Hybrid Mobile Edge Computing: Unleashing the Full Potential of Edge Computing in Mobile Device Use Cases

Andreas Reiter*, Bernd Prünster[†] and Thomas Zefferer[‡]
*A-SIT, Secure Information Technology Center
Graz, Austria
E-mail: andreas.reiter@a-sit.at
[†]Institute of Applied Information Processing and Communications (IAIK)
Graz University of Technology, Austria
E-mail: bernd.pruenster@iaik.tugraz.at
[‡]A-SIT Plus GmbH
Vienna, Austria
E-mail: thomas.zefferer@a-sit.at

***Abstract*—Many different technologies fostering and supporting distributed and decentralized computing scenarios emerged recently. Edge computing provides the necessary on-demand computing power for Internet-of-Things (IoT) devices where it is needed. Computing power is moved closer to the consumer, with the effect of reducing latency and increasing fail-safety due to absent centralized structures. This is an enabler for applications requiring high-bandwidth uplinks and low latencies to computing units. In this paper, a new use case for edge computing is identified. Mobile devices can overcome their battery limitations and performance constraints by dynamically using the edge-computing-provided computational power. We call this new technology Hybrid Mobile Edge Computing. We present a general architecture and framework, which targets the mobile device use case of hybrid mobile edge computing, not only considering the improvement of performance and energy consumption, but also providing means to protect user privacy, sensitive data and computations. The achieved results are backed by the results of our analysis, targeting the energy saving potentials and possible performance improvements.**

***Keywords*-Edge Computing, Fog Computing, Mobile Cloud Computing, XACML, Mobile Applications**

I. Motivation and Problem Description

Internet-of-Things (IoT) refers to the interconnection of everyday objects and sensors. The novelty in IoT is the creation of applications that make use of collected data in a new and innovative way. The prime example of IoT, which involves various sensors and local as well as global data analytics, is the use case of connected cars and car-driver assistance systems. Cars collect information about the traffic situation, the road, the weather and can display useful information to the driver. Furthermore, these data might be processed together with the data of other cars to calculate an accurate estimation of the car's arrival time, or influence and optimize the traffic light system.

This is just one example among countless others, but it already shows the need for external processing power. The current state-of-the-art solution is to process these datasets in cloud computing centers, requiring to transfer large amounts of data back and forth. This severely impacts the latency of the system which is a critical aspect for the connected cars use case. Tomanek et al. [1] have been measuring latencies to cloud services around the world for three years, resulting in round-trip-times starting from 25ms, but with most of the values above 100ms, which is too high for interactive applications requiring instant responses. To circumvent this issue, the concept of edge computing [2] was invented. With edge computing, computing resources are brought in the proximity of the users without relying on Internet connectivity. Edge- or fog computing is not a replacement for the cloud computing paradigm, but adds another layer of computational power in close proximity to the user. According to Luan et al. [3], cloud computing differentiates from edge computing in various ways: (a) Compared to edge computing, cloud computing can be seen as a centralized structure, whereas edge computing is massivey distributed, but with limited processing power. (b) The latency of end-user devices is driven by requiring multiple hops to reach the final destination host and inherently depends on Internet service providers and core Internet infrastructures. With edge computing these risks and limitations are mitigated. Local communication technologies can be used, providing low latencies.

The previous discussion was focussed on traditional IoT devices like sensors or systems in cars. We think that the described properties of edge computing, like their immanent low latency, are also important for other device classes, like smartphones and tablets. Coughlin [4] analysed current energy sources for mobile devices and concluded that in contrast to semiconductor's price-performance ratio, battery performance and price does not follow Moore's law. His estimations state that battery-energy density has been doubling every ten years. The number of transistors per chip has been doubling every 18 months. This results in a clear gap between innovative, pervasive applications with high energy demands and the available battery technologies.

Our solution to this problem tackles this issue from another direction. Instead of increasing battery capacity, we aim at reducing the overall energy consumption on the mobile device. Corral et al. [5] analysed the energy consumption of Android devices on component level. The results show that on all analysed devices the CPU is amongst the top three energy consumers.

Contribution: We propose a new computing model, Hybrid Mobile Edge Computing (HMEC), which makes use of edge computing-provided computing units and extends this model to suit the needs of interactive mobile applications. This enables an interactive and flexible usage of proximate and distant resources, also considering application offloading techniques, interoperability between different operation environments, discovery of available computing units and maintaining the user's privacy and data security.

II. Gap Analysis

Most of the existing solutions to application offloading are focused on and tailored to the cloud computing domain. In this chapter an analysis of existing systems, focussing on their applicability for edge computing use cases, is performed. We avoid to provide listings of existing frameworks, which is already available from Abolfazli et al. [6][7] and Reiter and Zefferer [8]. This chapter focuses on finding an abstraction which maps on the existing frameworks. First we identify current offloading approaches, then derive an abstract common architecture, and finally analyze the gaps in the introduced solutions regarding their applicability in edge computing use cases.

A. Current Offloading Approaches

The emerging field of Mobile Cloud Computing (MCC) already exploits the fact that the CPU is among the top energy consumers, by offloading computationally intensive tasks. The MCC approach basically makes use of two distinct offloading approaches: Using *Virtual-Machine-based offloading*, a runtime compatible to the user's device is provided in the cloud where unchanged mobile applications are executed. Offloading frameworks dynamically or statically identify computationally intensive tasks and migrate the application and all its associated data to the runtime environment in the cloud. Once the task is completed, the whole execution is migrated back to the mobile device. Without going into the details of the frameworks, this model is used by CloneCloud [9], ThinkAir [10] and COMET [11]. All of the mentioned frameworks are based on Android and Java. CloneCloud and ThinkAir provide a complete device clone in the cloud, whereas COMET provides a more lightweight Dalvik VM clone. Depending on the concrete implementation, this approach is capable of operating on unchanged applications, without having the source code available.

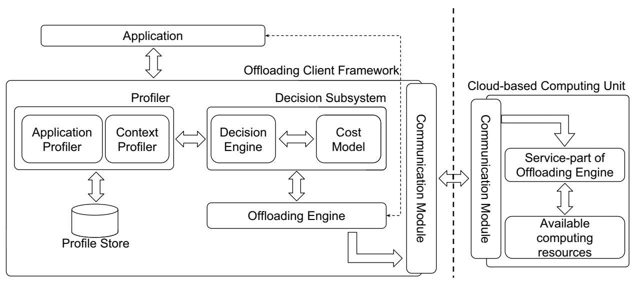

Figure 1: Abstracted architecture of currently available frameworks

The *Method-based offloading* approach is, compared to the virtual machine-based approach, more lightweight and offers more flexibility. On the other hand, it requires more effort in identifying the tasks considered for offloading and its associated data. In contrast to the virtual machine-based approach, it is more fine-grained and works on the function level. This model is used for example by the MAUI [12] framework, which enables offloading of .NET applications on mobile devices.

B. Abstracted Architecture

Scrutinizing these approaches and always cross-checking with the existing frameworks results in the abstracted architecture as illustrated in Figure 1. The architecture does not cover all peculiarities of the frameworks, but covers the overall functionalities of existing MCC approaches. The different levels of details on the client and server side clearly show that the focus of most implementations lies on the client-side and aspects like *how* to identify tasks of an application suitable for offloading and *how* to identify the required information required by the task.

Generally, offloading frameworks make a distinction between four components. The *Profiler* provides performance values for tasks in different contexts. The application profiler provides runtime estimations of tasks, whereas the context profiler provides performance values of the current execution context, like available bandwidth or latency to the remote system. Profiles have a store attached where profiling results can be stored for later reference.

The *Decision Subsystem* involves a decision engine and a cost model, with the goal of determining if it is beneficial to execute a task remotely. To come to this conclusion the decision engine gathers profile values from the profiler and applies them to the specified cost model. Cost models can put their focus on different aspects like increasing performance, minimizing energy consumption, or a well-balanced combination of both approaches.

The *Offloading Engine* performs the actual offloading operation on a technical level, including necessary synchronization of application state, migration of the application execution and reintegration of the results.

The *Communication Module* acts as the key component in the communication with the remote side in terms of establishing the communication link, and performing the communication required for the offloading operation.

C. Analysis

HMEC is a novel concept, with a clear relation to MCC. MCC puts its focus on cloud computing, with different properties, advantages and disadvantages. In the cloud computing domain, distances to users are only considered secondary. Often, MCC frameworks statically bind application instances to specific virtual machines or computing units in the cloud, without providing dynamic assignment. Mobile devices are not always connected through low-latency Wifi connections, most of the time they need to use the Universal Mobile Telecommunication System (UMTS) or Long Term Evolution (LTE) connections. A study from Horsmanheimo et al. [13] reveals that UMTS connections are subject to large variations in round-trip-time (RTT) of above 100ms during peak hours. LTE RTT values are quite stable at the moment, which might be due to the still low penetration of LTE. When combining all the factors, like relatively high latencies to computing units in the cloud [1], and unstable RTT values on the connection, it can be concluded that MCC is not well suited for solving the energy efficiency and performance problems of mobile devices in real-world use cases well.

A large potential of the edge computing paradigm to overcome these issues is projected. It tackles the main problem of variable and high latencies by bringing computing units close to the users. Their computational power is low compared to cloud computing resources, but they can be deployed in a massively distributed way. Due to their distributed deployment, offloading frameworks need to dynamically select one of the available computing units, and need flexible ways to discover and establish connections to these computing units. This was not among the goals of existing frameworks. Furthermore, most existing frameworks use the virtual machine-based offloading paradigm. This requires a cost and compute-intensive setup for the virtual machine, to match the parameters of the mobile device. According to Ha et al. [14] just in time provisioning of virtual machines takes several seconds or even up to minutes. Hence, virtual-machine-based approaches do not seem suitable for HMEC.

Data security and user privacy is another aspect not considered by any of the existing frameworks. Depending on the application and processed data, users need assurance that their security and privacy requirements are respected.

III. A Novel Hybrid Mobile Edge Computing Model

In this chapter the abstract architecture from Section II-B is extended to meet the requirements on dynamic usage of available computing units, flexible discovery, and connection establishment, as identified in Section II-C.

A. Improved Flexible Architecture

Starting from the abstract model an extended model is developed as illustrated in Figure 2. The model is still separated in a client and server model, but the concept of client and server is floating in the context of HMEC. With *server* we do not refer to a central server component, but to a component executed on all computing units. Furthermore, the same mobile device can act as a client in one interaction and as a server in another. The most obvious change is the introduction of a flexible structured network where computing units and mobile devices can log in and communicate with each other. This removes the static cloud connection from previous approaches and already adds flexibility to the system. The duties of this network are to bring together mobile devices, with the intention to offload tasks, and computing units, offering their computational resources. Computing units should be located in close proximity. Still, the network should enable mobile devices to connect to distant computing units.

In edge computing environments, the user might be subject to frequently changing environments (user is moving in a car or on the train). Searching for and connecting to appropriate computing units on-the-fly is time-consuming and can take up to several hundred milliseconds to complete. Therefore, a background task and storage was introduced which always keeps a list of available and ready-to-use computing units in the user's proximity. If one of the computing unit's performance values exceeds thresholds configured by the executed application, the discovery process is restarted and connected computing units are exchanged.

Partitioning and application profilers are scrutinized in other publications [9], [12], [15]. Whereas the decision engine is extended to support multiple cost models on an architectural level. Applications can use a balanced approach of the available cost models to influence the offloading decisions. One strategy could be to balance the performance and energy saving cost model with 70%/30% as long as the battery is above 50% and then gradually start decreasing the influence of the performance cost model to extend battery lifetime.

Another improvement is the introduction of the trust mediator component with the responsibility of establishing trust between the mobile device and the computing unit used for offloading. Without specifying any concrete mechanism, the model foresees a tight integration on the architectural level.

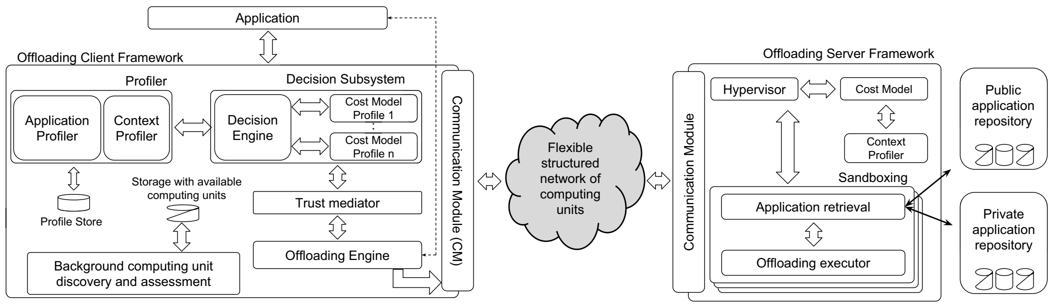

Figure 2: Novel Hybrid Mobile Edge Computing model

The server side was not intensively considered by existing frameworks. Several obstacles to overcome were identified. Basically, the offloading server part is structured as a task scheduler with the following properties:

- Executed tasks are not allowed to influence each other, or access each others procedures and data. This is ensured by a sandboxing mechanism, where each task is executed in a different compartment.
- The hypervisor acts as a control instance for the tasks and can schedule or stop tasks according to a predefined cost model. The cost model ensures that an operator can dedicate all or only a fraction of the available computing resources to the offloading framework. The state of a computing unit is monitored by the context profiler.
- Application retrieval is another important aspect. For a task to execute successfully, assuming a method-based approach, the source code of the task is required. The current state of the application clearly needs to be transmitted from the user's device, whereas the task's source code could also come from application pools to minimize the transmitted data size.

Although the model is open for implementations targeting different approaches, a method-based implementation is recommended due to its flexibility.

B. Integrating Multiple Computing Nodes

Up to this point, the realization of the flexible structured network was left open. It enables mobile devices to easily identify nearby computing units but also allows to make contact with distant units (e.g. units under user's control, or cloud computing units) due to trust issues.

It is assumed that the edge computing infrastructure is driven and deployed by the mobile network operators. They already provide a dense network of mobile network base stations and are therefore capable of providing a tight network of computing units. Basically, different proven interconnection strategies are available: client-server, client-cloud, and decentralized using peer-to-peer networks. The focus of this work clearly lies on breaking up centralized approaches by moving computing units to the edge of the network and increasing fail-safety. This is continued by relying on a decentralized peer-to-peer coordination network with local contact points, but still enabling connections on a global scale. The duties of the peer-to-peer-based coordination network are to act as a single point of contact per base station for computing units offering their resources on one hand, and for mobile devices requiring assistance on the other hand.

A deployment following these strategies is illustrated in Figure 3. Mobile devices always have a connection to one of the nearby base stations. A base station (or cluster of nearby base stations) is deployed side-by-side with a super-node and one or multiple computing units registering at the super-node. The super-node registers in the global-scale peer-to-peer network. Each of the super-nodes and computing units have a unique ID which is announced in the peer-to-peer network (e.g. by using a distributed hash table based peer-to-peer network, and maintaining a table of connected computing units per super-node). The network operator only needs to provide a link to the base station's associated super-node, where the mobile devices can query for available computing units, or query the information for a named computing node. The super-nodes are not involved in the communication between mobile devices and computing units, they only act as a mediator and assist in establishing a direct connection to minimize the overall latency. This approach basically solves two issues:

Discovery of nearby computing units: Other domains put a lot of effort in discovering nearby nodes with low RTTs. Lee et al. [16] developed a geo-location-based RTT predictor for pairs of IP addresses. Their predictor is based on the fact that geographically close IP addresses tend to have a connection requiring less hops than long distance connections, therefore they have a lower RTT. Still, this approach is subject to high levels of noise, and may only be used as a first-level filter. An improved approach, *Htrae*, was introduced by Agarwal and Lorch [17]. They use the results from the geo-location-based approach as a starting point and map the RTT values to coordinates in a virtual space. The coordinates are then

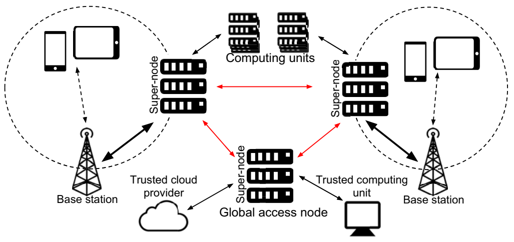

Figure 3: Edge computing deployment

refined based on actual observations. For mobile devices it is an even more complex procedure. RTTs on mobile devices vary depending on the quality of the network signal. Assumptions based on the IP address are not valid in this domain. The proposed approach solves this issue implicitly due to the edge computing paradigm. Mobile devices always have a link to the nearest super-node provided by the mobile network base station. The super node is aware of all directly connected computing nodes, offering minimal RTTs.

Maintaining global connectivity: For specific use cases, improving the performance is not among the top goals, e.g. using specialized hardware like smartcards or other tokens in a feature-based offloading scenario, or only being allowed to offload computations to specific computing units located in corporate-protected data centers. This is enabled by introducing a global access node, where other, unlocalized computing units can register their availability. Due to the peer-to-peer-like organization, all nodes behind one super-node can also communicate with nodes behind any other super-node, like the global access node.

IV. Data Security and Confidentiality Assurance

Data security, and confidentiality was not yet considered by any of the existing offloading frameworks in detail. Up until now, offloading frameworks only had connections to cloud-computing resources, which are trusted by definition. This changes with the application of edge computing. Basically, two interest groups can be identified who rely on the security of processed data and the associated functions: (a) Users, using devices to process their private data, (b) Corporations, processing corporate data.

In both scenarios, sensitive data might be processed which is not allowed to be offloaded to untrusted computing units. The identification of tasks operating on sensitive data is performed by the application developer. This section focuses on the identification and assurance of a computing unit's trust state.

Assessing the trust state of computing units depends on the used computing unit platform, as well as underlying hardware. The suitability of an assessed computing unit depends on the data owner's risk model. Corporations may only allow to process their data on corporate-controlled computing units, or computing units operated by explicitly trusted entities. Private users may also have strict requirements for applications operating on highly sensitive data, but weaker requirements for less sensitive data.

This section provides an overview of the considered methods, but it is not subject of this paper to establish the very details of the considered methods. However, the methods are evaluated regarding their security guarantees, difficulty of adoption and applicability for private and corporate use cases. The results are summarized in Table I. For server-based computing units the Intel Software Guard Extensions (SGX) were considered, as well as certificate-based approaches. For computing units built on top of mobile platforms (especially targeting the Android platform) attestation using Google SafetyNet and using mobile device management (MDM) solutions weres considered.

Google SafetyNet[1] is a tamper detection service provided by Google and included in the Play Services for their Android device landscape. It is constantly running in the background and reports status information to the SafetyNet servers. They analyse the reported data and reason on the overall trust state of the device. Applications running on the device can then query and request verifiable statements about the current state of the device. The weakness of this approach is its purely software-based nature. If an attacker has root-level access on the device they can also fake the SafetyNet reporting process. Therefore, SafetyNet is not suited for assessing computing units where computations on highly sensitive (corporate-) data is offloaded. SafetyNet increases data security for non-rooted devices. As long as Android does not provide a solution to this, security can not be guaranteed.

The *MDM*-based approach is similar to the SafetyNet approach, but focuses on corporate use-cases providing more fine greininess. Still, MDM-based approaches suffer from the same issues as the SafetyNet approach.

Intel SGX[2] is a recent technology which enables the execution of software in trusted environments on untrusted operating systems with support for remote attestation and remote provisioning. This is achieved by extending the CPU instruction set with special instructions enabling the creation of so-called *enclaves*, where applications are executed. This results in a minimal trusted computing base only containing the CPU firmware and hardware, as well as a pre-provisioned remote attestation enclave. SGX applications are not allowed to directly perform IO or execute system calls on the host operating system. Therefore, only SGX-targeted applications can be executed. Baumann et al. [18] provide a solution to this using a thin operating system layer in an enclave and enable the execution of unchanged applications. Due to limitations in the current instruction set, their solution

[1]https://developer.android.com/training/safetynet/index.html

[2]https://software.intel.com/en-us/articles/innovative-technology-for-cpu-based-attestation-and-sealing

Assessment method	**NONE**	**SafetyNet**	**MDM**	**Intel SGX**	**Certificate-based**
Security guarantees	LOW	LOW	LOW	HIGH	HIGH
Difficulty of adoption	LOW	MED	MED	HIGH	MED
Applicability for private use cases	HIGH	HIGH	LOW	HIGH	MED
Applicability for corporate use cases	LOW	LOW	HIGH	HIGH	HIGH

Table I: Comparison of trust assessment methods and their offered security guarantees, difficulty of adoption, and applicability for private and corporate scenarios. Possible values are LOW, MED, and HIGH for low, medium and high.

currently only works in a simulated SGX environment. Basically, creating an Intel SGX based offloading technique seems to be the most promising approach at the moment and currently is constrained by only limited availability of the SGX technology, as well as technical constraints.

The *certificate-based approach* is actually not an attestation method per se, but gives users the chance to establish a trust relationship with computing unit operators beforehand. Cloud computing certifications can help in assessing the trustworthiness of cloud and edge computing operators. Cloud and edge computing units then authenticate using certificates from user-trusted certificate authorities. This approach is a feasible approach for corporations and also for private users, but on a different scale. With this approach private users can add their home computers (and other computing units under their control) to the offloading network, and exclusively use it for tasks operating on highly sensitive data.

V. Implementation

The proof-of-concept implementation realizes all components as illustrated in Figure 3, beside the integration with base stations. Mobile network operators (MNO) are already exploring the use of edge computing using similar deployment scenarios but focusing on different aspects as highlighted in Section I. The integration with MNOs has no impacts on the overall performance and energy efficiency measurements, because MNOs only provide a link to the associated super node. The system contains various critical components and aspects like process isolation, interoperability between end-user devices and computing units, interconnecting multiple nodes, and establishing security between the different nodes. In the following chapter an high-level overview of the implementation is provided. We then pick the two most critical aspects and elaborate them in detail.

A. Big Picture

The super nodes are realized as Java components. They interconnect with other super nodes to form a robust peer-to-peer network. The implementation is based on an existing distributed hash table (DHT) backed peer-to-peer network implementation. Implementation details on the peer-to-peer network are available in Section V-C. Super nodes offer two connection points for other entities:

- A TCP and UDP port is opened for other super nodes to be able to join the peer-to-peer network and communicate with other nodes. These ports are also used for network address translation (NAT) gateway detection and NAT hole punching.
- Furthermore, a Websocket port is opened where mobile devices and computing units can connect to. The service connected to this port offers simple functionalities to acquire computing units for mobile devices and manage the availability and connectivity status of computing units. The separation between mobile device and computing unit is floating. We make a clear distinction between these conceptual components, because different components are executed on these entities. In practice, a mobile device could also act as a computing unit offering its computational power.

Mobile devices do not communicate through the super node with computing units. The super node (and peer-to-peer network) is only used to discover and establish a direct connection.

Mobile devices, conceptually acting as the client components, execute HMEC enabled client applications. For this purpose a client framework was developed realizing the client side architecture from Figure 2. The client framework also focuses on the interoperability issue between multiple devices, architectures, and operating systems. It is based on web technologies using the Dart programming language[3] therefore, it can be executed in all web environments such as web applications, web views for cross-platform applications, and using Dart-based cross-platform frameworks like Flutter[4].

The computing units, conceptually acting as the server components, execute the server application realizing the server part of the architecture from Figure 2. The server application reuses the client framework, but sets it in a different operation mode. The current proof-of-concept implementation does not provide a fully-fledged hypervisor to control and sandbox each offloaded task. Sandboxing was already introduced at multiple different levels, in the context of HMEC it can be considered as future work.

The global access node (GAN) again runs the super node component and seamlessly integrates into the peer-to-peer network. The GAN can be operated globally (for example

[3]https://www.dartlang.org/
[4]https://github.com/flutter/flutter/tree/master/examples/layers/raw

as a cloud service) or can also be operated at the user's site. Computing units connected to one of the global access nodes again execute the server application. Computing units can be configured to only accept offloading request from defined mobile devices. If users attach their private computers to the offloading network, they can pair their devices with the computing unit and establish a trust relationship using a certificate pinning-based approach. For other, base station attached nodes, one of the introduced approaches needs to be used to establish a trust relationship, if required. For the proof-of-concept implementation we explored the SafetyNet and MDM-based approach. The Intel SGX approach is a conceptual approach at the moment, due to technical limitations. During the work on this implementation we determined that the certificate-based approach results in a non-elastic solution, therefore we enhanced this approach by providing trusted certificates dynamically using data security policies, evaluated and provisioned externally.

We think, the two most crucial points in this implementation are the identification of tasks which should be considered for offloading and the assignment and management of associated data security policies, as well as the realization of the peer-to-peer network and its functionalities. Therefore, these aspects are elaborated in the following two chapters.

B. Identifying Tasks Considered for Offloading

The identification of tasks considered for offloading is done by the application developer on method-level granularity. The application developer applies annotations to the source code as shown in Listing 1. The annotation specifies which computing units (untrusted or trusted) may be used to offload this task. The actual technical realization of the offloading operation is performed by the offloading framework. A static source code analyzer processes the attached annotations and adds interception points for the offloading framework. Interception points are realized by wrapping the task in a separate method call where a static offloading framework instance has control of. Calls to the offloading framework instance contain all information required to execute the task. This static instance decides, based on information provided by the profiler and decision subsystem components, if execution should be migrated to the remote host. The current performance cost model bases its decisions on previously recorded data of local and remote executions. The energy saving cost model is more relaxed, in terms of also using computing units with higher latencies. Computing units are considered as long as the estimated execution time, including time required for offloading and integrating the received results, does not exceed the local execution time. For combined cost models, requirements on the energy saving model are increased accordingly.

Listing 1: Annotation to identify tasks considered for offloading with identification of the task sensitivity in the client application

```
@Offload(UNTRUSTED|TRUSTED)
Future doCalculation(p1, p2, {p3: "test"}){
<code>
}
```

This approach is suitable for applications run by private users. With support of the framework, users can specify which assessment types are considered trusted or which particular computing units are trusted. The identified issue with this approach (especially when processing corporate data) is that the classification of methods is defined by the application developer. Corporate data processed by a single task, might have different levels of sensitivity. Therefore, the framework also foresees a runtime evaluation of data sensitivity with integration in corporate policy frameworks. Modifying the annotation as illustrated in Listing 2, allows the framework to outsource the assessment of tasks at runtime. The assessment needs to be performed by a trusted entity, like the corporation owning the data (specified using the *decisionPoint* parameter, including a behaviour in case the decision point is not reachable). Corporations already make use of policy evaluation and enforcement infrastructures today. One popular and widely-spread policy definition language is XACML[5]. Our proposed framework and implementation provides an integration into these frameworks.

Listing 2: Annotation to identify tasks considered for offloading with outsourced identification of the task sensitivity

```
@OffloadWithPolicies(
parametersToSend: ['p1', 'p3'],
cookiesToSend: [],
decisionPoint: 'dp1',
methodName: 'doCalculation',
decisionPointUnreachable: 'local')
Future doCalculation(p1, p2, {p3: "test"}){
<code>
}
```

The data provided by the extended annotation is extracted and a policy decision request is issued. The implementation does not only target XACML, but also other policy environments, therefore, a use-case-tailored policy decision language is introduced. An adapter to the concrete policy language converts the request to the target language and the response back to the intermediate language. During the conversion process, environment parameters can automatically be included: time of request, remote IP address, connection technology used by the device. Based on this information, policies also considering the location of users and their environment can be realized, influencing the offloading process and the selection of computing units.

This already provides a powerful set of features but is still not suited for real world use-cases. The current approach requires that the decision point is always available, which basically neglects the advantages gained by relying on edge computing units. Furthermore, each single method call

[5]http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

requires the framework to issue a policy decision request to the policy infrastructure, resulting in massive overheads. As shown in Section VI, policy evaluation is fast, but in combination with latencies, renders the system unusable.

This motivates to add validity constraints as well as an advanced policy caching mechanism to the system. The validity constraints define how long policy decision responses should be kept in the mobile device's local cache. The advanced policy cache enables to locally store responses and adds acceptance criteria on the policy responses to minimize the necessary policy decision requests, but still gives the policy environment full control of the application execution. Policy creators can define constraints on the input parameters and input data on where re-using the responses is allowed using JSON Predicates[6]. The definition of validity and execution constraints is fully integrated in the policy environment. In XACML they are mapped to XACML obligations and are then converted to the policy-language-neutral representation.

C. Peer-to-Peer Network

The peer-to-peer network serves the purposes of (a) enabling the discovery of nearby computing units and (b) enabling the establishment of connections to distant nodes by using their unique IDs. The system, as illustrated in Figure 3, uses web technologies for the last mile, for connecting the mobile device to the super-node and for directly connecting devices with computing units. The Java-based peer-to-peer framework implementation as executed on the super nodes, provides the capabilities of distributed storing key-value data. Data stored on one node is instantly available on all other nodes. Furthermore, the network supports a direct communication between the nodes. The decision to use a Java-based core network implementation does not influence the overall motivation to maintain interoperability between different platforms, operating systems, or programming languages. End-user devices and computing units still completely rely on web technologies.

For the first requirement, only computing units connected to the same super-node are considered to guarantee minimal latencies. The super-nodes keep track of their locally connected computing units and return one of the available computing units on request. Other approaches where a runtime assessment of latencies takes place is not suited for this dynamic environment. The second requirement can only be fulfilled if a super-node can query the actual location of a particular computing unit. This is realized by using the distributed hash table functionality of the core peer-to-peer network. Once a computing unit connects to a super-node, an entry accessible by the computing unit's unique client ID containing the following data is added to the global hash table: the super-node the computing unit is connected to, the communication technology used by the computing unit (WebRTC, WebSockets), and details on how to reach the node. These approaches enable a very effective and low overhead usage of the available computing units.

VI. Evaluations

In this section our proof-of-concept implementation is evaluated regarding the aspects of energy-saving effects, performance improvement, and impact of the policy environment integration. We consider these as the key improvements introduced by our approaches.

A. Energy-Saving Effects

Performing accurate hardware-based power measurements on smartphones is a complex task and does not allow evaluations on component level. Android automatically captures battery related events which can be visualized using the battery historian tool[7]. This enables to show the battery consumption on a component level. To perform the evaluation, the system was setup on two Amazon EC2 *m4.2xlarge* nodes. To get a realistic scenario, a decent Internet connection was used, resulting in low latencies of 10ms to 15ms. Different mobile devices were used, a recent ZTE Axon 7 with Android 6.0.1 and an older Motorola G2 also with Android 6.0.1. The results of both devices are nearly identical, therefore they are consolidated into a single diagram. For the evaluation, a resource intensive raytracer application was used, which produces software rendered 3D images. It was executed with the local, performance, and energy saving execution profile. The results of this evaluation are shown in Figure 4. The stacked bar chart shows the percentage of total battery consumption per hour. For local only execution, 13.63% battery is consumed per hour. More than half of this consumption can be traced back to the actual performed calculations in the application. As highlighted in the diagram, even using the performance profile, a remarkable energy saving effect was achieved. Using the energy saving profile, the computational intensive application basically does not have a noteworthy energy consumption, but energy consumption of the Wifi network is slightly increased due to increased amount of offloading operations. A fixed factor in all three scenarios is the energy consumption of the screen.

B. Performance Evaluation

The performance evaluation uses the same test setup as illustrated in Section VI-A but with a different test application. We use a web application which applies a Password-based Key Derivation Function2 (PBKDF2) to derive a symmetric encryption key. The number of used iterations impacts the resistance against brute force attacks. With every added iteration the required time for an attacker is increased. Recommendations regarding PBKDF2 state that iterations

[6]https://tools.ietf.org/id/draft-snell-json-test-01.html

[7]https://github.com/google/battery-historian

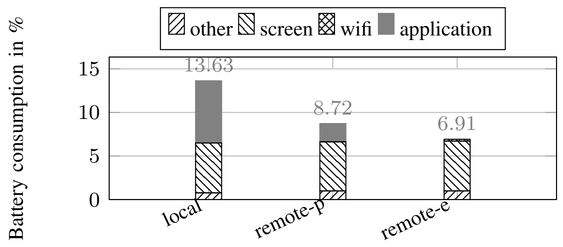

Figure 4: Energy consumption using the performance and energy saving profile stacked by component. Results using the performance profile are shown as *remote-p*, results using the energy saving profile are shown as *remote-e*.

should be increased as high as possible, and a trade-off between security and user experience needs to be found. The results of our offloaded PBKDF2 execution are illustrated in Figure 5. We performed PBKDF2 with 1000, 10000, and 100000 iterations locally and in an offloaded scenario. The evaluations were executed on the described Android devices, as well as in Desktop browsers. We even achieved a significant performance improvement for desktop browsers. The execution environment on the computing units is more powerful than the local browser environments, therefore even desktop computers and notebooks can benefit from this approach. The Android evaluation makes it clear that local execution of more than 1000 PBKDF2 iterations exceeds the feasible security-user experience trade off. Executing 10000 PBKDF2 iterations in an Android web environment locally, already takes about 29 seconds. In an offloaded execution scenario, even 100000 iterations can be performed in 1.2 seconds.

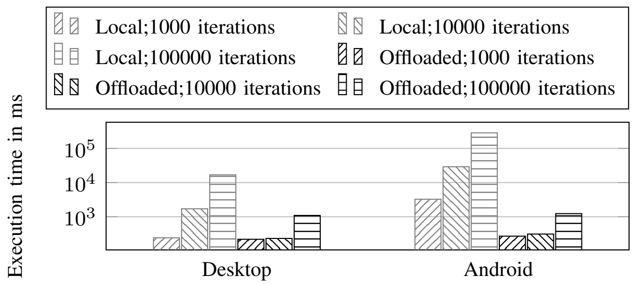

Figure 5: PBKDF2 performance graph on logarithmic scale

C. Policy Evaluation Environment Integration

For policy-enabled systems, the policy evaluation performance plays a major role for the overall application performance. The test setup is composed of an Amazon EC2 *m4.2xlarge* and *c4.2xlarge* instance. The *m4* instance type is a general purpose instance, whereas the *c4* instance type is a data-processing-optimized instance. The evaluation is

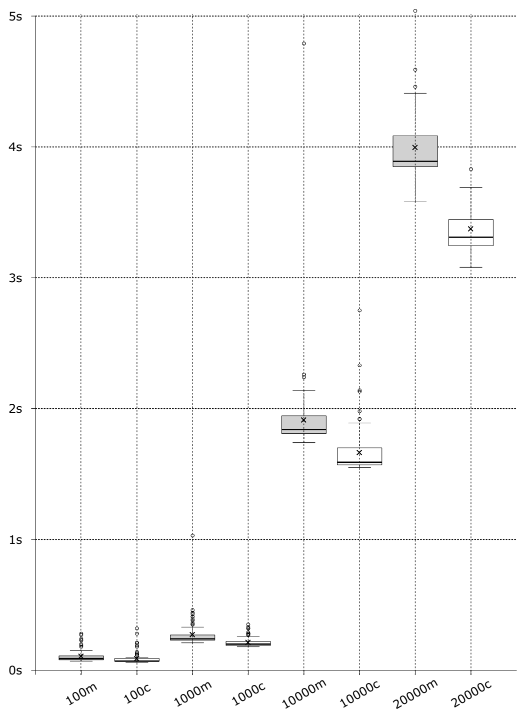

Figure 6: Policy evaluation performance

executed on a modified Apache OpenAZ[8] XACML policy environment, also used as the baseline in several commercial products. The evaluation time for a single policy decision request is recorded for a policy store containing 100, 1000, 10000, and 20000 policies. Real-world use cases deal with tens or hundreds of policies, but also recording corner cases, reveals properties like robustness and scalability of the system. The results of the evaluation are illustrated as a box plot in Figure 6. Results obtained using the *m4* instance are labelled with *m*, results obtained using the *c4* instance with *c*.

At first it can be observed that the used instance type only plays a marginal role for the performance. For the real world use cases the evaluation times are as follows: The mean evaluation time for a policy store with 100 policies is 110ms for m4 instances and 90 ms for c4 instances. For a policy store with 1000 policies, the mean evaluation time is 270ms for m4 instances and 210ms for c4 instances. Without policy decision caching enabled this basically renders applications unusable: Each offloaded task would be delayed by at least additional 90ms. With policy decision caching and enabling advanced validity constraints, only the first run of an offloaded task is delayed. All subsequent executions are not delayed. Therefore, for policy-enabled applications, caching and enabling validity constraints is a vital feature.

[8]https://github.com/apache/incubator-openaz

VII. Conclusion

In this paper we showed the need to enable the usage of edge computing resources beyond the Internet of Things for mobile devices to solve their energy and performance problem. Our proposed solutions show the enormous improvements in energy consumption and performance. Combining applications operating on sensitive data (like the evaluated PBKDF2 application) with policy evaluation infrastructures, improves the offloading approaches but still data owners remain in full control of the offloading destinations.

In the future we plan to extend the policy integration. Concretely we want to add operations like data masking or data anonymization to also enable offloading of sensitive data to untrusted nodes for uncritical operations. Furthermore, we plan to extend the server component by providing a fully-fledged hypervisor with decent sandboxing support.

References

[1] O. Tomanek, P. Mulinka, and L. Kencl, "Multidimensional cloud latency monitoring and evaluation," *Computer Networks*, vol. 107, 2016, ISSN: 1389-1286. DOI: http://dx.doi.org/10.1016/j.comnet.2016.06.011.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pp. 13–16, 2012, ISSN: 978-1-4503-1519-7. DOI: 10.1145/2342509.2342513.

[3] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. We, and L. Sun, "Fog Computing: Focusing on Mobile Users at the Edge," *Eprint arXiv:1502.01815*, pp. 1–11, 2015, ISSN: 10848045. DOI: 10.1016/j.jnca.2015.02.002. arXiv: 1502.01815.

[4] T. Coughlin, "A Moore's Law for Mobile Energy," *IEEE Consumer Electronics Magazine*, vol. 4, no. 1, pp. 74–82, 2015.

[5] L. Corral, A. B. Georgiev, A. Sillitti, and G. Succi, "A method for characterizing energy consumption in Android smartphones," *Proceedings - 2nd International Workshop on Green and Sustainable Software, GREENS 2013*, pp. 38–45, 2013. DOI: 10.1109/GREENS.2013.6606420.

[6] S. Abolfazli, Z. Sanaei, and A. Gani, "Mobile Cloud Computing: A Review on Smartphone Augmentation Approaches," *CoRR*, vol. abs/1205.0, 2012. arXiv: 1205.0451.

[7] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya, "Cloud-Based Augmentation for Mobile Devices: Motivation, Taxonomies, and Open Challenges," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 337–368, 2014, ISSN: 1553-877X. DOI: 10.1109/SURV.2013.070813.00285.

[8] A. Reiter and T. Zefferer, "Paving the Way for Security in Cloud-Based Mobile Augmentation Systems," in *3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (IEEE Mobile Cloud 2015)*, 2015, pp. 89–98.

[9] B. 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*, 2011, pp. 301–314, ISBN: 9781450306348.

[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 *Proceedings IEEE INFOCOM*, 2012, pp. 945–953, ISBN: 978-1-4673-0775-8. DOI: 10.1109/INFCOM.2012.6195845.

[11] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "COMET : Code Offload by Migrating Execution Transparently," pp. 93–106,

[12] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Stefan, R. Chandra, and B. Paramvir, "MAUI : Making Smartphones Last Longer with Code Offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, vol. 17, 2010, pp. 49–62, ISBN: 9781605589855.

[13] S. Horsmanheimo, N. Maskey, and L. Tuomimäki, "Feasibility study of utilizing mobile communications for smart grid applications in urban area," pp. 446–451, 2014.

[14] K. Ha, P. Pillai, W. Richter, Y. Abe, and M. Satyanarayanan, "Just-in-Time Provisioning for Cyber Foraging," pp. 153–166, 2013.

[15] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, "mCloud: A Context-aware Offloading Framework for Heterogeneous Mobile Cloud," vol. 1374, no. c, pp. 1–14, 2015. DOI: 10.1109/TSC.2015.2511002.

[16] Y. Lee, S. Agarwal, C. Butcher, and J. Padhye, "Measurement and estimation of network QoS among peer Xbox 360 game players," *Lecture Notes in Computer Science*, vol. 4979 LNCS, pp. 41–50, 2008, ISSN: 03029743. DOI: 10.1007/978-3-540-79232-1{_}5.

[17] S. Agarwal and J. R. Lorch, "Matchmaking for online games and other latency-sensitive P2P systems," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, p. 315, 2009, ISSN: 01464833. DOI: 10.1145/1594977.1592605.

[18] A. Baumann, M. Peinado, and G. Hunt, "Shielding Applications from an Untrusted Cloud with Haven," *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, pp. 267–283, 2014, ISSN: 0734-2071. DOI: 10.1145/2799647.