MADE: An MEC Supported Platform Independent and Version Agnostic Framework for Mobile Application Deployment

Arpit Tripathi
IIT Hyderabad & IDRBT Hyderabad, India
Email : cs20resch1006@iith.ac.in

Abhishek Thakur
IDRBT Hyderabad, India
Email : abhishekt@idrbt.ac.in

Abhinandan S. Prasad
IIT Ropar, India
Email : abhinandansp@iitrpr.ac.in

Bheemarjuna Reddy Tamma
IIT Hyderabad, India
Email : tbr@iith.ac.in

Subhrendu Chattopadhyay
IDRBT Hyderabad, India
Email : subhrendu@idrbt.ac.in

***Abstract*—Mobile applications are increasingly preferred by business organizations to engage with their customers. However, traditional mobile application deployments face challenges to handle diversity in devices like model release dates, OS support, storage, computing, etc. These challenges are more prevalent in developing economies due to the higher costs of purchase/upgrades, leading to the resale and redistribution of older mobile devices. In addition, the end users may not follow the best practices for cyber security hygiene. This work presents a network-enabled Multi-access Edge Computing based novel framework *MADE*, for the mobile app ecosystem that is platform-independent, release-agnostic, and secure. *MADE* reduces compute and storage requirements on the user device by offloading the mobile application's functionalities to the edge. The characteristics of the *MADE* framework are evaluated through multiple experiments in a 5G test environment across diverse applications. Analysis of top mobile banking applications from the authors' geography demonstrates that *MADE* is platform-independent, release-agnostic, and well-suited for transaction-based applications.**

***Index Terms*—Multi access edge computing, mobile application development, cross platform, resource management, virtual user interfaces**

I. Introduction

Business organizations (Org) often utilize mobile applications to (a) profile user behavior in a fine grained manner, (b) improve the user experiences [1], (c) utilize device-specific features like camera, microphone, etc. However, maintaining a mobile application, particularly deploying a mobile app using traditional [2] approach is a challenging task for organizations. This paper explores the caveats of the traditional approach to mobile app deployment and proposes improvements through an alternative framework.

The traditional approach [2] for mobile app deployment utilizes non-agile iterative development life-cycle where major web apps adopt continuous integration and continuous delivery/continuous deployment (CI/CD) based approach [3]. The rationale behind traditional non-agile deployment of mobile app is due to the device and platform dependency of the apps. Studies reveal that the diversity of hardware platforms and OS versions across available end-user devices (see Figs. 1 and 2) and mobile phones affect the user experience [4]. Additionally, app developers must consider compatibility with older, resource-constrained mobile phones. This is especially important in developing countries like India, where a significant portion of the population lives in low-income areas[1]. Therefore, apps targeted for mass must optimize battery consumption [5], device lifetime [6], etc. Due to these factors every version of app requires thorough testing using simulators and real devices prior to deployment, adding extra overhead to app maintainers. On the other hand, using older hardware and OS versions can compromise the security, which is well studied in the literature alongside the app's usability [7]. These deployment-related challenges result in app incompatibility for users and lost opportunities, especially for financial transaction-providing services. As a result, app maintainers frequently update and patch their apps (see Fig. 3) to support backward compatibility for multiple versions which leads to complex designs and an increase in operational expenditure (OPEX).

Recent advancements in Multi-access Edge Cloud (MEC) technologies [8], which provides platform-as-a-service (PaaS), addresses the app deployment issues by reducing the resource consumption [9] of the end user devices. Additionally, it enables developers to incorporate CI/CD by rapidly converting mobile apps into microservices. However, the migration to MEC eco-system may increase capital expenditure (CAPEX) and introduce integration complexity. This can be avoided by employing a framework that achieves the following design goals:

① **Platform Independence:** The framework should ensure that the services of the mobile apps work seamlessly, irrespec-

[1]https://data.worldbank.org/?locations=XM-XN-XT/

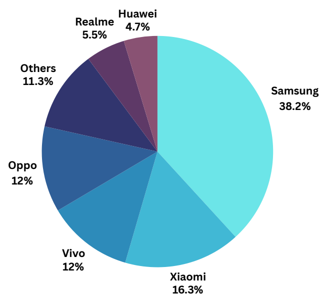


Fig. 1: User %-tage -for Android versions

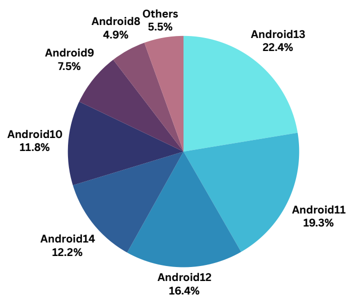


Fig. 2: Market share of Android mobile manufacturers in India

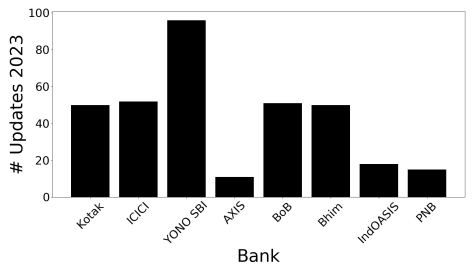


Fig. 3: Mobile app updates in 2023

tive of the end-user devices or OS version, to the maximum extent feasible.

② **Version Agnostic:** The framework should reduce the app version management overhead to achieve backward compatibility.

③ **Minimal Resource Consumption:** The resource consumption of the end user device should be reduced.

④ **Security Hardened:** As the MEC based framework introduces a new attack surface, the app deployment framework should be at least as secure as the *Traditional* app deployment ecosystem.

⑤ **Rapid Deployment and Adaptable:** The framework should allow CI/CD compatibility without extra overhead which can enable rapid deployment of new features. Moreover, the framework should support the existing mobile apps without any code augmentation.

In this work, we propose **M**obile **A**pp **D**eployment framework in an **E**dge enabled system (*MADE*) which addresses these design goals. The major contributions of this work are:

(a) A comprehensive analysis of mobile banking applications and the range of devices and OS versions they support.
(b) Proof-of-concept (POC) implementation using a MEC enabled physical 5G testbed. The POC implementation is useful to assess the feasibility and detailed requirements of the proposed framework.
(c) Exhaustive experimental evaluation to showcase the capabilities of the proposed framework. We empirically justify the potential of the solution to circumvent the issues in the existing eco-system. Our findings indicate that *MADE* reduces resource consumption for the end-user by marginally increasing the resource overhead on MEC. Additionally, it offers a more flexible ecosystem that facilitates hassle-free mobile app deployment.

The rest of the paper is organized as follows. Section II presents the architecture of *MADE*. The implementation challenges and experimental results are described in Section III. Section IV describes how *MADE* addresses the proposed design goals. The related work and conclusions are presented in Section V and Section VI, respectively.

II. Proposed Framework

The architecture of *MADE* comprises four major components: **User Equipment (UE)**s running a Graphical User Interface (GUI)-based app- "GUI Front-end Application" (GFA), **MECs** running the offloaded app features via "Mobile app EmulAtor middle-ware" (MEA), **orchestration for MEC**, and a **cloud service provider** running the "Back End Application" (BEA). The MECs and cloud service provider are managed by the Org Figure 4 shows the architecture of MADE. It has the following main components:

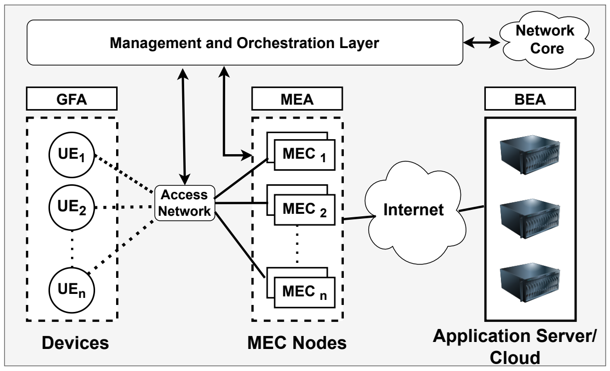


Fig. 4: Proposed MADE Architecture

ⓐ **UE:** UE devices are touch-screen-based mobile devices and use Network Operator's (NOP) services to achieve mobile connectivity. Further, the UEs can run GUI-based client applications to avail the services provided by the Orgs. These devices are also on an unlimited data plan. Therefore, uplink data volume is not a constraint for the user.

ⓑ **MEC:** In the proposed framework, MECs, placed near the edge of the network infrastructure, are capable of providing computing services. The mobile UEs offload their computation tasks to the MEC nodes to reduce resource consumption. The MEC nodes use containers to run the offloaded services.

ⓒ **Cloud Service Provider:** The services provided by the Org are hosted on a secure cloud platform. The choice of cloud (public, private, or hybrid) depends on the Org.

ⓓ **MEC Orchestration:** It manages the MEC nodes [10] to ensure efficient and optimal deployment of apps for various

Orgs. It coordinates with NOP to manage UE mobility and improve user experience.

A. MADE *Framework*

Building upon the previously outlined physical architecture (Fig. 4), *MADE* has three distinct services: a) GFA, b) MEA and c) BEA. GFA acts as a client capable of running a lightweight GUI to interact with and visualize the mobile application hosted in the MEA. In contrast to the typical client-server deployment for mobile applications (apps installed in the UEs), *MADE* deploys the mobile applications in the MEC using a mobile emulator. It provides isolated lightweight sandboxes for security, customization, and manageability. It assumes the presence of end-to-end network connectivity for a transaction-based app ecosystem. The ecosystem is realized using an application flow to access the services provided by the Org.

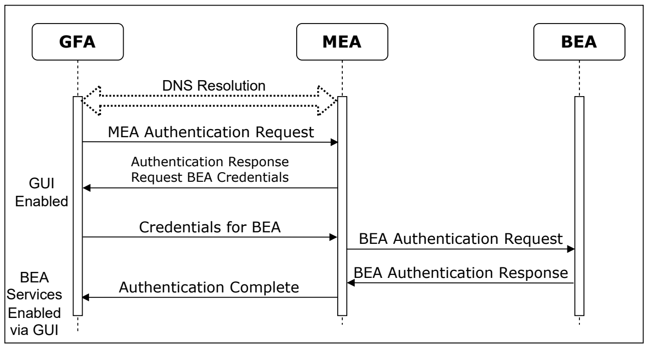


Fig. 5: Sequence diagram for authentication in *MADE*

Fig. 5 illustrates the application flow. Whenever the users want to access the services provided by the Org, they access the GFA, which results in the generation of a `DNS` query. The NOP intercepts the `DNS` request and resolves it with the address of the MEA service. Once the MEA service is discovered, the UE running GFA sends an initial authentication request to the identified MEA. The MEA, based on its internal authentication mechanism, authenticates the UE and requests for BEA credentials. The GFA shares the credentials, which are forwarded to BEA for authentication. As MEA and BEA are both owned by Org, the credentials are as secure as they were in *Traditional* ecosystem. The communication between MEA and BEA remains the same as that of the original Mobile App.

Next, we present a proof-of-concept (PoC) setup to evaluate the *MADE* framework.

III. Proof-of-concept and Evaluation

The experimental evaluation involves five major activities: (i) Creation of a testbed to realize POC for *MADE* along with a custom-built banking application (Custom-App) for controlled experiments, (ii) Evaluating platform independence for multiple devices running diverse Android versions, (iii) Version agnostic behavior of *MADE*, (iv) Overhead evaluation for four popular applications, and (v) Security hardening using a Custom-App.

A. Testbed

1) ***POC hardware setup****:* We have used an in-house 5G testbed to perform the experimentation. This testbed is used only to realize the *MADE* in a cellular deployment physically. The setup includes Open Air Interface (OAI)-based 5G environment[2], along with `ETTUS B210 USRP` as radio unit. We have used 3 Dell Optiplex 5080 workstations, each including an Intel i7-12700 CPU and 16GB of RAM, operating on the Ubuntu 22.04 OS. The first workstation is configured as MEC and 5G Core functionalities using `docker compose` for orchestration. We have deployed MEA using `docker`[3]. To enable GUI, MEA uses `noVNC` server[4] along with Android emulator. The second workstation is configured as a BEA for our Custom-App. We have used 6 physical android devices as UEs and 8 emulated UEs for testing purposes (listed in Table I). The emulated devices are deployed on the third workstation. The physical devices are connected to the cellular network via `ETTUS B210 USRP`. In each UE, we have deployed `Tiger VNC`[5] client app as GFA to interact with the MEA.

2) ***Custom Mobile Banking App****:* We developed a Custom-App for mobile banking to understand back-end server interactions. The mobile app interacts with its customized BEA via `REST` API calls. The BEA is composed of three components(“Authentication" and “Balance Enquiry" as business logic microservices and `MySQL` as database).

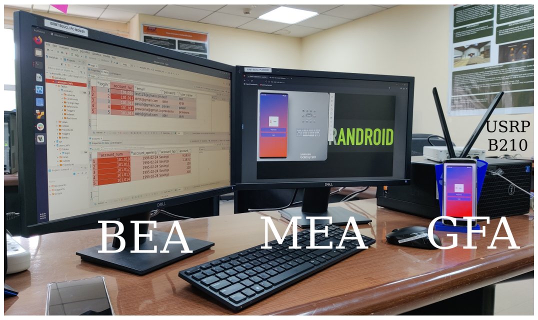


Fig. 6: POC set-up

3) ***UI automation and metrics collection****:* To maintain the homogeneity of metric computation across all devices, we have used `selenium`[6] to simulate user interactions. This maintains experimental consistency and avoids the unpredictability of human operations. During experiments, we encountered challenges in (a) enabling smooth automation of mobile applications using selenium web plug-in, and (b) collecting fine grained statistics. To overcome the first

[2]https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed
[3]https://www.docker.com/
[4]https://novnc.com/info.html
[5]https://tigervnc.org/
[6]https://www.selenium.dev/

challenge, we used `noVNC` server in the UE which allows us to access the UE display within a web browser, and control the app using `selenium` plugin in the browser. To address the second challenge, we have used `adb-shell` scripts to collect the mobile app performance statistics from UEs.

B. *Experiments*

Experiments were conducted over the setup mentioned above. Additional steps for the experiments are also captured as applicable.

*1) **Platform Independence**:* In addition to the pixel devices in Table I, we also included an `iOS` device, running `Tiger VNC` client as GFA for this experiment. We first attempted to install the mobile banking apps in Table I via `adb-shell` on all Android UEs in *Traditional* ecosystem. We observed that the latest releases of apps (mostly supporting Android 10 and above) failed to install on older devices, throwing errors similar to what is captured in Listing 1. However, when deployed with *MADE*, the GFA on the devices successfully interacted with MEA (with installed apps), irrespective of the hardware or the OS of the UE. This achieved the platform independence goal as identified in Section I:①.

```
$ adb install -r state-bank.apk
Performing Streamed Install
adb: failed to install state-bank.apk:
  Failure INSTALL_FAILED_NO_MATCHING_ABIS:
    Failed to extract native libraries,res=-113
```

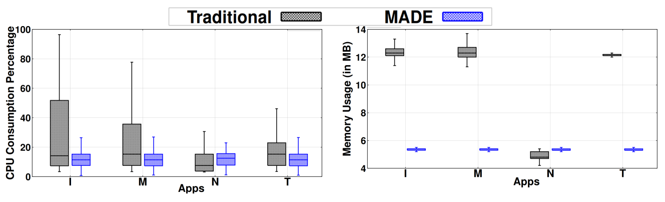
Listing 1: App installation failure

*2) **Version Agnostic**:* The version agnostic design goal, as mentioned in Section I:②, has been tested using twelve prominent mobile banking applications from the Indian banking sector. The initial observations revealed that most Orgs use a `standard` (full-feature) version of their apps. However, we see instances when Orgs maintain `lite` and limited-capability versions of their apps. These initiatives target a wider audience and are mostly designed for older Android versions. Compared to the `standard` app, these apps have limited features. For instance, the Table I shows that “Kotak" maintains two apps, “Kotak 811" app to “open a bank account," "view transaction details," “video verification," etc., whereas the standard “Kotak Mahindra Bank" app, allows additional features like “maintaining credit cards," “fund transfer," “bill payments" etc. Similarly, “State Bank of India" maintains a `lite` version of their “YONO Mobile Banking" app.

For *MADE* we installed `Tiger VNC` as GFA on all the GUI-compatible devices. We also installed the `standard` versions of Mobile apps at MEA. We observed that GFA could successfully access the app installed at the MEA, irrespective of the Android version on UEs.

*3) **Minimal Resource Consumption**:* We installed four popular Android mobile apps to compare the resource consumption and end-to-end behavior in *Traditional* ecosystem with the *MADE* ecosystem. The apps - Instagram (I)[7] for social networking, Maps (M)[8] for navigation, Nettools (N)[9] for network testing and troubleshooting, and Telegram (T)[10] for text messaging, were evaluated. The apps were selected to demonstrate a wide range of possible end-usage. These apps were first tested with the *Traditional* ecosystem. Later, the apps were installed over MEA and tested via GFA. The observations are as follows. CPU consumption and memory usage, were recorded using scripts executed through `adb-shell` for both scenarios. The plots are presented in Fig. 7a and Fig. 7b. Next, we analyze the observations based on the types of activities recorded for each mobile app, providing a clearer understanding of the findings.

For *(I)Instagram*, we automated user events like scrolling and watching video content, commenting on an existing picture, and uploading an image from the local storage. We instrumented the environment so that the same set of feeds was displayed across runs. Being rich in visual content, Instagram consumed more CPU and memory in the *Traditional* ecosystem when compared to *MADE*. The reduction in *MADE* was because complex processing of the feed was done at the MEA, and only a limited set of visible rectangles as framebuffers[11] were processed by GFA.



(a) CPU Consumption (b) Memory Consumption

Fig. 7: Resource utilization for *Traditional* and *MADE*

For *(M) Maps*, we automated three activities: exploring neighborhoods, downloading local maps, and scrolling around the neighborhood. In the *Traditional* ecosystem, the resource consumption increased with every activity; however, with *MADE*, only the current screen, without any dependency on the rest of the map can be viewed in the GFA. Therefore, *MADE* significantly reduced the resource consumption.

In the case of *(N) Nettools*, we automated activities for `ping`, `traceroute`, and network statistics collection. This app is text-based and rarely changes its UI. Therefore, it consumes very little resources while running in *Traditional* ecosystem. The *MADE* ecosystem, which requires screen refresh, increases CPU and memory utilization at UE.

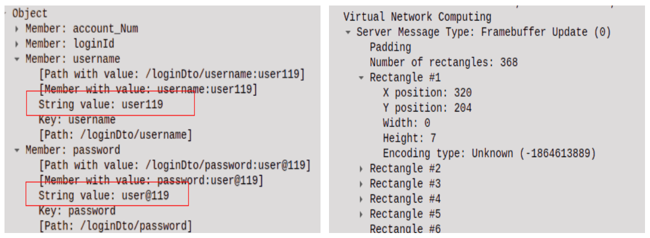
For *(T) Telegram* we automated text entry, sharing a `JPEG` picture and downloading a large `MPEG-4` file (more than 500MB). We observed that resource consumption was more with the *Traditional* ecosystem when compared with *MADE*.

[7]https://www.instagram.com/
[8]https://f-droid.org/en/packages/app.organicmaps/
[9]https://f-droid.org/en/packages/xyz.pisoj.og.nettools/
[10]https://telegram.org/
[11]https://datatracker.ietf.org/doc/html/rfc6143/

	Mobile Devices (Devices- D, Emulators- E)									
Apps	Pixel 2 (E)	Pixel 3(E)	Pixel 3a(E)	Pixel XL(E)	Pixel 5(D)	Pixel 5a(D)	Pixel 5a(D)	Pixel 6(D)	Pixel 6a(E)	Pixel 7(E)
	Android 6	Android 7	Android 8	Android 9	Android 10	Android 11	Android 12	Android 11	Android 12	Android 13
YONO SBI	X	X	X	X	1.23.92	1.23.92	1.23.92	1.23.92	1.23.92	1.23.92
YONO SBI LITE	X	X	5.7.1	5.7.1	5.7.1	5.7.1	5.7.1	5.7.1	5.7.1	5.7.1
BOI	X	X	X	2.0.0	2.0.0	2.0.0	2.0.0	2.0.0	2.0.0	2.0.0
Kotak 811	18.0.1	18.0.1	18.0.1	18.0.1	18.0.1	18.0.1	18.0.1	18.0.1	18.0.1	18.0.1
Kotak Mahindra	X	X	X	X	5.5.4	5.5.4	5.5.4	5.5.4	5.5.4	5.5.4
IDFC	X	X	X	1.31	1.113	1.113	1.113	1.113	1.113	1.113
PNB ONE	X	2.34	2.34	2.34	2.34	2.34	2.34	2.34	2.34	2.34
HDFC	X	X	X	3.2	11.2.3	11.2.3	11.2.3	11.2.3	11.2.3	11.2.3
Axis Mobile	X	X	X	X	X	X	X	9.8	9.8	9.8
VYOM Union Bank	X	X	X	X	X	X	X	8.0.23	8.0.23	8.0.23
BOB	X	X	X	X	3.6.4	3.6.4	3.6.4	3.6.4	3.6.4	3.6.4
IndOASIS	X	X	X	X	X	X	X	2.9.6	2.9.6	2.9.6
IPPB	1.13.2.0	1.13.2.0	1.13.2.0	1.13.2.0	1.13.2.0	1.13.2.0	1.13.2.0	1.13.2.0	1.13.2.0	1.13.2.0

TABLE I: App Compatibility with devices and Android release

To summarize, the *MADE* ecosystem consumed lower resources than the *Traditional* ecosystem for the three apps with interactive multimedia content. *MADE* addressed the minimal resource consumption goal(Section I:③). It ensures users with older and resource-constrained Android smartphones can use various applications. This will also enable the Orgs to expand (or at least retain) their user base without requiring users to upgrade their UEs.



```
Object
  Member: account_Num
  Member: loginId
  Member: username
    [Path with value: /loginDto/username:user119]
    [Member with value: username:user119]
    String value: user119
    Key: username
    [Path: /loginDto/username]
  Member: password
    [Path with value: /loginDto/password:user@119]
    [Member with value: password:user@119]
    String value: user@119
    Key: password
    [Path: /loginDto/password]
```

```
Virtual Network Computing
  Server Message Type: Framebuffer Update (0)
    Padding
    Number of rectangles: 368
    Rectangle #1
      X position: 320
      Y position: 204
      Width: 0
      Height: 7
      Encoding type: Unknown (-1864613889)
    Rectangle #2
    Rectangle #3
    Rectangle #4
    Rectangle #5
    Rectangle #6
```

(a) PCAP for Conventional App (b) PCAP for MADE

Fig. 8: Packet capture analysis for password detection

*4) **Security Hardening**:* We conducted experiments using Custom-App on both *Traditional* and *MADE*. To simulate attacks, we captured packets for communication with the UE. Analysis of the capture for *Traditional* shows that, by filtering `HTTP POST` data, the attacker can gain access to the information propagated by the UE (see Fig. 8a). However, when the app is deployed with *MADE*, the client (GFA) only exchanges framebuffer parameters[12] like coordinates, pointer events, and buffer update messages (see Fig. 8b). Extracting the credential from the packet traces in case of *MADE* is significantly difficult. This experiment helped us to demonstrate that the security hardening goal (as discussed in Section I:④), is achieved.

This experiment utilizes a simple baseline security model without enforcing complicated security controls. However, further security enhancements, such as OTP for multi-factor authentication, hardening of MEC runtime, usage of `HTTPS` etc. can be implemented to improve the baseline. Such additional controls will further improve the security posture for *MADE* compared to *Traditional* deployments.

IV. Discussion

We have experimentally shown that the proposed *MADE* framework addressed ① to ④ in the previous section. This section discusses how *MADE* achieves quick deployment and adaptability design goals ⑤. We also discuss some limitations of *MADE*.

A. *Quick Deployment and Adaptable*

The *MADE* framework allows the Org to run existing apps on MEA. While there is no additional effort for the developer, the end users are masked from the need of constant upgrades on account of new features, security patches etc. The framework ensures adaptability as it can easily accommodate architectural changes for mobile apps.

B. *Overhead of* MADE

There are operational overheads for the Org to setup MEC nodes, maintain user context, manage app releases, and orchestration for MEA. Execution of MEA is an additional cost for the Orgs or users. But, in the case of transaction-based apps, execution-related overhead may be negligible or less significant as the usage patterns are unlikely to peak simultaneously. In [11]–[13], the impact of introducing MEC nodes was studied and evaluated to improve the mobile app performance. Along similar lines, the Orgs can develop a strategy to balance the costs for a number of targeted UEs and the requirements of MEC nodes. Accordingly, the resources can be orchestrated by the NOP.

V. Related Work

In literature, the majority of the works address task offloading via remote display and task splitting in the context of mobile app deployment. These works are mainly motivated by the rapid advancements in the edge-cloud continuum. In [14], an optimization approach was proposed with respect

[12]https://datatracker.ietf.org/doc/html/rfc6143/

to the device's battery consumption, wireless bandwidth, and interaction latency in a cloudlet-based architecture. Similar objectives were addressed in [15] by offloading the mobile app features to the cloud while utilizing a GUI-supported user device. In [16], the efficacy of commonly used remote desktop protocols was analyzed using the experimental setup in the context of the user device and mobile app interaction for the cloud. However, these works did not address the implementation-level complexities of deploying mobile apps on the user equipment. On the other hand, the proposed *MADE* framework analyzes the implementation and application level complexities involved in utilizing such frameworks.

In [17], a dynamic task offloading framework for a two-tier cloud environment was presented, introducing complexity due to task offloading decisions. Similarly, [18] presented an estimation technique for various offloading schemes and evaluated in the context of the refactored mobile application. In [19], the task offloading mechanism was addressed using a workload balance algorithm for a drop computing scenario where tasks were offloaded to the neighbor user devices. This added overhead in mobile app development and deployment, which is avoided by *MADE*. Additionally, *MADE* addresses the security issues of collaborative computation in a drop computing platform.

VI. Conclusions and Future Work

The proposed *MADE* framework offloads the app services to the MEC from the UE by executing a GFA at the UE. We setup a POC and analyzed the performance of various apps in a OAI-based 5G MEC deployment. A minimal orchestration mechanism is implemented to run UE specific application instances from the MEC. Our experiments demonstrated that proposed *MADE* achieves the following goals (i) Platform independence, (ii)Version agnostic, (iii) Minimal resource consumption on UE, and (iv) Security hardening. Through analysis, we showed that the *MADE* framework also reduces the development and maintenance efforts for the Org. This work is an initial contribution towards the mobile app development ecosystem using a cloud-edge continuum.

The following activities are planned for *MADE*: (a) Trade-off analysis between usability and security-based on user surveys, subjective experiments, and by integrating *MADE* with a standard benchmarking tool; (b) Expanding the orchestration framework to seamlessly onboard mobile apps and deploy *MADE* in real networks; (c) Integrate the flow of various sensor-related information from GFA to MEA, including a vulnerability analysis for the entire framework; (d) Analysis of the physical setup realized with 5G testbed.

Acknowledgments

This research is supported through funding by the Dept. of Telecommunications, Govt. of India.

References

[1] T. Oliveira, M. Thomas *et al.*, "Mobile payment: Understanding the determinants of customer adoption and intention to recommend the technology," *Computers in human behavior*, vol. 61, pp. 404–414, 2016.

[2] A. Ahmad, K. Li *et al.*, "An empirical study of investigating mobile applications development challenges," *IEEE Access*, vol. 6, pp. 17 711–17 728, 2018.

[3] T. Tegeler, F. Gossen, and B. Steffen, "A model-driven approach to continuous practices for modern cloud-based web applications," in *Proc. of 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 2019, pp. 1–6.

[4] I. T. Mercado, N. Munaiah, and A. Meneely, "The impact of cross-platform development approaches for mobile applications from the user's perspective," in *Proc. of International Workshop on App Market Analytics*, 2016, pp. 43–49.

[5] H. Jiang, X. Dai *et al.*, "Joint task offloading and resource allocation for energy-constrained mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 7, pp. 4000–4015, 2022.

[6] J. Farman, "Repair and software: updates, obsolescence, and mobile culture's operating systems," *Continent*, vol. 6, no. 1, pp. 20–24, 2017.

[7] D. Smullen, Y. Feng *et al.*, "The best of both worlds: Mitigating trade-offs between accuracy and user burden in capturing mobile app privacy preferences," *Proc. of Privacy Enhancing Technologies*, 2020.

[8] J. Plachy, Z. Becvar *et al.*, "Dynamic allocation of computing and communication resources in multi-access edge computing for mobile users," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 2089–2106, 2021.

[9] J. Liu, J. Ren *et al.*, "Efficient dependent task offloading for multiple applications in mec-cloud system," *IEEE Transactions on Mobile Computing*, vol. 22, no. 4, pp. 2147–2162, 2021.

[10] M. Huang, W. Liu *et al.*, "A cloud–mec collaborative task offloading scheme with service orchestration," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5792–5805, 2019.

[11] R. Fedrizzi, C. E. Costa, and F. Granelli, "A framework to define a measurement-based model of mec nodes in the 5g system," *IEEE Networking Letters*, 2023.

[12] C. Fiandrino, A. B. Pizarro *et al.*, "openleon: An end-to-end emulation platform from the edge data center to the mobile user," *Computer Communications*, vol. 148, pp. 17–26, 2019.

[13] X. Vasilakos, W. Featherstone *et al.*, "Towards zero downtime edge application mobility for ultra-low latency 5g streaming," in *Proc. Of. IEEE Cloud Summit*. IEEE, 2020, pp. 25–32.

[14] M. Satyanarayanan, P. Bahl *et al.*, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[15] P. Simoens, F. De Turck *et al.*, "Remote display solutions for mobile cloud computing," *Computer*, vol. 44, no. 8, pp. 46–53, 2011.

[16] Y. Lin, T. Kämäräinen *et al.*, "Performance evaluation of remote display access for mobile cloud computing," *Computer Communications*, vol. 72, pp. 17–25, 2015.

[17] H. Mazouzi, K. Boussetta, and N. Achir, "Maximizing mobiles energy saving through tasks optimal offloading placement in two-tier cloud: A theoretical and an experimental study," *Computer Communications*, vol. 144, pp. 132–148, 2019.

[18] X. Chen, S. Chen *et al.*, "An adaptive offloading framework for android applications in mobile edge computing," *Science China Information Sciences*, vol. 62, pp. 1–17, 2019.

[19] V. F. Ilie, M. A. Bănică *et al.*, "A computation offloading framework for android devices in a mobile cloud," in *Proc. Of 19th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2023, pp. 98–103.