Ephemeral Apps

Ketan Bhardwaj
Georgia Institute of Technology
North Ave NW
Atlanta, GA, USA
ketanbj@gatech.edu

Ada Gavrilovska
Georgia Institute of Technology
North Ave NW
Atlanta, GA, USA
ada@cc.gatech.edu

Karsten Schwan
Georgia Institute of Technology
North Ave NW
Atlanta, GA, USA
schwan@cc.gatech.edu

ABSTRACT

Despite a tremendous increase in the number of mobile apps, coupled with their popularity with consumers, there exists a wide gap in app availability vs. their use. Recent trends suggest that this gap will further widen in the future. *Ephemeral apps*, proposed in this paper, lower the barrier for end-user app acceptance by removing the app installation step when 'trying out' new apps, without requiring modifications to current apps or any additional programming efforts by app developers. We estimate the resulting reduction in time-to-use for apps to be a factor of 10x by leveraging the emerging 'edge cloud' tier of the Internet.

1. INTRODUCTION

The number of apps available in the Google play store neared 1.5 million by the end of 2014. The fast pace and momentum achieved by the native app eco-system is evident in Figure 1.(b), which shows the number of new apps per month during 2014. While new apps abound, a wide gap exists in app availability and their acceptance. In fact, only 15% of total available native apps are downloaded more than 5,000 times, as shown in Figure 1.(a)[1]. The app discovery gap is likely to widen with the impending explosion of new apps prompted by the wearables, on-demand interactions in Internet-of-Things, situation-specific apps, etc.

The resulting quandary is that while end users prefer native apps over web apps or other browser-based interactions [1, 2], due to their faster performance, seamless access to device features, security, etc., a wide gap will continue to exists in app availability vs. their actual use. Simply put, with so many available apps, how can users search, install, and try them, even if they might benefit from or like them? This is particularly the case for apps with a transient usage model, i.e., those used only in contexts users experience infrequently. For a developers thriving on innovation and quick rollout of new apps, this poses a grave concern, as it creates a high barrier for acceptance of their apps.

[1]Data source: http://www.appbrain.com/stats/

HotMobile '16, February 26-27, 2016, St. Augustine, FL, USA
© 2016 ACM. ISBN 978-1-4503-4145-5/16/02...$15.00
DOI: http://dx.doi.org/10.1145/2873587.2873591

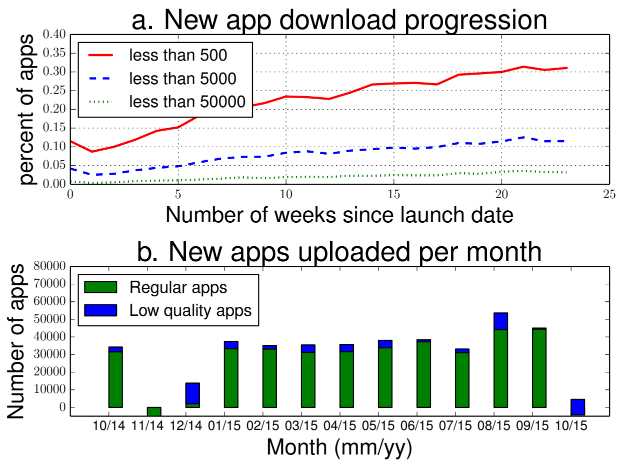

Figure 1: (a) App downloads in weeks after launch with more than 500, 5k, and 50k downloads (b) No. of apps uploaded to the Google Play store in 2014.

App discovery can be seen as two problems: (i) *contextual app selection* which refers to classifying and presenting relevant apps to users based on their current context, history of used apps etc., and (ii) *barriers to app's first use* which refer to effort on the end user's part to install and try out an app for the first time. This work focuses on the latter – reducing the barriers in app's first use by end users. For contextual app selection, we rely on existing mechanisms employed by app stores, e.g., recommendation engines, user reviews, etc., and addressing that problem is not the focus of this paper.

Specifically, we argue for *Ephemeral Apps* as new types of apps that (i) 'pop up' instantaneously on end-user devices in appropriate contexts, (ii) are transient, in that they can completely disappear including the state they might have created from the device once the end-user leaves the context, (iii) do not require any additional effort from app developers, (iv) do not need to change user behaviour of how they use apps today and finally, (iv) offer similar levels of performance, access to device features and security as native apps.

Ephemeral apps make it easy for users to 'try apps out' before deciding to install them permanently, hence reducing barriers in app discovery; or they can use them only within their current contexts, then discard them, expecting to experience an updated app next time in the same context, prompting the use of new available apps by end users and aiding with app discovery. This eliminates not only the need to explicitly install apps that may only be of temporary interest to a user, but also 'de-clutters' end user devices from too many apps and avoids device pollution from their persistent on-device state. It also reduces the need for users to deal with managing their app updates and seeing notifications for seldom used device-resident apps [4].

Achieving those goals, however, requires both infrastructure, and systems software support in Mobile OSes.

In terms of infrastructure, ephemeral apps benefit from the emerging 'edge cloud' tier infrastructure situated beyond the last mile of Internet, including wireless access points and routers, low-power microservers, and similar. Prior work has shown such *eBoxes* – short for 'edge boxes' – to be suitable for accelerating the delivery of mobile app services, for caching and streaming [4, 3], support for personal clouds [7, 5], for app acceleration, offlaod or aggregation via cloudlets [8], etc., and we believe this trend will continue due to increases in eBoxes' capabilities. We make another important observation that most apps turn on the network to access Internet anyways: out of top 5000 popular apps in the Google Play Store, we inspected, 98.9% of apps require Internet permissions. This highlights that using networked nearby resources for ephemeral apps puts minimal overhead on mobile devices.

In terms of system support, Ephemeral apps require the following:

- Mobile OSes - Android
  1. *Ab-initio app streaming support:* so that devices can directly run streamed apps, without requiring their installation,
  2. *Invisible app ephemerality support:* to keep device clean of the apps and the associated state created on device without any app specific modification,
- *eBox based app streaming server* to allow efficient app streaming server capabilities on eBoxes, while maintaining performance at par with native apps.

Our previous work showed how to fetch app components from a remote server and presented an efficient design of app streaming server [4] to manage app updates that can lead to potentially better app loading performance for streaming apps, subject to wireless connectivity. However, it assumes that apps are installed on end client devices before app streaming can occur. We also already explored mechanisms to maintain app cache on eBoxes [3]. In this work, we build on our previous work but are exploring (i) ab-initio app streaming support, i.e., streaming without any prior app and/or configuration on the device, and (ii) invisible app ephemerality, i.e., system level garbage collection, to purge the app along with its created/modified state, targeting Android devices. However, it is important to note that ephemerality is a weaker property than forensic deniability as proposed in [6] and therefore can be supported with much lower overheads on device side. The clear benefit of this approach is the elimination of the efforts involved in searching and installing apps. Stated quantitatively, while search time vs. the time required to identify some ephemeral app is hard to estimate, the time taken to download, install, and start using an app has a quantifiable lower bound. Figure 2 shows the time taken to simply install an app on an Android phone (Nexus 5) from a PC connected via USB cable using Android debug bridge (ADB), measured for 5000 apps - the top 100 in each category dowloaded from the Google Play store. Our preliminary investigation suggests that with ephemeral apps, this delay can be reduced by a factor of 10x, bringing the time required to try an app below 5 seconds.

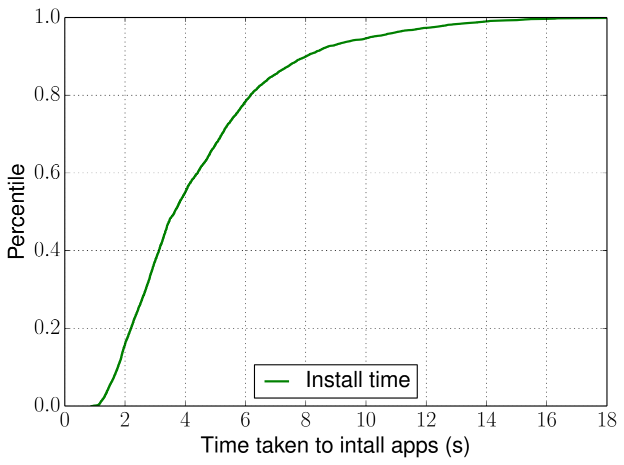

Figure 2: Showing CDF of the lower bounds on time spent during app installation.

2. MOTIVATION

The following factors motivate ephemeral apps:

Convenience: Interaction on-demand
With so many available apps for multiple devices owned by an user, it is very inconvenient for a user to search, install, and use those apps. Ephemeral apps can reduce this inconvenience, by providing apps to users based on their connectivity to an eBox. An important angle of looking at ephemeral apps is in context of the IoT market: it would be cumbersome to search and install apps for each such possible 'thing'.

Performance: Efficiency of native apps
End user preference for native apps is, in part, due to their faster response times or better performance vs. that seen for web apps and their consistent user interfaces mandated by the mobile OSes. Ephemeral apps can keep these characteristics, by making it possible for native apps to be streamed from eBoxes and run on devices with the efficiency of native apps.

Seamless access to device features
End user preference for native apps is also driven by apps being able to seamlessly access device features for better user experience. Ephemeral apps can preserve the benefits of native apps, and address web apps' limitations on access to device features, while maintaining the same levels of security as native apps.

3. GOALS

Ephemeral apps must have the following characteristics from which we derive the design goals:

Instantaneous: Ephemeral apps must be able to appear instantaneous in that a single ephemeral app should be able to appear on end-user devices within 300 ms. This rules out pushing full app installs on devices automatically and/or on-demand as the user goes from one context to other.

Transient: Ephemeral apps and the state that they create on end user's device must completely disappear from the device once the end-user leaves the context. Important to clarify is that the intent is to keep device clear rather than removing the trace of its execution as was the goal of earlier work like [6].

No additional effort from app developers: Making ephemeral apps must not be different from creating native apps. This is important from a practical standpoint of their acceptance by developer community.

No change in user behaviour: Using ephemeral apps must not present a learning curve for end users to use them i.e., users must be able to use them in same way as they use native apps installed on device.

Performance and device features: Ephemeral apps must be able to perform at same level as native apps and have access to device hardware on existing devices under existing permission models.

4. DESIGN CHOICES

We briefly present the available design space to support ephemeral apps in Android:

4.1 System layer for app streaming

Our goal is that functionality required to support ephemeral apps must leverage the existing app model supported on client devices. Modern devices with 'tall' mobile OS stacks are shown in Figure 4, and 'thinner', well packaged mobile apps on top. We posit that since the majority of system level functionality required by apps is abstracted by the app framework APIs, runtime, etc., provided by the mobile OS, it is possible to hide the complexity associated with app streaming and ephemerality completely from app developers, while providing efficient app execution leveraging devices' capabilities. On the eBox side, this also allows to leverage system level information like the order of app components to optimize streaming performance [4].

4.2 Streaming from cloud vs. eBox

Mobile devices are subject to well-understood constraints, such as power consumption. Using remote resources not subject to those constraints can, enhance user experiences on devices. Concerning the use of such remote resources for ephemeral apps, we argue the relative advantage of eBoxes vs. remote data centers in terms of (i) increasingly fast wireless networks, (ii) cheap storage at eBoxes to house app repositories and low access latency due to lower network distance.

4.3 Ephemeral vs. Native, web, thin client apps

Currently, most mobile OSes like Android, iOS etc. support the following app models:

- Native app: specifically designed to run on a device's OS, uses features exposed by platform via APIs and is constrained by OS semantics.
- Web app: the entire app or its parts are downloaded from the web during execution. It can be accessed from any web-capable mobile device without requiring operating system-specific customization.
- Hybrid apps: most native apps utilize web connectivity, and web apps provide offline modes. The resulting form of apps are referred to as hybrid apps.
- Thin client apps: require a remote access app, e.g., VNC client on device while actual execution happens remotely.

We posit that *Ephemeral Apps* can potentially obtain native app user experience in terms of responsiveness, access to device features, security, efficient operation than thin client apps with central management capabilities of web apps by allowing on-demand streaming of the native app components needed for execution, via eBoxes.

5. FEASIBILITY AND CHALLENGES

We are using Android to prototype ephemeral apps. The simplified interaction between the modified Android modules and an eBox-based ephemeral app server are depicted in Figure 4. Presented below are the technical challenges that need to be solved to support the ephemeral apps:

Handling diverse app Anatomy and Execution

A typical Android app is comprised of AndroidManifest.xml (app config file), classes.dex (JAVA classes), .arsc file (binary resource file), assets, res folders (static resources e.g., icons, strings etc.), native libraries, and miscellaneous components (certs, HTML help files etc), referred as app components which are needed during installation and/or execution. In Android, an app executes as an instance of the Android Runtime (ART) in which Android framework libraries are preloaded during fork of the Zygote process, and platform features are exposed via APIs. To make apps ephemeral, these components must be made available on demand, via streaming, in a way that does not hurt app performance. With the current architecture of mobile OSes (specifically, Android), it is a daunting task because each of these app components is handled in a different subsystem, e.g., classes.dex are loaded by the ART runtime, .arsc files are accessed using the asset manager, whereas resource in res or asset folder may be read explicitly. To makes matters worse, these system components can be used in different ways, e.g., app can use either a URI and/or binary buffer mode to load static assets. This also highlights an important point that existing mechanisms like on-demand class loading or JAVA reflection do not suffice for mobile app streaming.

Ensuring app Integrity, Sandboxing and Permissions

App integrity is established by downloading apps only from trusted app stores with self-signed certificates from developers. For ephemeral apps, since there is no installation step, the mobile OS and ephemeral app servers collaboratively need to make available app components and establish their integrity. We propose to use certificate based verification of the ephemeral app server as a valid provider of apps. The certificate may be provided by app stores or any other trusted third party and implement a merkel tree of signatures which can be used to verify individual app components as opposed to full app during installation.

Permissions required by apps are prompted to the user for approval on installation, at which time each app is assigned a UID, GID. These are then used to enforce permissions to access hardware features and inter-app communication using Binder IPC at linux kernel layer. Android application sandbox using Linux's discretionary access control (DAC) and SEAndroid mandatory access Control (MAC) to isolate an app data and code execution from other apps. For ephemeral apps, Android needs to support lazy allocation of UIDs, GIDs and handle app permissions at runtime - a feature introduced in Android M. We also need to define appropriate SEAndroid policies for ephemeral apps to ensure security guarantees during their execution.

Managing App State

Android provides no guarantees to apps about their state and expects them to manage it explicitly via state machine callbacks. This is consistent with our vision of ephemeral apps where we propose to capture their app state and clean it up. However, two types of persistent state are typically associated with an Android app:

System State:

By system state, we refer to the information Android maintains about or on behalf of apps during installation e.g., its package, (uid, gid) pairs, etc. and during execution e.g., web caches, cookies, data journals, per app shared data, app private files etc.

App State:

- *On-device state* refers refers to the state needed for app start or resume, created by an app during execution. e.g., shared document-like content handled by Android on devicesâĂŸ local storage and the content of internal state, like user preferences, handled using the APIs, etc.
- *Remote app state* refers to state explicitly committed to a cloud-based backend, e.g., saved game progress. This is consistent with our ephemeral app vision and requires no special treatment for ephemeral apps.
- *Content based state:* refers to the content used by an app during its execution that is not part of the app-specific content cache. Apps may fetch it from some back end service, their own network-based storage, or some third party, e.g., mobile ads, etc.

Ensuring app performance. Compared to web apps, while both app models require network interactions to fetch the app components, we posit ephemeral apps would perform better because of the number and size of web app components (JavaScript, HTML, CSS, etc.) vs. native app components that need to be fetched, and the additional processing in web apps launching a browser, creating DOM objects, handling dependencies and rendering those on device which would not be required for streamed native app components. Comparing to thin clients, these entail aggressive use of network to fetch screen contents, send user inputs, sync on-device sensors with remote thin client server for every user interaction, while rendering of the screen contents on end user device. In contrast, streamed native app components would require less use of network because of reduced payload to be fetched and the possibility of reuse of fetched app components (classes, UI images, etc.) during an ephemeral app session. This leads us to believe that ephemeral apps can be more responsive and would require less power than other alternatives for their execution.

Integration with app stores: Figure 3 depicts our vision of how ephemeral apps can be made available via collaboration by app stores and eBoxes, i.e., without app developers to buy into something new. We believe this is important to continue to leverage auditing capabilities of app stores that check apps for security before making them available to authenticated end users, and for continuing to support developers using pay per use model for ephemeral apps. However, this doesn't rule out alternate delivery channels where app stores can deliver apps to vendor equipment that they expose as ephemeral apps to end users. An additional interaction between eBoxes and app store is required to provide eBoxes with valid certificates that they can present to end user devices to verify their integrity as trusted app providers.

Maintaining user trust model: Ephemeral apps blur the distinction between web apps and native apps. A subtle point in introducing ephemeral apps in the app ecosystem is that users trust apps downloaded from app stores but the same is not true for web apps. This requires careful consideration of the trust model being used to distinguish end user's interaction with ephemeral apps from web app or native apps. We posit that the app store integration proposed above and the runtime permission model introduced in Android M in combination can be used to provide a workable trust model, where only apps verified by app stores will be provided to eBoxes or vendors that they can make available as ephemeral apps. The end user devices only show ephemeral apps being made available by trusted ephemeral app providers which is verified by certificate presented by an eBox. For apps that are shown in end user devices, users would be prompted for specific permissions as they continue using an ephemeral app, a feature that we developed but is also introduced in Android M.

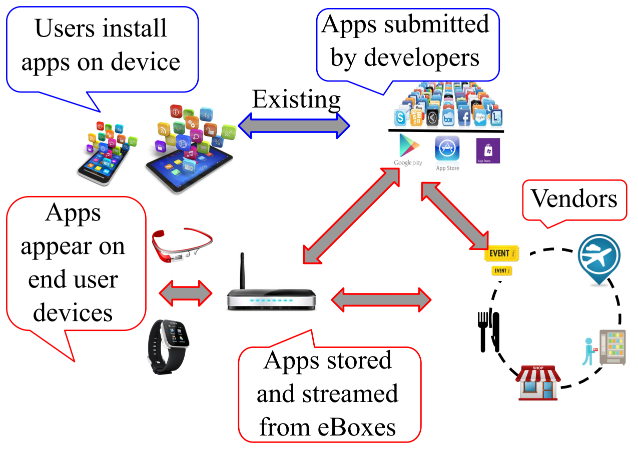
Users install apps on device
Apps submitted by developers
Existing
Vendors
Apps appear on end user devices
Apps stored and streamed from eBoxes

Figure 3: showing an overview of ephemeral app eco-system and its potential integration with existing app stores.

Preserving user privacy: Streaming ephemeral apps raises privacy concerns because with knowledge about which app components are fetched during execution, it is possible to infer what a user is doing with it. To address that, we plan to implement a differential privacy techniques while fetching app component to reduce these risks. We believe it would be an interesting proposition if we can actually leverage these extra requests to fetch somewhat related app components.

6. PROPOSED DESIGN

Ab-initio App streaming: support in Android is designed to work seamlessly for existing apps. It is designed to operate in two phases (i) *Conjuring phase* and (ii) *Execution phase*. The rationale behind the two phase-design of the ephemeral app life cycle is to (i) not require installation of apps on device and (ii) ensure faster app discovery by decoupling presentation of an app from its launch and execution. *Conjuring phase* starts when a user invokes his ephemeral home screen referred to as Stage - a new launcher app on users' device connecting to an eBox. Stage requests for list of available ephemeral apps; on receipt of the response, a certificate based verification is done to establish the integrity

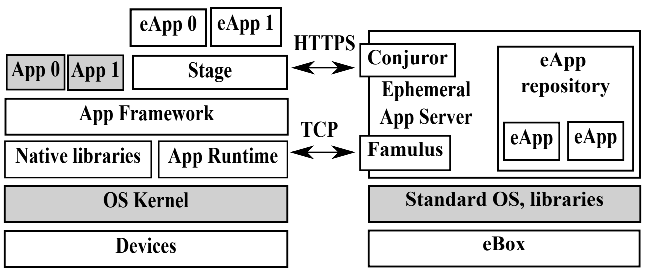

Figure 4: Showing the co-existence of existing and the proposed ephemeral app ecosystem.

of an eBox/app server as a valid app provider; If the user trusts it, Stage starts fetching icons of the available apps to populate the end user's screen/Stage. In addition to the icons, the response includes top level hashes from merkle trees of available ephemeral apps and information needed to launch an app, e.g., activity name or first class to be loaded.
Execution phase starts when a user decides to launch an ephemeral app. It is proposed as a new API in Android app framework which asks the activity Manager to start a new app runtime (ART) instance. Before forking an instance of the app runtime, the activity manager asks the package manager to assign a virtual blanket of permissions and to assign ephemeral uid, gid, that belong to a separate range from existing uid ranges used in Android, and which can later be used to handle permission exceptions to provide runtime permission approvals for ephemeral apps (e.g., if an app requires more than already assigned blanket permissions). To keep app sandboxing guarantees for ephemeral apps while also ensuring availability of storage for caching app components fetched during app streaming, we plan to implement a new SEAndroid security context and associated policy: ephemeral app domain. During execution, app streaming is designed to work seamlessly for existing apps by intercepting calls to the runtime's class loader, native library loader and the app framework's Asset Manager, and redirecting those to a remote (e.g., eBox-based) ephemeral app server via a Streaming Client embedded in the ephemerality manager discussed next. This ensures that even when the apps are not installed on a device, the running apps are offered the same runtime environment as if they are installed. Concerning state created by apps, which normally is stored in app specific folders on the device's local storage, ephemeral apps are assigned new location which also helps in providing system level app ephemerality discussed next.
App ephemerality: To track app's system state without support from apps, we plan to (i) route all ephemeral app component requests through a system wide ephemerality manager and (ii) add hooks in all app relevant framework APIs to capture app private state which can then explicitly logged on-device. Since, ephemerality manager requests the needed app components, it also checks each received app component for its integrity using the ephemeral app's top level hash, thereby ensuring runtime app integrity. Briefly, the ephemerality module must expose a native and JNI interfaces which can be used by the Android app framework APIs and by individual system services to request app components and log app state when apps use those APIs. These logs are the key to enabling system level ephemerality guarantees. Asynchronously saving, discarding and/or syncing of apps and their states can be supported in order to minimize ephemerality-related overheads on the system. The ephmerality module allows processing of the logs under different policies, each of which is derived from end user preference about the app state. For instance, if a user indicates that he wants to discard app state then, on app exit, the ephemerality manager can access the logs to gather app state generated by that app and deletes them followed by removal of the logs from the database. Similarly, if an end user chooses to sync state to an eBox, the Ephmerality Manager sends app state to an eBox before deleting the logs.
In our incomplete prototype, we support on-demand loading of app-specific classes, assets, etc. but requires a stub app (stage app) to be installed. Complete implementation of support for ab-initio app streaming and handling app ephemerality in Android is currently in progress. We discuss the proposed design below:
Stage - a new ephemeral apps launcher app: Stage app communicates with eBox based app streaming server for available apps. Before populating Stage with app icons, it established the integrity of eBox using certificate-based authentication.
App framework support for ephemeral apps
Package Manager: The package manager is modified to provide information to the app framework about available ephemeral apps, for assigning UID, GID in a different range than installed apps, and for creating app specific directories for their use in caching and/or for ephemeral app state at launch time.
Activity Manager: The enhanced activity manager supports the launch of apps in ephemeral mode i.e., launching new activity for ephemeral apps without interfering launch of with installed apps. At launch time, the developer signature attached with the app is checked. Additionally, we plan to implement a checksum verification for the app executable and components between device and app stores before start of actual streaming.
Lazy Permission handler: We added a new permission exception handler to the ActivityThread which is basically linux level thread running the application. When the exception handler is invoked when an ephemeral app incurs a permission that hasn't been assigned to it. This give rise to a permission grant dialog in similar ways the Android framework issues the 'application not responding' (ANR) dialog.
Asset manager: The framework's asset manager is modified to facilitate loading assets like icons, xmls by redirect ephemeral app's asset requests to an eBox vs. its own file system.
Android app runtime - ART
The following components of ART need to be changed to support ephemeral apps:
Ephemeral Class linker: We added a new class in ART that behaves similarly to existing class linker excepts that it searches and streams the requested app-specific classes from an eBox vs. the device's local storage.
Native library loader: Similarly, ART's interaction with the native library loader streams app-specific native libraries from the eBox vs. the device's local storage.
App Ephemerality support in Android
To seamlessly support ephemerality for unmodified android apps, we added a core library libephemeralutils which performs all network operations needed for streaming of app components from the Android app framework and ART run-

time on behalf of ephemeral apps. By doing this, it seamlessly facilitates logging of fetched app components. Additionally, it also provides support - native and JNI APIs - for explicit logging of any state or change in file created by an ephemeral app on phone's file system. We plan to instrument Android APIs that are used by apps to write to phones file system to create a per app log in form of a SQLite database. These logs are then used to clear device of any app state or changes made by an ephemeral app.

eBox-based ephemeral app server
The ephemeral app server consists two subsystems:
Conjuror: an HTTPS server establishing the integrity of an eBox as a valid app provider during connection time; it pushes apps to the end user device, i.e., app metadata and TCP port, which will be used for actual streaming of app components by a client. It is also responsible for maintaining an app repository on a SSD attached to it via USB 3.0 and performs app preprocessing when the apps are added or updated. Preprocessing includes extracting app metadata for app push, streaming app components.
Famulus: a TCP socket server handling on-demand streaming of app components for a single app; it implements the logic to translates app-specific URIs to entries on its own file system generated by conjurer, which contains uncompressed apps conforming to the same directory structure as would be on device with installed app.

7. PRELIMINARY EVALUATION

Experimental Setup: We used a Nexus 5 phone running Open Source Android v5.0.1.3 for device side evaluation. The phone is connected to eBox prototyped using Linksys WRT1900ac 802.11ac Wi-Fi router running OpenWRT. The same router with an attached SSD of 20GB is used to prototype the eBox based app streaming server.
Feasibility: We report that we have been able to run native existing apps as 'Ephemeral apps' via app streaming and app ephemerality on Android without requiring any changes in those apps.
Time to use: Figure 5(iii) shows the lower bounds of time to to start using an app when installing it from app store vs. using ephemeral apps. In addition, Figure 5 shows upto 10x reduction in time to use compared to installation and launch time of top 40 apps which include the top app in each category downloaded from Google Play Store. Also, important to note is that, launching app via app streaming doesn't lead to delay in app launch time and hence, leading us to believe responsiveness is comparable to native installed apps.

8. SUMMARY

We propose 'ephemeral apps' as a solution to address the gap in app availability vs. end-user acceptance, evaluated with a prototype implementation on Android. Future work will complete their implementation and perform additional evaluations with realistic multi-tenant loads.

Acknowledgement

We would like to thank our shepherd, Kaushik Veeraraghavan, for his insights during the preparation of the final version of this paper. This work is partially supported through research grants from Intel, VMware, and NSF CNS1148600.

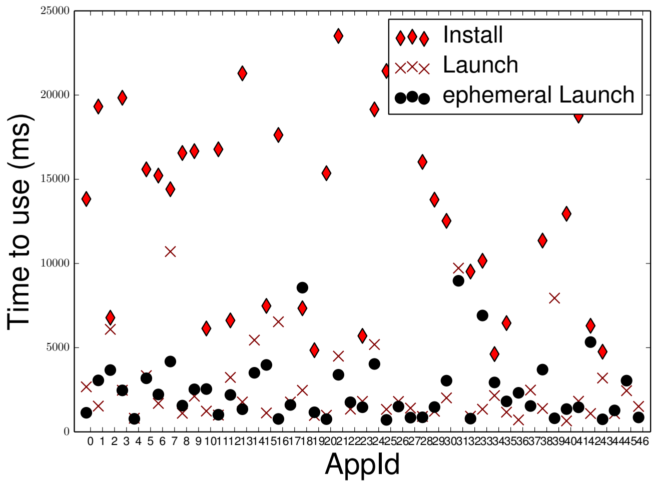

Figure 5: Comparing time to use of ephemeral apps made available from a nearby eBox vs. lower bounds of install and launch time for top 40 apps.

9. REFERENCES

[1] Apps solidify leadership six years into the mobile revolution - Flurry @ http://www.flurry.com/bid/109749/Apps-Solidify-Leadership-Six-Years-into-the-Mobile-Revolution.

[2] Mobile apps overtake pc internet usage in u.s. - cnn money @ http://money.cnn.com/2014/02/28/technology/mobile/mobile-apps-internet/.

[3] K. Bhardwaj, P. Agarwal, A. Gavrilovska, and K. Schwan. Appsachet: Distributed app delivery from the edge cloud. In *7th EAI International Conference on Mobile Computing, Applications and Services*, MobiCASE '15, 2015.

[4] K. Bhardwaj, P. Agarwal, A. Gavrilovska, K. Schwan, and A. A. Appflux: Taming mobile app delivery via streaming. In *2015 Conference on Timely Results in Operating Systems (TRIOS 15)*, Monterey, CA, USA, 2015. USENIX Association.

[5] K. Bhardwaj, S. Sreepathy, A. Gavrilovska, and K. Schwan. Ecc: Edge cloud composites. In *Proceedings of the 2014 2Nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, MOBILECLOUD '14, pages 38–47, Washington, DC, USA, 2014. IEEE Computer Society.

[6] A. M. Dunn, M. Z. Lee, S. Jana, S. Kim, M. Silberstein, Y. Xu, V. Shmatikov, and E. Witchel. Eternal sunshine of the spotless machine: Protecting privacy with ephemeral channels. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 61–75, Hollywood, CA, 2012. USENIX.

[7] M. Jang, K. Schwan, K. Bhardwaj, A. Gavrilovska, and A. Avasthi. Personal clouds: Sharing and integrating networked resources to enhance end user experiences. In *INFOCOM, 2014 Proceedings IEEE*, pages 2220–2228, April 2014.

[8] E. Koukoumidis, D. Lymberopoulos, K. Strauss, J. Liu, and D. Burger. Pocket cloudlets. *ACM SIGPLAN Notices*, 47(4):171, June 2012.