EarlyBird: Mobile Prefetching of Social Network Feeds via Content Preference Mining and Usage Pattern Analysis

Yichuan Wang
University of California, Davis
yichuan1118@gmail.com

Xin Liu
University of California, Davis
liu@cs.ucdavis.edu

David Chu
Microsoft Research
davidchu@microsoft.com

Yunxin Liu
Microsoft Research
yunliu@microsoft.com

ABSTRACT

Social networks are the most engaging applications on mobile devices, and they are becoming the main sources for users to consume content. However, content retrieval, especially for embedded links and multimedia, can often be too slow, too energy hungry or too expensive for on-the-go mobile users. To address these issues, we collect and analyze a large set of traces from over 6000 real-life users of a popular mobile Twitter client. Based on the unique challenges identified from our dataset, we present inference-based social network content prefetcher, EarlyBird. It uses the specific signals unique to social data in order to retrieve news feeds and associated links and multimedia ahead of users' usage. Our regression-based content prediction model is able to estimate a user's likely content interests 55% of the time. Second, we develop a prefetch scheduling scheme to maximize delay reduction under users' resource constraints. For validation, we apply Earlybird to our collected dataset. We show that on average users can reduce their delays by 62 % at the cost of no more than 3% battery and 40MB/month cellular data.

Categories and Subject Descriptors

C.2.1 [**Computer Communications Networks**]: Network Architecture and Design—*Wireless communication* ; H.2.8 [**Database Management**]: Database Applications—*Data mining*

General Terms

Performance

Keywords

Mobile prefetching; social network; user behavior

MobiHoc'15, June 22–25, 2015, Hangzhou, China.

http://dx.doi.org/10.1145/2746285.2746312.

1. INTRODUCTION

Mobile users spend an increasingly large amount of time on social applications. It is reported that 1/3 of mobile minutes are spent on Facebook and Twitter [2], and the time spent has been increasing 63% year-over-year [6]. Currently 55% of social networking consumption occurs on a mobile device [1].

Social posts have already become a mainstream channel for content consumption. Over 52% and 47% of the users get news from Twitter and Facebook respectively [3]. Whereas each post's text content is small (e.g. limited to 140 characters for Twitter and Weibo), embedded content — photos, videos and URL links — is often much larger. Additionally, emerging social networks such as Instagram (sharing photos) and Vine (sharing video cliplets) are placing greater emphasis on both mobile and multimedia. The increasing popularity and ubiquity of such content consumption calls for exceptional support from mobile devices.

However, the current experience of content consumption is far from satisfactory in terms of access delay and network availability. First, cellular network connection establishment and roundtrip times – at least 3 seconds [4] and often 10 seconds or more [13] – are excessive taxes to fluid and engaging social interaction, especially when users try to retrieve multimedia content from embedded URLs in social posts. Using a popular Android device and stable Wi-Fi connection, we crawled over 150000 links that mobile Twitter users clicked on. The average load time (not including rendering time) of a link is 16.71 seconds, and the median is 7.5 seconds. Compounding access latencies, mobile users often "snack" on social media in gap times such as while in elevators, on public transit, and in large gatherings (e.g., conferences, parties, and sporting events). Such scenarios often correlate to challenging network environments with weaker signals, congested networks, and moving devices. Lastly, the bandwidth cost of multimedia content, especially video, is high and intolerable for many cost-sensitive users on pay-per-bit data plans.

Unique features of mobile social content prefetching Existing work has carefully and extensively studied how to model, predict, and explore favorable network conditions for prefetching. However, social mobile content has its unique features. Consider the Twitter usage of a real-life user from our dataset, shown in Fig. 1. The bars show the numbers of tweets (we only concern about tweets with URL links) the user subscribed to, downloaded, and clicked during each

hour of the day, averaged over 30 days. We observe that a user only download a subset of the tweets she subscribes to (about 40% on average in our dataset, see §*User Behavior*), and time of the day clearly matters. Among the downloaded tweets with URL links, a user clicks only a subset of the links (about 11% on average). Moreover, although not shown in the figure, freshness is critical - 70% of content consumed is within one hour of generation (in our dataset). Therefore, **when** and **what** to prefetch are tightly coupled difficult questions. Specifically, a user does not value his subscribed content equally and thus we need to judiciously decide what to prefetch. In addition, *when* not only includes the classic network-condition-driven prefetching issue, but also user's access pattern. Last, *when* will also affect *what* due to the time-sensitive nature of social content.

One might think that prefetch on WiFi and prefetch at application launch should work well. However when we take a look at representative Twitter users, we see that fresh content attracts most attention, and is largely missed by sparse WiFi connections. Simulation with real-life trace shows that WiFi-only approach can only reduce delay by 5% on average. Therefore, simply prefetch on WiFi does not work well. Also, notice that since WiFi is often not available when our users launch Twitter, prefetch at application launch can be costly in terms of data and battery usage. Therefore, simply prefetch at launch does not work as well.

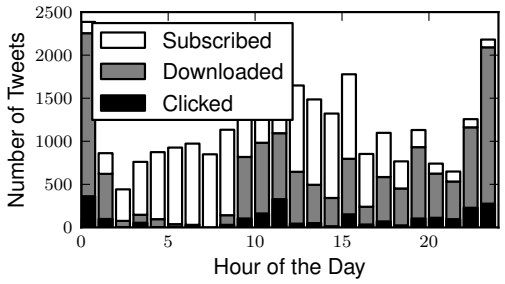

Figure 1: Twitter usage of a real-life user.

Earlybird To address such challenges, we present EarlyBird, a social network content prefetch system. Earlybird estimates user's potential interest in individual links base on history clicks. Such estimation, user access pattern, and dynamic network condition, will jointly decide when and how aggressively the content is prefetched.

First, we build a content access model to estimate a user's interests among the links she downloads. We employ a combination of content metadata, sender metadata, and receiver metadata in order to identify salient features that can be used in a prediction model. Real-life twitter click-through traces from 200 users show 55% accuracy of our prediction model.

Then, Earlybird jointly consider user's interests, access pattern, and dynamic network conditions to decide when and how aggressively the content is prefetched. Please note that, because a user on average only downloads 40% of the subscribed tweets, only using content prediction may result in high false positive, e.g., 2 a.m. to 9 a.m. for the user shown in Fig. 1.

Although prefetch-at-launch and prefetch-on-wifi do not perform well by themselves, integrating them with user's content preference, access pattern, and dynamic network condition, by leveraging the complementary the nature of the two, we can perform effective prefetch that adapt to users' diverse cellular data usage and battery consumption budgets.

Prefetch-at-launch caches URL content right after user launches the Twitter application. Tweets with high likelihood of click-through are prefetched in the reversed chronological order, from new to old. Prefetching at launch time helps filter out the unread tweets and increase the prefetching accuracy, but results in increased cellular data and energy usage. In particular, users tend to snack on social applications on public transport or elevators where network connection is poor or even disconnected.

To alleviate these issues, prefetch-on-wifi caches social content when the prefetching cost is low while taking into account user access pattern. In this regard, we can fully leverage the existing work on network-condition-based prefetching, using techniques including predicting favorable network conditions, using WiFi hotspots, and piggy-backing with other transmissions, etc [9, 27, 21].

By intelligently integrating these two approaches, we can optimize user experience (delay reduction) while accommodating users' different resource constraints, in terms of cellular data usage and battery consumption.

Data and Evaluation As a case study of EarlyBird, we collect user traces from Twidere [8], an Android Twitter client which has over 100,000 downloads and over 6,000 users having consented to reporting usage data to us (§*Data Collection and Statistics*). We then crawled over 30 million tweets in the data set using Twitter API. This provides a realistic evaluation of the practicality of the proposed approach and its corresponding challenges (§*Evaluation*). We show that, for example, on average users can reduce their delays by at least 62% at the cost of no more than 3% battery and 40MB/month cellular data consumption; or with 6% of battery and no extra cellular data consumption, which is a significant improvement over WiFi only approaches.

Contributions In this paper, we identify unique features in mobile social content prefetching, and develop a multi-dimensional approach, Earlybird, that leverages content preference, usage pattern, and dynamic network conditions to improve user experience, under their specific data and energy budgets.

- We build a regression-based content prediction model to estimate a user's potential interest in embedded mobile content.
- We develop a prefetch scheduling scheme that effectively integrates two complementary prefetching mechanisms to maximize delay reduction under users' resource constraints.
- We collect a large set of real-life mobile Twitter traces from over 6000 users, including Twitter application usage, link click, and network availability.
- We analyze the real-life user traces to identify the unique characteristics of Twitter user behavior on mobile devices, and their impact on prefetching.
- As a case study of EarlyBird, we comprehensively evaluate the performance of Earlybird using trace-driven emulations and simulations.

In this work, we use Twitter as a case study, however the techniques can be applied to other social network as well.

For example, Earlybird could benefit Weibo [5], a microblogging service similar to Twitter with 503 million registered users, with little changes. Using the same resource budget, multiple social networks would provide a larger pool of links to choose from, thus increase the prefetching performance of Earlybird.

2. PREFETCH ARCHITECTURE

Fig. 2 illustrates the architecture of EarlyBird. Prefetching is coordinated by a prefetch server, which retrieves all tweets for the given user as they are posted using the Twitter streaming API [7], and determines *what* embedded links in tweets and *when* should be prefetched at the user's mobile device.

The two primary components in EarlyBird architecture are *content prediction* and *prefetch scheduling*. Both components rely on the user profile that contains the historical social content accessed and network conditions encountered. Such information is collected and stored on the mobile device by the data collector, and is uploaded (daily) to the server only when the mobile device is both charging and on WiFi.

Based on the user profile, the content prediction model is trained to predict the likelihood the user will click the URL in a posted tweet if it is downloaded. Tweets are fed to the corresponding user's content prediction model to calculate the likelihood of click-through and are stored in the prefetch queue. Then, the scheduling model decides when to prefetch and which ones to prefetch from the queue, based on estimates regarding future network conditions, battery conditions, underlying network traffic pattern, and when the user will access social content. In addition, priority-based mechanisms, such as TCP-LP [19], can also be set up between the proxy and the mobile device to minimize the impact on other network traffic.

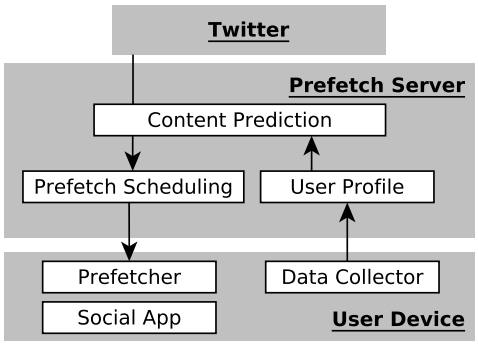

Figure 2: EarlyBird prefetch architecture.

3. DATA COLLECTION AND STATISTICS

To study the real life usage of social media on mobile devices, it is crucial to collect enough usage data from the general public. Although Twitter's tweets are publicly available, when and how users access Twitter is not. Therefore, we collected a large set of usage data from Twidere[1] users who agreed to provide this information to us. With the users' consent[2], we collect usage pattern, link click-through, and network availability. Because existing work has carefully studied the network-condition-based prefetching scheduling using fine-grained traces [27, 21], we focus our work and the corresponding data collection on the complementary components. Therefore, to limit the collection overhead and to avoid user burn out, we decide not to collect detailed network trace in a fine-grain manner, but only log the WiFi availability, which is a major leverage for prefetching. We note that existing traces and techniques for predicting and utilizing favorable network conditions apply here if needed when traces are available.

Data	Format
Launch Client	Timestamp
Download Tweets	Timestamp, Tweets ID list
Link Click	Timestamp, Tweet ID, Link URL
Image View	Timestamp, Tweet ID, Image URL
WiFi Connectivity	Timestamp, WiFi connection
Coarse Location	Timestamp, coordinates

Table 1: Data Collected

Table 1 lists the items collected. To minimize overhead, we only record the tweet IDs that users download and URLs clicked. We then download all the tweet content including links, which is publicly available, from Twitter API [7].

3.1 Data Statistics

In three months, we have collected data from over 6,000 users from all over the world with a diverse demographic composition. The users downloaded 300 million tweets written in 71 languages. The volume and diversity of data reflect the real-life behavior of the mobile social app users. Such realism in data is crucial for understanding and predicting mobile social application network traffic.

Tweets contain URL On average, 29.36% of tweets contain URL links. The percentage of links in tweets varies greatly among users due to their different followings on Twitter. In our prediction algorithm, we are primarily interested in tweets with links because tweets by themselves are simply short character strings. *Therefore, unless otherwise specified, we only consider tweets with embedded links, and only prefetch those links.*

3.2 User Behavior

Twitter access session Users access twitter frequently, but briefly. On average, users spend 17 minutes each time on twitter. However the median access session is only 16 seconds long.

Twitter inter-access time The Twitter inter-access time is the time between two adjacent Twitter accesses. The average inter-access time is more than 6 hours, while the median inter-access time is only 40 seconds. In such bursty accesses where users only access a small amount of content, the network setup and round trip time become a significant overhead. The long average inter-access time also means that

[1]Twidere is an Android Twitter client which has over 100,000 downloads[8].

[2]During the installation and update, the app discloses the information collected and we collect data only if the users choose to opt in. There are less than 15% of users choose to opt in, which indicates user privacy-awareness and the effectiveness of the privacy disclosure. In addition, we only keep a hashed user ID, which is not recoverable through a one-way hash function. Last, twitter contents and social graph are publicly available information.

a user will likely only access a portion of his subscriptions, i.e. tweets generated early during the inter-access time are likely ignored, which needs to be considered in prefetching.

Selective Content Consumption Users exhibit different behavior in terms of how many tweets they read as well as how many links they click on. On average a user downloads 829 tweets with a median of 278. On average, Twitter users only download about 40% of the tweets they subscribe to. Furthermore, users click 11% of the links in the tweet they download from their twitter stream on average, with a median of 6% and a standard deviation of 12.7%. Such selectivity brings challenges to *what* to prefetch.

Time-sensitivity Popular tweets attract interests quickly, then fade away just as fast. The life-span of links on social networks are less than three hours [12]. our data also validates this phenomena on Twitter. The time between posting and clicking of an URL link on Twitter is under one hour 70% of the time, and under 10 minutes 40% of the time. In another word, if prefetching is done one hour before the actual Twitter launch, it will miss at least 70% of the clicked links since they have not been posted yet at that time. Such time-sensitivity of content consumption along with its selective nature introduces great challenges in deciding *when* and *what* to prefetch.

Diurnal access pattern A subset of users show clear diurnal patterns, e.g., the user as shown in Fig. 1. Such diurnal patterns, significant in about 20% of users, can be leveraged to integrate *when* and *what* to prefetch.

4. SOCIAL CONTENT PREDICTION

In this section, we focus on the *what* component of prefetching. In the case of Twitter, the social content is embedded in the tweets as URL links, and we are interested in predicting whether a user will click the URL link *assuming the tweet is downloaded* (i.e., the black in the gray bar in Fig. 1). We build a personalized regression-based model to achieve the goal. First, we select a set of features to represent tweets with embedded links. Then, we use linear regression to build a social content *model* for each user to predict the likelihood of link click-through of the user's future links. This information is used in combination with the *when* component (§*Scheduling Prefetching*) for all possible prefetching.

We have tried several different machine learning algorithms. Regression-based methods are most suitable in our context because SVM methods are sensitive to kernel selection and decision trees are prone to overfitting. For regression based algorithms, linear regression slightly outperforms logistic regression (Fig. 3).

4.1 Initial Feature Set

The number of available features is relatively small in Twitter due to its short style. Therefore, we consider as many features as possible. The features we selected are from simple-to-extract metadata fields of a tweet, summarized in Table 2. There are two natural groups of features, one reflects the properties of a tweet, and the other reflects that of the author.

4.2 Regression-based Prediction

Given a set of k features, each tweet i with link is represented as a k-component feature vector f_i. We denote the binary target variable c_i as whether the user clicked the link in tweet i. We observe that c_i exhibits a coarse linear relationship with the feature vector f_i:

$$c_i = f_i \times W + e \quad (1)$$

where W is the linear coefficient vector, and e is the noise vector.

Message Features	Author Features
Hashtags contained	Name
Other users mentioned	Ghostwriters allowed
Media embedded	Avatar customized
Link position within tweet	Liked tweets count
Geocoordinates if specified	Follower count
Recipient if specified	Following count
Adult maturity rating	Presence in public lists, if any
Retweet count	Total number of tweets
Message length	Verified account status

Table 2: Features selected for click-through prediction

Model Training W is derived using linear regression of the feature vectors of the historical tweets against the observed user clicks during model training. Specifically, we use an L1-norm regularized L2-norm minimization algorithm to calculate the feature coefficients using n historical tweets. Formally, we have:

$$\underset{W}{\text{minimize}} \; ||FW - C||_2^2 + ||W||_1 \quad (2)$$

where L1-norm regularization, i.e., $||W||_1$, is used to prevent over-fitting. Each row of the feature matrix $F^{n\times k}$ is the feature vector of a historical tweet. $C^{n\times 1}$ is the click-through vector. The values of $C^{(i,1)}$ is 1 if the corresponding history tweet is clicked, and 0 otherwise. The coefficient vector for all features is denoted as $W^{1\times k}$. Solving Eq. (2) will give us the coefficient vector W.

Model Inference The product of feature vector f and coefficient vector W is a scalar value called click-through score S:

$$S = f \times W. \quad (3)$$

Ideally, S should be 1 for clicked tweet links and 0 otherwise. In practice, S is a real value roughly in the range of $[0, 1]$, and is proportional to the likelihood of click-through. The click-through score is a critical component in deciding whether a link will be prefetched. It is compared to a threshold value that depends on the *when* component based on access pattern, the time of prefetching, and the cellular data and battery budget. The details will be discussed in §*Scheduling Prefetching*.

4.3 Sequential Forward Feature Selection

We select as many as possible features in the initial set to be comprehensive. However, for each individual user, features without good distinguish power introduce noise in the prediction. Because users are different, we need to select a personalized set of efficient features for each user. We employ a greedy Sequential Forward Selection (SFS) algorithm to select the efficient feature set.

SFS is an iterative algorithm for picking an efficient subset from our initial feature set. SFS starts with an empty set F. In every iteration, for each feature x that is not already in F, we evaluate the prediction performance of feature set $\{x\} \cup F$ using the regression-based algorithm we proposed in §*Regression-based Prediction* based on *balanced accuracy*.

Balanced accuracy is a wildly used metric in the data mining community [11], defined as the average of *sensitivity* (the percentage of correctly predicted tweets that are clicked), and *specificity* (the percentage of correctly predicted tweets that are not clicked). SFS selects the feature set $\{x\} \cup F$ that maximizes the balanced accuracy, and then adds x to F before starting the next iteration. The SFS algorithm stops iterating when the balanced accuracy stops to increase.

4.4 Performance Evaluation

In Table 3, we list the selected features of users using the sequential forward feature selection algorithm. A first look at the popular feature set may be somewhat surprising, e.g., why is Avatar a feature and author not. This highlights the importance of *data-driven* approach; i.e., such features are supported by data instead of assumptions on what should be important. A more careful look also suggests sophistication among features. For example, having a customized avatar attracts user attention, and clearly distinguish the account from new users and bots.

Top Features	User Selected
Media embedded	92%
Geocoordinates if specified	69%
Ghostwriters allowed	56%
Avatar customized	53%
Adult maturity rating	47%
Verified account status	35%
Hashtags contained	34%
Recipient if specified	33%

Table 3: Top features selected by SFS

Fig. 3 shows the overall balanced accuracy of the linear regression prediction algorithm using the efficient feature set selected by SFS for each user. We also implement the same prediction algorithm using the logistic regression for comparison, and find that linear regression achieves slightly better performance.

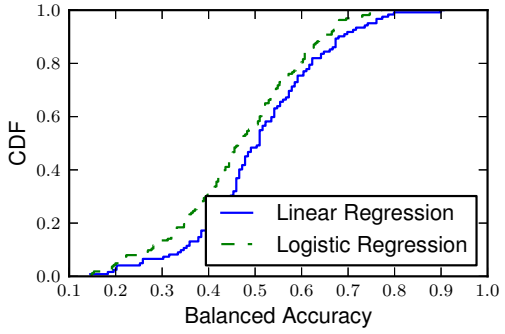

Figure 3: Balanced accuracy of linear regression and logistic regression.

5. SCHEDULING PREFETCHING

The unique features of social mobile content consumption brings interesting and challenging new dimensions in determining prefetch schedules. In particular, real trace analysis (§*Data Collection and Statistics*) suggests that social content consumption has 1) high selectiveness (partially addressed in §*Social Content Prediction*); 2) time-sensitivity; and 3) diverse access pattern. To address these issues, we present a systematic framework that integrates two complementary prefetching components, prefetch-at-launch and prefetch-on-wifi that best leverage content freshness and channel conditions. We adjust the two components to fit to *user-specific cellular data usage and battery consumption budgets*, as discussed in detail next.

Prefetch-at-launch The time-sensitivity of social mobile content suggests prefetch-at-launch, i.e., prefetch right after a user launches the Twitter application. This is the best time to fetch fresh content with high accuracy. However, prefetch-at-launch can be costly as it consumes additional cellular data and energy, and thus needs to be used judiciously. We achieve this goal by selecting an appropriate threshold and time-window. In particular, only links with a click-through score higher than the threshold (based on budget §*Tuning the Thresholds*) will be prefetched. In addition, we only prefetch links in posts within one hour. This time window is suggested by the data analysis in §*Data Collection and Statistics*, as 70% of click-through are generated within one hour. One can adjust these parameters if desired, for example, using different thresholds for different signal strength and using a different time-window. In this paper, we opt for the current simpler design. In addition to data and energy consumption, another challenge that prefetch-at-launch faces is channel condition. Users tend to snack on social applications on public transport or elevators where network connection is poor or even disconnected. This greatly deteriorates user experiences. To address these issues, we consider the following complementary component.

Prefetch-on-wifi Existing work has extensively studied network-condition-aware transmission/prefetching, based on techniques such as predicting and leveraging favorable network conditions, using WiFi hotspots, piggy-backing with other transmissions, and batching, e.g., in [22, 17, 31, 14, 24]. On the regard, we can fully leverage the existing results. In particular, based on earlier studies, we adopt two simple yet effective mechanisms: opportunistic prefetching on WiFi and piggy-backing/batching with existing transmissions. We choose these two for their effectiveness and low overhead, and note that other schemes can be equally adopted.

The main improvement on prefetch-on-wifi we make is to consider the user access pattern. In particular, we define P_t as the probability of a user accesses twitter at time (hour) t. Note that P_t varies significantly over time and across users, illustrated as in Fig. 1. We adjust how aggressive the prefetcher works based on the access probability: the click-through score for each tweet generated by content predictor (§*Social Content Prediction*) is weighted using P_t so that we prefetch more aggressively during active hours and vice versa.

Specifically, prefetch-on-wifi runs only when a WiFi network is available. It wakes up periodically (30 minutes in our setting) and waits for the next piggy-backing opportunity; i.e., transmissions initiated by other applications. Upon detecting such a transmission, the prefetcher checks-in with the prefetch server. The user device notifies the prefetch server with the links accessed by the user since the last check-in which are then removed from the user's queue. The click-through score of each tweet with links in the queue is weighted by $S' = S \times P_t$. The tweets with weighted

click-through score S' higher than a threshold are then compressed and sent to the user device in one batch. Again, the threshold depends on energy budget, and is coupled with the threshold of prefetch-at-launch, which are determined in §*Tuning the Thresholds*.

Fig. 4 illustrates how Earlybird operates. The first timeline shows when the tweets with links (A to K) are posted. The second shows the Earlybird operations: an prefetch-on-wifi operation during WiFi and a prefetch-at-launch one when the user launches the app. Links in tweets C,F and I,J are prefetched by these two mechanisms respectively, with respect to their corresponding thresholds. The third timeline shows the user activity, where prefetched contents in tweets F and I are consumed.

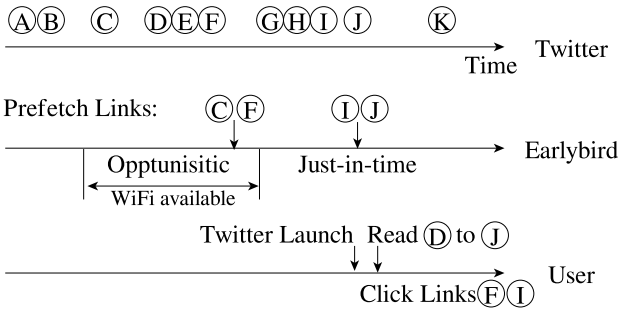

Figure 4: Earlybird operation.

5.1 Tuning the Thresholds

The two components of prefetching have complementary characteristics. Prefetch-at-launch consumes extra cellular data and battery; and prefetch-on-wifi reduces cellular data consumption, and may reduce or increase battery consumption depending on the accuracy of prefetching. Both cellular data and battery are precious resources on mobile devices, and thus must be considered in prefetching. Tuning the thresholds for the two components can adapt the prefetching to the user preference.

Threholds for fixed budget Our objective is to maximize the access delay reduction for each user under its specific cellular data budget and battery consumption constraints. Such constraints are set according to user preferences. In this paper, we take a data-driven approach to solve the above optimization problem. In particular, for each user, for each set of thresholds, we simulate Earlybird on the replay of the training data set. Based on the replay, we obtain the corresponding usage of energy and data, and the delay reduction. We then choose the appropriate threshold values based on the budgets. We note that this computation only needs to run once every few days, and can be done offline in a server, and thus do not impose heavy burdens on mobile.

Thresholds to maximize prefetching accuracy The above approach assumes that a user sets a cellular data and energy budget. Many users may not want to or know how to set such values. Therefore, an alternative way is for the system to select an appropriate threshold, e.g., a threshold that maximize the balanced accuracy in prefetching prediction, and report the cellular/energy consumption to users. As long as such consumptions are acceptable, the user does not need to do anything. We will present evaluation results on both in §*Evaluation*.

6. EVALUATION

In this section, we perform both emulation and simulation evaluations. Emulation evaluations allow us to study the behavior and randomness of users, networks, and batteries in a more realistic setting, including the current limited implementation of caching mechanisms. Trace-driven simulations allow us to evaluate a large variety of parameters and settings, and thus provide a more thorough understanding.

Emulation To realistically evaluate how Earlybird performs on an actual device and network, we run an emulation on an Android Nexus 5 for 60 long-term users from our dataset, each with at least 30 days of traces. They are also users who consumes URL links on Twitter, i.e. access more than 10% of the links in their Twitter streams. We choose these users because they have provided us with long and continuous traces to run statistically meaningful emulations.

We built the emulator as an Android application. The emulator runs on a Nexus 5 Android phone connected to T-Mobile 4G cellular network and also has access to a campus WiFi network. The emulator reads and replays the usage events collected from real-life users, including connecting to or disconnecting from WiFi networks, accessing Twitter, and opening URL links in Tweets. We run three emulations for each user, Earlybird, WiFi-Only, and On-Launch. In each emulation, the same user events are replayed, with the only difference of the caching strategies used. In Earlybird emulation, URL links are prefetched using the Earlybird algorithm. The threshold parameter used in the Earlybird algorithm is set to maximize the prefetching accuracy. In the WiFi-Only emulation, URL links are prefetched only when the phone is on WiFi. In the On-Launch emulation, the most recent 20 URL links are prefetched whenever the Twitter application is launched. The emulator opens URL links using the *WebView* provided by the Android framework. The WebView provides a notification when the web page has finished loading which is used by the emulator to calculate the loading delay of an URL link. We have observed that this notification is not precise, especially when there is dynamic content loaded using javascript in a web page. In most cases, the notification is delivered prematurely which leads to a conservative estimate of the loading delay. The emulator also relies on the Android WebView's caching mechanism to prefetch URL links. When prefetching a link, the emulator simply opens the link so that the content from the links can be cached by the WebView. We note that the implementation of WebView caching mechanism is limited. Ideally, when a user accesses the cached content is cached, the network traffic and network delay should reduce to zero. However, in the current implementation of WebView, we observe that 54.7% of network traffic for cached content. Therefore, when caching implementation improves, the benefit of prefetching would further increase.

Fig. 5 shows the delay reduction and total cellular data usage of each emulation. Without caching mechanism, users on average consume 1.1 MB per day on content. Earlybird reduces delay by 63.6 seconds per day and uses 1.25 MB per day on average, including both the necessary content network consumption and prefetch overhead. On-Launch reduces 84.3 seconds delay per day and uses 3.21 MB per

day. WiFi-Only emulation reduces delay by 7 seconds by day, but due to its low prefetching accuracy, it still uses 0.7 MB data per day, which is 0.4 MB smaller than that without prefetching. Fig. 6 shows the prefetching accuracy, volume, and effectiveness. Earlybird has a significantly better prefetch hit ratio, 48.9% on average, WiFi-Only has only 6%, and On-Launch has 24%. One limitation of the Android platform is that it only provides battery usage at percentage level. It's difficult to account for the energy consumption of each user and link. Therefore, we report the aggregated behavior here. For all of the users in the emulations, Earlybird uses 4.2 times of the total battery of Nexus 5, WiFi-Only uses 2.1 times, and On-Launch uses 6.1 times.

Trace-driven Simulation To evaluate the algorithm on a larger data set, and under multiple budget constraints, we run a trace-driven evaluation for 224 long-term users from our dataset, each with at least 30 days of traces.

The network trace only shows network availability, i.e., WiFi or cellular. We measured the load time and total size of each link in the dataset using an Android crawler we wrote. We measured on both 3G and WiFi networks in our lab. The average load time (not including rendering time) for links is 16.71 second, and the median is 7.5 second.

We use the energy model presented in previous work [10] to calculate the energy consumption of prefetching each link base on the link size.

Baselines For comparison, we also consider three prefetching benchmarks: 1) WiFi-only: every 30 minutes if on WiFi is available; 2) On-launch: when the app launches. (This is the most aggressive version of prefetching, as done in many current app implementations, as well as the content prefetching part of [23, 25]); and 3) 30-min: every 30 minutes regardless of network (this is to emulate default settings in some email/news applications). We note that 1) and 2) can also be considered as special cases of EarlyBird where we set the thresholds to $(0,\infty)$ and $(\infty,0)$, respectively.

Under Budget Constraints We first evaluate the performance of Earlybird under three different budget constraints: (0.5%, 600MB), (3%, 300MB), (6%, 0MB), where the former is the percentage of battery allowed per day and the latter is the total cellular budget per month. These three sets of constraints represent different user preference on energy and cellular data consumption. We also compare with the three baseline schemes.

In Fig. 7, we show the CDFs of users in terms of delay reduction (per day), data usage (per month), and battery consumption (per day). Overall, Earlybird under (3%,300MB) and (6%,0MB) shows very good delay reduction, while operating within their corresponding resource budgets. The results of (6%,0MB) might be surprising at the first glance as it shows significant delay reduction. We note that this highlights the complementary nature of prefetch-at-launch and prefetch-on-wifi. Although the total data budget is zero, because prefetch-on-wifi reduces cellular reduction, which can be used for prefetch-at-launch, and thus jointly significantly reduce the access delay for many users. On the other hand, (0.5%,600MB) shows rather limited improvement - primarily due to the tight energy budget. The time between posting and clicking of an URL link on Twitter is under one hour 70% of the time, and under 10 minutes 40% of the time. WiFi-only scheme often misses the window of opportunity since it is restricted by when the device connects to WiFi.

In Fig. 8, we show the details of the prefetching algorithms. Clearly, on-launch shows the best prefetch hit ratio and effectiveness, as its timing is perfect. WiFi-only shows the lowest effectiveness, which explains its limited performance in delay reduction. We also note that (0.5%,600MB) caches a much smaller number of links per day due to its tight energy budget, and thus its limited delay reduction.

The impact of thresholds Next, we carefully evaluate the impact of prefetching threshold used in Earlybird. We do not include baselines here as they do not use thresholds. To show the tradeoff between energy, data, and delay, we compare the following click-through threshold: (0.1,0.1), (0.5,0.5), (0.9,0.9). Intuitively, a lower threshold allows the user device to prefetch more content. Fig. 9(a) shows the number of tweets prefetched, where each curve represents a different threshold. On average, we prefetch 89 links per day for each user. Fig. 9(b) shows the prefetch hit ratio, which is the percentage of the prefetched tweets that are actually clicked by the user. The prefetch hit ratio is 12.5% on average. Fig. 9(c) shows the percentage of link clicks that are prefetched. Clearly, a lower threshold allows more content to be prefetched and thus increases the percentage of link clicks that are prefetched, at the cost of lower hit ratio. This figure clearly demonstrates the trade-off between unprefetched ratio and hit ratio.

The corresponding effects on user experience are shown in Fig. 10. Fig. 10(a) shows the delay reduction per day enabled by prefetching. Fig. 10(b) shows the cellular data usage of different thresholds. At (0.5,0.5), prefetching only uses 43 MB cellular data every month on average for each user, where the average total data volume is 890 MB for all the links each user clicked. Last, Fig. 10(c) shows the additional battery used by prefetching. The average battery consumption ranges from 0.7% to 4.3%, corresponding to threshold (0.9, 0.9) to (0.1, 0.1). Clearly, a lower threshold more effectively reduces access delay and cellular data usage, at the cost of slightly higher energy consumption. It worths highlighting that the energy consumption is low in our scheme - this illustrates the importance of piggy-backing and leveraging better network conditions (i.e., WiFi in this evaluation) in conjunction with content prediction.

	Earlybird (0.5, 0.5)	**on-wifi**	**at-launch**
Delay Reduction (Seconds)	78.04	34.81	72.58
Delay Reduction (%)	62.04	30.21	59.11
Battery Used (%)	2.59	2.1	2.41
Extra Data (MB)	43.81	-2.71	57.99
Hit Ratio (%)	12.54	8.49	12.39
Prefetched Ratio (%)	39.77	15.65	37.04

Table 4: Effect of prefetch-on-wifi and prefetch-at-launch.

Interaction between prefetch-on-wifi and prefetch-at-launch The results summarized in Tab. 4 highlight the complementary nature of prefetch-at-launch and prefetch-on-wifi. We compare the average performance of Earlybird (0.5, 0.5) with that of prefetch-on-wifi only with threshold 0.5 (equivalent to Earlybird $(0.5, \infty)$), and prefetch-at-launch only with threshold 0.5 (equivalent to Earlybird $(\infty,$

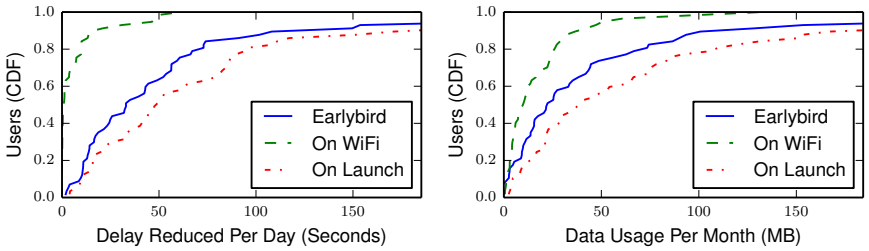

Figure 5: Prefetch emulation cost and benefit: delay reduction, cellular data usage.

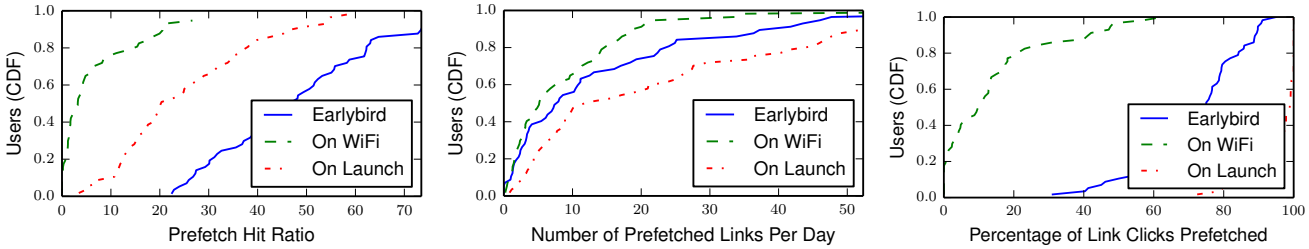

Figure 6: Prefetch emulation accuracy: efficiency, volume, and effectiveness.

0.5)) The combination of the two has higher hit ratio and prefetched ratio, and at the same time uses less data at the same time, comparing to either approach alone.

Impact of Training Set

In simulation, we use a subset of each user's data as the training set to build the prediction model and the scheduling policy, then use the others for performance evaluation.

First, we try several fixed sizes for the training set. Users download different amount of links per day, and only click a small portion of them. To avoid building a biased model, we make sure that the training set contains enough number of clicked links, and use the number of clicked links as the size of th training set. We tried the following training set sizes: 10, 50, 90, 130, 170.

We then compare the fixed training set with a rolling training set. For a rolling training set, the first training set contains 50 clicks is used in the beginning. The resulting model is used to predict the click-through of the following tweets, which is the test set, until we see 50 clicks in the test set. Then the previous model is discarded, and we use the test set, which also contains 50 clicks, to train a new model. This process is repeated until we processed all the tweets.

The average balanced accuracy increases from 71.9% to 77.4% as the training set size increase from 10 to 170. However, using a rolling training set of size 50, the average balanced accuracy is 80.8% which is higher than any fixed training set. The result indicated that user preferences do changes over time, using recent history data as training set yields higher training accuracy.

7. RELATED WORK

Prefetching on Mobile Systems In [15], the authors present an architecture for mobile prefetching where IMP (Informed Mobile Prefetching) is structured as a library to which any mobile application may link to. In IMP, it assumes that applications provide precise information on "what" to prefetch. We focus on the unique features of mobile social content, to address "what" and the interaction of "what" and "when". In [25], the authors show that inappropriate prefetching can be costly to users. They implement and evaluate Procastinator that decides prefetching based on a few factors including whether the user is on WiFi or cellular network, the status of the user's data plan, and whether the object is needed at the present time. The implementation achieves "zero-effort", i.e., it requires no developer effort, no source code, and no OS changes. The technique developed can be leveraged to in the implementation of Earlybird. In [23], the authors proposed an app prefetching algorithm, which predicts when and what application will be launched on mobile phone and prefetch applications accordingly. The content prefetching algorithm is similar to the prefetch-at-launch in this work, which is compared in §*Evaluation*.

Recently, CDN-based approaches have been proposed by researchers to increase QoE for web services, e.g. in [29, 26]. The authors propose to use social network relationships, read access patterns to efficiently distribute content geographically to lower bandwidth costs and increase QoE. Typically, CDN-based solutions do not deal with challenges that are special to wireless networks and personalized usage patterns.

Also relevant to our work is user profile and and how to leverage them for efficient transmission such as prefetching. Both Wiffler [9] and Bartendr [27] consider the setting of vehicular systems to offload cellular data traffic to either WiFi networks or to time-instants when signal strength is stronger. Breadcrumb [21] and smarttransfer [30] address the issue of predicting future network condition and user profile. We build upon such network-aware techniques while focusing on the unique properties of mobile social content.

Prefetching in general is well-studied under different context. In [22, 17], the authors explored user access prediction and prefetching for file systems and databases. [31, 14, 24] addressed the theoretical problem in prefetching such as cost benefit analysis.

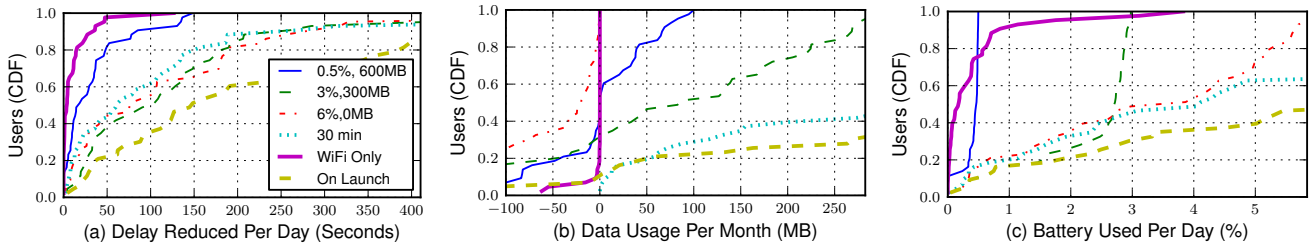

Figure 7: Prefetch cost and benefit: delay reduction, cellular data usage, and energy consumption.

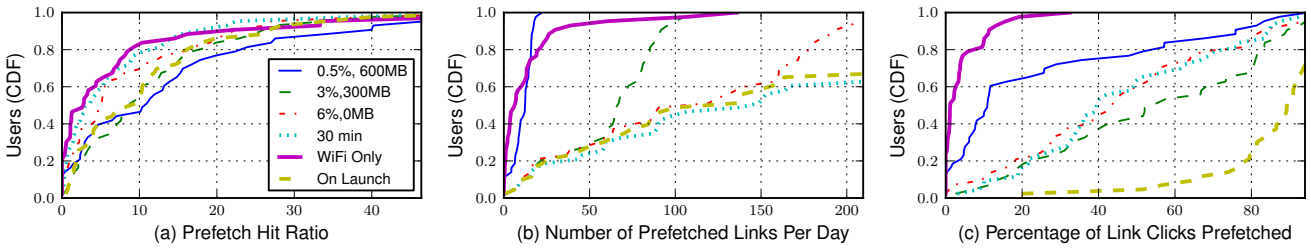

Figure 8: Prefetch accuracy: efficiency, volume, and effectiveness.

Twitter studies In [16, 20], the authors study the basic social graph such as following relationship, and statistics of Twitter, including number of user, posting and retweet. Factors that impact retweeting is also explored. Suh, et al. discussed the factors that affected the retweeting or retweetability of a tweet using PCA and generalized linear model [28]. Kunegis et al. also studied the likelihood of retweeting [18] based on content features. They only used 7 features to start with, and the effective features selected are quite different from us, showing that retweet and content access have very different models. We are the first to use the click-through dataset and build a content-prediction model for mobile networking.

8. CONCLUSION

Mobile users are consuming increasingly large amount of embedded content through social applications. Social content prefetching is a promising technique for improving user experience on mobile devices. By identifying the unique features of mobile social applications, we propose a unified mobile social prefetching framework, Earlybird, that jointly optimizes the *what* and *when* in prefetching, leveraging content preference, user behavior pattern, and dynamic network conditions. The proposed scheme can adapt to user-specific budget constraints in terms of (extra) energy and cellular data, with the objective of maximizing delay reduction. We use a data-driven approach to develop personalized content model and control variables (prefetching thresholds) to achieve this goal. Evaluation using real-life Twitter traces show promising performance of Earlybird. In addition to EarlyBird's immediate utility for Twitter users, we speculate that our approach of unifying social and network access patterns can be generalized to other social networks as well.

9. REFERENCES

[1] 2013 Mobile Future in Focus. `http://www.comscore.com/Insights/Presentations_and_Whitepapers/2013/2013_Mobile_Future_in_Focus`.

[2] Mobile apps: We interrupt this broadcast. http://blog.flurry.com/bid/92105/Mobile-Apps-We-Interrupt-This-Broadcast.

[3] News use across social media platforms. `http://www.journalism.org/2013/11/14/news-use-across-social-media-platforms/`.

[4] Public dataset: Google mobile network measurements. https://dev.twitter.com/.

[5] Sina weibo. `http://en.wikipedia.org/wiki/Sina_Weibo`.

[6] Social media report 2012: Social media comes of age. http://www.nielsen.com/us/en/newswire/2012/social-media-report-2012-social-media-comes-of-age.html.

[7] Twitter api. https://dev.twitter.com/.

[8] Twidere for twitter. `https://play.google.com/store/apps/details?id=org.mariotaku.twidere&hl=en`, 2013.

[9] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting mobile 3G using WiFi. In *Proc. of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*, pages 209–222, San Francisco, CA, June 2010.

[10] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: A measurement study and implications for network applications. In *Proc. of ACM SIGCOMM Internet Measurement Workshop (IMC'09)*, pages 280–293, Chicago, IL, November 2009.

[11] P. Baldi, S. r. Brunak, Y. Chauvin, C. A. F. Andersen, and H. Nielsen. Bioinformatics review. 16(5):412–424, 2000.

[12] Bit.ly. You just shared a link. how long will people pay attention? `http://blog.bitly.com/post/9887686919/you-just-shared-a-link-how-long-will-people-pay`.

[13] D. Chu, A. Kansal, J. Liu, and F. Zhao. Mobile apps: it's time to move up to condos. In *Proceedings of the 13th USENIX conference on Hot topics in operating systems*, HotOS'13, 2011.

[14] V. de Nitto Personè, V. Grassi, and A. Morlupi. Modeling and evaluation of prefetching policies for context-aware information services. In *Proceedings of the 4th annual ACM/IEEE international conference*

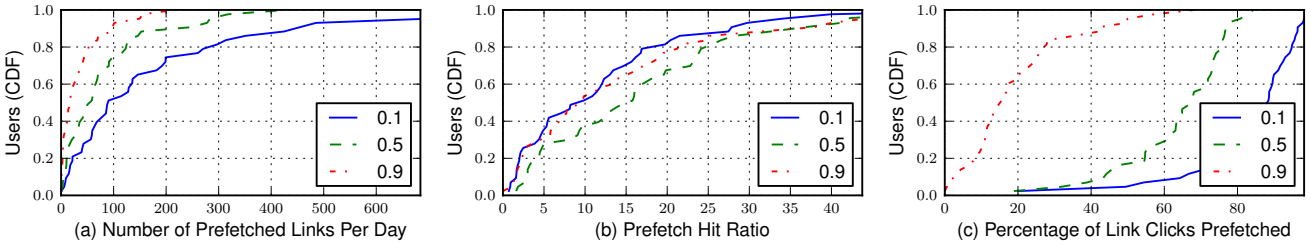

Figure 9: Prefetch accuracy: volume, efficiency, and effectiveness.

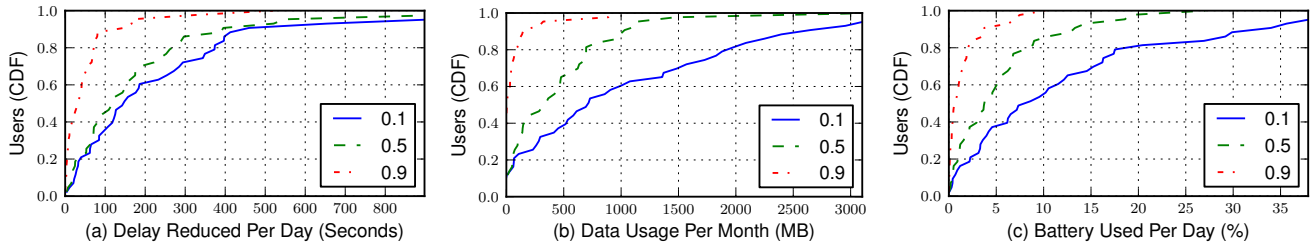

Figure 10: Prefetch cost and benefit: delay reduction, cellular data usage, and energy consumption.

on Mobile computing and networking, pages 55–65. ACM, 1998.

[15] B. D. Higgins, J. Flinn, T. J. Giuli, B. Noble, C. Peplin, and D. Watson. Informed mobile prefetching. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, MobiSys '12, pages 155–168, New York, NY, USA, 2012. ACM.

[16] A. Java, X. Song, T. Finin, and B. Tseng. Why we twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, pages 56–65. ACM, 2007.

[17] D. Kotz and C. S. Ellis. Practical prefetching techniques for parallel file systems. In *Parallel and Distributed Information Systems, 1991., Proceedings of the First International Conference on*, pages 182–189. IEEE, 1991.

[18] N. N. T. G. J. Kunegis and A. C. Alhadi. Bad news travel fast: A content-based analysis of interestingness on twitter. 2011.

[19] A. Kuzmanovic and E. W. Knightly. Tcp-lp: A distributed algorithm for low priority data transfer. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, pages 1691–1701. IEEE, 2003.

[20] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600. ACM, 2010.

[21] A. J. Nicholson and B. D. Noble. Breadcrumbs: forecasting mobile connectivity. In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, pages 46–57. ACM, 2008.

[22] M. Palmer and S. B. Zdonik. *Fido: A cache that learns to fetch*. Brown University, Department of Computer Science, 1991.

[23] A. Parate, M. Böhmer, D. Chu, D. Ganesan, and B. M. Marlin. Practical prediction and prefetch for faster access to applications on mobile phones. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 275–284, New York, NY, USA, 2013. ACM.

[24] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. *Informed prefetching and caching*, volume 29. ACM, 1995.

[25] L. Ravindranath, S. Agarwal, J. Padhye, and C. Riederer. Give in to procrastination and stop prefetching. In *ACM HotNets XII*, 2013.

[26] S. Scellato, C. Mascolo, M. Musolesi, and J. Crowcroft. Track globally, deliver locally: Improving content delivery networks by tracking geographic social cascades. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 457–466, New York, NY, USA, 2011. ACM.

[27] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. Padmanabhan. Bartendr: A practical approach to energy-aware cellular data scheduling. In *Proc. of ACM Annual International Conference on Mobile Computing and Networking (MobiCom'10)*, pages 85–96, Chicago, IL, September 2010.

[28] B. Suh, L. Hong, P. Pirolli, and E. H. Chi. Want to be retweeted? large scale analytics on factors impacting retweet in twitter network. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pages 177–184. IEEE, 2010.

[29] S. Traverso, K. Huguenin, I. Trestian, V. Erramilli, N. Laoutaris, and K. Papagiannaki. Tailgate: Handling long-tail content with a little help from friends. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 151–160, New York, NY, USA, 2012. ACM.

[30] Y. Wang, X. Liu, A. Nicoara, T.-A. Lin, and C.-H. Hsu. Smarttransfer: transferring your mobile multimedia contents at the right time. In *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video*, pages 71–76. ACM, 2012.

[31] T. Watson. Application design for wireless computing. In *Mobile Computing*, pages 363–373. Springer, 1996.