Kaala: Scalable, End-to-End, IoT System Simulator

Udhaya Kumar Dayalan, Rostand A. K. Fezeu, Timothy J. Salo, Zhi-Li Zhang
{dayal007,fezeu001,salox049,zhang089}@umn.edu
Department of Computer Science & Engineering, University of Minnesota – Twin Cities, USA

ABSTRACT

We introduce *Kaala*, a scalable, hybrid, end-to-end IoT system simulator that can integrate with diverse, real-world IoT cloud services. Many IoT simulators run in isolation and do not interface with real-world IoT cloud systems or servers. This isolation makes it difficult for experiments to fully replicate the diversity that exists in end-to-end, real-world systems. Kaala is intended to bridge the gap between IoT simulation experiments and the real world. The simulator can interact with cloud IoT services, such as those offered by Amazon, Microsoft and Google. Kaala leverages vendor-provided software development kits (SDKs) to implement the vendor-specific protocols that are necessary permit simulated IoT devices and gateways to seamlessly communicate with real-world cloud IoT systems. Kaala has the ability to simulate a large number of diverse IoT devices, as well as to simulate events that may simultaneously affect several sensors. Evaluation results show that Kaala is able to, with minimal overhead, seamlessly connect simulated IoT devices to real-world cloud IoT systems.

CCS CONCEPTS

• **Networks** → **Network simulations**; • **Computing methodologies** → *Simulation environments*; • **Software and its engineering** → *Simulator / interpreter*.

KEYWORDS

IoT devices, IoT simulator, IoT Gateway SDK, IoT cloud, 5G, Network

ACM Reference Format:
Udhaya Kumar Dayalan, Rostand A. K. Fezeu, Timothy J. Salo, Zhi-Li Zhang. 2022. Kaala: Scalable, End-to-End, IoT System Simulator. In *NET4us '22: ACM SIGCOMM 2022 Workshop on Networked Sensing Systems for a Sustainable Society (NET4us '22), August 22, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3538393.3544937

1 INTRODUCTION

The proliferation of IoT devices in recent years has made it possible to develop innovative smart services for homes, offices, businesses, cities and communities. Large cloud service providers, such as Amazon, Microsoft and Google, offer cloud-based IoT data analytics and AI services for collecting, storing and processing the massive amounts of IoT data these devices generate. Because these new cloud IoT services often obviate the need for IoT device vendors to deploy their own data centers, cloud services have become integral components of most IoT systems.

Cloud IoT service vendors, such as AWS [1], Azure [4], Google [7] and Alibaba, all aim to build their own IoT ecosystems, which currently do not interoperate with each other. According to the UNIFY IoT project, more than 360 IoT companies exist today [13]. While there are industry-led efforts to ensure interoperability between cloud IoT services (e.g., via CHIP [6]), non-interoperable cloud IoT services are likely to remain the rule, rather than the exception, for some time.

Prototyping, testing and evaluating new IoT devices and systems can be expensive. Beyond the cost of the devices themselves, setting up, configuring and maintaining a diverse collection of physical IoT devices can quickly become time-consuming, cumbersome, unwieldy and expensive. As a result, simulators are often used to test and evaluate new product ideas and designs early in the development process. Since cloud services have become a critical component of many IoT systems, it is beneficial that these simulations be able to evaluate end-to-end systems, from the IoT devices to the IoT cloud services. However, simulating end-to-end IoT systems as shown in Fig. 1 is complicated by the vendor-specific nature of these cloud IoT services. Each vendor requires IoT devices to implement a different set of protocols to authenticate, manage, and collect data from these devices. As a result, IoT devices are often effectively locked into one cloud IoT service provider [8] [9].

An alternative to waiting until physical devices have been developed and constructed, is to use IoT simulators to test and evaluate prospective IoT devices, systems, and designs. If these simulators and experiments are designed properly, simulation can significantly reduce the gap between proof-of-concept (PoC) implementations and real world deployments [5]. Unfortunately existing IoT simulators are limited in their capabilities and scopes. 1) Many simulators are designed to run on a single laptop, desktop or a server, and are therefore poorly positioned for large-scale simulations that require significant computational power. 2) Most of simulators fail to capture the large variety and diversity of IoT devices that exist today. For example, many are tailored to simulating only small sensors with low bandwidth requirements, ignoring a variety of IoT devices (e.g., surveillance cameras and autonomous vehicles) that consume large amounts of network bandwidth and require real-time cloud connectivity. 3) Perhaps more importantly, existing IoT simulators operate in isolation: they interface with only a limited number of types of IoT devices and can not be integrated with existing cloud services. In short, existing IoT simulators cannot be used to effectively test and evaluate prototype IoT systems, especially those that require computationally intensive subsystems, such as machine learning algorithms, Cloud IoT services, or IoT control mechanisms running on edge computing facilities. Chernyshev et al. [5] in a recent survey highlighted that an all-in-one simulator capable of

NET4us '22, August 22, 2022, Amsterdam, Netherlands

ACM ISBN 978-1-4503-9392-8/22/08…$15.00
https://doi.org/10.1145/3538393.3544937

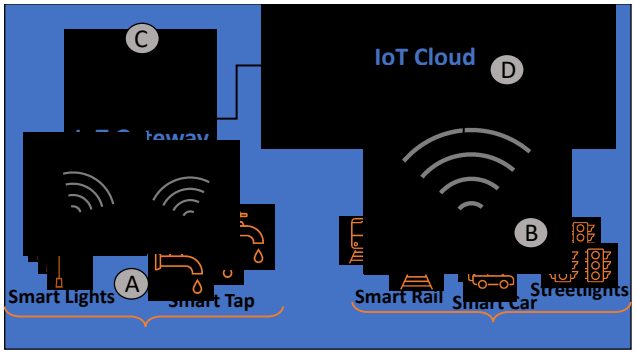

Figure 1: IoT (Edge) Devices, IoT Gateway and IoT Cloud

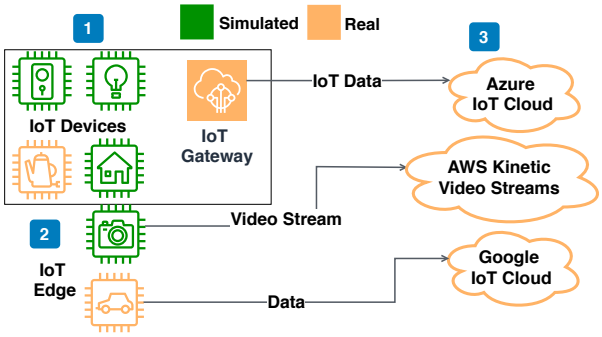

Figure 2: Kaala System Architecture

supporting an end-to-end IoT service has yet to be developed. New simulation tools and test-and-evaluation environments are sorely needed for unit testing and for the systematic evaluation of intelligent IoT services that include diverse, integrated devices, edge computing and cloud computing components.

In this paper, we present *Kaala* – a modeling, simulation and emulation platform that is capable of specifying IoT devices of various types, from low-powered sensors to smart IoT devices requiring high bandwidth, such as IoT devices that anticipate emerging 5G networks. In addition to simulating IoT devices, an important feature of Kaala is its ability to interface and connect with real-world cloud IoT services in an integrated fashion. The initial version of Kaala can use Amazon AWS [1], Microsoft Azure [4] and Google [7] IoT cloud services. The main design goal of Kaala is to help researchers and practitioners to quickly prototype various IoT scenarios, including those that use high-bandwidth 5G services, generate massive amounts of data, and to help bridge the gap that exists between simulators and real-world system. The major contributions of our paper are summarized below.

- (*Sec.* 2.1 & 4.1) We present Kaala: An IoT modelling and simulation platform that is able to specify IoT devices of various types to communicate with real-world cloud IoT systems, with AWS, Amazon and Google IoT Cloud platforms as case studies.
- (*Sec.* 2.2 & 4.2) Kaala is a scenario-based IoT simulator capable of mimicking various IoT scenarios such as "fire in a room or building" and "5G network capable data generation (including 4K/8K video IP cameras)" scenarios.
- Kaala is able to generate massive amounts of IoT data for prototyping data-intensive IoT applications.

The rest of the paper is organized as follow. We motivate the design and use cases of Kaala in *Sec.* 2. The design and implementation of Kaala are presented in *Sec.* 4 and *Sec.* 5, respectively. Kaala is evaluated in *Sec.* 6 and conclude the paper in *Sec.* 7.

2 CASE FOR KAALA

In this section, we use two case studies to argue the case for Kaala, while discussing their challenges.

2.1 Ability to Interact with Real Systems

Emergent IoT applications are extremely complex and operate in a very diverse IoT world. These are not just the smart speakers, smart thermostats, smart door locks in our homes, but also sensors used in domains like in the oil, gas and automobile industries. These applications support different systems connected through IoT. As shown in Fig. 3, there are no differences in data flow between real and simulated IoT devices. Researchers use simulators to study, prototype, test, and evaluate new IoT concepts and ideas. However, these simulators fail to reflect the complexity that exists in real IoT environments. For example, current IoT simulators do not simulate vendor-specific IoT devices [14]. The IoT simulator provided by AWS [3] can only simulate one type of IoT device. It simulates hard-coded IoT messages and does not simulate the network characteristics (TCP/IP stack) along with massive IoT data. To the best of our knowledge, current IoT simulators operate in isolation and do not interact with real cloud IoT system, failing to reflect the complexity present in the IoT world. Kaala is intended to remedy these deficiencies. Kaala simulates several vendor-specific IoT devices, (including those from AWS, Google, and Azure), and is able to connect to and communicate with real-world cloud IoT systems provided by these vendors. Kaala connects simulated IoT devices with real servers (within the complex IoT world), so that services provided by cloud service providers can be used, validated and verified. For instance, Kaala connects simulated IP cameras, temperature sensors, humidity sensors, and flame sensors to the Amazon's Kinetic video streams and builds logic around these sensors to simulate a fire event.

2.2 Scenario-based Data/Event Simulation

We use a *fire-in-a-building* event to make the case for the need of a scenario-based data generation and 5G service.

Fire in a building: IoT data generated by current IoT simulators are hard-coded [14]. That is, the data generated do not realistically models IoT data generated by real sensors. A more realistic approach might be to generate sensor data based on a distribution or based on the actual behavior of the sensor. Consider a *fire-in-a-room* scenario. When there is a fire in the building/room, the temperature in the room increases and the temperature sensor will report a higher value than usual. The smoke sensor will detect the smoke in the room and send the smoke alarm. The humidity in the room increases

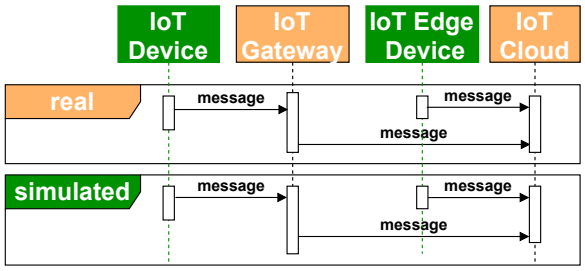

Figure 3: Sequence of data flow showing no differences in data flow between real and simulated IoT devices

and the humidity sensor will be sending the updated humidity value. Some sensors might malfunction, burn or lose connection because of the fire. Thus, relying a few sensors' data values will be problematic. We might want to analyze data from some other IoT devices (like a camera) to understand the prior and current situation of the fire event to call an ambulance. IoT simulators ought to model such scenarios. The inability of current IoT simulators to model real-world IoT scenarios present Kaala with a unique opportunity. We present more details of how Kaala achieves this in *Sec.* 4.2.

High-bandwidth data generation: IoT simulators ought to support next-generation network technologies such as 5G. With higher 5G throughput, next generation IoT applications should seamlessly adapt to 5G. However, this is not the case. This is due to the lack of tools (IoT Simulators) which foster the design, development and deployment of 5G-capable applications both on the client and server side. For instance, a video streaming service provider like YouTube or Netflix have millions of users watching videos. The throughput will depend on the quality of the video. Recently 8K videos require significantly higher bandwidth (5G speeds) to play a single frame compared to a 480p video quality. For example, a video of 'X' minutes requires 119 MB for a 240p video, 1038 MB for a 1080p video, and 7284 MB for an 8K video. Thus, prototyping 5G next generation IoT applications using an IoT simulator that can support the massive data workloads seen in production environment should be addressed. Current IoT simulators do not support modelling thousands of IoT devices with large amount of data. Kaala realises this challenge by simulating IP cameras that supports 8K video streaming and is able to scale to hundreds (and even thousands) as shown later in *Sec.* 6.

3 KAALA ARCHITECTURE

Fig. 2 shows the system design of Kaala. Kaala is able to integrate real and simulated devices while leveraging vendor-specific SDKs to connect them to real systems in the cloud. Next, we motivate and detail each layer in Kaala's architecture as shown in Fig. 4.

There are three main objective of Kaala. 1) Simulate various IoT devices, including vendor-specific IoT devices and connect them to particular vendor's cloud IoT services. 2) Simulate realistic data across all applicable devices to mimic real IoT service scenarios. 3) Generate real high-throughput data.

Network layer: The network layer connects all the other layers. It is a virtual network that models a real network. It contains core network components, like a DHCP server, a DNS server, a switch

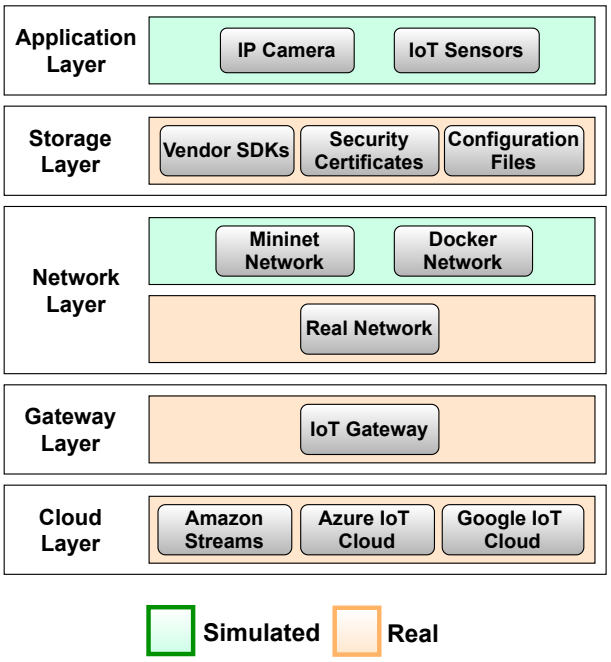

Figure 4: Kaala Layered Architecture

and a gateway. Each modelled physical device has a virtual IP address. This layer provides the resources needed for the gateway to connect to real systems using their domain names. Every instance of this layer creates a separate, isolated virtual network that can connect to real systems. Our evaluation results in *Sec.* 6 show quantitative statistics of network utilization for end-to-end IoT scenario experiments.

Cloud layer: This layer is the real cloud IoT systems that Kaala connects to. It authenticates, validates and accepts incoming connections and IoT data from the IoT gateway. The cloud layer provides core entities for massive data transformation, data analysis and interpretation. It provides data stream processing resources and cloud services for machine learning related tasks, business integration and user management. It is also responsible for notification and archival data storage.

Gateway layer: The IoT gateway connects Kaala to real cloud IoT systems. It runs vendor-provided SDKs, which contain the RESTFul APIs needed to connect to the the cloud layer. It serves as an MQTT broker (server) to IoT (Edge) devices and is a client that connects to the cloud. The MQTT broker receives IoT data from the network layer and translates the data into vendor-specific data formats via their SDKs. It retrieves security contexts from the storage layer, authenticates, validates and connects to the cloud layer to forward the IoT data to the IoT cloud.

Storage layer: The storage layer is composed of different vendor SDKs, security contexts, data storage and configuration files. Cloud service providers support different SDKs, RESTFul APIs, data formats, security mechanisms, certificates and keys. This layer is responsible of contacting the service providers and obtaining the updated certificates, keys and SDKs used by the gateway layer or the application layer for authentication and validation.

Application layer: This is the layer responsible for simulation configurations, parameters tuning, IoT device configuration, network configuration and experiment scenarios. The purpose of the application layer is to run application-specific logic. The application layer provides standard IoT device functionality, such as publishing messages but the architecture supports extension of IoT device-specific logic by inheriting from the base IoT device logic.

4 KAALA DESIGN

In this section, we discuss Kaala's design goals followed by a detailed design to guide our implementation. The key design goals for Kaala are four-fold: 1) connect simulated IoT devices to real cloud IoT services, 2) provide extensibility of IoT device characteristics for current and future IoT devices, 3) simulate real-time events coordinated across multiple IoT devices and 4) generate realistic data to support current and future technologies like 5G. In the next sub sections, we will discuss how Kaala achieves these design goals.

4.1 Interacting with Real-World Systems

To support connection with both simulated and real networks, we leverage Mininet [12] and docker [10]. Mininet has the capability to create various types of virtual networks using different types of switches and controllers, and each host will be running as a process. Docker has the capability to create virtual networks and each host will have its own container [10]. These Mininet and docker virtual devices have IP addresses assigned to them and therefore can connect to real physical networks. As discussed earlier in *Sec.* 2.1, we enable interoperation with real cloud IoT systems by integrating vendors' SDKs [4]. Both the Mininet and docker architecture supports running real application processes in the respective host instances. Using this, the simulated applications run the respective SDKs in the various hosts as processes. Since the host application runs as a process, the host has access to all the files stored in the machine in which the simulation framework is running. As a result, each host does not need to have the individual vendors' SDKs installed and can reuse the SDK installed in the machine in which the simulation framework is running. This architecture is highly scalable when compared to the architecture proposed by IoTNetSim in [14]. This is because IoTNetSim creates virtual machine for each of the simulated devices. Moreover, in IoTNetSim the SDKs need to be installed individually in each virtual machines. We compare the scalability of Kaala with IoTNetSim in *Sec.* 6. The key advantage of using Mininet is the ability to create a wide variety of multiple-network architectures based on a simple configuration [12]. Additionally, various network configurations with different types of controllers and switches can also be simulated and tested for IoT traffic.

Simulated IoT devices connect to the IoT Gateway and we cover the implementation details in *Sec.* 5. Earlier in *Sec.* 2, we motivated the need to simulate vendor-specific IoT devices. In our proposed simulation framework design, both generic IoT devices and vendor-specific IoT devices can be simulated. We isolated each of the IoT devices with its own network resources. As a result, simulated IoT device can express the characteristics of a real IoT device and also run application specific code for the respective simulated IoT device. The simulated vendor-specific IoT devices each run the respective vendor IoT device SDK to connect to the corresponding vendor-specific IoT Gateway or IoT Clouds.

4.2 Scenario-based Event Simulation

The design of Kaala supports most of the real-time scenarios. We have also discussed a couple of scenarios in Section 2.2. The basis of scenario-based simulation is to coordinate one or more simulated IoT devices to match values based on the scenarios at the same duration range. Kaala supports simulation of one or more scenarios either at the same time or in sequence. The fire-in-the-room scenario is a built-in scenario in the Kaala simulation framework. When there is a fire detected in the room, the smoke sensor detects the smoke, the door lock opens automatically, the temperature in the room increases, the humidity in the room increases as well and the IP camera in the room captures the video. The list of affected IoT devices and the respective values need to be configured in the scenario configuration file, which gets loaded while running the simulation framework.

Kaala supports the simulation of IP cameras and a default video will be played when the IoT device is started. During the fire event, a fire video can be specified in the configuration file and the simulated IP camera will start streaming the video with fire. By this design, various scenarios can be simulated in Kaala. For example, the scenario configuration feature in Kaala easily simulates the quality of video changing, based on the available network bandwidth or based on time.

Next, in order to simulate a high-bandwidth scenario, we design our simulated IP camera using a Real-Time Streaming Protocol (RTSP) server [11]. The server will listen in a port and the client will be listening in a different port. The producer of the video will connect to the server port and the consumer of the video will be connecting to the client port to play the video. Both the producer and consumer can be designed to run in any host, so that the traffic flows through the network. More about RTSP implementation and evaluation is discussed in Section 5 and Section 6.3.

5 IMPLEMENTATION

We used Mininet [12] and the Docker framework [10] to simulate the IoT devices. The simulator framework can simulate both generic IoT devices and vendor-specific IoT device. Since we are leveraging the Mininet framework, each IoT device runs in its own process and gets dedicated network resources. We used the NodeJS version of the Mininet framework to simulate the IoT devices. And for the Docker version, each IoT device runs in its own container. The generic IoT device uses the basic MQTT client and the vendor-specific IoT device uses the respective vendor specific client SDK to communicate to the IoT Gateway. The IoT Gateway information, including the authentication details required to connect to the vendor-specific IoT Gateway, are passed as parameters when starting the respective vendor-specific simulated IoT devices, so each device knows which IoT Gateway they need to connect, authenticate and communicate with. Each simulated IoT device also consists of a profile, which specifies the type of device it simulates and the list of properties associated with the IoT device. Using this

implementation, we were able to simulate vendor-specific IoT devices and at the same time, provide dedicated network resources to each of the IoT devices.

The simulation framework loads a configuration file during startup. This configuration files includes the list of IoT devices which needs to be simulated. And the support for new IoT devices can be easily added to Kaala by just adding a profile for the newly added IoT device. The new IoT device can either use the generic application logic which sends data periodically based on the configuration which is discussed next or implement its own application logic. Each IoT device entry in that list contains a list of properties and these properties can be easily extended for new properties. Some of the key properties include the name of the IoT device, the profile (light, HVAC, smoke sensor, etc.) of the IoT device, the type of device (generic or vendor-specific) and finally the list of properties and the respective time-interval to report to the IoT Gateway. If the device type is vendor-specific, then the authentication information including the certificates are passed as arguments through the proposed design.

There is a another configuration file, called the scenario configuration file. The main purpose of this configuration file is to specify when the scenario needs to be executed, the list of IoT devices properties which need to be included in the scenario and the values of the properties of those IoT devices during the scenario. The default values need to be specified at the end of the scenario-based configuration file, in order to complete the scenario. After a scenario finishes executing, the next scenario in the configuration file is executed or the same scenario is executed in a loop until the simulation is stopped.

As discussed in Section 4.2, there are two main components in simulating an IP camera, the producer and the consumer. The video that needs to be streamed in the IP camera needs to be configured in the simulation configuration file, along with the period of the stream. Based on the configuration, Kaala will connect to the RTSP server to send video data. The producer keeps producing the video to the RTSP server by connecting to the local RTSP server port and the consumers can consume the video by connecting to the client port of the RTSP server of the respective IP camera IoT devices.

The main advantage of Kaala over IoTNetSim is that IoTNetSim creates virtual images for each IoT device, but Kaala uses processes for each IoT device. We tried to perform additional performance evaluation with IoTNetSim, but due to the design of IoTNetSim, we were not able to control the periodic interval of the simulated sensors within the IoTNetSim framework.

6 EVALUATIONS

In this section, we seek to understand the scalability and performance of Kaala. We connect simulated IoT devices to a real networks (we use Amazon AWS as case study) and finally evaluate a scenario based simulation.

6.1 Scalability and Performance

Since the simulation are usually run in a machine by the developer or tester, we want to understand the scalability and performance of running Kaala in a regular machine. For the experimental setup, we used a Linux Virtual Machine (VM) which was allocated 4GB

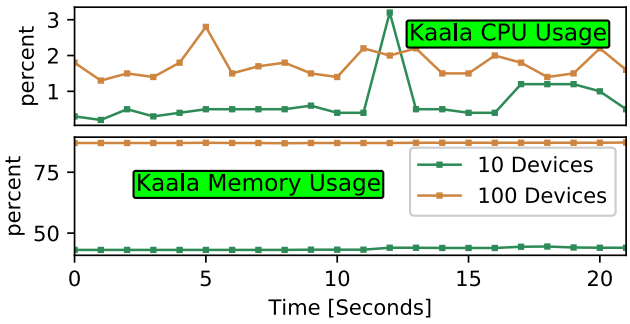

Figure 5: Kaala Performance Evaluation - Mininet

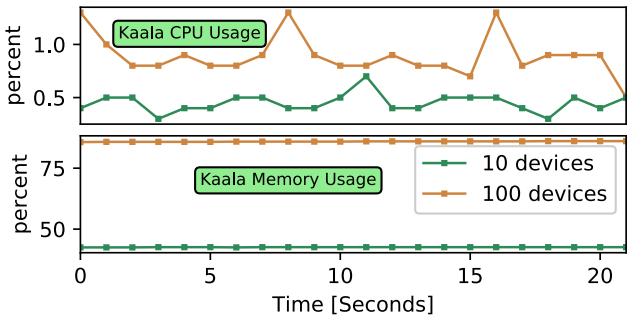

Figure 6: Kaala Performance Evaluation - Docker

memory, 2 processors and 20GB for storage. The VM was running Ubuntu. To understand the memory consumption and scalability capability of Kaala, we conducted two experiments; In the first and second experiment, we simulated 10 and 100 IoT devices respectively. The simulated IoT devices were a combination of temperature sensor, smoke sensor, humidity sensor, flame sensor and motion sensor. Each of these sensors publish their data based on their IoT device profile every 15 seconds. The data in Fig. 5 and Fig. 6 shows shows that as the number of sensors increases, the processor and memory usage increases significantly. Kaala is not just a simulator, it is an emulator as well. Each sensor runs its application logic in an individual process. So the processor and memory usage are expected to increase as the number of devices scale up. This is because as the number of IoT devices increases, more processes are created.

Since Kaala uses Mininet and the Docker framework to simulate IoT devices, for each IoT device, Mininet and the Docker framework create a network interface. So as the number of IoT devices increases, there is time taken to configure a new network interface in the host machine, get a new IP address and bring the interface up. The initial spike in both the memory and processor usage shows that both Mininet and the Docker framework consumes both memory and processor to create and setup the virtual network. An additional reason for the spike in resource usage is the spawning of processes or containers for each of the simulated IoT devices.

6.2 Interacting with Real Systems

For this experiment setup, we first created a user profile in the AWS IoT [2], and then configured an IoT device along with the necessary security certificates that are required to authenticate and connect

Features	IoTNetSim	Kaala
Vendor-Specific IoT Devices	No	Yes
MQTT Protocol Broker	No	Yes
Cloud Layer	Yes	No
Semi-Real IoT Devices	No	Yes
5G Capable Scenarios	No	Yes

Table 1: Comparison of Kaala with IoTNetSim

to the AWS IoT Cloud. The security certificates are downloaded in the host machine in which Kaala is running. Next, in the Kaala configuration, we specify that a vendor-specific IoT device needs to be simulated; the path to the downloaded security certificates are configured as well. These steps can be repeated for any number of vendor-specific devices. The same steps can also be followed for different vendors, like Microsoft Azure or Google IoT clouds, as well. Then, we start the simulation framework. Based on the loaded configuration, Kaala knows that a vendor-specific IoT device needs to be simulated and the application in the simulated devices will try to connect to the IoT gateway or IoT cloud using the provided security contexts. Once connected, the simulated IoT device will start publishing the data. Particularly, in this experiment, all the data is published to AWS IoT cloud.

6.3 Scenario-based Event Simulation

In this section, we assess the scenario-based event simulation in Kaala. First, for fire-in-the-room scenario, the necessary IoT devices were configured via the Kaala configuration file. This configuration file is also a pre-configured profile in Kaala. Initially, all the IoT devices will be publishing respective data to the IoT gateway or IoT cloud and the data will be related to normal operation in a room. In the scenario configuration file, the time to start the scenario based simulation will be specified. At that time, each IoT device property configured in the scenario configuration file will be configured with the value specified in the scenario configuration file. And, the values gets changed synchronously across all these IoT devices. The flame sensor will report 'true' stating that a flame has been detected. The temperature sensor shows a significant increase in the current temperature. Also, a video in which fire is shown is played exactly at the same time. Additionally, there can be two fire sub-scenarios. The first one is to make the motion sensor detect a movement in the room, while the second one has the motion sensor not sensing any movement or person in the room. This will help to validate scenarios like what happens when a person is in the room during the fire event and what happens when there is no person inside the room when the fire event occurs.

Next, sending high-bandwidth data was validated. As discussed in section 4.2, Kaala have an RTSP server running when simulating an IP camera IoT device. The server and client port of the RTSP server are configurable in the IP camera IoT device profile. As of now, the simulated IP camera IoT device can run server and client on the same port number. This can be a potential future work to support different ports for different IP camera IoT devices. An 8K video is sent to the simulated IP camera by connecting to the server port of the RTSP server. And, the consumer consumes the video by connecting to the client port of the RTSP server which is running in the simulated IP camera IoT device. Both the producer and consumer were run in a host other than the IP camera IoT device, so that the traffic is flowed in the network. The video being played by the producer can be completely controller by the application using the simulator configuration file. Additionally, the timing of different scenarios can be controlled and configured by the scenario-based configuration file. We also simulated a scenario in which the quality of video changes over time. For example, for the first 30 minutes, the producer was configured to produce 8K video and then for the next 30 minutes a 4K video was produced. This proves that Kaala is capable of simulating videos of different qualities to test different scenarios of streaming applications.

7 CONCLUSIONS

We have presented *Kaala* – a modelling, simulation and emulation platform that are capable of creating IoT devices of various types. Using our proposed simulation framework, we were able to simulate multi-vendor specific IoT devices in a single simulation framework. We also simulated real-time events like fire in a room/building scenario and evaluated how this work can be extended for other real-time scenarios. We were able to simulate devices which can generate large amount of data to verify and validate 5G technology.

ACKNOWLEDGMENTS

This research was in part supported by NSF under grants CNS-1814322, CNS-1831140, CNS-1836772, CNS-1901103, CNS-2106771 and CNS-212848.

REFERENCES

[1] AWS. 2018. Run Lambda functions on the AWS IoT Greengrass core - AWS IoT Greengrass. https://docs.aws.amazon.com/greengrass/latest/developerguide/lambda-functions.html
[2] AWS. 2020. aws/aws-iot-device-sdk-embedded-C. https://github.com/aws/aws-iot-device-sdk-embedded-C
[3] AWS. 2022. IoT Device Simulator. https://aws.amazon.com/solutions/implementations/iot-device-simulator/
[4] Microsoft Azure. 2020. Azure/azure-iot-sdks. https://github.com/Azure/azure-iot-sdks
[5] Maxim Chernyshev, Zubair Baig, Oladayo Bello, and Sherali Zeadally. 2017. Internet of things (iot): Research, simulators, and testbeds. *IEEE Internet of Things Journal* 5, 3 (2017), 1637–1647.
[6] CHIP. 2022. Project Connected Home over IP. https://www.connectedhomeip.com/
[7] Google Cloud. 2019. Overview of Internet of Things | Solutions | Google Cloud. https://cloud.google.com/solutions/iot-overview
[8] Udhaya Kumar Dayalan, Rostand A. K. Fezeu, Nitin Varyani, Timothy J. Salo, and Zhi-Li Zhang. 2021. ECIoT: Case for an Edge-Centric IoT Gateway. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications* (Virtual, United Kingdom) *(HotMobile '21)*. Association for Computing Machinery, New York, NY, USA, 154–156. https://doi.org/10.1145/3446382.3448667
[9] Udhaya Kumar Dayalan, Rostand A. K. Fezeu, Nitin Varyani, Timothy J. Salo, and Zhi-Li Zhang. 2021. VeerEdge: Towards an Edge-Centric IoT Gateway. In *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 690–695.
[10] Docker. 2022. Get Started with Docker. https://www.docker.com/get-started
[11] IETF. 2022. Real Time Streaming Protocol (RTSP). https://tools.ietf.org/html/rfc2326
[12] Mininet. 2022. Mininet - An Instant Virtual Network on your Laptop (or other PC). http://mininet.org/overview/
[13] UNIFY-IoT Project. 2016. Deliverable D03.01, Report on IoT platform activities. https://docbox.etsi.org/SmartM2M/Open/AIOTI/IoTPlatformsAnalysisToImprove/D03_01_WP03_H2020_UNIFY-IoT_Final.pdf
[14] Maria Salama, Yehia Elkhatib, and Gordon Blair. 2019. IoTNetSim: A modelling and simulation platform for end-to-end IoT services and networking. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. 251–261.