Interest-group discovery and management by peer-to-peer online social networks
Jianping He, David J. Miller and George Kesidis
CS&E and EE Depts, The Pennsylvania State University, University Park, PA, 16802
{jxh522,djm25,gik2}@psu.edu


Abstract—We address management of latent, emerging interest groups (IGs), spanning both unsupervised, distributed IG discovery and anycast-query forwarding, in peer-to-peer (P2P) on-line social networks. The P2P network is assumed to have at least one layer of superpeers (Diaspora pods) that support and moderate a group of ordinary peers/clients. There are a number of challenges here, including: i) semantic processing at scale to efficiently disambiguate word meaning in queries; ii) unsupervised estimation of the number of active IGs (essentially, cluster number estimation); iii) detection of IG churn and emergent IGs; iv) design of optimal query forwarding strategies to maximize successful query resolution and minimize the required number of hops, while achieving practical local cache searching and network communications overhead. We assume a common, fixed keyword lexicon for query formation, and propose an unsupervised online clustering algorithm, with customized Bayesian Information Criterion based model-order selection, running on each superpeer, to estimate the set of active IGs and to help achieve efficient query forwarding. The proposed method is evaluated on realistically simulated P2P network traffic, against both exhaustive cache searching and a random walk strategy.

I. INTRODUCTION

In the last decade, the rapid emergence of online social networks (OSNs) has fueled interest in social phenomena by computer scientists. There are now many enormously popular enterprises operating social networks, e.g., Facebook, Google+ , Twitter, and LinkedIn. These networks are attaining a prominent role in peoples’ lives. For example, most people in the U.S. have at least one Facebook account [21]. There are more than 900 million active users on Facebook [9], spending more than 7 hours monthly, on average [8]. Facebook or Twitter may be used to find new friends and share news and other content. From the user’s point of view, finding heretofore unknown users with certain interests or knowledge is one of the core utilities of social networks.

The current centralized OSN frameworks come with privacy concerns – many users are reluctant to share their personal information with a single for-profit company. Facebook takes advantage of its user profiles to distribute advertisements which represent more than 80% of its revenue [7]. Recent news articles have highlighted privacy issues regarding illegal information harvested by commercial enterprises, e.g., through the use of cookies [13], [2]. Information of interest includes the physical locations of mobile device users [3]. Also, privacy issues are associated with centralized storage of vast quantities of personal data, particularly online social interactions, by commercial entities for targeted advertising (e.g., [27] recently), which can also be disclosed under government warrant (particularly in countries with lax privacy laws, or during periods of civil unrest or war, but even in the U.S., e.g., [19]), and are tempting targets for hackers, e.g., [5].

Such “big brother” privacy issues have motivated “localized” social network architectures such as Diaspora [1], [15]. However, adoption of Diaspora has been limited due to the lack of internetworking of Diaspora pods controlling local social networks so as to conveniently connect different social groups. Of course, a distributed social network architecture comes with its own privacy challenges, particularly “user-to-user” privacy problems (typically solved by existing encryption, onion-routing and proxying/indirection mechanisms whose strength may be improved though the use of, e.g., fast-flux methods presently employed by botnet controllers).

In this paper, we focus on the management of latent (emerging, nascent) interest groups (IGs, a.k.a. communities of interest (Cols)), spanning both IG discovery and anycast-query forwarding, in P2P OSNs1. The P2P network is assumed to have at least one layer of superpeers (Diaspora pods) that support and moderate a cluster of ordinary peers/clients. IG identification for P2P networks has significant distributed semantic processing challenges at scale – dealing with spelling mistakes, synonyms, and translation issues for the keywords characterizing IGs (and for the queries targeting them) is much easier with substantial centralized resources, as available for web search engines. Though in the following we assume all participants draw keywords from a common dictionary, with each word possessing unique meaning, online thesauri (e.g., Princeton’s WordNet) and pods’ dictionaries can be used to disambiguate word meaning in queries and more accurately map their keywords to those characterizing known IGs2.

Also, a P2P OSN, particularly one pertaining to specific content, may need to interact with a content-centric networking (CCN) architecture [23], [4], [24] to more effectively deal with well-established (not latent) IGs at scale. That is, the first pod handling an IG anycast query may first “check”...

1Note that existing online IGs are today managed centrally within the aforementioned OSNs and others, e.g., deviantART.com.

2Considering that query keywords will not be arranged in sentences, grammatical “part of speech” classification will likely not be helpful here.
with a CCN and, failing that, the P2P OSN to which the pod belongs will handle the query in a distributed fashion. The birth of an IG (or division of an existing IG) may cause query-forwarding failure, owing to proximity of the new IG to existing IGs in keyword space (i.e., through common characterizing keywords), despite significantly different peer memberships. Queries to such latent IGs may be identified by rules to detect outlier queries or those “at the boundary” between two or more IGs. Such detected queries may be forwarded by a combination of random-walk and/or limited-scope flooding (RW/LSF) (instead of classification to the nearest IG in keyword space), in order to identify peers belonging to these nascent IGs. An alternative approach is discussed in Sec. VI below.

This paper is organized as follows. In Sec. II, we discuss related prior work on IGs in social networks. We give an overview of our IG anycast-querying P2P OSN system in Sec. III. In Sec. IV, we describe a mixture model and associated Expectation-Maximization (EM) learning framework to identify the trending IGs and their keyword characteristics. In Sec. V, we present the results of a performance study for a static group of latent IGs. An adaptive framework for dealing with dynamic latent IGs (IG churn) is discussed in Sec. VI. Finally, we conclude in Sec. VII.

II. RELATED PRIOR WORK ON INTEREST GROUPS IN SOCIAL NETWORKS

The study of search and group structure in social networks has a long history, e.g., [28], [22]. Recently in [14], a framework was given for analyzing a group of (anycast) queries to IGs, including semantic similarities, ultimately affecting measures of social ties between peers. For semantic similarity, they assume peers and queries are both profiled (via standard keywords) into a hierarchical “knowledge index”, allowing the distance between a query and a peer to be reckoned, with an associated “optimal” local forwarding decision. Capturing how social ties are learned through query resolution is an aim of [14]. Here, we are more interested in developing from first principles a practical online framework for “knowledge index” formation and management for query resolution. In another recent paper [16], latent IGs are also considered, with individual peer-nodes potentially belonging to more than one (i.e., the IGs are overlapping). [16] assumes a particular statistical model that relates peer and link features to the probabilities that peers belong to latent IGs. This model can be used to infer IG memberships given peer attributes, and also to infer peer attributes given IG memberships. Unlike [16], we do not presuppose a particular model of IG membership or an associated peer/IG feature space (e.g., the “islands” model of [12]). Rather, in our approach, each pod learns the set of IGs by clustering the successfully forwarded queries in its cache.

III. SYSTEM OVERVIEW AND ASSUMPTIONS

Assume there are \( N \) pods in the system, denoted \( \{P_1, P_2, \ldots, P_N\} \), used to form a P2P overlay network. Pod \( P_i \) can communicate with another pod \( P_j \) so long as \( P_i \) knows \( P_j \)'s IP address or other identifier. If \( P_i \) knows \( P_j \)'s address, they are deemed neighbors, i.e., there is a link between \( P_i \) and \( P_j \) in the Diaspora overlay network. We assume all links are symmetric. Due to limited system resources (including pod memory, transmission bandwidth and computing), each pod only maintains a small subset of pods as its neighbors. A user can directly communicate only with its pod.

The basic assumptions in this preliminary study are:

- A standard dictionary of \( \lambda \) uniquely defined keywords is shared among all users. Further, there is no a priori knowledge of semantic overlap between different keywords, i.e., it is a priori unknown which subsets of keywords may characterize (or have propensity for characterizing) an IG.
- Pods always know their active users since users need to log into their pods when entering the system.
- All social links are symmetric.
- No user or pod leaves or joins the system after initialization, i.e., the social graph is static.

A. Characterizing Interest Groups and Query Distance

Again, an IG is defined by a set of users who share common interests (and different IGs, just as pods, may have overlapping user membership [28], [22], [12]), and an IG can be uniquely characterized by a subset of unambiguous keywords from the common dictionary. A user who wishes to contact other users belonging to a common IG will launch a query using keywords indicative of that IG. However, every IG is initially latent, meaning users do not know the complete set of keywords characterizing an IG, and pods are not initially aware of the IGs of interest to their users. Moreover, some keywords may appear in more than one IG. In our study, we encode each IG as a bit vector of length \( \lambda \), where the \( a^\text{th} \) bit (1 ≤ \( a \) ≤ \( \lambda \)) is 1 if the \( a^\text{th} \) keyword “belongs to” the IG; otherwise the bit is assigned 0. Hence there are \( 2^\lambda \) possible IGs (based on all possible subsets of \( \lambda \)-bit queries). In practice, the number of active IGs is far less than \( 2^\lambda \). Like an IG, a query \( q_i \) can also be encoded as a bit vector. For our study, we assume a query to an IG is a randomly selected subset of the keywords characterizing it. For a given set of users, pods, and IGs, we are interested in how pods can correlate queries and thereby learn the trending latent IGs.

Let query \( q_i \) be a bit vector of length \( \lambda \), i.e., \( q_i \in \{0,1\}^\lambda \). There are many different possible distance definitions, e.g., Hamming distance for binary vectors or Euclidean distance for arbitrary vectors. We could also define the query distance between \( q_i \) and \( q_j \) as the reciprocal of their similarity

\[
d(q_i, q_j) = \left( \frac{\langle q_i, q_j \rangle}{|q_i||q_j|} \right)^{-1},
\]

where \( \langle q_i, q_j \rangle = \sum_{a=1}^{\lambda} q_{i,a} q_{j,a} \) is the inner product of \( q_i \) and \( q_j \).

3Implicitly, a kind of consensus protocol among pods would be in place to identify established IGs and transition them over to the CCN.

4The IGs are latent in that their peer memberships are not known a priori.
and $|q| = \left( \sum_{i=1}^{\lambda} (q_i^2) \right)^{\frac{1}{2}}$ is the norm of vector $q$. Obviously when $\langle q_i, q_j \rangle = 0$ (i.e., orthogonal queries have no common keywords), $d(q_i, q_j) = \infty$.

B. Forwarding: Search within a pod's query cache

Under reverse path forwarding (RPF) [25] [6], each query message keeps a record of pods it has visited to avoid forwarding loops, and a Time to Live (TTL) field to limit forwarding range and overhead. TTL is initialized by the source pod and decreased by 1 by each forwarding/relaying pod. When a query is either successfully resolved or is determined to be unsuccessful due to keyword mismatching or TTL expiring, the result is reverse-forwarded from the destination pod to all forwarding pods. Each pod on the path caches the query result to facilitate its future forwarding.

Each pod caches the $s$ most recently successfully resolved/forwarded queries. When a pod finds its local users cannot resolve query $q$, it will forward $q$ to another pod. To utilize the pod’s cached queries and their results in choosing the next hop pod, a trivial approach is to find an exact matching query in the cache and forward $q$ according to the matched query’s result. But it may not be easy to find an exact matching query due to the huge number of possible queries, IG evolution, etc. A better, distance (similarity) based approach is to find the closest (most similar) query to $q$ in the cache. However, such (exhaustive) search may be impractical when the cache size is large. Moreover, to have a high rate of successful query resolution, the pod’s cache size will indeed need to be large (and to grow with the number of active IGs).

To facilitate efficient resolution/forwarding of a query $q$, a pod may work with learned cluster centroids (in its forwarding table) rather than attempting to find an exact match to $q$ in its query cache. Indeed, search within the query cache could be restricted to cached queries that belong to clusters with centroids nearest to $q$. If there are a large number of clusters in the forwarding table, then for efficient search their centroids can be represented using a balanced decision tree structure (e.g., chap. 12 of [11]). Alternatively, the Delaunay graph formed by the centroids in keyword space could be sequentially searched starting at one centroid and greedily choosing the neighbor centroid closest to $q$.

Another alternative approach could involve use of hierarchical distributed hash tables (DHTs). In addition to achieving computationally efficient query resolution, clustering of the cache entries provides (for each pod) an estimate of the active IGs, i.e., each cluster is an estimated IG. Thus, clustering also provides a mechanism for each pod to (dynamically) learn the currently trending IGs.

Hence, we propose an approach where each pod clusters all queries in its cache and, where, for a given query, the next hop is found in a hierarchical fashion that exploits the clustering solution. There are many existing clustering algorithms, e.g., K-means [10], [18] and Expectation-Maximization [20]. Two important characteristics of a clustering solution for our problem are that: i) the clusters should be on-line adjusted as new queries arrive, to reflect the changing content in the cache; ii) accurate cluster number estimation is sought, since this number is the pod’s estimate of the number of active IGs. The latter problem will be addressed by Bayesian Information Criterion (BIC) based model-order selection [26].

In our proposed framework, there are four caches for each pod: LocalSuccessfulCache (LocalUnsuccessfulCache) which stores resolved (unresolved) queries forwarded to local users, and GlobalSuccessfulCache (GlobalUnsuccessfulCache) which stores resolved (unresolved) queries forwarded to other pods. There are two fields for each cache entry: query and nextHop. nextHop is either a local user identifier in LocalSuccessfulCache or LocalUnsuccessfulCache or a pod identifier in GlobalSuccessfulCache or GlobalUnsuccessfulCache. In future, we may add fields, such as average remaining hops to resolve queries, and success ratio.

Queries in the SuccessfulCaches are grouped into a number of clusters, cf., Sec. IV. Queries assigned to the same cluster are essentially estimated to belong to the same IG. The keyword centroid of each cluster is a vector of length $\lambda$, i.e.,

$\mathbf{c}_i = c_{i1} c_{i2} \ldots c_{i\lambda}$,

where $0 \leq c_{il} \leq 1$, $a = 1, \ldots, \lambda$ and $l = 1, \ldots, k$, $k$ the number of clusters. Since users’ interests may change, cached queries are re-clustered periodically, cf., Secs. V, VI.

Each pod also maintains a forwarding table of size $k$. Each row of the forwarding table (one per cluster) has two fields: a list of clustered query entries and their keyword centroid. Each query entry includes its querying keywords, next hop and a counter to record its hit frequency. The latter two fields are used for forwarding. Upon receiving a query $q$, a pod takes the following steps (failing one, the next is tried):

1) CCN lookup.
2) Search local users.
3) Find centroids closest to $q$ (let $c$ be the closest) and search these clusters within GlobalSuccessfulCache.
4) If there is an exact match to $q$, then forward the same way as for the matching cached query.
5) If $q$ is closer to $c$ than $\tau$ (< 1) times the distance between $q$ and its second nearest centroid, then use the forwarding table at $c$.
6) Forward $q$ via a RW/LSF framework (also during initialization).

C. Update the Cache and Forwarding Table

Each online cache can obviously only maintain a limited number of query results. We can simply drop the oldest cache entries in a First-In-First-Out (FIFO) manner and accordingly on-line update the clusters to reflect both the addition of the newest entry and removal of the oldest entry. Also, when receiving reverse-path-forwarded query results, a pod updates its corresponding cache entries.

IV. Mixture Model, EM Algorithm, and BIC-based Estimation of the Number of IGs

The cached queries stored in each pod are denoted $(q_1, q_2, \ldots, q_s)$, where $s$ is the size of the cache. The number of clusters (which will be estimated) is $k$. There are $\lambda$ distinct keywords used across all cached queries, where $1 \leq \bar{\lambda} \leq \lambda$.

\footnote{We did not exploit knowledge of unresolved queries, e.g., to adapt (in a supervised fashion) the forwarding decision rules, in our current approach.}
We introduce a matrix $P = [p_{j|i}]$, $(i = 1, 2, \ldots; j = 1, 2, \ldots, k)$, where $p_{j|i}$ is the (a posteriori) probability that query $q_i$ is generated by cluster $j$. The (prior) probability that an arbitrary query is in cluster $j$ is $\alpha_j$, $j = 1, \ldots, k$. Also, $B = [\beta_{j,a}]$ $(j = 1, 2, \ldots; a = 1, 2, \ldots, \lambda)$ is a probability matrix, where $\beta_{j,a}$ is the probability that the $a^{th}$ keyword of the dictionary is used in the query, given that the query was generated based on cluster $j$. These parameters define a Bernoulli mixture model, providing a generative stochastic model for the queries in the cache. The parameters of this model $\Theta = \{B, \{\alpha_j\}\}$ can be estimated, starting from initial parameter estimates, via the Expectation-Maximization (EM) algorithm, which iteratively ascends in the data log-likelihood function, with each iteration performing Expectation (E) and Maximization (M) steps.

**E-step:** The joint likelihood of a query $q_i$ and its generation by mixture component (cluster) $j$, given the parameters at iteration $t$, $\Theta^{(t)}$, is:

$$p^{(t+1)}_{i,j} = \alpha_j \prod_{a=1}^{\lambda} (\beta_{j,a})^{q_{t,i}} (1 - \beta_{j,a})^{1-q_{t,i}}.$$  

(2)

Applying Bayes rule, we obtain the new posterior cluster membership probabilities:

$$p^{(t+1)}_{j|i} = p^{(t+1)}_{i,j} \sum_{j=1}^{k} p^{(t+1)}_{j,i}.$$  

(3)

**M-step:** In the maximization step, we update $\alpha_j^{(t+1)}$ and $\beta_{j,a}^{(t+1)}$:

$$\beta_{j,a}^{(t+1)} = \frac{\sum_{i=1}^{N} p^{(t+1)}_{j|i}}{\sum_{i=1}^{N} p^{(t+1)}_{j|i}}, \quad \alpha_j^{(t+1)} = \frac{1}{s} \sum_{i=1}^{N} p^{(t+1)}_{j|i}.$$  

The value of the log-likelihood objective function, at iteration $t + 1$, is:

$$\log L(k)^{t+1} = \sum_{i=1}^{N} \sum_{j=1}^{k} \alpha_j^{(t+1)} \prod_{a=1}^{\lambda} (\beta_{j,a}^{(t+1)})^{q_{t,i}} (1 - \beta_{j,a}^{(t+1)})^{1-q_{t,i}}.$$  

(4)

Ideally, EM is terminated when $\log L(k)$ reaches a local maximum. In practice, iterations are stopped when $|\log L(k)^{t+1} - \log L(k)^t|$ is less than a predefined threshold.

Query $q_i$ is deemed to belong to cluster $j$ if $p_{j|i} \geq p_{j'|i}$, i.e., based on a MAP classification rule.

**BIC based Model-Order Selection Approach:** Each pod uses a BIC penalty to determine the number of components/clusters in its cache. A naive BIC penalty is:

$$\text{BIC}(k) = \frac{1}{2} (k - 1 + \lambda k) \log(s) - \log L(k),$$  

(5)

where $(k - 1 + \lambda k)$ is the number of free parameters to describe the mixture model parameters $^9$. However, because the cached queries represent a small subset of all possible queries, there may be some keywords that do not appear in any cached queries, i.e., there may be some zero columns in the matrix $B$. Based on this, we can get the following BIC formula:

$$\text{BIC}(k) = \frac{1}{2} (k - 1 + \lambda k) \log(s) - \log L(k),$$  

(6)

where $\lambda (1 \leq \lambda \leq \lambda)$ is the number of distinct keywords occurring over all the cached queries. The number of distinct

\begin{itemize}
  \item $^9k - 1$ “$\alpha$” parameters and $k\lambda$, “$\beta$” parameters.
\end{itemize}

An even more efficient BIC codelength penalty term can often be obtained in practice with the observation that the total number of keywords used by any given IG (and hence by any mixture component) is much smaller than the number of keywords used across all queries in the cache – i.e., there are likely many zero entries, even in the matrix $B'$. These can be encoded as follows.

1. Introduce a new bit matrix $V = [v_{i,j}]$, where $1 \leq i \leq k$ and $1 \leq j \leq \lambda$, $v_{i,j}$ is 1 if $\beta_{i,j}$ is not 0.
2. $k$, the matrix $V$, the non-zero terms in matrix $B'$, and all the $\alpha$ variables specify the mixture model.

Accordingly, we obtain the BIC criterion for this model as:

$$\text{BIC}(k) = \lambda k + \frac{1}{2} (k - 1 + \lambda) \log(s) - \log L(k),$$  

(7)

where $\lambda' (1 \leq \lambda' \leq \lambda k)$ is the number of non-zero terms in $V$. When $V$ is sparse, this BIC penalty is more efficient than the preceding ones, leading to a more accurate estimate of the number of components (IGs)$^{10}$. The chosen number of clusters

$$k_{opt} = \arg \min_{1 \leq k \leq C} \text{BIC}(k),$$  

(8)

where $C$ is a predetermined upper bound on IG clusters per pod, which can be chosen based on complexity considerations.

V. NUMERICAL STUDY FOR STATIC LATENT IGs

A. Benchmark Algorithms

We compare our algorithm with two others. In all the methods, each pod maintains a cache to store most recently forwarded queries and their corresponding results. Each query has a TTL field to limit query scope. Before forwarding a query to another pod, a pod forwards the query to a subset of its local users. If the query is not resolved, it is forwarded to another pod. The two methods we compared with are:

1. Random Walk$^{11}$ with Caching (RW+Caching): When receiving a query, a pod searches its cache for an exact match. If finding one, the pod forwards the query to the corresponding pod. Otherwise, the pod forwards the query to a (randomly chosen) neighbor.
2. Closest Query Forwarding (Closest): When receiving a query, a pod searches its cache as in RW+Caching. If it fails to find an exact match, the pod finds the cache entry with smallest Euclidean distance to the query. The query is then forwarded to the nextHop for this cache entry. Note this potentially has much more overhead than EM-BIC, with exhaustive cache search.

B. Performance Metrics

There are two classes of performance metrics. Those of the IG learning algorithm, averaged over all pods, are: IG-Number Accuracy (the ratio of the learned number to the true number of IGs)$^{12}$ and IG-Name Accuracy (the distance between a

$^{10}$For the numerical results that follow for the simple case of static IGs, the BIC penalties (6) and (7) yielded comparable performance.

$^{11}$More precisely, $N^{-1} \sum_{t=1}^{T} \log \sum_{k_{opt}=1}^{C} \text{BIC}(k)$, where $\tau_p$ is the total number of unique IGs in pod $P$’s cache and where there are $N$ pods in total.
learned IG centroid and the nearest true IG vector, averaged over all centroids for all pods). The latter metric captures accuracy of the individual learned IGs. Performances of the forwarding algorithm, averaged over all queries, are: Average Query Hops (whether queries were successfully resolved or not), and Query Success Probability: the ratio of number of successfully resolved queries to queries launched.

C. Simulation Setup

In a preliminary simulation study, we took the dictionary size $\lambda = 256$. The true number of latent IGs was 50. The number of keywords in each IG was chosen (uniformly) at random from 10 to 15. The number of keywords in each query was randomly chosen from 3 to 7. There were $N = 1024$ pods in the system. Each pod had a random number of neighboring pods randomly chosen between 3 and 4. Each pod had a random number of local users chosen uniformly between 1 and 2, where we assumed two users of the same pod had at least one IG in common. Also, each user had 2 to 3 other IGs selected uniformly at random from the 50 IGs in play. We ran the each simulation for $R = 500$ rounds/trials, where each user generated a query with probability $p = 0.5$ in every round. We assumed that the terminal (destination) user for a query contacts the user who generated the query to determine whether the query is resolved or not, and the query-relaying pods are informed of resolution success or failure.

We used (merged) SuccessfulCache size $s = 75$ for each pod. The EM-BIC update period was 25 queries. To track performance changes, each pod logged its metrics periodically, after every 50 new query-cache entries (SuccessfulCache touches). Understanding that the pods act asynchronously, the reported metrics were averaged over all pods, based on 50 new SuccessfulCache touches for each pod. Finally, the simulation was repeated 10 times, with the metrics averaged over these runs, to guarantee good statistical confidence on the results.

Our simulation parameter choices were made to avoid "trivial" scenarios; e.g., if users have large group memberships, pods’ query caches are even larger and the pod-topology is highly connected, then this obviously leads to very low mean query-forwarding length (close to 1 hop) once the pods’ query-caches fill up. Such scenarios likely do not realistically scale, particularly considering dynamic scenarios with IG and pod churn. As we simulated static scenarios, we were interested in performance as the query caches grew from empty, where random-walk plays a significant role in query forwarding, i.e., before the cache fills up for the first time and all of the static IGs of the users are essentially learned by their own pods and their neighbors. Modifying some of the parameters led to predictable performance results, e.g., larger caches led to lower query-hops with greater success for Closest and EM-BIC (and had no effect on RW+Cache, which only uses the cache for exact matching).

D. Simulation Results and Discussion

From Figures 1 and 2, the forwarding performance of EM-BIC is better than Closest and (as expected) much better than RW+Cache. This can be explained as follows. There are essentially two performance stages as a function of SuccessfulCache touches – a “transient” stage, for SuccessfulCache touches $<200$, where there is great variety in the queries and a large number of IGs represented in the cache, and an “equilibrated” stage, for SuccessfulCache touches $>200$, where both the variation in the cached queries and the number of represented IGs is much lower. In the transient stage, the advantage of EM-BIC over Closest in query hops is explained by the greater reliability of a centroid (an average over queries) compared to an individual cached query, considering the keyword overlap among IGs which may cause incorrect forwarding by the closest query at this (transient) stage. The advantage in the equilibrated stage is explained as follows. EM-BIC query forwarding first assigns the current query to a cluster via the MAP rule and then finds the nearest (Euclidean distance-based) cached query belonging to this cluster. A cached query that is in fact closer to the current query but which does not belong to this (MAP-assigned) cluster is more likely to be an outlier of the cluster to which it belongs and, thus, to be less reliable for forwarding. Again, EM-BIC could be further improved by tracking average residual hops to the destination in the SuccessfulCache (easily inferred from query TTL under RPF), i.e., SuccessfulCache search jointly considering keyword-vector and average residual query-hop distances (as well as success ratio).

Table I shows IG clustering performance. Consistent with the Figures and the expectation of one cluster per IG (owing to the assumption of independent, uniform sampling of keywords per query and to the accuracy of BIC-based order selection), we see monotonic improvement in the IG name metric (it decreases) with the number of SuccessfulCache touches. The IG number accuracy (as indicated in Table I) has an asymmetric convex (U-shape) characteristic, as a function of SuccessfulCache touches, with good accuracy achieved for (relatively) large number of touches. This can be understood from the interplay between two effects: the number of IGs in a pod’s cache, which first grows very large and then decreases with the number of SuccessfulCache touches, and the equilibrated stage of the cache – in this stage, (judiciously forwarded) queries added to the SuccessfulCache will bootstrap improved accuracy of the learned IG clusters, so that the IG number accuracy increases during this stage.

VI. DISCUSSION: (LATENT) IG CHURN

Our EM-BIC approach and experiments have only considered the static case, where there is a fixed set of latent IGs. In practice, some IGs will become “stale” (unused) over time, while other mature IGs may be handed over to the

\[ \text{Table I shows IG clustering performance. Consistent with the Figures and the expectation of one cluster per IG (owing to the assumption of independent, uniform sampling of keywords per query and to the accuracy of BIC-based order selection), we see monotonic improvement in the IG name metric (it decreases) with the number of SuccessfulCache touches. The IG number accuracy (as indicated in Table I) has an asymmetric convex (U-shape) characteristic, as a function of SuccessfulCache touches, with good accuracy achieved for (relatively) large number of touches. This can be understood from the interplay between two effects: the number of IGs in a pod’s cache, which first grows very large and then decreases with the number of SuccessfulCache touches, and the equilibrated stage of the cache – in this stage, (judiciously forwarded) queries added to the SuccessfulCache will bootstrap improved accuracy of the learned IG clusters, so that the IG number accuracy increases during this stage.} \]
CCN. Moreover, nascent, heretofore unseen IGs may appear, abruptly and asynchronously. Accordingly, beyond merely online updating of clusters to account for entering and departing cache entries, there is also a need for both on-line creation as well as annihilation (removal) of mixture components/IGs. A mixture component $j$ can be removed whenever its mass ($\alpha_j$) falls below a predefined threshold, indicative of its “staleness”. Trial-inclusion of new components into the model can be evaluated with respect to the BIC criterion, with new components falls below a predefined threshold, indicative of its “staleness”. 

VII. Future Work

In future, we would like to evaluate an extension of our approach for dynamically tracking IG churn. We would also like to exploit feedback on unresolved queries, i.e., perform some type of lightweight supervised adaptation of our forwarding mechanism, to account for (and adapt to) unsuccessful query forwarding. Similarly, we can envision an adaptive reputation system overlay, used to help resolve multiple forwarding candidates.

TABLE I

<table>
<thead>
<tr>
<th>metrics</th>
<th>SC touches</th>
<th>50</th>
<th>250</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>IG Num. Accu.</td>
<td></td>
<td>0.136</td>
<td>0.105</td>
<td>0.449</td>
<td>0.607</td>
</tr>
<tr>
<td>IG Name Accu.</td>
<td></td>
<td>3.08</td>
<td>2.20</td>
<td>1.33</td>
<td>1.05</td>
</tr>
</tbody>
</table>

Fig. 1. Average num. query hops vs SuccessfulCache touches.

Fig. 2. Query success prob. vs SuccessfulCache touches.