Precog: Action-Based Time-Shifted Prefetching for Web Applications on Mobile Devices

Shruti Sanadhya[1,3], Uma Parthavi Moravapalle[2,3], Kyu-Han Kim[1], Raghupathy Sivakumar[2]

[1]Hewlett Packard Labs, Palo Alto, CA
[2]Georgia Institute of Technology, Atlanta, GA
[3] Co-primary authors

ABSTRACT

Wireless providers today are highly motivated to improve efficiencies of spectrum usage. One approach to achieve this is to shift the load from expensive cellular networks to cheaper WiFi networks. In this context, we propose Precog, an *action-based prefetching* solution for time-shifted WiFi offloading. We argue that traditional prefetching solutions, that rely on the URLs visited in the past by a user for predicting future access, are ineffective in today's dynamic, interactive, and personalized web. Precog addresses this issue by remembering, not the exact URL accessed in the past, but the actions performed on a particular website. The actions are remembered as interactions with the content layout, which stays consistent over a long period of time. Unlike prior offloading solutions that require concurrent cellular and WiFi connectivity, Precog offloads cellular content over time-shifted WiFi access. We evaluate Precog over both synthetic and real user datasets to demonstrate its benefits.

1 INTRODUCTION

Cellular technologies such as 3G and 4G, in tandem with WiFi, serve as the fundamental access mechanisms for mobile users. Recent studies show that the availability-cost trade-offs of these technologies result in users relying heavily on both for data-access. Mobile devices today are almost always equipped with heterogeneous interfaces with each interface having distinct cost-bandwidth-connectivity tradeoffs. WiFi is low-cost and high bandwidth, but is not always available for use; whereas cellular is high-cost and low bandwidth, but is almost always available. A recent survey of Internet pricing plans [1] shows that cellular networks charge \$10/GB (high end plans). On the other hand, cable networks, which provide WiFi access, charge \$0.20/GB, an order of magnitude cheaper. Additionally, data is almost unlimited on WiFi, but monthly data limits are imposed on cellular network. *This creates an unbalanced cost problem for mobile users.*

Corresponding author: Uma Parthavi Moravapalle (parthavi@gatech.edu)

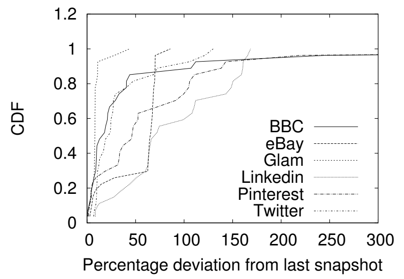
This work was funded in part by the National Science Foundation under grants CNS-1513884, CNS-1319455, and IIP-1701115, and by the Wayne J. Holman Endowed Chair.

HotWeb'17, San Jose / Silicon Valley, CA, USA
© 2017 ACM. 978-1-4503-5527-8/17/10. . . \$15.00
DOI: 10.1145/3132465.3132473

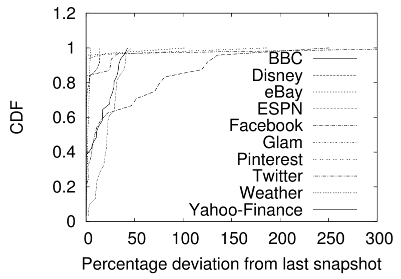
Both mobile users and network operators are continually struggling to reduce their data costs and network operation costs, respectively. Previous works have considered WiFi offloading to address the unbalanced cost problem by shifting ongoing cellular traffic on to WiFi [2–4]. But these solutions rely on *concurrent availability* of both cellular and WiFi networks. However, access to wireless and cellular networks is mostly time- and location- shifted. Users connect to WiFi while at home, office, coffee shops and rely on cellular connectivity while in transit, visiting client offices or shopping in stores. An important question to ask here is: *How can time-shifted access over the cheaper (WiFi) network be leveraged more effectively to reduce cost of the more expensive (cellular) network access?*

In this paper, we consider a solution that can predictively *prefetch* over WiFi what a user is likely to access over cellular in the future. While prefetching in itself is a well-studied topic in related research, we argue that existing prefetching solutions [5–7] are unsuitable to the specific problem considered in this paper because: (i) They are predominantly URL-based, i.e., they remember the exact URLs accessed in the past and only prefetch a subset of those. Such solutions severely under-perform in today's web which is more dynamic and uses complex client-side technologies; (ii) They perform just-in-time prefetching and were not necessarily designed for time-shifted prefetching; (iii) Bandwidth savings can only be achieved if the entire prefetched object is reused in a later access, and not if the object is changed to any degree; (iv) They are viewed as a client side solution, and resource constraints both on the mobile device and the wireless link necessitate a re-thinking of such an architecture;

In this context, we present *Precog*, an action-based solution that performs time-shifted prefetching over WiFi to reduce cellular data consumption. Briefly, *Precog* remembers not the exact URL accessed in the past, but the actions performed on a particular website. Even though web content changes frequently, there remains (by design) *visual consistency* in the layout of websites. We leverage this consistency in *Precog*'s design. *Precog* reaps benefit from the prefetched content at a sub-object granularity through network deduplication. We propose *Precog* for web applications on the browser for smartphones and tablets as browsing is the second most popular application on mobile phones [8]. In its architecture, *Precog* is designed as a client-and-proxy model, where *precog-client* can be a web-browser add-on and *precog-server* (or proxy) can be placed as a cloud service. We evaluate *Precog* over real web content fetched through synthetic user and network traces and show that *Precog* can give 45% byte savings over cellular network with a 35% prefetch efficiency over WiFi. We also compare *Precog* with name-based prefetching using real traces and show that *Precog* is 1.4× better.



(a) Change in URLs



(b) Change in DOM

Figure 1: Dynamic web content

2 MOTIVATION

This section examines five main changes in the way users interact with the web on mobile devices and limitations that existing prefetching solutions face.

Network unawareness and just-in-time behavior: In existing solutions, prefetching is triggered based on: (i) content accessed in the past [5, 6], where access to URL *A* triggers the fetching of URL *B* which was accessed most subsequent to URL *A*, in the past, or (ii) time[7], where most accessed URL at given time of day is prefetched every day at the same time. The cost of network access has never been considered as a trigger for prefetching, as these solutions were focused on wired environments. With the prevalence of smartphones and tablets, network heterogeneity has become more pronounced. Additionally, content is prefetched just-in-time for consumption in the same access session.

Inability to handle dynamic web: Prior works on prefetching [5–7] propose a *name-based prefetching*, which relies on the names, in the form of Universal Resource Locators(URLs), of web content to decide what to prefetch. Specifically, *name-based prefetching* remembers users' browser history in the form of URLs visited in the past. It then selects the URLs most likely to be accessed again and prefetches them. The assumption that users access the exact same URL every time does not hold in today's Internet. The world wide web is becoming increasingly more dynamic. Websites keep updating content to incorporate latest trends and preferences. Content management systems on the web provide a scalable approach for web management, but entail dynamic naming strategies.

To support our observation we analyzed the change in number of URLs for six popular mobile websites (obtained from Comscore), for a week. We downloaded each website, up to second level of reference, every six hours, supplying a cookie file for login where needed.We then computed the number of new URLs *added* in a snapshot as a percentage of URLs in the previous snapshot. From the CDF of the percentage change (see Figure 1a), we saw that there is significant variance in the changes on different websites. Some websites changed less than 25% across 80% of the instances while others changed more than 40% for half of the instances.

While the above analysis is a pure server-side analysis of six websites, we also did an analysis of user accesses across different websites. We considered the LiveLab dataset [9], which is collected from 24 volunteers in Rice University using iPhone 3GS from Feb 2010 to Feb 2011. This data set contains web browser history collected from Safari browser every night. Each record contains a timestamp and a hashed URL. For each user, we computed the percentage of URLs which repeat in the trace. Additionally, we also computed the percentage of domain names, that occur more than once in the entire trace. We observed that, on an average, 81% of domain names are repetitive, while only 21% of the URLs repeat in the year long trace. This shows that users access similar but not exactly the same content. Hence, there is significant variation in website content and user behavior in terms of the object URLs, reducing the efficiency of a pure name-based approach.

Agnostic to client-side logic: Over the past decade, applications have evolved such that logic is driven not just by server, but also by client-side technologies such as AJAX, Flash, JavaScript, etc. Web has become more interactive and response time is significant in defining performance. Thus technologies like AJAX, which allow for asynchronous updates of webpages, have been adopted in web applications. Additionally, richer applications need non-standard technology to perform complex tasks, e.g., Flash, Java-applet, ActiveX, which have become popular in the past decade. Additionally, today's collaborative web uses complex user data objects which are sent to servers, further necessitating sophisticated client-side logic. A prefetching solution that runs a simple HTTP GET request for a single URL cannot incorporate newer, and more impactful, client-side logic governed by new technologies.

Ineffective for personalized access: In addition to web dynamics and newer technologies, content is becoming increasingly personalized on the web. Several websites, such as Facebook, YouTube, etc, require username and password information to provide per user customized content. Name-based prefetching is inefficient in such cases. The prefetching algorithm needs to remember the state in which the user accessed a URL and recreate that state on the web server to get access to the new URL. A stateless prefetching approach cannot prefetch such content.

Overheads: Earlier work on prefetching was mainly developed for desktop environments, and is agnostic to resource constraints imposed by the CPU, memory and battery capacity of mobile devices. Thus, all the prefetching logic could traditionally run on the end-device as the sole purpose of prefetching was to reduce web latency. However, a prefetching solution for smartphones and tablets needs to be designed such as to minimize load on the device resources as well as the wireless link.

2.1 The Case for Action-Based Prefetching

Given that name-based prefetching is ineffective, we propose a solution which leverages consistency on a different front to perform prefetching - *Despite the change in URLs, website layouts stay* *consistent.* By the usability principle of consistency[10], web site layouts should follow same design templates, despite changes in the content, to ensure high usability of the website.

On the web, content layout is modeled in the form of HTML Document Object Model(DOM). The content of a webpage is structured using a DOM tree of HTML tag nodes, with the *html* tag node as the root. We now look at a set of ten popular mobile sites, including the six studied in section 2 above, and study the changes in DOM tree of the homepage of each website, every six hours over six days. We calculated the total number of insertions and deletions to reflect the tree change. Figure 1b shows CDF of percentage change in the number of nodes in the tree. We can observe that almost all websites show less than 40% change in DOM across all snapshots. For 60% of the snapshots the variation is less than 25% for most sites. This is in contrast to the URL changes we observed in section 2. Thus, even when content on the webpage changes, the consistency in layout of content is maintained through the DOM tree.

In addition to consistency in web layout, users are also consistent in their actions. To substantiate this, we analyzed a click data set provided for public use by Microsoft[11]. This data set includes results of monitoring areas on $microsoft.com$ where each of 38000 users click during a week. We found that *over 80% users clicked on less than five areas*. These results show that users are consistent in their activity on a website. At the same time, the LiveLab results show these actions do not always lead to same URLs. While there is limited consistency in URL names, there is consistency in the user activity and in the content layout on websites. Thus, instead of learning names of content accessed in the past, *consistent user actions can be learned from the past and applied to consistent content layouts to predict dynamic future content.* This forms the core idea of our solution *Precog*.

3 THE PRECOG SOLUTION

Precog is an action-based prefetching solution for time-shifted WiFi offloading. At a high-level, *Precog* consists of a *precog-server* located in the cloud, in the data path of cellular access, and a *precog-client* sitting on the mobile device. The *precog-client* tries to learn the *actions* users perform on the web, while over cellular. When on WiFi access, it sends these actions to *precog-server* which uses them as foreknowledge of the events expected over next cellular session and performs these actions in advance to cache content for cellular access. The content prefetched by *precog-server* is pushed to the *precog-client* over WiFi. For future cellular access this cached content is used for network deduplication between *precog-server* and *precog-client*, reducing the cost of data access. The key design elements of *Precog* are best described as answers to the following questions:

When are user actions recorded? Due to the unbalanced cost problem, user access pattern over WiFi and cellular networks are different. As *Precog* focuses on predicting future cellular accesses, *precog-client* only records user actions while the device is connected to cellular network.

How are user actions recorded? Past prefetching techniques have recorded user actions on webpages in the form of content names, i.e. URLs. This approach is not robust, as shown in Section 2. An alternate approach can be to record the graphical coordinates of each user click or text entry. However, this approach is not robust against screen rotation, page scrolls, etc. Instead, *Precog* relies on the HTML DOM tree to record user actions. Every element on the page can be identified based on its location in the tree, which does not change often, providing consistent user experience.

Each user action UA_i on the webpage can thus be identified as a (*DOM descriptor*, *Event*) tuple, where *DOM descriptor* is used to locate a node in the DOM tree and the *Event* is any interaction with the element, such as click, checkbox selection or form submission. *Precog* only considers user actions that trigger HTTP GET requests and stops recording actions if a non-signin HTTP POST is triggered. Handling POST messages is discussed further in section 6. The *DOM descriptor* is described using DOM attributes. Some nodes in the tree can be easily identified through an *id* attribute associated to them. Nodes which do not have an *id* attribute are identified using a (*start_id*, *path*) tuple, where *start_id* is the nearest DOM parent node which has an *id* attribute and *path* is the relative tree path from *start_id* to the target node. The *DOM descriptor* is thus specified as the three tuple $start\_id, path, target$, where *start_id* and *path* are used to locate the node and $target$ is the attribute of the node on which action was performed.

How are sequences of actions grouped together? All the actions recorded by *precog-client* are uploaded to the *precog-server* over WiFi. The user may perform a sequence of DOM actions, separated by non-DOM actions that do not involve interaction with the DOM tree (e.g., opening the browser, changing a tab, etc.). While DOM actions depend on the result of previous DOM actions, such dependencies are typically broken by non-DOM actions. Hence, *precog-server* groups sequences of dependent actions into sessions and separates different sessions by identifying non-DOM actions.

Precog-server records each session in the form of session trees. The static URL which is accessed at the start of the session becomes the root of the tree. This can be the first URL user typed when the browsing session started or when the user clicked a bookmark. Any action performed on the root URL becomes a child of the root node. Performing any action UA_i at time t_j can trigger HTTP requests for webpage W_{ij}. Any DOM actions performed on webpage W_{ij} adds children nodes to node UA_i. Thus, each node in the tree represents the URL reached by visiting the root URL and following all the actions in the path from the root to the node. A separate session tree is maintained for each static URL visited by the user. *Precog-server* maintains a forest of session trees to remember all action sessions for a user.

How to select actions to prefetch? To determine which nodes on the tree to prefetch, *precog-server* also maintains a counter for the number of times a user has performed the corresponding action (hit_i) for each node in the session tree. Every time user repeats a set of actions, i.e., traverses a preexisting path in the session tree, *while on cellular network*, the counter on each node on that path is incremented. When the prefetch decision is to be made, all nodes in the tree are ranked according to their hit counter and top k nodes in the tree are prefetched. The count k is set to the median of the number of actions performed per cellular connectivity period. Note

cbsinteractive.com	nbcuni.com	walmart.com	amazon.com
vevo.com	apple.com	nytimes.com	yelp.com
about.com	weather.com	epsn.go.com	bbc.com

Table 1: List of websites crawled

that the hit counter of a parent node is always higher than its child. Hence the dependencies of each action are prefetched before the action.

How to perform prefetch? Based on the ranking determined above, *precog-server* fetches the highest ranked content when the mobile device is connected over WiFi. The *precog-server preplays* the actions using a *headless browser*. The headless browser runs in the background allowing the cloud based module to scale up to several users. Simultaneously, the *precog-client* module on the mobile also checks if the available disk, CPU and battery are above a certain threshold. As long as the system resources permit, content is fetched from the *precog-server* over WiFi to ensure that the latest content is cached on the device.

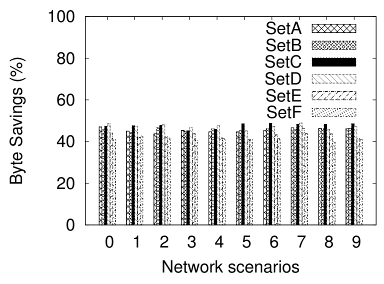
Only the root node of each session tree has a fixed URL which can be directly requested through a browser. For all interior nodes, the user actions are *preplayed* to request the content created as a result of those actions. Thus, to prefetch the child node UA_1 of $Root\_URL_1$, first $Root\_URL_1$ is opened in the headless browser and then action UA_1 is performed. The resulting HTTP GET request, say for URL W_1', fetches the latest content corresponding to that node in the tree. Each UA_i is preplayed using DOM methods and attributes. For example, if the *DOM descriptor* of UA_1 is (sid_1, $index_1$, $attr_1$) then the target element of the action is located using JavaScript DOM methods.Once the target node has been determined, the recorded *Event* is preplayed on it. The result of each preplay is used to perform further related actions.

How to use the prefetched content? The content prefetched by *precog-client* from the *precog-server* can be cached in the browser or at any lower layer. However, prior work [12] shows that web caching on mobile devices does not effectively leverage redundancy in content, due to faulty implementation of the HTTP protocol. Instead, *Precog* uses network deduplication(dedup) to conserve bandwidth both over WiFi and cellular network. Briefly, a *dedup-source* located on the *precog-server*, intercepts prefetched traffic coming from *precog-server* to *precog-client*; segments the traffic into chunks of byte-sequences; and sends across only the hash of a byte-sequence if it is a repeating sequence. The *dedup-destination*, here the *precog-client*, then inflates any hashes back to the original byte-sequences and caches the content locally. While dedup reduces the network footprint of *Precog* over WiFi, its main benefit is reaped when the dedup cache built over WiFi is used to reduce bytes downloaded over cellular, which cannot be achieved by a browser cache.

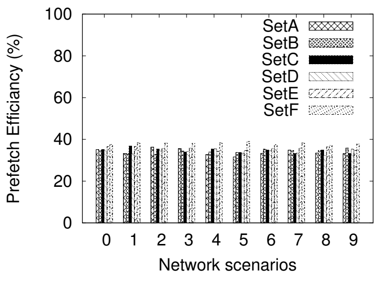
4 EVALUATION

4.1 Synthetic traces:

We evaluate *Precog* using real network traffic generated through synthetic user and network traces. Each user trace lists the different action paths followed by a user, per hour of the day. Each action



(a) Byte Savings



(b) Prefetch Efficiency

Figure 2: Performance of *Precog* over Synthetic traces

path is defined as *(Root_URL_i, UA_a, UA_b, ...)*. The trace creation has the following components:

- *Creating the universal set of action paths*: We first create a large set of possible paths that a user may traverse by crawling twelve popular sites listed in Table 1. We use WebDriver, a component of Selenium browser automation tool to click on all possible elements on each website, traversing links up to a reference depth of ten[1]. This exercise generates a set of over 470,000 action paths.
- *Creating individual user traces*: Next, we extract user traces from this universal set. The main criteria for user trace creation is to have users with different levels of redundancy in their trace. To generate a week long user trace, we assume that user accesses network for sixteen hours per day and seven days a week. Every hour, the *trace generator* randomly selects *access_count* paths from the universal set for that session. For each of the path selected for a session, the module further selects a random depth from one to ten and truncates each path up to that depth. These truncated paths are then tagged with the hour value and added to the user trace. The parameter *access_count* controls the redundancy in user actions per trace as users with more paths per hour have larger intra-session redundancy (accessing the same site multiple times over cellular) and hence lower redundancy across sessions. We vary *access_count* as 100, 250, 500, 1000, 2000 and 4000 and refer to these six user categories as *SetA*, *SetB*, *SetC*, *SetD*, *SetE* and *SetF*. Ten user traces are generated for each set.
- *Creating network traces*: In order to evaluate each user over multiple network profiles, we also create ten network profiles. Again, we assume that network is accessed sixteen hours a day for seven days a week. We also assume that network stays same for at least an hour. For each of the 112 hours, we randomly select either WiFi or cellular connectivity with equal probability. One exception to this random selection is that the start of the day, i.e. the user starts the day at home, where there is WiFi connectivity.

[1]We do not use any authentication based websites in this evaluation and the only event performed is clicks on HTML anchor tags on each page.

We next run the *Precog* algorithm on each combination of user and network traces. A custom *trace analyzer* is built using Python. In each run, the first six days, i.e. first 96 hours, of the traces are used to build the forest of session trees. Every hour the *trace analyzer* checks the network connectivity and if it is cellular, all the paths in that hour are used to grow the forest of session trees. Accesses over WiFi connectivity are ignored. No network traffic is downloaded for the first 96 hours. At the 97^{th} hour, prefetching is done as the user is on WiFi. The *trace analyzer* computes *next_access*, which is the median number of accesses seen in the past over cellular. During the prefetch session, the top *next_access* nodes in the forest, based on the hit count, are prefetched. Again, WebDriver is used to preplay the actions on Firefox and *real* network traffic is recorded using Tcpdump. When the user next switches to cellular network, all the actions for that hour are also replayed using WebDriver and the session trees are also populated. The *real* network traffic generated in this session is also recorded.

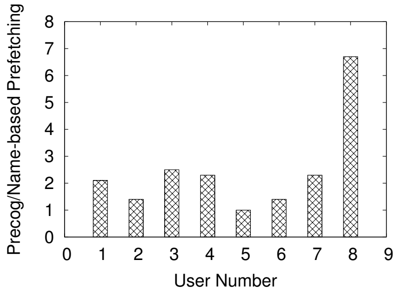
As name based prefetching provides negligible improvement because of reasons stated in section 2, we compare performance of synthetic traces against achievable ideal performance. For this purpose, We track the following from the two traffic dumps generated by the *trace analyzer*: (i) Bytes downloaded in cellular session ($Bytes_{cellular}$), (ii) Bytes downloaded in WiFi session ($Bytes_{WiFi}$), and (iii) Bytes *in all byte sequences* matching across cellular and WiFi session ($Bytes_{match}$). We compute the byte savings achieved by *Precog* as: $\frac{Bytes_{match}}{Bytes_{Cellular}}$ and the prefetch efficiency as: $\frac{Bytes_{match}}{Bytes_{WiFi}}$.

Figure 2a shows the average byte savings observed across the users for each network trace. We observe that *Precog* saves 45% bytes over cellular. From Figure 2b, we observe that for every 100 bytes downloaded over WiFi, 35 bytes are saved over cellular network using prefetched dedup cache. Note that the cost of data access over cellular is around *50 times* of WiFi access. Thus, a 35% prefetch efficiency with a 45% byte savings, still gives a 45% cost reduction over cellular.

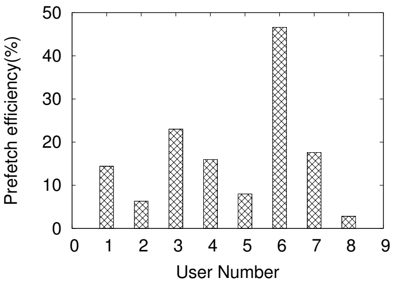
4.2 User traces:

We also evaluate *Precog* on action traces collected over a week from eight volunteers who used Firefox on their Android smartphones. In order to record the actions, we developed an add-on for Firefox on Android that dumps every DOM action (either a click or a form submit) along with a $DOM_{descriptor}$ to a file stored on the phone's sdcard. For 'input' actions (for example: form submit), we also store the input values. To obtain the the type of network (mobile or WiFi) on which the corresponding action was performed, we asked the users to install an application that uses Android's Connectivity Manager API to monitor when the active network changes, and dumps it to a file along with the time. At the end of the week, the volunteers gave us an action file and a connectivity file. Capture of Non-DOM actions performed by a user (such as, navigation actions like back, manual changes to the location bar etc.) was not possible through an add-on, as the browser does not trigger events corresponding to these actions. However, we obtain this information by post-processing the traces.

Given the action and connectivity file per user, each user's actions were split into sessions of network access. For every session, we preplay the actions predicted by the *Precog* algorithm initially, wait



(a) Improvement over name-based prefetching



(b) Prefetch Efficiency

Figure 3: Performance of *Precog* over Real User Traces

for half an hour, then play the actions in the next session. We run tcpdump during prefetching and playing the actual session. We preplay actions as follows: for every *new* action, we load the webpage on SlimerJS and use CasperJS scripting utility to perform DOM actions.

Given the traffic dump for each prefetch and actual access session, we compute the performance of *Precog* and name-based prefetching on the traces. To measure the performance of name-based prefetching, HTTP responses in each session are matched against the responses seen in all prior sessions. The number of URLs prefetched is restricted to the median number of URLs fetched in past sessions.

From these traces, we observed that byte savings per user vary from 3 to 37%. The low performance numbers are a result of low amount of traffic for some users. However, Figure 3a shows the ratio of byte savings by *Precog* and name-based prefetching, where we observe that *Precog* provides more than 1.4× improvement over all users, rising as high as 6.7×. This further shows that pure name-based prefetching is ineffective. The prefetch efficiency for these traces ranged from 3 to 47%, as shown in Figure 3b.

5 RELATED WORK

WiFi offloading has been studied in several prior work [13] in the context of vehicular networks [14], non-vehicular scenarios [3] and urban area offloading [4]. Recently, several commercial products have also come up to assist data offloading from cellular networks to WiFi[2, 15]. However, all these solutions rely on concurrent WiFi (or wired back-haul) and cellular access or propose delays in transfers. Several prefetching solutions have been proposed for wired and desktop environments. [5] tries to leverage URL history across users to build a dependency graph of URLs. On the other hand, [7] proposes to build a probability count for each URL visited by a single user at a certain hour of the day. A prefetching architecture for mobile devices was proposed in [6]. All these solutions suffer the limitations identified in Section 2. Recently, [16] proposed an

API for mobile applications which can decide whether performing a prefetch is beneficial in terms of energy and bandwidth saving. *Precog* is a complementary approach to this solution as it predicts what to prefetch. [17] proposes a time-shifted prefetching mechanism for based on URL and keyword patterns. However, it is specific to news pages and suffers the same disadvantages as name based prefetching.

6 DISCUSSION AND FUTURE WORK

Robust replay: As we observe in section 2.1, DOM trees are not entirely static and infrequent changes may happen. Thus some of the actions recorded for a user can fail on new DOM layouts. If *Precog* finds a scenario where no node can be found for the recorded DOM locator, e.g., child index is out of range, then the preplay is aborted. No further actions in the current action sequence are executed.

Computational overheads: *Precog* minimizes the burden of prefetching on mobile devices. The three main modules running on the mobile device are Firefox add-on, a connectivity monitor, dedup-destination and Activity uploader(a simple FTP service). We believe the Firefox add-on does not add much overhead on the CPU. None of the users whom we gave the add-on and connectivity monitor app, complained about battery or delay in page load. The most complex module in the *precog-client* is that of the Dedup destination. Recent work[18] has shown that much complex implementations of dedup-destination can run on current smartphones, without prohibitive CPU/Memory overheads.

Impact on battery: MAUI[19] showed that WiFi gives 102 KB/Joule efficiency while 3G networks give only 36 KB/Joule. The trace analysis in Section 4 shows that *Precog* gives 35% efficiency, thus around $3\times$ bytes needs to be prefetched over WiFi to save a given number of bytes over cellular. This ratio is similar to the ratio in energy consumption across these interfaces, hence not very burdensome. Reducing these overheads further is part of future work.

Handling video traffic: Video traffic is a dominant part of Internet traffic. There are multiple granularities at which prefetching can offset the cost of cellular video access: (i) The entire video can be prefetched to serve future requests from the user, (ii) some popular sections of the video can be prefetched based on a popularity metric or (iii) the video advertisements, which appear during a video, can be prefetched. In this work we have evaluated the performance of *Precog* for the first granularity and plan to explore others in future.

Privacy concerns: The action-based approach of *Precog* may create privacy concerns as accessing authentication protected content can be intrusive to users. *Precog* addresses this by exposing to the user whether *Precog* is enabled for that domain or not. Additionally, *Precog* does not preplay any actions (except sign-in) that trigger an HTTP POST request, as POST requests can lead to state changes at the server, which are irreversible. We plan to investigate these more in future.

Extension to non-web applications: *Precog* is not just restricted to browser and is easily extensible to mobile applications developed over HTML5 as they also follow the DOM structure. In addition to this, any application that exposes its UI elements to *Precog* can reap the benefits of prefetching. AndroidViewClient is a tool to expose a tree structure of different views in Android applications, providing scope for solutions like *Precog*. We plan to investigate this in future.

7 CONCLUSION

In this paper, we have presented *Precog*, an intelligent prefetching solution for time-shifted WiFi offloading. *Precog* employs an action-based network-aware prefetching for mobile web-browser traffic, and leverages consistency in users action and web content layouts. Compared to the traditional name-based approach, *Precog* is a name-independent network-aware prefetching solution and has addressed fundamental challenges in dealing with dynamic and personalized contents by recording and predicting user's actions. As a result, *Precog* helps mobile devices achieve time-shifted WiFi offloading over cellular networks. We have implemented and evaluated *Precog* through the analysis of both large-scale synthetic trace and real-life user trace to show it's benefits. As part of future work, we plan to extend *Precog* to make more opportunistic prefetching decisions and handle authentication based traffic.

REFERENCES

[1] S. Sen et. al. Incentivizing time-shifting of data: a survey of time-dependent pricing for internet access. *Communications Magazine, IEEE*, 50(11):91–99, November 2012.

[2] Aruba WiFi Offload. www.arubanetworks.com/pdf/solutions/SB_Offload.pdf.

[3] Kyunghan Lee et al. Mobile data offloading: how much can wifi deliver? In *ACM SIGCOMM '10*, pages 425–426.

[4] S. Dimatteo, Pan Hui, Bo Han, and V.O.K. Li. Cellular traffic offloading through wifi networks. In *2011 IEEE 8th International Conference on Mobile Adhoc and Sensor Systems (MASS)*, pages 192 –201, oct. 2011.

[5] V. Padmanabhan and J. Mogul. Using predictive prefetching to improve world wide web latency. *SIGCOMM Comput. Commun. Rev.*, 1996.

[6] Tong Sau Loon and Vaduvur Bharghavan. Alleviating the latency and bandwidth problems in www browsing. In *Proc. of the USENIX Symposium on Internet Technologies and Systems*, pages 20–20, 1997.

[7] Kelvin Lau and Yiu-Kai Ng. A client-based web prefetching management system based on detection theory. In *Web Content Caching and Distribution*, volume 3293 of *Lecture Notes in Computer Science*, pages 129–143. Springer Berlin Heidelberg, 2004.

[8] H Falaki et al. Diversity in smartphone usage. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 179–194, 2010.

[9] The Livelab Project. livelab.recg.rice.edu/index.html.

[10] William Lidwell, Kritina Holden, Jill Butler. *Universal Prinicples of Design*. Rockport Publishers, 2003.

[11] Anonymous Microsoft Web Data Data Set. archive.ics.uci.edu/ml/datasets/Anonymous+Microsoft+Web+Data.

[12] Feng Qian et al. Web caching on smartphones: ideal vs. reality. In *ACM MobiSys*, 2012.

[13] F. Rebecchi et al. Data offloading techniques in cellular networks: A survey. *IEEE Communicatios Surveys Tutorials*, 17(2):580–603, '15.

[14] Aruna Balasubramanian, Ratul Mahajan, and Arun Venkataramani. Augmenting mobile 3g using wifi. In *ACM MobiSys*, 2010.

[15] Ruckus 3G/4G oFFload. www.ruckuswireless.com/carriers/3g-offload.

[16] B. D. Higgins et al. Informed mobile prefetching. In *ACM MobiSys'12*.

[17] J. Han et al. Network agile preference-based prefetching for mobile devices. In *IEEE International Performance Computing and Communications Conference (IPCCC)*, pages 1–8, Dec 2014.

[18] S. Sanadhya et al. Asymmetric caching: improved network deduplication for mobile devices. In *Proc. of MobiCom*, 2012.

[19] E. Cuervo et al. Maui: making smartphones last longer with code offload. In *ACM MobiSys*, 2010.