Application-specific Acceleration Framework for Mobile Applications

Byungkwon Choi, Jeongmin Kim, Dongsu Han
KAIST

1 Introduction

Minimizing response times for mobile applications is critical for quality user experience which often impacts the revenue of mobile services. Generalized approaches to accelerated mobile applications (e.g., TCP acceleration, SPDY, compression) are less effective because they do not take account for app-specific behaviors. In contrast, app-specific approaches build proxies tailored to specific applications [5] by leveraging the app-specific protocol behaviors to enable dynamic caching and/or prefetching. With edge cloud services being deployed in mobile network (e.g., mobile edge computing [2]) and network function virtualization, enabling automatic deployment of middlebox services, plenty of opportunities exist in introducing app-specific accelerators. However, the core challenge is that developing an app-specific proxy is time consuming and tedious because it often requires understanding and manual analysis of application-level protocols and interactions. Therefore, only a select number of applications have enjoyed the benefit.

This paper addresses the problem of scaling the number of applications. To this end, we present a framework for mobile app acceleration that leverages automatic protocol analysis to automatically discover opportunities for app acceleration. Our framework takes Android app binary as input and leverages existing binary analysis framework for Android apps to identify the dependency relationships between HTTP requests and responses that applications generate. The framework then automatically finds out when and where to prefetch or perform dynamic caching. This information is passed to developers to generate app-specific proxies.

We present the framework design and a preliminary result that demonstrates its viability. We take a popular application and demonstrate that the resulting proxy dramatically cuts down latency. We envision that this framework will be used to accelerate many mobile applications. As future work, we plan to extend the framework to automatically generate app acceleration proxies by automating the proxy generation.

SIGCOMM '16, August 22–26, 2016, Florianopolis, Brazil
 ISBN 978-1-4503-4193-6/16/08.
DOI: http://dx.doi.org/10.1145/2934872.2959049

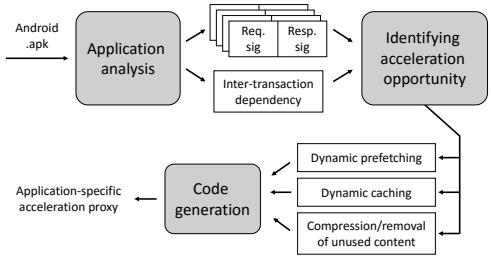

Figure 1: Design overview of the framework

2 Motivation

Mobile apps and services have experienced tremendous success. Developers create server/mobile applications and send them to cloud service providers and open app market respectively to get services instantly deployed. However, in-network services, such as app acceleration proxies, have been used for a small number of popular applications. Two barriers exist in deployment: 1) an infrastructure for automatic deployment has been missing; and 2) their development requires significant effort. We believe recent advances in mobile edge computing and NFV will address the first issue. This paper addresses the second issue. In particular, we would like to create (virtual) network functions for enhancing the user experience of mobile applications.

3 Framework Overview

Figure 1 illustrates main components of our framework.

Application analysis: Our framework uses output of Extractocol [4] which offers comprehensive analysis of Android app protocol behaviors. Extractocol only uses Android app binary as input and outputs regular expression (regex) signatures for each request and response that the app generates. It accurately reconstructs HTTP transactions by pairing a request with its corresponding response. It also infers dependencies between transactions by tracking which fields in a request message come from an earlier response.

Identifying acceleration opportunity: This step identifies accleration opportunities by using the output of Extractocol. For dynamic prefetching, Extractocol gives two kinds of information; what and when to prefetch. Extractocol tells which fields of request come from earlier responses. For requests with fields coming from earlier responses, it will be useful to prefetch responses of the requests because the re-

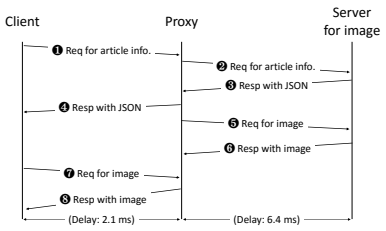

Figure 2: Process of image prefetching

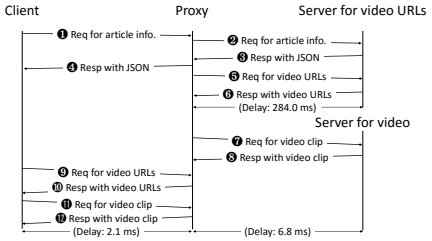

Figure 3: Process of video prefetching

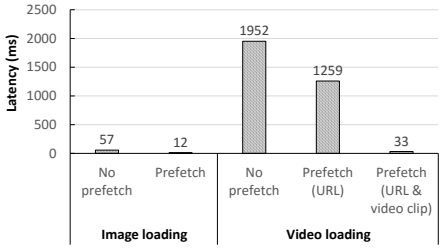

Figure 4: Image/video loading time

sponses are likely to be requested by client afterward. We can also know when to prefetch by using regex signatures of the earlier responses. If the output of the first step contains such requests, the proxy will prefetch the responses.

Code generation: Based on the result of the previous step, we can develop an application specific proxy. As future work, the proxy code is also automatically generated.

4 Case Study: Preliminary Results

Prefetching proxy: To demonstrate the viability of our framework, we implement a prefetching proxy according to the output of our framework and evaluate it. Our target app is a BBC News app, which has two kinds of news articles; ones that have images and ones that have a video clip. BBC News receives a JSON response for each article.It includes the article information, such as textual contents and image/video clip URLs, and is delivered to the app upon loading the homepage of the app or accessing adjacent articles.When an user clicks a certain image article, the corresponding image is requested by using an URL in the JSON.

The proxy prefetches images/video clips using regexes from Extractocol. Figure 2 describes how to prefetch images. The proxy prefetches an image (❺ and ❻) when article information reaches the proxy (❸). The proxy sends images to client (❽) when app requests it (❼). For video clip articles, the JSON of article information includes an URL of the other JSON containing video URLs in terms of bitrate. When touching play button, the JSON with video URLs is requested first and followed by another request for video clip. Figure 3 describes how to prefetch video clip. The proxy prefetches video clips by requesting JSON with URLs of video clips (❺ and ❻) and video clip itself (❼ and ❽) right after the response with article information arrives at the proxy (❸). We can save both latencies for the JSON with video URLs and a corresponding video clip from server.

Preliminary evaluation: We measure the latency decrease due to prefetching. We use Samsung SM-N916S to run BBC News. We implement the proxy based on `mitmproxy` [3] 0.17, sitting between the mobile device and servers. BBC news uses three types of servers: origin server where it fetches image/video URLs, image severs, and video servers. Image and video servers are CDN servers located closer to the device than the origin server. Figure 2 and 3 show latency measurements between mobile device, proxy, and servers.

Figure 4 shows latency of loading image/video.We measure the average latency of 20 runs. Note, the proxy is used in all cases even when it does not perform prefetching. We measure the time of the first byte sent at the proxy, which represents the best case savings. For image loading, it takes 57 ms on average to get images from the server without prefetching. With prefetching, the latency decreases to 12 ms. The average size of images was 57.4 KB.

Without prefetching, the total latency to get a video clip is 1,952 ms.With URL prefetching (i.e., prefetching the JSON containing video URLs), the latency is reduced to 1,259 ms. This is because it saves additional RTT and processing latency of HTTP and JSON messages. In both cases, we measure the time it takes for the proxy to receive the first 4MB chunk of video. This is because Android MediaPlayer starts playing when the first 4MB of video data arrives [1]. When video clip is prefetched, the proxy starts to send the first byte of video after 33 ms. Assuming bandwidth of the mobile device is 27 Mbps (average LTE bandwidth in South Korea), the total latency observed by the app in getting the first 4MB will be about 1218 ms.

5 Conclusion and Future Work

This paper presents a framework for mobile app acceleration leveraging automatic protocol analysis. The framework takes only app binary as input and discovers opportunities for app acceleration. Using the information our framework provides, one can generate app-specific proxies. The preliminary results show an app-specific acceleration proxy whose design derived from our framework can dramatically reduce response times of app. Our framework enhances the quality of user experience for mobile apps and services. We believe our framework will be particularly useful in accelerating various mobile apps in lightly multiplexed environments, such as the mobile edge cloud. Finally, we plan to automate the whole process including the code generation.

6 Acknowledgement

This work is in part supported by IITP funded by the Korea government (MSIP) under B0126-16-1078 and B0101-15-1368 and by NRF of Korea under NRF-2013R1A1A1076024.

7 References

[1] Default Buffer Size of Android MediaPlayer Class - StackOverflow. http://stackoverflow.com/questions/4413300/change-buffer-size-on-mediaplayer.

[2] ETSI - Mobile Edge Computing. http://www.etsi.org/technologies-clusters/technologies/mobile-edge-computing.

[3] mitmproxy. https://mitmproxy.org.

[4] H. Choi, J. Kim, H. Hong, Y. Kim, J. Lee, and D. Han. Extractocol: Automatic extraction of application-level protocol behaviors for android applications. In *ACM SIGCOMM (poster)*, pages 593–594. ACM, 2015.

[5] D. Rayburn. AT&T Partners With Cotendo For App Acceleration, Will Challenge Akamai. http://blog.streamingmedia.com/2010/10/att-partners-with-cotendo-for-app-acceleration-will-challenge-akamai.html.