# Looxy: Web Access Optimization for Mobile Applications with a Local Proxy

Yao Guo, Mengxin Liu, Xiangqun Chen
Key Laboratory of High-Confidence Software Technologies (Ministry of Education),
School of Electronics Engineering and Computer Science, Peking University, Beijing, China
{yaoguo, liumengxin, cherry}@pku.edu.cn

***Abstract*—Efficient web caching and prefetching can help optimize mobile application web accesses through eliminating network traffic and reducing human perceived latency. However, due to limited storage and computation resources, it is not practical to implement web caching and prefetching mechanisms on mobile devices such as smartphones. This paper proposes Looxy, a mechanism to optimize mobile application web accesses by offloading the caching and prefetching functionalities to a local proxy. As a result, mobile applications can benefit from these optimizations without incurring extra computation and storage overhead on smartphones. Looxy does not require modifications to the applications or mobile operating systems, making it applicable to different mobile operating systems and devices. Experimental results show that Looxy can save about 20% of Internet traffic with real user behaviors and improve application response speed by about 1.8X.**

***Keywords—proxy, web access optimization, mobile applications***

## I. INTRODUCTION

With the rapid development of mobile devices such as smartphones, mobile applications (*apps* for short) have replaced mobile web browsers as the major consumer of mobile web traffic on smartphones. A recent study [1] on the behavior of 24 iPhone users found that dedicated mobile applications are used by users to visit the web much more frequently than browsers. Although web browsers often implement many network optimization methods like caching and prefetching, most mobile apps have little optimization on web accesses. A recent study on 1,300 top ranked Android applications showed that 58% of them suffer from the flaw of imperfect web caching [2]. Most of them waste network traffic due to requesting the same web content from the origin server more than once, which may also cause extra latency in responses.

Another difference between mobile apps and browsers is that users tend to have more fixed behavioral patterns in using apps than browsers. When using browsers to explore the Internet, users are more likely to visit webpages that are new to them rather than the pages they have visited before. Most of the webpages are visited no more than once, while most webpages the users will visit in the future are the webpages they have never visited before. As a result, prefetching for mobile browsers is a tough task, as most prediction algorithms can hardly predict new webpages in reality. The reason is because most prefetching algorithms are relying on user browsing histories as their training data. As a result, their prediction also falls into the browsing histories. However, these repeated visits represent only a small fraction of user behaviors in the future.

As for mobile apps, one app usually focuses on a small range of features. These features are implemented by a group of APIs that communicate with the servers to retrieve the requested content, while the URLs in the request typically keep unchanged in a relatively long period for each app. Thus there is a good chance that a predicting algorithm can benefit from this. Although the content sent from the server under the same URL may change from time to time, the content during a short period may not change dramatically, which provides an opportunity to prefetching these contents.


Fig. 1 The Fox News mobile app.

For example, the Fox News app shown in Fig. 1 is a popular news app from Google Play. It splits news into different categories displayed by tags. When clicking a tag, the app will send a request to retrieve the most recent news text, pictures and videos. A request for weather is sent every time we click on a content. As the news content and weather information will not change during a short time period, much web traffic is actually redundant. When using this app, users are unlikely to read all the news. They will only browse a small subset within the categories. So every time they use this app, the requests they make are similar. It makes predicting user behaviors and prefetching contents much easier for apps compared to web browsers.

Though caching and prefetching can optimize the network accesses of an app in theory, they might also increase the storage and computation cost significantly. In the real world, smartphones have limited resources for storage and computation. In addition to that, smartphone batteries may also suffer if we put these optimization methods onto smartphones.

978-1-5090-5932-4/17/$31.00 ©2017 IEEE

This extra cost may outweigh the benefits from these mechanisms.

*Offloading* [3] is a very popular method to deal with limited resources on smartphones. With offloading, the computation-intensive part of an app can be migrated to a remote server (or cloud) to speed up its execution, or use a remote storage system to overcome the storage limits on smartphones. As web caching and prefetching require lots of computation and storage space, offloading to a remote server or cloud appears to be beneficial. However, because our goal is optimizing network accesses, if we put the computation and storage in a remote server that is connected through the Internet, we may lose most of benefits brought by web caching and prefetching because accessing the Internet is relatively slow anyway.

Meanwhile, implementing caching and prefetching on the client side for each app is not an easy task. For example, Android APIs only provide a generic interface for caching. The developers still have to implement the caching details on their own. Prefetching needs efforts to test algorithm accuracy and train models for predicting user behaviors. These tasks will increase the burden of mobile developers and mobile devices.

In order to achieve the benefits of web caching and prefetching, while making them transparent to the mobile devices and mobile app developers, this paper proposes *Looxy*, an approach to offloading the caching and prefetching functions to a local proxy. A *local proxy* is defined as a server or PC (a desktop or laptop) connected to the Internet that a smartphone can access through a fast WiFi connection. Many laptops can act as a wireless access point (AP) with their embedded wireless adaptors. When that option is unavailable, many small devices such as the USB WiFi mini wireless router can transform a nearby PC into an access point in a heartbeat, allowing smartphones to take advantage of their Internet accesses through WiFi connections.

With Looxy, we are able to provide transparent web access optimizations to mobile apps running on smartphones. Developers need no extra effort to modify their apps to adopt Looxy. Apps can benefit from caching to reduce network traffic and prefetching to their improve response speed as well. This approach is also compatible with existing apps and independent of mobile OS platforms. In addition, different users that connect to the same proxy can share their cached and prefetched contents, which provides further potential to optimize the network accesses by eliminating duplicated requests among different users.

## II. Related Work

### A. Caching for Smartphone Applications

Caches have become a standard component for modern browsers for many years. The traditional caching scheme for web browsers is to store frequently visited web resources (e.g., pictures, CSS files, JS files) locally, which is usually guided by the cache control region resided in HTTP headers [4].

Traditional caching schemes are focused on data caching. Zhang *et al.* proposed Smart Caching [5] to cache the intermediate style computation results. As there are lots of redundant computation in style formatting, this method can help reduce the rendering time of a revisited page. Wang *et al.* has extended the approach to exploit the similarities between webpages to further improve browser rending [6].

CacheKeeper [2] has shown that the flaw of imperfect web caching is common in Android apps. They designed a system-level web caching service for smartphones, which performs HTTP 1.1 compliant web caching transparently for mobile applications running above.

Our work is focused on the data cache level and inspired by the design of CacheKeeper. However, we move the cache mechanism out of the smartphone so that it will bring no burden to mobile devices or mobile app developers.

### B. Prefetching for Smartphone Applications

Prefetching can help reduce human perceived latency significantly. For example, Pocketweb [7] uses machine learning to model user behaviors. The model is used to predict future web accesses and prefetch the content in a timely manner. This method trades memory capacity and computation power for lower latency of web access. It requires modification to the operating system platform.

Parate *et al.* proposed an app prediction algorithm APPM [8], which can predict not only which app will be used next, but also when it will be used with high accuracy. Their algorithm needs no prior training and can be used dynamically. However, this kind of study aims to reduce app launch time, instead of optimizing network traffic.

Because smartphones have limited storage and battery, network prefetching on mobile devices is typically not practical. However, after we move the functionality to a local proxy, we are able to apply complicated and aggressive web prefetching techniques as on a personal computer.

### C. Offloading for mobile devices

Smartphone apps are becoming more and more powerful, while the need for computation, storage and communication also increases. Mobile devices limited by device size and battery capacity can hardly satisfy these requirement. However, it is possible to offload some computation to the cloud, which possesses almost unlimited computing and storage resources.

CloneCloud [3] is one of the first mechanisms that can automatically transform mobile apps to benefit from the cloud. It uses a combination of static analysis and dynamic profiling to partition app tasks and offload part of their execution from mobile devices onto the device clones operating in a computational cloud. This method can help improve performance and reduce energy consumption.

MAUI [9] is another system that offloads the code to the cloud rather than the execution status to achieve better energy savings and lower latency.

Many other offloading approaches [10], [11], [12] have been proposed recently, however they mainly focus on offloading computation to the cloud to improve app performance. In contrast, we target at offloading system-level caching and prefetching services rather than part of an app to a local proxy.

## III. System Design

### A. The Looxy Architecture

First of all, Looxy relies on the existence of a local proxy, which resides in the vicinity of the smartphone. The local proxy can serve as a wireless router, or it connects to the same local wireless router that the smartphone can access.

Looxy implements caching and prefetching capabilities on the local proxy, which locates between the smartphone and the Internet and handles all the HTTP requests and responses between the smartphones and servers.

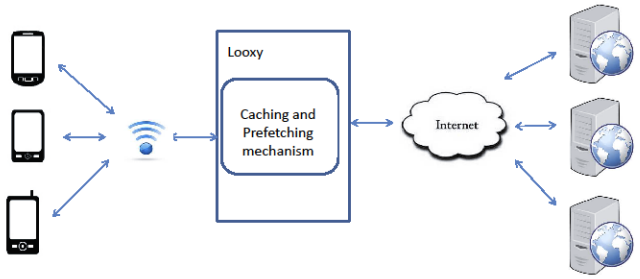


Fig. 2 The overall architecture of Looxy.

Fig. 2 illustrates the overall architecture of Looxy. Looxy acts as a normal web proxy for smartphones to save mobile data (3G/4G) traffic, which we assume already exists in many environments. We choose to place the optimization mechanisms on a local proxy for three reasons. First, we can achieve application and system transparency by placing the mechanisms outside of a smartphone. Second, we can take advantage of the rich storage, computation and network usage of a personal computer (or server) to relieve the burden on the smartphone side. Finally, using a local proxy instead of a remote cloud, the high speed of local WiFi connections provided extra benefits that might otherwise be impractical.

The design of Looxy contains the following components: the HTTP handler, cache manager, log analyzer and the prefetching engine. The relationship and workflow between the components are shown in Fig. 3.

### B. *HTTP Handler*

As a normal proxy, the *HTTP handler* in Looxy takes over all HTTP requests and responses that connect to the proxy. First, it logs messages from HTTP request headers such as the methods, URLs and timestamps. These messages will be used to predict user behaviors by the *log analyzer*. Second, the handler will search the cache for each request to examine whether it has already been cached. If the request hits in the cache, the cached content will be returned to the app immediately without communicating with the remote servers. Third, it also captures all responses to the cache manager to decide whether to cache them. Finally, it provides the responses to the prefetching engine for prefetching prediction.

### C. *Cache Manager*

When a response comes from the HTTP Handler, the *cache manager* decides whether or not to cache the content. The cache logic in Looxy is based on the RFC 2616 specification. The manager sets the expiration time for each cached content. Through our experiments, we found that many responses do not declare their cache control policy. In this case we set a heuristic expiration time for these responses to take advantage of the caching mechanism.

The cache manager also deals with the prefetched content from the prefetching engine, so that we can combine the caching and prefetching mechanisms together. The requests that hit in a cached content that the app has already visited and the requests that have been prefetched by the prefetching engine can be performed in a uniform way. The prefetched contents are given a short expiration time as we predict the contents will be used in a short time. If not, we can clean them up to save space.

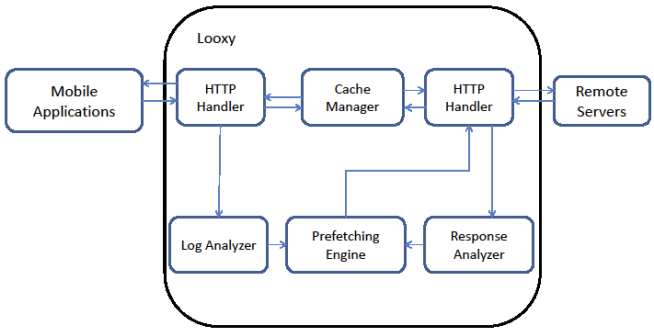


Fig. 3 The workflow of Looxy.

To speed up cache lookup, the manager uses a hash table to store the relations between the URLs and responses. The manager scans the table periodically to remove expired entries. Thus, we can save space while keeping the content fresh.

### D. *Prefetcher*

The *prefetcher* can be further divided into three components: the *log analyzer* processes all the requests to generate user patterns; the *response analyzer* parses all responses to collect links; and the *prefetching engine* controls the prefetching policy and the speed of prefetching.

*1) Log Analyzer.* The log analyzer collects all the HTTP requests from the HTTP handler and uses specific prefetching algorithms to identify app behavior rules from these logs periodically. The results of the analyzer are groups of URLs that are highly likely to be related to each other. They are fed to the prefetching engine to predict future user behaviors.

To find frequent patterns, associations and correlations among a set of logs, we chose the Apriori algorithm [13] which has been frequently used to determine association rules between webpages. Using this algorithm, we are able to identify groups of requests that are highly related to each other. Once a request hits in a group, other requests in the same group are likely to be visited within a short time period. As apps usually request Internet resources in a fixed range, although the Apriori algorithm makes predictions based on history data, it is highly likely to occur in the future.

*2) Response Analyzer.* A response comes from the server may contain links that can be used later in its body. These links usually refer to the content related to the response that may attract the user's attention. If we prefetch these contents, when a user clicks on the link, the content can be sent to the user immediately. The response analyzer is used to parse the response body and identify all the links to other contents.

*3) Prefetching Engine.* The prefetching engine in Looxy can work in two modes: *moderate mode* and *aggressive mode*.

In the moderate mode, the prefetching engine combines the requests from the HTTP handler with the rules coming from

the log analyzer to fetch contents. When a request comes, the engine matches all URLs in the group to check if the request URL is among one of them. If the URL hits in a group, the engine will create a new thread to fetch all the URLs that have not been visited before in the same group asynchronously to the cache manager. The cache manager stores the responses and sets an expiration time for them.

In the aggressive mode, the prefetching engine processes the HTTP request the same way as it is in the moderate mode. When a response from the server comes to the HTTP handler, the response analyzer parses the response body to pick up all the URLs in it. Then the engine creates another new thread to fetch the content.

Though prefetching can bring some benefits, it does not come without side effects. Prefetching too aggressively will exhaust the network bandwidth. If a request comes but the prefetching engine does not respond because it is busy doing prefetching, it might cause that the real request takes much longer time. So the prefetching engine must control the number of concurrent prefetching threads and leave enough bandwidth for normal requests. The engine dynamically adjusts the prefetching strength based on the available bandwidth and the number of URLs waiting for prefetching.

## IV. Experiments and Evaluation

We have implemented a prototype of the proposed Looxy mechanism. The proxy side program is implemented based on an open source program *proxpy* [14]. We implement our caching and prefetching mechanisms by extending *proxpy*.

### A. Experimental Setup

We evaluated Looxy on a Xiaomi smartphone (model HM1) running Android 4.2 with a four-core 1.5GHz CPU and 1GB memory. During the experiments, the smartphone was connected to a WiFi access point and the Looxy proxy was located on a PC that resides in the same LAN with the phone.

In our experiments, we chose three popular apps, among them Fox News and USA Today are famous news client apps and DianPing (similar to Yelp) is one of the most popular local-based service apps in China. For each app, we recruited three different users to use each app for about half an hour and recorded the logs for the log analyzer to generate predicting rules. Then users will use the app for another five minutes, during this period we record their operations. These records will be used to replay user actions with and without applying the Looxy optimizations. Our experiments are meant to simulate the behavior of real users.

We compare four different network optimization setups:

- *Baseline*: The proxy provides no network optimization. It works as a normal web proxy server. In this way we can collect the network traffic data without modifying the apps and operating system on the smartphone. We can also control the network bandwidth through proxy settings to maintain the same network condition in all four mechanisms. The results in this setup will be used as the baseline for our experiments[1].
- *Caching Only*: Only the caching mechanism is enabled in Looxy. In this case, we can detect the redundancy in application network traffic and how caching helps reduce network traffic and improve response time.
- *Caching + Moderate Prefetching*: We enable caching and moderate prefetching based on the rules generated by the log analyzer. In this setup, the prefetching engine does not work in an aggressive way, we can detect the effectiveness of prefetching, the time reduction and the overhead of this prefetching scheme.
- *Caching + Aggressive Prefetching*: Both caching and aggressive prefetching are enabled in Looxy. In this setup, the prefetching engine works in an aggressive way, aiming at further reducing latency. We will compare whether it achieves better effectiveness compared to the scheme mentioned above and compare their network traffic overhead.

### B. Evaluation

*1) Network Traffic.* The caching mechanism can help reduce network traffic by storing frequently used objects to avoid redundant requests. However, prefetching will pre-load some contents to reduce the latency of each operation, when the prefetched contents do not result in a future request hit, it will cause extra network traffic overhead.

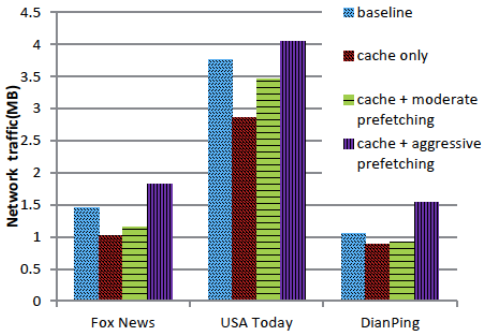


Fig. 4 Network traffic comparison in different setups.

Fig. 4 shows the network traffic measured on the Looxy server in different experimental setups. The results show that caching helps reduce network traffic significantly. There is an average of 22% network traffic reduction in our experiments. Even with moderate prefetching, the traffic can still be reduced by 13%. That is because in the moderate mode, the prediction is based on the user behavior that is permeant in a while. Most of the predictions could match a later request, so the storage overhead does not increase dramatically.

We also observe that if the prefetching engine works in the aggressive mode, it will use more traffic than the baseline with no caching and prefetching. We further analyzed the prefetching process and found that some apps offer different sizes of pictures to adapt to smartphones of different screen sizes. The links of these pictures were provided in a response body, the apps will select the appropriate link to download. In addition to that, some response bodies themselves were over 300KB, which refer to too many links. Only a small fraction of these links were used later on.

[1] Alternatively, we could also use the original apps with no proxies as the baseline. However, it is impossible to measure the network traffic details of a mobile app without modifying the apps or the mobile platform.

When Looxy works in the moderate mode, we break down all the network requests into three categories: the requests that were served directly by servers, the requests that hit in the cache and the requests that matched a prefetched content. The ratio of these three requests are shown in Fig. 5. We can see that about 22% of the requests hit in the cache and about 6% of the requests are served by the prefetching engine.

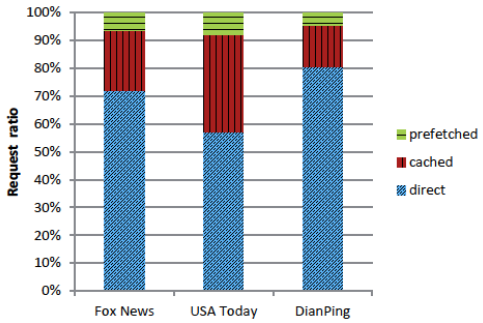


Fig. 5 The breakdown of network requests.

The news apps are more likely to make duplicated requests. We analyzed the request logs and found that they requested the same images more than once when a user read the news list and clicked on a news detail. Some CSS and JavaScript files are also transferred more than once during the process.

*2) Application Performance.* When a request matches a content in the cache from previous requests or prefetching contents, the response can be sent to the app immediately, the response time of an operation can be reduced accordingly.

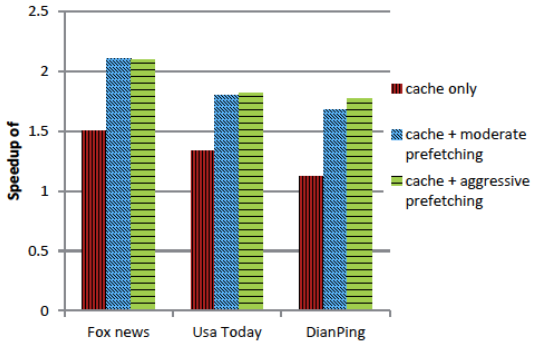


Fig. 6 Comparison of response time in different setups.

We recorded the time for each user click, which represents the time interval between the first HTTP request and the last HTTP response of all the HTTP transactions of a click. The latencies are calculated under each setup and the average speedup of the tested apps are presented in Fig. 6.

We can see that the speedup is about 1.31x with caching only, which is increased to about 1.85x when prefetching is applied in addition to caching. There are no obvious differences between moderate and aggressive prefetching schemes.

We further analyze the prefetching process and find that the predicted contents of the moderate scheme is in the critical path of the response process. The prefetched contents contain the links and information for future requests. Only after these contents reach the app, the succeeding requests can start.

We also find that most of the predictions in the aggressive scheme do not match a later request. That is because the used links are only a small portion of all the links. Furthermore, some apps do not directly use the links from the response. They will add some context arguments like locations, device information, user information and application information to the end of the links. In this case, although the content is the same as the original link, we cannot use this URL to match these different links into the same content.

## V. Concluding Remarks

This paper presents Looxy, a mechanism to optimize the web accesses of smartphone applications. Looxy offloads network caching and prefetching onto a local proxy that resides in the vicinity of the smartphone. With Looxy, applications can take advantage of network optimizations transparently, with no modifications of the mobile apps or operating systems needed. We have implemented a prototype of Looxy and demonstrated its effectiveness through experiments on popular mobile apps.

## Acknowledgment

This work was partially supported by National Key Research and Development Program Grant No. 2016YFB1000105 and National Natural Science Foundation of China Grant No. 61421091.

## References

[1] C. Tossell, P. Kortum, A. Rahmati, C. Shepard, and L. Zhong, "Characterizing web use on smartphones," in CHI, 2012, pp. 2769–2778.

[2] Y. Zhang, C. Tan, and L. Qun, "Cachekeeper: A system-wide web caching service for smartphones," in UbiComp, 2013, pp. 265–274.

[3] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in EuroSys, 2011, pp. 301–314.

[4] H. Wang, J. Kong, Y. Guo, and X. Chen, "Mobile web browser optimizations in the cloud era: A survey," in MobileCloud, 2013, pp. 527–536.

[5] K. Zhang, L. Wang, A. Pan, and B. B. Zhu, "Smart caching for web browsers," in WWW, 2010, pp. 491–500.

[6] H. Wang, M. Liu, Y. Guo, and X. Chen, "Similarity-based web browser optimization," in WWW 2014, Seol, Korea, Apr 2014.

[7] D. Lymberopoulos, O. Riva, K. Strauss, A. Mittal, and A. Ntoulas, "Pocketweb: Instant web browsing for mobile devices," in ASPLOS, 2012, pp. 1–12.

[8] A. Parate, M. B¨ohmer, D. Chu, D. Ganesan, and B. M. Marlin, "Practical prediction and prefetch for faster access to applications on mobile phones," in UbiComp, 2013, pp. 275–284.

[9] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in MobiSys, 2010, pp. 49–62.

[10] F. Mehmeti and T. Spyropoulos, "Is it worth to be patient? Analysis and optimization of delayed mobile data offloading," in INFOCOM, 2014.

[11] M. V. Barbera, S. Kosta, A. Mei, V. C. Perta, and J. Stefa, "Mobile offloading in the wild: Findings and lessons learned through a real-life experiment with a new cloud-aware system," in INFOCOM, 2014.

[12] L. Xiang, S. Ye, Y. Feng, B. Li, and B. Li, "Ready, set, go: Coalesced offloading from mobile devices to the cloud," in INFOCOM, 2014.

[13] R. Agrawal, R. Srikant et al., "Fast algorithms for mining association rules," in VLDB, vol. 1215, 1994, pp. 487–499.

[14] "proxpy," https://code.google.com/p/proxpy/.