Introduction to the Computation Offloading from Mobile Devices to the Edge of Mobile Network

Jakub DOLEZAL, Tomas ZEMAN

Department of Telecommunication Engineering, Faculty of Electrical Engineering,
Czech Technical University in Prague, Jugoslavskych partyzanu 1580/3, 160 00 Prague, Czech Republic

jakub.dolezal@fel.cvut.cz, tomas.zeman@fel.cvut.cz

DOI: 10.15598/aeee.v17i4.2695

Abstract. *This paper introduces the concept of Small Cell Cloud (SCC) composed of multiple Cloud-enabled Small Cells (CeSCs), which provide radio connection for mobile User Equipment (UE) such as smart-phones or wearables such as smart glasses. Moreover, CeSCs host computations offloaded from UEs in a way similar to centralized cloud, yet different in its proximity to users. Proposed client-server architecture of SCC conveys mechanisms for moving offloaded computations from the UEs to CeSCs. Real-life implementation of the SCC architecture relies on custom-developed Offloading Framework which is responsible for low-level communication between the UE and the SCC. The Offloading Framework is accompanied by an Augmented Reality (AR) app, which employs intensive computations for discovery of places of interest. Such app is latency-sensitive, a criterion which makes computation offloading beneficial due to its ability to decrease latency. The combination of the Offloading Framework and the AR app makes up an SCC testbed used for further performance evaluation. Numerous measurements are carried out to examine the impact of various parameters. Based on Proof-of-concept implementation and thorough measurements, it has been revealed that computation offloading can decrease overall latency as much as to 47 % and energy consumption on the UE side to 56 %.*

Keywords

Computation offloading, performance evaluation, Small Cell Cloud.

1. Introduction

Current advances of mobile User Equipment (UE), such as smartphones, tablets and wearables (including smart glasses), lead to the state of omnipresent connection to the Internet and subsequent heavy reliance on the server side. Mobile developers largely exploit this situation, pushing the boundaries of the UE and systematically adding more logic to mobile apps, which naturally leads to high utilization of CPU and shortened battery life. Although the UE gradually becomes much more powerful than in the past, offering more raw power to developers and mobile apps, this trend has to face the limitation of battery capacity, which in turn limits the duration of computations behind the apps. Even though current UE is equipped with powerful CPUs, the cost of energy for exploiting its full potential may prove to be very high [1].

The traditional way to overcome battery capacity problem is simply to let the server-side or cloud perform the computation-intensive logic operations. Many techniques facilitate such effort, for example Remote Procedure Call (RPC). The key idea behind RPC is to let a mobile app act as a client to call a procedure or a method deployed on a remote computer accessible through a network [9]. The theoretically unlimited capacity and scalability of cloud lead to the idea of replacing the standard RPC by computation offloading. The difference from RPC consists in the possibility to call a method in a unified manner without explicitly specifying where the responsible core resides – be it either a mobile app or cloud. The decision should be made by a dedicated software module (decision maker) which ensures that energy or latency savings are attained. Such decision relies on profiling of the previously executed computations and on estimation of the upcoming ones. Profiling always involves energy consumption. Moreover, certain CPU and memory usage data are collected for on-UE execution, as well as radio

conditions and resource allocation for offloaded execution.

In traditional mobile networks, communication involves multiple routers, each contributing to the overall latency. Moreover, Radio Access Network (RAN) consisting of widely-spread Macro Cells (MCs) can contribute to the latency even more in case of traffic congestion by degrading the quality of service [1]. Hence, relying on mobile data networks when offloading to the centralized cloud can introduce major drawbacks for real-time apps which require low latency, such as augmented or virtual reality.

To overcome these difficulties, the idea of Small Cell Cloud (SCC) consisting of Cloud-enabled Small Cells (CeSCs) is introduced [2]. The purpose of SCC is to bring computation infrastructure to the vicinity of users through decentralization of mobile network and cloud. Basic Small Cells (SCs) were supposed to be deployed in crowded areas such as squares, offices, schools, shopping malls etc. to increase coverage and network throughput and to decrease energy consumption and latency [8]. The CeSCs merge the capabilities of the SCs with cloud computing – having sufficiently powerful CPU, memory and special middleware to be able to perform user computations on-demand. Assigning of hosting CeSCs to offloading computations is orchestrated by Small Cell Manager (SCM).

There is a set of similar technologies, such as Fog Computing or Mobile Edge Computing (MEC), on the same basis as SCC. The MEC is an innovative standard that should merge cloud computing and mobile networks [6, 7]. The difference between MEC and SCC is that the former is a general architecture which brings computations required for radio allocation from the core of a mobile network to MCs on its edge to minimize signaling overhead, while the latter seeks for a way to spare some portion of MEC's computation capacity for the host users' computations.

In the future, the MEC could serve as a technology enabler for a new generation of low-latency mobile apps relying on intensive computations hosted by CeSCs, thus saving battery life. The the recent years, a deployment of MEC has surged worldwide [29]. Yet its impact is limited to increased radio coverage and throughput as the means of providing better data connectivity mostly for smartphones. Data carriers are in a constant seek to expand their business and to make a transition from mere data delivery to become service providers. Deploying storage in a proximity to users in a form of Content Delivery Networks (CDNs) is a common practice [31] exercised by major cloud companies [30]. Adding computation faculties seems to be the next step to reduce latency, yet doing so have to be preceded by thorough examination of proper architectures and technologies.

The contribution presented in this paper includes the description of offloading architecture, proof-of-concept implementation of the offloading framework, and finally performance evaluation for various metrics and parameters. Due to the novelty of SCC that results in the lack of appropriate hardware or firmware, software emulation is used as a workaround to approximate real-life conditions as close as possible.

The rest of this paper is organized as follows: the next section provides an overview of related work, Sec. 3. introduces the SCC testbed including the distributed nature of computation offloading, Sec. 4. explains measurement setup, Sec. 5. demonstrates advantages of the SCC by showing performance measurements, and finally Sec. 6. concludes the paper.

2. Related Work

The field of computation offloading is mapped by various papers pointing out two major problems. The first one is device profiling which aims to gather low-level information about CPU and memory usage. The second one is decision making whether a piece of code should be offloaded, based on previous profiling.

More general approach of architecture overview and discussion on suitable apps is followed in [10]. However, any experiments, algorithm proposals or new contributions are lacking. Theoretical simulations without any real-life measurements are provided by [11].

The hands-on results are delivered by papers examining various offloading-ready apps, including games, face recognition and translation from one language to another, accompanied by their performance evaluations over Wi-Fi and 3G networks [12]. Multiple offloading techniques are evaluated also by [13]. Radio-related offloading conditions are analyzed by [14]. Another offloading framework is rigorously described and its performance evaluated for various scenarios, including N-queens problem, face detection, virus scanning and image combination, in [15].

To conclude, the existing papers seem to prefer more theoretical approach or rely on benchmarks only. Such methodology omits more complicated problems which emerge only in real-life scenarios. This paper strives to share deeper knowledge of computation offloading, which in turn requires a real-life, non-trivial app prepared for thorough examination.

Some of the existing papers focus on more general radio- or cloud-related aspects, not really dealing with the offloading itself: distribution of payload over 5G networks, resource allocation and radio signaling [16]; the benefits of infrastructure virtualization

and software-defined networks [17]; improvements of TCP performance of high-throughput radio [18].

The scalability of Mobile Edge Computing for offloading is assessed in [26]. A radio-based architecture is proposed with focus on the domain of Internet of Things. A similar approach has been published in [28]. The deployment of MEC along software defined ultra-dense networks is further described [27]. The paper proposes various models for communication of controlling.

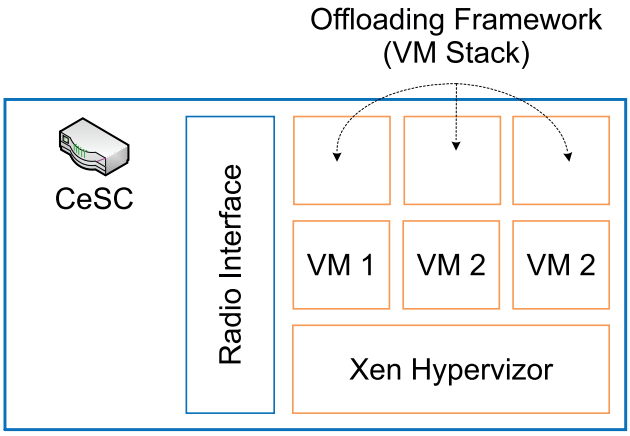
Models for computation offloading to the SCC are proposed along with theoretical discussion, yet not really proven in any proof-of-concept setup linked to real conditions [19]. The femto-cloud is a concept sharing some traits with the MEC, including the computation offloading, but only simulations results are provided to demonstrate its benefits [20].

To conclude the findings mentioned above, MEC is one of the main fields of mobile engineering being backed by an array of contributions. Yet the majority of them examines MEC from radio perspective, often taking simulation approach of evaluation. Computation offloading is indeed seen as a part of MEC tech stack, yet the evolution of proposed architectures is aligned with radio backgrounds of their authors. The lack of testbeds serving for proof-of-concept purposes is surely the major setback for the community.

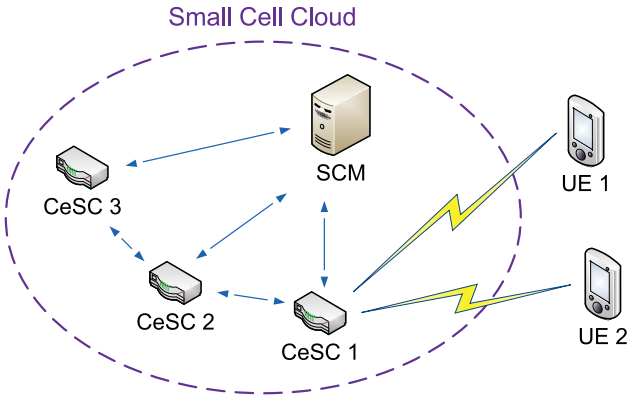
This paper introduces MEC testbed composed of emulated SCs. Each SC is built on top of general-purpose hardware and customized with communication- and offloading-enabled software. This way facilitates an insight into offloading and an investigation of real MEC behavior.

3. SCC Architecture

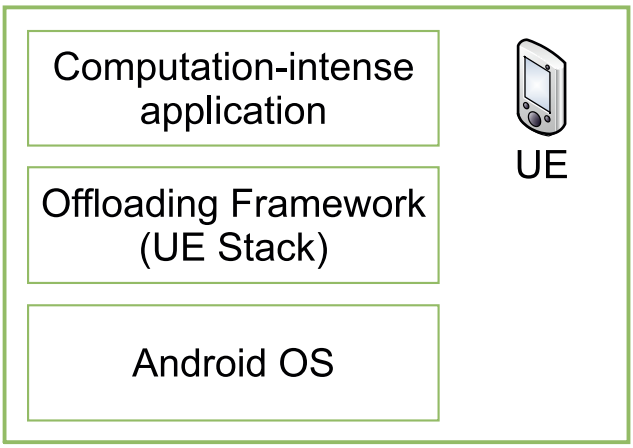
This section explains the offloading architecture of the Small Cell Cloud (SCC) and its proof-of-concept implementation as depicted in Fig. 1. The core concepts of the Small Cell Cloud (SCC) are explained in [2]. The SCC contains multiple Cloud-enabled Small Cells (CeSCs), each acting as both a radio access point for the UE and a host for offloaded computations demanded by the UE [3]. The computations inside the CeSCs are managed by a hypervisor running one or more Virtual Machines (VM), depending on users's activity and CeSCs hardware configuration. The UE is connected to the SCC through one of the CeSCs via Wi-Fi (the current setup) or LTE (the desired final state). Offloaded computations may be hosted by the CeSC through which the UE is connected (the optimal state) or by another one if the former does not have spare computational capacity. Users' privacy is very important; when the offloaded computation is finished and the results are sent back to the UE, the allocated memory is purged so that the results cannot be accessed by a subsequent user.



(a) Components of CeSC consisting of radio end-point and computation hosting based on a hypervisor and the developed Offloading Framework.



(b) Network topology of the SCC.



(c) Integration of the app with the Offloading Framework.

Fig. 1: The architecture of Small Cell Cloud (SCC).

The CeSCs are assigned the incoming offloading requests by the overseer entity called Small Cell Manager (SCM). The SCM strives to minimize latency and over-

head within the SCC and to optimize resource usage across the SCCs [4]. It is the responsibility of SCM to know the SCC topology and status, health and free capacity of all CeSCs within the SCC. The communication protocol between the SCM and the CeSCs is concealed from the UE for security reasons; the CeSCs communicate with the SCM on behalf of the UE [3]. To prevent the single point of failure design flaw, there is one or even more secondary SCMs shadowing the current knowledge of the primary SCM so that they can replace the primary one fast.

3.1. Offloading Framework

The Offloading Framework is a piece of software which facilitates the communication between the UE and the SCC. The UE stack of the Offloading Framework is added to offloading-enabled apps during compilation. In runtime, the UE stack acts as a client on the app's behalf and handles networking and low-level system operations required for the offloading to work.

A part of offloading-enabled app performs heavy computations which can be offloaded. In the terminology of the Offloading Framework, this part is called offloadable module and its execution is monitored and profiled by the UE stack. The module is designated to be offloadable using Java annotation [22]. Other parts of the app are not relevant for offloading, mostly due to low latency or direct access to UE's hardware such as screen or location sensor. After the app is launched, the UE stack begins with SCC connection and module profiling. Once the SCC becomes available, the UE stack authorizes and authenticates itself to provide security and privacy. Then the app installation file is retrieved from the UE and uploaded to the SCC to be ready for offloaded execution.

Future versions will cease this perpetual uploading and deliver a repository of certified offloading-enabled apps. The repository will provide a stack of functions ranging from basic storage to app-specific profiling information.

From now on, executions of offloadable modules are intercepted in runtime and a decision is made if the execution will proceed on-UE or offloading request to the SCC should be made instead. The current implementation provides limited support for runtime decisions; a constant decision is made instead during performance evaluation below. The implementation of runtime decision making is underway, so the UE stack will be able to learn from the previous executions and to predict latency and energy consumption in the future. In case of offloaded execution, the UE stack sends offloading request to the SCC through the CeSC to which it is connected. This CeSC asks the SCM to select which CeSC should host the computation. If the hosting CeSC is different from the CeSC to which the UE is connected, the latter simply forwards the offloading request to the former, as shown in Fig. 2. As soon as the offloading request arrives to the hosting CeSC, its data are passed to the VM Stack of the Offloading Framework which is pre-installed on the VM image.

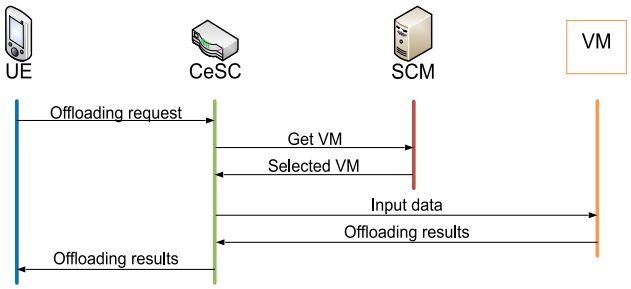


Fig. 2: Resource allocation at the beginning of the offloading process – the CeSC serves as a relay between the UE and the VM.

The logic behind switching between the on-UE and offloaded executions of the offloadable module is transparent to the calling part of the app which is guaranteed to obtain indistinguishable results. The technique of calling the same method on the UE and VM side is accomplished collaboratively by both stacks of the Offloading Framework. Signature of the desired method along with input data is serialized by the UE stack, then transferred to the VM, and finally deserialized by the VM stack which launches the method using Java reflection [21], waits for its results and sends them back to the UE. Currently, the Framework supports Android OS [23]; the VMs run Android x86 image.

The already existing apps can leverage the offloading by annotating the classes containing offloadable code with the *@Offloadable* keyword defined by the Offloading Framework. Methods of such classes are divided to offloadable and non-offloadable; the former have to be annotated with the same *@Offloadable* keyword. Moreover, designated parameters and class fields with direct impact on computation complexity are also annotated by *@Variable* keyword. While some papers propose automatic learning of offloading algorithms (which parameters and fields are important), such approach is hard to comprehend and predict for a developer. Instead, the Offloading Framework approach is to let a developers make decisions based on their knowledge of their own code.

All method calls to the annotated classes are intercepted in runtime by bytecode instrumentation on low level of Android Runtime. Calls to non-offloadable methods are intercepted, exercised on the target class yet also stored and queued for later offloading execution. Calls to offloadable methods are intercepted in a different manner – it is first decided if offloading is beneficial, based on variable values retrieved in run-

time using Java Reflection mechanism. In case of offloading execution, the previously queued calls of non-offloadable methods are serialized and sent to the VM as a part of the offloading payload. This approach is based on convention over configuration approach while striving for unified way of method calls. No matter if offloading actually happens, calls of both non-offloadable and offloadable methods are handled as local calls without dealing with stubs, brokers or any similar concepts typical for Remote Procedure Calls.

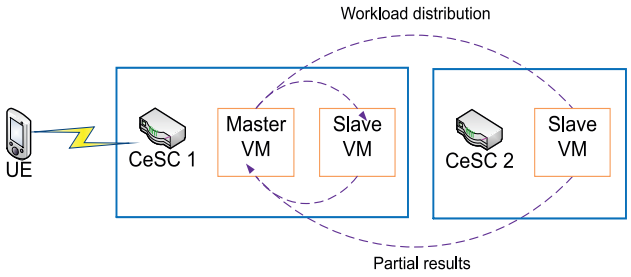


Fig. 3: Calling of a computation-intensive method is intercepted by Java reflection, decision and subsequent offloaded execution is done transparently for the developer.

3.2. Offloading Communication

This section describes in detail how all elements of the SCC mentioned above work together to make offloading work. The CeSCs firmware supports data adding, removing and purging on demand, hence allowing fast changes of the SCC topology and providing additional computation capacity exactly in the user's location. The firmware is built on top of Xen hypervisor [24] managing an array of VMs, which actually host offloaded computations. Computations hosted by the VMs are stateless for higher performance and security. Subsequent offloading requests demanded by the same UE can be hosted by different VMs and even CeSC depending on the total amount of offloaded executions.

There are more strategies how the CeSC can report its status to the SCC. One of them repeatedly uploads status data to the SCM including CPU and memory utilization so that the SCM can assign offloading requests accordingly [3]. Moreover, the SCC is capable of distributing a single offloading request to multiple parallel VMs, each having a disjunctive subset of input data. Although it is preferable to keep all parallel VMs on the same CeSC due to networking overhead, it is not always possible due to resource fragmentation. In such situation, parallel VMs are selected by the SCM with the criterion of minimum networking overhead and close position regarding the SCC topology. The workload employs single master – multiple slaves approach as shown in Fig. 3. The master VM divides the workload to slave VMs, merging the slaves' contribution into the response to the UE. The parallelization is transparent to the UE since the UE stack communicates with the VM stack of the master only.

Regarding the optimization towards efficient offloading patterns, the apps should always break too complex modules into simpler ones. Such pattern makes the process of decision-making faster and less error-prone; it also reduces networking overhead due to sending data to the SCC and receiving data by the UE. Even the adaptation to quickly changing radio conditions is much faster if execution can return quickly before the previously considered conditions change [5].

4. Experimental Setup

The setup of performance evaluation is thoroughly described in this section. In order to demonstrate how offloading can improve execution of intensive computations, the Offloading Framework needs some app which actually employs such computations while requiring low latency. For this reason, an Augmented Reality app Percipio was selected to serve as the base scenario for all further measurements in this paper. The public version of Percipio can be obtained from Google Play [25]. The performance evaluation is conducted with slightly modified version integrated with the Offloading Framework.

Percipio runs on Android UE and discovers places of interest visible in the UE's field of vision. A marker for each discovered place is than rendered on the UE's screen as an overlay over its camera image. Augmented Reality is a perfect example of technology suitable for computation offloading due to its CPU-heavy computations running in a loop and loosely coupled logic behind place discovery. Percipio is installed on Sony Xperia Z3 smartphone and Samsung Galaxy Tab 10.1 tablet, the configuration of which is shown in Tab. 1. Unfortunately, SCC is not yet commercially available at the time of writing this paper. Software emulation was chosen instead as the only means of delivering real-life results. CeSCs within SCC run on general-purpose servers, the configuration of which is specified in Tab. 2. Instead of using LTE connection, the SCC is available through a dedicated Wi-Fi network.

While the on-UE latency is caused by CPU computations only, the total offloading latency consists of three subsequent phases: 1) the radio phase contains both sending and receiving of data between the UE and the CeSC; 2) during the storage phase, definitions of places are loaded from an external storage by the VM; 3) during the calculation phase, computations are run by the VM instead of the UE. The on-UE and offloaded executions of Percipio are compared in terms of the following metrics:

- *Latency* – is the measure of time from the beginning of a computation to its end. The exact nature of latency depends on the execution type. For an on-UE execution, it is simply the duration of computation in the UE's CPU. For an offloaded execution, the latency is a sum of partial steps behind offloading, such as data upload from the UE to the SCC, data download from the SCC to the UE and computation on the SCC-side.
- *Energy* – means how much energy Percipio consumes during computation or offloading. For an on-UE execution, the energy is mostly consumed by CPU. For an offloaded execution, the energy is mostly consumed by radio transmission, since the UE's CPU is almost idle. In both cases, the energy does not include the consumption of background Android processes and screen.

Results for the aforementioned metrics depend on Percipio-relevant parameters explained below. These parameters are set in runtime before conducting measurements to reveal their impact on offloading.

- *Discovery Range* sets how distant a place of interest can be from the UE's location in order to be loaded by Percipio into memory and effectively become a part of the input data. Higher Discovery Range increases the number of places loaded and the volume of the input data. Double Discovery Range increases the volume of the input data four times, which means quadratic incrementation of computation complexity.
- *Parallelization* sets the size of elements working in parallel on place discovery. The exact meaning of an element depends on execution type. For an on-UE execution, the element is a simple thread having been assigned to the UE's CPU core. For an offloaded execution, the element is a thread of CeSC being managed by a hypervisor. Computation workload is always distributed equally among all available elements. Parallelization value of one means that the computation is carried over in a sequential way as it is intended for a benchmarking purpose.

Every measurement is run in a cycle and repeated 129 times. Data from the first iteration are removed from the final results to prevent pollution from cold-start – allocation of threads, IO operations, etc.

5. Performance Evaluation

Figure 4 shows the impact of different values of discovery range on the latency. For very short discovery ranges (up to 200 m), the complexity of computations is low. For UE 1, the latency during offloading is higher for discovery range 100 and 200 m, respectively. Conversely, offloading latency for UE 2 is reduced for the same discovery ranges. This difference is caused by inferior CPU performance of UE 2. It is evident that offloading of low-complex computations from powerful UE can mitigate faster completion on the VM side due to offloading radio overhead. Due to quadratic computation complexity, the on-UE latency steeply rises from discovery range of 300 m onwards. Although the absolute value of latency of an offloaded execution also slowly increases for larger values of discovery range, more powerful CPU on CeSC side is capable of completing it much faster compared to the UE's CPU. Moreover, the overhead of offloading has lesser impact on the overall latency for more complex computations. For discovery range of 300 m, offloading decreases the latency by 44 to 61 % for the UE 1 and UE 2, respectively. For discovery range of 500 m, offloading reduces the latency even more by 46 to 48 % for the UE 1 and UE 2, respectively. Gradually increasing computation complexity seems to diminish the differences between various UE types. This is apparently caused by the differences in CPU performance.

Tab. 1: Configuration of the measured UE.

Specification	UE 1	UE 2
CPU Clock	2.5 GHz	1.9 GHz
RAM Memory	3 GB	2 GB
Battery Capacity	3100 mAh	8220 mAh
Android Version	6.0.1	4.4

Tab. 2: Configuration of CeSC hosting the computations.

CPU Type	CPU Clock	Memory
Intel Core i5	2.4 GHz	8 GB

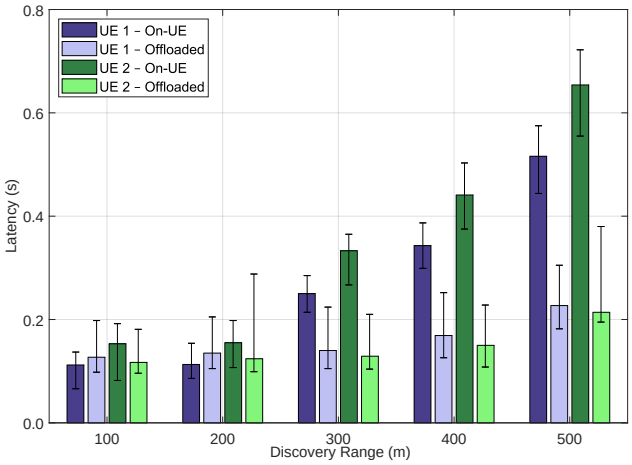


Fig. 4: Impact of discovery range of an Augmented Reality app on latency.

Figure 5 depicts the impact of discovery range on the energy consumption. For discovery ranges up to 200 m, offloading consumes more energy compared to on-UE execution. However, the absolute amount of energy spent by both on-UE and offloaded executions is negligible for both UE types (less than 200 mJ). For discovery range of 300 m, energy consumption while offloading is further decreased by 27 to 43 % for the UE 1 and UE 2, respectively. For discovery range of 500 m, offloading yields even more significant decrease by 44 to 56 % for the UE 1 and UE 2, respectively. The efficiency of offloading increases for more complex computations, since the volume of data transmitted between the UE and the CeSC rises much slower compared to the computation complexity. This pattern makes offloading efficiency actually dependent on radio conditions instead of computation complexity. Signal strength of radio connection was kept constant during these measurements. Complementary, dedicated measurements below examine the impact of decreased throughput.

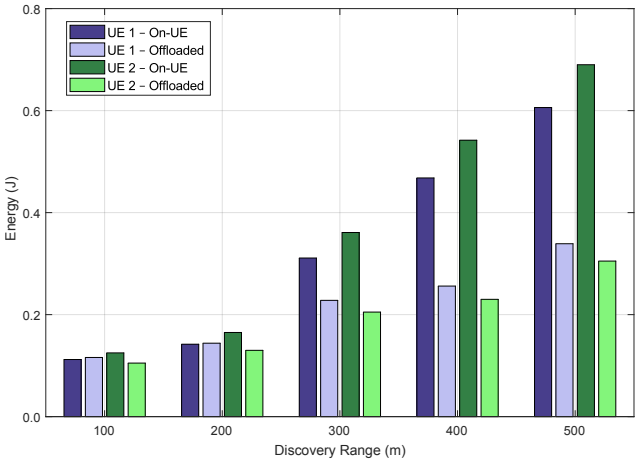


Fig. 5: Impact of discovery range of an Augmented Reality app on energy consumption.

Figure 6 shows how parallelization influences the latency. The discovery range of 300 m is used, hence the latency for no parallelization corresponds to the results above. On-UE parallelization is implemented by threads running on available CPU cores. For UE 1, each running thread decreases the latency – 2 threads by 17 %, 3 threads by 11 %, and 4 threads by 9 % compared to no parallelization, 2 threads, and 3 threads, respectively. For UE 2, more threads also reduce the latency – 2 threads by 25 %, 3 threads by 10 %, and 4 threads by 8 % compared to no parallelization, 2 threads, and 3 threads, respectively. The parallelization scaling hereby exponentially decreases when running more threads. Although the UE 1 has a quad-core CPU, and therefore is capable of running up to 4 threads in parallel, the necessity to access shared memory degrades its computation power. Actually, running 3 or more threads does not exploit CPU cores at 100 % – mean utilization of each core is 77 % for 3 threads and 62 % for 4 threads. Offloading parallelization distributes workload among multiple VMs, as described in Subsec. 3.2. For UE 1, putting 2 VMs to work decreases the latency by 14 % compared to no parallelization. Other VMs added gradually further decrease the latency even more – 3 VMs by 15 % and 4 VMs by 4 % compared to 2 VMs and 3 VMs, respectively. For UE 2, having 2 VMs reduces the latency by 13 % compared to no parallelization. More VMs reduce the latency in a similar manner – 3 VMs by 15 % and 4 VMs by 6 % compared to 2 VMs and 3 VMs, respectively. Similarly to on-UE execution, parallelization scaling also exponentially decreases when running more VMs. This is caused by the role of offloading and parallelization overhead, which mitigates the decreased computation time.

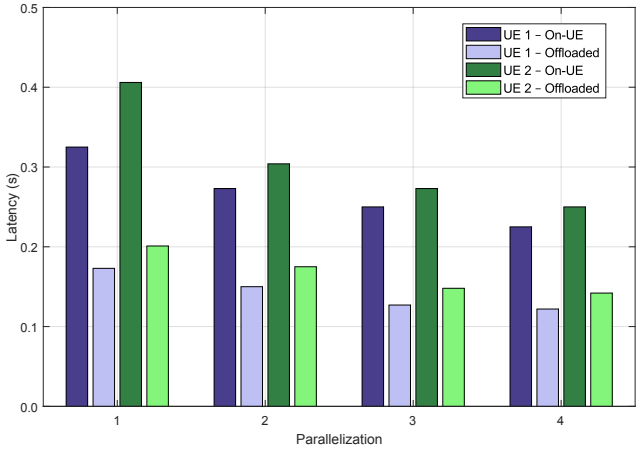


Fig. 6: Impact of parallelization on latency; parallelization is achieved by CPU cores for on-UE execution or by VMs for offloaded execution.

Figure 7 depicts how parallelization influences the energy consumption. For UE 1, adding more threads one by one decreases the energy consumption – 2 threads by 9 %, 3 threads by 13 %, and 4 threads by 6 % compared to no parallelization, 2 threads, and 3 threads, respectively. For UE 2, the pattern of reduced energy consumption for more threads remains the same – 2 threads by 10 %, 3 threads by 12 %, and 4 threads by 8 % compared to no parallelization, 2 threads, and 3 threads, respectively. There are actually two complementary patterns that influence the overall energy consumption – while each working CPU core increases the consumption, the decreased latency decreases it. The combination of these patterns explains why the energy consumption is slightly reduced. For offloading parallelization, the energy consumption of 250 mJ is constant for both UE types, no matter how many VMs are running. The energy is actually spent by the UE on transferring data between itself and the master CeSC, while the CPU is idle when waiting for the results. Since the volume of data is the same for

various numbers of VMs, the energy consumption remains constant. The amount of consumed energy actually depends on signal strength and network throughput.

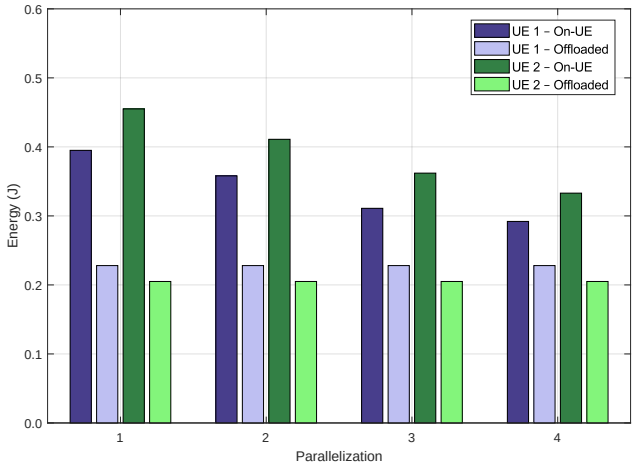


Fig. 7: Impact of parallelization on energy consumption; parallelization is achieved by CPU cores for on-UE execution or by VMs for offloaded execution.

6. Conclusions

The architectures of SCC and the underlying Offloading Framework are introduced in this paper and their key principles are explained. Offloading of computations from the UE to the MEC can be beneficial for computation-intensive apps that require low latency, such as Virtual or Augmented Reality, e.g. Percipio app. Percipio conducts intensive computations in order to discover places visible in the user's field of vision. Since MEC is not commercially available, software emulation is performed on general-purpose hardware. Contemporary deployments of MEC are limited to radio access while omitting computation faculties. The Offloading Framework can be used for real Android apps without any further requirements on the UE-side. The future apps can be easily adapted to support computation offloading while the framework handles low-level communication with the MEC. Such scenario can enable future business models for mobile carriers by offering MEC's spare computation capacity to third parties. Communication overhead in such case is bound to be very small due to the proximity of MEC to users, hence providing an edge to apps such as games, Augmented and Virtual Reality.

Offloading of Percipio's computations from the UE to the SCC provides significant reduction (up to 47 %) of latency. At the same time, energy consumption can be decreased by up to 56 %. It is important to mention that the actual latency and energy savings depend on the scale of computations, effectiveness of data transfer from the UE to the SCC and vice versa, and CPU/chipset of the UE. The data transfer is the main element of offloading overhead, since an app's internal state has to be converted to low-level data and sent to the SCC. Until a reply arrives back to the UE, its CPU is actually idle, and therefore most of energy is spent during radio transmission. Another CPU-heavy app running on a different UE may not yield the same results. The general criterion of suitability of a particular app for offloading depends on the comparison between the computation and offloading overheads. Low-complexity computations have lower latency and energy consumption than offloading overhead, and therefore offloading does not bring any savings in such case.

Acknowledgment

This paper is supported by the CTU research grant No. SGS16/156/OHK3/2T/13.

References

[1] DOLEZAL, J. and L. KENCL. Improving QoE of SIP-based Automated Voice Interaction in Mobile Networks. In: *8th International Conference on Network and Service Management and Workshop on Systems Virtualiztion Management*. Las Vegas: IEEE, 2012, pp. 329–335. ISBN 978-1-4673-3134-0.

[2] LOBILLO, F., Z. BECVAR, M. A. PUENTE, P. MACH, F. LO PRESTI, F. GAMBETTI, M. GOLDHAMER, J. VIDAL, A. K. WIDIAWAN and E. CALVANESSE. An architecture for mobile computation offloading on cloud-enabled LTE small cells. In: *Wireless Communications and Networking Conference Workshops*. Istanbul: IEEE, 2014, pp. 1–6. ISBN 978-1-4799-3086-9. DOI: 10.1109/WCNCW.2014.6934851.

[3] PUENTE, M. A., Z. BECVAR, M. ROHLIK, F. LOBILLO and E. C. STRINATI. A Seamless Integration of Computationally-Enhanced Base Stations into Mobile Networks towards 5G. In: *81st Vehicular Technology Conference*. Glasgow: IEEE, 2015, pp. 1–5. ISBN 978-1-4799-8088-8. DOI: 10.1109/VTCSpring.2015.7145645.

[4] MUNOZ, O., A. PASCUAL-ISERTE and J. VIDAL. Optimization of Radio and Computational Resources for Energy Efficiency in Latency-Constrained Application Offloading. *IEEE Transactions on Vehicular Technology*. 2015, vol. 64, iss. 10, pp. 4738–4755. ISSN 0018-9545. DOI: 10.1109/TVT.2014.2372852.

[5] BECVAR, Z., J. PLACHY and P. MACH. Path selection using handover in mobile networks with cloud-enabled small cells. In: *25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication*. Washington: IEEE, 2014, pp. 1480–1485. ISBN 978-1-4799-4912-0. DOI: 10.1109/PIMRC.2014.7136402.

[6] ETSI White Paper No. 11. *Mobile Edge Computing: A key technology towards 5G*. London: The European Telecommunications Standards Institute (ETSI), 2015.

[7] ETSI GS MEC 003 V1.1.1. *Mobile Edge Computing (MEC); Framework and Reference Architecture*. Sophia Antipolis: The European Telecommunications Standards Institute (ETSI), 2016.

[8] ANDREWS, J. G., S. BUZZI, W. CHOI, S. V. HANLY, A. LOZANO, A. C. K. SOONG and J. C. Zhang. What Will 5G Be?. *IEEE Journal on Selected Areas in Communications*. 2014, vol. 32, iss. 6, pp. 1065–1082. ISSN 0733-8716. DOI: 10.1109/JSAC.2014.2328098.

[9] CERAMI, E. *Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL*. 1st ed. Sebastopol: O'Reilly Media, Inc., 2002. ISBN 0-596-00224-4.

[10] LIU, F., P. SHU, H. JIN, L. DING, J. YU, D. NIU and B. LI. Gearing Resource-Poor Mobile Devices with Powerful Clouds: Architectures, Challenges, and Applications. *IEEE Wireless Communications*. 2013, vol. 20, iss. 3, pp. 14–22. ISSN 1536-1284. DOI: 10.1109/MWC.2013.6549279.

[11] CHEN, X. Decentralized Computation Offloading Game for Mobile Cloud Computing. *IEEE Transactions on Parallel and Distributed Systems*. 2015, vol. 26, iss. 4, pp. 974–983. ISSN 1045-9219. DOI: 10.1109/TPDS.2014.2316834.

[12] CUERVO, E., A. BALASUBRAMANIAN, D. CHO, A. WOLMAN, S. SAROIU, R. CHANDRA and P. BAHL. MAUI: Making Smartphones Last Longer with Code Offload. In: *8th International Conference on Mobile Systems, Applications, and Services*. San Francisco: ACM, 2010, pp. 49–62. ISBN 978-1-60558-985-5. DOI: 10.1145/1814433.1814441.

[13] GENG, Y., W. HU, Y. YANG, W. GAO and G. CAO. Energy-Efficient Computation Offloading in Cellular Networks. In: *23rd International Conference on Network Protocols*. San Francisco: IEEE, 2015, pp. 145–155. ISBN 978-1-4673-8295-3. DOI: 10.1109/ICNP.2015.20.

[14] BARBERA, M. V., S. KOSTA, A. MEI and J. STEFA. To Offload or Not to Offload? The Bandwidth and Energy Costs of Mobile Cloud Computing. In: *2013 Proceedings IEEE INFOCOM*. Turin: IEEE, 2013, pp. 1285–1293. ISBN 978-1-4673-5944-3. DOI: 10.1109/INFCOM.2013.6566921.

[15] KOSTA, S., 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*. Orlando: IEEE, 2012, pp. 945–953. ISBN 978-1-4673-0773-4. DOI: 10.1109/INFCOM.2012.6195845.

[16] BARBAROSSA, S., S. SARDELLITTI and P. DI LORENZO. Communicating While Computing: Distributed mobile cloud computing over 5G heterogeneous networks. *IEEE Signal Processing Magazine*. 2014, vol. 31, iss. 6, pp. 45–55. ISSN 1053-5888. DOI: 10.1109/MSP.2014.2334709.

[17] LIU, J., T. ZHAO, S. ZHOU, Y. CHENG and Z. NIU. CONCERT: A Cloud-Based Architecture for Next-Generation Cellular Systems. *IEEE Wireless Communications*. 2014, vol. 21, iss. 6, pp. 14–22. ISSN 1536-1284. DOI: 10.1109/MWC.2014.7000967.

[18] TAORI, R., Y. B. CHANG, H. J. KANG, S. K. BAEK, Y. M. SON and J. S. PARK. Cloud Cell: Paving the way for Edgeless Networks. In: *Global Communications Conference*. Atlanta: IEEE, 2013, pp. 3546–3552. ISBN 978-1-4799-1353-4. DOI: 10.1109/GLOCOM.2013.6831623.

[19] CHEN, X., L. JIAO, W. LI and X. FU. Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing. *IEEE/ACM Transactions on Networking*. 2016, vol. 24, iss. 5, pp. 2795–2808. ISSN 1063-6692. DOI: 10.1109/TNET.2015.2487344.

[20] HABAK, K., M. AMMAR, K. A. HARRAS and E. ZEGURA. Femto Clouds: Leveraging Mobile Devices to Provide Cloud Service at the Edge. In: *8th International Conference on Cloud Computing*. New York: IEEE, 2015, pp. 9–16. ISBN 978-1-4673-7287-9. DOI: 10.1109/CLOUD.2015.12.

[21] Oracle Corporation. Trail: The Reflection API. In: *Oracle: The Java™ Tutorials* [online]. 2019. Available at: `https://docs.oracle.com/javase/tutorial/reflect/`.

[22] Oracle Corporation. Lesson: Annotations. In: *Oracle: The Java™ Tutorials* [online]. 2019. Available at: `https://docs.oracle.com/javase/tutorial/java/annotations/`.

[23] Google Inc. Android | The platform pushing what's possible. In: *Android* [online]. 2019. Available at: https://www.android.com.

[24] Xen Project. In: *The Linux Foundation Projects: Xen Project* [online]. 2019. Available at: https://www.xenproject.org.

[25] DOLEZAL, J. and Z. BECVAR. Percipio. In: *Google Play* [online]. 2017. Available at: http://play.google.com/store/apps/details?id=com.zeevoy.percipio.

[26] LYU, X., H. TIAN, L. JIANG, A. VINEL, S. MAHARJAN, S. GJESSING and Y. ZHANG. Selective Offloading in Mobile Edge Computing for the Green Internet of Things. *IEEE Network*. 2018, vol. 32, iss. 1, pp. 54–60. ISSN 0890-8044. DOI: 10.1109/MNET.2018.1700101.

[27] CHEN, M. and Y. HAO. Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network. *IEEE Journal on Selected Areas in Communications*. 2018, vol. 36, iss. 3, pp. 587–597. ISSN 0733-8716. DOI: 10.1109/JSAC.2018.2815360.

[28] WANG, F., J. XU, X. WANG and S. CUI. Joint Offloading and Computing Optimization in Wireless Powered Mobile-Edge Computing Systems. *IEEE Transactions on Wireless Communications*. 2018, vol. 17, iss. 3, pp. 1784–1797. ISSN 1536-1276. DOI: 10.1109/TWC.2017.2785305.

[29] Small Cell Forum. Small cells market status report December 2017. In: *Small Cell Forum Releases* [online]. 2017. Available at: https://scf.io/en/documents/050_-_Small_cells_market_status_report_December_2017.php.

[30] Google. Google Cloud CDN – Low Latency Content Delivery. In: *Cloud CDN* [online]. 2019. Available at: https://cloud.google.com/cdn/.

[31] Netflix. Netflix | Open Connect. In: *Netflix | Open Connect* [online]. 2019. Available at: https://openconnect.netflix.com/en/.

About Authors

Jakub DOLEZAL was born in Prague, Czech Republic. He received his M.Sc. from Czech Technical University in Prague in 2010. His research interests include computation offloading and Augmented Reality.

Tomas ZEMAN was born in Prague, Czech Republic. He received his M.Sc. from Czech Technical University in Prague in 1989 and Ph.D. in 2001. His research interests include data communication and networking.