2016 IEEE 36th International Conference on Distributed Computing Systems

RichNote: Adaptive Selection and Delivery of Rich Media Notifications to Mobile Users

Md Yusuf Sarwar Uddin[1], Vinay Setty[2], Ye Zhao[1], Roman Vitenberg[3], and Nalini Venkatasubramanian[1]

[1]Department of Computer Science, University of California Irvine, CA, USA, {yusuf.sarwar,nalini,yez}@uci.edu
[2]Max Planck Institute for Informatics, Germany, vsetty@mpi-inf.mpg.de
[3]University of Oslo, Norway, romanvi@ifi.uio.no

***Abstract*—In recent years, notification services for social networks, mobile apps, messaging systems and other electronic services have become truly ubiquitous. When a new content becomes available, the service sends an instant notification to the user. When the content is produced in massive quantities, and it includes both large-size media and a lot of meta-information, it gives rise to a major challenge of selecting content to notify about and information to include in such notifications.**

We tackle three important challenges in realizing rich notification delivery: (1) content and presentation utility modeling, (2) notification selection and (3) scheduling of delivery. We consider a number of progressive presentation levels for the content. Since utility is subjective and hard to model, we rely on real data and user surveys. We model the content utility by learning from large-scale real world data collected from Spotify music streaming service. For the utility of the presentation levels we rely on user surveys. Blending these two techniques together, we derive utility of notifications with different presentation levels. We then model the selection and delivery of rich notifications as an optimization problem with a goal to maximize the utility of notifications under resource budget constraints. We validate our system with large-scale simulations driven by the real-world de-identified traces obtained from Spotify. With the help of several baseline approaches we show that our solution is adaptive and resource efficient.

I. Introduction

In recent years, notification services have become truly ubiquitous. Today, we receive contextualized notifications about events and items of interest from service providers, news agencies, social media websites, community services, vendors of goods and services etc. For example, we can get a notification on Facebook when a friend uploads a new photo or updates her profile. On Spotify, a notification can be sent when a friend starts streaming a music track, or edits a shared playlist [1]. Mobile app developers can opt to deliver notifications to devices using standardized notification services, e.g., those provided by Apple and Google.

Scalability is a significant challenge given the staggering amount and volume of notifications being generated today: In Spotify, daily bandwidth consumption due to transferring events from the client software and delivering notifications to millions of users is around 2TB [2]. A significant fraction of these notifications are being delivered to mobile devices, which have constraints related to bandwidth availability, residual battery capacity and data plan costs. Many content items, in particular, those associated with media content such as video, pictures, or audio, are large. They are also rich in terms of the amount of metadata used to describe the content: a music stream can be accompanied by a plethora of information about the genre, artist, song, album, popularity, social friends who listened to the same stream, etc.

Unfortunately, notification providers today often resort to delivering a short message indicating that content is available (without much information about the specifics) for all the items that seem remotely relevant. Such a service expects users to select the relevant content, initiate a session with the provider and pull the content as desired. This approach is deficient in several respects: (a) the sheer number of notifications can overwhelm the user, (b) a barebone notification may not provide sufficient information for the user to make an intelligent selection, (c) the delayed pull-based approach generally results in higher latency, and (d) simplistic delivery schemes do not exploit connectivity information for improved scheduling (such as periods of lower loads or efficient usage of dataplan minutes). These factors call for selective delivery with respect to both the choice of items in the notification and specific associated information.

In this paper, we propose ***RichNote***, an end-to-end framework that addresses the problem of delivering *rich notifications* to users in constrained settings, e.g. in mobile environments. *Rich notifications* are notifications augmented with enhanced content (including metadata, degraded content and partial content snippets). This is especially useful for media information related to social interaction, e.g., in Spotify, and information accessible through social network sites, e.g. audiovisual content in Facebook. The pushed notifications may include any of a multitude of media presentations that can be scaled in a variety of well-known ways – thumbnails of album cover images, previews of video or audio streams, a random track for an album, etc. Scalable encoding can be employed to degrade the quality of media content.

There are fundamental challenges that must be addressed to realize the above vision, prior to embarking on a large scale implementation. A key question is that of balancing the selection of content items for delivery with the selection of representation for each item. Even for a fixed number of content items and a small number of predetermined discrete presentation levels, the question of what to deliver and how involves nontrivial decisions: Is it better to deliver a homoge-

1063-6927/16 $31.00 © 2016 IEEE
DOI 10.1109/ICDCS.2016.107

neous presentation level across all the items or include concise presentation for most items while focusing on extended presentation for a selected few? Can we quantify how the utility decays over time if the notification is delayed? How does this vary across media types–images, music, video streams etc?

RichNote, uses a multi-step approach to answer the above questions and address the creation and delivery of rich notifications of social media contents to mobile devices. In the first phase, we consider the **selection problem** from the end user perspective to maximize mobile user satisfaction under resource constraints (e.g. bandwidth, energy, storage, dataplan cost). We develop a comprehensive model of user satisfaction that combines content utility (strength of user interest in a particular content item) with presentation utility (i.e., enhanced quality of user experience due to rich notifications) by combining a range of factors from user-specified preferences, social network ties, content parameters, etc. The outcome is a ranked list of notifications for delivery. The second phase deals with the delivery of the items in the ranked list. We develop **adaptive scheduling** techniques to determine when and how to deliver rich notifications content given the underlying dynamics of the user, device and network and develop optimized scheduling methods to adapt these plans as conditions in the network and mobile device change. We transform the above steps into corresponding optimization problems and develop efficient algorithmic solutions for them. While the selection model and scheduling techniques can be applied to any content type and any notification service and setting, we consider its application to a popular large global-scale platform that is inherently rich media, namely the Spotify music streaming service (discussed further in Sec. II).

To illustrate the effectiveness of the *RichNote* method, we evaluate it using de-identified production traces from Spotify. As the baselines, we employ two notification delivery methods commonly used in the industry–FIFO and UTIL (highest utility first). We measure and compare several performance metrics capturing quantity and utility of the delivered notifications as well as energy efficiency and latency of *RichNote* and the baseline methods. We demonstrate that *RichNote* always delivers close to 100% notifications at different presentation levels while the baseline methods suffer due to their fixed presentation level. We then show that *RichNote* doubles notification utility value compared to the baseline methods without compromising energy consumption and queuing delay. We also show that unlike baseline methods, *RichNote* adapts to different settings such as available data budget, change in battery status, dynamic network availability, to maximize notification utility.

The unique aspect of our contribution is that we merge the pub/sub paradigm with rich data delivery and mobile systems into a single framework, *RichNote*, in a manner that exploits the nature of social media content. Today's social media systems incorporate pub/sub technologies and content delivery methods; albeit as independent, dissociated tasks. We conjecture that a more holistic approach that combines these tasks will result in improved systems that support delivery of more relevant content to users and flexibly adapt to the underlying system constraints. *RichNote* has the ability to scale content granularity and adapt content presentation – this allows us to diversify content presentation and staging in a manner personalized for the user. Furthermore, existing methods for content ranking and summarization are either offline (information retrieval) or deal with data items that have a simple predefined structure (data streams). *RichNote*, in contrast, performs efficient online analysis of rich media information and creates enriched content for notification delivery. In summary, key novel elements of this work include:

- A structured approach to modeling utility of rich notifications both in terms of relevance of content to the user and its presentation quality and a semantics-based content selection approach (Sec. III).
- Design of the *RichNote* architectural framework and algorithms that combine scalable content transformation techniques with subscription-based data selection and adaptive notification delivery methods (Sec. IV).
- Validation of the *RichNote* framework/algorithms using real-world de-identified traces from the Spotify platform (Sec. V).

II. Use Case: Rich notifications in Spotify

To lend focus to our techniques, we apply and evaluate the *RichNote* two-phase approach to a popular music streaming service, Spotify, which utilizes notifications to inform subscribers about the availability of potentially interesting content. Streaming music delivery services, in general, and Spotify, in particular, serve as an interesting use case for rich notifications due to its unique characteristics. Firstly, music, as a modality of content allows for perpetual content playback - people listen to music while conducting other day-to-day activities. This results in significant resource demands on the infrastructure. Second, the choice of music is very personalized in multiple dimensions – genre, artist, activity etc. This implies the need for customizing the rich notifications based on specific user preferences. Finally, the library of potentially accessible and available content is huge and growing. Matching large numbers of content items to the plethora of users who want continuous feeds is a challenge.

Today, Spotify is known to use the topic-based pub/sub paradigm for delivering notifications arising from music-associated social interaction among its users. The topics may correspond to users friends, artist pages or publicly available music playlists [1]. The publications for these topics are notifications about friends listening to music tracks, new album releases, and updates to followed playlists respectively. To deliver these diverse types of publications, Spotify has deployed a hybrid pub/sub engine that allows notification delivery to users in two different modes:

Real-time mode: Social notifications such as friend feeds are delivered in real-time so that users can receive them instantly in the Spotify client software. Real-time processing is needed because: (1) updates are frequent and large in number compared to other publications (2) they provide an opportunity

for users to take instant actions based on activities of friends. A large number of real-time notifications will cause information overload for human users [3]; methods for selecting a subset of notifications in an efficient manner have been proposed in prior work [3]. While these methods address information overload to some extent, the actual utility of these notifications to end-users is not considered.

Batch mode: Notifications about album releases and playlist updates are relatively small in number and less frequent and hence are good candidates for delivery in batch mode. The batch mode also facilitates analysis of notifications for computing utility and selective delivery to the users, for offline viewing. However, these notifications contain textual metadata with remote links to media content; offline users may not be able to view them. The ability to provide rich notifications that include media content samples in addition to textual metadata enables richer experiences during offline viewing.

RichNote improves the existing Spotify notification scheme in two ways. First, we address the inherent utility/timeliness tradeoff between the two delivery modes - while the real-time mode approximates utility computation because it is computationally intensive, the batch mode facilitates utility-driven delivery that is less frequent. *RichNote* incorporates a *round-based* model for notification delivery where notifications are analyzed, selected and delivered in discrete time frames called *rounds* - this provides a middle-ground between the real-time and batch modes and allows us to tune time duration of each round proportional to the frequency of the feed. For example, friend feeds can be delivered every few minutes whereas notifications related to artist and playlists can be delivered in every few hours. Second, the textual presentation mode of Spotify notifications has limited utility to users. *RichNote* alleviates this by introducing the concept of rich notifications that enables notifications to be composed with different presentation levels. This process is driven by the utility of the content to users, quality of presentation level and mobile resource limitations such as cellular data budget. For example, a notification about a music track from a favorite artist can be enriched with 20 second preview of the song which is readily available for the user provided there is sufficient data budget.

Using trace data from de-identified Spotify logs (notifications and user response to these notifications), we provide an analysis of utility and derive the probability of a user being interested in an item. The resultant model is used to select and create the rich notification content. Furthermore, since the Spotify client is not open for modifications such as creating and delivering enriched notifications, we perform utility assessment of a presentation form in a rich notification by conducting separate subjective user studies.

III. Utility Modeling and Problem Formulation

We consider the problem of selective delivery of notifications from the end user perspective rather than that of a service deployment. The selection is customized for each user individually. As motivated in Section II, we assume that the content is generated and becomes available for the user in discrete time frames, referred to as *rounds*. The content consists in many items, a selection of which needs to be delivered to the user on a mobile resource-constrained device. Each content item may be represented in a notification at a selected presentation level. Presentation levels also trade the balance between utility for the user and resource constraints. The main problem we address is—given a set of candidate content items, find a subset of the items and their respective presentation levels such that the overall utility is maximized. To address the challenges in delivering large-scale rich multimedia notifications in social interaction systems such as the one deployed by Spotify in conjunction with its music streaming service, we need to solve the three main problems: (1) notification utility modeling, (2) modeling notification presentation levels, (3) notification selection and delivery to maximize the utility under resource constraints.

A. Utility modeling

In this paper, we consider rich notifications for multimedia content. The utility of these notifications depends on the utility of the content to the users (*content utility*). Since the notifications are delivered in different presentation levels, we assume that each presentation level has a different utility value to the user (*presentation utility*). In this section, we elaborate on these two utility models and explain how we combine them.

We denote utility of a notification as $U(i, j)$, where i is the content item, and j is the presentation level (we assume that the presentation has discrete levels). The content utility of item i denoted as $U_c(i)$, corresponds to the relevance of the content to the user derived from the content attributes; it is thus independent of the presentation. The goal of the content utility is to quantify the probability that a user is likely to consume a given content item. The presentation utility $U_p(i, j)$ quantifies the value derived from a specific presentation of the content. Presentation utility depends on the metadata as well as the duration and quality of the multimedia content included in the notification. For example, music content can be presented with a 5-second or a 10-second preview and using different bit rate sampling. Similarly, video samples can also be presented in combinations of duration and quality. Since presentation utility is subjective; in this paper, we obtain it from a user survey.

Given content and presentation utility values, we define the combined utility as follows:

$$U(i, j) = U_c(i) \times U_p(i, j) \tag{1}$$

We now look into how we can compute these two utility components for each content item.

Content utility. $U_c(i)$ can be defined as a measurement of how likely the user would be interested in consuming content i given the user's current context and content attributes. This probability depends on the user's explicit preferences, various attributes of the content, popularity of the content across user base, and the relationship (e.g., social rank) with the user who generated the content. It may also depend on the recency of the content (aging factor). All of these factors form a *feature*

space. Given a feature space with appropriate training data we can train a machine learning classifier to predict the utility of the content. In Section V-A we explain the specific classifier model and feature space we consider for the Spotify use case.

Presentation Utility. As mentioned above, the same content has different utilities for different presentations. Utility function, $U_p(i, j)$, captures the level of satisfaction of the j-th presentation of content i with respect to a base presentation, say, for example, against the full content. We model utility for a given presentation as a real-valued function dependent on the inclusion of metadata as well as a subset of the original full content.

The utility function always exhibits the following properties:

- Utility should monotonically increase for richer presentation levels (higher j). This is because intuitively, as more details are added to the presentation, utility should also increase. The reason for this is "information never hurts". We also assume in our case no content is unwanted for a given user.
- Utility should follow "diminishing returns", i.e., the gain in utility for including specific details decreases as the level of presentation increases.

Obviously, the utility function is highly application-dependent. We elaborate on the specific presentation utility model we use in the Spotify use case in Section V-B.

B. Presentation levels

We assume that content i can be presented or notified to the user at any of several available presentation levels, namely at levels $1, 2, \cdots, k_i$, where level 1 represents the smallest possible presentation with only a set of essential metadata and without any media sample. Successive presentations then have *strictly* higher presentation levels enriching the presentation content accompanied by some part of the media content in question. The presentation at level k_i represents the *largest* possible presentation for an individual content notification. Notifications gradually become richer and larger in size when additional pieces of information are added to the presentation. The pieces of information that accompany a certain content to enrich the notifications are application specific, and we assume that a certain "generator" exists that produces these presentations at different level of details. Different generators may exist for different content types, which are developed by the content providers. We assume that these presentations are *strictly ordered* in their sizes and utility. We also define level 0, which means no presentation at all (with zero size and zero utility). This level represents the case when the notification for this content is not sent to the user.

C. Selection and scheduling problem

Our objective is to choose content presentations so as to maximize overall utility subject to two types of resource constraints—data and energy budget. At each round, data budget limits the number of bytes to deliver while energy budget specifies how much energy the user device is allowed to spend. Let $X(t)$ be the set of items that are to be selected at round t. Denote by η a mapping from a content item i to the presentation level chosen for i. $\eta(i)$ is 0 if content i is not chosen at all. The utility of selected items is therefore, $U_t = \sum_{i \in X(t)} U(i, \eta(i))$. Let $s(i, j)$ denote the size of j-th presentation of content i and $\rho(i, j)$ denote the estimated amount of energy required to download item i at presentation level j by the mobile user (following a specific energy model).

Given each user having a data budget, $B(t)$, and energy budget, $E(t)$, for the current round, the problem of notification selection is to find presentation for each content item, that is mapping η, so as to

$$\text{maximize} \quad U_t \tag{2a}$$

$$\text{subject to:} \sum_{i \in X(t)} s(i, \eta(i)) \leq B(t) \tag{2b}$$

$$\sum_{i \in X(t)} \rho(i, \eta(i)) \leq E(t) \tag{2c}$$

The above is an instance of the multi-choice Knapsack (MCKP) [4] problem with two weight constraints. Following MCKP terminology, presentations are object *categories*, utility corresponds to *profits*, and presentation sizes and energy values map to *weights*. The problem is reportedly NP-Hard [4], even with a single weight constraint. We advance the intuition from that work and develop a greedy heuristic.

While the problem definition above is for a single round t, it can be used for a series of rounds as well. The objective of the scheduler is to maximize long term utility of notification delivered, while satisfying constraints at each round. Budgets, $B(t)$ and $E(t)$ do also evolve over time in rounds. Moreover, another implicit condition needs to be ensured, which is—the queue of the scheduler should remain bounded or stable over time. In the following, we describe our scheduling algorithm for selecting notifications using the classical Lyapunov control strategy [5] that offers an elegant way to balance among utility, constraints and queue stability at the same time. In effect, the objective function of the above MCKP is modified with an adjusted utility score that we describe next.

IV. Scheduling Notification Delivery

In this section, we devise our scheduling algorithm and design a scheduler to deliver rich notifications to mobile users. Figure 1 shows various steps of the scheduler. At each round, new content items arrive in the scheduler and are added to the *incoming queue*. The scheduler then generates different possible presentations of each content item and assigns appropriate utility to them, and places them into a *scheduling queue*. The scheduler then selects a subset of notifications in their appropriate presentations out of the scheduling queue and pushes them to the *delivery queue*, from where the notifications are finally pushed to the mobile devices. The goal of the scheduler is to transfer items from the scheduling queue to the delivery queue maximizing overall utility of notifications under budget constraints while keeping the scheduling queue stable i.e. finite average length.

Inspired by [6] we use Lyapunov optimization framework [5] to select different notification presentations. The

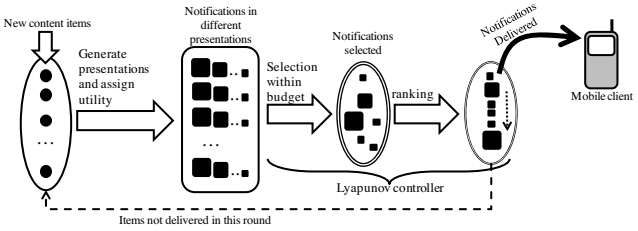


Figure 1. Scheduling workflow.

framework enables optimizing certain objectives under queue stability using *Lyapunov drift* analysis. The technique involves defining a non-negative scalar function, called a *Lyapunov function*, whose value depends on the current queue backlog. The Lyapunov optimization guarantees that a series of actions that *minimize* the Lyapunov drift over time stabilizes queues while maximizing the desired objectives—for our case, utility. More specifically, maximizing U_t under queue stability translates to:

$$\text{minimize } \Delta(L(t)) - V \times U_t \tag{3}$$

for some control parameter V where $\Delta(L(t))$ is the Lyapunov drift over a suitably defined (explained below) Lyapunov function $L(t)$. The control knob, V, balances between utility and queue size: choosing larger V favors utility over queue backlog.

For brevity, we move energy constraint (2b) to the objective. Lyapunov allows this by converting the quantity in question (here energy) to become yet another queue (virtual) in the system. We therefore define two queues: the scheduling queue, $Q(t)$ and a virtual queue, $P(t)$, that keeps track how much energy is allowed to spend in the current round. Note that $P(t)$ corresponds to energy budget $E(t)$ (introduced in Equation 2c). Ideally, $Q(t)$ should remain bounded and $P(t)$ should remain close to some constant over rounds. Let this constant be κ. We define a Lyapunov function as follows:

$$L(t) = \frac{1}{2}\left(Q^2(t) + (P(t) - \kappa)^2\right)$$

Lyapunov drift is given by: $\Delta(L(t)) = \mathbb{E}[L(t+1) - L(t)]$, where the expectation is conditioned over the current values of $Q(t)$ and $P(t)$. According to Lyapunov theory, minimizing drift $\Delta(L(t))$ ensures $Q(t)$ to remain stable and $P(t)$ to remain close to κ over time.

The drift depends on how $Q(t)$ and $P(t)$ evolve over time. Let $X_s(t)$ denote byte-size of all presentations in $X(t)$, that is $X_s(t) = \sum_{i \in X(t)} s(i)$, where $s(i) = \sum_j s(i,j)$. When a notification is delivered, all presentations of the same item are dropped from $Q(t)$. Similarly, let $X_e(t)$ be the estimated energy required to transfer items in $X(t)$. Hence, we have the following queue updates:

$$Q(t+1) \triangleq [Q(t) - X_s(t) + \nu(t)]^+ \tag{4}$$

$$P(t+1) \triangleq [P(t) - X_e(t) + e(t)]^+ \tag{5}$$

where $\nu(t)$ are the new items added to the queue during the current round and $e(t)$ is an additional budget added to the current energy budget, both of which are assumed to bounded by some constants. Consequently, for a suitably chosen constant β it can be shown that: $\Delta(L(t)) \leq \beta - \mathbb{E}[Q(t)X_e(t) + (P(t) - \kappa)X_e(t)]$. Applying the above expression in (3) and replacing minimization with maximization with the objective function negated, we obtain our optimization under data budget constraint as follows:

$$\begin{aligned} &\max\ \mathbb{E}[Q(t)X_s(t) + (P(t) - \kappa)X_e(t)] + V \times U_t \\ &\text{st.} \sum_{i \in X(i)} s(i,j) \leq B(t) \end{aligned} \tag{6}$$

We can further simplify the formulation if we assume x_{ij} to denote whether content i is selected in presentation level j. In that, we have: $X_s(t) = \sum_{i,j} x_{ij} s(i)$, $X_e(t) = \sum_{i,j} x_{ij} \rho(i,j)$, $U_t = \sum_{i,j} x_{ij} U(i,j)$. Putting all these into the objective function, we obtain the following optimization:

$$\begin{aligned} \max \quad & \sum_{i,j} x_{ij} U_a(i,j) \\ \text{st.} \quad & \sum_j x_{ij} = 1, \forall i, \text{ and } \sum_{i,j} x_{ij} s(i,j) \leq B(t) \end{aligned} \tag{7}$$

where $U_a(i,j) = Q(t)s(i) + (P(t) - \kappa)\rho(i,j) + V \times U(i,j)$ is an *adjusted* utility for presentations that takes into account current queue size ($Q(t)$), current energy level ($P(t)$) and the original presentation utility ($U(i,j)$). The first constraint forces exactly one presentation to be chosen per item. We propose a greedy heuristic to solve the above MCKP instance.

MCKP Heuristic. While 0/1 MCKP is NP-Hard, fractional MCKP obtained by considering x_{ij} to be a real within [0, 1] can be solved *optimally* by the following greedy algorithm [4]: start with presentation 0 for all items, then make a series of "upgrades" (described later) in presentations, until the budget is exhausted and the last item can only be partially upgraded. The integral solution is the same but without the last fractional upgrade. Therefore, the difference in overall utility obtained between the greedy integral solution and the greedy fractional solution is at most the utility obtained by the last fractional upgrade, which can be made arbitrarily small if presentation sizes are significantly small compared to the budget.

The algorithm defines *utility-size gradient*, $\nabla(i,j)$, as the utility to size difference ratio for two successive presentations:

$$\nabla(i,j) = \frac{U_a(i,j+1) - U_a(i,j)}{s(i,j+1) - s(i,j)}, \text{ for } 0 \leq j < k_i$$

Algorithm IV outlines the steps in our solution. It starts with selecting *all* contents with presentation 0 (set $\eta(i) = 0, \forall i$). It then finds item i' that has the *largest* utility-size gradient against its currently chosen presentation (step 5) and "upgrades" its presentations (step 8). The process continues until budget is exhausted. While the original algorithm in [4] upgrades to the *best* presentations by skipping a few in between which may have negative gradients, we rather move to the next presentation level because in our case utilities are monotone across presentations. Its runtime is $O(n + k \log_2 n)$, for $k = \max_i k_i$, due to building a max heap ($O(n)$) at the

beginning with initial gradients, followed by at most k updates, each $\mathrm{O}(\log_2 n)$.

Algorithm 1 SelectPresentations (*budget*)

$\eta(i) \leftarrow 0, \forall i$, $totalsize \leftarrow 0$, $done \leftarrow$ *false*;
while *not done* **do**
  $i' \leftarrow \arg\max_i \nabla(i, \eta(i))$
  $sizegain \leftarrow s(i', \eta(i')+1) - s(i', \eta(i'))$
  if $totalsize + sizegain \leq budget$ **then**
    $\eta(i') \leftarrow \eta(i') + 1$
    $totalsize \leftarrow totalsize + sizegain$
  else $done \leftarrow$ *true*
  end if
end while
return items for which $\eta(i) > 0$

Algorithm 2 Notification Scheduling Algorithm

1) **Notification Selection**. In round t, clear the current delivery queue. For set of items, I, in the scheduling queue, find
$$X(t) = \text{SelectPresentations}(B(t))$$
Transfer items in $X(t)$ to the delivery queue and sort them in descending order of their utility values.
2) **Data and energy budget update**. In round t, add θ to $B(t)$, and add $e(t)$ to $P(t)$ if $P(t) \leq \kappa$.
3) **Budget Deduction**. On delivery of item i: set $B(t) = B(t) - s(i,j)$ and set $P(t) = P(t) - \rho(i,j)$. Drop all presentations of i from the scheduling queue.

Scheduling Algorithm. Now, we present the scheduling algorithm for notification delivery (Algorithm 2). At each round, the scheduler chooses the best notifications from the scheduling queue and pushes them to the delivery queue. It also orders notifications in descending order of their utility values. At each round, user's data budget and energy budget are also updated. Each user specifies a budget (in bytes) per round, θ. The scheduler allocates new budget at each round ($B(t)$ is incremented by θ) and allows budget to roll over in the next round if not used in the current round. When a user downloads a notification from the delivery queue, an equal number of bytes is deducted from the data budget. Energy budget is also replenished in a similar fashion, but at a variable rate, $e(t)$, depending on the current battery status of the device.

V. Experimental Evaluation

In this section, we first explain the data-driven utility modeling based on the Spotify traces and user surveys we conducted. Then using trace-driven simulations we validate the *RichNote* and compare it with two baseline approaches.

A. Content Utility Modeling Using Spotify Trace Data

In the context of the Spotify use case, we model content utility based on real de-identified notifications collected from the Spotify production system for a duration of Jan 1, 2015 to Jan 7, 2015. Specifically, the logs contained notifications and music activity feeds sent to the users and the corresponding mouse activity by the users. We consider a notification to have a higher utility to a user if it was clicked on by that user. However, this does not imply that notifications not clicked on by a user have lower utility. This is because users may not be paying attention to all the notifications, e.g., they may not be looking at the Spotify client when the notification was delivered. To address this problem we assume that a notification has lower utility if the user hovered the mouse pointer over the notification without actually clicking it. This ensures that the user was giving some degree of attention to the notification and yet the user opted not to view that notification by clicking on it.

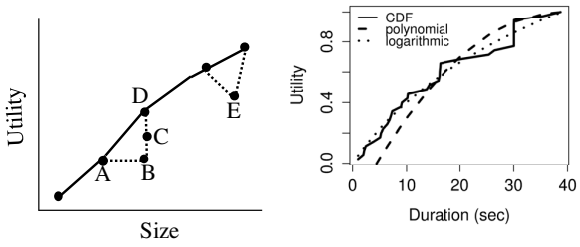
Since the goal is to model the utility of the music content in the notifications, it is critical to include features of the content that are indicators of its utility. Therefore, we combine the mouse activity data with the metadata such as attributes of the music track, album or artists data obtained using Spotify public APIs. While these attributes aid in assessing the content importance, they do not capture the social importance to the user. To address this, we obtain the social relationships of the user by combining the Spotify de-identified social graph [1] with the mouse activity. This provides us with available social ties between the recipient and the sender of the notification. In summary, we obtain the following features:

- Social ties between the sender and recipient of the notification; intuitively, a notification from a friend or favorite artist has a higher utility to the user.
- Popularity of the music track, album and artist that the notification is about. The popularity is a normalized score between 1 and 100 obtained via Spotify public APIs based on their streaming frequencies in Spotify.
- Timestamp of when the user clicked on or hovered over the notification, e.g., weekday/weekend, day/night, etc.

First we filter out notifications without corresponding mouse activity from the dataset. Using the above features we approach the utility prediction as a binary classification task with two classes: "clicked" or "hovered" for a content item i represented by the variable $x_i \in \{0, 1\}$. The goal is to learn a function that predicts if a notification i with its corresponding features mentioned above will be "clicked" ($x_i = 1$) or "hovered" ($x_i = 0$) by the user. For each notification, since we also know whether the notification was clicked on or hovered over by the user, we can exploit this information for a supervised learning technique. In this regard, we train a binary classifier using the well-known *Random Forest* (RF) classification method [7]. RF is an ensemble of many decision trees that determines the class of a notification along with a confidence score in the form of probability $Pr(x_i)$ for the predicted class x_i. Using these confidence scores, we assign content utility of an item i as below:

$$U_c(i) = \begin{cases} Pr(x_i = 1), & \text{if } x_i = 1 \\ 1 - Pr(x_i = 0), & \text{otherwise} \end{cases}$$

To evaluate the effectiveness of the learned classifier model and to ensure that we are not over-fitting to the training data we performed a five-fold cross validation. In this process, we divide the input data in to five equal parts. Then each part



(a) Useful presentations (b) Utility based of audio duration

Figure 2. Presentation utility as observed from user survey.

acts as test data while the rest of the four parts are used for training. We use the popular Weka data mining software [8] to train and test the classifier. For Spotify data with the RF classifier, we got a precision of 0.700 and accuracy of 0.689.

B. Presentation Utility Using User Survey

Presentation utility reflects the impact of the metadata and portion of the media content included into rich notifications. This part of the utility is subjective and depends on individual's taste and experience w.r.t. specific media content in question. We conduct a subjective user survey to analyze user's feedback for different possible presentations and assign utility scores accordingly. We consider music audio as our media content.

Audio content can be presented in many different ways varying across a wide range of attributes, including sampling rate (range: 8KHz–48KHz), bitrate (64kbps–320kbps), duration (couple of seconds to several minutes), channel (mono or stereo), and compression technique (wav, ogg, mp3, etc). The impact of different attributes cannot be considered separately because of potential correlations. The space of all possible attribute combinations is represented by the Cartesian product of the attributes. Fortunately, we do not need to explore the entire space for the purpose of selecting a presentation. The salient feature of this space is the trade-off between the size and utility. Thus, we do not need to consider a combination of attributes if another combination yields the same or smaller size, yet a higher utility. Consider Figure 2(a), notice that B is not a useful presentation given A, because A provides the same utility for a smaller size, and similarly D provides a higher utility than same-sized B and C.

We conducted a survey to derive audio content utility based on two attributes, namely sampling rate and duration. We used four rates, 8, 16, 32 and 44KHz, and five durations, 5, 10, 20, 30 and 40 seconds. We produced a set of corresponding audio samples and asked a number of users to rate them in the scale of 0–5, indicating to what degree they are satisfied listening to these samples. We observed that utility scores for these 20 presentations varied from 0.3 to 3.3 and resulted in only six useful presentations, which constituted a monotone rise in utility scores across their respective sizes (similar to the illustration in Figure 2(a)).

Utility based on audio duration. Arguably, utility of an audio content presentation mostly depends on its duration. We studied how user choice varies in choosing a suitable duration for a good audio notification. We conducted another survey among 80 users where we provided a few music tracks (average duration 276 seconds) and asked the users to listen and *stop* at the point when they think the duration was barely enough for a good notification. We define utility, $util(d)$, of an audio sample of duration, d, to be proportional to the fraction of users who prefer a notification to be equal or smaller than that duration. That means, CDF of duration is translated into utility value. We model $util(d)$ as one of the two possible functions, namely logarithmic $a + b\log(1+d)$ and polynomial $a\left(1-\frac{d}{D}\right)^b$, for some constants a, b and D. Using linear regression, we deduce these constants from our survey results obtaining the following two utility functions:

$$util(d) = -0.397 + 0.352 \times \log(1+d) \quad (8)$$

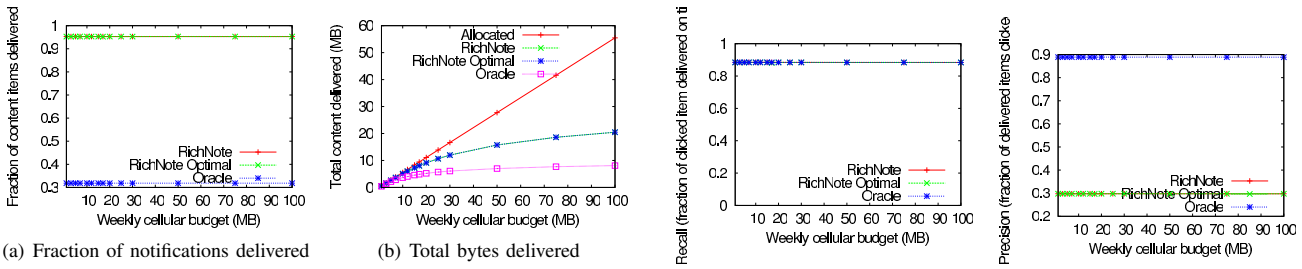
$$util(d) = 0.253 \times \left(1 - \frac{d}{40}\right)^{2.087} \quad (9)$$

From our survey results, logarithmic function showed a better fit (Figure 2(b)) so we use this function in our experiments. These surveys, though limited in scale, give useful insights about deriving presentation utility of media content. A wide scale survey through crowdsourcing can give better results.

C. Experimental Setup

We use a custom event-based simulator [6] written in Java. Using the de-identified Spotify traces, we conduct the simulations for top 10k users with maximum number of delivered notifications in the traces. This allows us to focus on users for whom the resource budget constrains are important. Note that while we run simulations using 10K users, our solution can potentially scale to a much larger user base using a backend parallel platform since our solution can work in rounds and independently for each user. For each user, we replay all notifications intended for them as a stream of content items arriving at our scheduling and delivery system (i.e., a broker), see Section IV. The broker then prepares notifications for these items in specific presentations and selects them via the `SelectPresentations` method as shown in Algorithm 2. The selected notifications are scheduled for delivery by moving them to a queue from where they are delivered to the end users. We assume that users are on their mobile devices and are connected to the broker sporadically through a cellular connection with a budgeted data plan. A separate trace (obtained from [6]) of timestamped battery status per user is also provided to mimic energy drain and battery recharge patterns of the devices.

To compare *RichNote* performance, we use two baselines: (1) **FIFO** that delivers notifications in the order of their delivery timestamps in the trace, and (2) **UTIL** that delivers notifications in decreasing order of utility score. The experiments are designed to demonstrate the adaptiveness of *RichNote* by comparing metrics such as the precision, recall and utility of the delivered notifications with the baselines under varying data budget. We chose these two baselines because they are the widely used techniques in the industry. For example, in Spotify FIFO is used in real-time mode and UTIL strategy is



(a) Fraction of notifications delivered (b) Total bytes delivered (c) Recall (d) Precision

Figure 3. Performance metrics for different scheduling technique.

used in batch mode. Since for mobile devices energy efficiency is critical we simulate the battery consumption and measure the energy efficiency. Finally we evaluate the effectiveness of the scheduling algorithm by measuring the queuing delays.

We assume that each user specifies a certain data budget within which she prefers to receive these notifications. This rate is denoted as *budget per week* indicating the upper limit on data consumption by the notifications. We vary this budget from 1MB to 200MB. Each user also specifies a certain amount of energy to spend per round (κ in our scheduling algorithm). We assume each round to be 1 hour (3600 seconds) and set κ to 3KJ per hour. We set Lyapunov control parameter, V, to 1000. These settings are used from [6].

We generate different presentation levels by varying the duration of music tracks at a fixed bitrate of 160 kbps (Spotify default bitrate). Specifically we consider six levels: first level contains metadata only and five more levels with 5, 10, 20, 30 and 40 seconds of previews in addition to metadata. For *RichNote* we do not need to set a presentation level as it switches between them according to the available budget, but for both baseline approaches (FIFO and UTIL) we need to fix the presentation level to mimic state-of-the-art techniques. We assume average metadata size is 200 bytes (as mentioned in [2]), consisting of information about a music track, artist, album names, and a URL link to the track. While assigning presentation utility, we assume a small portion of utility (about 1%) is due to metadata and the rest is from the audio content, the latter part is specified by the logarithmic utility function in Equation 8. At 160kbps bitrate, the size of a d-sec preview is $d\times$20KB assuming no audio compression is used.

We use the following performance metrics to compare *RichNote* with the chosen baselines:

Delivery ratio: The fraction of notifications delivered.

Precision/recall ratio: We measure *precision* as the fraction of delivered notifications (before the recorded click time in the Spotify trace) that are clicked on by the users, and *recall* as the fraction of total clicked notifications that are delivered to the users.

Average utility: Average utility of delivered notifications to users computed using Equation 1.

Download energy: Energy spent in downloading notifications based on the energy model from [9].

Queuing delay. The time between when a notification arrives in the broker and when it is delivered.

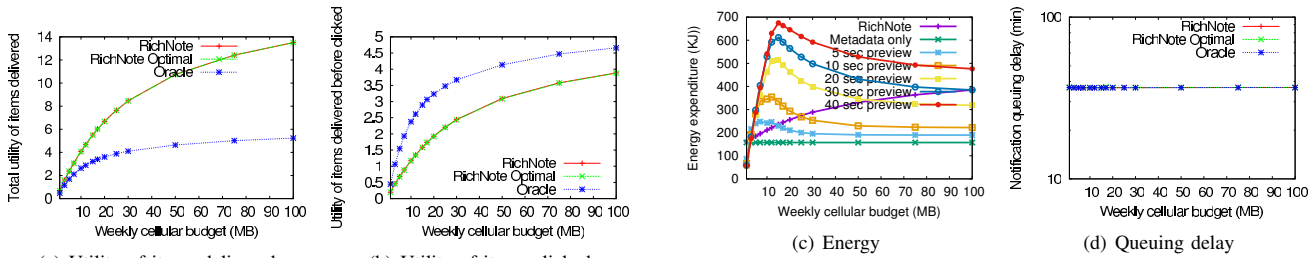
In all experiments for all methods unless mentioned explicitly the values of these metrics are averaged across all users.

D. Experimental Results and Analysis

1) Performance Comparison of Different Methods: In the first set of experiments the goal is to compare performance metrics of different methods by varying available weekly data budget from 1 MB to 100 MB. In these experiments we fix the presentation level of FIFO and UTIL to metadata with 5s and 10s previews. This matches the current behavior of Spotify embedding an URL to 10s song preview in some notifications. In Fig. 3(a), it can be observed that delivery ratio increases as the data budget increases. An interesting observation here is that *RichNote* always delivers close to 100% notifications while FIFO and UTIL need a higher data budget to deliver more items. This is because *RichNote* adapts the presentation level to lower presentation levels when the data budget is limited. On the other hand since FIFO and UTIL have fixed budget they can deliver only limited fraction of notifications with the given limited budget. Fig. 3(b) shows the total amount of data delivered by different methods. *RichNote* delivers more bytes than its counterparts again because of its presentation adaptation. Furthermore, *RichNote* does not use up the entire budget allocated because data delivery is limited by other factors, such as network and energy availability.

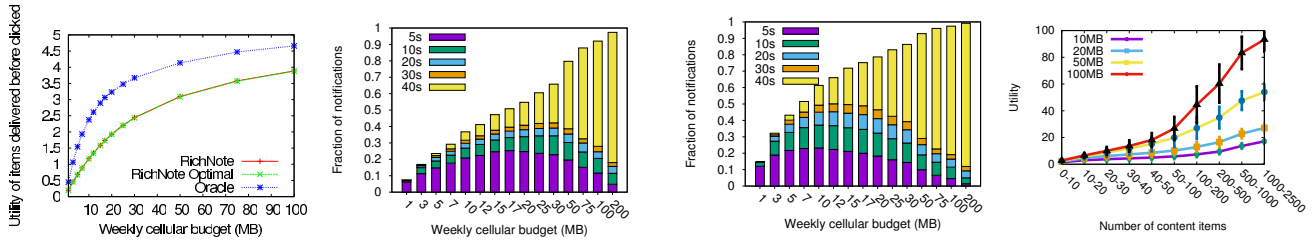
Figures 3(c) and 3(d) show recall and precision results for the three methods. We observe that *RichNote* has higher recall and precision than other techniques. This is directly related to the fact that *RichNote* delivers close to 100% notifications within the given budget. On the other hand precision is limited to 0.3 because of high recall. Note that it is possible to achieve higher precision using *RichNote* by only delivering notifications with higher utility value. However, *RichNote* makes use of all the available data budget to deliver more notifications even when they are not being clicked on by the users.

In Fig. 4(a), we show utility of all delivered notifications aggregated across all the users for different methods. We see that *RichNote* achieves higher utility than FIFO and UTIL since it is designed to maximize utility. Since *RichNote* does not strictly follow FIFO, notifications may be delivered after the actual click time in the input data. To mitigate this effect *RichNote* employs the round-based model. In Fig. 4(b) we



(a) Utility of items delivered (b) Utility of items clicked (c) Energy (d) Queuing delay

Figure 4. Utility of delivered items and energy consumption and delivery latency in different scheduling techniques.



(a) Utility of clicked items (b) Presentations (c) Cellular and Wifi (d) Utility across users

Figure 5. Adaptation of *RichNote* and Utility Across Users

can see that *RichNote* still has a higher utility compared to the FIFO and UTIL methods even among the clicked items.

Fig. 4(c) shows energy expenditure comparison between UTIL and *RichNote* (we omit FIFO because it shows similar results). We can notice that the energy expenditure of *RichNote* is steady and stable compared to UTIL. Unlike UTIL that has spikes at a lower budget, *RichNote* strives to control energy consumption and keep it below the specified threshold of 500KJ (given by κ, 3KJ per hour for 7 days). However, at the data budget 100MB *RichNote* energy consumption matches UTIL, this is due to the higher volume of data with richer presentation delivered by *RichNote*. since In addition, *RichNote* adapts presentations based on network dynamics, which results in significantly lower queuing delay (Fig. 4(d)).

2) Presentation Adaptation of RichNote: Next, we analyze the adaptive behavior of the *RichNote* method. In Fig. 5(a) we can see that *RichNote* outperforms all fixed presentation methods. Apparently, no single fixed presentation method performs well with respect to the utility in all scenarios, which illustrates their limitation. For example, it can be observed that when the data budget is lower than 20MB, presentation levels with previews shorter than 20s perform better, but when the budget is between 20MB and 50MB, the presentation level with a 20s preview performs the best and finally when the budget is greater than 50MB, presentation levels with previews longer than 30s achieve higher utility values. This can also be supported by observing presentation levels selected by *RichNote* as shown in Fig. 5(b) with a stacked bar chart. For brevity, we do not show "metadata only" presentation in the figure, which is simply the missing fraction in each stack. We observe that for a data budget lower than 3MB only 10% notifications contain media preview and the rest 90% are delivered as "metadata only". As the budget grows, more items are delivered at higher presentations (e.g. at 20MB, nearly 20% items are delivered with a 40s preview). These observations illustrate that *RichNote* adapts well to the data budget.

3) Network Conditions: In this experiment we show that *RichNote* adapts to network conditions such as when wifi is available in addition to cellular network. We simulate network condition by using a Markov transition model (as given in [6]) among three states, namely WIFI, CELL and OFF, which indicates whether the user is on wifi, cellular or none. We use 50% probability to remain in the current network condition and equal probability of transiting to cell or wifi when off. As shown in Fig. 5(c) when devices use wifi, they receive richer presentations than cellular only option (shown in Fig. 5(b)) because wifi allows more data to deliver.

4) Performance Across Users: In Fig. 5(d) we measure the utlity achieved for users with different notification delivery rates. We measure this by introducing user categories based on the number of content items. Then we assign users with a given number of content items in to the corresponding categories. The utility is shown as an average across all users in the same category along with the error bars indicating the deviations from the average. We can observe that users with higher number of items benefit more.

5) Lyapunov effects: The value of the Lyapunov control framework manifests itself in ensuring continued and stable performance despite changes in connectivity and energy budget. We observe this adaptation capability through a) more delivered data with more leftover bandwidth (Fig. 3(b)), b) lower queuing delays (Fig. 4(d)), c) ability to exploit new additional networks, e.g., Wifi (Fig. 5(c)). We conducted experiments measuring the sensitivity of *RichNote* to Lyapunov control knob, V, and observe that *RichNote* performs uniformly better in all these settings.

VI. Related Work

Large-scale notifications systems are popular in many domains, e.g. in public safety (CityWatch, CodeRED, Everbridge), societal and personal information sharing (Twitter, FourSquare, Facebook), traffic and weather alerts (www.wunderround.com, weather.gov).

Efforts in the multimedia community have explored the use of complex transcoders [10] and multilayered encoding for scalable storage and transmission of rich voluminous content. Encoding formats such as H.264/SVC [11] encode media (e.g. video) into a stream of multiple layers allowing in-network methods to adapt layers based on wireless dynamics. Recent work also combines multilayer coding techniques with scheduling on hybrid networks (e.g. cellular-adhoc) to improve dissemination [12]. Content adaptation with varying degree of fidelity amid of resource scarcity is used in Odyssey [13]. Content prefetching techniques have focused on addressing connectivity and device energy constraints [14], [15].

A key goal of pub/sub systems has been the instant and efficient notification of publications to subscribers that may be content-based or topic-based [16]. Key focuses include scalability and efficiency through efficient subscription management, event-matching [17]–[20], and optimal overlay design [21]. Recent work addresses in-network customization of individual publications based on user/device profiles (e.g. spoken dialects, device footprint [22]) and overlay adaptation with fast-changing subscriptions in mobile settings. *RichNote*, in contrast, alters the semantic content of the publication itself to create a succinct, yet informative experience for the user.

Traditional content recommendations are driven by analysis of the content and user profile [23]–[25]. A direct application of these techniques to *RichNote* may not be feasible: methods that rebuild entity graphs to re-evaluate relationships would be computationally prohibitive in our dynamic setting where arrival of new content is the norm. Such schemes might also be unsuitable for mobile deployment. While *RichNote* inherits the spirit behind the systems, it also addresses the additional complexities associated with multimodal content, mobile user and dynamic network constraints and the consequent scaling of recommendations. Recent efforts use crowdsourcing and multimedia information for a richer experience [26], [27].

VII. Concluding Remarks

Customizing notifications while managing diverse updates and hard-to-predict content arrival rates requires a careful analysis of user-content relevance and mobile device dynamics. In this paper, we explored a structured approach to quantify the utility of rich content and developed techniques to formulate an appropriate rich notification delivery system. To the best of our knowledge, *RichNote* is the first framework to explore adaptation of rich media notifications in a unified manner at multiple levels involving the user, content and infrastructure.

Acknowledgment

This work was supported by NFS award nos. CNS1528995, CNS1450768, CNS1059436, CNS1063596 and 1447720. The first author acknowledges HEQEP CP 3137 of Bangladesh for funding his research visit to UCI.

References

[1] V. Setty, G. Kreitz, R. Vitenberg, M. van Steen, G. Urdaneta, and S. Gimåker, "The hidden pub/sub of spotify:(industry article)," in *ACM DEBS*. ACM, 2013, pp. 231–240.

[2] V. Setty, R. Vitenberg, G. Kreitz, G. Urdaneta, and M. van Steen, "Cost-effective resource allocation for deploying pub/sub on cloud," in *IEEE ICDCS*. IEEE, 2014, pp. 555–566.

[3] V. Setty, G. Kreitz, G. Urdaneta, R. Vitenberg, and M. van Steen, "Maximizing the number of satisfied subscribers in pub/sub systems under capacity constraints," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 2580–2588.

[4] P. Sinha and A. A. Zoltners, "The multiple-choice knapsack problem," *Operations Research*, vol. 27, no. 3, pp. 503–515, 1979.

[5] L. Georgiadis, M. Neely, and L. Tassiulas, *Resource allocation and cross-layer control in wireless networks*. Now Publishers Inc, 2006.

[6] N. Do, Y. Zhao, S.-T. Wang, C.-H. Hsu, and N. Venkatasubramanian, "Optimizing offline access to social network content on mobile devices," in *IEEE INFOCOM*, 2014.

[7] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[8] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[9] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *ACM SIGCOMM IMC*, 2009.

[10] J. Xin, C. Lin, and M. Sun, "Digital video transcoding," *IEEE*, vol. 93, no. 1, pp. 84–97, 2005.

[11] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the h. 264/avc standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 17, no. 9, pp. 1103–1120, 2007.

[12] N. Do, C. Hsu, S. Jatinder, and N. Venkatasubramanian, "Massive live video distribution over hybrid cellular and ad hoc networks," in *IEEE WoWMoM*, Lucia, Italy, June 2011, pp. 1–9.

[13] B. Noblea and M. Satyanarayanan, "Experience with adaptive mobile applications in odyssey," *Mobile Networks and Applications*, vol. 4, pp. 245–254, 1999.

[14] B. Higgins, J. Flinn, T. Giuli, B. Noble, C. Peplin, and D. Watson, "Informed mobile prefetch," in *ACM Mobisys'12*, 2012.

[15] P. Mohan, S. Nath, and O. Riva, "Prefetching mobile ads: Can advertising systems afford it?" in *ACM Eurosys'13*, 2013.

[16] R. G. P. Eugster, P. Felber and A. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, 2003.

[17] K. Jayaram, C. Jayalath, and P. Eugster, "Parametric subscriptions for content-based publish/subscribe networks," in *Middleware*, 2010.

[18] F. Fabret, H. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha, "Filtering algorithms and implementation for very fast publish/subscribe systems," in *ACM SIGMOD*, 2001.

[19] H. Jafarpour, S. Mehrotra, N. Venkatasubramanian, and M. Montanari, "Mics: An efficient content space representation model for publish/subscribe systems," in *ACM DEBS*, 2009.

[20] A.-M. Kermarrec and P. Triantafillou, "Xl peer-to-peer pub/sub systems," *ACM Comput. Surv.*, vol. 46, no. 2, Nov. 2013.

[21] R. Baldoni, R. Beraldi, V. Quema, L. Querzoni, and S. Tucci-Piergiovanni, "Tera: topic-based event routing for peer-to-peer architectures," in *ACM DEBS*, 2007.

[22] H. Jafarpour and B. H. et al, "Ccd: Efficient customized content dissemination in pub/sub," in *Middleware*, 2009.

[23] D. Liu, G. Ye, C. Chen, S. Yan, and S. Chang, "Hybrid social media network," in *ACM Multimedia*, 2012.

[24] J. Noel and et al., "New objective functions for social collaborative filtering," in *ACM WWW'12*, 2012.

[25] H. Ma and et al., "Recommender systems with social regularization," in *ACM WSDM*, 2011.

[26] D. Deng, C. Shahabi, and U. Demiryurek, "Maximizing the number of worker's self-selected tasks in spatial crowdsourcing," in *SIGSPATIAL'13*, 2013.

[27] L. Kazemi, C. Shahabi, and L. Chen, "GeoTruCrowd: Trustworthy query answering with spatial crowdsourcing," in *ACM SIGSPATIAL*, 2013.