Application-aware IoT Camera Virtualization for Video Analytics Edge Computing

Si Young Jang
School of Computing
KAIST
Daejeon, South Korea 34141
Email: sy.jang@kaist.ac.kr

Yoonhyung Lee
School of Computing
KAIST
Daejeon, South Korea 34141
Email: nyongee16@kaist.ac.kr

Byoungheon Shin
School of Computing
KAIST
Daejeon, South Korea 34141
Email: bhshin@kaist.ac.kr

Dongman Lee
School of Computing
KAIST
Daejeon, South Korea 34141
Email: dlee@kaist.ac.kr

***Abstract*—Video analytics edge computing exploiting IoT cameras has gained high attention. Running such tasks on the network edge is very challenging since video and image processing are both bandwidth-hungry and computationally intensive. Unlike traditional computing systems, IoT cameras are heavily dependent on the environmental factors such as brightness of the view. In this paper, we propose an edge IoT camera virtualization architecture. For this, we leverage an ontology-based application description model and virtualize the IoT camera with container technology that decouples the physical camera and support multiple applications on board. We also develop an IoT camera reconfiguration scheme that allows IoT cameras to dynamically adjust their configuration to environmental context changes without degrading application QoS. Experimental results based on our prototype implementation show that the responsiveness of our system is 2.8x faster than existing approaches in reconfiguring to the environmental context changes.**

I. Introduction

Edge computing [1], also referred to as cloudlet [2], fog computing [3] and MEC [4], is an emerging technology that enables users to exploit mobile devices or Internet-of-Things (IoT) in close proximity. Recently, video analytics edge computing exploiting IoT cameras has gained high attention. However, since video and image processing are both bandwidth-hungry and computationally intense, running such tasks on network edge is still very challenging. If the cameras constantly send all of their frames in a raw form to the cloud for further processing, the number of frames to be sent can easily exceed the network capacity or increase the queue size on the processing pipeline. Such circumstances can eventually lead to the inability to meet the requirements of target video analytics applications.

Recent studies [5], [6], [7] show that less bandwidth utilization, higher accuracy and energy efficiency can be achieved by coordinating multiple smart cameras and transferring only necessary information to a cloud or an application. In these systems, an edge controller acts as an intermediary between applications and edge IoT cameras. The controller performs frame selection/filtering from multiple cameras, directs a target camera to use an appropriate detection algorithm, or selects a specific camera according to the application requirements. That is, the controller manages a set of cameras and provides the applications it serves with a higher-level or *virtualized* view of underlying cameras and their capabilities, in other words, filtered video stream(s) according to an application's needs.

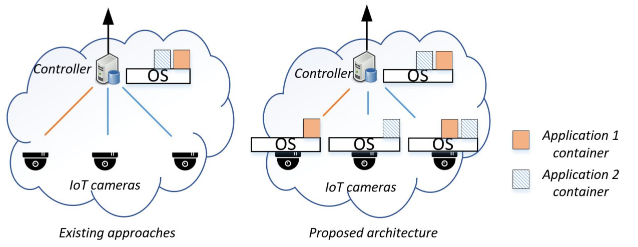


Fig. 1: Camera Edge IoT Cloud.

Unlike a traditional computing resource, an edge IoT camera is heavily dependent on context changes in its surrounding environment [8]. Furthermore, a subtle change in an environmental factor such as brightness leads to immediate service quality degradation. However, the existing solutions mentioned above fail to promptly recognize the environmental context changes, which are often critical to video analytics applications. This is because the controller can only validate the usefulness of the video/frame after they are received from the cameras. The camera records poor quality (or less-useful) frames until the controller makes adjustments to the IoT Camera. This leads to a new approach that enables an IoT camera to instantly recognize the context changes and accordingly adjust its configuration (e.g. streaming parameters) for applications that its controller serve while application specific frame manipulation is continued to be done by the controller.

In this paper, we propose an edge IoT camera virtualization architecture which enables an IoT camera to be virtually configured to support multiple applications and dynamically adapt its configuration to environmental context changes for consistent QoS support for applications that it feeds video streams. For this, the proposed architecture features three key components: 1) an interpreter for understanding the application requirements: We leverage an ontology-based application description model to express the relationship between the type of application and the parameters affecting its QoS. The model is then referenced by the IoT camera in the system. 2) IoT camera virtualization to serve more than one

978-1-5386-9445-9/18/$31.00 ©2018 IEEE
DOI 10.1109/SEC.2018.00017

video analytics application: we virtualize the IoT camera with container technology and create a virtual camera to decouple the physical camera and support multiple applications on board. 3) context-aware IoT camera reconfiguration to adapt to dynamic environmental context changes: we devise a heuristic IoT camera reconfiguration scheme based on the application requirements to be able to quickly adapt to the environmental changes that may affect the performance.

To validate our system, we implement a prototype system with multiple video-recording IoT devices which can run less powerful yet, lightweight object detection algorithms. The experimental results show that virtualization of multiple IoT cameras to serve applications and reconfigure the cameras in our proposed approach can significantly enhance performance in terms of object detection accuracy, computational cost, and response time. Our adaptive reconfiguration scheme can adapt to context changes up to 2.8 times faster than existing controller-based approaches.

In summary, our contributions are as follows:

- We design an architecture which enables an IoT camera to be virtualized for video analytics in an edge computing environment. For this, we define an ontology-based interpretation API which is referenced by all IoT cameras. We then leverage the containerization technology to create virtualized camera containers with a dynamic reconfiguration scheme which adjusts a video stream for different applications depending on their requirements.
- We prototype the proposed architecture on our IoT camera edge cloud testbed, evaluate the proposed QoS-sensitive and context-aware dynamic reconfiguration scheme, and show that it effectively adapts to context changes and improves the performance of video analytic cloud services.

The rest of the paper is organized as follows: In Section 2, we explain a motivating video analytics scenario in edge IoT camera cloud and emphasize the need of application-aware virtualization for IoT cameras. A detailed explanation of our architecture is described in Section 3. In Section 4, we present our experiment setup, the experimental results of our testbed as well as an analysis of these. Related work to this research is explained in Section 5, and we conclude in Section 6 with future work.

II. IoT Camera as an Edge Cloud Node

Video cameras have become more powerful [9] as they are equipped with advanced processors and single-board computers like Raspberry Pi [10]. IoT cameras will therefore not only be providers of image and video data but also providers of computing resources. In this section, we present a motivating example that shows how IoT cameras with application-aware virtualization can improve an application's performance and adaptability in the presence of dynamic environmental context changes and explain what should be done to support it.

A. Motivating Scenario

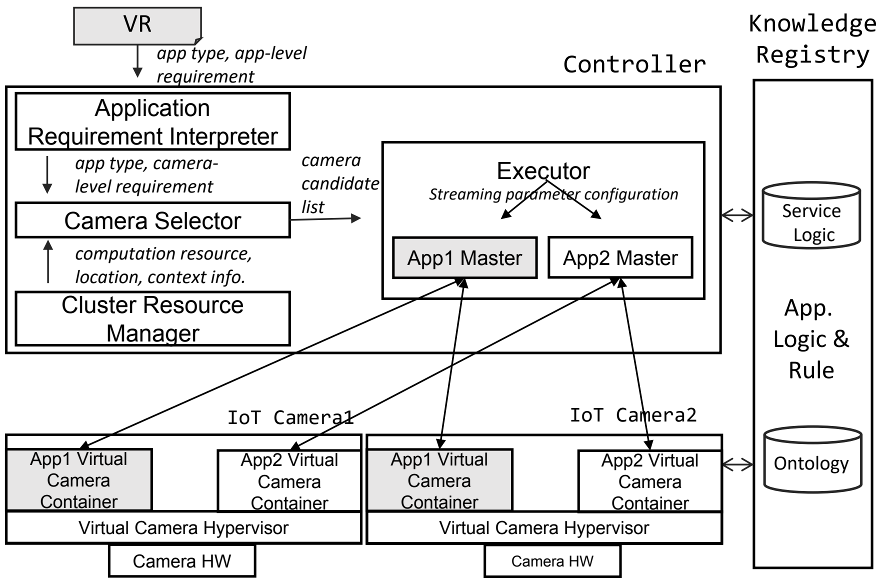
In a public place such as an airport, there are various video analytics applications continuously and simultaneously running to ensure public safety. For example, a surveillance camera always run a monitoring application which detects suspicious actions such as someone dropping a suitcase and then disappearing. When such an action is detected or the police asks for cooperation to find the suspect, the camera quickly runs an identification application to identify the people who are present in the scene while still running the monitoring application at the same time. Unlike monitoring, the identification application requires a higher frame quality to recognize a person's face. Thus, after starting the application, a camera increases its resolution to better recognize faces. When the camera identifies a person with criminal records, it rapidly starts a tracking application to follow the person who is likely to commit a public offense. While the tracking application is running, it should not be allowed to lose a target person so the camera increases its frame rate to avoid missing events. In addition, when an environmental context changes (e.g. the light is suddenly turned off), the camera switches to a preprocessing or detection algorithm that is less affected by luminosity so that it can keep tracking the person without missing due to the frame quality degradation.

B. Application-aware and dynamically reconfigurable IoT camera virtualization

A typical camera driver in the Linux environment only allows a single process at a time to access the camera. However, from the scenario above, several applications need to share a single camera. Such a scenario is also mentioned in [11], where a camera's stream is used for both analyzing the traffic at a crossroad and to answer to another service. Scenarios like this stress out the importance of having a way to generally represent an application's requirements so that the provided resources can fit its needs.

As shown in the example scenario above (Section II.A), a single camera can run different applications: one for monitoring, another for face identification and one last one for tracking. However, there exist common parameters such as required frame rate that can be expressed in the same way because both are using a common video capability. This parameter can also affect the QoS of the corresponding application service, which also varies from application to application. For instance, if the IoT camera is not aware of these application requirements, the computational power on the device will be equally distributed among all applications. In this situation, the tracking application may suffer in performance from slower frame rate than required while the monitoring application does not. This requires a single physical IoT camera to host multiple virtual cameras where each virtual camera configures its setting according to the corresponding application.

As IoT cameras are placed close to an edge environment, dynamic contexts affect directly the performance of the video analytics applications. For instance, a tracking application is very sensitive to the luminosity degradation of a camera due to external factors such as shade or weather change. The application should perform additional adjustment (e.g. brighten, sharpen image) to overcome video quality degrada-



VR
app type, app-level requirement
Controller
Knowledge Registry
Application Requirement Interpreter
app type, camera-level requirement
Camera Selector
computation resource, location, context info.
Cluster Resource Manager
camera candidate list
Executor
Streaming parameter configuration
App1 Master
App2 Master
Service Logic
App. Logic & Rule
Ontology
IoT Camera1
IoT Camera2
App1 Virtual Camera Container
App2 Virtual Camera Container
Virtual Camera Hypervisor
Camera HW

Fig. 2: The proposed architecture.

tion. Additional bandwidth is required to send more frames to the application for these adjustments. However, if a camera is capable to directly deal with such local environment context changes and adjust its configuration to support video quality consistently, the application can reduce the networking and processing cost due to handling extra video frames and perform its task more effectively.

III. Design and Implementation

In this section, we describe our architecture by carefully employing previously discussed design considerations. We give an overview of the architecture and explain its key components and operations in detail.

A. Overview

Figure 2 shows the overall architecture of the proposed scheme. It consists of three main components: Controller, IoT Camera and Knowledge Registry. As mentioned before, an IoT Camera is a node which has not only specific functionalities (i.e. video recording) but also computing power. The **Controller** accepts video requests from applications via the cloud (or the user) and interprets them by referencing the Knowledge Registry for application-aware camera configuration (section III.B). To select target IoT Camera(s), the Controller probes service capabilities of the IoT Cameras. The service capability is calculated by an IoT Camera and the Controller uses it to allocate suitable IoT Cameras for a request in a global view. The Controller sends a start command to serve video application containers to IoT Cameras (section III.C). Each **IoT Camera** is virtualized by splitting the physical camera into multiple virtual cameras (i.e. application containers), and initializes as many containers as necessary using Docker [12]. The video frames are monitored by hypervisor for context changes which degrades the QoS of the application and delivers configuration setting to the application containers along with the video frames. It also prioritizes and adjusts video frame configurations when the physical camera is not able to support all the requirements of currently running application containers (section III.D). The hypervisor feeds each virtual camera container to adjust streaming parameters upon detecting changes in the context affecting QoS (e.g. luminosity, blurriness, etc) (section III.E). A detailed description of these processes is elaborated in the following subsection.

However, the following aspects are beyond the scope of our paper:

- Increasing the accuracy, processing speed or energy efficiency of the computer vision algorithm.
- Task offloading mechanism for efficient video analytics processing.

B. Application-aware Configuration

In order for an IoT Camera to process and filter video frames according to an application's needs, it has to understand the application's requirements. For this, we devise an ontology based relationship between an application and its QoS-affecting parameters. To represent the capability of an IoT device in an edge IoT cloud in terms of service functionality and correlation with other IoT devices, ontology can be considered one of most plausible solutions [13], [14], [15]. It can express the representation of IoT resources in a semantically structured way. Figure 3 depicts an example of a service ontology. This is then loaded to the Knowledge

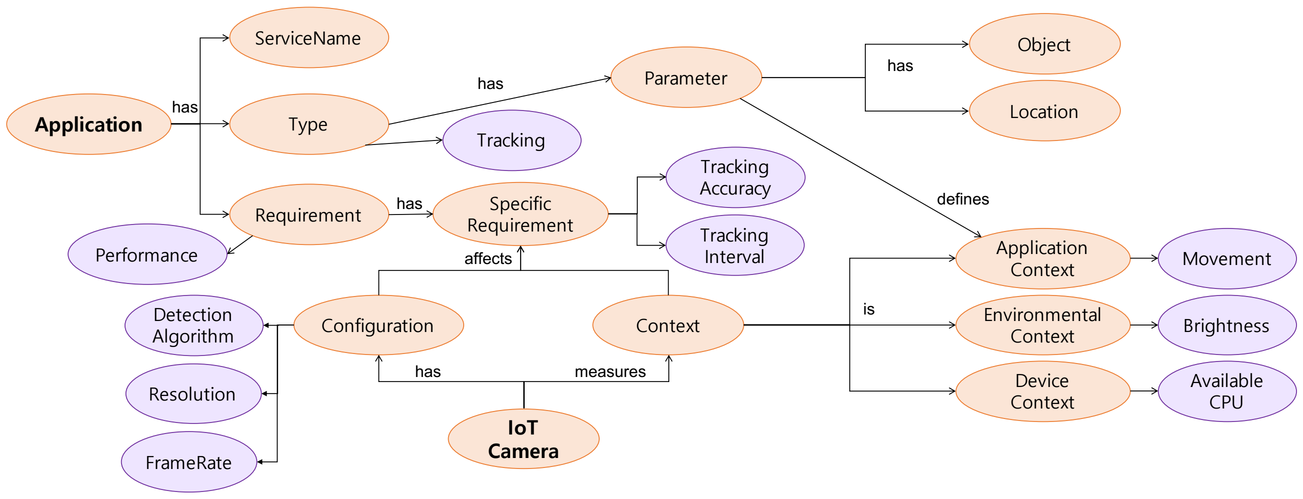


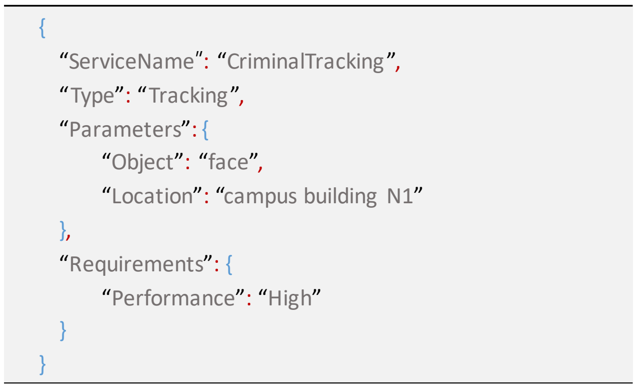
Fig. 3: A service ontology model in the Knowledge Repository.

Registry for both Controller and IoT camera to reference. We describe the operation and relationship between the application and its parameters below.

Knowledge Registry. The Knowledge Registry acts as a shared knowledge base between applications and IoT cameras. It consists of service ontology and service logic database. The service ontology defines conceptual relationships between application requirements, camera configuration, and contexts. Ontology is used for translating concepts coming from different layers, and such a translation process is called quality-of-service (QoS) mapping. We regard the Knowledge Registry as a separated entity from the Controller and IoT camera. Likewise, knowledge instances including ontology and service logics can be updated by edge cloud administrators, IoT manufacturers or cloud service developers without recompiling the Controller and IoT camera software. The Knowledge Registry spans across the Controller and the IoT Camera and provides intelligence to understand the application type and context parameters to support application-awareness. Nonetheless, the physical placement of the Knowledge Registry can be together with the Controller depending on economic or networking conditions.

For each application type, its logic representing the relationship between the QoS, contexts, and the configuration is built in the Knowledge Repository. Application logic is an instance of the ontology model, and the example of tracking applications is shown in Figure 3.

Video Request. Video application requirements (VR) is delivered via a message to the Controller. A VR describes the abstract features and requirements of an application which utilizes IoT cameras. All description formats stem from the keywords of the service ontology in the Knowledge Repository. Since existing edge computing approaches specify QoS requirements related to computation resources [16], the VR description extends the QoS requirements by decoupling application requirements from video-specific configurations.



```
{
  "ServiceName": "CriminalTracking",
  "Type": "Tracking",
  "Parameters":{
      "Object": "face",
      "Location": "campus building  N1"
  },
  "Requirements":{
      "Performance": "High"
  }
}
```

Fig. 4: An example of a Video Request for tracking applications.

A VR consists of the VR name, a service type, filtering parameters, and application requirements. The service type helps the Controller to initialize corresponding IoT cameras with default values from the Knowledge Registry. A VR requirement is described as a key-value pair, and the key is one of the ontology classes in the QoS. The value must be one of the possible values for each QoS, which are predefined in the service ontology as enumerations. A VR can contain multiple VR requirements so that an application can specify conditions for selecting the best IoT camera(s) to allocate its corresponding application container.

We provide an application programming interface (API) to accept VRs from arbitrary applications. We define a VR message format using JSON since it is a lightweight, human-readable, platform-independent and highly compatible format that is good for representing attribute-value pairs of data and it is also utilized in [17], [18]. Figure 4 describes an example of VR in JSON format.

```
[
    (camvirt:Performance camvirt:hasWeight “High”)
    ->
    (camvirt:TrackingAccuracy camvirt:isRequired 0.8)
    (camvirt:TrackingInterval camvirt:isRequired 2.0)
]
[
(?x camvirt:measures camvirt:Brightness)
(?x camvirt:has camvirt:Resolution)
(?x camvirt:has camvirt:DetectionAlgorithm)
(camvirt:Brightness camvirt:hasValue ?a)
(camvirt:Resolution camvirt: hasValue “800x600”)
(camvirt:DetectionAlgorithm camvirt: hasValue “HOG”)
ge(?a, 60)
le(?a, 150)
->
(?x camvirt:has  camvirt:TrackingAccuracy)
(camvirt:TrackingAccuracy camvirt:hasValue 0.85)
]
```

Fig. 5: Example rules for a tracking application.

C. Controller: Dynamic IoT camera selection

When a VR is received, the Controller starts probing IoT Cameras and select a list of appropriate IoT cameras to run the corresponding application. The Controller also reallocates cameras according to the changes of application context such as walking trajectory of a target person. The controller is comprised of Cluster Resource Manager to manage IoT cameras and its context values, Camera Selector and Executor to pass execution messages to IoT Cameras.

Application Requirement Interpreter. The application requirement interpreter converts high-level requirements into camera resource level ones. First, the application requirement interpreter reads a VR and extracts the application type and requirements. Using the application type, it loads the corresponding application logic from the Knowledge Repository. The application logic is constructed as an ontology model and a set of heuristic rules. The rules are based on the ontology model and formatted like Figure 5. Note that this set of rules may be optimized accordingly to the IoT camera operating environment and video application. The interpreter converts high-level requirements into application-specific requirements such as tracking accuracy and tracking interval by triggering a rule engine to executes the rules. Then, by using the relation between specific requirements and configuration which is described in the application logic, the interpreter translates the specific requirements into camera-level requirements in terms of configuration such as frame rate: ($>$1.5), resolution: (480x360, 800x600).

Cluster Resource Manager Our system leverages cluster orchestration tools such as docker swarm or Google Kubernetes for resource management. The Cluster Resource Manager on the controller maintains an overlay network with IoT cameras and probing their computational resource availability. In an IoT environment, the placement of a container is critical in sustaining application QoS as the controller needs to monitor context changes at an IoT camera to make camera selection upon video request. For this, we assume that the controller maintains the topological location information of its belonged IoT cameras.

Along with computational resource availability, the cluster resource manager requests for available IoT cameras resource upon video application request. It keeps track of non-computational factors of the other IoT cameras. That is, adding situation awareness to the IoT by probing context changes observed at a target camera. The information required for the application context is predefined in the application logic, and their data type and format are different depending on the application type and the situation. For example, in our tracking application, the mobility of an object is considered when determining the performance of an IoT camera. It requires information on whether an object has been detected and the coordinate information of the area in which the object is detected. On the other hand, in the customer counting application, the number of objects currently detected is considered.

Camera Selector. The Camera Selector selects suitable IoT cameras based on the interpreted camera-level requirement from the Requirement Interpreter, IoT Camera’s both computation and non-computation information from Cluster Resource Manager, and the selection policy from the application logic. Depending on the application type, the camera selection policy varies because the degree of quality variation is different depending on the application context. Typical camera policies include coverage and focusing. In the coverage policy, if there is no specific target object, it is intended to contain all the required ranges. Counting and monitoring are examples of the application types to which the coverage policy is applied. Conversely, in the case of the focusing policy, there is a specific target object and it is intended to select the camera around the object. A typical application type is tracking. In addition, various policies can be added/selected in the Knowledge Registry and can be changed at any time.

We illustrate how camera selection can be done by taking a tracking application where focusing policy is applied. First, the Controller checks the camera configuration and selects the camera with the highest object detection rate among the cameras near a target object. If the object detected by the corresponding camera moves beyond a certain point, another IoT camera in the object’s movement direction is selected based on the camera topology information.

Executor. Based on a selected allocation policy from the Camera Selector, the Executor modifies the parameters of Dockerfile and requests each camera to instantiate a virtual camera container by using the Dockerfile. An example of Dockerfile is shown in Figure 6. Then, the Executor creates an Application Master container that collates results from all virtual camera containers allocated for a target application as shown in Figure 2.

```
FROM resin/rpi-raspbian:stretch
# Install build tools and libraries for OpenCV.
RUN apt-get update && apt-get install -y
COPY install_dependency.sh /
RUN sudo chmod 755 install_dependency.sh
RUN ./install_dependency.sh

# Load and install pip3 requirements.
COPY requirements.txt /
RUN pip3 install -r requirements.txt
COPY imutils /usr/local/lib/python3.5/dist-packages/imutils/

# Load application program.
COPY detection_app1.py /

# Expose port for handling frames.
EXPOSE 7759

# Load object model for the algorithm.
RUN mkdir cascades
COPY haarcascade_frontalface_default.xml /cascades/

# Run application.
CMD ["python3", "detection_app1.py"]
```

Fig. 6: An example of a Dockerfile which runs on IoT Camera.

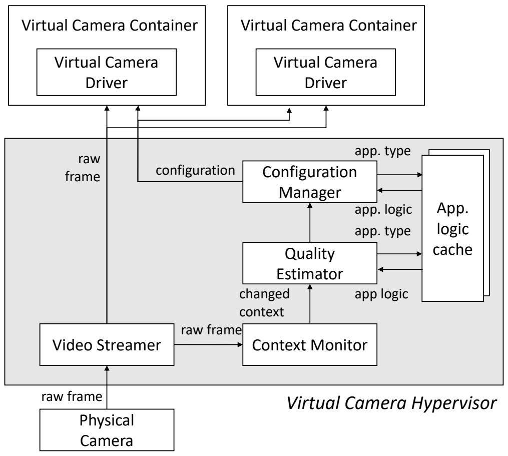


Fig. 7: Virtual Camera Hypervisor.

D. Virtual Camera: Camera Virtualization

In order for an IoT camera to simultaneously run multiple video surveillance applications, we leverage container-based virtualization (OS level virtualization) instead of virtual machine (hardware virtualization). We apply Docker's [12] open source, container-based virtualization as it is a feasible solution in resource constrained IoT cameras for quick allocation [19].

1) Virtual Camera Container: Virtual Camera Container includes a process performing video analytics and a virtual camera driver for a target application. Virtual Camera Container is application-specific and created when the a VR message is interpreted at the controller. For instance, human detection operation from Section II.A is processed here. This container polls video frames from the Virtual Camera Hypervisor, where the configuration is selected for the application needs. The Virtual Camera Container manipulates (e.g. lower resolution) the video frames and streaming parameters such as sampling rate with the given configuration by the Configuration Manager.

Virtual Camera Driver. A Virtual Camera Driver which lies in the Virtual Camera Container has a built-in function that imports frames from application-specific cameras that are not physically present. From the application's perspective, the camera seems to exist locally but in reality the virtual camera receives the modified frames generated remotely by the physical camera and delivers them to the application. There are as many virtual drivers created as the number of applications running on the IoT camera.

2) Virtual Camera Hypervisor: To virtualize an IoT camera for supporting multiple applications, we devise Virtual Camera Hypervisor. It dynamically configures a Virtual Camera Container according to the application QoS requirement and context changes. There are four main modules: *Video Streamer*, *Context Monitor*, *Quality Estimator*, and *Configuration Manager*. Figure 7 shows the Hypervisor in detail.

Video Streamer. It is directly connected to a physical camera and a virtual camera container to mediate the video stream between physical camera and virtual camera container. It gets video frames from the physical camera and sends them to the virtual camera container. In the Video Streamer, the camera configuration is set with the highest quality to satisfy any application requirement.

Context Monitor. It periodically updates the context from video frames which are obtained by the Video Streamer. It only monitors environmental contexts which can affect application performance. It also checks if the environmental context has an impact on the application QoS by performing simple image processing operations which do not consume an excessive amount of computation (e.g. calculating the difference in pixel values, calculating average pixel values). If more than one applications need the same context, the context monitoring is only performed once at a time to avoid duplication. This process continues periodically. The Context Monitor updates contexts which affect the current running applications. Monitored contexts are stored and used to calculate estimated quality. Whenever there is a change in the environmental context, the Quality Estimator gets notified.

Quality Estimator. It estimates QoS change of the applications which are running on the IoT Camera based on the information of the environmental context changes from the Context Monitor. The current configuration of each virtual camera is stored and maintained, and application logic is cached just after the application request is delivered to the hypervisor if the application logic cache is not stored in the hypervisor. In other words, we take the application logic cache to find the relationship model between context, camera resource and QoS, and estimate the degree of QoS change according to the context change in the current virtual camera configuration based on this. If the estimated QoS is out of the required QoS range or the difference from the previous estimated QoS surpasses a threshold, it notifies to the Configuration Manager to reconfigure the Virtual Camera Container.

Configuration Manager. The Configuration Manager adjusts the configuration of the Virtual Camera Container according to the QoS requirement of a target application. Based on heuristic relationship between environmental context change and camera configuration (e.g. processing algorithm), a new configuration is made to guarantee that it leads to a resulting QoS residing within the requirements of a target application and consuming the smallest amount of computational resources. The configuration is sent to the Virtual Camera Container. In addition, since two or more application video streams must be served, to avoid conflict we prioritize the application container by type, request time and request entity. When applications with different priorities are executed at the same time, the configurations of all running applications should be balanced. For this, we define *Global Quality (GQ)* for each application $i \in I$ where I is a set of currently running applications. The GQ is expressed as follows:

$$GQ = \sum_{i \in I} w_i Q_i \quad , \quad \sum_{i \in I} w_i = 1$$

where w_i and Q_i are the weight and the quality of the application i, respectively. Each weight is set by the relative priority among applications. There is a threshold which is the lower limit of GQ. The threshold is set to a sum of values which are multiplied by each minimum required quality and priority. When GQ drops under threshold, the Configuration Manager restore the value by reconfiguring the virtual camera container. To effectively maintain the GQ, the Configuration Manager improves the configuration of the application having higher weight and degrades the QoS of the application which has lower weight. It selects a near-optimized configuration set by iterating all possible sets.

E. Context-aware dynamic camera reconfiguration

The overall flow of context-aware reconfiguration is shown in Figure 8. In order to ensure that the quality of the frames received by the application is not lower than the requirements due to the context changes which greatly affect the application QoS such as the brightness and motion in the surrounding environment, Context Manager monitors the context changes. If the context has changed, the Quality Estimator evaluates to what extent the QoS will be affected. If the context change is estimated to affect the application QoS, the configuration is updated at the Configuration Manager. The configuration is finally sent to the Virtual Camera Container through overlay networks and the process in the container starts updating parameters and sends applied frames to the client. When the changed configuration causes other applications' qualities degrade and thus the global quality becomes lower than the threshold, the Context Manager balance qualities to adjust configurations as the previous section mentioned.

IV. Evaluation

In order to validate the design of our architecture, we implement a proof-of-concept system with multiple IoT cameras running surveillance applications. We describe our experiment setup, discuss evaluation metric and analyze the response time

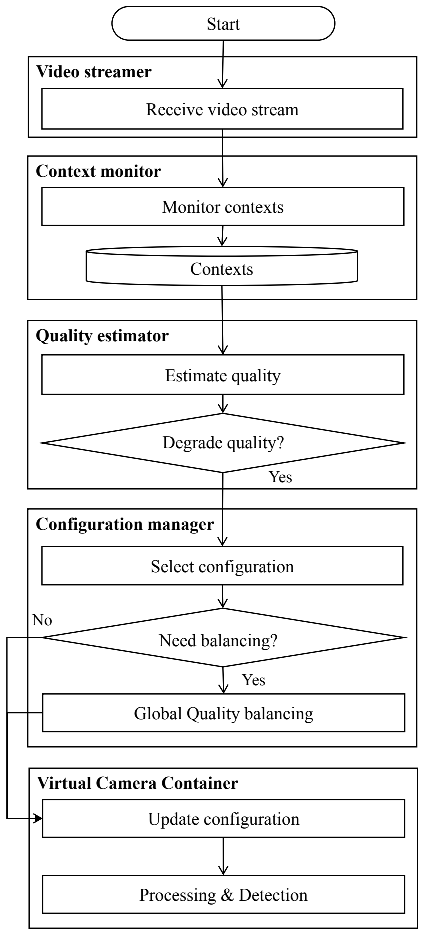


Fig. 8: An overall flow of context-aware IoT reconfiguration.

and accuracy of our application-aware virtualization for IoT cameras.

A. Experimental Setup

Controller and IoT Camera. To implement both the Controller and IoT Cameras, we use the Raspberry Pi 3 [10], since current smart home gateways or home server in general have similar computing power and capacity [20]. However, the Controller can also be flexibly implemented on desktop computer with more computing power. For modeling an IoT camera, we add a Raspberry Pi Camera Module on the Raspberry pi device to record video on the camera. We place the Controller without connecting any other peripherals. All nodes in our prototype testbed are wireless, thus transmit packets through built in WiFi (IEEE 802.11n) in 2.4GHz channel.

Our virtualization system is implemented in Python3.6, and utilizes the lightweight MQTT communication protocol between the Controller and IoT Cameras. To ensure lightweight virtualization and quick allocation of an isolated user space instance, we use Docker's[12] container solution in the IoT

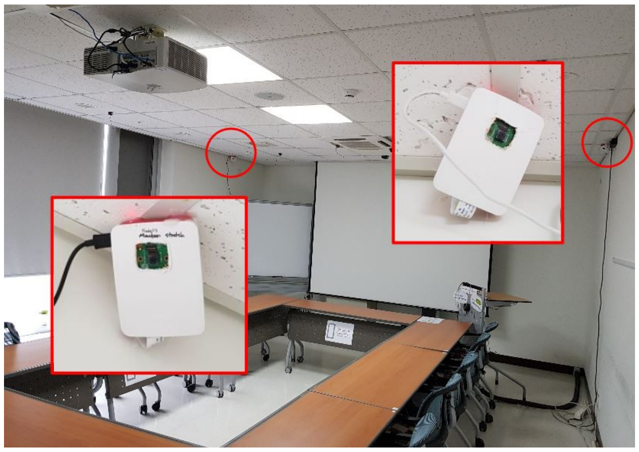


Fig. 9: Testbed for a video analytics application.

cameras. We use Docker version v17.12.1-ce to implement our system. The hypervisor is implemented as a standalone single container.

Video analytics applications. We implement two prototype video analytics applications (monitor and tracking) in Python3.6. We utilize built-in detection algorithms from OpenCV3.3 [21]. We use HOG [22] and Haar cascade [23] algorithms which are reasonably lightweight and thus suitable for Raspberry Pi.

Experiment environment. We conduct our experiments at a 4.6 m × 7.0 m seminar room and hallway in our university campus. We install four IoT Cameras and one Controller implemented as mentioned above. Our testbed is shown as Figure 9.

Experiment scenario and dataset. To evaluate our system, we use two different types of applications: tracking and monitoring. Tracking application tracks a person by continuously detecting a human figure and sends the cropped frame, which contains the detected human, to the Controller. Monitoring application monitors the situation by detecting the movement in comparison to the previous frame and sends the changed area in the frame to the Controller. In our video set, few people walks in random order in the seminar room. To make the experiment data consistent, we use prerecorded videos to guarantee reproducibility and perform an accurate experiment. The video is played from the raspberry pi IoT node to imitate real-time streaming for our experiment.

B. Application-aware IoT Camera Virtualization

We compare our system with one without IoT camera virtualization. The quality metric used in our experiment is the ratio of *useful* frames for the application. For instance, if the system delivers five video frames to the application and the application detects a person in every frame, then the application quality becomes 1.0. We measure the detection interval to be one second.

	without Virtualization	with Virtualization
Launch time (s)	1.02313	2.599536
Processing time (s)	0.238586	0.240057
CPU usage (%)	64.04865	64.13919

TABLE I: Performance of a video analytics application without virtualization and with virtualization (Docker).

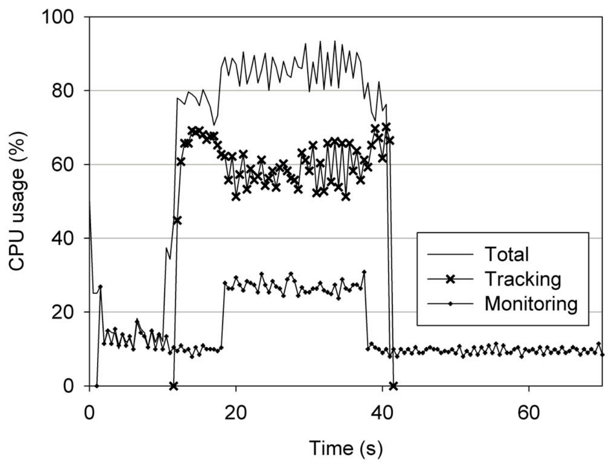


Fig. 10: Running tracking and monitoring applications on IoT camera.

Feasibility of IoT camera virtualization. To test the feasibility of virtualizing an IoT camera using Docker, we run two test cases: 1) running a video analytics application on a bare OS without virtualization framework and 2) running the same application on a Docker container.

We measure the launch time overhead, which is the time from when an IoT camera is requested to instantiate the docker image to when the image is ready to execute a target application. We also measure the processing time of one video frame. Additionally, we measure the CPU usage of an IoT camera during processing and how much additional computing power is consumed in the proposed system. The results shown in Table I are obtained by averaging ten experiment runs.

The execution time without virtualization takes approximately 1.0231 sec while that with virtualization is 1.57 sec more at launch time. However, the differences in frame processing time and CPU usage are reasonably low, that is, 1.4 msec and 0.09% higher respectively. Except for the launch time, there is small difference. The launch time can be minimized if the container is pre-initialized which will be discussed in Section IV.E.

Multiple applications through IoT camera virtualization. When more than one application are running, an IoT camera detects the change of its resource caused by the applications and reallocates the camera resource for each application so that the resources are distributed accordingly between multiple applications. For this experiment, we run a monitoring application with one virtual container and then run a human tracking application with another virtual container. Figure 10 depicts the CPU usage of running both of the application throughout the experiment.

The monitoring application is always running to detect sudden changes on the screen, and the reallocation request of a tracking application is received at 13 sec. When a person is detected on the screen of this IoT camera, the monitoring application detects pixels of the video frame change and increases the frame rate to be able to capture the person more accurately, which increases the CPU usage. Due to this event, the CPU usage of monitoring application increased by 21%. For tracking application, the CPU usage is adjusted from 69% to 51% as the monitoring application does not degrade the quality of tracking application. The detection ratio of the tracking application stayed at 1 throughout the running time.

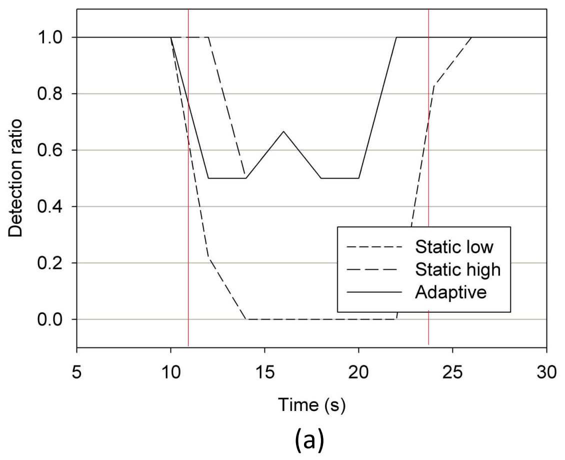
C. Context-aware IoT camera reconfiguration

For evaluation, we conduct two experiments to see how meaningful it is to change the configuration of an IoT camera depending on the environment, and how adaptive the camera itself is at recognizing the context and performing reconfiguration.

Impact of dynamic reconfiguration In order to evaluate the performance of adaptive reconfiguration, we use three schemes: a policy applying a fixed high-quality configuration, a policy applying a fixed low-quality configuration, and a context-aware adaptive reconfiguration policy during the experiment. In the experiment, the tracking application is performed. High quality configuration is set to detect at 600x450 resolution with the HOG algorithm which has high recognition rate and high computing consumption. Low quality configuration is set to detect at 400x300 resolution with a relatively simple detection algorithm, Haar cascade algorithm. The results are shown in Figure 11

The x-axis shows the time and the y-axis indicates CPU usage for the top graph (Figure 11a) and application quality for the bottom graph (Figure 11b). The first red vertical line on the graph indicates a change in context (light gets dimmed) and the second vertical line indicates another change in context (light gets back to normal).

In our dynamic reconfiguration scheme, before the lighting gets dark, the IoT camera has a low quality configuration (400X300, haar cascade) to detect. However, when its environment gets darker, it switches to high quality configuration (600X450, HOG) to avoid quality drop. But, for static low quality configuration, it is not able to recognize the change, which leads to reduced quality. Having high quality configuration consistently consumes approximately 1.5 times more CPU compared to low-quality configuration scheme. On the other hand, our adaptive reconfiguration scheme incurs 12% more CPU overhead during the light out, which lasts for approximately 11 seconds in our experiment.



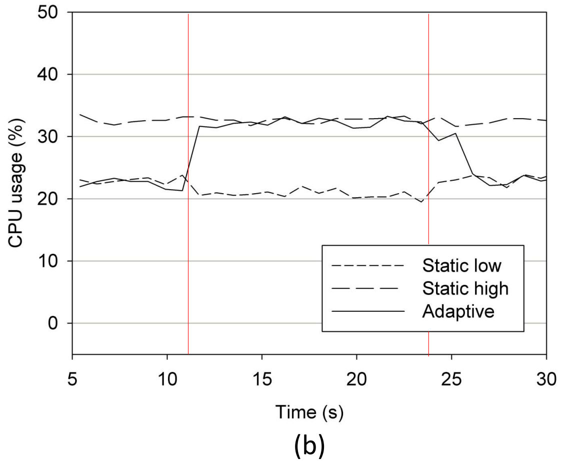


Fig. 11: Comparison of computation usage and quality between static and dynamic IoT camera configuration scheme.

In summary, by applying an adaptive reconfiguration policy, the quality of the application, that is, the detection ratio, is similar to high-quality configuration while the CPU overhead is significantly reduced. The total CPU overhead difference between high-quality configuration and dynamic configuration gets bigger when the duration of dimming is shorter. This eventually provides room for the resource constrained IoT camera to effectively handle more than one application on board.

Adaptation time for context-aware reconfiguration In this experiment, we evaluate how much time it takes to understand the application requirement and to reconfigure the setting on the IoT camera when the context changes.

- Controller-based: video streaming and context monitoring is done on the camera while the controller is responsible for selecting a new configuration for the camera[5], [6].
- Camera-based: video streaming, context monitoring and camera reconfiguration are done at the IoT camera.

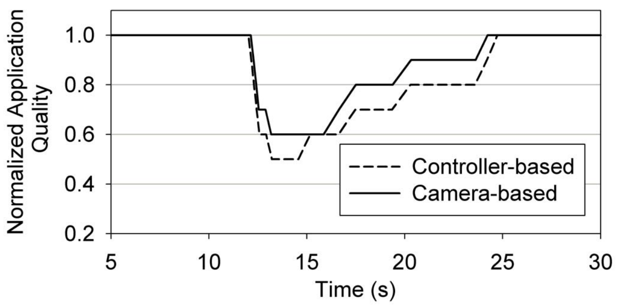


Fig. 12: Change in the quality of application with camera-based reconfiguration vs controller-based approach.

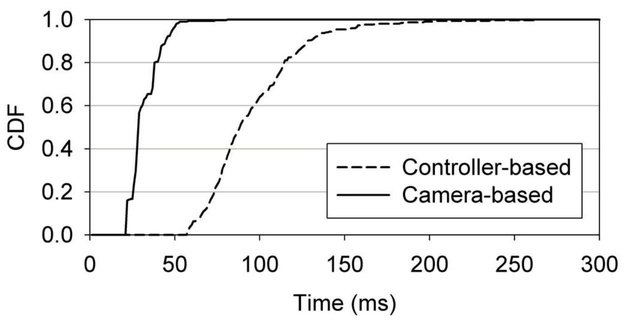


Fig. 13: CDF of reconfiguration time of camera-based vs controller-based approach .

The results are depicted in Figure 12. The x-axis and y-axis show the running time and the normalized application quality of a tracking application respectively. The normalized application quality refers to a normalized moving average value of correctly detected objects over given time. The vertical lines at 13th and 24th second show a drop and rise when a environmental context changes (i.e. a sudden change in luminosity). The quality of the application drops in all cases at 13 seconds into the experiment. Controller-based approach shows that the system reverts (0.1sec) back to a normal value because only meta data (brightness) is sent for evaluation. The quality of detection ratio in the proposed camera-based approach recovery is faster (0.025 sec) as reconfiguration is done on the camera as well. Throughout the context change duration, controller-based approach fails to accurately detect approximately two seconds compared to the camera-based approach due to transmission of context values over network.

Figure 13 shows CDF of the processing time for both approaches, dashed line is the result for the controller-based approach and solid line is the result of the proposed one. As shown in the figure, the reconfiguration takes about 58ms to 197ms for the controller-based approach and 22ms to 80ms for camera-based approach. The reconfiguration time in the existing approach is much varying and larger than the proposed approach. This is due to transmission delay of transmitting

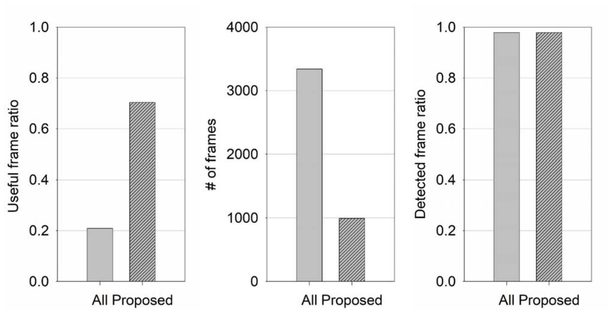


Fig. 14: The total number of frames, useful frame ratio, and detected frame ratio of different IoT camera allocation policies.

frames through the network, which are not relevant in the proposed approach.

Thus, the proposed system has up to 2.8 times better responsiveness compared to one without reconfiguration at the IoT camera.

In video analytics applications, it is important to adapt to the environmental changes quickly so to assure application QoS. We take advantage of the IoT camera's ability to judge the situation on its own, which is the most adaptive to the systems we have so far offered.

D. Dynamic IoT camera selection

Application-aware IoT camera selection.

We evaluate the performance of the application-aware IoT camera selection scheme in our system using an object detection scenario. The object detection application's quality depends on the number of *useful* video frames which actually contain necessary information such as objects of interest or a target area to be checked. The simplest way to obtain the highest object detection rate is to use all video frames taken from all connected cameras which is not desirable due to explosive amount of video frames delivered through the network. Our system can select IoT cameras which can actually provide useful frames for an application by understanding its requirements. We compare our system with the approach of using all frames using three metrics: the number of delivered frames, useful frame rate which indicates the proportion of useful frames to received frames in the application, and object detection rate. For object detection, we use videos which captured a scenario of a person walking around in a seminar room. The total time length is 120 seconds, and during that time, a person moves around the seminar room in one direction.

Figure 14 shows the comparison between our system and the approach using all frames. Each graph from (a) to (c) shows the total number of frames delivered to the application, useful frame ratio, and object detection rate, respectively. The approach using all frames delivers 3340 frames to the application while our system delivers only 992 frames. The number of useful frames during the experiment is 698 where each frame contains a moving person. The approach using

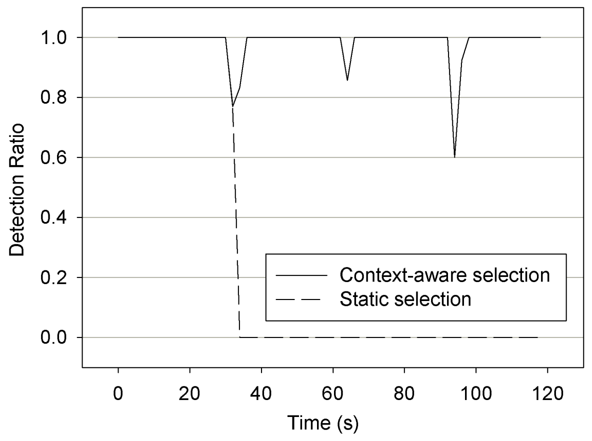


Fig. 15: Application QoS while moving to different locations.

all frames then shows useful frame ratio 0.21 because many frames do not contain a target person. On the other hand, each IoT camera in our system understands the application's requirement and checks if a captured frame contains a person, and only frames including a person are sent to the application. The useful frame ratio is thus 0.70. For both approaches, the object detection ratio is more than 0.98. Therefore, our system can reduce a significant amount of network overhead for delivering video frames while the application quality is not reduced.

Context-aware IoT camera selection.

In order to evaluate our system's ability to dynamically reallocate IoT cameras based on changes in environmental contexts, we compare our system with an existing approach which has a static camera allocation policy. We use the same scenario of a moving person in a seminar room and measure object detection rate as application quality. The experimental result is shown in Figure 15. The X-axis is the time elapsed during video tracking application execution, and the Y-axis is the quality of the tracking application. The detection rate of the existing static allocation scheme (orange line) goes down to zero after 30 seconds while our system (blue line) constantly shows high detection rate with three points of lower values where the selection of IoT cameras is changed. This is because in the existing approach, the camera used by the tracking application is not changed although the person has moved to the area of another camera's sight. Without context awareness, the situation of a target person moves away from a camera to another one cannot be handled. On the other hand, our system dynamically selects an appropriate IoT camera as a result of running a dynamic camera selection policy. Since the policy recognizes the context that a person moves out of a certain camera's sight, a set of cameras used by the application is reconfigured and the application quality is kept high.

E. Discussion

As described in IV.B, the proposed system incurs 1.57 sec more than one without camera virtualization at launch time. This can be critical for some real-time applications. For instance, if a surveillance application recognizes a specific person and wants to run a human tracking application right after, 1.57 seconds would be long enough for the person to run away from sight. In this case, our system can launch a general-purpose container at camera initialization and keep it paused for faster restart, reducing the launch time. Since the time to resume a paused container in Docker is 0.4 sec on average, we can reduce the delay by than a second. However, the paused container keeps resources of its host machine and they cannot be utilized by other running containers. The best way of solving this issue is to apply more lightweight virtualization schemes which are actively discussed in the literature [24], then the amount of time for launching a container can be reduced further.

V. Related Work

We categorize several prior works regarding video analytics in edge computing environments.

A. video analytics in edge computing environment

Recent studies on video analytics for edge computing bring part of an application's logic to an edge controller and the cameras themselves to reduce the overall amount of data which needs to be sent to the cloud. Vigil [5] proposes a content-aware filtering and selection mechanism to reduce network overhead among edge cameras nodes. Similarly, EECS [6] considers battery driven camera nodes and proposes offline and online methods to maintain a high detection accuracy while saving as much energy as possible and thus extending the lifetime of the cameras. ACTION [7] studies how to combine and filter several partly overlapping camera views of a scene to get the highest accuracy possible while respecting a given bandwidth constraint. As we can observe from these works, their architecture consists of one central controller (or master) that coordinates a series of slave nodes to reduce the total bandwidth usage at the edge while also respecting other metrics like accuracy or energy consumption. Our work is different from these works in that IoT cameras in our system can detect and adapt to context changes without being controller dependent.

VideoStorm [17] proposes a real-time video stream analytics system over large clusters. In one phase, their offline profiler creates a query-resource quality profile which consists of the camera and algorithm parameters that work best for a type of query. Then, the online scheduler allocates resources to each query based on the previously created quality profile to maximize performance on quality and lag. This work is orthogonal to ours since our goal is to enable the application-aware virtualization of IoT cameras so that the camera nodes can make decisions upon dynamic context changes. The result of our video stream could be sent to VideoStorm for further analysis.

In Lavea [25], mobile devices offload tasks to an edge server. At the same time, an overloaded edge server can offload tasks to other edge servers / edge clouds. The authors devised an edge-first design and formulated an optimization problem to select the tasks to be offloaded in order to minimize the response time. In this architecture, video processing tasks are offloaded to the edge cloud, while our work focuses on having the processing tasks done at each IoT camera without the help of an intermediate controller or server.

Firework [18] suggests a hybrid cloud and edge processing platform for video analytics. The platform is also thought for data sharing since the authors foresee that camera nodes could belong to different stakeholders. While Firework implements its framework as a Java class, we deploy docker containers, which offer a greater flexibility for application providers. Also, unlike our work, Firework cannot adapt the camera views to the applications' needs.

Panoptes [26] allows multiple applications to share a single steerable camera by virtualizing the camera's view (orientation, resolution, zoom). If the camera has to steer away from the application's desired view, its virtual view replays the last available virtual image. The authors design a mobility-aware scheduler that also learns motion patterns to maximize the capture of events of interest for the applications. However, this approach still requires an external support, not by itself, for performing an adjustment task unlike the proposed one.

Chameleon [27] exploits deep convolution neural network (NN) for efficient video analytics. However, it is too heavy for a controller to periodically update the NN hyper parameters depending on a situation (e.g. less video capturing during a less traffic situation). Instead, the controller probes sample frames and selects the best set of NN configuration parameters considering temporal and spacial correlation of the video streams over time and across multiple video streams. As discussed in the paper, it is a challenging task to figure out how this job can be done by an IoT camera.

VI. Conclusion & Future work

In this paper, we propose an edge IoT camera virtualization architecture for video analytics applications. In order to enable an edge IoT camera to understand application requirements and environmental contexts, we define an ontology-based knowledge base with relationships between application features and IoT camera parameters. Leveraging container-based virtualization technique, we virtualize an IoT camera to adjust frames for application requirements. Our proposed architecture can select appropriate IoT cameras considering application requirements and dynamically changing environmental contexts. The experimental results with our prototype implementation verify that the proposed scheme enables edge IoT cameras to effectively adapt to environment changes in terms of application quality and computation and network overhead.

Our future work includes the following points. First, we plan to extend our architecture with reinforcement learning to dynamically adjust camera resource reconfiguration based on the outcome of running video analytics applications. This would allow us to find more complex relationships between the parameters for more accurate reconfiguration on IoT cameras. It would also help us improve the efficiency of multi-camera cooperation support we're currently working on. Second, multiple applications will be handled more efficiently by applying priorities to applications so that critical applications aren't downwardly adjusted by another application during the situation.

Acknowledgment

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.B0717-17-0034,Versatile Network System Architecture for Multi-dimensional Diversity). We would like to thank Dionisio Vendrell Jacinto and Hongmin Yang for their contributions on the implementation and testing of this work.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *IEEE Personal communications*, vol. 8, no. 4, pp. 10–17, 2001.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[4] M. Patel, B. Naughton, C. Chan, N. Sprecher, S. Abeta, A. Neal *et al.*, "Mobile-edge computing introductory technical white paper," *White Paper, Mobile-edge Computing (MEC) industry initiative*, 2014.

[5] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 2015, pp. 426–438.

[6] T. Dao, K. Khalil, A. K. Roy-Chowdhury, S. V. Krishnamurthy, and L. Kaplan, "Energy efficient object detection in camera sensor networks," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 1208–1218.

[7] T. Dao, A. Roy-Chowdhury, N. Nasrabadi, S. V. Krishnamurthy, P. Mohapatra, and L. M. Kaplan, "Accurate and timely situation awareness retrieval from a bandwidth constrained camera network," in *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*. IEEE, 2017, pp. 416–425.

[8] S. Y. Jang, H. Choi, Y. Lee, B. Shin, and D. Lee, "Semantic virtualization for edge-iot cloud: issues and challenges," in *Proceedings of the 2nd Workshop on Cloud-Assisted Networking*. ACM, 2017, pp. 55–60.

[9] Y. Elkhatib, B. Porter, H. B. Ribeiro, M. F. Zhani, J. Qadir, and E. Rivière, "On using micro-clouds to deliver the fog," *IEEE Internet Computing*, vol. 21, no. 2, pp. 8–15, 2017.

[10] "Raspberry Pi," https://www.raspberrypi.org/, accessed: 2017-12-11.

[11] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[12] "Docker," https://www.docker.com/, accessed: 2017-12-11.

[13] W. Wang, S. De, R. Toenjes, E. Reetz, and K. Moessner, "A comprehensive ontology for knowledge representation in the internet of things," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*. IEEE, 2012, pp. 1793–1798.

[14] A. I. Maarala, X. Su, and J. Riekki, "Semantic reasoning for context-aware internet of things applications," *IEEE Internet of Things Journal*, vol. 4, no. 2, pp. 461–473, 2017.

[15] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[16] Y. Xiao, M. Noreikis, and A. Ylä-Jaäiski, "Qos-oriented capacity planning for edge computing," in *Communications (ICC), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1–6.

[17] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance." in *NSDI*, vol. 9, 2017, p. 1.

[18] Q. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Firework: Data processing and sharing for hybrid cloud-edge analytics," *IEEE Transactions on Parallel and Distributed Systems*, 2018.

[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] P. Liu, D. Willis, and S. Banerjee, "Paradrop: Enabling lightweight multi-tenancy at the network's extreme edge," in *Edge Computing (SEC), IEEE/ACM Symposium on*. IEEE, 2016, pp. 1–13.

[21] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[22] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[23] P. Viola and M. J. Jones, "Robust real-time face detection," *Int. J. Comput. Vision*, vol. 57, no. 2, pp. 137–154, May 2004. [Online]. Available: https://doi.org/10.1023/B:VISI.0000013087.49260.fb

[24] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, "Consolidate iot edge computing with lightweight virtualization," *IEEE Network*, vol. 32, no. 1, pp. 102–111, 2018.

[25] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 2573–2574.

[26] S. Jain, V. Nguyen, M. Gruteser, and P. Bahl, "Panoptes: servicing multiple applications simultaneously using steerable cameras." in *IPSN*, 2017, pp. 119–130.

[27] J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, and I. Stoica, "Chameleon: scalable adaptation of video analytics," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 2018, pp. 253–266.