V-PRISM: An Edge-Based IoT Architecture to Virtualize Multimedia Sensors

Anselmo Luiz Éden Battisti
MídiaCom Lab / Institute of Computing
Universidade Federal Fluminense
Niterói, RJ, Brazil
anselmo@midiacom.uff.br

Débora Christina Muchaluat-Saade
MídiaCom Lab / Institute of Computing
Universidade Federal Fluminense
Niterói, RJ, Brazil
debora@midiacom.uff.br

Flávia C. Delicato
MídiaCom Lab / Institute of Computing
Universidade Federal Fluminense
Niterói, RJ, Brazil
fdelicato@ic.uff.br

***Abstract*— Multimedia sensors have recently become a major data source, giving rise to the Internet of Multimedia Things. Since multimedia applications are usually latency-sensitive, data processing in the cloud is not always effective. A strategy to minimize delay is to process the streams closer to the data sources, exploiting the resources at the edge of the network. We propose V-PRISM, an architecture to virtualize multimedia sensors with components deployed and executed at the edge tier. V-PRISM can reduce the resource consumption of IoT devices, the network traffic, and the end-to-end delay while increasing the ROI (Return On Investment) for infrastructure providers.**

Keywords*— *Virtual Multimedia Sensors, Internet of Things, Edge Computing

I. INTRODUCTION

With the proliferation of the Internet of Things (IoT) and its integration with cloud computing, a new paradigm called Cloud of Things (CoT) has recently emerged [1], which exploits the synergy between IoT and the cloud. In this scenario, environments, people and objects are continuous sources of data generation that are consumed by applications that receive data streams through the cloud. The cloud provides a huge amount of processing and storage capabilities for the data generated by IoT devices while abstracting their heterogeneity. By offering sensing and actuation *as a service*, cloud providers broaden their portfolio of services for users and applications.

One of the enabling technologies of the CoT paradigm is virtualization. Virtualization is the logical abstraction of hardware and/or software features and it promotes the decoupling between entities that produce and consume resources. Virtualization is at the core of cloud computing, but it has also been used in other contexts, such as communication networks [2], and more recently for sensors and sensor networks [3].

Among many types of components that make up the IoT infrastructure, and therefore, the CoT, multimedia devices are very important. In a report produced by Cisco [4] it is estimated that by 2022 about 80% of the Internet bandwidth will consist of multimedia streams. The importance of this type of device has given rise to the concept of Internet of Multimedia Things (IoMT) [5]. In the IoMT, sensors are cameras and microphones with limited processing and storage capacity, which connect to heterogeneous devices and diverse applications. Traditional sensors, such as temperature, pressure, and light detection, typically generate discrete data. On the other hand, multimedia sensors produce continuous, massive data streams with nontrivial structure and temporal significance. Such features make the processing of multimedia streams more complex than traditional sensor data. Also, there is high heterogeneity in the communication protocols and data formats supported by multimedia devices, adding an extra layer of complexity in the acquisition, processing, and consumption of data. Particular features of multimedia streams in the context of IoMT make the use of traditional already existing sensor virtualization models not suitable. It becomes necessary to design models tailored for multimedia sensors. We claim that models and mechanisms to abstract the complexity of multimedia sensors will enable the development of innovative services in relevant areas such as health, security, education, and entertainment.

In a typical CoT system, sensor-generated multimedia streams are virtualized in the cloud and delivered on-demand to applications. However, the virtualization of sensors and the availability of virtual sensors as cloud services may not satisfy delay requirements of multimedia applications, which can be quite strict. A promising strategy that has gained momentum recently to decrease delay is to migrate (part of) processing from the cloud to the edge of the network, placing it closer to the sensing devices (data sources). Such strategy is exploited by recent paradigms of Edge [6] and Fog computing [7]. Some benefits achieved by using these paradigms are decreasing delay, better use of available resources and increasing data security/privacy. Multimedia applications benefit from running in low latency environments, so creating an architecture for virtualization of multimedia sensors at the edge is a promising approach.

In this context, the goal of this work is to propose an architecture for virtualization of multimedia sensors at the network edge, called V-PRISM (Virtual Programmable IoT Multimedia Sensor). V-PRISM follows a three-tier architecture encompassing the cloud, the edge and the things tiers. Its components are responsible for processing multimedia streams produced by physical sensors. Stream processing is performed by the Virtual Multimedia Sensors (VMS) that are deployed at the edge tier. V-PRISM is designed to reduce processing and battery consumption of physical sensors, decrease bandwidth consumption in the IoT network, and decrease delay for multimedia applications. In this work, we define the IoT Network as the wireless network that connects smart objects to the edge node. In addition, by using data from a single physical sensor as data source to many different VMS whose processed data will be sharing among multiple applications, V-PRISM contributes to increase the ROI for infrastructure providers.

Besides including a set of software components that concretize the goals of the proposed architecture, V-PRISM also provides templates for creating various types of virtual multimedia sensors (VMS). To the best of our knowledge,

978-1-7281-5503-6/20/$31.00 ©2020 IEEE

V-PRISM is the first proposal of a general virtualization architecture tailored to multimedia sensors and whose components run at the edge of the network.

The rest of the paper is structured as follows. In Sec. II we discuss related work and in Sec. III we describe the design of our architecture. We present the evaluation results of our proof-of-concept in Sec. IV and V. Final remarks and future work are presented in Sec. VI.

II. Related Work

As commented earlier, in the Cloud of Things (CoT) paradigm [1], the cloud acts as an intermediate layer between sensor data and applications. Traditional cloud computing is strongly based on resource virtualization, to make it easier for multiple users to share a single physical instance of a resource or application. Cloud virtualization also manages workload, transforming traditional computing and making it more scalable, cost-effective and efficient. In the CoT, the strategy of virtualization has been adopted to reduce management complexity generated by sensor heterogeneity, since it helps to abstract specific features of physical sensors, thus promoting interoperability among devices. Virtualization also contributes to decrease the sensor infrastructure maintenance costs and increase the service availability. Sensor virtualization has been studied by several researchers [3], [8]–[10]. This section discusses some existing approaches for sensor virtualization.

One of the first approaches for sensor virtualization was proposed in [11], where each virtual sensor acts as a web service whose data is consumed by multiple applications. The goal was creating a three-layer architecture, including a device access layer, semantic layer, and the service layer. Each layer abstracts the heterogeneity of the layer immediately below. Another strategy for defining a virtual sensor is the creation of virtual sensor networks (VSN). In [12], the authors define a VSN as a specialization of a Wireless Sensor Network (WSN). Such specialization consists of two parts, the first is the physical sensor infrastructure (SInP) and the second is the sensor data abstraction and aggregation layer. An example of VSN application was proposed by [13] where a set of heterogeneous cameras was used in the composition of services such as detection, recognition and tracking of objects. Another work adopting this strategy is [3], in which the Osiris framework is proposed. In Osiris, each physical sensor is mapped to a layer called SensorNet and applications consume data in the VirtualSensorNet layer.

Two important features for sensor virtualization are how the data is categorized and how it is accessed. One way to accomplish those tasks is by creating ontologies. An example of such strategy was proposed in [14]. Each sensor publishes its data in a repository and informs the data types it provides, while applications request the data type they need for the central repository. The data description for each sensor is in the OGC[1] standard specified by W3C.

A hybrid approach regarding where the data stream is processed was proposed in [9], in the context of traditional sensors data (not multimedia data). Olympus is an information fusion and cloud-of-sensors-based decentralized WSN virtualization model. It aims at making the best use of the cloud and the physical WSAN environments by finding a balance between two possible approaches for running services: centrally (inside the cloud), and locally (within the physical sensors, exploiting the device capabilities, albeit limited, for data processing).

The proposal presented in [15] brings a virtualization model where processing nodes at the edge tier can also be used as a location for the deployment of virtual sensors. In such approach, virtual sensors are divided into three categories: **SensingVN** that represents simple data types and provides a stream of raw data sensed by one or many physical nodes, **DatahandlingVN** that provides value-added information to meet user requests by employing information fusion techniques and **ActuationVN** that provides actuation capabilities over the physical environment. In addition, a collaborative process for creating virtual nodes on edge nodes adjacent to the requested node has been implemented. This feature allows the distribution of virtual sensor processing across the edge tier.

In this section, we presented some proposals for the virtualization of traditional sensors. In contrast, in our proposal, the goal is to virtualize multimedia sensors at the edge. Virtualized multimedia sensors generate complex multimedia data, so it is necessary to propose a new architecture for the creation and management of those new virtual sensor types.

III. V-PRISM Architecture

In this section, V-PRISM architecture is presented. It follows a three-tier architecture encompassing the cloud, edge and things tiers [16]. The architecture components are shown in Fig. 1, they are responsible for processing multimedia data streams that are produced by multimedia physical sensors (MS) and consumed by applications (APP). APPs are typically deployed in the cloud and physical multimedia sensors are in the things tier (IoMT). Stream processing is performed by Virtual Multimedia Sensors (VMS). The VMS and all V-PRISM components are deployed in the edge tier.

A. VMS and SRC

A Virtual Multimedia Sensor (VMS) is a component responsible for processing a multimedia stream, e.g. converting a WAV stream to MP3 or performing face recognition from a video stream. A VMS sends the processed data directly to applications (that requested the respective stream). For each specific type of multimedia stream processing functionality, there will be one type of VMS.

Each VMS has one or more input ports (connected to an SRC or another VMS) and many output ports. Notice in Fig. 1 that the arrow connecting the VMS and the application is unidirectional. This fact implies that the application does not have a direct communication channel with the VMS. If the application needs to send a control message to a VMS, this must be done using the communication features of an IoT Broker (explained below) and V-PRISM.

Sources (SRC) are the components responsible for forwarding a multimedia stream produced by a multimedia sensor (MS) to a given VMS. Each SRC is connected to a smart object sensor. If the smart object has more than one sensor, e.g. camera and microphone, and the output stream is multiplexed, then there will be only one SRC connected to

[1] http://www.opengeospatial.org/

that smart object. Each SRC is programmed to connect to a specific device type (e.g. USB camera) and to send a multimedia stream type, e.g. H.264 video, to one or more VMS. This approach makes the architecture flexible for incorporating different device types. An SRC is data stream agnostic since it has no knowledge of the content being carried. This feature decreases the complexity of creating new types of SRC. An SRC can send data to multiple VMS.

A physical device may have multiple sensors and can provide data in various formats and compliant with different protocols. Each sensor must be related to a single SRC. This relationship is unary and must be configured using V-PRISM stream descriptors, during the startup process. The type of SRC used must be compatible with the type of data stream and the communication protocol available by the physical sensor. That is, in an implementation, an SRC operates as a driver, interpreting specific data formats and protocols.

There is a strong connection between an SRC and its physical sensor. This is because the physical sensor will provide data in a predefined format, requiring manual configuration (during the setup process) between the SRC and its respective physical sensor. In contrast, the link between SRC and VMS is weak since the SRC can be changed at run time. The exchange of control messages between SRC and VMS is asynchronous and performed using the publish-subscribe pattern.

A VMS is defined as a tuple VMS = {P, O, S, D, C} where, **P** is a set of VMS configuration parameters, such as saturation level, noise level, quality of the video, etc.; **O** is a set composed of SRC and/or VMS instances used as input streams. **S** is the software instance that performs multimedia stream processing; **D** is an {IP, PORT} tuple that defines the address used to send the output multimedia stream (it can identify an application or another VMS). **C** is the category of VMS (detailed below).

B. VMS Category

As VMS is a new concept, here we are proposing a way to categorize them. From a broad perspective, we claim that any kind of VMS can be classified in one of the following categories. Each category defines the role of the VMS in the relationship between the physical device (stream producer) and the application (stream consumer).

1. Thing: a VMS in this category only forwards data and enhance in the VMS some attributes of a physical device. For example, a VMS can have a fault-tolerant feature that the physical device does not. Another useful feature of this VMS Category is to perform multicast stream delivery. An example of *Thing* is illustrated in Fig. 2(a).

2. Process: a VMS in this category processes the input stream and delivers it to the application or VMS in a different format but with the same nature. One example of VMS in this category is a VMS that receives a WAV stream and delivers it in MP3 format to an application, both are multimedia audio streams (same nature) but in different formats (WAV and MP3). An example of *Process* is illustrated in Fig. 2(b).

3. Service: a VMS in this category processes the input stream and changes its nature. One example of VMS in this category is a VMS that processes a video stream and detects inappropriate behavior like physical aggression or irregular parking and deliver to an application metadata about it. An example of *Service* is illustrated in Fig. 2(c).

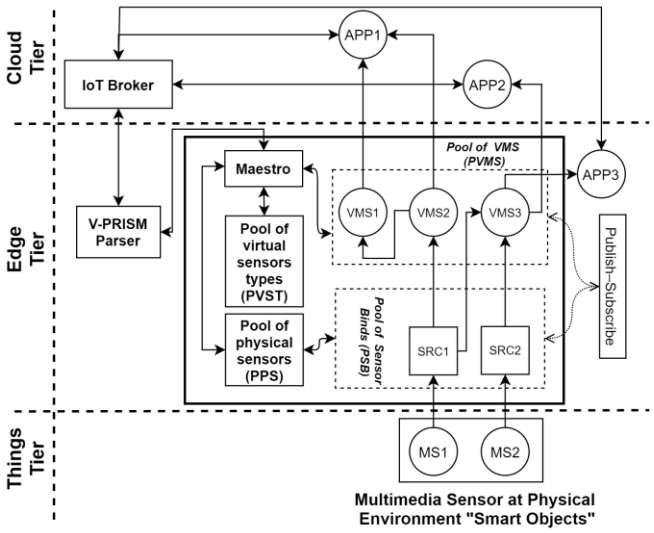

Fig. 1. V-PRISM Architecture

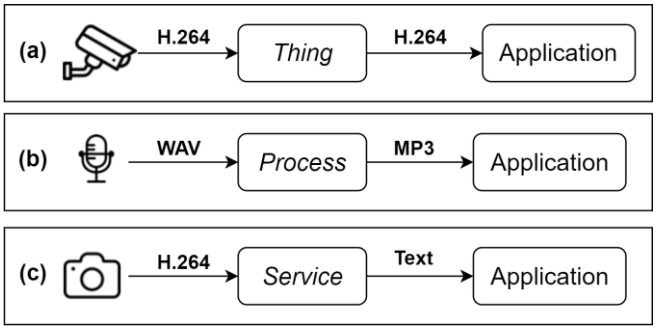

Fig. 2. VMS Categories

C. V-PRISM Components

In this section, we describe in detail the components of V-PRISM, depicted in Fig. 1.

1. Pool of Physical Sensors (PPS): It is a repository that stores physical sensor metadata. Among the stored data, we can highlight the format of the generated stream, the physical location of the sensor, its data-sampling rate, etc. The attributes stored about each sensor can be distinct and depend on the physical nature of the multimedia sensor.

2. Pool of Sensors Bind (POSB): It is a repository for maintaining the relations (bind) between an SRC and a VMS. It is responsible for defining which SRC is providing data for which VMS.

3. Poll of Virtual Sensor Types (PVST): It is a repository that stores VMS types. A VMS type is a software that performs some processing in a multimedia stream. Some possible examples are noise detection, color grayscale video conversion, buffered stream forwarder. During V-PRISM deployment, it is necessary to define which VMS types can be started by applications.

4. Pool of VMS (PVMS): This component improves fault-tolerance. It stores data about all created VMS. It also performs proactive monitoring over the VMSs and if there is

any failure then the VMS can be disabled, and another one can be instantiated and linked to the respective SRC.

5. Maestro: This is the orchestration component of V-PRISM. It provides APIs used by the V-PRISM Parser so that V-PRISM internal data is accessed by applications. It is through Maestro that external demands created by applications reach the internal V-PRISM components.

6. IoT Broker (IB): To provide isolation between applications that require a VMS and V-PRISM, all the communication between them (except multimedia data stream transmission) is accomplished through an IoT Broker. An IoT Broker is a type of Message Broker [17]. Using an IoT Broker allows heterogeneous systems to communicate.

7. V-PRISM Parser (VP): It is an intermediate component between IB and Maestro. The goal of the VP is to promote independence between V-PRISM and the IB. Its loose coupling enables the adoption of V-PRISM in an IoT environment already deployed.

D. Initialization of a VMS

Fig. 3 shows the steps to create a VMS. Initially, the application sends a message to the IB asking about which possible VMS can be instantiated. Once receiving the list of available types of VMS, the application can then request to create a VMS. For example, "creating a camera-type VMS in *Corridor 1* whose output stream is grayscale and format is H.264". The creation process is asynchronous; the application will only know about the progress of the process either by polling the IB or through a call-back function. It is important to note that this type of communication is independent of V-PRISM and it is up to the application to manage its complexity. After the IB receives the request to create the VMS, it will communicate with Maestro using the VP. After Maestro receives the VMS create request, it performs the operation. Then a message will be sent about the status of the process. The edge node where the VMS will be deployed will be chosen by Maestro. The algorithm for this task will be defined in the V-PRISM implementation. One possible approach was proposed in [15].

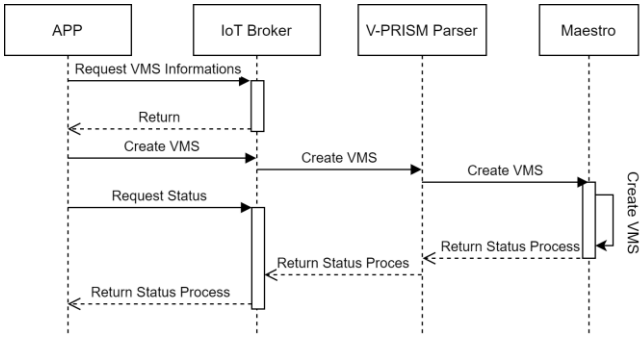

Fig. 3. Sequence diagram for creating a VMS

IV. ALFA - A PROOF-OF-CONCEPT IMPLEMENTATION

We built a prototype implementation of the V-PRISM architecture, called ALFA. The source code and additional information can be found at GitHub [2]. ALFA uses FIWARE [18] as IoT Broker. FIWARE was chosen for the implementation of V-PRISM because it is open source, API oriented, and uses Docker to virtualize its functions. FIWARE provides building blocks, called Generic Enablers (GEs), which are general-purpose components available for use at the cloud and edge layers. GEs are packaged and distributed in container images, thus reducing the effort to integrate new components required to implement the proposed V-PRISM architecture. Communication between FIWARE and Maestro is performed using the ALFA_FIWARE module, which provides APIs that allows communication between ALFA's internal modules and applications.

In ALFA, Docker implements all the functionalities of the virtualization system, namely the orchestration (create, destroy, monitor and configure) of containers that run VMS via API for process automation. The authors in [19] argue that adopting light virtualization with container technology can bring benefits to IoT. This was our main motivation for adopting Docker, a light virtualization tool. Besides saving processing and memory, starting a Docker container is usually faster compared to a traditional virtual machine [20]. In our proposal, it is necessary that the initialization of containers does not generate overload to the host system since it is usually an edge node with low capacity. It is important to note that the Docker is not running in the multimedia device but in the edge node where energy constraints are not so strict. Also, Docker provides automatic mechanisms for load balancing.

In our proof-of-concept implementation, all VMS and SRC are Docker Containers. The flexibility of this approach allows those components to be developed in any programming language. Source Code 1 shows a VMS, which receives at port 5000 an input video stream, does a crop operation and sends the resultant stream to an application that is listening in a DEST and PORT address.

SOURCE CODE 1 VMS EXAMPLE

Line	Comand
1	FROM V-PRISM/docker-gstreamer
2	ENTRYPOINT exec gst-launch-1.0 gst-launch-1.0 \ udpsrc port=5000 caps = "application/x-rtp, media = (string) video, clock-rate=(int)90000, encoding-name=(string)H264, payload=(int)96" \ ! rtph264depay \ ! decodebin \ ! videocrop top=200 left=100 right=4 bottom=0 \ ! x264enc \ ! rtph264pay \ ! udpsink host=DEST port=PORT

V. V-PRISM EVALUATION

This section describes the experiments performed to evaluate V-PRISM. The first experiments evaluate whether a smart object can deliver multimedia streams for various applications that expect different streams from those produced by the physical sensor. The following experiment assesses the consumption in terms of battery and processing in the smart objects, as well as bandwidth consumption on the IoT network when multiple applications require streams from the same object. It is noteworthy that the application proposed for this experiment does not reflect the full range of possibilities of a VMS, but it was chosen because it represents a simple application and thus allows isolated analysis of the evaluated variables. Finally, an experiment was designed to evaluate the delay difference between a VMS deployed in the edge and another in the cloud. The experiments were performed on virtualized machines in VirtualBox, with Ubuntu 18.04, 8GB RAM and 4 Intel (R) Core (TM) i7-8565U processors. Docker version 19.03 was used to virtualize ALFA components.

[2] https://github.com/midiacom/alfa

A. *Test Cases and Architecture Validation*

For the first test case, a VMS type called Noise Detector was created. According to our proposed categorization, this VMS type is *Service*. It receives audio streams and determines its volume. If the volume is higher than a threshold configured during the VMS instantiation, then a noise detection alert is sent to an MQTT server. Fig. 4 shows the VMS parameters within ALFA, the MQTT server that receives the alerts, and the Docker container that runs the VMS. In this VMS, the alert is sent whenever the noise volume has a power greater than 0.02.

In the second test case, two types of virtual sensors were created: Video Crop (cuts a specific video region) and Video Grayscale (color to grayscale video conversion). In our categorization, both VMS types are *Process*. Two Video Crop VMSs were instantiated to demultiplex a camera stream that was filming two doors, as illustrated in Fig. 5 (a), and the video segment of each door was delivered to a distinct application, as shown in Fig. 5 (b) and (c). Fig. 5 (c) shows that the stream delivered to the application was initially processed by Video Crop and then by Video Grayscale, creating a processing thread between two VMS thus validating the proposed architecture that increases ROI.

The third test case was the creation of a virtual sensor type called Video Merge. It was used to send to an application the video stream that is the result of combining streams from many different cameras. In our categorization, this VMS type is *Process*. In Fig. 6, we can see a Video Merge application showing the combination of two different videos. The video on the left was generated by a test camera and the video on the right was from a webcam.

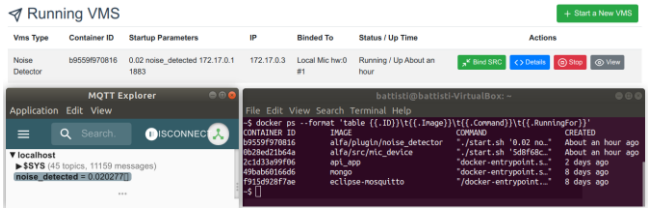

Fig. 4. Noise Detector VMS Run Screens

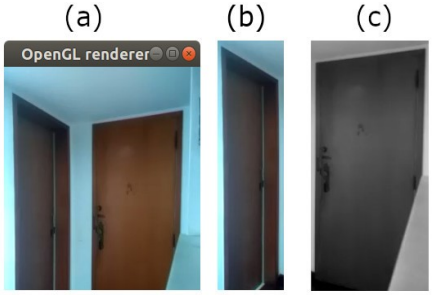

Fig. 5. Video Demultiplexing

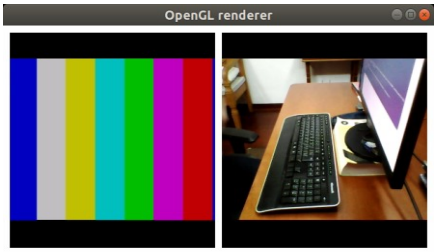

Fig. 6. Video Mosaic

B. *Reducing IoT Device Resource Consumption*

To evaluate the use of V-PRISM on IoT devices, an experiment was performed where a Moto G5 smartphone (representing the IoT device) with Android 8.1 and an RTSP server provides 640x480 H.264 video for several applications. The communication was done using WiFi. Experiments were performed in two scenarios. In the first one, applications access data directly on the smartphone, whereas in the second, applications access data mediated by V-PRISM. Each experiment was run 5 times for 5 minutes with 1, 2, 5, 10 and 20 applications. The results are in Fig. 7, Fig. 8 and Fig. 9. Data were collected using Batterystats. It can be observed that, as expected, the greater the number of applications directly accessing the smartphone to collect the data, the greater the resource consumption. In contrast, when the access is made through an SRC, resource consumption in the IoT device is unchanged. The results show that there are advantages of using V-PRISM, because it keeps CPU, battery and network usage constant in the IoT device. The sharing of the resources provided by the same physical node among multiple applications, leveraged by our architecture, has the potential to increase the ROI for IoT infrastructure providers. Energy consumption and resource sharing are pointed out in [7] as challenges in the IoT environment.

C. *Comparison Between Edge and Cloud in Terms of Delay*

To evaluate the use of cloud and edge for processing multimedia streams, an experiment was developed where two Noise Detector VMSs were created. One was deployed at the edge tier and the other at the cloud. A beep sound was played 80 times in a 5 seconds interval and the multimedia stream resultant was sent to both VMS in parallel.

The audio stream bitrate was 1Mb/s. The one-way-delay was calculated using the ping command. The delay between the SRC and the cloud was 67.5ms whereas the delay between SRC and edge was negligible. The available bandwidth between SRC and cloud was 100Mbps and at the time of testing there was no network congestion. The source was physically in Southeastern Brazil and the cloud was in US East (Northern Virginia). The time interval between successive sound recognitions in the VMS in the cloud and in the edge was calculated. The average noise detection time in the edge was 918ms with a standard deviation of 88ms; in the cloud it was 967ms with a standard deviation of 137ms. Edge detection was in average 49ms lower. Multimedia and real-time applications are latency-sensitive [21]. Thus, proposals that reduce the total delay are increasingly important considering the demand for this kind of application.

The standard deviation of detections in the edge was lower than the cloud, thus pointing to higher predictability of delay in the edge than in the cloud. Another important point to notice is that the average difference between detection time in the edge and in the cloud was less than the delay between SRC and cloud, meaning that cloud processing time was shorter than edge processing time. This happens because the cloud has a higher computing power than edge devices. Therefore, we can conclude that the choice between edge or cloud processing should consider not only the delay between the physical device and the cloud but also the processing time of the multimedia stream.

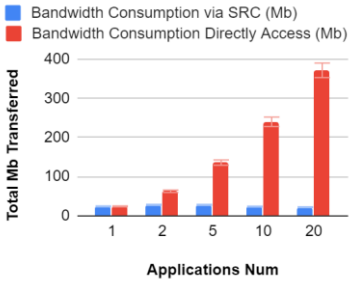

Fig. 7. Data traffic in the IoT network

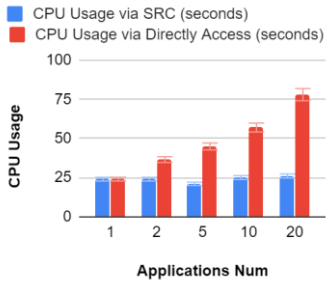

Fig. 8. IoT device CPU consumption

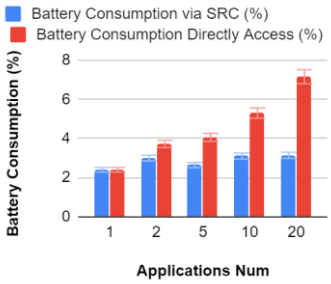

Fig. 9. IoT Device Battery Consumption

VI. Final Remarks

In this paper, we presented the V-PRISM architecture. We illustrated and validated the proposed architecture by the creation of ALFA, a prototype implementation, and the development of four types of VMS, namely Video Merge, Video Crop, Video Grayscale, and Noise Detector. Creating new types of VMS allows new services to be delivered without the need to install new physical devices, thus alleviating the efforts involved in evolving the system and potentially increasing the ROI for IoT infrastructure providers. We also assessed the consumption in terms of battery, processing, and IoT network bandwidth when using V-PRISM. Obtained results pointed out significant benefits with the adopting of the architecture. Without V-PRISM, the greater the number of applications that directly consume the device's multimedia stream, the higher the bandwidth, processing, and battery consumption. When applications consume multimedia streaming through a V-PRISM VMS, regardless of the number of applications, the resources consumed on the device remain unchanged. These results can be even more significant if the application running in the IoT device is CPU intensive. As future work, we intend to develop new CPU-intensive types of VMS such as facial identification and speech recognition, to validate the issue of delay reduction when multimedia stream processing is performed in the edge.

Acknowledgment

This work is partially supported by São Paulo Research Foundation – FAPESP, through grant number 2015/24144-7. Debora Saade and Flávia Delicato are CNPq Fellows.

References

[1] A. Botta, W. De Donato, V. Persico, and A. Pescapé, “Integration of Cloud computing and Internet of Things: A survey,” Futur. Gener. Comput. Syst., vol. 56, pp. 684–700, 2016.

[2] M. Pattaranantakul, R. He, Q. Song, Z. Zhang, and A. Meddahi, “Nfv security survey: From use case driven threat analysis to state-of-the-art countermeasures,” IEEE Commun. Surv. Tutorials, vol. 20, no. 4, pp. 3330–3368, 2018.

[3] F. Santos and R. Guerra, “OSIRIS Framework : construindo sistemas de monitoramento com redes de sensores sem fio para compartilhar dados,” SBRC, 2015.

[4] T. Barnett, J. S. Jain, A. Usha, and T. Khurana, “Cisco Visual Networking Index (VNI) Complete Forecast Update , 2017 – 2022,” 2018.

[5] S. A. Alvi, B. Afzal, G. A. Shah, L. Atzori, and W. Mahmood, “Internet of multimedia things: Vision and challenges,” Ad Hoc Networks, vol. 33, no. May, pp. 87–111, 2015.

[6] W. Yu et al., “A Survey on the Edge Computing for the Internet of Things,” IEEE Access, vol. 6, no. c, pp. 6900–6919, 2017.

[7] M. Mukherjee, L. Shu, and D. Wang, “Survey of fog computing: Fundamental, network applications, and research challenges,” IEEE Comm Surv. Tutorials, vol. 20, no. 3, pp. 1826–1857, 2018.

[8] M. E. Khansari, S. Sharifian, and S. A. Motamedi, “Virtual sensor as a service: a new multicriteria QoS-aware cloud service composition for IoT applications,” J. Supercomput., vol. 74, no. 10, pp. 5485–5512, 2018.

[9] I. L. Santos, L. Pirmez, F. C. Delicato, S. U. Khan, and A. Y. Zomaya, “Olympus: The cloud of sensors,” IEEE Cloud Comput., vol. 2, no. 2, pp. 48–56, 2015.

[10] A. Zaslavsky, C. Perera, and D. Georgakopoulos, “Sensing as a Service and Big Data,” Proc. Int. Conf. Adv. Cloud Comput., no. May 2014, 2013.

[11] S. Alam, M. M. R. Chowdhury, and J. Noll, “SenaaS: An event-driven sensor virtualization approach for internet of things cloud,” IEEE NESEA, pp. 1–6, 2010.

[12] M. M. Islam, M. M. Hassan, G.-W. Lee, and E.-N. Huh, “A Survey on Virtualization of Wireless Sensor Networks,” Sensors, vol. 12, no. 2, pp. 2175–2207, Feb. 2012.

[13] P. Kulkarni, D. Ganesan, P. Shenoy, and Q. Lu, “SensEye: A Multi-tier Camera Sensor Network,” in ACM Multimedia, 2005, p. 229.

[14] J. P. Calbimonte, H. Jeung, O. Corcho, and K. Aberer, “Semantic sensor data search in a large scale federated sensor network,” CEUR Workshop Proc., vol. 839, pp. 23–38, 2011.

[15] M. P. Alves, F. C. Delicato, I. L. Santos, and P. F. Pires, “LW-CoEdge: a lightweight virtualization model and collaboration process for edge computing,” World Wide Web, 2019.

[16] W. Li et al., “System modelling and performance evaluation of a three-tier Cloud of Things,” Futur. Gener. Comput. Syst., vol. 70, pp. 104–125, 2017.

[17] E. Kovacs, M. Bauer, J. Kim, J. Yun, F. Le Gall, and M. Zhao, “Standards-Based Worldwide Semantic Interoperability for IoT,” IEEE Commun. Mag., vol. 54, no. 12, pp. 40–46, 2016.

[18] FIWARE, “About us,” 2019. [Online]. Available: https://www.fiware.org/about-us/. [Accessed: 15-Jul-2019].

[19] R. Morabito, “Virtualization on internet of things edge devices with container technologies: A performance evaluation,” IEEE Access, vol. 5, pp. 8835–8850, 2017.

[20] B. Xavier, T. Ferreto, and L. Jersak, “Time Provisioning Evaluation of KVM, Docker and Unikernels in a Cloud Platform,” Proc. - 2016 16th IEEE/ACM Int. Symp. Clust. Cloud, Grid Comput. CCGrid 2016, pp. 277–280, 2016.

[21] S. Maheshwari, D. Raychaudhuri, I. Seskar, and F. Bronzino, “Scalability and performance evaluation of edge cloud systems for latency constrained applications,” Proc. - 2018 3rd ACM/IEEE Symp. Edge Comput. SEC 2018, pp. 286–299, 2018.