Near Real-time Optimization of Activity-based Notifications

Yan Gao, Viral Gupta, Jinyun Yan, Changji Shi, Zhongen Tao, PJ Xiao, Curtis Wang, Shipeng Yu, Romer Rosales, Ajith Muralidharan, Shaunak Chatterjee
LinkedIn Corporation
Mountain View, CA, USA
{yagao,vigupta,jiyan,cshi,ztao,pjxiao,cuwang,siyu,rrosales,amuralidharan,shchatterjee}@linkedin.com

ABSTRACT

In recent years, social media applications (e.g., Facebook, LinkedIn) have created mobile applications (apps) to give their members instant and real-time access from anywhere. To keep members informed and drive timely engagement, these mobile apps send event notifications. However, sending notifications for every possible event would result in too many notifications which would in turn annoy members and create a poor member experience.

In this paper, we present our strategy of optimizing notifications to balance various utilities (e.g., engagement, send volume) by formulating the problem using constrained optimization. To guarantee freshness of notifications, we implement the solution in a stream computing system in which we make multi-channel send decisions in near real-time. Through online A/B test results, we show the effectiveness of our proposed approach on tens of millions of members.

KEYWORDS

Notifications, Stream computing, Optimization, Machine learning

ACM Reference Format:
Yan Gao, Viral Gupta, Jinyun Yan, Changji Shi, Zhongen Tao, PJ Xiao, Curtis Wang, Shipeng Yu, Romer Rosales, Ajith Muralidharan, Shaunak Chatterjee. 2018. Near Real-time Optimization of Activity-based Notifications. In *KDD 2018: 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3219880

1 INTRODUCTION

Social media applications (e.g., Facebook, LinkedIn, Instagram, Twitter, Whatsapp) provide enormous value to members, as evident from their ever-growing usage and engagement numbers. One salient fact unifying the value realization for all of these applications, is that they require the member to visit the application. Until a few years back, these visits were predominantly to the desktop webpage. Now, mobile traffic (primarily native app traffic) is accounting for a majority of usage, a result of ever-increasing mobile penetration and growing amount of time spent on mobile phones.

In the last few years, another trend that has become an important factor in mobile app usage is the use of notifications. Before the advent of notifications, these applications relied on a member's organic intent and information need for site visits and value realization. Emails formed an alternate mechanism to induce members to visit, but the feedback loop (between email sent and an ensuing member visit) is typically hours or days.

Mobile app notifications shortened that feedback loop to minutes because of their inherently intrusive nature. Also, notifications have higher engagement rates compared to emails, primarily due to the selection bias of app installers who are eligible to receive notifications. Hence they are much more effective in bringing users back than email. The greater effectiveness of notifications is accompanied by members' expectation of a much higher relevance bar compared to email, for what is notification worthy.

This member expectation is met in two different ways. Some event categories are deemed **always** notification worthy based on product goals, member expectation, extensive user experience research and experimentation. Examples of such "unfiltered" notifications include member-to-member(s) messaging (on Facebook Messenger, Whatsapp, LinkedIn messaging) and member-to-member invitations (on Facebook and LinkedIn). For other event categories, the signal-to-noise ratio can be lower, and hence notifications triggered by such events are deemed "filter-eligible". Sometimes, such notifications are also decorated with member interpretable reasons (e.g., "Jill shared for the first time") to help members understand why they were notified about this event.

The main consideration while sending such "filter-eligible" notifications is the utility we provide to members by notifying them about a particular event. While a high utility notification has the potential to delight a member, a low utility one can annoy a member. Some considerations that go into the decision of what to notify a member about include, but are not limited to:

- Meet a member's information need by helping them discover interesting content from people of interest and/or topics of interest
- Enabling members to discover the triggering event and act on it if applicable in a timely fashion
- Enabling members to discover the triggering event, and other related or unrelated events which they have not done previously, since they have not visited in a while
- Not overwhelm or annoy the member with too many notifications

In this paper, we focus on the problem of determining which notifications to send in the "filter-eligible" category. More specifically, the candidate set of events that we consider are members posting, sharing or reacting (e.g., liking or commenting on) to other events, which typically show up in a recipient member's social feed (also referred to as the news feed). The member who posts, shares or reacts will henceforth be referred to as an actor, and the member who receives a notification about the actor's action (i.e., the event) – or views it in her news feed – as the recipient or the viewer.

The news feed recommendation is a well-studied ranking problem [3, 4, 6, 18]. Upon a member (i.e., viewer) visiting the site, all updates from the viewer's network (i.e., actors) are scored to reflect the member's propensity to interact with them, and then displayed based on a sorted order of the scores. Further inter-update considerations like topic-diversity, actor-diversity and advertisement-density are often made.

The corresponding notification problem is related yet different. When an actor acts (by posting, sharing or reacting), we need to evaluate which viewers in the actor's network would like to be notified about this event.

Both problems (feed ranking and notification filtering) need an underlying model to determine the relevance of an update for a specific viewer, and they share the same high-level objective of creating an engaging experience for the viewer. However, there are a few key differences.

- To rank a feed, we score a set of eligible updates for a given viewer, whereas for notifications, we score a set of eligible viewers for a given update
- Feed ranking is done when a viewer visits the site, whereas candidate notifications (for an actor's network) are scored when an actor takes an action
- Additional considerations differ. For instance, diversity and freshness are very important for feed, whereas timeliness and high precision are critical for notifications

In this paper, we describe our experience of building an in-production system which solves the aforementioned problem. Our solution includes defining utilities to capture the key factors at play, proposing an optimization formulation to balance the various utilities, detailing various nuances in the solution space, and some special considerations for near real-time near-optimal decision making. We also describe a near real-time system which has been implemented to ensure timely delivery of high quality notifications. Finally, we share several experiments on our app use base to demonstrate validation of both our overall approach and of several proposed improvements.

The paper is organized as follows: Section 2 introduces the notifications portfolio at LinkedIn, to provide detailed and real-life context of the problem. We formulate the problem by specifying the various utilities at play and present a framework to solve it in Section 3; next, we describe our near-line infrastructure which is used to score and serve the relevant notifications in near real-time in Section 4; Finally, we describe online A/B test results in Section 5 which show the benefit these notifications bring to members, and summarize our findings along with some future work in Section 7.

2 NOTIFICATIONS AT LINKEDIN

LinkedIn is the world's largest professional network with more than 546 million members in over 200 countries [12]. In order to achieve LinkedIn's mission, which is to "connect the world's professionals to make them more productive and successful", we facilitate members connecting to one another and interacting in various forms.

Members obtain value from LinkedIn by growing their network and discovering relevant activities within their network through various LinkedIn products. Some examples include the news feed, the people you may know (PYMK) product, private messaging and job search. Members have two primary mechanisms to visit the site – either organically, or via an email or mobile app notification. For the rest of the paper, we will focus on members with the mobile app, and hence on organic visits to the app and notification-triggered visits.

2.1 The portfolio

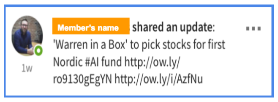
One form of interaction is between two or a small group of members – e.g., forming a connection (by sending or accepting an invitation), private messaging. In most social networks, including LinkedIn, these are almost always deemed relevant by members, and hence form the "unfiltered" category of notifications.

The second category of notifications is about activities in a recipient's network. This category is the focus of this paper (as alluded to in Section 1). There are 4 primary types of notifications in this category:

- "Shared-by-your-network" or SBYN. When an actor shares an update, and members connected to the actor receive a notification about the share. An example is shown in Figure 1a.
- "Posted-by-your-network" or PBYN (shown in Figure 1b). Similar to a share, a post differs slightly in that it is a long-form article self-authored by the actor. A share refers to short-form posts (e.g., status updates) or URL re-shares.
- "Talking-About" or TA (Figure 1c). When an actor likes or comments on an update. There is also an aggregate version of this notification, when multiple actors within the recipient's network like or comment on an update.
- "Mentioned-in-the-news" or MITN. Since LinkedIn is a professional network, members are often interested when their connections appear in some news articles. LinkedIn has an automated system to extract named entities from all ingested articles, and map these entities to members on the site. When such a member (the actor in this use case) is identified, her network can be potentially informed about that news article. An example is shown in Figure 1d.

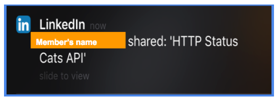
These 4 notifications – hereafter referred to as **activity based notifications** – together make up the portfolio that we will focus on in this paper. Since the total volume of candidate notifications (in this portfolio) for almost every user is quite high, this category of notifications are "filter-eligible". In order to create a good member experience, we need to aim for high precision and significantly reduce the noise.

One salient feature of this notification portfolio is the value of timeliness. There is tremendous value in being able to deliver a notification about a highly relevant activity as soon as the activity happens. This provides the recipient an opportunity to join the conversation around the activity soon after it transpires. This timeliness need is met by building a custom near real-time system, which will be described in detail in Section 4.



Member's name shared an update: 'Warren in a Box' to pick stocks for first Nordic #AI fund http://ow.ly/ro9130gEgYN http://ow.ly/i/AzfNu

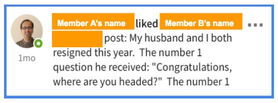
(a) SBYN



LinkedIn now

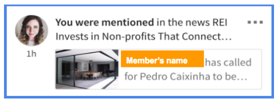
Member's name shared: 'HTTP Status Cats API'

(b) PBYN



Member A's name liked Member B's name [redacted] post: My husband and I both resigned this year. The number 1 question he received: "Congratulations, where are you headed?" The number 1

(c) Talking About

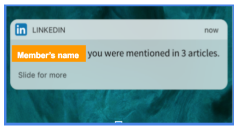


You were mentioned in the news REI Invests in Non-profits That Connect…

Member's name has called for Pedro Caixinha to be…

(d) MITN

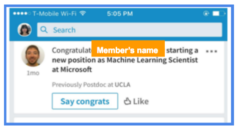
Figure 1: Examples of activity based notifications



LINKEDIN now

Member's name you were mentioned in 3 articles.

(a) push notification

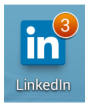


T-Mobile Wi-Fi 5:05 PM

Congratulat Member's name starting a new position as Machine Learning Scientist at Microsoft

Previously Postdoc at UCLA

(b) in-app notification



(c) badging

Figure 2: Examples of notification channels

Delivering such notifications in near real-time also introduces special optimization challenges. If we knew *a priori* all the candidate notifications, then our optimization formulation would be much simpler, as we could do batch processing. With the requirement of timely delivery, we have to anticipate the quality of future activities and take special care to ensure that we deliver only the best activity-based notifications that will come up over a certain time period. These challenges will be laid out in more detail in Section 3.

There is a third category of "filter eligible" notifications which are generated by LinkedIn to meet a member's information needs. These include , job recommendations, birthdays, work anniversaries and job changes of connections among others. We will not address this set explicitly in this paper since most of them do not have a real-time nature to them and hence can be tackled by batch optimization methods.

2.2 Notification channels

Android and iOS are the two most popular mobile platforms. Both these platforms provide two different notification channels:

- in-app notification
- push notification

An in-app notification is a type of message that is delivered within the app. The external change when such a message is delivered is visible on the app icon in the form of either an orange dot on the app icon, or an increase in a counter value inside the orange dot. An example of how the app icon is badged is shown in Figure 2c. The content of the notification itself is only visible inside the app (as shown in the screenshot in Figure 2b). In-app notifications do not generally use ringtones or buzz and hence are considered less intrusive. An in-app notification is always accompanied by the corresponding badging change on the app icon. The orange dot (along with its counter if applicable) disappears as it is reset when a member visits the app and the relevant tabs inside the app which resulted in the external badge.

A push notification – an example for iOS is shown in Figure 2a – is a type of message *sent to a mobile device* to deliver pertinent information to members. These appear as alert-like messages on the home screen (on iOS) or in a notification area at the top of the screen (on Android) and normally come along with ringtone or vibration, which is deemed as potentially intrusive, especially if the notification content is not relevant and timely to the recipient.

For activity-based notifications, LinkedIn follows the rule that a push notification will always be accompanied by the corresponding in-app notification (and badging), but not vice-versa. The reason for this is two-fold:

- Push notifications are more intrusive, and hence should be used more selectively than in-app notifications.
- In-app notifications persist in the notification tab inside the app, and hence are more easily revisited. Push notifications, once dismissed or clicked, disappear from the mobile notification home. LinkedIn prefers to give members the option of revisiting any notification sent to them at a later point if they so wish.

As a result of the two channels, and their relationship as mentioned above, the decision making process for every candidate notification can be broken down into 2 sequential steps – 1. Whether to send it as an in-app notification, and 2. Whether to send it as a push notification, only if it is being sent as an in-app notification as well. This decision making process is called channel selection and is further detailed in Section 3.3.

2.3 Notification disablement

Only members who have a mobile app installed (and have notifications from the app turned on) are eligible to receive notifications. There are separate controls for (un)subscribing (from) to each of the two notification types.

The LinkedIn app also provides fine-grained settings to members to select which notification types they would like to receive (or stop receiving). This is also the industry standard and is adopted by most major social media apps.

When members are unhappy with the set of notifications they are receiving, they often disable notifications. Most members choose to disable all notifications from the app, instead of going to settings and turning off specific notification types. The cost of such a disablement is very high - firstly because we have caused a very poor member experience which may reduce her propensity to use the app in the future, and secondly because we now lack the ability to surface timely and relevant information nuggets to the member (even if we improved our ability to determine what is relevant, which evidently needed more work for this member).

Unlike engagement models, which are easily attributed to a specific update, disablement decisions are often the result of several sub-par sends over many days or weeks. Thus, it is almost impossible to attribute a disablement to a specific notification. In this paper, we will not address this complexity but instead add constraints on notification volumes that can be sent to a member to ensure a good member experience (and also keep disables in check).

3 PROBLEM FORMULATION AND SOLUTION

The goal of activity based notifications at LinkedIn is to inform the user about engaging and relevant activity in their network in a timely fashion. As members share/like/comment on content in the feed, notifications can be used to inform other LinkedIn members who would find the update relevant. The process of identifying the right notification to send can be broken down to three parts.

First, we identify the candidate recipients for an update. This step takes into account the perceived relevance to the recipient, while also considering the coverage of notification candidates for recipients who may have different activity levels.

Next, we identify the relevance of each update using response prediction models that predict users engagement on candidate notifications.

To determine the notifications to send to our members, these models are then used as a part of an optimization problem to allow us to select the best notifications for our members, either using a global or a personalized approach. While we solve a constrained model for a sample time period, we will see that the solution derived will be implemented in a streaming basis to allow us to independently make a decision on each candidate notification. As we explain below, these different stages have multiple factors we take into account to allow us to satisfy different objectives of our notifications platform.

3.1 Recipient candidate selection and fanout

This first step in determining which notifications to send is the recipient candidate selection for each update. To keep the notifications socially relevant, the candidate set is restricted to members following or connected to the actor. We call this candidate generation process the "fanout". However, this still includes recipients who may not find the actor's update relevant. To further restrict the candidate set, while maintaining relevance, we consider the following approaches to choose certain recipients

- Feed interaction edge affinity (or "Edge affinity" for short) fanout- In this method, we restrict recipients to members who have had interactions with the actor's updates in the feed in recent history. This is captured by the presence of a non-zero interaction based edge affinity feature, which is computed based on the historical interaction of the recipient on the actors updates in the feed.
- Connection strength fanout- Here, we restrict recipients to members who either belong to the previous set or members ("Edge affinity" based fanout) who are predicted to have a good connection strength, ie. the connection strength is above a predefined threshold we use across applications at LinkedIn. Connection strength a probability score of the likelihood of members interaction on LinkedIn, including member to member interactions outside the feed (eg. messaging). This allows us to expand the candidate set beyond the previous set, especially for our less active feed members. Sending notifications to this user base can help communicate the value of the content shared on the LinkedIn platform.

Each of the methods above have some advantages over the other. The candidate set size for connection strength based fanout is 10x the size of the set generated using edge affinity based fanout, so the edge affinity based fanout is more friendly to scale. It is also important to note that different candidate selection methods can potentially benefit users in different segments of their lifecycle. The edge affinity based selection benefits recipients who frequently use the LinkedIn feed, and are generally engaged with the actors updates, while the connection strength based methods can help notify our newly onboarded or less engaged users about relevant updates from their connections/follows.

3.2 Response Prediction Models

Once the recipient candidate set for an update is identified, we want to evaluate the relevance of the update for each candidate recipient and only send the notifications which are highly engaging. We consider two alternative models to capture this

- pClickPush - A model that predicts the probability of an user clicking on the push update in the phone notifications tab (as we saw in Figure 2a).
- pClickInapp - A model that is used to predict the probability of user interacting with the notification within LinkedIn notifications tab as we saw in Figure 2b.

We use logistic regression models for response prediction, and use ℓ_2 regularization when training our models. For either model, let y_{ui} denote user u's response to a notification i described by feature vector $\mathbf{f}_{ui}$, then the corresponding Bernoulli random variable $\mathbb{Y}$ is modeled by

$$\text{Prob}\{\mathbb{Y} = 1|F\} = \frac{1}{1 + \exp(-\theta^T \mathbf{f}_{ui})} \tag{1}$$

where θ is the model coefficient vector we need to learn.

The prediction (or scoring) process is as follows: once θ has been learned from training process, for any new notification $\mathbf{f'}_{ui}$, we use the mean of the Bernoulli distribution as the predicted response:

$$\text{Prediction} \quad = \quad \frac{1}{1 + \exp(-\theta^T \mathbf{f'}_{ui})}$$

Data Collection for modeling. To enable data collection for our initial models, we launched a heuristic model based on affinity features. Notifications were sent when the affinity feature exceed a predetermined threshold, and the engagement on these notifications provided the initial training datasets. Once we ramped our initial experiments which used the models described in this section, this data collection method was retired. Currently, we use a random sample of data collected from every activity based notification sent out by all active models during that time period.

We generate a snapshot of features used during scoring time for the notifications sent through our system. This is joined with our tracking data to generate the labels. Click on a push notification generates a positive label for a pClickPush model, while clicks on the notifications tab generates a positive label for the pClickInapp models. This data is used to generate training, test and validation datasets.

In some cases, we want to explore the use of new features, which may not be snapshotted during scoring. In this scenario, we typically join the new feature with our datasets before our training process.

model	AUC	overall O/E ratio
pClickPush	0.723	1.001
pClickInApp	0.73	1.0056

Table 1: Offline evaluation of response prediction models

Features for the models. We use four classes of features in feature vector $\mathbf{f}_{ui}$:

- Actor features: the features for the user who performed the activity at LinkedIn network such as share/like/comment. This feature class includes user's profile features (industry, education, position, company, skills and etc), the user's activity level (active user or not), the user's historical actions such as # clicks (or # dismiss) on different kinds of notifications within different time intervals.
- Item features: the features for the activity and the corresponding entity (article, image, video ...). This includes the creation time of the entity, the timestamp of the entity, the popularity of the entity (#likes/comments/reposts), topic, and context embedding features.
- Recipient features: the features for the targeted recipient selected by Fanout. They are basically the same set of features as the actor features but for the user who will receive the notification.
- Edge features: features for the actor-recipient pair. For example, the affinity score between actor and recipient reflects the historical interactions between the two users in feed. Another example is seniority difference between actor and recipient. The intuition is that recipient is more interested in the entity from an user who are more senior in a professional capacity.

With the feedback and feature data, we train the logistic regression model on Hadoop using in-house optimization libraries [7]. The regression models are trained multiple times against different regularization parameters and we select the best model depending on our off-line metric (i.e., AUC).

Offline Evaluation of models. We use two metrics to evaluate the performance of our response predication models:

1. Area under the receiver operating characteristic curve (AUC): This is to verify the classification accuracy of the model.

2. Observed to expected ratio (O/E ratio): This is to verify the scale of the model. This metric is computed as the number of positive test examples divided by the sum of predicted probabilities for all test examples. An O/E ratio close to 1 is desired. In addition to the overall O/E ratio, we also compute the ratio across 10 buckets. For example, bucket 0 is referred as to first 10% of the sorted prediction probabilities in ascending order. We compute individual O/E ratio for each bucket labeled from 0 - 9. Note that the accuracy of prediction across different percentage bucket allows us to validate the model consistency.

We present the offline performance evaluation in the Table 1. Table 2 presents the percentile O/E ratio for pClickPush model, as the other model has similar results.

decile	O/E ratio	decile	O/E ratio
0	1.027	5	1.005
1	1.009	6	1.002
2	1.024	7	0.988
3	1.009	8	0.972
4	1.007	9	0.977

Table 2: O-E ratio in decile for pClickPush model

3.3 Notification Volume Optimization

In general we want to send the most relevant notifications to the user, while controlling for the negative impact, usually noticed through users disabling the notification channel. However, since channel disable models are difficult to build (due to the attribution issue of whether one/multiple notifications lead to the disable), we choose to control the volume of notifications since this tends to be closely related. We use two optimization approaches to identify the best notifications to send under these volume constraints - global and personalized. We describe those below

3.3.1 Global threshold. The problem of whether to send a notification can be posed as a multi objective optimization problem. The primary objective is to maximize the overall engagement via notifications while constraints are introduced to restrict the total number of notifications sent. Let u be the user index and $i \in I(u)$ be the index of a notification candidate for user u. Also let N be the total number of notification candidates across all users. Let p_{ui} be the predicted engagement if the user u is sent the notification i, and x_{ui} the probability that the notification i is sent to the user u. We can use either of the models described before to determine the predicted engagement. Then

$$max \sum_u \sum_{i \in I(u)} p_{ui} x_{ui} - \gamma \frac{1}{2} \sum_u \sum_{i \in I(u)} x_{ui}^2 \tag{2}$$

$$s.t \quad \sum_u \sum_{i \in I(u)} x_{ui} \leq \alpha N \tag{3}$$

$$\forall_{ui} \quad 0 \leq x_{ui} \leq 1 \tag{4}$$

The first term in the objective above is the overall engagement, but we also have added a quadratic regularization term to it. This is to allow the original problem to be strongly convex in addition to allowing us to easily convert to simplified dual solutions below. This is necessary to allow us to implement the solution in a streaming fashion, as we will see in the see ahead. The weight for the regularization term, γ is typically chosen to be very small to approximate the original intended objective. We also limit the total number of notifications sent to a predetermine proportion α of total notification candidates. Let λ^* be the optimal Lagrangian for the first constraint. The optimal solution can be expressed as [10]

$$x_{ui} = \Pi_{[0,1]} \frac{(p_{ui} - \lambda^*)}{\gamma} \tag{5}$$

where Π is the projection operator. When $\gamma \to 0$, then x_{ui} can be approximated to be either 0 or 1 according to

$$x_{ui} = 1 \iff p_{ui} \geq \lambda^* \tag{6}$$

Note that it is also possible to directly find λ^* without searching for the optimal solution. Let $IC(x)$ be the inverse cumulative distribution function of p_{ui} calculated over the entire candidate set across all members. Then

$$\lambda^* = IC(\alpha) \tag{7}$$

In general, we train the above model on historical datasets. The optimal λ obtained can then used as a filtering threshold. Any new notification is scored by response prediction models and using the formula in Eq. (6) to decide send or not. Therefore, the decision for each notification is independent of other notifications even in the presence of coupled constraints which allows it to be implemented in a streaming system.

3.3.2 Personalized thresholds. In general, we have observed qualitatively that the global solution above can send multiple notifications to active users who regularly interact with content, while sending few or no notifications to our lesser active users. Also, when implementing the above solution, we want to ensure that we do not send too many notifications to a single user. This is generally done by adding additional constraints to drop the notification if the number of notifications sent during some time window exceed a predetermined threshold. Since we score each notification as they come in, we may actually send a suboptimal notification that arrived earlier and exhaust the constraint on number of notifications before more optimal and high engaging notifications are generated.

To improve the solution by addressing the two drawbacks above, we propose to generate personalized thresholds. We formulate the problem for each user u (We drop the index u in the following equations). For a particular user, let d be the index for the day of the week and j be the index of a notification candidate for day d. Let p_{dj} be the predicted engagement for a notification j in a day d. For each user, the send/no send decision is the solution of the following optimization problem, which constrains the number of notifications sent to a user to be less than c_d which is a positive integer.

$$max \sum_{d,j} p_{dj}x_{dj} - \gamma\frac{1}{2}\sum x_{dj}^2 \tag{8}$$

$$s.t \quad \forall_d \sum_j x_{dj} \le c_d \tag{9}$$

$$\forall_{d,j} \quad 0 \le x_{dj} \le 1 \tag{10}$$

If λ^{d*} be the optimal lagrangian for the dth constraint $\sum_j x_{dj} \le c_d$. Then similar to the solution above, when we assume $\gamma \to 0$, x_{dj} is either 0 or 1 according to

$$x_{dj} = 1 \iff p_{dj} \ge \lambda^{d*} \tag{11}$$

One property of the solution above is that the thresholding function changes for each time period. Like the solution to the global optimization problem, it is possible to obtain the threshold for each period as

$$\lambda^{d*} = O^{c_d}(\{p_{dj}\} \quad j \in 1...n_d) \tag{12}$$

where O^k is the k'th ordered statistic of the given set.

As one can expect, when applying the solution above directly to score new notifications, we often find that no notifications are sent for a large subset of users. This is because, unlike the global solution, we do not have enough samples per user each day to identify the underlying statistical distribution of the notification engagement for each user which makes the estimate above very noisy. We found the following estimate to work slightly better in practice. We first compute the second highest utility p_{di}^+ for each day, and use a single threshold which is the minimum of these estimates for the entire time period of consideration. This estimate allows us to increase coverage of our notifications with a slight compromise on quality.

3.3.3 Channel Selection. Push notification is more intrusive than in-app notification with a badge update. This requires us to be more judicious in its use, ensuring that only the highest quality notifications are sent in this channel. We split the channel selection problem into two steps. First we decide whether to send a notification, and then decide on which channel to send the notification.

We identify a threshold beyond which we sent the notifications as a push using the global threshold method. This is to ensure that we maintain high quality, as personalized thresholds (as describe in Section 3.3.2) may lower thresholds to increase coverage for some members who may not have access to quality notification candidates. We use a lower value of α ($\alpha^{push} \ll \alpha^{send}$) in the optimization approach to derive the global thresholds, as we want to send very few notifications as push. From 7, we can see that the threshold for sending a push notification is higher (in practice much higher) than the send/no-send decision thresholds.

4 NEARLINE INFRASTRUCTURE FOR LARGE SCALE NOTIFICATIONS

We want to generate notifications to inform our members soon after when activity happens in the members' connections or following network. Because most social-network activities are time-sensitive, sending nearline notifications as connection/followee shares or likes an update helps to increase engagement, create viral loop in the network and facilitate members' conversation with their network. We implement our near-line relevance system by utilizing Apache Samza [2, 14], which is a distributed system for stateful and fault-tolerant stream processing. We plug in RocksDB [8] to Samza for stateful processing. In our case, the state is typically our features.

Figure 3 describes the system diagram. First, we describe the First Pass Ranker (FPR). In this system, social-network activities are continuously ingested into a pre-fanout processor through Kafka [1, 11] queue. Upon receiving an activity, the processor fetches activity's actor features and item features (see above discussions for feature descriptions), and puts them to compose the content item. Content item is then passed to a layer called fanout in which recipients for each item are fetched from an external graph service, where recipients are the actor's connections or followers. After the fanout, an item is expanded to multiple notification items, one for each recipient. Those items are handled by a repartitioner that partitions the items based on their recipient id and distribute them to the scorers. As items arrive, each scorer container fetches both recipient features and edge features, performs feature transformations and computes the score from the response prediction models. With this score, FPR performs the first filter (with a predefined low threshold) to remove the low quality or irrelevant content to reduce volume and save computational cost for downstream systems.

The items that pass the first filter are sent to our Second Pass Ranker (SPR) notification service platform. The SPR system is also a nearline system built upon Samza, which is partitioned by recipient again. This notification service processes notifications from multiple FPR rankers thus maintains the full historical notification profile for a recipient in its local database. In addition, this system is responsible to consume multiple LinkedIn realtime tracking events for member and store them as recipient real-time features. When notification item arrives, SPR fetches the real-time features as well as member notification profile features and uses both the first pass ranker score and the features to compute the SPR (final) score. At this layer, we make channel selection decision based on thresholds and the SPR score. Depending on the decision, a notification is either sent to push channel plus in-app channel or in-app only channel or even being dropped.

These systems interact with the offline training and feature producer jobs. Data and features generated from the FPR and SPR are snapshotted and sent back to offline training pipeline to further train response prediction models. New models can then be deployed to the service where they can be A/B tested and ramped.

Across the the whole pipeline, features are created and accessed in multiple ways:

- First Pass Ranker - Actor/Item features: These are retrieved by issuing a query to an external service.
- First Pass Ranker - Recipient/Edge features: These are pre-computed offline and then pushed to the FPR scorer in daily basis. Features are partitioned by recipient and stored in scorer's local RocksDB to expedite the fetching to under 10 ms. This design choice is necessary to scale feature fetches because of the massive amount of requests post fanout.
- Second Pass Ranker - Real time features: Real time features are gathered by listening to real time tracking signals. For example, if a push notification is tapped on by a specific user, the event will get fired and send to Samza nodes through Kafka in realtime. These features are sharable/common across all notifications so they are located here. Another important feature is member notifications profile, i.e. how many notification has been delivered for a member. This type of features have strong indication of receptiveness of further notifications.

Note that the offline feature producer generates (potentially preprocessed) features and pushes them to the service so that they can be used for online scoring.

Since the features are stored at different places, scoring will also happen individually at each of these components as well.

- FPR Scoring: Scoring in the FPR occurs after fanout. This is mainly to score items with static features (recipient and edge) and item features which are available in the post fanout request and filter out irrelevant items to reduce processing time and load for the SPR.
- SPR Scoring: As discussed before, some features are more accurate within the online service nodes. For example, the number of notifications sent within the recent x hours or the time elapse since the last visit. These features are only available at service side. We pass the notification items with FPR score to the online system where it fetches realtime features and re-scores the items for SPR (Second-Pass-Ranking) score. The final decision is computed based on SPR (i.e., channel selection)

Because of the rich connectivity of LinkedIn network, even after aggressive fanout selection the number of notifications generated after fanout could expand to multiple order of magnitudes of the number of input content items which brings up the scalability issue. With the high-performance streaming computing system and recipient based partition, we can easily scale out the system to increase the throughput capacity to meet the performance requirement. We also use local feature stores for recipient and edge features in the scorer to allow high QPS feature fetches. Additionally, filtering at the FPR limits the loads on the SPR. Using the Nearline infrastructure we are able to deliver notifications to our members in a few seconds from the point the activity occurs.

5 EXPERIMENTS

We set up an online A/B test experiment to evaluate the performance of our notification relevance models. The goal of the experiments is also to allow us to explore the advantages and disadvantages of different modeling design choices in our relevance systems. Members in the treatment bucket received the notifications for their network's activity on content based on the relevance decision. The baseline is send-none bucket in which members do not receive any notification belonging to those notification types. In some related work [10] and [9], the baseline is the send-all bucket (members receive all emails without any filter). In our case, the volume of notifications a member in the send-all bucket would receive would be extremely intrusive and detrimental to the user experience.

To evaluate the success of our models, we collect several metrics of interest. For engagement, we measure sessions as the key criteria. In general, as we send these notifications we expect users to engage with the LinkedIn app to consume the content they have been notified about. To measure the negative impact of notifications, we track channel disables. In addition, we can measure user coverage using a member day metric (number of members who received at least a notification on the day multiplied by number of days) . User coverage is an important growth criteria which attributes long term benefit, e.g., converting dormant users to active users. When trying to improve the quality of models, we use click to send ratio (CTS) as the main metric.

5.1 Global threshold

We first explore using the global threshold model for channel selection, as described in Section 3.3. Using this model we derive two thresholds for each model. The first (and less selective threshold) is used to select the updates which are sent as an inapp notification. The higher threshold is used to decide whether to send a push update or a badge update. For each notification, we use the features

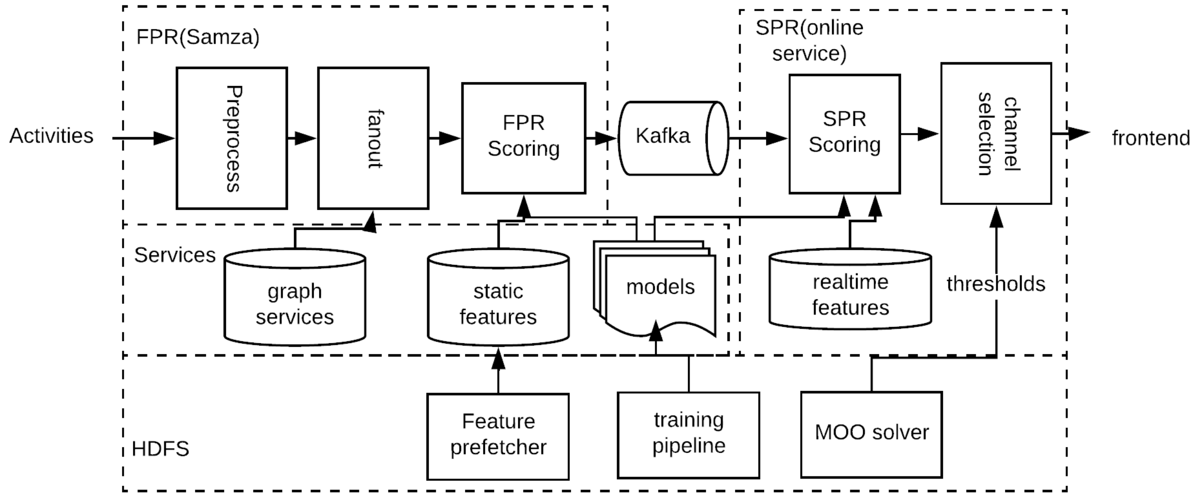


Figure 3: System Diagram

fanout type	session lift (%)	disable incr. (%)	member day (%)
EA	+1.3	+2.2	+2
CS	+2.1	+5.0	+2.7

Table 3: Experimental results with pClickPush model comparing different fanout logic

% as in-app	% as push	session lift (%)	disable increased (%)
50%	10%	+2.4	+8.2
50%	5%	+2.4	+0

Table 4: Experimental results with pClickPush model

to generate a score, and compare against these two thresholds to determine the notification send and channel decision.

5.1.1 Effect of fanout choices and push thresholds. For the first set of experiments, we use the pClickPush model to send notifications. The baseline used here is a bucket where we send no content notifications. The first experiment compares the effect of two fanout choices - edge affinity based (EA) and connection strength based (CS). The baseline is the no-send bucket. As we expected, the results captured in Table 3 show that by using a more wide reaching connection strength based fanout, we are able to increase member day metrics. In both the cases, we are seeing an increase in sessions. The session lift, member day numbers reported are site-wide lift numbers which directly correlates to increase in revenue. Though we see a significant relative increase in the disable rate, because the disable rate is so small, the absolute change in the disable rate is very small. Since connection strength based models provide better coverage, all the following models use this fanout procedure.

One of the drawbacks of the previous models is the increase in channel disable rate when we send these notifications. We identified that push notifications (due to their intrusive nature) cause higher disable rates. We launched and compared two variants, where we fixed the total volume of notifications sent, while varying the volume of push notifications. The results, summarized in Table 4 show that decreasing push notifications does not lead to decreasing in sessions, but reducing disable rate. Again, we compare the models against the no-send bucket.

The experiment highlights that badge updates are very effective yet non-intrusive medium to engage members attention and bring them to the app. We use the more restrictive volume targets for the next set of experiments.

Notification type	Normalized historical CTS %
MITN	2.5
PBYN	1.32
SBYN	1.31
TA	1

Table 5: Engagement rates by notification types, compared to engagement of TA notifications

5.1.2 Improving notification quality. The experiments in this sub section compare the effectiveness of different response prediction models (pClickInapp and pClickPush) on the quality of the notifications. The experiment in this section are motivated by the following intuition. For push notifications, an user can only infer minimal information about the type and content of the notification, due to limited screen space and less discerning contextual information. In contrast, in app notifications provide rich context, including title, author and type of notification. We believe that pClickInapp models may be a better predictor of overall quality of the notifications.

When serving notifications using the pClickPush models, we noticed that different notification types have different levels of engagement on the app, as seen in the Table 5 which contains CTS of notifications served by the previous model. MITN is considered a far superior quality than TA. Our pClickPush model is not able to predict these CTS, and as a result, the notifications that we send may be suboptimal for user experience.

Model	% Incr. in Sends	% Incr. in Clicks	% Incr. in CTS
InApp	-2.17	23.73	26.4

Table 6: pClickInApp Vs pClickPush - Overall A/B testing results

Notification type	% Increase in CTS
MITN	-24.6
PBYN	+31.9
SBYN	+5.6
TA	+46.2

Table 7: pClickInApp Vs pClickPush - Increase in clicks by notification types

The inapp model (pClickInapp) predicts the members engagement on inapp notifications. When we use this model to send notifications while keeping volumes constant, we expect to reshape the distribution of notifications. Table 6 shows the results of an A/B experiment comparing the pClickInapp model with the pClickPush model. We use CS fanout for pClickInApp model and we found similar lifts as shown in (Table 3) when using CS fanout Vs EA fanout. While we try to match volumes with the previous models, we have other rule based filters which cause slight changes in the volume of notifications sent by the new model. As expected, we see a huge increase in notification CTS and notification clicks. We did not see any statistically significant change in sessions as expected. We can also see the increase in CTS by notification type in Table 7. For all notification types except MITN we see an increase in CTS, for MITN the decrease in CTS is explained by increased sent volume.

5.2 Personalized thresholds

As described in Section 3, by adding constraints for per-user volume we can generate a personalized threshold for each user. To experiment the benefits of this approach, we added a set of per-user constraints that enforce each user to receive at most one in-app notification per day. We generate a personalized threshold for each user from last two weeks of data on a daily basis and push to our nearline scoring systems. We use the connection strength fanout along with the inapp model in this set of experiments.

We compare the online metrics of the personalized threshold model against the one with uniform global threshold in Table 8. Both of these models are trained on the inapp datasets. In addition to session lift (%) and CTS increase (%), we also take user coverage metrics into consideration.

Note that compared to global threshold, personalized threshold strategy reshapes user coverage in finer granularity. It lowers thresholds for less active users and provides more liquidity for them, and increases thresholds for active users and only allows high quality content to be sent. In order to verify it, we group users to four segments according to their activeness: daily active users (DAU), weekly active users (WAU), monthly active users (MAU), and dormant users (Dormant). Then, measure the metrics for each segment. One byproduct of increasing the number of notifications is the decrease in notification CTS metric as we send more notifications for some user segments who were previously unreachable. On the other hand, we also see an increase in CTS for our daily active users as we choose a more restrictive threshold which prevents a suboptimal notification from exhausting daily notification limits as described in Section 3.

Segment	session (%)	send (%)	member day (%)	CTS (%)
overall	+0.4	+1.9	+1.2	-2.2
DAU	0.0	-1.0	+0.1	+0.4
WAU	+1.3	+3.9	+0.9	-5.1
MAU	+0.7	+10.3	+1.5	-8.6
Dormant	+0.4	+15.3	+1.4	-0.5

Table 8: Personalized vs. Global Threshold: session & coverage by member segments

6 RELATED WORK

Push notifications were first introduced by Apple in 2009 [13]. In 2010, Google released its own service, Google Cloud to Device Messaging [15]. Mobile app users have, over time, gotten used to receiving information from notifications, and when used judiciously, notifications can be very effective in driving mobile engagement.

One problem for generating activity notifications is to identify relevant content. [5] discusses how to find high quality content from social media by utilizing the connection graph. Identifying relevant activity in a member's network is also a commonly studied problem in the context of ranking a news feed [3]. [17] describes an inference-based social network content pre-fetcher that loads and caches relevant content before the app launch to address access delay and wi-fi network availability issues.

However, there is very little published work about strategic content dissemination in a social network via push notifications. Related work, in the context of emails, has been done by Gupta et al. [10] who have built a framework to email relevant content to members by optimizing positive metrics and limiting negative metrics at the same time. A follow-up work [9] extends the optimization to a broader set of metrics in the ecosystem. As addressed in [16], push notifications should be relevant, novel and timely. Unlike email which often are generated in batches, push notifications for social activities need to be sent in a timely fashion, i.e., as soon as the activity happens. This imposes unique challenges in the system and modeling design. We built our relevance backend in a near-real time system and we successfully shortened the end-to-end latency to a few seconds.

Our infrastructure for the notification system currently handles a massive volume of messages every day. Every second, there are up to tens of thousands of activities ingested, with tens of millions of candidates (generated from the tens of thousands of source activities) scored. This paper discusses strategies to distribute relevant social activities through push notifications, and the system architecture for a large scale near-line system that tackles time sensitive relevance problems.

7 CONCLUSION

To the best of our knowledge, this is the first reported work that optimizes the distribution of content activity notifications through a near real-time system. We define the problem as deciding which "filter-eligible" activity based notifications to send in order to drive member engagement. We describe our experience building a near-line service that sends such notifications to mobile app users within a few seconds to a few minutes. To ensure a great member experience, we limit the maximum number of such notifications any member can receive. This translates into a constraint which is part of a multi objective optimization formulation to balance this send constraint with other engagement utilities.

To fit the near real-time system, we prove that the solution can be approximated as a (set of) threshold(s) with which we can make the send and channel selection decision(s) in near real-time. Through A/B testing, we illustrate that using the relevance models to control notifications can effectively optimize various utilities. The best performing models are currently generating notifications to our members.

The approach presented here can be expanded to support most "filter-eligible", near real-time notification systems - by defining the fanout logic to generating a candidate set, building the corresponding response prediction models and then solving the optimization problem to achieve the right trade-off on the metrics of interest. In addition, the approach presented here can be expanded to consider more utilities - for example update virality, downstream virality (for activity notifications) and notification dismisses. We are pursuing some of these directions currently.

REFERENCES

[1] 2018. Apache Kafka. (2018). https://kafka.apache.org/

[2] 2018. Apache Samza. (2018). http://samza.apache.org/

[3] Deepak Agarwal, Bee-Chung Chen, Rupesh Gupta, Joshua Hartman, Qi He, Anand Iyer, Sumanth Kolar, Yiming Ma, Pannagadatta Shivaswamy, Ajit Singh, and others. 2014. Activity ranking in LinkedIn feed. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1603–1612.

[4] Deepak Agarwal, Bee-Chung Chen, Qi He, Zhenhao Hua, Guy Lebanon, Yiming Ma, Pannagadatta Shivaswamy, Hsiao-Ping Tseng, Jaewon Yang, and Liang Zhang. 2015. Personalizing linkedin feed. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1651–1660.

[5] Eugene Agichtein, Carlos Castillo, Debora Donato, Aristides Gionis, and Gilad Mishne. 2008. Finding High-quality Content in Social Media. In *Proceedings of the 2008 International Conference on Web Search and Data Mining (WSDM '08)*. ACM, New York, NY, USA, 183–194. DOI: http://dx.doi.org/10.1145/1341531.1341557

[6] Andrew Bosworth and Chris Cox. 2013. Providing a newsfeed based on user affinity for entities and monitored actions in a social network environment. (March 19 2013). US Patent 8,402,094.

[7] LinkedIn Corporation. 2016. Photon ML. https://github.com/linkedin/photon-ml. (2016).

[8] Facebook. 2016. RocksDB. https://github.com/facebook/rocksdb/wiki. (2016).

[9] Rupesh Gupta, Guanfeng Liang, and Romer Rosales. 2017. Optimizing Email Volume For Sitewide Engagement. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM '17)*. ACM, New York, NY, USA, 1947–1955. DOI: http://dx.doi.org/10.1145/3132847.3132849

[10] Rupesh Gupta, Guanfeng Liang, Hsiao-Ping Tseng, Ravi Kiran Holur Vijay, Xiaoyu Chen, and Romer Rosales. 2016. Email Volume Optimization at LinkedIn. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 97–106. DOI: http://dx.doi.org/10.1145/2939672.2939692

[11] Jay Kreps, Neha Narkhede, Jun Rao, and others. 2011. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*. 1–7.

[12] LinkedIn. 2018. About Linkedin. (2018). https://about.linkedin.com/

[13] Donald Melanson. 2009. iPhone push notification service for devs announced. (2009). https://www.engadget.com/2008/06/09/iphone-push-notification-service-for-devs-announced/

[14] Shadi A Noghabi, Kartik Paramasivam, Yi Pan, Navina Ramesh, Jon Bringhurst, Indranil Gupta, and Roy H Campbell. 2017. Samza: stateful scalable stream processing at LinkedIn. *Proceedings of the VLDB Endowment* 10, 12 (2017), 1634–1645.

[15] Daniel Rubio. 2010. Google Cloud Messaging for Android (GCM) Unveiled, to Replace C2DM Framework. (2010). https://www.infoq.com/news/2012/08/GoogleCMReplacesC2Dm

[16] Luchen Tan, Adam Roegiest, Jimmy Lin, and Charles L.A. Clarke. 2016. An Exploration of Evaluation Metrics for Mobile Push Notifications. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '16)*. ACM, New York, NY, USA, 741–744. DOI: http://dx.doi.org/10.1145/2911451.2914694

[17] Yichuan Wang, Xin Liu, David Chu, and Yunxin Liu. 2015. Earlybird: Mobile prefetching of social network feeds via content preference mining and usage pattern analysis. In *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM, 67–76.

[18] Mark Zuckerberg, Andrew Bosworth, Chris Cox, Ruchi Sanghvi, and Matt Cahill. 2012. Communicating a newsfeed of media content based on a member's interactions in a social network environment. (May 1 2012). US Patent 8,171,128.