Predicting Social Dynamics based on Network Traffic Analysis for Content Management over Information Centric Networks

Satadal Sengupta, Rohit Dhangar, Anand Kumar, Sandip Chakraborty, Bivas Mitra
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur, India - 721302
Email: {satadal.sengupta.nit, dhangar.rohit, anandsit043}@gmail.com, {sandipc, bivas}@cse.iitkgp.ernet.in

***Abstract*—Proliferation of online social networks (OSNs) in recent times has resulted in an unprecedented surge in the volume of multimedia content that is consumed by users on a daily basis. Popular OSNs such as Facebook and Twitter enable users to view and share embedded videos and images on their feeds, which attract more and more users with time, creating repeated requests for the same piece of content. The situation is particularly acute when cellular network is being used for data download. Such user access patterns have prompted the research community to focus heavily on the emerging paradigm of Information- or Content-Centric Networking (ICN/CCN). In CCN, in-network content management (like content distribution, caching etc.) is of utmost significance, and forms the crux of an enhanced user experience. In this paper, we argue that social dynamics among OSN users have a direct correlation with content-popularity, both at a geographically global as well as local level. We exhibit a strategy to identify viewing and sharing patterns of Facebook users served by a cellular base station, and thereby to infer the social dynamics among them. We propose a mechanism to estimate content popularity based on the social dynamics between users using the learned patterns over time. The strategy is validated with proof-of-concept experiments on real data as well as extensive simulations using a comprehensive simulation framework.**

I. Introduction

The cellular network has been subjected to unprecedented volumes of data traffic in recent times. The rise of online social networks (OSN), such as Facebook, Twitter and YouTube, have resulted in a never-seen-before demand for multimedia on mobile devices [1]. Facebook and Twitter, among other OSNs, allow embedded videos on the user's feed ("timeline"). These embedded videos can be streamed in-place; there is a user-adjustable 'autoplay' feature available these days which starts playing the video as soon as it comes in-focus on the application. These videos can be liked or shared by a user to make them appear on the feeds of other users socially connected to her. All these factors contribute to the growing data demand at the content servers.

The Internet at present works on the concept of *host-resolution*, i.e., content is served to users via requests to appropriate hosts. However, the user today is more concerned with the specific content that she desires to access, rather than the host serving it. A new paradigm of networking is being looked at to support such named content access, known as *Information-* or *Content-Centric Networking* (ICN/CCN) [2], [3]. Architecture of the future Internet is an important area of research in the present context, and ICN/CCN has emerged as a major piece of the puzzle. In order to keep up with the ever growing demand for content, CCNs employ the concept of *in-network management* of content storage and distribution, where intermediate routers or base stations apply content storage and distribution policies based on the underlying content usage patterns [2], [4], [5], [6], [7], [8].

It is noteworthy that even though CCNs assume content popularity as the default criterion for deciding content storage (caching) and distribution policies [6], [7], [8], the definition of *content popularity* plays an important role in determining the effectiveness of the caching and distribution mechanism. Content caching and distribution decisions in CCNs are made based on the history of accesses for a specific content, meaning that the more number of times a specific content is accessed, the more time it spends in the cache [4], [5], [7]. On the other-hand, the replication decisions for content distribution server is based on the geo-location of that server and the content popularity patterns in that location [6]. However, we argue that past history may not always be indicative of future access patterns. For example, say an instructor for a class of 10 shares a video with his students as a part of his tutorials. A naive CCN caching implementation would cache the content for a considerable duration due to the 10 accesses – however, it is highly unlikely that this content will be accessed any further in the future. A better metric would be the probability that content-popularity will increase in the near future.

In this paper, we propose a novel criterion for effective *in-network management* of content caching and distribution in CCNs. We argue that the *social dynamics* between OSN users directly affect future content-popularity, and can therefore serve as strong indicators in making effective content management decisions. A casual glance at the present content-access scenario clearly brings out a strong correlation between data demand for a particular piece of content, and the social dynamics that are associated with it. Not only does *social dynamics* affect popularity of content globally, but it also has a bearing on access patterns among individuals in geographical proximity of each other. In densely populated areas, such as those in a developing *smart* city, socially connected users are often part of the same locality, sometimes even served by the

same network gateway.

For example, let us suppose that there is a video clip going around where the Director of a premier residential institute in the country has given a rather controversial statement about the state of affairs within the institute. While the audience interested in this piece of content is small in volume, within this specific institute, the video has the potential to go viral. Let us suppose that the residents of the institute are served by a network gateway. There will be a sudden surge of requests for this video through this gateway as and when more and more people share and discuss the content. An efficient content management mechanism would automatically cache this piece of content at the gateway and other intermediate content caching locations (to avoid the gateway being the bottleneck), thus reducing the access delay for concerned users.

It immediately follows that while current networks do not have the ability to leverage on the social dynamics among users being served, it would be immensely useful to have a mechanism to do so. In this paper, we propose a strategy to infer the social dynamics between users served by a network gateway under a information centric networks, by looking at the packet traces generated at the gateway (or proxy) and identifying events (like content viewing, content sharing etc.) from their OSN usage (§ III). We address various situations that may arise while arriving at such inferences, and systematically look at ways of minimizing erroneous predictions. To the best our knowledge, this is the first work in this kind that predicts the OSN behavior by observing the packet traces at a network gateway (§ IV). As mentioned earlier, such information can be effectively use to derive a caching policy or content distribution decisions at the network gateway, based on the popularity of the content generators or content distributors (the person who "share" a content in Facebook or "retweet" it at Tweeter). While the prediction mechanism is inferred based on the analysis over a real data set from users, we also develop a comprehensive simulation framework to analyze the accuracy of our OSN behavior prediction mechanism (§ V and § VI).

II. Related Works

Research on Content Centric Networks dates back to 2010, when project COMET[1] was launched. The design principles were formulated in [9]. In the seminal paper *Networking Named Content* [2], Jacobson et. al. introduces the idea of Content-Centric Networking (CCN), which aims at decoupling content location from its identity, access, and security.

Psaras et. al. develop a mathematical model for a single router in [10], based on continuous-time Markov chains, to assess the duration for which a content is cached in the router. The model is then enhanced to account for multiple routers, based on some simple approximations. The same authors propose an in-network caching scheme called *ProbCache* in [11] for Information Centric Networks (ICNs). The strategy aims at reducing caching redundancy by approximating the capability of paths to cache contents, and then caching content probabilistically. Both homogenous and heterogeneous cache sizes are considered by the authors in this work.

Pavlou et. al., in their work called *Internet-scale content mediation in information-centric networks* [12], propose an approach where content characteristics, server load, and route distance are considered for making content-delivery decisions. The objectives are to choose the best cached copy to serve, in order to optimize network utilization, while maximizing the users' quality of experience.

In their work titled *More control over Network resources: an ISP caching perspective* [13], Tuncer et. al. propose a low cost cache management approach by which ISPs can have more control over their resources, through deployment of caching points within the network according to user demand characteristics.The service enables caching of popular content specific to an ISP, thus helping in serving client requests from within the network instead of fetching them from the origin servers. The proposed work characterized user demands into (i) Global Content Popularity – GCP and (ii) Geographic Distribution of Interests – GDI.

While all the above works talk about content popularity being the basis of intelligent caching mechanisms, there have been limited insights into how content popularity can be known in advance. The general approach has been to cache content reactively, where a content is deemed popular only after a sufficient number of accesses have already taken place. A very few works have addressed social behavior in the management of information centric networks, as summarized next.

Catanese et. al. describe two ad-hoc and privacy-compliant Facebook crawlers in [14], which are able to gather friendship data for a large number of users. The data is anonymized and represented as an undirected graph, with users as nodes and friendships among them as edges. They analyze specific properties of social-network graphs such as centrality measures, distribution of friendship, etc. and conclude that their partial datasets are able to emulate the complete Facebook graph with considerable accuracy. In our work, we use a similar approach where we represent OSN data as an undirected graph, and make use of social connections between users.

In their work titled *Learning to Discover Social Circles in Ego Networks*, McAuley et. al. presents a novel machine learning task to identify user's circles on Google+ and 'lists' on Facebook and Twitter. For detecting the circles, a model is developed that combines network structure as well as user profile information. The problem of automatically discovering user's social circles is studied and formulated as a node clustering problem on user ego network (network connections between user's friends). We use the Facebook friendship dataset contributed by the authors in our current work.

III. System Architecture

In this section, we provide an overview of the system proposed by us. Figure 1 summarizes our approach and presents

[1] http://www.comet-project.org/

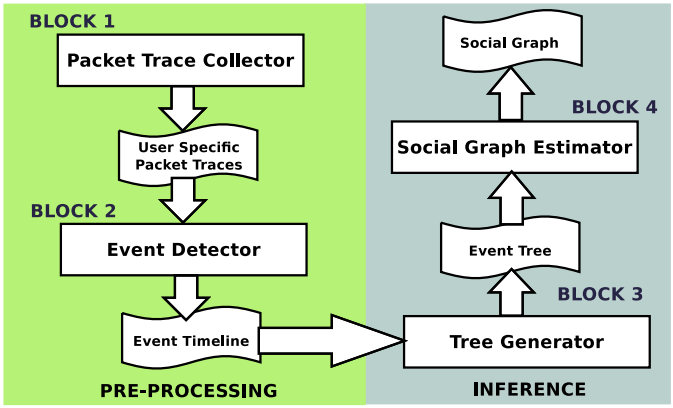


Fig. 1: System Architecture

the building blocks of the system. The system is an end-to-end aggregation of logical blocks which enable inference of useful social dynamics among users based on just their network data usage. Blocks 1 and 2 together form the *Pre-Processing* subsystem while Blocks 3 and 4 combine to create the *Inference* subsystem.

The first block in the system, labeled *Block 1*, is the *Packet Trace Collector*. This module runs on the network gateway as a background service, and logs packet traces for each user. We assume that it is possible for the operator at the network gateway to attribute each packet to a unique user, since it has information about the traffic generated by each devices (based on the device's source IP address and source MAC address).

The second block, labeled as *Block 2* in the figure, is the *Event Detector*. This module first filters packets generated from *facebook.com*[2] and then uses *deep packet inspection* [15] to read packet payloads. Based on unique signatures for the ***share*** and ***view*** Facebook events, the events are detected and a time-stamp is attached to each such event. These $(user\_id, content\_id, event, timestamp)$ quadruplets form the *events timeline*, which is the output of *Block 2*.

Block 3 is the *Tree Generator* – this module takes into account the sequence of events as obtained in the *events timeline* and generates an *event tree* for each unique piece of content. An *event forest* is thus generated and is forwarded for further processing to the next logical block.

The last and final logical block is *Block 4*, the *Social Graph Estimator*. This module implements an event-tree aggregation strategy which produces a directed weighted graph as the intermediate output. The edge weights represent confidence values for edges to belong in the final inferred graph – based on a threshold on this confidence value, some edges are dropped as coincidental edges, while the rest are retained. This filtering step produces the final directed, weighted inferred social graph. This inferred graph, we argue, embeds features which aids the operator in designing a superior caching mechanism.

The system outlined in this section is described in-detail in the following section.

[2]We consider Facebook as an example test case to infer social dynamics from OSN events. Any OSN service can be used here as long as similar events are detectable from packet traces, using the similar procedure that we use here.

IV. System Design

This section describes the system in detail. We uncover each block shown in Figure 1 and explain how it is implemented.

However, before we delve into the specifics, it is important to point out that the *event detector* block requires a discrete set of packets to work with. While *packet trace collector* can keep working continuously, it will be required to pause it periodically, accumulate the traces collected over the last period, feed it to the successive block for processing, purge old traces, and then resume trace collection. An important factor here is the length of each period, or the *time window* after which the social dynamics are recomputed. We leave this decision to the service provider, to be taken based on factors such as traffic generation rate at the gateway and desired freshness in results.

In the following sub-sections, we peek inside each building block of our proposed system:

A. Packet Trace Collector

The packet trace collector forms the first block of the system. It runs as a background service at the network gateways and logs packet trace for each user. The trace can be collected using standard trace collection applications like `tcpdump`. These packet traces are then fed as input to our next block, i.e., the *event detector* block.

B. Event Detector

An event may consist of either of the 2 activities: (i) viewing a piece of content by a user, (ii) sharing a piece of content by a user. "Piece of content" in this context may refer to any content which is large enough for caching to be worth-while. Typically, these contents are videos; however, large images may also be worth-caching.

The *event detector* performs the following tasks:

Input: The *event detector* takes a set of packet traces as input from the *packet trace collector*.

STEP 1: *Packet Filter* – Information present in the packet headers is used to filter out packets originating from Facebook.

STEP 2(a): *Viewing Event Detection* – The event that a user is viewing a video, can be inferred by monitoring the traffic pattern generated by the user. Non-video traffic usually follows a monotonous, low bit-rate pattern; however video streaming traffic exhibits a distinct bursty pattern with troughs in between bursts. Figure 2 shows plots for the number of bytes generated per unit time – the traffic pattern generated while playing a video is shown in figure 2a, while figures 2b and 2c show patterns generated while browsing the web and loading an image, respectively. The bursty pattern for video is clearly distinguishable from the pattern generated during other user actions. This pattern can be used to detect the time during which a video is being played.

However, merely deducing the fact that a user is streaming a video is not sufficient – we must be able to attribute each

video to a unique identifier. We observed that each time a user starts streaming a video, a domain name system (DNS) query is sent to get the URL of the particular video. Depending on the source of the shared video, e.g., YouTube, DailyMotion, or Facebook (embedded) video, the video title may or may not appear encrypted in the DNS packet. Numerous experiments performed by us show that even if it is encrypted, the generated URL is unique, and all future accesses of this video will appear in the traces with the same URL. For example, the following URL was recorded in the HTTP GET request when a certain Youtube video was shared on Facebook (screenshot from Wireshark is shown in figure 3): `https%3A%2F%2Fvideo.fbom1-1.fna.fbcdn.net %2Fhvideo-xtp1%2Fv%2Ft42.1790-2%2F11158692_367216400134622_1667525658_n.mp4`. The unique part of this URL, which represents the video ID, is the string after **%2Fv%2F** – in the above example, the string is **t42.1790-2%2F11158692_367216400134622_1667525658_n.mp4**.

We use these observations to uniquely identify the content being streamed – note that it is not important to know the title or contents of the video being streamed for our purposes, as long as we can successfully detect all access instances of the same video.

The quadruplet $(user\_id, content\_id, VIEW, timestamp)$ is generated and inserted as an entry into the *events timeline* every time a view event is detected.

STEP 2(b): *Sharing Event Detection* – We observe that Facebook makes a call to `graph.facebook.com` when the share button is hit. In Figure 4, the line highlighted in orange shows a case. Our hypothesis is that a user will share a content within seconds of viewing it (completely or partially) if she likes it; she will, in most cases, not come back to share it after a long duration. We set the time window to 20 seconds in our experiments – if a user streams a video, we assume that she will either share it within 20 seconds after streaming has ended, or never share it[3]. Thus, we use traffic pattern detection in conjunction with packet header information to decide which content has been shared.

The quadruplet $(user\_id, content\_id, SHARE, timestamp)$ is generated and inserted as an entry into the *events timeline* every time a share event is detected.

Output: An *events timeline* is generated as output, which is a consolidation of all the quadruplets generated from the current batch of packet traces. This timeline is now fed as input to the next block.

C. Event Tree Generator

Once the *event timeline* is obtained, the next task is to generate *event trees* for specific contents based on their order of occurrence. This is taken care of by the third block in the system, which is the *event tree generator* module. Every user is represented as a node of the tree and the parent-child relationship is inferred from the nature of the detected events and their order of occurrences. The tree generation process follows the following conditions:

1) The user featuring in the chronologically first SHARE quadruplet entry in the timeline for a $content\_id$, becomes the root of the tree.
2) Any node viewing the content becomes the child to the node which shared the content. If multiple users have shared the content before this view, then it becomes tricky to decide which user's *SHARE* is the significant event. We consider different scenarios during our evaluation to address this issue, and provide an indication of what could be the best strategy to decide the significant *SHARE*.
3) No node can be a parent (including root) unless it shares the content.
4) If a user views a specific content more than once, only the first view in the timeline is considered for generating the tree; a user cannot share the same content twice trivially (not allowed by Facebook).

TODO: Add pictorial description

An Example Scenario: Consider that in the current timeline, A shares a video, and B & C watch the video. So, A becomes the root, and B & C become the children. Further, say C shares the video and after that D views the video. Irrespective of whether D views the one shared by A or C, D is added as a child of C. There can be a special case, where a node E shares a video but doesn't view it before the sharing. Then E becomes a separate root and we get a forest.

This process is executed for all the contents being shared and finally the *event forest* is obtained. This event forest is then used by the next block to estimate the probable social graph.

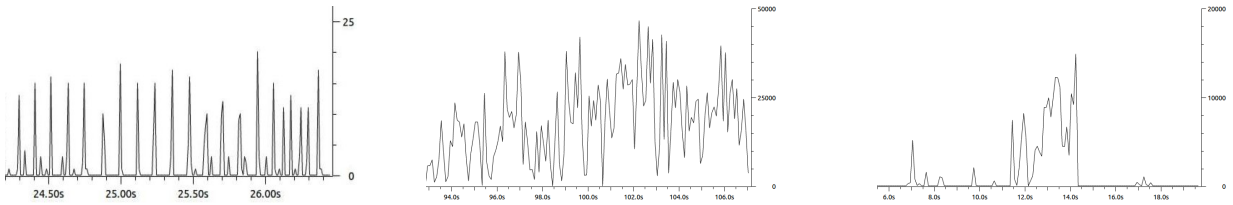
D. Social Graph Estimator

The social graph estimator makes use of the generated event trees to estimate the social graph. The graph generation algorithm works as follows: We start with an empty graph $G_{inferred}$. First, we add all nodes to the graph which have appeared at least once in any of the event trees. Next, for every event tree, we iterate over its edges, and the parent-child relationship between the two nodes is carried forward to $G_{inferred}$, in either of the following ways: (i) If the directed edge doesn't exist, add the direct edge with edge weight 1, (ii) If the directed edge already exists, increment the edge weight by 1. Finally we get a directed weighted graph, where the weight of an edge $W_{v,u}$ shows for how many contents, the node pair had a parent-child relationship. The edge confidence $Conf_{v,u}$ for a particular edge (v, u) is computed as follows:

$$Conf_{v,u} = \frac{W_{v,u}}{\sum_i W_{v,i}} \quad (1)$$

where (1) holds for all i for which edge (v, i) exists.

[3]We may miss some exceptional sharing cases using this hypothesis; however, our interactions with social media users indicate that most sharing events follow this trend.



(a) Traffic Pattern for Video Streaming (b) Traffic Pattern for Browsing Event (c) Traffic Pattern for Image Load

Fig. 2: Network Data Traffic Pattern for Different User Actions

```
Frame 11: 495 bytes on wire (3960 bits), 495 bytes captured (3960 bits)
Linux cooked capture
Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
Transmission Control Protocol, Src Port: 50273 (50273), Dst Port: 32947 (32947), Seq: 1, Ack: 1, Len: 427
Hypertext Transfer Protocol
▸ [truncated]GET /cache-thru?remote-uri=https%3A%2F%2Fvideo.fbom1-1.fna.fbcdn.net%2Fhvideo-xtp1%2Fv%2Ft42.1790-2%2F11158692_367216400134622_1667525658_n.mp4%
  User-Agent: stagefright/1.2 (Linux;Android 5.1)\r\n
  Host: 127.0.0.1:32947\r\n
```

Fig. 3: Example URL for Video File

```
18 0.468951 31.13.93.3     192.168.1.133 TCP      68 443→44329 [ACK] Seq=1 Ack=1487 Win=101 Len=0 TSval=
19 0.476522 192.168.1.133 192.168.1.1   DNS      80 Standard query 0x1760  A graph.facebook.com
20 0.477951 192.168.1.1   192.168.1.133 DNS     162 Standard query response 0x1760  CNAME api.facebook.
21 0.479091 192.168.1.133 31.13.93.3    TCP      76 59317→443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK
22 0.502654 31.13.93.3    192.168.1.133 TLSv1.  951 Application Data
23 0.502744 192.168.1.133 31.13.93.3    TCP      68 44329→443 [ACK] Seq=1487 Ack=884 Win=396 Len=0 TSva
28 0.532315 192.168.1.133 31.13.93.3    TCP      76 36978→443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK
31 0.732604 31.13.93.3    192.168.1.133 TCP      76 443→59317 [SYN, ACK] Seq=0 Ack=1 Win=13980 Len=0 MS
```

Fig. 4: Sharing Event: Call to `graph.facebook.com`

Based on a *confidence threshold*, which is a tunable parameter set on the edge confidence, edges are either retained or dropped. The threshold is used to distinguish between edges which are estimated to be present in the original social graph and edges which have been formed coincidentally due to assumptions made while generating the event trees. The directed weighted graph with retained edges is the final estimated social graph $G_{inferred}$.

Algorithm 1 shows the procedure followed for inferring the social graph. The list of all *event trees* and a threshold on the edge confidence $W_{confidence}$ are fed as inputs to the algorithm. An intermediate output of the algorithm is the aggregated graph $G_{intermediate}$, and the final output is the estimated social graph $G_{inferred}$.

The algorithm takes each event tree $CurrentEventTree$ and loops over all of its edges. If an edge $DirectedEdge$ is already present in $G_{intermediate}$, then the edge weight is incremented by 1; otherwise, the edge is added to $G_{intermediate}$ with edge weight 1. Once all the edges of $CurrentEventTree$ have been worked upon, it is deleted from the list of all *event trees*, i.e. $EventForest$, and the process is continued till the list is exhausted. $G_{intermediate}$ is now the intermediate estimated graph.

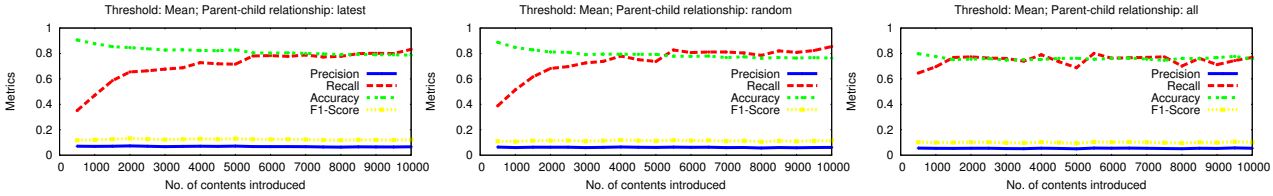
In the final step, $G_{inferred}$ is formed from $G_{intermediate}$ by retaining only those edge whose confidence $Conf_{DirectedEdge}$ is greater than the confidence threshold $Conf_{threshold}$. The final output $G_{inferred}$ is a directed, weighted graph, with each edge representing an inferred social relationship between the end-points.

V. Simulation Framework

In the previous section, we present a detailed discussion about the various modules which our system comprises of. As one might have predicted, performing a large-scale experiment to validate the effectiveness of social dynamics-driven content-popularity estimation would require huge volumes of user packet traces from the gateways. Due to various reasons, including but not limited to privacy concerns, it is extremely difficult to obtain such data. In absence of large scale data, a logical course-of-action is to use simulation to evaluate our hypothesis.

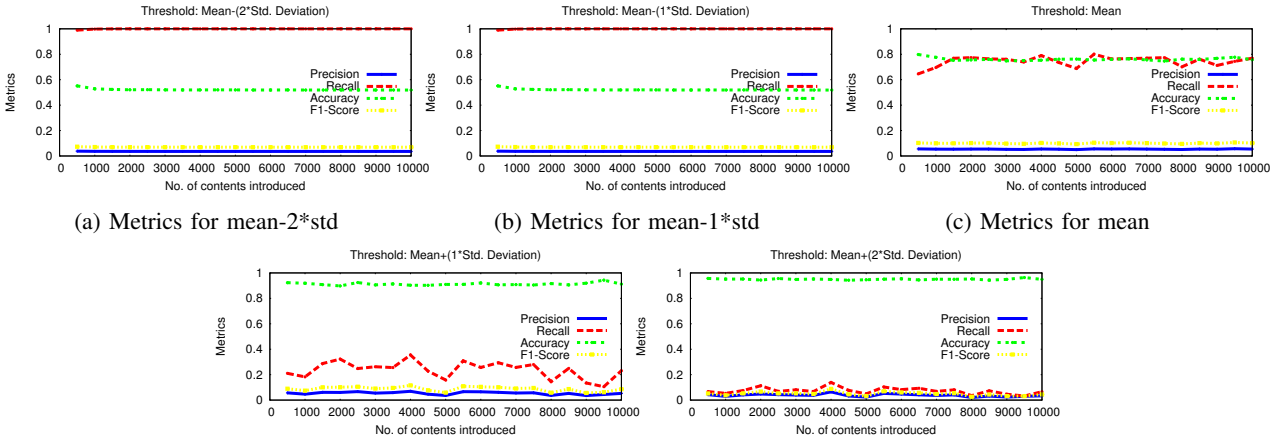
In this section, we present a comprehensive simulation framework to evaluate our proposed social dynamics inference strategy. It is important to point out that the simulation framework replaces the "pre-processing" sub-system shown in Figure 1, which generates the input to the "inference" sub-system. The "social graph estimator" module works on the output generated by the simulation framework to produce the inferred social graph. As a means of evaluating the effectiveness of our inference algorithm, we compare relevant edges (described later) in the original social graph input to our system ($G_{friendship}$), and the social graph inferred by our strategy ($G_{inferred}$). The simulation framework works as follows:

STEP 1: ***Real-world social network*** **–** We start with an existing social (Facebook) graph obtained from SNAP [16] consisting of 4039 nodes and 88,234 edges. Each node represents a Facebook user and an edge between two nodes indicates that the two users are friends on Facebook. This



(a) Metrics for 'latest' (b) Metrics for 'random' (c) Metrics for 'all'

Fig. 5: Classification metrics for variation in content count – parent-child relationship scheme



(a) Metrics for mean-2*std (b) Metrics for mean-1*std (c) Metrics for mean

(d) Metrics for mean+1*std (e) Metrics for mean+2*std

Fig. 6: Classification metrics for variation in content count – threshold

undirected graph forms our $G_{friendship}$.

STEP 2: *Model influence propagation* – Each node v in $G_{friendship}$ is now assigned 2 vectors of probability values – P^v_{view} and P^v_{share}. $P^v_{view}[u]$ denotes the probability that node v **views** content when node u shares it, while $P^v_{share}[u]$ denotes the probability that node v **shares** content when node u shares it. For node v, $P^v_{view}[u]$ and $P^v_{share}[u]$ are defined only for all nodes u which are neighbours of v in $G_{friendship}$.

Studies [17] have shown that influence propagation in social networks often follow a Bernoulli distribution with success value p^u_v, which is the ratio of the number of times node v successfully influences node u to the number of attempts by node v. Following this, for each node v, we generate random values from a binomial distribution[4] with no. of trials n and success probability p, and use those to populate the vectors P^v_{view} and P^v_{share}. More precisely, in the content-view case, if for node v, the random value generated for another node u is r, then $P^v_{view}[u] = r/n$. The pair (n, p) is fed as input parameters to the simulation framework.

STEP 3: *Model content popularity* – In reality, different pieces of content may have different levels of popularity. For example, 'Gangnam Style' is more popular than the pilot episode of a regional daily soap. For purposes of simulation, we assume that a given piece of content can belong to one of the three levels of popularity – (i) highly popular (level 3), (ii) popular (level 2), (iii) not-so-popular (level 1). Depending on the popularity level of the current content being circulated, nodes will exhibit increased or decreased likelihood of watching or sharing the content – this behavior is captured by providing a proportional boost to the probability values in P^v_{view} and P^v_{share}.

STEP 4: *Model content introduction* – In a real scenario, a piece of content may be introduced to a given social graph at multiple nodes simultaneously, i.e., many users may independently watch a video and then share it with their respective friends. We call these nodes as the *points of introduction* for a specific piece of content. This phenomenon is well-studied by Sun et. al. in their work called 'Gesundheit' [18], where they model contagion across fans (users who "like") of Facebook pages. They find that across pages with more than 1000 users (sufficiently large user-base), the mean number of points of introduction comes out to 14.8% of the total number of users, with a standard deviation of 7.9%. We use their findings in our setting for 2 reasons: (i) We are dealing with the same social network, i.e., Facebook, which ensures that the social network dynamics remains same in our case, and (ii) Users sharing a page can be interpreted as users sharing 'content' (a Facebook page being the content here), which perfectly describes the phenomenon we are trying to model.

Once we choose a number n from a uniform distribution with mean and standard deviation as mentioned above, we

[4]Influence propagation in social networks can be modeled in a variety of ways, including time-varying models. We choose the static binomial distribution model for simplicity.

Algorithm 1: Social Graph Estimation

Input: List of all *Event Trees*, Threshold edge confidence $Conf_{threshold}$
Output: Estimated Social Graph $G_{inferred}$
$EventForest$:= List of all *Event Trees*;
$G_{intermediate} := NULL$;
while $EventForest$ *is not empty* **do**
  $CurrentEventTree$:= The first *Event Tree* in $EventForest$;
  foreach $DirectedEdge \in CurrentEventTree$ **do**
    if $DirectedEdge \in G_{intermediate}$ **then**
      Increment weight of corresponding directed edge in $G_{intermediate}$ by 1;
    end
    Add corresponding directed edge to $G_{intermediate}$ with weight 1;
  end
  Delete $CurrentEventTree$ from $EventForest$;
end
$G_{inferred} := NULL$;
foreach $DirectedEdge \in G_{intermediate}$ **do**
  $Sum_W = 0$;
  foreach $OutgoingEdge$ *from node on which* $DirectedEdge$ *is incident* **do**
    $Sum_W := Sum_W + W_{OutgoingEdge}$
  end
  $Conf_{DirectedEdge} := W_{DirectedEdge}/Sum_W$;
  if $Conf_{DirectedEdge} > Conf_{threshold}$ **then**
    Add $DirectedEdge$ to $G_{inferred}$;
  end
end

randomly choose n nodes in the graph and assign those as the *points of introduction*.

STEP 5: ***Model content propagation*** **–** Each *point of introduction* is considered to be the root of a new content-propagation tree. Nodes in one-hop neighborhood of the root perform one of the following actions depending on their probability values for the root node: (i) view followed by share, (ii) only view, (iii) neither view nor share. Activity (i) results in the generation of an edge between the root and this one-hop neighbor, and also in the extension of the tree so that the neighbor is now the root for the child subtree. Activity (ii) renders the next-hop neighbor a leaf node. Activity (iii) does not prompt any tree-generation based operation. All view and share event-quadruplets, as described in § IV, are logged, which forms the *events timeline*.

STEP 6: ***Model gateway cluster*** **–** Remember that we design our scheme keeping in mind a typical network gateway. It is highly unlikely that a gateway will have knowledge of the entire set of nodes in the friendship graph $G_{friendship}$. In fact, the gateway will serve only a small, geographically close subset of these nodes at best. Let us call the sub-graph of $G_{friendship}$, which consists of nodes that the gateway has knowledge about, and the corresponding edges, $G_{cluster}$.

As we have argued in § I, the population served by a gateway is likely to be a well-knit community, say, the students in a campus. In order to model this population sampling, we use *Louvain community detection* [19], [20] to identify well-knit communities. We perform our graph estimation for the following clusters individually: (i) cluster with maximum size, (ii) cluster with maximum edge density. For each of these cases, the *events timeline* is filtered to retain only events corresponding to nodes in the current $G_{cluster}$. The filtered *event timeline* serves as input to the *social graph estimator*. This concludes the simulation framework.

Social Graph Estimator: Given an *events timeline* as input, the *social graph estimator* does the following:

1) Form trees for each content introduction. Note that one unique content can result in the formation of multiple trees if it has been introduced in the $G_{cluster}$ by multiple nodes.
2) Employ the algorithm for *Social Graph Generator* described in § IV.

In the following section, we evaluate the effectiveness of the *Social Graph Estimator* in re-generating $G_{cluster}$ from the *events timeline* supplied as input to it.

VI. Evaluation

In order to evaluate the effectiveness of our strategy in inferring social dynamics among users, we study the effect of changing the following variables in our simulation framework:

1) Number of contents introduced into the network during one run of the simulation, i.e., *content count*.
2) The (mean, standard deviation) pair of the *view-share probability distribution*. Every user is assigned a view probability and a share probability at random from this distribution, for every other user, as explained in **STEP 2** of § V. These probability values decide how active a user is likely to be, in response to another user's actions.
3) The content-level probability boost for view probability and share probability. Every probability value, as assigned in 2 above, is given a boost to model content popularity, as explained in **STEP 3** of § V.
4) The confidence threshold value, $Conf_{threshold}$.

The simulation cases are as listed in table I. The first column represents the simulation parameter being varied, as described in the list above. The second column gives the range of values for the corresponding parameter in the first column. Subsequently, the third and fourth columns represent the step-increase in the value of a parameter and the default value of the parameter, respectively. In order to study the effect of varying a parameter on the inference accuracy, we vary only one parameter at a time. All the other parameters are maintained at their default values. Only the parameter being varied is changed in the coresponding range of values, with a step-increase as given in the *Step* column.

While implementation of the other 3 variables is fairly self-explanatory, the *content-level probability boost* is implemented as follows. One may recollect from **STEP 3** of § V that

TABLE I: Simulation Parameters

Variable Parameter	Range of Values	Step	Default Value
Content Count	500 – 10,000	1,000	1,000
View Share Probability – Mean	0.3 – 07	0.1	0.5
View Share Probability – Std. Deviation	0.1 – 0.2	0.05	0.15
Content Level Boost – View	0.0 – 0.8	0.1	0.3
Content Level Boost – Share	0.0 – 0.8	0.1	0.3
Confidence Threshold	[Mean + (n*Std. Deviation)], n = {-2, 2}	1 * Std. Deviation	N/A

there can be 3 levels of popularity for a partiular piece of content: (i) highly popular (level 3), (ii) popular (level 2), (iii) not-so-popular (level 1). Let the value chosen in a particular simulation, as given in table I, be called the *Min-level boost*. The actual boost for levels 1, 2 and 3 are then calculated according to the following equation:

$$Prob.\,boost\ for\ level\ k\ =\ [1{+}0.2{*}(k{-}1)]\ *\ Min{-}level\ boost \quad (2)$$

As is clear from the above equation, the *Min-level boost* is actually the boost for level 1 (since k=1), and the boost for levels 2 and 3 increase by 20% from the boost for the next lower level. Now, the view and share probability values for every defined $P^v_{view}[u]$ and $P^v_{share}[u]$ are re-computed as follows:

$$P^v_{view}[u] = (1.0 + Prob.\ boost\ for\ level\ k) * P^v_{view}[u] \quad (3)$$

$$P^v_{share}[u] = (1.0 + Prob.\ boost\ for\ level\ k) * P^v_{share}[u] \quad (4)$$

where k is the popularity level of the current content being propagated.

One important point to note is that the inference algorithm also depends on the way a parent-child relationship is decided when there are multiple shares before a particular view. This phenomenon is described in sub-section 4.3, under point no. 2 in the tree generation process. During our simulations, we make this decision in three different ways, as described below:

1) The latest *SHARE* is considered as the significant event. We call this the *Latest Share* case.
2) A *SHARE* is chosen at random from the list of shares before the *VIEW*, and considered as the significant event. We call this the *Random Share* case.
3) No distinction is made between all the candidate *SHARE* events – all of those are considered and an edge is formed corresponding to each of them. We call this the *All Share* case.

In the third case though, the event cascade no longer remains a tree, since there are edges from multiple parents (due to candidate share events) to a single child (due to the view event). However, since our algorithm eventually congregates all the event trees into a graph, this difference in visualization of the cascade becomes immaterial [5].

[5]The confusion created by calling the cascade an *event tree* can be avoided simply by choosing to call it an *event graph*, without changing the working of our algorithm at all.

We perform one separate round of simulations for each of the aforementioned parent-child relationship deciding strategies. The simulation results and corresponding analysis are given in the following sub-sections.

A. Impact of variation in Content Count

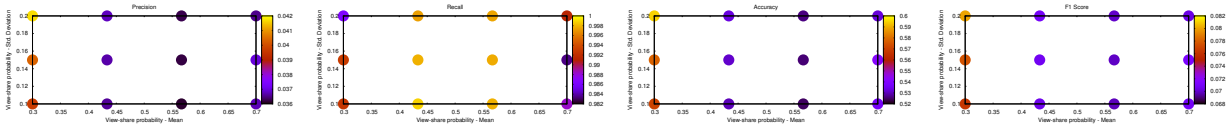
The impact on inference accuracy due to the variation in *Content Count* is shown in figure 5. We make the following observations:

1) For parent-child-relationship scheme variation, as shown in figure 5, we find that the recall rate remains fairly high and shows an increasing trend, as content count is increased. The effect is more pronounced in case of 'latest' and 'random' schemes, whereas it is mild in case of 'all'. The accuracy too remains high, and while it shows a slight decrease, the changes are marginal. Precision remains low, however, and as a consequence, F1-score remains low as well. Increasing recall and sustained high accuracy can be attributed to the fact that more number of contents allow for more difference in confidence values to build, between genuine edge and coincidental edge case, since more and more edges are added over time.
2) In case of threshold variation, as shown in figure 6, we observe that recall starts at 1.0 for low thresholds, and shows a steady decrease as threshold is increased. Accuracy shows the opposite effect – it steadily increases as the threshold is increased, and almost reaches 1.0 when threshold is at mean plus twice std. deviation. Decreasing recall is an effect of higher thresholds allowing lesser and lesser edges to be considered as genuine, therefore causing our algorithm to miss out on many genuine edges. Higher accuracy is due to more stringent checking, which allows far lesser number of false positives to occur. Precision and F1-score remain low throughout. Also, all 4 metrics remain fairly steady across change in content count.

B. Impact of variation in Mean and Standard Deviation of the View-Share Probability Distribution

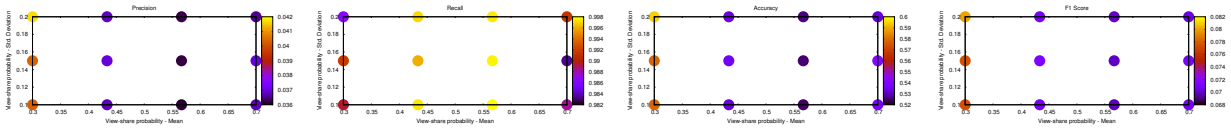
The impact on inference accuracy due to variation in the mean and standard deviation of the view-share probability distribution is shown in figure 7. The following observations can be made:

1) The results show that low values of mean and standard deviation perform better for all metrics, in general.



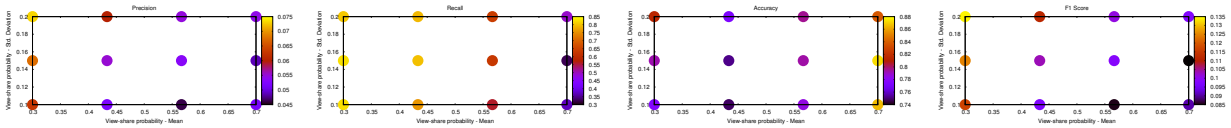
(a) Precision (b) Recall (c) Accuracy (d) F1 Score

(e) Classification Metrics for threshold = mean - (2*std. deviation)



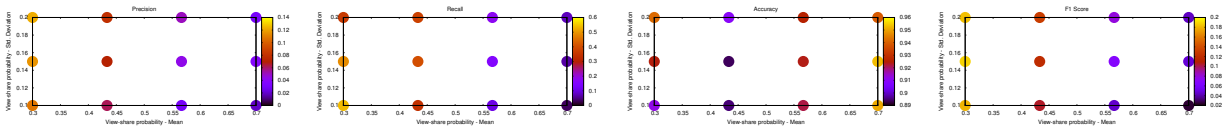
(f) Precision (g) Recall (h) Accuracy (i) F1 Score

(j) Classification Metrics for threshold = mean - (1*std. deviation)



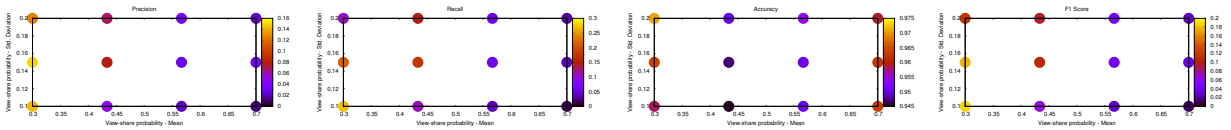
(k) Precision (l) Recall (m) Accuracy (n) F1 Score

(o) Classification Metrics for threshold = mean



(p) Precision (q) Recall (r) Accuracy (s) F1 Score

(t) Classification Metrics for threshold = mean + (1*std. deviation)



(u) Precision (v) Recall (w) Accuracy (x) F1 Score

(y) Classification Metrics for threshold = mean + (2*std. deviation)

Fig. 7: Impact of view-share probabilities' mean-std. deviation variation

Recall rate shows shift from being maximum at medium values of mean, to being maximum at lower values of mean, as threshold is increased.

2) Accuracy shows a change from being maximum at low values of mean, to being maximum at higher values of mean.
3) Precision and F1 score remain low throughout, as observed earlier. The values are slightly higher at low values of mean and standard deviation.

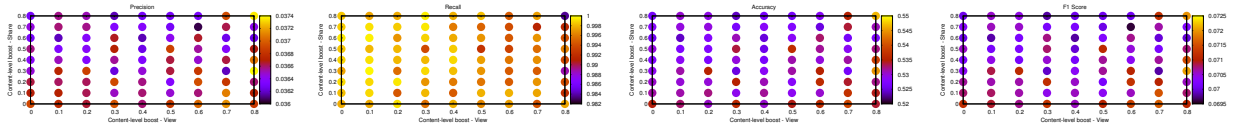
C. Impact of variation in Content-level Boost for View and Share Probability

Performance changes due to variation in the content-level boost of view and share probabilities, are indicated in figure **??**. We observe the following from the figures:

1) We observe fairly mixed results and no definite trend, except in case of recall.
2) Recall moves from being very high throughout to lower throughout as threshold is increased.

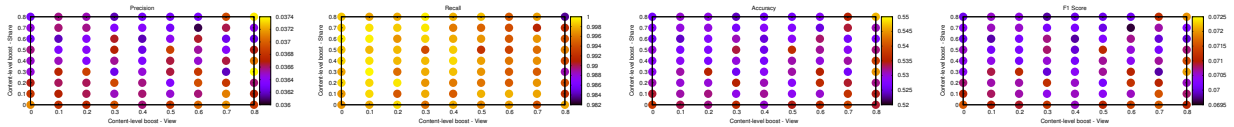
VII. Conclusion

In this paper we have presented a methodology to cache most frequently used social networks contents on the base station itself. Using Packet traces it is possible to identify various user activities and after identifying these activities by running code on base station we can make base station learn social dynamics among various users. By tracking various user activities we infer social graph among the users. Although our small scale results are quite encouraging and suggest the feasibility of given methodology, In order to show some results we need to perform this experiment on large scale using different devices.



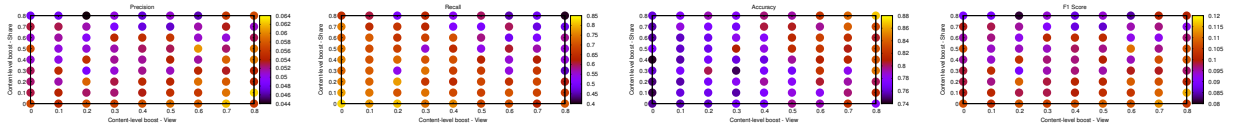
(a) Precision (b) Recall (c) Accuracy (d) F1 Score

(e) Classification Metrics for threshold = mean - (2*std. deviation)



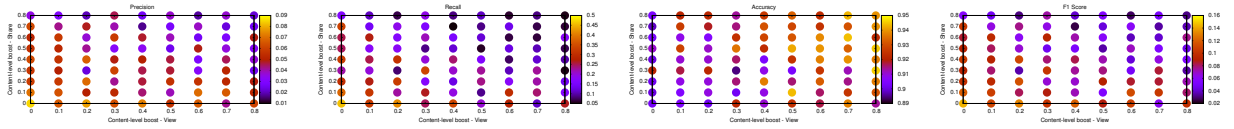
(f) Precision (g) Recall (h) Accuracy (i) F1 Score

(j) Classification Metrics for threshold = mean - (1*std. deviation)



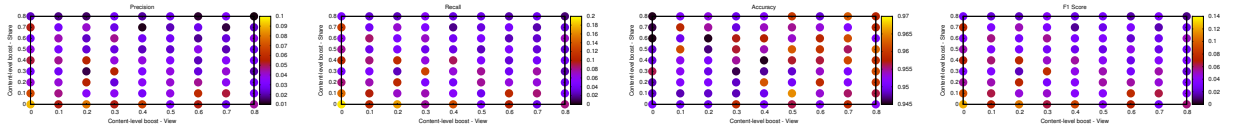
(k) Precision (l) Recall (m) Accuracy (n) F1 Score

(o) Classification Metrics for threshold = mean



(p) Precision (q) Recall (r) Accuracy (s) F1 Score

(t) Classification Metrics for threshold = mean + (1*std. deviation)



(u) Precision (v) Recall (w) Accuracy (x) F1 Score

(y) Classification Metrics for threshold = mean + (2*std. deviation)

Fig. 8: Impact of content-level boost for view-share probabilities

REFERENCES

[1] Deloitte, "Digital media: Rise of on-demand content," [Accessed 06-May-2016]. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/in/Documents/technology-media-telecommunications/in-tmt-rise-of-on-demand-content.pdf

[2] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[3] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, 2012.

[4] J. Choi, J. Han, E. Cho, T. Kwon, and Y. Choi, "A survey on content-oriented networking for efficient content delivery," *IEEE Communications Magazine*, vol. 49, no. 3, pp. 121–127, 2011.

[5] D. Perino and M. Varvello, "A reality check for content centric networking," in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*. ACM, 2011, pp. 44–49.

[6] Z. Su and Q. Xu, "Content distribution over content centric mobile social networks in 5G," *IEEE Communications Magazine*, vol. 53, no. 6, pp. 66–72, 2015.

[7] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. Leung, "Cache in the air: exploiting content caching and delivery techniques for 5G systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 131–139, 2014.

[8] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[9] G. García, A. Bben, F. J. Ramón, A. Maeso, I. Psaras, G. Pavlou, N. Wang, S. Spirou, S. Soursos, E. Hadjioannou *et al.*, "Comet: Content mediator architecture for content-aware networks," in *Future Network & Mobile Summit (FutureNetw), 2011*. IEEE, 2011, pp. 1–8.

[10] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, and G. Pavlou, "Modelling and evaluation of ccn-caching trees," in *NETWORKING 2011*. Springer, 2011, pp. 78–91.

[11] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *Proceedings of the second edition of the ICN workshop on Information-centric networking*. ACM, 2012, pp. 55–60.

[12] G. Pavlou, N. Wang, W. K. Chai, and I. Psaras, "Internet-

scale content mediation in information-centric networks," *annals of telecommunications-annales des télécommunications*, vol. 68, no. 3-4, pp. 167–177, 2013.

[13] D. Tuncer, M. Charalambides, R. Landa, and G. Pavlou, "More control over network resources: An isp caching perspective," in *Network and Service Management (CNSM), 2013 9th International Conference on*. IEEE, 2013, pp. 26–33.

[14] S. A. Catanese, P. De Meo, E. Ferrara, G. Fiumara, and A. Provetti, "Crawling facebook for social network analysis purposes," in *Proceedings of the international conference on web intelligence, mining and semantics*. ACM, 2011, p. 52.

[15] Wikipedia, "Deep packet inspection," [Accessed 06-May-2016]. [Online]. Available: https://en.wikipedia.org/wiki/Deep_packet_inspection

[16] J. J. McAuley and J. Leskovec, "Learning to discover social circles in ego networks," in *NIPS*, vol. 2012, 2012, pp. 548–56.

[17] A. Goyal, F. Bonchi, and L. V. Lakshmanan, "Learning influence probabilities in social networks," in *Proceedings of the third ACM international conference on Web search and data mining*. ACM, 2010, pp. 241–250.

[18] E. Sun, I. Rosenn, C. Marlow, and T. M. Lento, "Gesundheit! modeling contagion through facebook news feed." in *ICWSM*, 2009.

[19] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[20] T. Aynaud, "Community detection for networkxs documentation," [Accessed 06-May-2016]. [Online]. Available: http://perso.crans.org/aynaud/communities/