Abstract—Network latency in mobile apps is the first and foremost concern since the majority of apps frequently fetch data from the Internet and mobile devices rely on wireless networks. To minimize network latency, we propose a novel prefetching technique which has the potential of reducing latency to “zero”. Our approach aims to prefetch latency-hogging HTTP requests in mobile applications, which enables immediate responses of the on-demand user requests. We identify the request candidates for prefetching by static analysis, rewrite the app to interact with a proxy instead of the original server, and prefetch HTTP requests based on runtime QoS constraints. A prototype is implemented for an Android app and the optimized app showed a significant latency reduction.

I. INTRODUCTION

Over the past decade, pervasive mobile connectivity has changed the way we perceive the world and interact with each other. People can be connected with each other at anytime via personal computing devices—mobile devices. In this context, user-perceived latency is the first and foremost concern as it directly impacts user experience and often has severe economic consequences. Previous study [6] showed that network transfers are often the bottlenecks and mobile apps spend between 34-85% of the time in network transfer. This is as expected since the majority of apps frequently fetch data from the Internet. Yet, mobile devices rely on wireless networks, and such networks can exhibit low bandwidth, high latency, and intermittent connectivity [2].

Thus, reducing network latency becomes one of the most effective ways of improving mobile user experience. We propose a novel client-centric prefetching technique that minimizes the network latency in mobile apps. In this paper, latency is defined as the user-perceived latency caused by network, i.e., the response time of a HTTP request. Prior art has primarily taken domain-specific approaches that focus on the “actual” latency reduction in different domains, such as optimizing network paths, expanding the deployment of high-speed network. However, such techniques are fundamentally incapable of reducing user-perceived latency to “zero” due to the performance bottlenecks, such as network speed. Our research takes a different approach, aiming to prefetch latency-hogging HTTP requests, which enables immediate responses of the on-demand user requests. Prefetching overlaps speculative executions with on-demand executions, and it is capable of improving existing latency reducing techniques by removing latency completely.

II. RELATED WORK

Prefetching is an entrenched idea in the mobile computing community. However, most previous works have been focused on “how” to prefetch under different QoS trade-offs [2], while leaving “what” to prefetch as an open question. The state-of-the-art solutions leave the “what” question either to application developers who have to explicitly annotate application segments that are amenable to prefetch [5] or to servers that provide “hints” to clients on what to prefetch based on the history of the requests. There are several key limitations when relying on developers and server support. First, such approaches are not automatic and pose significant manual efforts in both development and deployment. Second, most of today’s mobile applications depend extensively on heterogeneous third-party servers which make deploying server-side “hints” difficult and not scalable [8]. Third, server-side solutions introduce a series of privacy concerns because they require user information from mobile devices to be effective.

However, building a client-centric predictive model to answer “what to prefetch” is a challenging task. Existing works have been building domain-specific models for prefetching. For example, ForeCache [1] builds its model based on data tiles in interactive visualization according to user’s recent movements and past behaviors. EarlyBird [7] focuses on the social network domain and only prefetches the constant URLs in Tweets. Another body of work focuses on the browser domain to optimize the process of request parsing, such as predicting sub-resources [8], providing development support for speculative execution [5]. To the best of our knowledge, we are the first one to propose a general-purpose predictive model on top of the program structure, which provides more accurate prefetching candidates based on the actual program behavior.

III. APPROACH

To address the limitations of current work, we designed a client-centric approach that is automatic, easy to deploy, and extensible to different domains. Our approach optimizes the original app based on static analysis, prefetches future HTTP requests and caches the responses, so that the on-demand HTTP requests are responded immediately from local proxy with “zero” network latency. The insight is that there is usually an idle slot between two user interactions (e.g., user think time), which provides the potential to prefetch latency-hogging operations. The approach overview is shown in Figure 1. The original app is rewritten based on the information extracted by static analysis so that the optimized app interacts with the local proxy instead of the original server. In detail, our approach has three phases: static analysis, app instrumentation, and dynamic adaptation, which are explained in the following.

Static Analysis: In the static analysis phase, we analyze the original app statically to extract the Callback Control Flow Graph (CCFG) and HTTP Request Model (HRM). The CCFG shows the sequence of callbacks from the framework to the application code, both for lifecycle callbacks and for event handler callbacks [9]. CCFG represents the state transitions
between user-driven events so that all the possible future states, such as the states after “onClick”, are known at a current state. Only those HTTP requests in the immediate-next states are selected for prefetching at a current state. Long-term future requests are not prefetched because the cache is more likely to expire. The HRM is a string based model that represents the string values of the HTTP requests including target URL, a set of headers, and parameters [3]. The string values of the identified HTTP requests are interpreted statically using Violist [4]. For those values that are unknown statically (e.g., user input), all the definition spots in different paths are marked for app instrumentation. Definition spots are the line numbers where the concrete string values are obtained at runtime.

App Instrumentation: The app instrumentation phase is also static, the original app is instrumented based on the information extracted in the static analysis phase. The instrumentation is two-fold: (1) For those definition spots identified in the static analysis phase, a statement is inserted after each definition spot in order to send the concrete definition to the proxy at runtime; (2) For those identified HTTP requests, the target URLs are replaced with the address of the proxy so that the prefetched responses can be returned immediately from the local proxy on the user’s device.

Dynamic Adaptation: In the dynamic adaptation phase, the modified app is executed at runtime, interacting with the proxy instead of the original server. The proxy is deployed on the mobile device, and it decides “what” and “when” a certain HTTP request should be sent to the original server and the corresponding response is stored in the local cache. The dynamic adaptation has two dimensions: (1) adapt to user behavior habits, filtering out the unlikely prefetch candidates; (2) adapt to runtime QoS requirements, controlling how much to prefetch at runtime. Figure 2 shows a closer look of the proxy. Proxy stores all the HTTP requests identified by string analysis in static analysis. At runtime, after the unknown string value is known at the definition spot, the modified app sends the concrete value to the proxy. If every unknown values of one HTTP request are known, the certain HTTP request will be considered as a request candidate. Proxy also conducts a cost-benefit analysis which monitors the runtime QoS conditions (i.e., battery life, cellular data usage, bandwidth) to schedule the prefetch according to QoS constraints.

IV. IMPLEMENTATION
A prototype has been implemented to optimize an Android app as a proof of concept. The original app fetches the weather information from OpenWeatherMap based on user’s input and also fetches the weather information of user’s favorite city in every search. Our approach identified that all the values of “favorite city request” are known statically and the values of two other requests are based on user input (e.g. city name). We then rewrote the original app so that the values of user input will be sent to the proxy once they are known at runtime. The proxy is implemented as an Android Service and it checks whether all the values of the HTTP requests in the immediate-next states are known as the user state transits. If so, the proxy sends the actual request to the original server and stores the result. The instrumented timestamps showed that the response time of the HTTP requests is reduced to 0ms instead of 200ms on average in the original app in a WiFi network environment. Currently, we are improving the prototype to automate the approach so that it will be effective and evaluated on a large number of apps in the wild.

V. CONCLUSION
We aim to lay the foundation of client-centric approaches for latency minimization in mobile applications by leveraging program analysis techniques. We believe client-centric approaches have large potential on latency minimization for mobile devices since they are immediate deployable, privacy-preserving, and easy to scale [8]. In addition, prefetching technique is a promising direction but the precision is an important problem because mobile devices are resource-constraint. Current works mainly rely on the history data and user behavior to predict “what” to prefetch [1], [7], which requires learning time and it is unable to achieve a high precision. Differently, our approach makes prefetch decisions based on the static structure of the whole program, which is a more reliable source of “what” to prefetch.

REFERENCES