Computation Offloading on 5G Core Network for a Highly Delay Sensitive Real Time App

Pierre Domachowski, Haga Randrianaly, Marie-Jose Montpetit, Noel Crespi

Computation Offloading on 5G Core Network for a Highly Delay Sensitive Real Time App

DOMACHOWSKI Pierre
Msc DANI
Télécom SudParis
Evry, France

RANDRIANALY Haga
Msc DANI
Télécom SudParis
Evry, France

MONTPETIT Marie-José
Adjunct Professor
Ecole de Technologie Supérieure
Montréal, Canada

CRESPI Noel
Msc DANI Coordinator
Télécom SudParis
Evry, France

***Abstract*—In a landscape where the Internet of Things (IoT) is rising, its application in diverse scenarios, including emergency situations, becomes crucial. Considering the necessity for precise, quick and adaptive actions when it comes to emergencies, IoT-collected data and data-driven tools occur to be relevant solutions to save lives.Building from the description of a first-aid tool based on augmented reality (AR) and computer vision, this paper explores a possible solution to extend its geographic mobility and reducing latency. We propose to combine a task division and allocation strategy with the use of edge computing leveraging Edge and 5G Network, in order to address the high latency issue when it comes to cloud computing.**

***Index Terms*—Low Latency, Edge Computing, Augmented Reality, 5G**

I. Introduction and Description

A. Introduction

As illustrated in recent demonstration video [1], the potential of connected defibrillators combined with smart glasses in public spaces to save lives is undeniable.

These glasses offer a complete guidance to individuals on how to rescue a potential victim in need of assistance enabling any person to become a far better rescuer.

Nevertheless, implementing such solutions necessitates high computational power and minimal latency, as we are told in [1], maintaining an end-to-end latency of no more than 20 milliseconds is crucial to prevent discomfort and motion sickness in users.

While cloud computing is commonly used, it cannot provide significantly low latency mainly because of the distance between the user equipment (UE) and the computing servers.

Moreover, emergencies can occur everywhere including areas with limited bandwidth and considerably far from traditional cloud computing resources. Heavy resources consuming applications designed to deal with emergencies, such as the one we study in this article accumulate two main issues:

- they need a very high responsiveness and minimal latency,
- they need to cover the largest possible zone to save a maximum number of lives

[1]LinkedIn post with a video description of the first-aid smart glasses: https://www.linkedin.com/feed/update/urn:li:activity:7092724445252898817/?utm_source=share&utm_medium=mem

Therefore, this paper aims to explore one approach to compute closer to the location of the emergency using Edge resources and wireless communications to offload the computation part of the application closer from the UE. This approach capitalizes on the reduced latency, efficient video transmission, and enhanced monitoring capabilities that optimal utilization of the 5G Core (5GC) can offer. We leverage these functionalities by addressing two key issues:

- Network-induced latency
- Computation time

To tackle these challenges, we propose to offload the computation tasks to the equipments within the 5G core network (5GC).

B. Description

Our emergency involves a rescuer and a victim. The rescuer uses our Emergency kit, also called UE which is composed of an on-battery Box with an antenna, wired to the AR display device, also called glasses. These glasses also hold a camera, recording what the rescuer sees.

The central box contains a local storage and a modem that converts the recorded video into radio signal. The inputs of the application are the frames of the recorded video content given by the camera on the glasses.

As illustrated in Fig.1 the computation steps of the application's execution consist in live video object detection tasks and are done in the network thanks to the computational power of Edge resources.

More precisely, On each frame recorded by the camera, many elements such as electrodes and sternum are to be recognized and numerically highlighted on the glasses in a precise order to clearly explain to the rescuer how to proceed.

Finally, the Edge computation servers send back the computational results, also called annotations, to the Emergency kit to display them on AR glasses. The practical considerations on how to display the annotations on the glasses is out of the scope of our article and is already addressed in works such as [6].

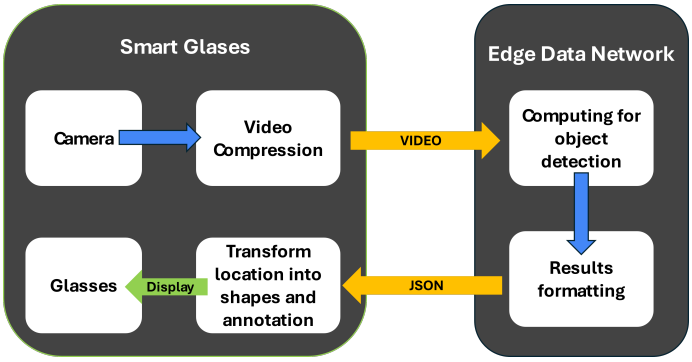

Fig. 1. General Workflow of the Application

C. Definitions

- Application Client (AC): application resident in the UE performing the classical client function of a three tier application.
- Edge Enabler Client (EEC): provides supporting functions needed to enable AC(s) to communicate with the Edge servers.
- Edge Application Server (EAS): application server resident in the EDN (Edge Data network), performing the server computation functions.
- Edge Configuration Server (ECS): provides configuration functions needed for the EEC to connect with EES(s).
- Edge Enabler Server (EES): provides supporting functions needed to link EAS(s) and AC through EEC.

II. Architecture : Reducing Network Latency

As mentioned in [2], our AR application can benefit significantly from the reduced latency, improved video transmission, and enhanced monitoring capabilities provided by optimal utilization of 5GC.

In this section, we describe how we will leverage edge architecture in our application. Building on the equipment and connection definitions in [3], we detail the entire process that enables the AC of the defined emergency kit to connect with the appropriate EAS(s). Communications between edge servers and clients will be facilitated through various APIs and reference points, organized into edges ranging from 1 to 9.

A. Preliminary step

The first step is the setup of the EEC of the defibrillator. We initialize the EEC, such that its look for and find the addresses of the reachable ECSs before the first use of the defibrillator. This part enables the defibrillator to connect to the EASs faster when it will be used for the first time.

B. Connection to the EASs

As illustrated in Fig 2. the first step for the UE's EEC is to connect to the most suitable EES. To do so, the EEC authenticates itself to the chosen ECS and sends information. Based on this information, the ECS will choose the best EES and send back its address to the EEC. The second step for the EEC is to send a service provisioning request to the chosen EES. This will allow our AC to connect to the EASs for a certain time. Then, the EEC sends an EAS discovery request to the EES. The answer to this request contains a list of the available EASs and KPIs such as the available resources (RAM, computing power…). Now that information to enable the data traffic between the AC and the EASs are well known, we perform a selection of the best EASs, this part is detailed in part III. As the volume of data exchanged is very high in our case, the available bandwidth must be very high between the AC and the chosen EAS(s). That is why, the session establishment includes with a specific Quality of Service (QoS) in terms of Bandwidth.

C. Final step

At the end of the process, when the glasses will be turned off, the session will shut down and the EEC will unregister from the ECS, allowing the selected EAS(s) to release the allocated resources.

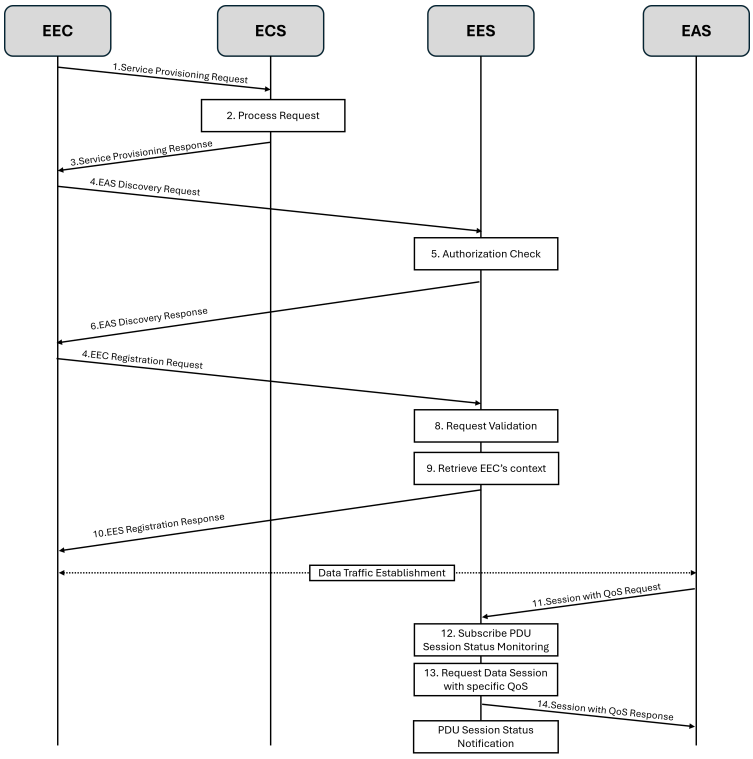

Fig. 2. Connection to the EASs process chart

III. Reducing Computing Time

A. Object detection model

To get relevant annotations displayed on the user's glasses, we must use an object detection model which takes images as an input and sends back annotations as an output.Therefore, one way to reduce computing time is reducing the inference of the object detection model.

In [4], Haogang et Al. did a comparative analysis of YOLO about many metrics such as model accuracy, frames-per-second (FPS), memory usage, CPU usage, and energy consumption depending on hardware characteristics, inputted video frame size, and the SxS grid in which each frame is divided (written S value). According to their results, the version of YOLO is the "main factor affecting performances." In contrast "the influence of the size of input data on inference performance is quite slight." Considering those insights, we choose to work on model choice only among the previously cited metrics to reduce latency.

Especially, YOLOv8 is likely to be the most relevant model in our use case to reduce computation time, as it can work up to 70 FPS according to [5] which is above the minimum of 60 fps usually considered for AR applications, and the 41fps reached by former YOLO versions mentioned in [4].

Furthermore, [6] provides an example of YOLO usage for object detection using Microsoft HoloLens. The best performances are reached with YOLOv3, with a precision of 96.27% under real-time constraints.

YOLO has also been used a lot in medical imaging and in particular in Personal Protective Equipments as mentioned in [7] despite some limitations. In fact it is explained in [7] that YOLO requires a large dataset to be fine-tuned on in order to fit our needs in terms of object detection. We need a very good precision but on a very small number of objects. YOLO's precision is also affected by object scale and occulted objects. These situations may often occur in emergency situations such as those our YOLO's implementation may deal with.

Eventually, even if YOLO seems to fit our first constraints in terms of latency and precision, we may face different issues while implementing it for our use case. These problems may result into a substantial work on the model's fine-tuning and general implementation which is out of scope for this article.

B. Task division

When the data traffic is ready to be established, we must select the best EASs considering their available resources in order to run the different tasks of the defined object detection model, to reduce the computation time of our application. The first step is to decompose our application in tasks that have to be executed either sequentially or in parallel. Our application can be divided into many object detection ones, for example, detection of the victim's sternum, detection of the electrodes… And in this example, these tasks must be done after the victim's shirt is torn apart. Shirt detection and sternum detection are then tasks to be executed sequentially. By applying the same logic to all the tasks of the application, we can build a planning of our application's execution such as the one in Fig.3. where horizontal tasks (T1 and T2A) must be done sequentially, one after the other, and vertical tasks (T2A and T2B) can be done in parallel.

C. Task allocation

When the EEC receives the list of all available EAS(s), their routing information and their available resources, it determines the optimal task allocation to minimize overall latency based on the previously constructed schedule. The overall latency is written:

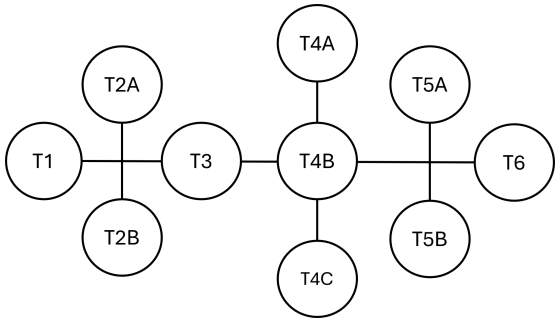

Fig. 3. Example of application's task scheduling

$$L = \max(L_i) \tag{1}$$

where L_i is the latency between the glasses and the EAS_i, Particularly:

$$L_i = L_{Ni} + L_{Ci} \tag{2}$$

where L_{Ci} is the latency induced by the computation time and L_{Ni} is the latency induced by the network. We have two options for sending data traffic across the network: either send the entire data set or send compressed video. The choice between these methods will depend on the available bandwidth (*Bw*), selecting the one that induces the least latency.

Consequently,

$$L_{Ni} = min(L_{comp}, L_{decomp}) \tag{3}$$

where

$$L_{comp} = L_{compression} + L_{dist} + L_{decompression} + \frac{flow_{comp}}{Bw} \tag{4}$$

and

$$L_{decomp} = L_{dist} + \frac{flow_{decomp}}{Bw} \tag{5}$$

And,

$$L_{Ci} = max(t_{Cj}), \tag{6}$$

where t_{Cj} is the computation time of all the tasks executed in parallel on the EAS_i at a given time.

To find the best method, the EEC will give to the AC the routing information of the EAS(s) to evaluate the available bandwidth between them by setting a UDP connection and will evaluate the latency induced by the distance between the AC and the EAS by pinging it. We assume that $L_{compression}$ and $L_{decompression}$ are known from previous tests. Next, we compute the uncompressed data flow in bytes using the image definitions and frame rate. We also assume the compression rate is known from prior tests.

Finally, to select the most relevant EAS for each task, we propose to follow the steps outlined below, which we also implemented in Python[2].

[2]Code available in this repository: `https://github.com/holygramp/CIOT-2024/blob/main/Task_allocation.py`

Step 1: Sort each EAS

- Pinging and setting up a UDP connection between all reachable EASs and the AC, in order to evaluate the available bandwidth and the latency induced by the network between the AC and all the EASs.
- sort each EAS in ascending order depending on the ping latency between AC and each EAS

Step 2: Optimize task execution

- Allocating to the closest EAS as many tasks as possible considering its available resources.
- Considering the resources needed to perform task A and B:
  - If A and B have to be executed sequentially, the resources required are the sum of the resources for task A and task B.
  - If A and B have to be executed in parallel, the resources required are the maximum of the resources for task A and task B.

Step 3: Continue and Repeat steps

If the closest EAS cannot perform all the application computation, the residual tasks are allocated to the next closest EASs. Then, this process continue until all the computation of the application can be performed.

Other algorithms for task allocation may be more general and optimized, such as the one proposed in [8] but in our case, we deal with a very specific application and not a general one. Furthermore, some of the hypothesis they make are not verified in our case such as the independence of the tasks.

Therefore, we propose the algorithm detailed above that better suits our requirements.

D. Default functioning mode

We also propose a backup solution in order to guarantee a minimum functioning state of our defibrillator. If in a specific emergency, despite all the work that we have done to reduce the overall latency of the system, the latency becomes too high, we will be able to detect it and switch our glasses into a default minimum functioning mode with the functionalities that don't require real-time treatment. In other terms, the one that require fewer resources and no connection to external computing servers such as massage rhythm or instruction lists for example.

As mentioned before and in [1] an overall latency of 20 milliseconds is required, therefore, our glasses will be continuously computing the actual latency in order to raise a warning when it exceeds 20 milliseconds for too much time.

When the warning is raised, the glasses will stop the display of the bounding boxes and other annotations that require a low latency to be wisely displayed to enter a mode where only the basic and non real-time annotations will be displayed, such as instructions for example.

This mode shall remain until the overall latency goes under the 20 milliseconds threshold for a significant time.

IV. Discussion and Future Work

The next step that we will work on will be to have a simulation of our application. We will be working with simu5G, an ETSI compliant emulation and simulation framework based on OMNeT++ for MEC applications. We will then compare our simulation with results obtained by using cloud computing in terms of overall latency. We are currently working on this simulation we expect to get these results by the beginning of the conference, in October. Another very important work that has to be done in order to confirm the validity of our approach will be to validate the performances of YOLO considering our tasks. In fact, if YOLO is not precise enough or too slow computationally speaking, we will need to look for or build a better model, specialized for our application. And finally, another very important task to validate our approach will be to study the augmented reality part and how to tackle 2 issues:

- how to record the input video
- how to display the annotations on the glasses

These are two practical issues that need to be tackled in order to ensure that our application will be usable in reality.

References

[1] M. Torres Vega, et al. "Immersive Interconnected Virtual and Augmented Reality: A 5G and IoT Pespective", in Journal of Network and Systems Management, 1963, October 2020,pp 1–2.

[2] I. Kunze, K. Wehrle, D. Trossen and M.J. Montpetit " Use Cases for In-Network Computing draft-irtf-coinrg-use-cases-00".

[3] ETSI TS 123.558 version 17.3.0 release 17: "5G System Enhancements for Edge Computing".

[4] Benchmark Analysis of YOLO Performance on Edge Intelligence Devices, Haogang Feng, Gaoze Mu, Shida Zhong, Peichang Zhang 1and Tao Yuan, MDPI

[5] Houda Orchi, Mohamed Sadik, Mohammed Khaldoun, Essaid Sabir: "Real-Time Detection of Crop Leaf Diseases Using Enhanced YOLOv8 algorithm"

[6] Haythem Bahri; David Krčmařík; Jan Kočí Accurate Object Detection System on HoloLens Using YOLO Algorithm

[7] A Comprehensive Systematic Review of YOLO for Medical Object Detection (2018 to 2023) Mohammed Gamal Ragab; Said Jadid Abdulkadir; Amgad Muneer; Alawi Alqushaibi; Ebrahim Hamid Sumiea; Rizwan Qureshi; Safwan Mahmood Al-Selwi ; Hitham Alhussian

[8] Anamaria-Raluca Oncioiu1, Florin Pop1, and Christian Esposito Asymptotic Load Balancing Algorithm for Many Task Scheduling