Multipath Computation Offloading for Mobile Augmented Reality

Tristan BRAUD*, Pengyuan ZHOU†, Jussi KANGASHARJU†, and Pan HUI*†

*The Hong Kong University of Science and Technology - Hong Kong
†University of Helsinki - Finland
[1]Email: braudt@ust.hk, pengyuan.zhou@helsinki.fi, jussi.kangasharju@helsinki.fi, panhui@cse.ust.hk

***Abstract*—Mobile Augmented Reality (MAR) applications employ computationally demanding vision algorithms on resource-limited devices. In parallel, communication networks are becoming more ubiquitous. Offloading to distant servers can thus overcome the device limitations at the cost of network delays. Multipath networking has been proposed to overcome network limitations but it is not easily adaptable to edge computing due to the server proximity and networking differences. In this article, we extend the current mobile edge offloading models and present a model for multi-server device-to-device, edge, and cloud offloading. We then introduce a new task allocation algorithm exploiting this model for MAR offloading. Finally, we evaluate the allocation algorithm against naive multipath scheduling and single path models through both a real-life experiment and extensive simulations. In case of sub-optimal network conditions, our model allows reducing the latency compared to single-path offloading, and significantly decreases packet loss compared to random task allocation. We also display the impact of the variation of WiFi parameters on task completion. We finally demonstrate the robustness of our system in case of network instability. With only 70% WiFi availability, our system keeps the excess latency below 9 ms. We finally evaluate the capabilities of the upcoming 5G and 802.11ax.**

I. Introduction

Mobile Augmented Reality (MAR) may be the most computationally-intensive multimedia application, with strict real-time constraints. A typical MAR application processes large amounts of data, such as video flows, to display a virtual layer on top of the physical world. These operations usually run on mobile devices such as smartphones or smartglasses that can only execute basic operations. The increase in the performance and ubiquity of networks allows remote devices and servers to execute larger portions of code. However, the latency constraints of MAR applications (sometimes less than 20 ms [1]) are such that both the available bandwidth and computing power on a single link is not sufficient for in-time processing. Furthermore, wireless links characteristics vary extremely in mobility conditions and may cause severe service degradation or interruptions [2].

To get more insights about current wireless networks' situation, we perform a simple experiment. Table I presents the round-trip times (RTT) measured between a smartphone (LG Nexus 5X) and several potential offloading devices: another smartphone, connected using WiFi Direct (1 m distance), an Alibaba Cloud virtual machine through WiFi[1] via *eduroam*[2] and LTE, a Google Cloud virtual machine through WiFi and LTE, as well as the first reachable server on each link to emulate an Edge server. We average our measurements over 100 ICMP packets. The latency increases dramatically with the distance between the client and the server. D2D shows RTTs as low as 3.5 ms. The WiFi access point (AP) several meters away adds 0.2 ms, and the Alibaba cloud server 2 ms. As the Google Cloud server is located about 1,000 km away to emulate a more distant cloud provider, latency is multiplied by eight compared to the local Alibaba server. LTE also adds noticeable latency relatively to WiFi: 16 ms for an Edge server and 10 to 19 ms for a Cloud server.

TABLE I: Average network round-trip time measured for different offloading mechanisms.

D2D	Edge	Edge	Alibaba	Alibaba	Google	Google
WiFi D	WiFi	LTE	WiFi	LTE	WiFi	LTE
3.5 ms	3.7 ms	19.9 ms	5.5 ms	24.9 ms	42.2 ms	52.4 ms

In these conditions, maximizing in-time task completion involves striking an intricate compromise between transmission-related delays and computation time. To provide such performance, MAR applications should not only dynamically offload their computations in parallel over the set of available devices but also exploit the multiple available links to minimize transmission delays. For instance, a pair of smartglasses may connect to a companion smartphone through WiFi Direct, several Edge servers through WiFi and LTE, and even one or several servers located in the cloud for heavier computations.

In this paper, we develop a scheduling algorithm for task assignment over multiple links to an heterogeneous set of servers composed of D2D companion devices, edge servers and cloud servers connected through both WiFi and LTE. We design a multipath, multiple server offloading mechanism to provide more ressources for in-time MAR tasks completion. Providing multiple resources in parallel also enables robust fallback in case of link degradation or failure for uninterrupted MAR service. We develop a latency model taking into consideration the various elements of the system and

[1]Unless specified otherwise, WiFi refers to the 802.11ac standard.
[2]https://www.eduroam.org/

perform optimizations to aggregate tasks based on their final destination. We finally evaluate this algorithm through both a real-life implementation and extensive simulations. After comparing our solution to single-path and naive multi-path task allocation, we analyze the impact of the access link and the computing power of servers on the task distribution and the in-time task completion. Finally, we evaluate the robustness of our model to the instabilities, and expand our work to the upcoming 5G and 802.11ax.

Our contributions can be summarized as follows:

1) A model of latency for multipath task offloading.
2) A scheduling algorithm to allocate tasks over multiple wireless links to D2D, edge, and cloud servers.
3) A real-life implementation of a multipath, multi-server mapping application that reduces latency by 10% compared to the state-of-the-art.
4) An extensive set of simulations to characterize the system. Our algorithm can withstand high bandwidth drops and high latency variations without impacting tasks completion. In sub-optimal scenarios such as intermittent connections, excess latences are kept below 9 ms.

II. Related Works

Computation offloading was one of the main motivation for computer networks. In a memo considered as the first documented evocation of computer networks (1963), J.C.R Licklider justifies the need for device interconnection to enable access to distant computing resources [3]. In recent years, the explosion of the mobile device market shed new light on these problems. Many cyber-foraging solutions for mobile applications were developed, whether in the cloud [4], [5], the edge of the network [6], or exploiting D2D communication [7].

Offloading frameworks enhance the capabilities of hardware-limited mobile devices. These frameworks focus on the data partitioning problem as well as its implementation in mobile devices. MAUI [8] focuses on the energy consumption of mobile devices to perform offloading. The authors provide the data partitioning through a simple annotation system. CloneCloud [9] modifies the application layer virtual machine to automatically offload tasks to a remote server with no intervention on the application. ThinkAir [10] distributes offloaded functions in parallel among multiple virtual machine images in the cloud. Cuckoo [11] is another generic framework aiming at offloading computation with minimal intervention from the application developer. Finally, Breitbach et al. [12] decouple task scheduling from the data, to accelerate edge computing. Several other studies were directly focused on AR-specific offloading. Back in 2003, Wagner et al proposed to offload AR tasks to a distant machine [13]. In [14] Shi et al provide guidelines for MAR applications on wearable devices. Finally, Overlay [15] exploits cloud offloading for AR applications. However, these works focus on pure cloud or edge computing, with eventual distribution over several servers positioned at the same level in the network. Moreover, they neglect LTE links due to their high latency and variance. In this paper, we argue that offloading to servers located at different levels of the network has a significant impact on task completion, and that LTE can be used as a fallback link in certain conditions.

In parallel to these generic and AR-specific cloud offloading frameworks, new applications were developed, exploiting either D2D or edge computing. D2D communication is defined as the direct communication between two mobile devices [16], [17]. D2D communication has been used for offloading over Bluetooth [18], WiFi Direct [19], or even NFC [20]. Mobile edge computing is considered as an extension of the current cloud model, in which the servers are brought as close as possible to the access point to reduce network-induced latency and avoid congestion in the core network. However, this causes problems as containers are usually large in size and cumbersome to deploy [21]. This new paradigm attracted a lot of attention, not only from academia [22], [23], [24], but also the industry [25], [26]. We integrate both paradigms in our model in association with cloud computing, as their distinct characteristics can prove essential for enhancing the experience of offloading applications.

More recently, studies started to focus on the networking aspects of computation offloading, whether from an energy perspective [27], to reduce data transmission [28], or optimize mobility [29]. The networking challenges of AR have been evoked in several articles. [30] proposes to combine AR offloading and Information-Centric Networks to optimize data transmission. [31] focuses on the application layer constraints, while [32] insists on transport layer optimizations. EARVE considers the case of high mobility users such as vehicules [33]. Finally, Cicconetti et al. [34] propose to distribute edge computing over multiple servers. However, they only consider a single access link. In this paper, we push forward these studies by analyzing multipath offloading for MAR among servers located at various levels of the network. We acknowledge the variety of access links and server hardware to propose a new task allocation model.

III. Modeling an AR application

In this section, we decompose a typical AR application (such as Vuforia [35]) into a set of tasks and propose the deadlines and computational/networking costs for each one of them. We use the application model proposed by Verbelen et al [36], to which we add a new component – the *Feature Extractor* – to further enhance tasks parallelization. Let us consider a context-aware AR web browser [37]. Such an application analyzes the surroundings of the user and combines geographic location and temporal information with computation-heavy image processing algorithms to display the websites of specific locations in AR. Image processing is the most substantial computation task, not only due to the necessary raw computing power but also because of the frequency of the process. We design this application as presented Figure 1. This process starts with a video source (here, a camera) capturing pictures. These pictures then go through a *Feature Extractor* that isolates the main points of interest for future processing. These features are then fed

TABLE II: Tasks parameters.

Module	Renderer	Feature extractor	Tracker	Mapper	Object rec.
Task	$T_{r,k}$	T_e	T_t	T_m	T_o
Input	Frame, Metadata	Frame	Feature points (FP)	FP, World Model	FP
Output	Rendered Objects	Feature Points	Position	World Model	Object Prop.
L_{data}	high	variable	medium	medium	medium
L_{res}	high	variable	low	medium	low
Deadline	$\tau d, min$	variable	$2\tau_{d,min}$	$3\tau_{d,min}$	$4\tau_{d,min}$
X	low	variable	medium	high	high

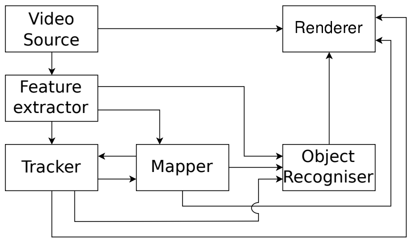


Fig. 1: Main components of a typical MAR application.

into three interdependent components: the *Mapper* creates a dynamic map of the 3D world, the *Object Recognizer* performs fine-grained analysis of the picture to locate specific objects, and the *Tracker* tracks the objects in subsequent frames. The result of these three modules is then fed with the camera images into the *Renderer* which is in charge of combining them to display the virtual layer on top of the real world.

We break down the control flow of the application as a set of N tasks $\{T_n\}$. Each task $T_n(t)$ can be characterized by its data size $L_{data,n}$, the computation results size $L_{res,n}$, the number of CPU cycles required to run the task X_n, and the deadline $\tau_{d,n}$, so that the total execution time $\tau(T(t))$ is inferior to $\tau_{d,n}$. The task parameters for each component are presented Table II. The *Video Source* gets the video frames from the camera. This operation requires access to the hardware and can only run on the device. Nowadays, most cameras operate between 30 to 60 Frames per second; the minimum deadline is thus: $\tau_{d,min} = \frac{1}{FPS}$. At the other extremity, the *Renderer* aggregates the results from the computation modules and overlays them on top of the video frames. This operation has to be performed every frame and as such is generally not offloaded to a distant machine. However, in the case of restricted hardware or heavy rendering, offloading through a low latency network may be the only solution to meet the deadline. We consider a set of k objects to render in parallel, with deadline $< \tau_{d,min}$. The *Feature Extractor* extracts feature points from the camera frames. This component can have different resolutions and deadlines, depending on the component using the feature points as input. The Tracker requires a lower resolution than the Mapper or the Object Recognizer while having a shorter deadline. The *Tracker* tracks objects on the frame. This module should process 15 to 20 FPS

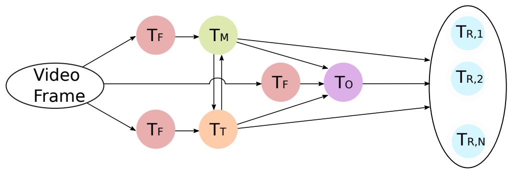


Fig. 2: Task dependency graph.

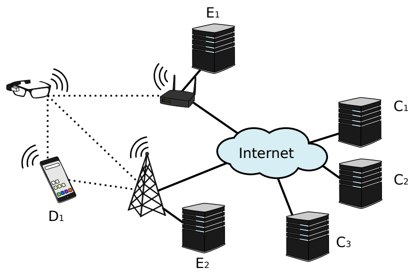


Fig. 3: Environment model: a pair of smartglasses is connected to several computing units located at different extremities of the network: device to device (D_k), edge (E_k) and cloud (C_k).

for seamless operation, so the overall deadline for feature extraction and position estimation should be no higher than $2\tau_{d,min}$ [36]. The *Mapper* creates a model of the world out of the feature points extracted by the *Feature Extractor*. It identifies new feature points and estimates their position in space. The Mapper requires a higher image resolution than the Tracker. However, it is less delay-constrained and can be called every few frames. We estimate a deadline between $2\tau_{d,min}$ and $4\tau_{d,min}$ for feature extraction and world modeling. Finally, the *Object Recognizer* identifies objects out of the feature points and returns their position. Similarly to the Mapper, the object recognizer does not require to be run in real-time and can be called every $4\tau_{d,min}$ to $8\tau_{d,min}$.

We extract the dependency graph in Figure 2. All tasks are interdependent; however, we can split the dependency graph for parallel processing [36]. As the world model and the object list do not require an update every frame, tasks $T_{e1}+T_r$, $T_{e2}+T_m$ and $T_{e3}+T_o$ can be processed in parallel. The only dependency is the combination of feature extraction with another task. By keeping track of the previous instance's results, this model avoids passing on the excess latency.

IV. System Model

We consider the scenario presented in Figure 3. A mobile device (here smartglasses) executes a MAR application. Due to the low processing power of smartglasses, computation is offloaded to companion devices, edge servers and cloud servers connected through WiFi Direct, WiFi and LTE.

A. Available resources

At a given time t, the client has a set of N tasks $\{T_n(t)\}$ to run, as defined Section III. The client connects to the servers through a set of access links $\{L_i\}$. We consider that at any time t, only a subset of size I is available to the user, depending on the actual availability of networks. Those links are characterized by their delay $\tau_i(t)$ and bandwidth $B_i(t)$, variable over time. We consider the mobile network to be down when LTE is not available, as UMTS can not provide the minimum throughput and latency requirements for MAR. The smartglasses connect to a set of offloading devices through these networks. The set of devices includes J directly connected companion devices $\{D_j\}$, K edge servers $\{E_k\}$ at the WiFi or LTE access point, and L cloud servers $\{C_l\}$. They are characterized by their computing power $CPU_{\{j,k,l\}(t)}$. In the case of cloud servers, we consider the connection to be established through an aggregate link $L_{aggr,i}$ composed of one of the access links belonging to $\{L_i\}$ and a backbone network with additional latency $\tau_{backbone}$. We consider the access link as the bottleneck of the network. The resulting link $L_{aggr,i}$ is characterized by its latency $\tau_{aggr,i}(t) = \tau_i(t) + \tau_{backbone}(t)$ and bandwidth $B_{aggr,i}(t) = B_i(t)$.

B. Resource allocation

The execution time of task $T_k(t)$ is a function of the original transmission time $\tau_{tr}(t)$, the computation time on the server $\tau_{comp}(t)$ and τ_{res} the transmission time of the result:

$$\tau(T_n(t)) = \tau_{tr,n}(t) + \tau_{comp,n}(t+\tau_{tr}) + \tau_{res,n}(t+\tau_{tr}+\tau_{comp}) \tag{1}$$

with:

$$\tau_{tr,n}(t) = \begin{cases} \tau_i(t) + \dfrac{L_{data,n}}{B_i(t)} & Edge\ or\ D2D \\ \tau_i(t) + \tau_{backbone}(i) + \dfrac{L_{data,n}}{B_i(t)} & Cloud \end{cases} \tag{2}$$

$$\tau_{comp,n}(t) = \frac{X_n}{CPU_{j,k,l}(t)} \tag{3}$$

$$\tau_{res,n}(t) = \begin{cases} \tau_i(t) + \dfrac{L_{res,n}}{B_i(t)} & Edge\ or\ D2D \\ \tau_i(t) + \tau_{backbone}(i) + \dfrac{L_{res,n}}{B_i(t)} & Cloud \end{cases} \tag{4}$$

We consider that the smartglasses can estimate the channel conditions, as well as the available resources available on the servers at all times. Offloading $\{T_n(t)\}$ to a server comes down to assigning resources so that:

$$\tau(T_n(t)) < \tau_{d,n}\ \forall n \tag{5}$$

$$\min \Psi = \sum_{n \in N} \tau(T_n(t)) \tag{6}$$

C. Multipath Cloud Offloading

Cloud servers are positioned further in the network than edge servers and companion devices. As a result, transmitting over WiFi or LTE results in a lower difference in overall latency.Considering a set of access links $\{L_i\}$ connected to a backbone network to a cloud server, multipath transmission reduces overall network latency when:

$$\sum_{i \in I} \tau_i(t) + \frac{k_i}{B_i(t)} < \min_i \left(\tau_i(t) + \frac{L}{B_i(t)} \right) \tag{7}$$

Here, k_i represents the amount of data transmitted over link L_i. In the most common case, only two links are present: LTE and WiFi. The system becomes:

$$\begin{aligned} &\tau_{WiFi}(t) + \frac{K}{B_{WiFi}(t)} + \tau_{LTE}(t) + \frac{L-K}{B_{LTE}(t)} \\ &< \min\left(\tau_{WiFi}(t) + \frac{K}{B_{WiFi}(t)}, \tau_{LTE}(t) + \frac{L-K}{B_{LTE}(t)} \right) \end{aligned} \tag{8}$$

with optimal K found when $\tau_{WiFi}(t) + \dfrac{K}{B_{WiFi}(t)} = \tau_{LTE}(t) + \dfrac{L-K}{B_{LTE}(t)}$:

$$K = \left(\tau_{WiFi} - \tau_{LTE} - \frac{L}{B_{LTE}} \right) \cdot \frac{B_{WiFi} B_{LTE}}{B_{LTE} - B_{WiFi}} \tag{9}$$

These principles can also be applied to edge servers, although edge servers display such low latency that the interconnection between the two networks may introduce a larger relative delay, reducing the impact of multipath transmission.

D. Mobility

Many MAR applications are used in mobility scenarios. The user will thus experience regular disconnections and long handovers due to his mobility. Tasks offloaded to the edge will have to be transmitted through the backbone network to be recovered through another link. We envision three scenarios: (1) If completion prevails over latency, accept the additional delay and transmit the results through the backbone network. (2) In the case of latency-sensitive tasks, discard the task as soon as the device leaves the access point (3) Flag critical tasks at the application level to offload them to a reliable resource (cloud server, companion device) in priority.

E. Data consistency

In distributed offloading, computation results may be transmitted to the rest of the system for further computation. The synchronization between a server i and the rest of the system adds a delay $\tau_{sync,i}$, the time to propagate data to the system.

$$\tau_{sync,i} = \max_j (\tau_{tr,i \to j}) \tag{10}$$

F. Sequential processing

Several tasks may be assigned to the same link or the same server. We process tasks sequentially: only one task can be transmitted or processed at a given time on the same resource. Sequential processing permits fine-grained resource allocation as it allows prioritizing tasks according to their deadlines. Moreover, assigning a task to a given resource does not modify the status of already assigned tasks. Therefore:

$$\tau_{comp,n}(t) = \tau_{sched,n} + \frac{X_n}{CPU_{j,k,l}(t)} \tag{11}$$

$$\tau_{tr,n}(t) = \tau_{wait,n} + \tau_i(t) + \frac{L_{data,n}}{B_i(t)} \tag{12}$$

$\tau_{sched,n}$ being the time to wait for the task to be scheduled on the server, and $\tau_{wait,n}$ the delay before transmission on the link. If the task $T_n(t)$ has to be executed sequentially on the server, transmission of the task can be delayed by an additional $\tau_{wait,n}$ as long as $t + \tau_{tr,n} \leq t + \tau sched, n-1$.

G. Tasks Dependencies

The simplest task model is the *data-partition model*, in which we only consider a pool of independent tasks at a given time. However, most AR systems cannot be decomposed into independent tasks, as most components require the output of a previous component as an input, as shown Section III. We consider three main tasks dependencies models. For a set of N interdependent tasks, the dependencies can either be linear, parallel or a combination of both. When a set of tasks are linearly dependent, if we consider that each task result has to be reported to the smartglasses before executing another, the total execution time is:

$$\Psi = \sum_{n \in N} \tau(T_n(t)) \tag{13}$$

In the case of parallel dependencies where the input of task T_N depends on the output of parallel tasks T_1 to T_{N-1}. The execution time of $N-1$ tasks dispatched over N_{res} servers is therefore constrained by the following equation:

$$\Psi < \tau(T_1(t)) + \frac{N-2}{N_{res}} \max(\tau(T_n(t))) + \tau(T_N(t)) \tag{14}$$

$n \in [2, N-1]$

Finally, tasks can show more intricate interdependencies. We can resolve this kind of topology by aggregating parallel or linearly dependent tasks in nested clusters, with an overall latency of $\tau_{cluster}$, until the full system turns into a linear or parallel combination of clusters.

H. Optimizations

Interdependent tasks introduce new constraints in the system, but also provide new opportunities for optimization. A set of N linearly dependent tasks can be considered as a single task of deadline $\sum_{n \in N} \tau_{d,n}$, transmitted on the same link and executed on the same server. The execution time of this set can be reduced by transmitting all tasks sequentially and executing them as soon as they are received and the previous task completed. The overall delay for this set of tasks becomes:

$$\psi = \tau_i(t) + \sum_{n \in [1,N]} \max\left(\frac{L_{data,n}}{B_i(t)}, \tau_{comp,n-1}\right) + \tau_{comp,n} + \tau_{res,N} \tag{15}$$

Similarly, for parallel dependencies, all tasks may be transmitted right after task T_1, reducing the total time to:

$$\Psi < \tau(T_1(t)) + \frac{N-2}{N_{res}} \max_n(\tau_{comp,n}) + \max_n(\tau_{res,n}) + \tau(T_N(t)) \tag{16}$$

$n \in [2, N-1]$, assuming that $\tau_{tr,n} < \tau(T_1(t)), \forall n \in [2, N-1]$

V. Scheduling algorithm

In this section, we propose a scheduling algorithm to allocated tasks over the set of available links and servers. We first introduce a system for independent tasks, then discuss the implications of tasks with linear and parallel dependencies.

Algorithm 1 Scheduling algorithm.

```
Input: Network bandwidth $\{B_i(t)\}$, latency $\{\tau_i(t)\}$, server available capacity $\{CPU(t)_j\}$, set of tasks $T_n(t)$, set of link/server combinations $(L_i, D_j, E_k, C_l)$
1: for task in T.sort($\tau_d$) do
2:     for link,server in combination do
3:         compute $\tau_{link,server}$
4:     end for
5:     allocate task $T_n$ to link/server with lowest $\tau_{link,server}$
6:     remove $T_n$ from T
7: end for
Output: Task allocation
```

A. Independent tasks

We consider sequential task allocation as it allows greater flexibility (see Section IV-F). We consider several metrics:

- $\{\tau_n\}$ the set of task completion times over all available servers and links.
- $\alpha_{\min} = \frac{\min(\tau)}{\tau_d}$

We assume that the set of links and servers is small enough to compute $\{\tau\}$ for all tasks in a reasonable amount of time. In our use case represented in Figure 3, the system features three links, one companion device, two edges servers and three cloud servers for a total of 9 possible combinations. We propose a simple two-steps algorithm to solve the resource allocation problem. While all tasks have not been assigned, we select the task with the closest deadline τ_d. We then compute the set of $\{\tau link, server\}$ corresponding to all possible link/server combination and allocate the task to the combination with the lowest $\alpha_{link,server}$. We summarize this method in Algorithm 1. This conservative algorithm aims at maximizing the number of tasks that can be processed in-time. Therefore, tasks that can complete in the least amount of time relative to their deadline are assigned first.

B. Interdependent tasks

In this section, we consider interdependent tasks. For more general relationships, the dependency graph can be decomposed into several linear or parallel clusters.

1) Linear dependency: : A set of N linearly dependent tasks can be considered a single task of deadline $\tau_d = \tau_{d,N}$. The task execution time of the aggregate can be computed using Equations 11 and 12. The aggregate is then assigned to a single server. If the tasks have independent deadlines without a constraint on the overall deadline or if we can estimate each deadline as a function of the overall deadline (for instance, $\tau_n = \frac{\tau_d}{X_n} \sum_{n' \in N} X_{n'}$), we can assign the first task and schedule the following tasks after it completes, recomputing the deadlines according to the actual completion time.

2) Parallel Dependency: : The set of parallel tasks can be considered as a cluster of independent tasks with deadline $\tau_d = min(\{\tau_{d,n}\})$ and response time $\tau = max(\tau_n)$. Tasks in this cluster can be allocated as independent tasks, using the redefined deadline and completion time for task assignment.

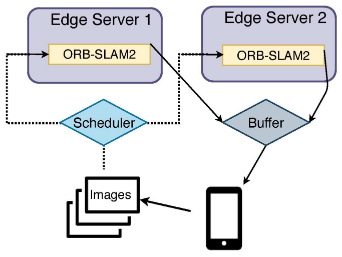


Fig. 4: System function flow

VI. Evaluation

In this section, we implement and evaluate a real-life proof-of-concept of our system. We develop a showcase application that performs real-time localization offloading by offloading Simultaneous Location And Mapping (SLAM) tasks to multiple edge servers simultaneously in real-time.

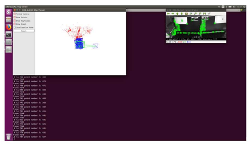
A. Implementation

On the Linux platform, we deploy a SLAM system with `ORB-SLAM2` [38], one of the most popular real-time SLAM libraries. We implement the client device on the Android platform, following the flow in Figure 4. The client phone captures images and sends the JPEG-encoded grayscaled frames at 30 FPS to edge servers either through the inbuilt scheduler or through a naive random allocation algorithm in real-time. Each edge server processes the received frame with ORB-SLAM2 and sends the results back to the client phone via a result buffer. The client phone pulls the results from the buffer and displays them in AR, as shown in Figure 5. The client sends compressed images at 30 FPS to the programs in the server(s) via two sockets. The average compressed frame size is 64 Kb. The server returns 12 Kb of data. Our implementation works with the smartphone's monocular camera (on Android phone) but is also compatible with stereoscopic cameras. To emulate a standard mobile user, we install the client-side application on a Huawei Mate9 Pro smartphone, with a 2.4 (1.8) GHz octa-core HiSilicon Kirin 960 CPU and 4GB of memory. We deploy the edge servers on two MSI GS65 Stealth 8SG[3], each of which has a 6-core I7-8750H CPU, 32GB of memory, and an Nvidia RTX 2080 Max-Q GPU. The hardware of our edge server is similar to a medium-priced commodity edge server.

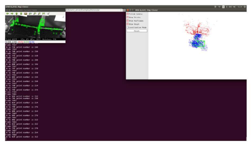
B. Single Server vs Scheduling vs Random

To test the performance of the scheduler, we throttle the uplink (client to server) bandwidth to 3,000 Kb/s using `wondershaper`[4]. The value is low enough to eliminate the impact of interferences (both at system and network level) that may lead in throughput variation. Furthermore, it emulates a low-connectivity scenario that is susceptible to happen with user mobility. We split this bandwidth between edge server 1 and 2 according to the following ratios: $1/3$, $1/4$ and $1/5$.

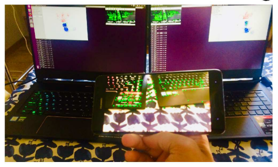
[3]https://www.msi.com/Laptop/GS65-Stealth-8SX/Specification
[4]https://github.com/magnific0/wondershaper



(a) Edge Server 1.

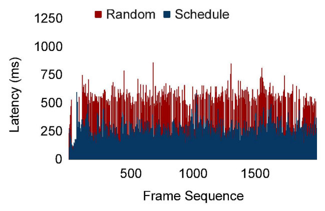


(b) Edge Server 2.

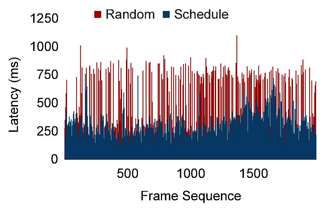


(c) Simulatneous mobile client offloading to two edge servers. Both servers maintain a map of the environment (a and b). The application combines both maps and displays it (c).

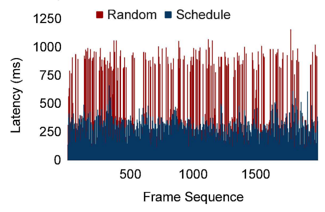
Fig. 5: Prototype system.



(a) Bw. ratio $1/3$.

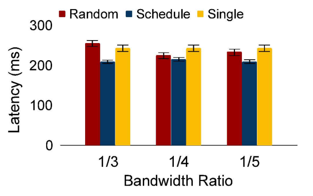


(b) Bw. ratio $1/4$.

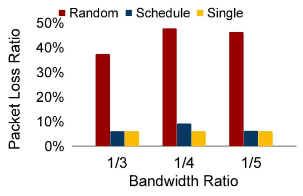


(c) Bw. ratio $1/5$.

Fig. 6: Latency vs time. Scheduling leads to lower average latency compared to unmanaged multipath offloading. Our algorithm also leads to significantly lower peak latency.



(a) Latency with standard error of the mean (SEM).



(b) Packet Loss Ratio.

Fig. 7: Result Comparison. Our scheduling algorithm leads to lower latency than single path offloading with similar bandwidth, at the cost of a slight raise in packet losses.

For each test, we sample 2000 frames. We plot the feedback latency in Figure 6. *Schedule* and *Random* refer to the tests in which the client sends requests via scheduler or randomly to the two servers. We also carry out a *Single* edge server

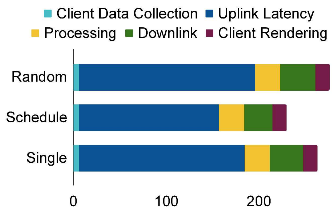


Fig. 8: Latency decomposition for bandwidth ratio $1/3$. The transmission latency is significantly lower with our scheduling algorithm, even compared to single-path offloading.

TABLE III: Base Network parameters.

	D2D	LTE	WiFi	Backbone
τ	1.75ms	9.95ms	1.85ms	1.15ms-19.25ms
B	50Mb/s	30Mb/s	100Mb/s	xxx

experiment via a 3000 Kb/s uplink uplink. We decompose the average latency for each system in Figure 8. As summarised in Figure 7, multi-server with scheduling always outperforms single-server and multi-server without scheduling considering the combinatory performance of feedback latency and transmission packet loss ratio. Interestingly, multi-server scheduling noticeably decreases the transmission latency compared to single-server offloading, while the aggregated total bandwidth remains the same. Due to the limited bandwidth (only 3 Mb/s), the transmission latency takes most of the time. With a higher throughput, we could reach round-trip latencies as low as 55 ms with a 7 ms round trip WiFi latency, and 26 ms processing latency, which is enough to perform mapping one frame out of two, similarly to the deadlines fixed in Section IV.

VII. Model Evaluation

To further validate the results of Section VI, we simulate more complex scenarios over various network conditions[5].

We use the AR application model presented Section III, with the two following simplifications: (1) The Feature Extraction step is performed on the same server than the dependent tasks, and (2) All JPEG frames are 64Kb, similarly to Section VI. As such, our model consists only of four tasks, Render object T_r, Track Objects T_t, Update world T_m, and Recognize objects T_o. We use the system in Figure 3 as our simulation setup.

At first, we consider fixed network bandwidth and latency. Although such a model does not represent a realistic wireless networks, it allows us to draw valuable insight into the effect of the system's intrinsic parameters on task repartition and in-time completion. We use the measurements from Section I for our typical values (see Table III). We assume that the access network is the bottleneck, thus the backbone network's bandwidth is equal to the access link's. To set up the task parameters, we perform real-life measurements with the following assumptions: (1) $25 CPU_{smartphone} < 5 CPU_{edge} < CPU_{cloud}$, (2) $CPU_{smartphone} = 200$, and (3) $\tau_{rendering,smartphone} = 10ms$. As such, for $CPU_{smartphone} = 200$, $CPU_{edge} = 1000$, and $CPU_{edge} = 5000$, we have $X_r = 2$. We then set up an Edge server (see configuration Section VI) running `OpenCV`[6] for tracking, `ORB-SLAM2` [38] for mapping, and `YOLO` [39] for recognizing objects. We use 64Kb JPEG frames as input (average frame size in Section VI) and measure the average processing time and output size. We then convert these measurement into the aforementioned metrics by using the formula $X = \frac{\textit{measured time}}{CPU_{edge}}$. As a result, for a 64Kb JPEG frame, $X_t = 10$, $X_m = 26$, and $X_o = 15$, and $L_{res,t} = 4\,Kb$, $L_{res,m} = 12\,Kb$, and $L_{res,o} = 4\,Kb$. Finally, rendering the object requires not only the JPEG frame, but also other data (object location, world model), and thus we consider $L_{data,r} = 100\,Kb$. We summarize these measurements in Table IV. We perform the following simplifications: (1) At all time t, the client has a good estimation of all the main parameters of the system: delays, bandwidth, and server capacity. (2) All the parameters are constant for a given task from the moment it is assigned. (3) All tasks are triggered by new frames.

We run all the simulations over 200 s for a 30 FPS application, with rendering performed for 3 objects every 30 ms, tracking every 60ms, world modeling every 90 ms and object recognition every 120 ms, for a total of 24,500 tasks.

A. *Offloading to multiple servers*

We first measure the impact of the following offloading scenarios: D2D only, edge computing only (single and multipath), cloud only (1 to 3 servers), and a combination of all the servers and links. We set the cloud latency to $7.5ms$, corresponding to a server in a nearby city. Figure 9 shows the average completion times of tasks, the number of tasks that could not meet their deadline, and the average excess latency.

TABLE IV: AR application measurements.

	T_r	T_t	T_m	T_o
τ_d	20 ms	60 ms	90 ms	120 ms
Measured L_{data}	100 Kb	64 Kb	64 Kb	64 Kb
Measured L_{res}	100 Kb	4 Kb	12 Kb	4 Kb
Converted X	2	10	26	15

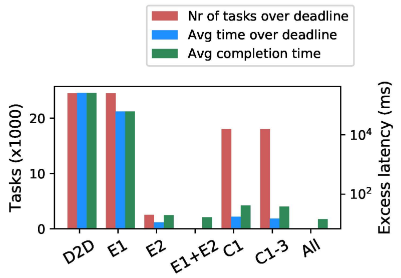
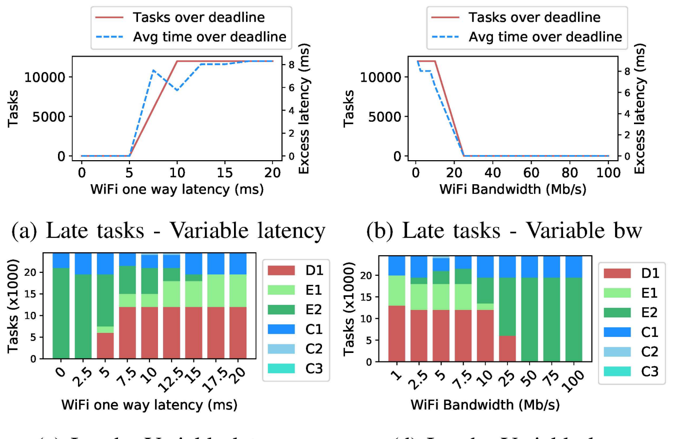


Fig. 9: Single and multiple servers offloading. Using several servers lead to reasonable excess latencies (<10 ms). 100% in-time task completion by offloading to all resources in parallel

[5]Code available at: https://github.com/Braudt/mpath-simulator-percom

[6]https://opencv.org/



(a) Late tasks - Variable latency (b) Late tasks - Variable bw

(c) Load - Variable latency (d) Load - Variable bw

Fig. 10: Impact of WiFi latency and bandwidth (bw) on task allocation and excess latency. For latencies $> 5\,ms$, in-time completion rate drops to 50%. Minimum completion rate is achieved for bandwidth $> 25\,Mb/s$

The companion device is not powerful enough to offload the full application, excess latencies accumulate exponentially, over 250 s. Similarly, due to the high latency of LTE links, using LTE leads to a buildup of latencies, around 60 s. Single path edge computing with WiFi shows only 10% of tasks completing late, and an average excess latency below 12 ms. Cloud servers compensate the high network latency with high computational power. However, excess latency remains high. Finally, transmitting to both edge servers through LTE and WiFi allow completing 100% of the tasks in-time with an average completion time of 16.6 ms. Unsurprisingly, the combination of all elements in the system leads to 100% on-time completion with the lowest completion times (avg=14.5 ms).

B. Impact of network conditions

In the previous section, we consider network conditions close to optimal with latencies and bandwidth fixed at values measured on the campus of the Hong Kong University of Science and Technology where both LTE and WiFi cover the entire area with little interference between the APs. We now analyze the case of degraded WiFi performance.

We first evaluate the effect of WiFi bandwidth and latency on performance. Figure 10(a) presents the influence of the WiFi latency on the in-time task completion, and Figure 10(b) the task allocation. A slight increase in link latency causes response times to exceed the deadlines. When the one way delay reaches 17.5 ms, the number of late tasks and the average excess latency stabilize and the allocation algorithm does not consider the WiFi network anymore. Tasks are primarily allocated through D2D and LTE. The mobile network is never used when $\tau_{WiFi} < 5\,ms$. In between, tasks are offloaded through both WiFi and LTE until $\tau_{WiFi} \geq 15ms$. On Figure 10(b) and 10(d), we represent the effect of WiFi bandwidth variation on task completion and task allocation. When the WiFi bandwidth is $< 25\,Mb/s$, tasks start not to

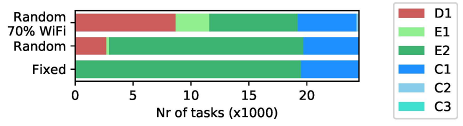


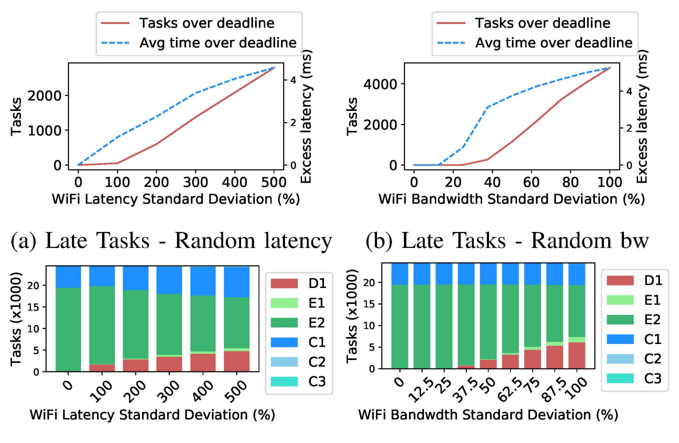
Fig. 11: Task Allocation for Fixed channel, Random channel, and Intermittent channel. WiFi instability causes more tasks to be assigned through D2D and LTE (30%).

be completed in-time, although with minimal impact on the excess latency. Under 2.5Mb/s, the WiFi network stops being used. Although WiFi link latency is primordial for in-time tasks completion, our system is much more tolerant to lower bandwidths. Furthermore, even in sub-optimal conditions, we keep the latency lower than 9 ms, thanks to the task reallocation to the companion device and the LTE edge server.

After analyzing the impact of both latency and throughput, we evaluate the effect of link instability. We represent the delay and bandwidth as random Gaussian variables centered around the values in Table III. To set the standard deviation, we perform measurements on the university campus. The WiFi measurements are performed on Eduroam, while the LTE measurements are done on a commercial operator's network. We send 100 consecutive Echo Request packets to the university server through each link. WiFi presents a standard deviation of 65% due to the extremely low average latency (2.7 ms one way). The latency varies between 1.7 ms and 12.1 ms LTE latency displays a standard deviation of 13%, ranging from 7.2ms to 13.55ms. Regarding bandwidth, we compute the standard deviation of 10 1 Mb transfers in Iperf. We find a standard deviation around 50% for both WiFi and LTE.

We compare fixed channel and random channels using the aforementioned parameters. Accessible WiFi networks are not as ubiquitous as LTE. In our simulation, we reflect this phenomenon by disconnecting the WiFi network randomly 20% of the time. For a random channel, the amount of tasks exceeding their deadline remains $< 4\%$, with excess times around 4 ms. With random WiFi disconnections, it reaches 28%, with excess latency around 7 ms. In both cases, the task repartition shown in Figure 11 is way more diverse than for fixed network conditions. The WiFi edge server is less used, while more tasks are allocated to the companion device as well as the cloud servers through the LTE links. Random WiFi disconnections increase the usage of cloud servers as the higher computing power compensates the increased link latencies. In both random channels and random channels with disconnections, LTE and D2D are used for more than 30% of the allocations. When WiFi is not available, tasks are first offloaded to the companion device as LTE latency is too high for in-time completion. We then simulate a single-server system using either the WiFi or the LTE network. We approximate all tasks to be sent as a single task using Equations 15 and 16. In both cases, 100% of the tasks do not finish on-time, and latency builds up to a system collapse.

As our LTE network is relatively stable, although rarely



(a) Late Tasks - Random latency (b) Late Tasks - Random bw

(c) Load - Random latency (d) Load - Random bw

Fig. 12: Impact of WiFi latency and bandwidth (bw) standard deviation on task allocation and excess latency. The system can withstand up to 100% stdev for bandwidth and 25% stdev for bandwidth until late tasks happen.

used in this experiment, evaluate the limits of WiFi instability on task completion rates and times. We consider the LTE link as fixed and vary the standard deviation of bandwidth from 0 to 100% (latency stdev=65%), and the standard deviation of latency from 0% to 500% (bandwidth stdev=50%). We present the results in Figure 12. Over 100% latency standard deviation, tasks start not to be completed in-time. The number of late tasks raises linearly with the standard deviation of latency. On the other hand, bandwidth is much more sensitive to random variations, and tasks can not complete in-time for a stardard deviation over 25%. Moreover, excess latency remains below 5 ms forr both parameters, thanks to the task reallocation.

C. Future-proofing: 5G and 802.11ax

The model presented above considers the current network technology. However, 5G is already being deployed in multiple locations, and 802.11ax certification was started September 19, 2019. Both bring a considerable increase in bandwidth. 802.11ax increases data rates up to 600 Mb/s with 80 MHz channels [40], while 5G users reported download speeds between 600 Mb/s and 1.8 Gb/s [41]. We reflect on these technologies in a final simulation. We consider a 802.11ax link with a 600 Mb/s bandwidth (stdev=100 Mb/s), and a latency similar to the previous WiFi settings. Regarding LTE, we estimate a 1,000 Mb/s bandwidth (stdev=100 Mb/s), and a latency of 1 ms [42] (stdev=0.13 ms, similar to the previous setting). These parameters represent a moderately stable scenario for these future technologies. We expect the system to withstand our application with no late tasks. We simulate the same application running at 60 FPS, with a deadline of 30 ms for mapping, tracking, and object recognition.

We show the task allocation on Figure 13. As expected, the load is more balanced between the WiFi and the mobile network. Interestingly, the cloud servers are still used to the high computational cost. In the case of a 60 FPS application

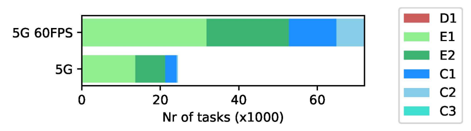


Fig. 13: Task Allocation for our model application in 5G and the same application running at 60FPS. The load is more balanced between the 5G link and the 802.11ax link as the bandwidth and latencies are closer.

with all tasks performed for each frame, 30% of the tasks fail to complete in-time, although the excess latency remains below 2 ms. Indeed, computing power becomes the bottleneck and more tasks get assigned to the cloud. This experiment shows that although 5G may be a game-changer, some applications may still not be fully offloaded due to the high computation cost of some tasks and the amount of data to send.

VIII. Conclusion

In this paper, we modeled and analyzed the impact of multipath transmission on multi-server offloading. We modeled a multiple-servers, multiple-links architecture for offloading a typical MAR application to companion devices, edge, and cloud, and designed an algorithm for tasks allocation. We evaluated this algorithm through both real-life implementation and extensive simulations. Our algorithm performs better than random allocation and single-server offloading for an equivalent aggregated bandwidth. Allocating tasks over a wide range of networks and servers noticeably improve performance over single path offloading. Even a very low power companion device can distinctly increase the in-time completion rates. We also shed light on the dependency of the system on latency stability. The system was extremely resilient, and requires extreme conditions to fail compared to its single-server counterpart. Offloading over such a heterogeneous system provides robust fall-back in case resources are not available. Over the range of simulations, excess latency was kept below 9 ms in conditions where single-server systems would collapse. Finally, we showed that despite 5G and 802.11ax being significant improvements, the most computation-heavy applications may still not be fully offloaded for in-time completion.

In the future, we plan to further evaluate the system and extend our real-life implementation. We will focus on optimizations that can happen in the backbone network, such as task aggregation and chaining. Besides, we will evaluate the overhead of our solution, both in terms of network and server capacity estimation, and in terms of practical network implementation, including connection establishment, discovery, and handover. Finally, we will study in detail the impact of 5G and WiFi 6 through real-life network measurements.

Acknowledgment

This research has been supported in part by project 16214817 from the Hong Kong Research Grants Council, the 5GEAR project and the FIT project from the Academy of Finland.

References

[1] M. Abrash, "Latency – the sine qua non of ar and vr." Accessed 23-02-2017.

[2] T. Braud, T. Kämäräinen, M. Siekkinen, and P. Hui, "Multi-carrier measurement study of mobile network latency: The tale of hong kong and helsinki," in *15th International Conference on Mobile Ad-hoc and Sensor Networks*, December 2019.

[3] J. Licklider, "Memorandum for : Members and affiliates of the intergalactic computer network," 1963.

[4] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mob. Netw. Appl.*, vol. 18, pp. 129–140, Feb. 2013.

[5] M. Shiraz, A. Gani, R. H. Khokhar, and R. Buyya, "A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing," *IEEE Communications Surveys Tutorials*, vol. 15, pp. 1294–1313, Third 2013.

[6] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *CoRR*, vol. abs/1702.05309, 2017.

[7] S. Yu, R. Langar, and X. Wang, "A d2d-multicast based computation offloading framework for interactive applications," in *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, Dec 2016.

[8] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, (New York, NY, USA), pp. 49–62, ACM, 2010.

[9] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, (New York, NY, USA), pp. 301–314, ACM, 2011.

[10] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 Proceedings IEEE INFOCOM*, pp. 945–953, March 2012.

[11] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, *Cuckoo: A Computation Offloading Framework for Smartphones*, pp. 59–79. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[12] M. Breitbach, D. Schäfer, J. Edinger, and C. Becker, "Context-aware data and task placement in edge computing environments," in *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom*, pp. 1–10, March 2019.

[13] D. Wagner and D. Schmalstieg, *First steps towards handheld augmented reality*. IEEE, 2003.

[14] B. Shi, J. Yang, Z. Huang, and P. Hui, "Offloading guidelines for augmented reality applications on wearable devices," in *Proceedings of the 23rd ACM International Conference on Multimedia*, MM '15, (New York, NY, USA), pp. 1271–1274, ACM, 2015.

[15] P. Jain, J. Manweiler, and R. Roy Choudhury, "Overlay: Practical mobile augmented reality," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '15, (New York, NY, USA), pp. 331–344, ACM, 2015.

[16] A. Asadi, Q. Wang, and V. Mancuso, "A survey on device-to-device communication in cellular networks," *IEEE Communications Surveys Tutorials*, vol. 16, pp. 1801–1819, Fourthquarter 2014.

[17] A. Fahim, A. Mtibaa, and K. A. Harras, "Making the case for computational offloading in mobile device clouds," in *Proceedings of the 19th Annual International Conference on Mobile Computing &#38; Networking*, MobiCom '13, (New York, NY, USA), pp. 203–205, ACM, 2013.

[18] B. Han, P. Hui, V. S. A. Kumar, M. V. Marathe, J. Shao, and A. Srinivasan, "Mobile data offloading through opportunistic communications and social participation," *IEEE Transactions on Mobile Computing*, vol. 11, pp. 821–834, May 2012.

[19] D. Chatzopoulos, K. Sucipto, S. Kosta, and P. Hui, "Video compression in the neighborhood: An opportunistic approach," in *2016 IEEE International Conference on Communications (ICC)*, pp. 1–6, May 2016.

[21] L. Civolani, G. Pierre, and P. Bellavista, "FogDocker: Start Container Now, Fetch Image Later," in *UCC 2019 - 12th IEEE/ACM International Conference on Utility and Cloud Computing*, (Auckland, New Zealand), pp. 51–59, ACM, Dec. 2019.

[20] K. Sucipto, D. Chatzopoulos, S. Kosta, and P. Hui, "Keep your nice friends close, but your rich friends closer – computation offloading using nfc," in *2017 Proceedings IEEE INFOCOM*, 2017.

[22] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, pp. 3590–3605, Dec 2016.

[23] S. Barbarossa, S. Sardellitti, and P. D. Lorenzo, "Communicating while computing: Distributed mobile cloud computing over 5g heterogeneous networks," *IEEE Signal Processing Magazine*, vol. 31, pp. 45–55, Nov 2014.

[24] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, pp. 14–23, Oct 2009.

[25] "Mobile-edge computing-introductory technical white paper." Accessed 29-07-2017.

[26] Intel, "Real-world impact of mobile edge computing (mec)." Accessed 29-07-2017.

[27] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9, April 2016.

[28] Y. Li and W. Gao, "Code offload with least context migration in the mobile cloud," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 1876–1884, April 2015.

[29] K. Wang, M. Shen, J. Cho, A. Banerjee, J. Van der Merwe, and K. Webb, "Mobiscud: A fast moving personal cloud in the mobile network," in *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, AllThingsCellular '15, (New York, NY, USA), pp. 19–24, ACM, 2015.

[30] C. Westphal, "Challenges in networking to support augmented reality and virtual reality," 2016.

[31] W. Zhang, B. Han, and P. Hui, "On the networking challenges of mobile augmented reality," 2017.

[32] T. Braud, F. H. Bijarbooneh, D. Chatzopoulos, and P. Hui, "Future networking challenges: The case of mobile augmented reality," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1796–1807, June 2017.

[33] P. Zhou, W. Zhang, T. Braud, P. Hui, and J. Kangasharju, "Enhanced augmented reality applications in vehicle-to-edge networks," in *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pp. 167–174, Feb 2019.

[34] C. Cicconetti, M. Conti, and A. Passarella, "Low-latency distributed computation offloading for pervasive environments," in *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom*, pp. 1–10, March 2019.

[35] "Vuforia." https://vuforia.com/. Accessed: 2017-07-30.

[36] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*, MCS '12, (New York, NY, USA), pp. 29–36, ACM, 2012.

[37] K. Y. Lam, L. Hang Lee, T. Braud, and P. Hui, "M2a: A framework for visualizing information from mobile web to mobile augmented reality," in *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom*, pp. 1–10, March 2019.

[38] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, pp. 1255–1262, Oct 2017.

[39] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018.

[40] M. Turner, "Wi-fi 6 explained: The next generation of wi-fi," sep 2019.

[41] I. Fogg, "5g users now experience max download speeds over 1000 mbps in 4 countries," sep 2019.

[42] N. Alliance, "5g white paper," *Next generation mobile networks, white paper*, 2015.