern where socket will be blocked unless it has successfully received a reply or request.

Figure 9 shows an example code to send socket request and check the running status of Zookeeper service. In case service is not running for couple of checks, Agent will note the downtime and send the status as alerts to the Visibility Center through Kafka-based shared messaging interface. These alerts are stored in MongoDB with headers for Micro-Box ID, time, functionality name and downtime. As given in Table 3, to fulfill the requirement of real-time, resilience, and low polling time is set to check running Box-Agents as well as running functionalities.

For reliable transfer of visibility data from Micro-Boxes to SmartX Visibility Center, a shared messaging interface with high throughput, fault-tolerance, and metadata management features is essential. To address the visibility data transfer issue, shared messaging interface is implemented through chosen software, named after Kafka. The Shared messaging interface is accessed simultaneously by multiple entities for writing and reading visibility data. To perform essential coordination between multiple entities, we use open-source software called Apache Zookeeper for automatically managing metadata and synchronization issues. Messages are retrieved from Kafka brokers. A better solution is that offsets are not committed automatically based on a time interval. This would ensure that the Kafka-consumer only commits offsets after it receives an acknowledgment from the downstream-consumer signaling successful messages consumption. This leads to duplicate messages in downstream systems, but no data loss. At the interconnection, data is sent through Kafka. Specifically, two scenarios that can cause visibility loss are handled as below,

1. *Network connection is down at the Micro-Box or Visibility Center*: By default, shared messaging interface retains limited visibility data for a short period. We enable and increase the short-term storage at the Micro-Box to retain 24-hours of visibility during connection loss, incorporating timestamp, tagging schemes in a separate topic for each type of visibility. On connection revival, visibility data is restored with the matching topic maintaining the time sequences.
2. *DB Applications or Shared messaging interface not running*: Although running functionalities are being monitored by Center-Agent and restored within 30 seconds, any visibility data fetched from shared messaging interface and not delivered to the DB will be lost. Thus, we check the running status of DB application before collecting visibility from shared messaging interface to ensure 100% retention. Second, by applying retention policy Kafka cluster durably persists published records for 24 hours.[28]

TABLE 4 Kafka configurations for reducing data loss possibilities

Type	Configuration	Default	New Value
Producer	Batch size	16384	50000
	Linger Time	Does not wait	20ms
	Block.on.buffer.full	False	true
	Retires	None	LONG.MAX_VALUE
Consumer	max.partition.fetch.bytes	1 MB	10 MB
	Auto.offset.commit disabled		
Broker	replica.fetch.max.bytes	1 MB	1MB

TABLE 5 Active monitoring metrics and parameters

Layer/Test Type	Monitoring Type	Time between Tests (Min/Hrs.)	Test duration	Packet Rate/ Bandwidth	Packet size (bytes)	Protocol	Direction
Resource/ Ping (RTT)	Active	10 m	-	-	1000	-	-
Resource/ Throughput-TCP	Active	24 h	20 sec	-	-	TCP	Send
Resource/ Throughput-UDP	Active	24 h	10 sec	10 MB	-	UDP	Send
Physical/ One-way Latency	Active	10 m	-	10 packets/sec	20	-	-
Physical/ Liveliness with Tower	Active	10 m	-	-	-	-	From Micro-Box

To achieve the requirement of consistency and availability in the data collection, we set the following parameters in the producer part of Kafka as shown in Table 4.

4.3 | Flow summarization and optimal polling for maintaining network and storage load

To satisfy visibility data summarization requirements, flow-centric visibility is collected from playground boxes. Using SmartX tagging, information is collected from system generated activities as shown in Figure 10. These tags include packet header and flows summarization, where Visibility data is aggregation based on the statistical values for five-minute window by using multiple functions (average/max/min/std_dev) against the key identifier such as SmartX Micro-Box name as ID. Additionally, the collection of visibility data also helps to solve the visibility data processing issue of distributed playground resources locality and data timestamp issues.

Due to various complexity issues, it is difficult to select a precise polling interval for multiple measurements in a single trial. For example, when performing active monitoring measurements over the shared underlay networks, a high frequency of measurement tests can cause congestion and affect multiple stakeholders, so lower polling interval is a suitable selection. For physical and virtual resources, we select a fixed polling interval to acquire in-depth visibility. For playground visibility, selected multi-layer metrics are provided in Reference 26. For accuracy of operation, measurement collection relates to a continuous and consistent collection of monitoring from active, passive and liveliness of Micro-Box, collected at the Visibility Center as per defined frequency interval. Moreover, for reliable and consistent collection, we select specific metrics for active monitoring as shown in Table 5. Site connectivity is evaluated by throughput[7] and latency.[8]

IO Visor employs the extended Berkeley Packet Filtering (a.k.a. eBPF)[11,29] which provides offers in-kernel VMs with byte-code tracing program execution.[30] IO Visor-based packet tracing has the key benefit to monitor and trace kernel and

[7]Throughput refers to the maximum traffic bandwidth.
[8]Latency is the time elapsed during packet transmission.

TABLE 6 Passive monitoring metrics and parameters

Layer/Test Type	Monitoring Type	Tool	Time between Tests	Collected metrics
Resource/Utilization	Passive	Collectd	10 sec	CPU, load, memory, etc.
Flow/Packets tracing	Passive	IO Visor	Event-based	Packet Precise Collection
Flow/packet sampling	Passive	Apache Spark (flows)	5 min	Summarized Flows from Packet precise collection

user events (through kprobe and uprobe), subsequently to keep statistics in maps fetched on the points of interest.[31] By attaching user-space program into Linux networking socket to filter packet data, kernel sends only the matched filtered required data in bytecode to a user-space program. This allows a reduced overhead to trace the packets by defining a specific information field. For flow layer visibility, by leveraging eBPF and IO Visor,[32] we implemented a visibility software that collects information from each packet with a small number of CPU cycles, since it directly raw packets, copied from the network interface of Micro-Box.

Packet precise collection leveraged user-space tracing python program to process network packets at user-level. This program, from the traced packet, extracts nine-tuples information from the packets and add a timestamp to the extracted header fields of the packet. The extracted tuples consist of Machine IP/Hostname, ip version, Source/Destination IP Address, Source/Destination port, protocol, TCP window size, and packet length. Tracing is run based on events generated and collected for five minutes interval in file-based temporary storage. Flow aggregation techniques reduce the amount of visibility data by discarding redundant packets information and merging multiple packets records with similar properties. The extracted traced packet file is utilized for creating summarized flows before sending it to the Visibility Center. A customized script in Scala is developed to generate flow summarization. Furthermore, for efficient processing of visibility data, Apache Spark is employed, since Spark has core advantages of operation speed and extensive API support. Summarized flows are sent to DataLake for storage and analytical purpose. The following command displays execution parameters when running this program.

sudo /$SPARK_HOME/bin/spark-submit –class SmartX.multiview.flowcentric.Main –master local[*] –driver-memory 4g /opt/MultiView-Dependencies/multi-view-flowcentric-aggregate_2.11-0.1.jar*‘*$SmartX-microbox’

As shown in Table 6, the proposed solution generates summarized collection after every 5 minutes at the Micro-Box instead of doing the same at the Visibility Center. In a packet precise collection, the data size increases significantly after a time period. Also, during the misuse of network traffic, such as DoS (Denial of Service) attack, tracing files could quickly increase to hundreds of MB in the given collection period. To tackle this issue, summarized flows significantly decrease network load and subsequently consume limited storage at SmartX Visibility Center.

4.4 | Maintaining SmartX multi-view visibility at the tower

To estimate the persistence in collected measurements and liveliness of resources across Micro-Boxes, customized Java-based functionality is written to aggregate the stored time-series data as a total number of collected measurements and expected collections in terms of percentage/sum/average as shown in Table 7. Measurements are stored at SmartX Visibility Center, inside DataLake provided DataStores such as MongoDB and Elasticsearch. A scheduler-based program

TABLE 7 Measurement parameters for daily visibility report

Measurement Type	Summarization Type	Unit
Latency with Connected resources	Average	Milliseconds
Uptime with connected resources	Total Collection	Percentage
Throughput (TCP/UDP)	Run Once	MB/Sec
Ping Generated from each Micro-Box	Collection/Expected total	Count
Liveliness with Visibility Center	Total Collection	Percentage
IO Visor	Total Collection	Percentage
Collectd	Collection/Expected total	Count

summarizes these collected measurements daily (24 Hours) at a pre-defined time interval. These summarizations are then sent as the overall operational status of playground resources to playground operators in the form of daily visibility reports.

In order to implement the multi-belt onion-ring visualization for OF@TEIN+ playground, we leverage open-source visualization library called psd3 that supports multi-level pie charts. For deployment of onion-ring visualization, Node.js JavaScript run-time is selected due to its non-blocking I/O and event-driven features. To visualize measurement and tracing metrics data, we utilize open-source software named Kibana. To realize Onion-ring visualization, a separate database is built that stores updated configuration and status data of playground entities. This configuration or status database is managed via MongoDB collections.

5 | PERSISTENT SMARTX MULTI-VIEW VISIBILITY: OPERATION AND VERIFICATION

This section verifies that visibility can be persistently maintained by demonstrating the prototype that we have built to fulfill the requirements listed in Section II. First, we explain the operation and verification target environment. Afterward, we verified the role of Agent-based software to keep functionalities running persistently. Next, we present the use cases for verification of low resource consumption through a simulated Dos attack and by comparing multiple parameters for active/passive monitoring. Finally, we present the verification scenario for persistent collection of visibility data at the interconnections and SmartX Tower. The verification environment refers to Figure 1, where resources are distributed in geographical regions are interconnected by overlay-based networks. These resources provide support for developers to manage various functionalities. A prototype of proposed solution can be found on GitHub (https://github.com/SmartX-Team/SmartX-MicroBox.git).

5.1 | Verification target

Deployment of proposed solution at initial-stage utilizes server-based hardware for Visibility Center with the specs: Intel® Xeon CPU E5-2690 V2@3.00GHz, memory DDR3 12x8GB, HDD 5.5TB, and 4 network interfaces of 1Gbits/s. For Micro-Box, we utilized Supermicro SuperServer E300-8D server. The Mini-1U server has 4 CPU cores with 2.2GHz Intel processor, 32 GB memory, and 240 GB of hard disk. It has one dedicated physical interface for IPMI-based remote access management through CLI (command-line interface) and web UI (user-Interface). In addition, it has 2 x 10G + 6 x 1G network interfaces. Visibility Center utilizes Ubuntu 16.04.4 LTS OS, while Micro-Box is configured with Ubuntu 18.04.2 LTS OS, with a minimum kernel version of 4.4.0. Playground developers are provided with a dedicated tenant to execute different networking experiments. Each Micro-Box is prepared with Basic Functionalities responsible for collecting the visibility data. We utilize 14 Micro-Boxes configured with three types of functionalities, that is, Core, Basic, and Application. Each box is running functionalities at synchronized time and interval for a consistent and comparable status of playground operations.

5.2 | Visibility data preparation

Initially, the operational aspect of visibility is managed by preparing 14 Micro-Boxes configured at distributed locations with different REN networks. The purpose was to observe varying network behavior for a longer period (ie, months) to identify issues in maintaining visibility, followed by applying the proposed solution with verification. Furthermore, based on time-series based verification, we categorized stable and unstable Micro-Boxes. For the experiment purpose, we present visibility data from Micro-Boxes at five different sites with varying network behavior. These include site#1: GIST, South Korea, Site#2: UM, Malaysia, Site#3: CHULA, Thailand, Site#4: RUB, Bhutan. Each Micro-Box is configured with '$' version software which includes "basic functionalities" responsible for sending visibility data to the Visibility Center, as part of multi-view visibility. After prep ration, we observed the visibility for one month as *Observation period*, that is, from June 01 to June 30, 2019. During this period, for active monitoring (underlay network layer), 4320 data instances are collected, each for ping, latency, liveliness, and 60 data instances for bandwidth from every Micro-Box. Similarly, for BPF based tracing (flow layer), 8640 data instances are collected from each resource amounting to approximately 1.8/∼4.5 GB

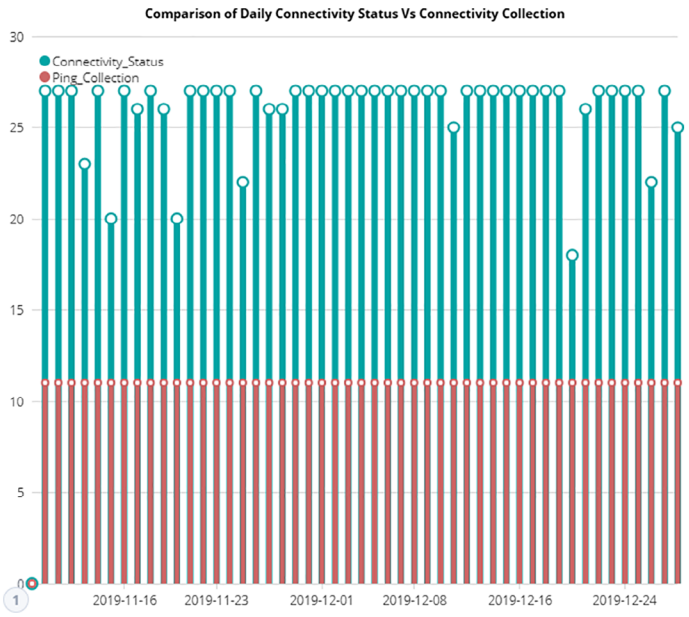


FIGURE 11 Comparison of connectivity status with visibility data (ping) collection over time-line

in size. For "collectd" (physical layer) approximately 257 000 data instances are collected from each Micro-box. To assist network operators a concise visibility report is generated daily based on aggregation of visibility data from the past 24 hours. During the report generation, 1800 data instances and 210 files are generated for a month.

During the Observation period, we identify and resolve several operational issues related to maintaining persistent visibility such as limited connectivity, network overhead, and disk size. During limited connectivity, timely visibility collection at the Visibility Center is lost. Kafka-based producer though manages to stores data temporarily, but it falters during long connection outages or when broker is not available. To tackle this issue local temporary storage with tagging is enabled which retains the visibility data for 24 hours to prevent loss. Furthermore, collection failures are recorded to identify the frequency, time, and cause of failure. Next, to handle network overhead, and disk size due to a large amount of visibility data produced during flow layer measurement, we implement summarized collection. For troubleshooting, historical visibility data is enabled to assist playground operators.

After handling the above-mentioned issues, we observed the visibility data for one month as *production period* (ie, 10 July ~10 August). Figure 11 graphically represents the timeline of maintaining visibility data over unreliable network connectivity by comparing network connectivity status with the collection of visibility data (ping) received over a day. As can be seen, the horizontal coordinates represent date and vertical coordinates depict count of collection. In this section to incorporate time-series based visualizations, we use data visualization tools such as Elasticsearch, Kibana, and Timelion, enabling the visualization for near real-time as well as historic visibility data.

5.3 | Multi-agent and shared messaging interface for maintaining functionalities and visibility data

In correspondence to requirement R2, Agents are required to monitor, restart, and alert the status of running functionalities as well as Box-Agents. As mentioned in the design Section 4.2-second scenario, in case DB applications are not running Agents restart the service and log the event with downtime at shown in Figure 12B. Operators can utilize this report to troubleshoot functionality issues in a particular remote resource. Center-Agent additionally communicates with Box-Agent using messaging patterns. Center-agent then keeps track of running Box-Agents periodically through asynchronous messaging as shown in Figure 12A.

On receiving a request from Center-Agent, a reply is sent back to confirm the alive status of Agent. In case Box-Agent is down, Center-Agent logs the event and restarts the Box-Agent. Scalability in terms of increasing resources is handled by a configuration file containing a list of functionalities and sources to monitor. New Micro-Box or functionalities are included by adding the details in the configuration file. As shown in Figure 12B, at the Visibility Center, alerts are aggregated using data stored in MongoDB database, with tags such as time, functionality name, source box name, and total downtime.

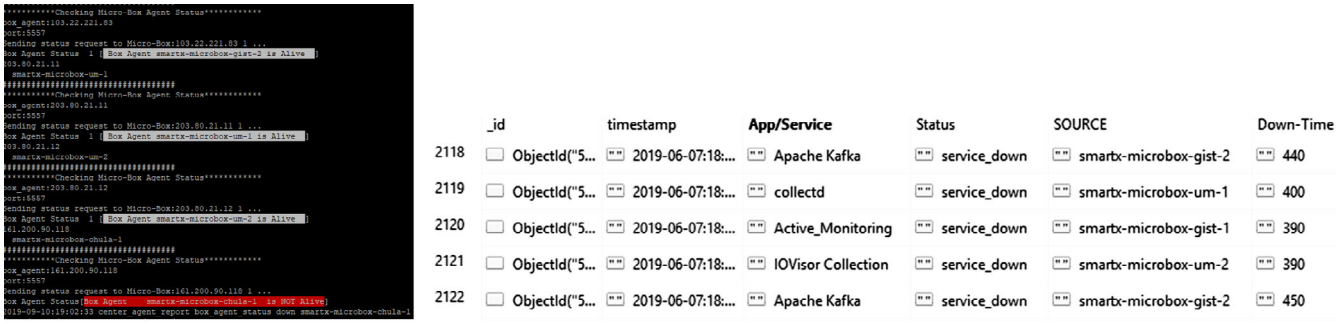


FIGURE 12 A, Report of Center-Agent checking status of Box-Agents. B, Status of running functionalities at Micro-Boxes

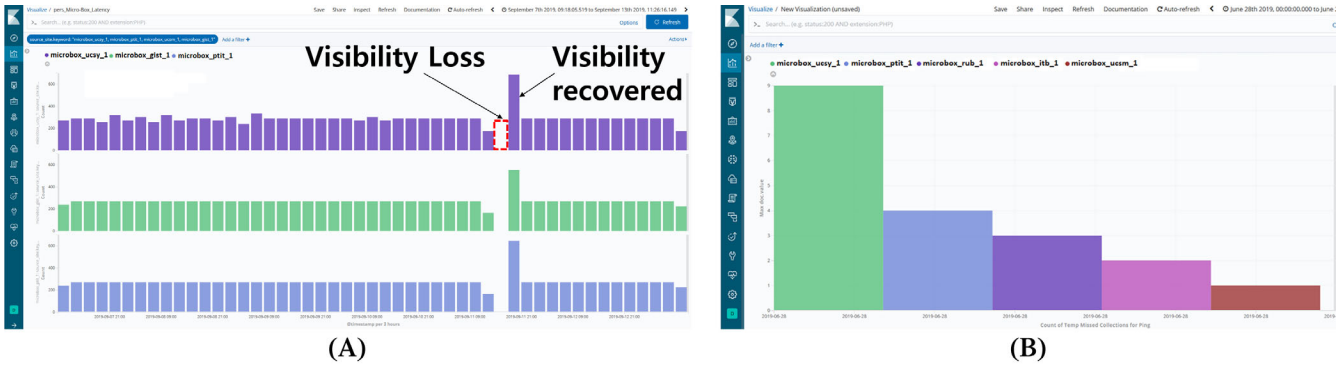


FIGURE 13 A, Visibility collections from Micro-Boxes. B, Count of measurements missed during 24 Hrs Collection

In Figure 13, the horizontal coordinates represent time-line on a daily basis while vertical coordinates shows the count of latency measurement collections for each of the Micro-Box. Figure 13A shows collection of visibility data (latency) for three Micro-Box on a time-series visualization, to verify persistent transfer with no data loss under network connectivity loss described in first Scenario of Section 5.3. As shown in Figure 13A, at August 23, 2019, 09:00 visibility collection is stopped due to loss in network connection. Once the connection is restored at approx. 21:00, lost visibility collection (approximately 700) is delivered at the Visibility Center. Figure 13B provides another view for number of visibility data loss per day on a time-series visualization.

5.4 | Flow summarization and optimal polling for maintaining network and storage load

This section caters to requirement R3. It is observed that during packet precise collection, visibility data may induce network load and consume large storage at the Visibility Center. To verify persistence collection, we compare the network usage and storage allocation for unmodified SmartX MVF with the proposed summarized collection. Figure 14 shows the comparison of volume generated during packet tracing and proposed scheme of flows generation at the Micro-box. There is a reduction of visibility data from packet tracing during observation period (ie, month of June) compared with flows generation during production period (ie, 10 July) for selective Micro-Boxes.

For verification of timely delivery, we estimate the liveliness of Micro-Boxes with the Visibility Center and visualize them in parallel on a timeline. Figure 15 describes count of measurement collection in the vertical panel and daily timeline horizontally. As shown in this figure, visibility data is synchronized among all the Micro-Boxes with regular intervals.

Next, the benefits of summarized flows are analyzed as a percentage decrease in resource usage at both the SmartX Tower and Micro-Box. In summarized collection, packet precise collection is aggregated based on similarity of five tuples during five-minute period. In the summarized collection, as shown in Table 8, we can reduce single Micro-Box collection size to 94% and bandwidth usage to 84%.

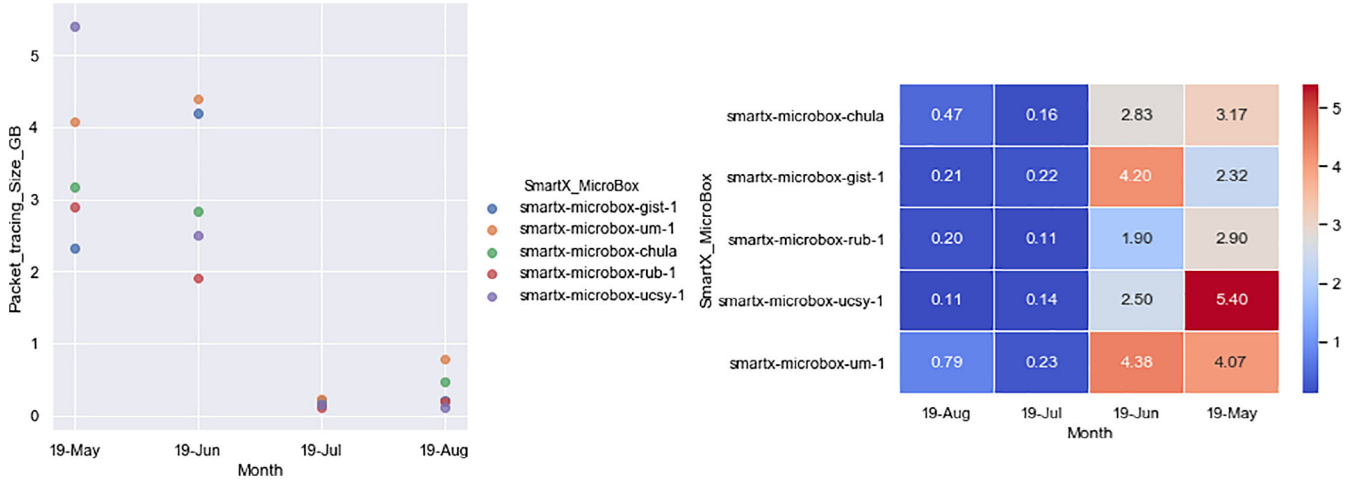


FIGURE 14 Visualization of volume generated during packet tracing and flows generation

FIGURE 15 In-sync and timely delivery of Visibility data

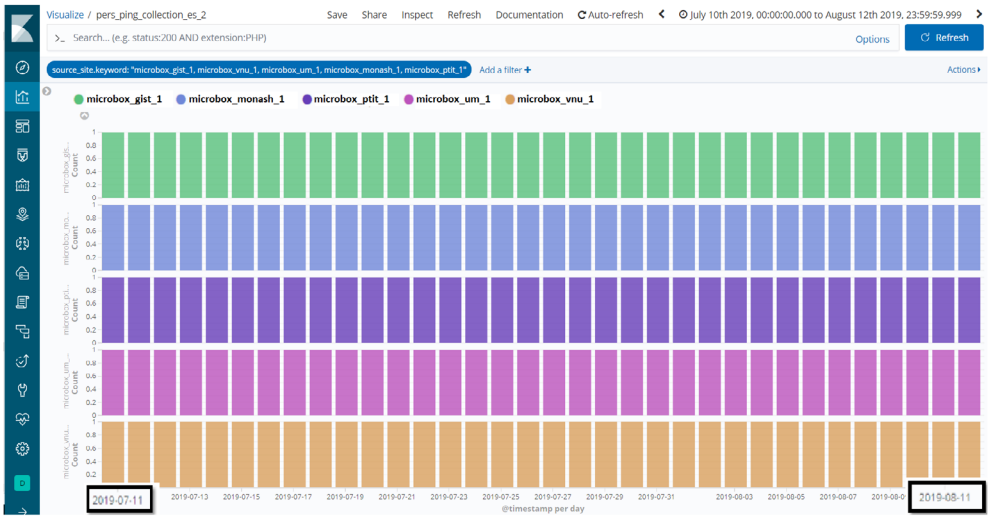


TABLE 8 Comparison of packet precise collection and summarized flow during normal and DoS attack scenario

Collection Type	# of Lines	File Size (MB)	Bandwidth Used (MB/Sec)	% Size reduction	% Bandwidth reduction
Packet precise collection	4118	0.432	0.4531	94.4%	84.6%
Summarized collection	165	0.024	0.06981		
Packet precise collection (DoS Attack)	483186	49	51	83.26%	83.29%
Summarized collection (Dos Attack)	56619	8.2	8.52		

Next, to evaluate the sustainability of proposed approach, we ran a denial-of-service attack (DoS), which is an attack executed by a single attacker to shut down the target service by flooding the victim's server using attacker's own computer and network. The purpose of this scenario is to test the packet collection capabilities by sending incomplete requests to keep the connection open. Usually, if the server gets overburdened with requests or connections, it is exhausted and can no new connection can be accepted anymore. We conduct an application-based DoS attack to test against several targets. We used Slowloris and GoldenEye open source tools, which are HTTP DoS attack that affects servers. We send requests periodically (every 15 seconds) to keep the connections open. If the server closes a connection, we keep on creating a new one. The test employs 100 workers as randomly generated user agents, sending HTTP request on 500 sockets. The

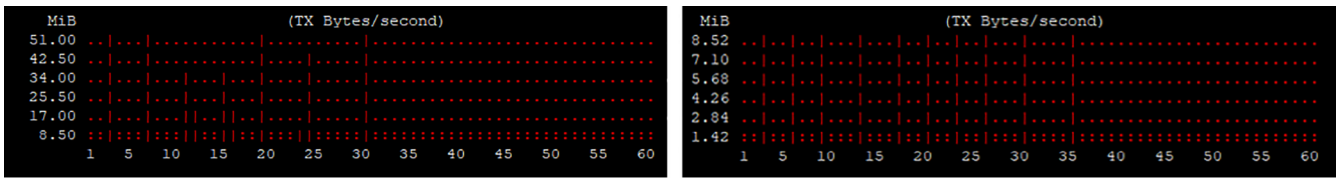
command pattern is #~python goldeneye.py <URL> -w 100 -d. We compare parameters such as bandwidth utilization and file storage size during this DoS attack scenario. As expected the resource consumption drastically increased, however, as depicted in Table 8, summarized collection reduces both the bandwidth and disk storage by about 83%.

Figure 16 shows disk size consumption during packet tracing collection and proposed summarized collection, both during Normal operations and Dos attack scenarios. As shown for Packet precise collection 49MB Bandwidth consumed, whereas 8MB bandwidth consumed for summarized collection during the DoS attack scenario. The result shows the proposed approach is capable to sustain in resource-intensive tasks.

In the next scenario, we evaluated the resource consumption overhead at the Visibility Center by observing the packet precise collection and comparing the bandwidth and storage parameters for a day (24 hours). In the playground, collections are stored at the Visibility Center from distributed Micro-Boxes. These collections from multiple sources are then combined with the help of integration routines. As shown in Table 9, over a 24-hour period, 162 MB of visibility data is collected at the Visibility Center. In comparison, summarized collection caused a significant 96.91% reduction in data storage consumption.

Next, we evaluate the optimal set of parameters configuration for performing visibility collection and sending it to the Visibility Center. In the SmartX Multi-View visibility environment, even though there is no single best value of parameters for visibility collection; still, the visibility collection aim is to maintain a persistent-level of resource consumption at an optimal level. Five sets of proposed parameters are defined and tested, where each set of polling consists of tests related to active and passive monitoring as shown in Table 10.

These parameters varied from high to low polling intervals. Visibility collection should balance resource consumption, to guarantee the best trade-off between minimizing monitoring overhead and polling delay, we selected a polling interval that adapts parameter settings to the characteristics of low resource consumption. For computing network bandwidth during comparison, we utilized vnstat4. It is a light, minimal resource usage, network traffic monitoring tool that keeps a log of network traffic for selected traffic and gives summarized historical output. We used daily network average, in this case, using command "vnstat -d -i eno1." As shown in Table 11, different sets of measurement parameters are applied to



(A) (B)

FIGURE 16 A, Bandwidth consumed for packet precise collection (during DoS attack). B, Bandwidth consumed for summarized collection (during DoS attack)

TABLE 9 Comparison of packet precise collection and summarized flow for 24-hour

	File Size (MB)	Bandwidth Used (MB/Sec)	% Size reduction	% Bandwidth reduction
Packet precise collection	162	84.25	96.91%	94.04%
Summarized collection	5.01	5.02		

TABLE 10 Comparison of multiple polling intervals for measurement

	Polling#1	Polling#2	Polling#3	Polling#4	Polling#5
Micro-Box	Micro-Box-chula-1	Micro-Box-monash-1	Micro-Box-itb-1	Micro-Box-um1	Micro-Box-um2
Ping/latency	20 min	10 min	5 min	2 min	1 min
Bandwidth	24 Hour	12 Hour	6 Hour	1 Hour	30 min
Collectd	20 sec	10 sec	5 sec	3 sec	2 sec
IO Visor	10 min	5 min	3 min	2 min	1 min

TABLE 11 Bandwidth comparison of multiple polling intervals for measurement

Polling Intervals	Polling#1	Polling#2	Polling#3	Polling#4	Polling#5
Avg. rate for Network Bandwidth	1.33 Mbit/s	1.15 Mbit/s	7.74 Mbit/s	17.85 Mbit/s	22.72 Mbit/s

distributed boxes. Based on the network bandwidth, the polling interval # 2 is a comparatively optimal option to schedule measurement tests.

5.5 | Maintaining SmartX multi-view visibility at the tower

To verify requirement R4 and to emphasize the verification of maintaining multi-view visibility data on daily-basis for distributed Micro-Boxes, Figure 17 shows comparison of visibility collection during training and production period. After maintaining persistent visibility during the test period, in terms of collection rate and size improvement (ie, from 91%~99% to 99%~100%) is observed at stable sites for multi-view visibility collection. This persistent visibility is vital for troubleshooting playground operational issues.

Next, we develop a customized tool to aggregate one-day visibility data as summarized Daily visibility data. As shown in Figure 18A, to verify the persistence visibility, collections from DataLake are counted for the one day and compared with expected collections count (ie, 288) or to estimate the percentage of successful collection. These results are stored in MongoDB and Elasticsearch for report generation and visualization respectively. The generated summarized report is disseminated as an automated daily email to the playground operators and developers group.

For visualizing collection of multiple measurements (liveliness, ping, latency etc.) at the Data-lake as time-series visibility data, we developed a dashboard. Figure 18B shows collection of latency measurement across two sites (GIST-1, Korea, and UM-1, Malaysia) on time-line for over a month period. As shown the horizontal values describe measurement of latency (ms) values, whereas the vertical coordinates represent daily time-line for these measurements. Experiment shows a steady measurement collection for both sites for the entire-period. Thus enabling us to maintain the collection in response to requirement R4.

For management and utilization of playground operations by end-user on IoT devices, we developed a cloud-native service through Kubernetes and distributed it through the OF@TEIN + playground. One Kubernetes cluster was formed with Edge IoT-gateways distributed at Multi-site of OF@TEIN+ playground. Figure 19 shows the listing of Kubernetes cluster nodes on P+O (provisioning + orchestration) Center residing at Playground Tower.

To facilitate OF@TEIN+ playground developers and operators with visualization and to verify multi-view visibility from distributed Micro-Boxes, we leveraged onion-ring visualization as shown in Figure 20. Visualization dashboard enables the playground operators and developers to verify the overlay-based playground topology and to check association with the underlay WAN networks belonging to separate network operators. From the inner ring to the outer one, the underlay resource layer consists of international network PoPs (TEINSG and TEINHK), national PoPs (eg, THAIREN,

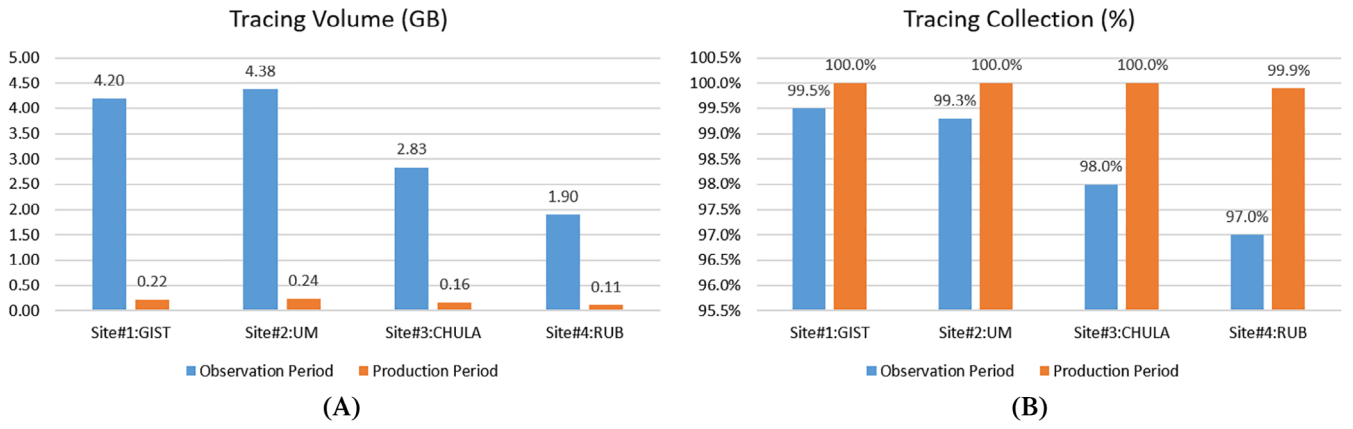


FIGURE 17 A, Results for tracing volume experiment. B, Results for tracing collection experiment

IOVISOR Daily Collection

Date	Total Expected Collection	microbox-gist-1	microbox-gist-2	microbox-um-1	microbox-um-2	microbox-chula-1	microbox-itc-1	microbox-vnu-1
2019/09/20	288	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%

Liveliness of Box with Visibility Center

Date	Source	microbox-gist-1	microbox-gist-2	microbox-um-1	microbox-um-2	microbox-chula-1	microbox-itc-1	microbox-vnu-1
2019/09/20	Visibility_Center	100.0%	100.0%	97.92%	97.92%	96.53%	83.33%	98.61%

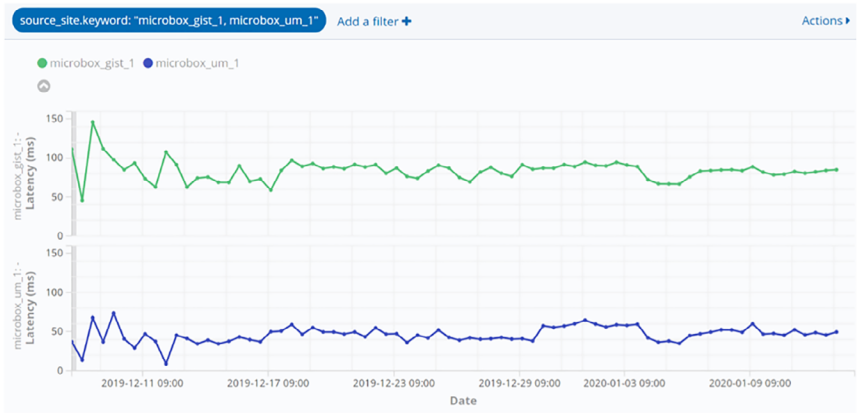
PING Generated Daily Collection

Date	Total Expected Collection	microbox-gist-1	microbox-gist-2	microbox-um-1	microbox-um-2	microbox-chula-1	microbox-itc-1	microbox-vnu-1
2019/09/20	144	144	144	144	144	144	144	144

Daily Uptime(percentage) Report based on Ping

Date	Total Expected Collection	microbox-gist-1	microbox-gist-2	microbox-um-1	microbox-um-2	microbox-chula-1	microbox-itc-1	microbox-vnu-1
2019/09/20	microbox-gist-1	-	100.0%	98.61%	99.31%	97.92%	79.86%	98.61%

(A)



(B)

FIGURE 18 A, Results for summarized collection using daily visibility report. B, Time-series visualization for latency measurements of micro-boxes

```
tein@tower-gist-po-center:~$ kubectl get no -o wide
NAME                        STATUS     ROLES    AGE    VERSION   INTERNAL-IP       EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION      CONTAINER-RUNTIME
smartx-microbox-chula-1     Ready      <none>   246d   v1.14.1   161.200.90.118    <none>        Ubuntu 18.04.2 LTS   4.15.0-47-generic   docker://18.9.2
smartx-microbox-gist-1      Ready      <none>   251d   v1.14.1   103.22.221.85     <none>        Ubuntu 18.04.2 LTS   4.15.0-62-generic   docker://18.9.7
smartx-microbox-gist-2      Ready      <none>   251d   v1.14.1   103.22.221.83     <none>        Ubuntu 18.04.2 LTS   4.15.0-62-generic   docker://18.9.7
smartx-microbox-itb-1       Ready      <none>   218d   v1.14.1   167.205.51.41     <none>        Ubuntu 18.04.2 LTS   4.15.0-66-generic   docker://18.9.7
smartx-microbox-itc-1       NotReady   <none>   202d   v1.14.1   203.176.131.65    <none>        Ubuntu 18.04.2 LTS   4.15.0-54-generic   docker://18.9.7
smartx-microbox-um-1        Ready      <none>   251d   v1.14.1   203.80.21.11      <none>        Ubuntu 18.04.2 LTS   4.15.0-51-generic   docker://18.9.7
smartx-microbox-um-2        Ready      <none>   248d   v1.14.1   203.80.21.12      <none>        Ubuntu 18.04.2 LTS   4.15.0-51-generic   docker://18.9.7
tower-gist-po-center        Ready      master   251d   v1.14.1   103.22.221.51     <none>        Ubuntu 18.04.2 LTS   4.15.0-43-generic   docker://18.9.2
```

FIGURE 19 Listing Kubernetes cluster nodes on P+O Center

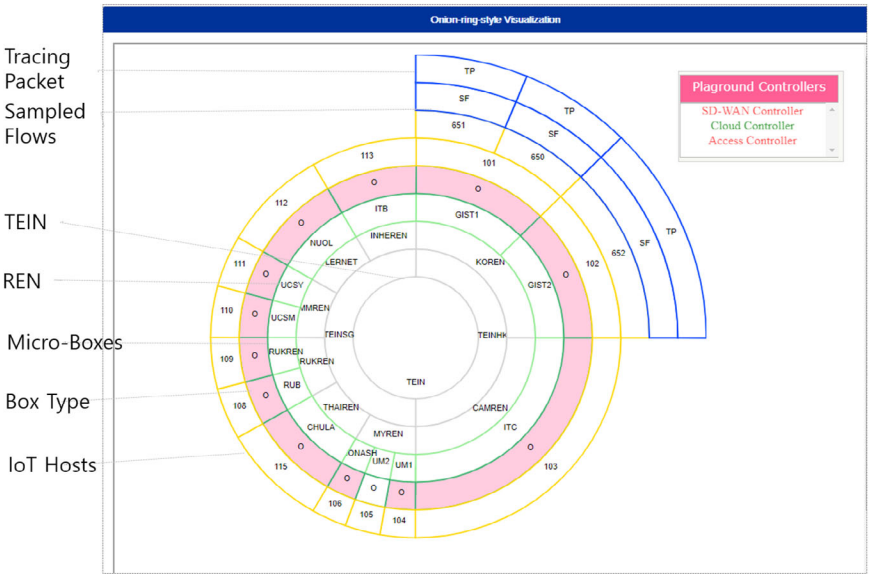


FIGURE 20 Onion-ring visualization for OF@TEIN+ playground

KOREN,Ĕ) and sites (eg, UM, CHULA,Ĕ). On top of sites, physical boxes are shows. Next ring shows the containerized resources with their IDs. The blue ring shows samples flows and traced packet of each containerized resource.

6 | CONCLUSION

In this article, we proposed, demonstrated, and evaluated distributed cloud-native edge box capable of maintaining persistent multi-view visibility in OF@TEIN+ playground. We introduced an approach to persistently maintain SmartX Multi-view visibility collection at a centralized location from cloud-native edge boxes utilizing open source tools. The proposed design increases the resilience and improves distributed resource visibility to maintain data against loss during resource and connectivity failures. The lightweight network packet-precise flows collection component in the proposed solution minimizes overall network load and storage by improving from SmartX MVF. We have operated and validated the presented approach through visualization support which includes time-series interactive dashboards that are created in Kibana and secondly through onion-ring visualization. Validation of persistent visibility data for specific time-period as aggregation is presented through a multi-view daily visibility report. As future work, we want to evaluate the Spatio-temporal summarized visualization of SmartX multi-view visibility for OF@TEIN+ distributed cloud-native edge boxes. We also want to add new APIs to the solution and explore the possibility of using temporal summarization to detect anomalies. To facilitate multi-view visualization, we would include the collection of multiple layers in a synchronized and summarized way.

ACKNOWLEDGEMENTS

K-ONE project: This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2015-0-00575, Global SDN/NFV OpenSource Software Core Module/Function Development). Cybersecurity project: This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2017-0-00421, Cyber Security Defense Cycle Mechanism for New Security Threats).

ORCID

Muhammad Ahmad Rathore https://orcid.org/0000-0002-0461-2501
Muhammad Usman https://orcid.org/0000-0002-9598-0704

REFERENCES

1. Jonathan A, Uluyol M, Chandra A, Weissman J. Ensuring reliability in geo-distributed edge cloud. Paper presented at: Proceedings of the 2017 Resilience Week (RWS) 2017 Sep 18; 2017:127-132; Wilmington, DE: IEEE.
2. Kratzke N, Quint P-C. Understanding cloud-native applications after 10 years of cloud computing-a systematic mapping study. *J Syst Softw*. 2017;126:1-16.
3. Risdianto AC, Usman M, Kim JW. SmartX box: virtualized hyper-converged resources for building an affordable playground. *Electronics*. 2019;8(11):1242.
4. Kim J, Cha B, Kim J, et al. OF@ TEIN: an OpenFlow-enabled SDN testbed over international SmartX rack sites. *Proc Asia-Pacific Adv Netw*. 2013;36:17-22.
5. Risdianto AC, Kim NL, Shin J, et al. OF@ TEIN: a community effort towards open/shared SDN-cloud virtual playground. *Proc Asia-Pacif Adv Netw*. 2015;40:22-28.
6. Usman M, Risdianto AC, Han J, Kang M, Kim JW. SmartX multiview visibility framework leveraging open-source software for SDN-cloud playground. Paper presented at: Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft 2017); Jul 3; 2017:1-4; Bologna, Italy: IEEE.
7. Usman M, Ahmad RM, Kim JW. SmartX multi-view visibility framework with flow-centric visibility for SDN-enabled multisite cloud playground. *Appl Sci*. 2019;9(10):2045.
8. Usman M, Risdianto AC, Han J, Kim J. Interactive visualization of SDN-enabled multisite cloud playgrounds leveraging smartx multiview visibility framework. *Comput J*. 2018;62(6):838-854.
9. Tawrit Muhammad. *NREN Sustainability:A Realistic Roadmap. The 3rd Asi@ConnectGovernors' and Project Meetings. Technical Report*. Singapore: Asi@Connect; 2018.
10. Mesbahi MR, Rahmani AM, Hosseinzadeh M. Reliability and high availability in cloud computing environments: a reference roadmap. *Human-Centr Comput Inf Sci*. 2018;8(1):20.
11. Rathore MA, Risdianto AC, Nam T, Kim JW. *Comparing IO Visor and Pcap for Security Inspection of Traced Packets from SmartX Box*. New York, NY: Springer; 2017:1263-1268.

12. Usman M, Kim JW, Mạnh NT. Multi-belt onion-ring visualization of OF@TEIN testbed for SmartX multi-view visibility. Proceedings of 2019 the 9th International Workshop on Computer Science and Engineering WCSE_2019_SPRING, Yangon, Myanmar; 2019:6–10.
13. Nobre JC, Mozzaquatro BA, Granville LZ. Network-wide initiatives to control measurement mechanisms: a survey. *IEEE Commun Surv Tutor*. 2018;20(2):1475-1491.
14. Usman Muhammad, Risdianto Aris Cahyadi, Han Jungsu, Kim JongWon, Van Huynh, N. Physical-virtual topological visualization of OF@ TEIN SDN-enabled multi-site cloud. 2017 International Conference on Information Networking (ICOIN) 2017 Jan 11, 2017:622-624; IEEE.
15. Risdianto AC, Thet PM, Iqbal A, et al. Deploying and evaluating OF@ TEIN access center and its feasibility for access federation. *Proc Asia-Pacif Adv Netw*. 2016;42:34-40.
16. Smit M, Simmons B, Litoiu M. Distributed, application-level monitoring for heterogeneous clouds using stream processing. *Future Generat Comput Syst*. 2013;29(8):2103-2114.
17. Gao H, Xu Y, Yin Y, Zhang W, Li R, Wang X. Context-aware QoS prediction with neural collaborative filtering for Internet-of-Things services. *IEEE IoT J*. 2020;7(5):4532-4542. https://doi.org/10.1109/JIOT.2019.2956827.
18. Gao H, Duan Y, Shao L, Sun X. Transformation-based processing of typed resources for multimedia sources in the IoT environment. *Wirel Netw*. 2019;11276():1-17.
19. Gao H, Huang W, Duan Y. The Cloud-edge based dynamic reconfiguration to service workflow for mobile ecommerce environments: a QoS prediction perspective. *TOIT*. 2020;1(1). https://doi.org/10.1145/3391198.
20. Montes J, Sánchez A, Memishi B, Perez MS, Antoniu G. GMonE: a complete approach to cloud monitoring. *Future Generat Comput Syst*. 2013;29(8):2026-2040.
21. Katsaros G, Kübert R, Gallizo G. Building a service-oriented monitoring framework with rest and nagios. Paper presented at: Proceedings of the 2011 IEEE International Conference on Services Computing; 2011:426-431; Washington, DC: IEEE.
22. Katsaros G, Gallizo G, Kübert R, Wang T, Fitó JO, Espling D. *An Integrated Monitoring Infrastructure for Cloud Environments*. New York, NY: Springer; 2011:149-164.
23. Tordsson J, Djemame K, Espling D, et al. *Towards Holistic Cloud Management*. Newcastle upon Tyne: Cambridge Scholars; 2012.122–150.
24. Iannaccone G, Diot C, McAuley D, Moore A, Pratt I, Rizzo L. The CoMo white paper. *INTEL RESEARCH TECHNICAL REPORT IRC-TR-04-17*. Cambridge: Intel Research; Technical Report IRC-TR-04-17, Intel Research; 2004.1–8.
25. Brandón Á, Pérez María S, Montes J, Sanchez A. Fmone: a flexible monitoring solution at the edge. *Wireless Communications and Mobile Computing*. 2018;2018(2):8–15.
26. Risdianto AC, Kim J. Prototyping Media Distribution Experiments Over OF@ TEIN SDN-Enabled Testbed. *in Proc. APAN Network Research Workshop, Nantou, Taiwan*; 2014;38:12-18.
27. Kachris C, Soudris D, Gaydadjiev G, et al. *The VINEYARD Approach: Versatile, Integrated, Accelerator-Based, Heterogeneous Data Centres*. New York, NY: Springer; 2016:3-13.
28. Kreps J, Narkhede N, Rao J. Kafka: a distributed messaging system for log processing. *Proc NetDB*. 2011;11:1-7.
29. Corbet Jonathan. Extending extended BPF. Technical Report. N/A: Jonathan Corbet; 2014. https://lwn.net/Articles/603983/. Accessed July 07, 2019.
30. Syed Affan Ahmed. *Exploring eBPF, IO Visor and Beyond. Technical Report*. 2016. https://www.iovisor.org/blog/2016/04/12/exploring-ebpf-io-visor-and-beyond. Accessed August 01, 2019.
31. Gregg Brendan. *Linux 4.x Tracing Tools: Using BPF Superpowers*. Boston,MA: usenix, LISA16; 2016.
32. Syed AA. Exploring eBPF, IO Visor and Beyond; 2016 Accessed May 7, 2020.

How to cite this article: Rathore MA, Usman M, Kim JW. Maintaining SmartX multi-view visibility for OF@TEIN+ distributed cloud-native edge boxes. *Trans Emerging Tel Tech*. 2020;e4101. https://doi.org/10.1002/ett.4101