BarMan: a Run-Time Management Framework in the Resource Continuum

Michele Zanella, Filippo Sciamanna, William Fornaciari

name.surname@polimi.it, DEIB, Politecnico di Milano

Abstract

Over the last years, the number of IoT devices has grown exponentially, highlighting the current Cloud infrastructure limitations. In this regard, Fog and Edge computing began to move part of the computation closer to data sources by exploiting interconnected devices as part of a single heterogeneous and distributed system in a computing continuum viewpoint. Since these devices are typically heterogeneous in terms of performance, features, and capabilities, this perspective should encompass programming models and run-time management layers. This work presents and evaluates the BarMan open-source framework by implementing a Fog video surveillance use-case. BarMan leverages a task-based programming model combined with a run-time resource manager and the novel BeeR framework to deploy the application's tasks transparently. This enables the possibility of considering aspects related to the energy and power dissipated by the devices and the single application. Moreover, we developed a task allocation policy to maximize application performance, considering run-time aspects, such as load and connectivity, of the time-varying available devices. Through an experimental evaluation performed on a real cluster equipped with heterogeneous embedded boards, we evaluated different execution scenarios to show the framework's functionality and the benefit of a distributed approach, leading up to an improvement of 66% on the frame processing latency w.r.t. a monolithic solution.

Keywords: Fog Computing, Resource Management, Programming model, Task allocation strategy

1. Introduction

With the advent of IoT, a huge number of embedded devices (sensor nodes, actuators, mobile and wearable devices...) can be interconnected to acquire increasing amounts of data that are further processed remotely for a variety of purposes [1]. The numbers are quite impressive: according to IDC [2], there will be 60 billion of connected entities by 2025, generating about 80 ZB of raw data. This should lead to 1256.1 Billion USD of market share by 2025. The main problem is that the enormous amount of data is mainly transferred to the Cloud for processing, risking the saturation of the current network infrastructure. Nevertheless, the increasing size of required data-centers and their power consumption will become unsustainable in a long-term perspective. Another problem concerns the current IoT infrastructure, that relying on an internet connection, is not

Contributions	Fine-grained Distributed Resource Management			Application Programmability		Real	Real-world
	Device mobility	Resource Heterogeneity	Allocation & Balancing	Unified model	Transparent task distribution	test-bed	use-case
BarbequeRTRM[1]	✓	✓	✓				
libmango[1]		✓		✓	✓		
BeeR					✓		
SmokyGrill		✓				✓	
Video surveillance							✓

Table 1: Summary of the contribution of the paper with respect to the open challenges

suitable for real-time applications. Consequently, new approaches emerged in the last years, aiming to shift part of the data processing back to the edge devices, where data is generated [3, 4]. Such a strategy allows us to [5]: (a) extract small-sized information from the raw data, avoiding a high bandwidth consumption; (b) reduce the transmission latency to the Cloud, thus enabling faster response; (c) increase the reliability of the system leveraging local nearby devices; (d) exploit near-to-data devices and limit data-centers growth and energy consumption. In this regard, the Fog computing paradigm [6] includes a layer between the edge and the Cloud that can provide computational and storage services. This paradigm has been extended in the next years to include all aspects from networking to management, introducing the *resource continuum* concept [7, 8]. This leads to a dynamic, modular, and heterogeneous distributed system, that gathers different paradigms by implementing a resource continuity from remote high-performance infrastructures to the near-to-data embedded devices.

In this context, different open research challenges has been highlighted in the literature [9, 10]. Firstly, the possibility to *control and manage* such a system. In particular, in order to fully exploit the available devices and their heterogeneity advantages, it is required a *unified programming model* that allows the development of modular applications and a mechanism to provide a *transparent distribution* of application's tasks. Secondly, there is the need for a *run-time management system* able to: (a) efficiently discover and organize nodes and (b) map application execution units to the available resources in order to meet the application requirements. In this regard, *proper task mapping and resource allocation strategies* have to consider optimizations at system-wide (e.g., performance, balancing, energy consumption, reliability...) and device levels (e.g., load, utilization, energy budgeting, temperature...). Finally, at experimental standpoint, there are many synthetic benchmarks and simulation software. However, in the research field there is a high demand for *real-world use-case applications* that can be used in experimental settings, as well as an *accessible and physical hardware test-bed* to perform measurements.

Starting from these considerations, this paper aims at delivering a run-time managed open-source framework (*BarMan*) for executing multitasking applications on heterogeneous distributed embedded systems able to optimize the resource utilization, taking into account the per-task application requirements. This opens up the possibility of considering aspects related to the energy and power dissipated by the devices and the single application. Table 1 summarizes the contribution of our work concerning the aforementioned challenges. In particular, we use the *BarbequeRTRM* open-source resource manager (RM), enforced with an allocation strategy that optimizes performance, basing on run-time resource status and network availability. The manager is integrated with our *libmango* library, which is a resource-aware task-based programming model that hides the underlying resource heterogeneity. These two tools are extended with the novel

	Scala-bility	Device Mobility	Resource Heterog.	Energy Awareness	QoS Mgmt	Openness	Run-time config.
MobileFog	✓	✓	✓				
Distribuited Dataflow	✓		✓		✓	✓	
FogFlow	✓		✓		✓	✓	
Our Solution	✓	✓	✓	✓	✓	✓	✓

Table 2: Summary of Fog management frameworks

BeeR framework for multi-node execution support. Finally, we developed a real-world Fog video surveillance application to evaluate the frameworks on a self-built real cluster (*SmokyGrill*) equipped with different embedded boards. It is worth to be mentioned that the developed use-case and framework allow us to present the methodology that can be further generalized and also used for other type of Fog scenarios (e.g., automotive [11], health care...).

The paper is structured as follows: Section 2 presents the state-of-the-art related to the addressed problems. Section 3 describes the BarMan framework used to develop and execute the Fog distributed application, which is presented in Section 4. In Section 5, we present a task allocation strategy for low-latency applications. Finally, in Section 6, we analyze the results of the experimental phase. Section 7 concludes the paper.

2. Related Works

In recent years, the research interest in the open challenges of Fog computing has grown enormously. In the following, we will provide an overview and comparison between the main achievements, focusing on aspects related to our work: management frameworks and task allocation strategies. Table 2 shows the provided and enabled features of our solution with respect to the analyzed ones.

2.1. Management frameworks

While research concentrated on architectural aspects of Fog systems, not many proposals provide complete and available frameworks able to implement such architectures[12]. In particular, two main aspects need to be considered in evaluating the frameworks: (1) how the application can be developed and integrated, (2) how the resources are shared and allocated to the application. The first aspect is related to the availability of a suitable programming model; the latter requires a system to manage applications and Fog devices.

Hong et al. [13] developed *MobileFog*, which provides a high-end programming model and allows dynamic scaling of resources allocated to applications. In their work, authors consider various heterogeneous nodes organized hierarchically and associated with a certain geophysical location. The application is formed by different processes that can be mapped on the various devices (i.e., Fog or Cloud). However, this solution does not allow to define and monitor the application's QoS requirements; moreover, it does not consider aspects like device mobility, and it is not open source nor standardized, limiting the inter-operation with other systems.

In 2018, Cheng et al. [14] extended the well-known Dataflow programming model to operate in a Fog environment, developing *FogFlow*. In this solution, applications are represented with a directed graph where each node can have inputs, outputs, and runs as an independent processing unit. The management system is typical deployed on the Cloud, and it is in charge of: (a) providing interfaces to manage deployed services; (b) managing docker image repository; (c) service orchestration; (d) device discovery. Although the framework supports heterogeneity, openness and scalability, it is still dependent on the Cloud. Furthermore, it does not support device mobility and fine-grained resource management of nodes (e.g., power management, energy budgeting, app tuning...).

Similarly, Distributed DataFlow [15] presents a distributed approach where DDF applications do not have a central management but each node takes its own decision. Furthermore, the scenario evaluated is only IoT (due to type of tasks and inter-communication protocols), which differs from our computing intensive scenario.

2.2. Tasks allocation strategies

To exploit the frameworks, it is necessary to feed the management system with a proper strategy able to decide how to distribute the applications (or their different modules) onto the various devices. In this sense, the strategy takes decision considering one or more optimization metrics: (a) computation latency; (b) network latency or connectivity; (c) resource utilization; (d) power consumption; (e) energy budget. In the context of this work, we present some research solutions focusing on aspects (a), (b) and (c). It is worth mentioning that all of these works are based on simulation and are not integrated with real frameworks nor tested on real hardware.

In [16], authors consider computing latency on a system composed of different source nodes connected to all the available computing nodes and proposed three different policies to minimize the blocking probability (i.e., the ratio between the rejected tasks and the total number of tasks). Then, they analyzed the results, noticing that it is possible to obtain the best performance by lowering the latency due to its quickly available resource, thus reducing the turnaround time and rejection probability. Similarly, Souza et al. [17] propose an ILP model to minimize the total service latency. Differently, Mukherjee et al. in [18] developed an algorithm to distribute the workload while reducing the overall response time and cost of computation. Tanganelli et al. [19] developed a strategy to design and deploy the devices in a cyber-physical environment minimizing installation cost, it also considers the mapping of different tasks based on available resources and location. Similarly, authors in [20] consider the device's deployment problem, including also energy-related aspects. Brogi et al. [21], instead, proposed a centralized QoS-aware deployment approach to find eligible deployments of IoT applications over a Fog infrastructure in a context-aware manner considering latency and bandwidth usage during optimization. Similarly to [16], but considering different QoS metrics, [22] presents an algorithm that chooses whether to execute the application in a single device, on the Cloud, or distribute it among the available nodes, taking into account resource constraints. Finally, Skarlat et al. [23] proposed a hierarchical approach based on a genetic algorithm to find a suitable service placement solution. Their approach takes into account Fog devices capabilities, cost of the solution and latency.

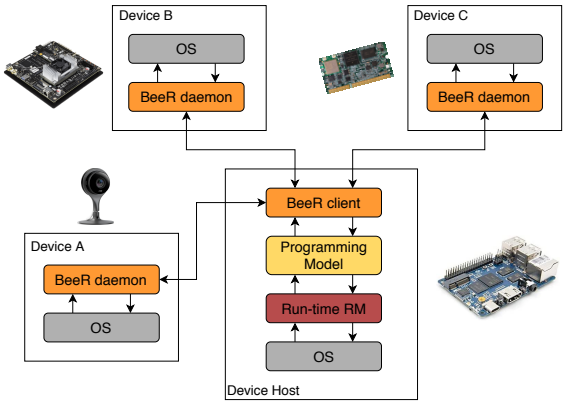

Figure 1: An overview of the BarMan framework's modules.

3. The BarMan framework

The *BarMan* framework integrates two existing frameworks and extends their capabilities to enable a run-time managed execution of multi-tasking applications, on a heterogeneous and distributed (embedded) computing system. Mainly, BarMan integrates a resource manager for high-end embedded and HPC systems (the BarbequeRTRM), the `libmango` programming model for modular applications and the BeeR framework to make them work with distributed systems. Considering Figure 1, in the following paragraphs, we focus on each of them, describing their main features.

3.1. The Barbeque Run-Time Resource Manager

The *Barbeque Run-Time Resource Manager* (BarbequeRTRM)[24] [2] is a modular, open-source and extensible run-time RM developed to transparently manage the allocation of computing resources to multiple concurrent applications while taking care of both applications' QoS requirements and dynamic resource availability. Currently, it supports several types of platforms (e.g., embedded, HPC, mobile...) and its distributed version supports multi-device cooperation. The key aspect of such a manager is the so-called *Adaptive Execution Model*, which allows the applications to be controlled and reconfigured by the manager in order to respect performance/power constraints. In this sense, various allocation policies can be easily plugged to deal with specific optimization metrics (e.g., latencies, bandwidth, power consumption...).

In BarMan, the RM is plugged into the programming model to interact with Fog applications. Thus, we can implement our allocation strategy (called *policy* in the jargon) to evaluate the entire setup.

3.2. The Programming Library

The `libmango` Programming Library [25][3] was born with the idea of providing an API, through which implementing multi-tasking applications running on heterogeneous

[2] https://bitbucket.org/bosp
[3] https://bitbucket.org/mango_developers/mangolibs

HPC platforms. This means that each task may support multiple target architectures, such that the actual execution of them (also called *kernels*) can be offloaded on different possible computing devices. The tasks can run concurrently, assuming that the inter-dependencies are satisfied (i.e., a parent task is providing input data). The goal of the library is the same as the well-known OpenCL, with a main big difference: the application code must provide a *task-graph* based representation of itself, leaving the task mapping to the underlying run-time RM. This, from one side, lifts the developer from the burden of embedding the task mapping logic in the application code, while leaving to the system management software the possibility of optimizing the resource assignments, based on metrics like performance, load balancing, energy efficiency and so on. In our model, the task-graph includes two types of nodes: *tasks* and *buffers*This extends the common DAG definition used in most of the state-of-the-art works, making more explicit the inter-task dependency, also in terms of memory requirements (i.e., buffers). Looking at programming paradigms like the aforementioned OpenCL, in fact, the tasks exchange data via memory buffers. From a resource management standpoint, this approach fits better the cases of future heterogeneous platforms [26]. Here, a computation node features multiple processing units and memory nodes. Therefore, the resource allocation process must map both the tasks to the processing units and the buffers to the memory nodes. This is to optimize the task processing time, the memory access latencies and the load of interconnected infrastructure. As it should be easy to imagine, this applies to distributed systems as well, where the latencies due to data transfers are orders of magnitude greater than the values typically observed on single-node systems.

In BarMan, we wanted to exploit and extend this approach to the distributed embedded systems scale. The result is a unified programming model, through which we can maximize the portability of multi-tasking applications between heterogeneous single-node and distributed systems, leaving the run-time optimizations to the system management layers.

3.3. BeeR: Enabling Multi-node Application Execution

We extended the aforementioned tools with the BeeR framework, an open-source[4] lightweight library and run-time daemon, to provide dynamic content offloading to remote embedded devices. It consists of two components: (1) a client library used to extend the programming library and (2) a daemon that can run on a device and handles requests from applications.

During the resource allocation phase, the run-time RM will provide an appropriate allocation plan starting from the selected policy, taking into account all the available local and remote systems resources. The BeeR client library, then, will manage the remote allocation of tasks and buffers by defining, for each remote system, the set of resources assigned by the manager and enabling the communication with the BeeR daemon instances running on the specific devices. In this regard, the daemon uses a combination of the ZeroMQ library for implementing the TCP client-server message exchange pattern, and the Google Protocol Buffer for message serialization and the definition of the interfaces.

On the device side, each BeeR daemon receives a portion of the application task-graph, represented by a subset of tasks and buffers. It allocates the buffers on the

[4]https://bitbucket.org/dark-corner/beer-framework

local device through the *POSIX* shared memory API and natively executes the kernels, leveraging the *fork+exec* system calls.

Additionally, the daemon provides minimal dependability support, allowing the resource manager to retrieve status information regarding each managed device.

4. The Smart Surveillance Application

We decided to realize an application [5] that could fully exploit BarMan's potentials and provide workload heterogeneity. To do so, we started from one of the Fog simulated use-cases proposed by Gupta et al. [27]. In their work, they have proposed a modular surveillance system that could track the moving objects in a given area acting on parameters (PTZ) of multiple security cameras. From this application model, we removed the PTZ control module and considered the presence of a single security camera. Then, we developed a task-based application leveraging the *OpenCV* library for the image processing parts. In the following sections, we will briefly present the application's design and optimizations.

4.1. Application's structure

The application is formed by the following four kernels:

- *Motion kernel*: this kernel reads the video streams and, after some pre-processing steps, performs background subtraction in order to find the parts of the image in motion. The processed frame and the resulting bounding boxes are forwarded to the *classifier kernel*;
- *Classifier kernel*: this kernel receives video frame and the motion information to performs object classification using a deep neural network (DNN). After the classification task is completed, the results are compared with the results of the motion in order to determine which of the classified objects are in motion. Then, results are sent to the *tracker kernel*;
- *Tracker kernel*: this kernel uses the results of *classifier kernel* to track the moving objects through a tracking-by-detection approach. Tracks information are then forwarded to the *GUI kernel*;
- *GUI kernel*: this kernel presents a simple user interface where the video stream is displayed and the tracked objects are enclosed in rectangles showing their id and class.

4.2. Latency optimization

During the development, we noticed that most of the time needed to complete a computation on a single frame was spent in the *classifier kernel* and it is bound to the choice of the neural network. For that reason, we decided to develop a second version (Figure 2) in which the *classifier kernel* is detached from the main workflow and the *tracker kernel* uses the results of the *motion kernel* to perform tracking. Such an approach

[5] https://bitbucket.org/fsciam/fogsurveillance

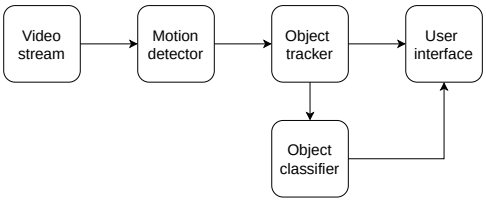

Figure 2: Parallel version of the use-case application

allows us to unlink execution times of the tracking and classification tasks, opening the possibility of choosing the neural network based on the quality of service (QoS) required or on the available resources. On the other hand, the detections generated by the *motion kernel* may be less accurate, and the delay introduced between the start of the tracking and the classification of the object may not be compatible with the requirements of some surveillance scenarios. However, this approach leads us to obtain a 20 times improvement in the execution time.

5. Run-time management: allocation policy

Analyzing our application, we can identify three main QoS metrics: detection accuracy, detection latency and tracking latency. The nature of the application's structure makes complex to find an optimization problem that could consider all the metrics at once. This, combined with the fact that detection accuracy and latency are exclusively bounded to the DNN chosen, led us to split the tasks scheduling problem into two steps. Firstly, we choose a DNN that meets and balance the accuracy and latency requirements. Then, we solve the task allocation problem aiming to minimize latency. In the following subsection, we will formally define the problem and explain our design choices.

5.1. Problem statement: LAVA Policy

Starting with the DNN choice problem, we decided to characterize each DNN with two parameters: a_n, $\beta_{n,r}$. The former indicates the accuracy of the DNN n registered on the COCO dataset [28], while the latter is used as latency indicator of the DNN n when executed on a resource type r. This parameter is defined as $1 - \frac{Latency_{n,r}}{\max_{x \in \mathbb{N}} Latency_{x,r}}$ for each resource type available on the devices. For simplicity, we assumed that the value $\beta_{n,r}$ does not depend on the device used for the latencies measurements.

According to the usage scenario, it may be more important to achieve minimal latency rather than high accuracy in classifying objects in the video stream. For that reason, we introduce the weight γ, whose value is between 0 and 1, which allows us to balance the latency and the accuracy of the DNN depending on the requirements.

All the parameters are then inserted in the objective function (1) where y_n is the optimization variable that indicates if the $n-$DNN is selected. The function considers both DNN mean latency and accuracy to select the best DNNs available depending on the desired behavior expressed by the value of γ. In particular, with $\gamma = 0$ we will

obtain the DNN with maximum accuracy, while with $\gamma = 1$ we will obtain the one with minimum latency.

$$\text{minimize} \quad \sum_{n \in \mathbb{N}} (1 - (\gamma \times \beta_{n,CPU} + (1 - \gamma) \times a_n)) \times y_n \tag{1}$$

Once we have identified the best DNN $\hat{n}$, we can find the tasks placement that guarantees minimum application's latency.

To do so, we had to characterize each device d indicating: the device's total and available CPU quota (c_d^{total}, c_d^{free}), acceleration possibility (s'_d), input and output capabilities (in'^i_d, out'^o_d). We, also, indicated for each task t: the task's required CPU quota ($c^{req}_{t,d,r}$), acceleration possibility (s_t), usage of DNN (n_t), input and output requirements (in^i_t, out^o_t). Moreover, for each task we defined the parameter $e^{\alpha_d}_{t,d,r}$ that is the latency of task t on device d with load α_d using resource type r. The latencies are computed at design time, while the α_d term is chosen at run-time based on the device's actual load. Then, we introduced $L_{t,d,r}(4)$ that is the estimated latency for task t on device d considering resource type r and the load to which the device d is subjected. The load on device d is computed as shown in equation (3) considering the ratio between the free resources of type CPU and the total number of them.

The total execution time of a task is composed of its execution, reception of the input data and transmission of the result. The presence of wireless and wired connections with different bandwidths makes it necessary to estimate the input/output transfer times of tasks during scheduling to achieve latency minimization. Given the amount of data exchanged and the need to pass all communications through the host node, the possible overhead of multi-hop communication between the host and a wireless device could not be compatible with strict requirements on latency, so we considered only point-to-point wireless connections. In this regard, Chanho Jin et al.[29] investigated the possibility of real-time data transmission of high-definition images for wireless medical, showing that data-rate transmission of Wi-Fi Direct outperforms WLAN architecture. In the case of wireless links between the devices, the host node can measure, for each wireless device, the available bandwidth of the connection.Those measures are periodically made using network analysis tools like *Iperf* and the estimates are updated using an exponential average formula. In particular, for each wireless device, we have:

$$B^{up}_{d,n+1} = \alpha \times BW^{up}_{d,n} + (1 - \alpha) \times B^{up}_{d,n}$$

$$B^{down}_{d,n+1} = \alpha \times BW^{down}_{d,n} + (1 - \alpha) \times B^{down}_{d,n}$$

Where $BW^{up}_{d,n}$ and $BW^{down}_{d,n}$ are the measured bandwidth of the link between the host and device d at time n in both directions, while $B^{up}_{d,n+1}$, $B^{down}_{d,n+1}$ and $B^{up}_{d,n}$, $B^{down}_{d,n}$ are respectively bandwidth estimates at time n and $n+1$. The weight $\alpha \in [0, 1]$ controls the balancing between recent and past history in our prediction.

For what concerns wired connection delay, we decided not to consider them because they are negligible if compared to the delays of wireless links.

Finally, after the addition of the maximum quantity of input data that the task t sends/receives ($data^{in}_t$,$data^{out}_t$) at each iteration, we introduced: the parameter w_d that represents if the device d is connected wirelessly and the variable $x_{t,d,r}$ that represents

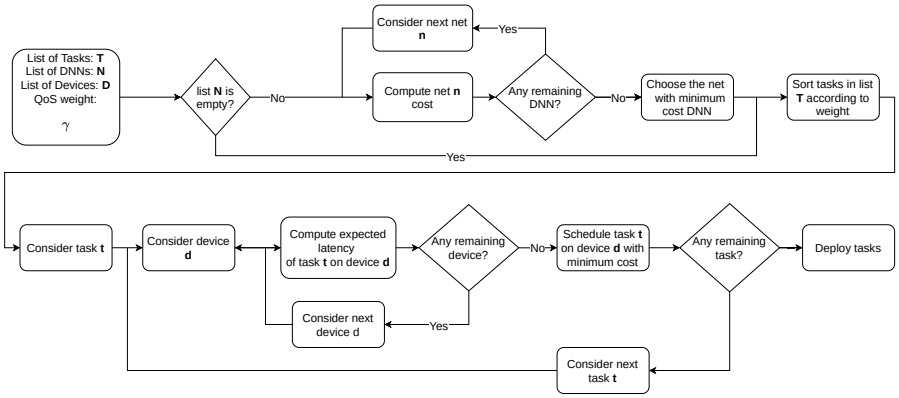

Figure 3: LAVA policy flow chart.

if the task t is executed using resource type r on device d. The cost function to be minimized (2) is simply the sum of all the $L_{t,d,r}$.

$$\text{minimize} \quad \sum_{t\in\mathbb{T}}\sum_{d\in\mathbb{D}}\sum_{r\in\mathbb{R}} L_{t,d,r} \times x_{t,d,r} \tag{2}$$

$$\text{subject to} \quad \alpha_d = ceiling((1 - \frac{c_d^{free}}{c_d^{total}}) \times 100) \tag{3}$$

$$L_{t,d,r} = e_{t,d,r}^{\alpha_d} \times (1 - n_t \times \beta_{\hat{n},r}) + w_d \times [(\textstyle\sum_{t'\in\mathbb{T}}\sum_{r\in\mathbb{R}} x_{t',d,r}) \times (\frac{data_t^{in}}{B_d^{up}} + \frac{data_t^{out}}{B_d^{down}})] \tag{4}$$

5.2. LAVA heuristic

We proposed a greedy policy to solve the problem formally defined in the previous section. As shown in Figure 3, the *LAtency Versus Accuracy* (LAVA) policy takes a list of application tasks, a list of devices available, a list of DNNs and a QoS weight, and returns a sub-optimal solution. First of all, we select the optimal DNN based on γ value and cost function (1). Then, we iterate over tasks list that has been ordered in descending order depending on the value $c_{t,d,r}^{req} \times (1 + out_t^{SCREEN} + in_t^{VIDEO})$. For each task, we search for the device and resource that will minimize execution and communication latencies, also taking into account the slowdown that the task could lead to tasks, if any, that have already been scheduled on that device. We decided not to discard devices on which the available resources are not enough to satisfy task requirements. By doing so, we will be able to schedule all the tasks. However, in the search for the best placement for the task, the fulfillment of its requirements on resources is taken into account to make the best choice possible. Once we have found an assignment for each task, the algorithm ends and the scheduling information is returned.

6. Experimental evalution

The experimental evaluation consisted of two main steps. At first,we analyzed the execution latency of the kernels on the cluster's boards. Then, we executed the distributed

application testing the RM, the *Beer* framework and the policy on multiple scenarios. At this step, we compared the performance improvements with respect to a monolithic execution. This aims at simulating the worst case in which non-modular applications are executed on a single low-end edge device.

For all the tests, we used a short video taken from the *VIRAT* dataset [30], simulating the stream of a surveillance camera. The video is recorded at 30 FPS and 1920x1080 pixels resolution. In the segment used in our tests, that counts a total number of 506 frames.

6.1. Experimental Setup: SmokyGrill

To perform the experimental evaluation, we built a real cluster, called *SmokyGrill*, with different embedded boards creating a heterogeneous distributed hardware setup. In this regard, we chose three board types to replicate the availability of low-end and high-end devices in real situations.

First of all, we deployed and run the BarbequeRTRM on a *Banana Pi BPI-R*1, that acts as the starting node of the test application. It is equipped with a 1.0 GHz A20 ARM Cortex A7 dual-core processor and 1 GB DDR3 of RAM. Then, we deployed the BeeR daemons on the interconnected boards to distribute the application's tasks. In particular, we used the following devices:

a) two NXP Freescale i.MX6Q SABRE development boards, featuring an ARM 32 bit Cortex A9 1.2 GHz quad-core CPU and 1 GB DDR3 SDRAM; the default `cpu_freq` governor is set to `performance` with boosting disabled;

b) one Jetson TX2 module with a 2.0 GHz quad-core ARM 64 bit Cortex A57 and 8GB LPDDR4 memory; it is also equipped with 256 CUDA cores; the default `cpu_freq` governor is set to `schedutil`;

c) one x86 Odroid-H2, equipped with an Intel 2.5 Ghz quad-core processor and 4GB of dual-channel DDR4 memory; the default `cpu_freq` governor is set to `powersave` with Intel boost enabled.

We decided to maintain the default CPU governors to simulate a real scenario in which different boards can have proper power settings.

From the networking point of view, the boards have been interconnected with a Gigabit Ethernet switch.

6.2. Kernels and Devices Parameters

We computed and set the value of the parameters defined in Section 5 for tasks and devices of our application in order to use the *LAVA* policy. First of all, we define the sets:

- $\mathbb{T} = \{Motion, Classifier, Tracker, GUI\}$;
- $\mathbb{N} = \{YOLOV3, MobileNetV2\}$;
- $\mathbb{D} = \{Odroid, Jetson, Freescale1, Freescale2\}$;
- $\mathbb{I} = \{VIDEO\}$;

Task	in_t^{VIDEO}	out_t^{SCREEN}	s_t	n_t
Motion	1	0	1	0
Classifier	0	0	1	1
Tracker	0	0	0	0
GUI	0	1	0	0

Table 3: Values of the tasks parameters

Device	$in_d'^{VIDEO}$	$out_d'^{SCREEN}$	s'_d	$c_{d,CPU}^{total}$
Freescale1	1	1	0	400
Freescale2	1	0	0	400
Odroid	0	1	0	400
Jetson	0	0	1	400

Table 4: Values of the devices parameters

- $\mathbb{O} = \{SCREEN\}$.

Then, we define DNN parameters:

- $a_{YOLOV3} = 0.55$;
- $a_{MobileNetV2} = 0.22$;
- $\beta_{YOLOV3,CPU} = 0$ and $\beta_{YOLOV3,ACC} = 0$;
- $\beta_{MobileNetV2,CPU} = 0.9$ and $\beta_{MobileNetV2,ACC} = 0.65$;

Finally, we defined the parameters related to tasks and devices as reported in Tables 3 and 4 respectively.

6.3. Kernels execution characterization

In this first step, initially, we profiled the execution of the single kernels on the various boards. In Table 5, we can see the results of the execution of the *motion kernel*, *tracker kernel* and *GUI kernel*. Due to its low computing resources, the Freescale board, registered the worst execution times in all the kernels. Instead, the Odroid reported the best performances in *tracker* and *GUI* kernels. Finally, Jetson registered the best performance in the execution of *motion kernel*, either by using only the CPU or by taking advantage of GPU acceleration.

For what concerns the *classifier kernel*, the application is compatible with both *YOLO V3* and *MobileNet V2*, whose structures can be found in [31] and [32]. Thus, considering the available resources, we have the possibility to launch the kernel with the most suitable network. In particular, although *YOLO V3* is more precise than *MobileNet V2* (0.55[31] vs 0.22[32] on COCO Dataset as mean Average Precision), we chose the *MobileNet* because is the state-of-the-art for mobile object classification and requires less resources. We evaluated the latency of the two neural networks on the boards obtaining the results shown in Table 5. Due the heterogeneous nature of the Jetson board, we evaluated both the utilization of the CPU and GPU separately. At this regard, the GPU-accelerated execution of both DNN shown the best latencies, recording a 10× reduction for *YOLO V3* and 2× for *MobileNet V2* w.r.t the CPU execution. Finally, due to its limited resources,

	Freescale	Jetson CPU	Jetson GPU	Odroid
Motion	210.18	19.18	10.01	54.04
Tracker	0.47	0.14	N/A	0.13
GUI	24.05	6.75	N/A	5.19
Classifier(1)	38881.97	2932.62	287.38	2022.73
Classifier(2)	2591.36	252.44	105.50	172.13

Table 5: Latency of single kernels executed individually and without any external load on SmokyGrill boards (in ms). Classifier (1) YOLO V3 (2) MobileNet V2.

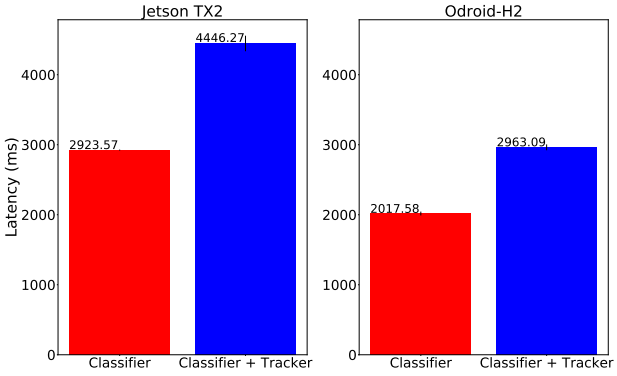

Figure 4: Latency of *classifier kernel* and *tracker kernel* on single SmokyGrill boards

the Freescale board performed poorly with both the DNN while the Odroid board showed similar performances to the ones of the Jetson when not accelerated.

Afterward, we ran some more test, only using the Jetson and Odroid boards, to analyze how the concurrent execution of the *classifier Kernel* with another kernel on the same device would affect the performance. In detail, to see how sensible is the *classifier Kernel* to the presence of another kernel, we ran, as second kernel, the *tracker kernel*,which represents the best-case scenario in terms of resource demanding. As we can see from Figure 4, despite the modest resource usage of the *tracker kernel*, the execution time of the *classifier kernel* increased. In particular, in both boards, the execution times are slower by about 50% with respect to the single kernel execution. This proves the importance of considering the device's current load during task allocation and expected latency computation. For this reason, we measured the latencies of the tasks executed on the various boards at different CPU load levels (0%, 10%, 25%, 50%, 75%, 100%), profiling the $e^{\alpha_d}_{t,d,r}$ parameter introduced in Section 5.1.

6.4. Network and Framework overheads

In our evaluation, we need to know the overheads introduced by the framework and the application's networking. In detail, the size of the exchanged data, except for the classifier kernel, depends on the contribution of three terms. The first term is fixed and it depends on the size of the frame used by the application, while the second term depends on the maximum amount of objects present and tracked on the frame and the

Devices		Tasks		DNNs	
$D0$	$Freescale1$	$T0$	$Motion$	$N0$	$YOLOV3$
$D1$	$Freescale2$	$T1$	$Classifier$	$N1$	$MobilenetV2$
$D2$	$Jetson$	$T2$	$Tracker$		
$D3$	$Odroid$	$T3$	GUI		

Table 6: Values of the devices,tasks and DNNs parameters

	DNN	$D0$	$D1$	$D2$	$D3$	Pipeline Latency(ms)	Classification Latency(ms)
$S0$	$N0$	{}	$\{T0,0\}$	$\{T1,1\}$	$\{T2,0\},\{T3,0\}$	215.68	287.07
$S1$	$N0$	$\{T0,0\}$	{}	$\{T1,0\}$	$\{T2,0\},\{T3,0\}$	228.57	3310.78
$S2$	$N1$	{}	$\{T0,0\}$	N/A	$\{T1,0\},\{T2,0\}$ $\{T3,0\}$	246.86	247.89
$S3$	$N0$	$\{T0,0\},\{T3,0\}$	N/A	$\{T1,1\},\{T2,0\}$	N/A	247.71	215.22
$S4$	$N1$	$\{T0,0\},\{T3,0\}$	N/A	$\{T1,1\},\{T2,0\}$	N/A	247.58	79.55
$S5$	$N1$	$\{T0,0\},\{T2,0\}$ $\{T3,0\}$	$\{T1,0\}$	N/A	N/A	300.82	4413.72
$S6$	$N0$	{}	$\{T0,0\}$	$\{T1,1\}$	$\{T2,0\},\{T3,0\}$	215.44	287.00

Table 7: LAVA Scheduling results

data structures that contain them. Finally, the third term has a fixed size and consists of an integer value representing the number of the objects present and tracked on the frame. The BeeR framework requires this last term to send multiple objects. Known the data structures' size, we ran the application to determine the maximum quantity of data exchanged per frame processed, which is $519,368$ Bytes as input for *classifier kernel* and $1,038,700$ Bytes as the output of *tracker kernel*.

6.5. Policies execution

In the last evaluation step, we executed the distributed application, testing the policy in multiple scenarios with different devices' load levels and availability.

6.5.1. Preliminary considerations

Before commenting on the results, we need to premise some considerations. 1) Having only two DNNs, we used only integer values of the γ weight in the policy evaluation. 2) Regarding devices availability, the choice of those devices configurations aims at covering the following three categories of scenarios, that are characterized by: (a) the availability of multiple low-end and multiple high-end devices with accelerators; (b) the availability of multiple low-end and some high-end devices with possibly accelerators; (c) the availability of only multiple low-end devices. 3) We used some name convention in the results visualization as reported in Table 6. 4) The choice of the connections aims at covering different scenarios where they are only wireless, wired or mixed.

6.5.2. The LAVA policy execution

Among the various scenarios, we selected the results of those reported in Table 8, annotating the type of resources and their initial load level and the presence of accelerators. Moreover, the devices, as well as the input and output capabilities described in Section 5.1, can be connected to the host through a wired connection, $802.11n$ or $802.11ac$ interfaces. Given the fact that the cluster boards do not have wireless capabilities yet, we simulated symmetric wireless connection assuming as maximum bandwidth, for each

	γ	$D0$	$D1$	$D2$	$D3$
$S0$	0	$\{0\%, CPU\}$ $802.11n$	$\{0\%, CPU\}$ $802.11n$	$\{0\%, CPU\}, \{GPU\}$ $802.11ac$	$\{0\%, CPU\}$ $802.11ac$
$S1$	0	$\{75\%, CPU\}$ $802.11ac$	$\{50\%, CPU\}$ $802.11n$	$\{10\%, CPU\}$ $802.11n$	$\{50\%, CPU\}$ $802.11n$
$S2$	1	$\{10\%, CPU\}$ $802.11n$	$\{10\%, CPU\}$ $Wired$	N/A	$\{10\%, CPU\}$ $Wired$
$S3$	0	$\{10\%, CPU\}$ $802.11ac$	N/A	$\{10\%, CPU\}, \{GPU\}$ $Wired$	N/A
$S4$	1	$\{10\%, CPU\}$ $Wired$	N/A	$\{75\%, CPU\}, \{GPU\}$ $802.11ac$	N/A
$S5$	1	$\{10\%, CPU\}$ $802.11ac$	$\{25\%, CPU\}$ $802.11n$	N/A	N/A
$S6$	0	$\{0\%, CPU\}$ $Wired$	$\{0\%, CPU\}$ $Wired$	$\{0\%, CPU\}, \{GPU\}$ $Wired$	$\{0\%, CPU\}$ $Wired$

Table 8: LAVA Scenarios description

type of interface, the one used by Kaewkiriya[33], so a total bandwidth of 145 Mbits for $802.11n$ interface and 870 Mbits for $802.11ac$ interface.

Among the scenarios, in $S0$, $S1$ and $S5$, all the devices are connected through a wireless connection but with different device availability and load levels. In particular, scenarios $S0$ and $S1$ consider scenarios where there are both high-end and low-end devices, while $S5$ considers the situation where only low-end devices are available. The remaining scenarios show heterogeneous configurations for what concerns both processing capabilities and network connections.

Considering the task scheduling results in Table 7, in scenario $S1$, we can see that task $T0$ was placed on device $D0$, although $D1$ has more resources available, thanks to the faster wireless connection. Instead, results related to other scenarios could depend on the difference in terms of computing power between the boards or due to the lack of computing power of the Freescale boards, which makes scheduling preferable on more performing devices, even if those devices are connected wireless, as we can see in $S4$. Finally, $S6$ represents the best possible case where computational heavy tasks can be executed on high-end wired-connected devices. It is worth noticing that the policy respects the task's input/output constraints.

6.5.3. Monolithic vs Distributed comparison

Finally, to show the benefits of the distributed application execution, we measured the frame processing time of the multi-threaded monolithic application executed only on the Freescale board (due to app constraints). Measurements have been taken considering the best scenarios possible, where the board does not have any external load and DNN *MobileNet V2* selected. Figure 5 shows a comparison between the obtained results and some distributed configurations analyzed during policies evaluation. As we can see, distributed execution can reduce the times compared to running all modules in parallel on a single board, unless the boards present have few resources available or can not fulfill task requirements. Otherwise, considering scenarios $S5$ and $S6$ in Table 7 with multiple boards available, our approach can improve the execution time by 53% and 66%, respectively.

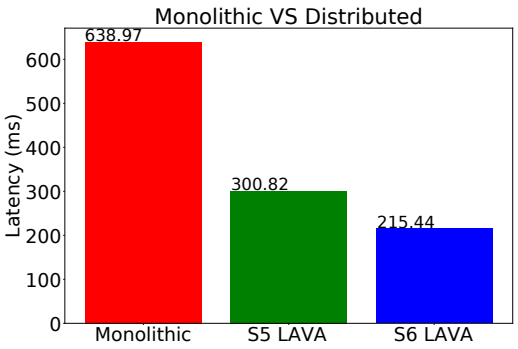

Figure 5: Comparison of pipeline execution times

7. Conclusions and Future Works

In this work, we presented the *BarMan* run-time managed framework, combined with the design and implementation of a video surveillance use-case and a task allocation policy in a Fog computing scenario. In this specific use-case the first step of the policy is related to the application's specific parameters. Thus, in a generic application target, this step could be adapted by choosing the required application's parameters (i.e., other QoS requirements) or avoiding this step if no parameters are needed (i.e., no DNN or other tuning parameters). Afterwards, we profiled the performance of the application's tasks on the various interconnected devices. Then, we investigated the execution of the policy in multiple scenarios. The results showed the ability to correctly find the best task placement considering application's constraints and resources. Finally, we evaluated the differences between the baseline monolithic and the distributed approach, showing in the latter case an improvement in the execution times even by 66%. Since the presence of multiple battery-powered devices characterizes Fog environments, our framework allows future policies to leverage models to estimate the energy-related cost of the task-device mappings, thus enabling power management techniques.

Work is in progress to exploit the programming model to perform reconfiguration of the application at run-time, allowing us to reschedule tasks to meet the specified QoS levels without discontinuing application operation. In this case, it is interesting to investigate the impact of tuning the application's specific parameters according to the status and the constraints of the devices. Finally, we can analyze the policy scalability basing on three scenarios: (1) in case of further hardware types, they can be divided in the three presented categories. For each it is possible to associate a model that can be refined at run-time; (2) in case of more concurrent applications the policy is invoked for each application, basing on start time or customizable priority;(3) the overhead introduced by the policy using a large number of devices could be evaluated through simulation, which could open to future and different evaluation works.

Acknowledgment

This research was partially supported by RECIPE H2020 EU project (grant no. 801137).

References

[1] G. Massari, M. Zanella, W. Fornaciari, Towards distributed mobile computing, in: 2016 Mobile System Technologies Workshop (MST), 2016, pp. 29–35. doi:10.1109/MST.2016.13.

[2] I. D. C. (IDC), Iot growth demands rethink of long-term storage strategies, https://www.idc.com/getdoc.jsp?containerId=prAP46737220, accessed: 01/02/2021.

[3] P. Ravindra, A. Khochare, S. P. Reddy, S. Sharma, P. Varshney, Y. Simmhan, Echo: An adaptive orchestration platform for hybrid dataflows across cloud and edge, in: International Conference on Service-Oriented Computing, Springer, 2017, pp. 395–410.

[4] D. Schäfer, J. Edinger, S. VanSyckel, J. M. Paluska, C. Becker, Tasklets: Overcoming heterogeneity in distributed computing systems, in: 2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW), IEEE, 2016, pp. 156–161.

[5] M. Zanella, G. Massari, W. Fornaciari, Enabling run-time managed distributed mobile computing, in: Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms, PARMA-DITAM '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 39–44. doi:10.1145/3183767.3183778.
URL https://doi.org/10.1145/3183767.3183778

[6] F. Bonomi, R. Milito, J. Zhu, 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, MCC '12, Association for Computing Machinery, New York, NY, USA, 2012, p. 13–16. doi:10.1145/2342509.2342513.
URL https://doi.org/10.1145/2342509.2342513

[7] X. Masip-Bruin, E. Marin-Tordera, A. Jukan, G. Ren, Managing resources continuity from the edge to the cloud: Architecture and performance, Future Generation Computer Systems 79 (2018) 777–785.

[8] M. Zanella, G. Massari, A. Galimberti, W. Fornaciari, Back to the future: Resource management in post-cloud solutions, in: Proceedings of the Workshop on INTelligent Embedded Systems Architectures and Applications, INTESA '18, 2018, p. 33–38. doi:10.1145/3285017.3285028.

[9] M. Mukherjee, e. Al., Survey of fog computing: Fundamental, network applications, and research challenges, IEEE Communications Surveys & Tutorials 20 (3) (2018) 1826–1857.

[10] M. Zanella, G. Massari, W. Fornaciari, Run-time managed mobile application execution, in: 2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC), 2019, pp. 74–77. doi:10.1109/FMEC.2019.8795323.

[11] N. Mainardi, M. Zanella, F. Reghenzani, N. Raspa, C. Brandolese, An unsupervised approach for automotive driver identification, in: Proceedings of the Workshop on INTelligent Embedded Systems Architectures and Applications, 2018, pp. 51–52.

[12] D. Lan, e. Al., A survey on fog programming: Concepts, state-of-the-art, and research challenges, DFSD '19, Association for Computing Machinery, New York, NY, USA, 2019. doi:10.1145/3366613.3368120.
URL https://doi.org/10.1145/3366613.3368120

[13] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, B. Koldehofe, Mobile fog: A programming model for large-scale applications on the internet of things, in: Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing, MCC '13, Association for Computing Machinery, New York, NY, USA, 2013, p. 15–20. doi:10.1145/2491266.2491270.
URL https://doi.org/10.1145/2491266.2491270

[14] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, A. Kitazawa, Fogflow: Easy programming of iot services over cloud and edges for smart cities, IEEE Internet of Things Journal 5 (2).

[15] N. K. Giang, M. Blackstock, R. Lea, V. C. M. Leung, Developing iot applications in the fog: A distributed dataflow approach, in: 2015 5th International Conference on the Internet of Things (IOT), 2015, pp. 155–162. doi:10.1109/IOT.2015.7356560.

[16] K. Intharawijitr, K. Iida, H. Koga, Analysis of fog model considering computing and communication latency in 5g cellular networks, in: 2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops), 2016, pp. 1–4. doi:10.1109/PERCOMW.2016.7457059.

[17] V. B. C. Souza, W. Ramírez, X. Masip-Bruin, E. Marín-Tordera, G. Ren, G. Tashakor, Handling service allocation in combined fog-cloud scenarios, in: 2016 IEEE International Conference on Communications (ICC), 2016, pp. 1–5. doi:10.1109/ICC.2016.7511465.

[18] M. Mukherjee, et Al., Transmission and latency-aware load balancing for fog radio access networks, in: 2018 IEEE Global Communications Conference (GLOBECOM), 2018, pp. 1–6. doi:10.1109/GLOCOM.2018.8647580.

[19] G. Tanganelli, L. Cassano, A. Miele, C. Vallati, A methodology for the design and deployment of distributed cyber–physical systems for smart environments, Future Generation Computer Systems 109 (2020) 420–430. doi:https://doi.org/10.1016/j.future.2020.02.047.
URL https://www.sciencedirect.com/science/article/pii/S0167739X19325245

[20] E. Fraccaroli, F. Stefanni, R. Rizzi, D. Quaglia, F. Fummi, Network synthesis for distributed embedded systems, IEEE Transactions on Computers 67 (9) (2018) 1315–1330. doi:10.1109/TC.2018.2812797.

[21] A. Brogi, S. Forti, : QoS-aware deployment of IoT appli-cations through the fog, IEEE Internet Things J. 4(5) 4 (5) (2017) 1185–1192.

[22] D. Sarddar, S. Barman, P. Sen, R. Pandit, Refinement of resource management in fog computing aspect of qos, International Journal of grid and Distributed Computing 11 (5) (2018) 29–44.

[23] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, P. Leitner, Optimized iot service placement in the fog, Service Oriented Computing and Applications 11 (4) (2017) 427–443. doi:10.1007/s11761-017-0219-8.
URL https://doi.org/10.1007/s11761-017-0219-8

[24] P. Bellasi, G. Massari, W. Fornaciari, Effective Runtime Resource Management Using Linux Control Groups with the BarbequeRTRM Framework, ACM Trans. Embed. Comput. Syst. 14 (2) (2015) 39:1–39:17. doi:10.1145/2658990.
URL http://doi.acm.org/10.1145/2658990

[25] G. Agosta, W. Fornaciari, G. Massari, A. Pupykina, F. Reghenzani, M. Zanella, Managing heterogeneous resources in hpc systems, in: Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms, PARMA-DITAM '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 7–12. doi:10.1145/3183767.3183769.
URL https://doi.org/10.1145/3183767.3183769

[26] G. Agosta, W. Fornaciari, D. Atienza, R. Canal, A. Cilardo, J. Flich, C. Hernandez Luz, M. Kulczewski, G. Massari, R. Tornero Gavila, M. Zapater Sancho, Challenges in deeply heterogeneous high performance systems, in: 2019 22nd Euromicro Conference on Digital System Design (DSD), 2019, pp. 428–435. doi:10.1109/DSD.2019.00068.

[27] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, R. Buyya, ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments, Software: Practice and Experience 47 (9) (2017) 1275–1296. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2509, doi:10.1002/spe.2509.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2509

[28] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, C. L. Zitnick, Microsoft COCO: common objects in context, CoRR abs/1405.0312.
URL http://arxiv.org/abs/1405.0312

[29] C. Jin, J. Choi, W. Kang, S. Yun, Wi-fi direct data transmission for wireless medical devices, in: The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014), 2014, pp. 1–2. doi:10.1109/ISCE.2014.6884461.

[30] S. Oh, A. Hoogs, A. Perera, N. Cuntoor, C. Chen, J. T. Lee, S. Mukherjee, J. K. Aggarwal, et Al, A large-scale benchmark dataset for event recognition in surveillance video, in: CVPR 2011, 2011. doi:10.1109/CVPR.2011.5995586.

[31] J. Redmon, A. Farhadi, Yolov3: An incremental improvement, CoRR abs/1804.02767.

[32] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition.

[33] T. Kaewkiriya, Performance comparison of wi-fi ieee 802.11ac and wi-fi ieee 802.11n, in: 2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA), 2017. doi:10.1109/CSCITA.2017.8066560.