Stir: Spontaneous Social Peer-to-Peer Streaming

Anh Tuan Nguyen †, Baochun Li ††, Michael Welzl †, and Frank Eliassen †
† Department of Informatics, University of Oslo, Norway
†† Department of Electrical and Computer Engineering, University of Toronto

Abstract—Dealing with high churn rate is very challenging in live peer-to-peer streaming. State-of-the-art studies try to mitigate the problem by exploiting peer dynamic models, analyzing traces from real world systems, or using enhanced coding techniques, e.g., network coding. However, the origin of the problem — the interest of users on the streaming session — has never been investigated as a solution to the challenge. Applications of social networking in peer-to-peer systems, especially on file sharing, have received recent research attention. We believe that a tight integration between high level social networking of users and low level overlay of peers would bring significant benefits in dealing with peer churn rates and providing personalized sharing, e.g., instant messaging, and Twitter-like commenting. Such social network formation maximizes the potential of collaboration between peers in a session, and provides reliable indication to deal with high churn rates. Our simulations with real social data and peer dynamic traces have demonstrated the benefits of Stir and shed light on building such a system in practice.

I. INTRODUCTION

The flourishing of peer-to-peer (P2P) streaming has been witnessed recently with the popularity of current P2P streaming systems, e.g., CoolStreaming [1] and PPLive [2]. It is the P2P paradigm that enables the current best effort Internet infrastructure to provide streaming services to thousands of users with low cost servers. However, one of the most challenging problems in such systems is high peer churn rates, i.e., peers stay in a session for a short period of time. When a peer leaves, others who are receiving packets from it are negatively affected.

There has been substantial amount of research on dealing with peer churns. State-of-the-art studies have modeled P2P systems with churn to better understand it [3], analyzed traces of real world systems to understand join and leave patterns [4], or used specialized coding techniques, e.g., network coding [5]. Although they have shown improvements in system performance, the impact of peer churn can not be minimized as its origin has not been considered. The arrivals and departures of users depend on their interests on the session, e.g., users leave quickly because they do not like the content. We believe that knowledge about user interests is critical when dealing with peer churn.

Recently, the popularization of new social network sites that allow individuals to construct personal profiles, connect to people, and keep up with friends has shown that users are indeed interested in sharing their common interests on networked applications. For example, at the time of this writing, Facebook [6] is the second most-trafficked website in the world [7]. It currently has 400 million active users who spend 500 billion minutes per month to interact with over 160 million objects (pages, groups, and events) [8].

Applications of social networking to P2P networking have emerged with the objective of improving P2P system performance. The idea is that connections among friends are more reliable than those among strangers. Some studies have shown advantages of social-based P2P systems in file sharing [9]–[11]. However, in those systems, the establishment of connections among peers is based on social relationships among users, which are not formed in the context of a peer-to-peer session but, e.g., imported from other social networks. Because friends in such a separate social network do not always have similar interests, they may not necessarily join the same peer-to-peer session. Consequently, a user may not acquire enough qualified connections and suffers degraded quality as a result.

As an example, in Flickr [12], each user has a friend (contact) list. Some user can create a group on a particular topic, and others can join the group to share their interests on photos of group members. We have evaluated friendship among members of Flickr groups: How many friends of a member also join the group? or Are members of a group also friends of each other? For 20 groups with different sizes and topics we have considered, the answer is that in most of the groups, members do not have any, or very few, friends in their group. Figure 1 demonstrates the observation with the friendship network of members of “Macbook Pro” group.

![Fig. 1. Friendships among members of the “Macbook Pro” group.](image360x139to515x259)

We believe that a very tight integration between high level social networking of users and low level overlay of peers would bring significant benefits in dealing with peer churn to improve the system performance. In this paper, we present
Stir, a new spontaneous social P2P streaming system. The distinct feature of Stir is that friendship is spontaneously formed during a streaming session by means of spontaneous communication, e.g., instant messaging (IM) and Twitter-like commenting. Such spontaneous social networks are then exploited directly by the underlying streaming protocol. On one hand, the streaming protocol can get benefits from the social network by being able to establish durable connections among peers, which are useful in mitigating effects of peer churn. On the other hand, users have some priority in resource allocation of the streaming protocol run at their friend peers.

To minimize additional bandwidth caused by social activities, we design a decoupled IM server exploiting cloud computing [13]. To the best of our knowledge, Stir is the first spontaneous social P2P streaming system proposed in the literature, which offers satisfactory and personalized streaming services to a large number of users. Simulation results demonstrate benefits of Stir and shed light on building such a system in practice.

In summary, our contribution includes:

- A novel idea of forming social networks spontaneously inside a P2P streaming systems. This removes the problem of sparse friendship networks in a P2P session when social contacts are imported from other social networks. With efficient communication means of IM, we show that the social traffic costs are negligible.
- A social-based streaming protocol, named Stir, whose important component is a social-based partner manager. Based on social knowledge, the partner manager can choose partners to exchange data intelligently. As a result, Stir deals with high churn rates better than existing systems.

The remainder of this paper is organized as follows. Section II presents the motivation for Stir with spontaneous social networking, IM, and Cloud computing. Section III describes the design of Stir with its architecture. A social-based streaming protocol, exploiting the spontaneous social network, is described in Section IV. Section V presents simulation results and analysis. Justification on the costs of social traffic is mentioned in Section VI. Related work is summarized in Section VII. Section VIII concludes the paper and draws future research.

II. THE CASE FOR SPONTANEOUS SOCIAL NETWORKING

In social networking, the term homophily is defined as the tendency of people with similar characteristics to be connected [14]. This homophily principle has been studied and applied in many areas of computer networks. In addition, many studies have demonstrated that there is a correlation from social networks to user behavior on the Web [15]. Particularly, it has been shown that people who chat with each other are more likely to share interests. The more time they spend talking, the stronger this relationship is.

What are actual benefits of forming social networks inside P2P streaming? From the above social principles, users who join and stay in a streaming session are likely to have some similar interests on the stream. Therefore, providing a means of communication between users will not only offer more entertaining services, e.g., it is more exciting to watch a football match with others than alone, but also create social relations among them. From social communications and activities, user behavior can be predicted, e.g., the more friends a user has in a session, the more she is interested in and the longer she stays in the session. Therefore, if connections between peers are established based on social relationships, they are more reliable and durable. In other words, reliable users should connect to each other, and as such they will not be seriously affected by departures of those who do not have a strong interest in the session and stay for a very short period of time. This naturally minimizes the impact of peer churn to maintain continuous playback. Friendship is spontaneously formed during a session but lasts longer than one session. Users have their profiles with a friend list. Connections among friends are immediately established when they join another session together. The coincidence of joining the same session of friends has a high probability because they have very similar interests. Such a streaming system offers personalization as users can show their personality and make friends. In addition, users, who stay long in the session, are rewarded with a stable quality stream regardless of high churn rates.

What is a suitable social communication means in the scenario? The foremost requirement for a communication means in our system is that it does not consume much bandwidth, which is a critical resource in streaming. Another criterion is that it is able to achieve a certain level of synchronization because live streaming is highly synchronized among users (small time lag). Among many means ranging from non-interactive, e.g. emailing, to real time interactive, e.g., voice chatting, IM is the most suitable one. Many studies on characteristics of IM [16]–[18] have shown that (1) chat messages exchanged among users are usually very short, but (2) they are expressive enough to support a variety of informal communication tasks in a semi-synchronous way.

How about costs of IM servers? IM needs a server to route text messages since each user is identified by a unique screen name, not by an IP address. The place to store and update user profiles is also important when the system has thousands, or even millions, of users. Although IM does not consume much bandwidth of each user, the bottleneck is at the server because it needs to handle thousands of connections, e.g., in a large P2P IPTV system. In Stir, rather than putting the tasks to the streaming server, we propose an architecture to separate these social services from the streaming server. Taking advantages of cloud infrastructures (IaaS) and platforms (PaaS) [13], large-scale services can be built and deployed without the need of caring about expensive servers. Google have proposed a framework for Cloud-to-Device Messaging services that allow a third-party application server to send data to its clients via the Google Cloud [19]. Currently, such services are already available for Google Android devices. We believe that decoupling social services from P2P systems is a trend in future.
III. STIR: SCHEMATIC AND ARCHITECTURAL DESIGN

STIR is a multi-channel P2P streaming system that provides live streaming content. One example of such a system in reality is P2P-based IPTV. Each Stir user has a profile containing a unique identifier, a password, a friend list, and the current IP address. A user $U$ needs to log in to the system first before watching any channel. There are no privacy concerns in the Stir architecture, as such login information does not have to reveal any personal information — an email address would suffice. The authentication process is controlled by a cloud-based user manager. If $U$ is authenticated, the manager sends the profile back to $U$ and a notification with $U$’s IP address to the streaming server. The streaming server will send the channel list to $U$. When $U$ selects a channel, the streaming server will send an IP address list of some available peers in the system to $U$. This step is similar to the joining process in traditional P2P streaming systems. Now, $U$ can create connections with other users to download video data. This initial phase is illustrated in Figure 2.

In Stir, friends are to be established on-the-fly and spontaneously, as a group of users watches the same channel. While watching a channel, $U$ can post comments to one of the Stir online forums. This forum is only visible to those who are watching the same channel with $U$. $U$ can have a private chat (via IM) with another user $V$ if she knows $V$’s identifier, and the chat can be entirely conducted in a web browser, as the channel is being played live. $U$ can add $V$ to her friend list at any time spontaneously. The list of friends constitutes a state that carries over from one session to the next: being in the friend list of $U$ means that $U$ will know the status (if $V$ is in the system or not, which channel $V$ is watching) of $V$ whenever $U$ logs in to the system. The friend list is updated to the user manager when it is changed.

STIR uses a pull-based streaming protocol, which means each peer will pull video data from other peers based on buffer map exchanges, e.g., similar to CoolStreaming [1]. However, different from traditional P2P streaming which has one list of anonymous peers (called neighbors), each Stir peer has two lists for potential partners. In addition to the neighbor list, which can be updated by a gossip-based mechanism, the friend list containing friends who are also watching the same channel is also used. In addition to an IP address, each item in the lists may contain some statistical data collected from the social activities of users, which are stored in the social log. These statistical data will be used by the partner manager and the scheduler. The partner manager selects potential partners from the lists for requesting data based on data availability in the playback buffer, social factors and network metrics. The scheduler schedules requests of missing data, and sends them to the selected partners. Received video packets are stored in the playback buffer and will be sent to the player when the playback deadline is reached. Figure 3 shows the components and their interactions.

As being shown in Figure 3, a user may talk with some people, while video packets are exchanged with other ones. This is a key difference compared to existing studies on social-based P2P, which assume that there are always enough friends in the system to have trusted connections and discourages connections with strangers. In Stir, both social relationships and network metrics are taken into account to choose partners. This is helpful for new users who do not have (or have very few) friends.

IV. A SOCIAL-BASED P2P STREAMING PROTOCOL

With the main components being presented in the previous section, this section goes into design details of the partner manager to understand how the underlying P2P overlay relates to the high level social network, and how social data can be used in streaming. Other components can be inherited with modifications from a traditional pull-based streaming protocol, e.g., CoolStreaming [1], or an adaptive one, e.g., Chameleon [20].

A. Friendship and Benefits: a Tradeoff

The general idea of a social-based P2P streaming system is that the establishment of network connections between peers is guided by social relationships between users. However, if a protocol design is heavily based on friendship and discounting connections with others, peers may not acquire enough qualified connections to maintain smooth playback, because their friends may not have sufficient bandwidth. The objective is not only to give friends some priority in resource allocation but also achieve smooth playback for the peer itself. We propose to use a utility function, which takes both social factors and
network metrics into account, as the basis for mitigating the tradeoff.

1) Social Metrics and Network Metrics: The following three metrics should be considered when a peer \( P \) evaluates another peer \( Q \):

- **Network Capacity**: includes different ‘physical’ capacity of \( Q \). This kind of metric can include bandwidth capacity, RTT to \( P \), physical distance, etc.
- **Social ‘Capacity’**: captures the ‘prestige’ of \( Q \) in the social network based on its social relations and activities. This can combine several factors: number of friends, number of social messages \( Q \) has delivered, etc.
- **Friendship**: represents the direct social relation between \( P \) and \( Q \). Is \( Q \) a friend of \( P \)? How often do they chat with each other? How many IM are exchanged between them?

2) Utility Function: There could be several ways to define a utility function combining all above metrics. In this work, we have experimented with the following simple, yet effective, weighted combination:

\[
U_Q = (1 - \alpha - \beta) \cdot C(Q) + \beta \cdot S(Q) + \alpha \cdot F(P, Q)
\]

where \( C \) is a capacity-related function, \( S \) is a social-related function, and \( F \) is a friendship-related function. \( \beta \) is called social coefficient, as it determines how important social capacity is in the evaluation. \( \alpha \) is called friendship coefficient, as it determines the priority of friendship. In traditional non-social P2P streaming, \( \alpha \) and \( \beta \) are 0 as social factors do not exist. In previous social-based P2P systems, \( \alpha \) is close to 1 as they discourage connections with strangers. By experimenting with wide ranges of values of \( \alpha \) and \( \beta \), we understand interactions of the social network and P2P network, and how benefits may be derived from these interactions.

In this paper, the capacity function \( C \), the social function \( S \), and the friendship function \( F \) are calculated for \( Q \) in a list \( L \) as follows:

\[
C(Q) = \frac{B_Q}{\max_{i \in L}(B_i)}
\]
\[
S(Q) = \frac{N_Q}{\max_{i \in L}(N_i)}
\]
\[
F(P, Q) = \begin{cases} 
1, & \text{if } P \text{ and } Q \text{ are friends} \\
0, & \text{otherwise}
\end{cases}
\]

where \( B_i \) is the bandwidth capacity of peer \( i \), \( N_i \) is the number of friends of \( i \).

**B. Partner Manager**

The interaction between the social network and the overlay network happens in the partner manager, which determines a group of active peers for sending and receiving data. When a Stir peer \( P \) selects partners, its partner manager calculates utility values of peers in the neighbor list and the friend list based on the utility function. After that, it sends partner requests to a certain number of peers, which has the highest utility values. If a candidate \( Q \) accepts the request, \( P \) adds \( Q \) to its partner list, which is used for buffer map exchanges and video data requests. The partner acceptance check is also based on the utility function. The algorithms for the selection of partners at \( P \) and the acceptance check at \( Q \) are presented in Algorithm 1 and Algorithm 2, respectively.

**Algorithm 1 Partner Selection at \( P \)**

\[
\begin{align*}
NL &: \text{neighbor list.} \\
FL &: \text{friend list.} \\
U &: \text{the list of utility values.} \\
N &: \text{the maximum number of partners (system parameter).} \\
SL &: \text{sorted list in increasing order.} \\
\text{utlCal}(X) &: \text{calculates the utility value of } X. \\
\text{sortOnUtil}(X, Y) &: \text{sorts } Y \text{ on } X \text{ and returns the sorted list.} \\
\text{isAccepted}(X, Y) &: \text{sends a request to } Y \text{ for acceptance check on } X. \\
\text{addPartner}(X) &: \text{adds } X \text{ to the partner list.} \\
L &= NL \cup FL; \\
\text{for } i = 1 \text{ to } L.length \text{ do} \\
& \quad U[i] \leftarrow \text{utlCal}(L[i]); \\
\text{end for} \\
SL &= \text{sortOnUtil}(U, L); \\
i &= 0; \\
\text{while } (i < SL.length \land Q.no\_of\_partners < N) \text{ do} \\
& \quad \text{if } (\text{isAccepted}(P, SL[i])) \text{ then} \\
& \qquad \text{addPartner}(SL[i]); \\
\text{end if} \\
i &= i + 1; \\
\text{end while}
\end{align*}
\]

As shown in Algorithm 2, each peer has an acceptance list that contains peers whose partner requests have been accepted. Being in the acceptance list of \( Q \) means that \( P \) can send data requests to \( Q \) and will be served if \( Q \) has sufficient resources. Different from the partner selection process that depends fully on the utility value of candidates, the acceptance check has to give friends higher priority than others, regardless of their social capacity or bandwidth. This is a design principle of Stir to encourage people making friends and sharing their interests.

**C. Packet Scheduler**

In traditional pull-based P2P streaming protocols, buffer maps are exchanged between a peer and its partners to decide who will deliver which packets. However, in Stir, before the buffer map exchange, the peer needs to send a confirmation request to each partner to make sure that it is still in the acceptance list of the partner. After that, buffer maps can be exchanged, and packets can be requested from confirmed partners.

The reason for this step is that the acceptance list of a peer can be changed during the streaming session. For example, at the beginning, a peer is willing to serve non-friend peers because its friends have not joined the session yet. However, when receiving partner requests from friends and the acceptance list is full, it has to remove some non-friends from
the list to serve the friends better. As a result of this, the partner list of those non-friend peers is changed, and needs to be updated by invoking the partner selection. This additional action does not cause much overhead because the size of the partner list is small, $5-10$ peers. In addition, only the partner list of those who have very few friends and low social capacity may be changed frequently due to their low utility values. Other tasks of the scheduler, e.g., sending requests for urgent packets first, can be done in traditional ways, e.g., the one of CoolStreaming [1]. Thanks to knowledge from the social network, with a set of socially selected partners, we expect that a traditional pull-based packet scheduler, could achieve significant improvements in streaming quality.

V. STIR: EXPERIMENTAL RESULTS

To evaluate our design, we implement STIR in our own discrete-event flow-based simulator. To make the simulator more realistic, we use the extended version of the max-min fair rate allocation which was used in our previous work [20]. Since user behavior and the friendship establishment process can not be simulated, we focus on interactions between the social network and the P2P overlay, while making assumptions on the social network formation.

Algorithm 2 Acceptance Check at Q

```
AL: acceptance list.
AN: the maximum number of accepted peers.
addAcceptedPeer(X): adds X to the acceptance list.
isFriend(X, Y): whether X is a friend of Y.
removePeer(X): remove X from the acceptance list.
if (P ∈ AL) then
    return true;
end if
if (AL.length < AN) then
    addAcceptedPeer(P);
    return true;
else
    min_util ← ∞;
    for i = 1 to AL.length do
        U ← utlCal(AL[i]);
        if (not isFriend(AL[i], Q) ∧ min_util > utlCal(AL[i]))
            then
                min_util ← U;
                r ← AL[i];
        end if
    end for
    U ← utlCal(P);
    if (isFriend(P) ∨ U > min_util) then
        removePeer(r);
        addAcceptedPeer(P);
        return true;
    end if
end if
return false;
```

A. Data Preparation and Assumptions

We need (1) a real-world social graph representing a friendship network, and (2) join and leave times of real-world users.

1) Social Graph: We are not aware of the availability of social graphs formed spontaneously in a particular context as in the case of Stir. As an alternative, we use a network of people who are interested in a particular topic to represent the network of users joining a streaming session. In particular, we develop our own “graph data provider” plug-in in NodeXL [21] to retrieve friend lists of members of a group in Flickr. With the URL of any public group, the provider can collect user IDs of members of the group and their friends. Since the member list of a public group and the friend list of a user are public, we do not violate any privacy rules of Flickr. From the dataset, we form a social network for our experiments as follows:

- All group members are considered as users of a streaming session, i.e., a Flickr group is considered as a streaming session.
- If a member $P$ is in the contact list of another member $Q$, they are friends of each other in the session. Since we consider relationships among users who join the same session, non-members in the contact lists of the members are removed.

Finally, we have a friendship network of users who join the same session. We have collected datasets of different Flickr groups with different sizes to choose a suitable one, which (1) has about one thousand members (a medium-large P2P session), (2) whose network is not very sparse or very dense because we have observed many groups having only few connections among members, while in some specialized groups (for experts), almost all members know each other. It may seem a bit far-fetched to assume that common interest in a topic for pictures in Flickr would correlate with common interest in a live stream in a P2P streaming system. On the other hand, social interaction data from people who are, e.g., watching a TV channel together were not available to us, and we therefore believe that a network of common interests and friendship (as opposed to mere friendship connections) is as close as we can get.

After the selection process, we choose the dataset from the group “Photo Computer Art”. When we collect the dataset, this group has 1280 members. The CDF of the degree of the members in the friendship network is shown in Figure 4, which indicates that $\sim 10\%$ of members have no friends, $\sim 80\%$ have fewer than 20 friends, and the other $20\%$ have from 20 to 54 friends.

2) Peer Dynamics: From the snapshot trace of PPLive [2], available at [22], we extract the join and leave times of users for a period of time (2 hours) on a particular channel. We use three datasets with different levels of peer churn: low, medium, and high. The CDF of the stay duration of peers in the datasets are shown in Figure 5.

$^1$The graph data provider is open source and available upon request.
3) **Assumptions on User Behavior:** We need to combine the peer dynamic datasets and the social dataset to have a complete picture: who are friends of whom, and when they join and leave the session. Based on the discussion in Section II about the spontaneous social networking formation, we believe that the following assumptions are reasonable: (1) friendship indicates similar interests on the content, and (2) the more friends a user has, the longer she stays in the system. With these assumptions, we can join the datasets as follows:

- Sort peers on their stay duration.
- Sort users on the number of friends they have.
- Assign join and leave times to users so that peers with the longest duration have the highest number of friends, etc.

4) **Bandwidth Settings:** We set the streaming rate to 400 Kbps, and the bandwidth capacity of each peer is assigned randomly to one of the following values (download, upload): (450, 300), (550, 450), (700, 650), and (750, 700) Kbps. With 1280 peers, the server upload rate is set to serve 65 (~ 5%) peers simultaneously.

B. **Comparison with Existing Work**

We implement a CoolStreaming-like protocol based on the description in [1] and a network coding (NC) based protocol, called NCStream, in our simulation to evaluate their performance in terms of playback skip rates. The packet scheduler of CoolStreaming and Stir are quite similar, except for the change mentioned in Section IV. On the other hand, the implementation of NCStream is based on the implementation we did in our previous work [20]. The key difference between Stir and the other protocols is the partner manager. While CoolStreaming and NCStream base only on network metrics to choose partners, Stir takes social relationships into account.

1) **On Peer Dynamics:** In this experiment, (α, β) in the partner selection, and the acceptance check of Stir are set to (0.2, 0.3), and (0.7, 0.2), respectively. The skip rate of the systems under different peer dynamic scenarios is shown in Figure 6.

Figure 6 shows that the three systems can achieve similar low skip rates when the network is quite stable (low dynamic), actually NCStream is the best protocol in this case because NC helps to utilize the bandwidth better. However, the performance of them are notably different in the medium and high dynamic scenarios. Only ~ 0.54% of playback segments are skipped by Stir peers in the high dynamic case, while the percentage for NCStream and CoolStreaming is 0.71% and 4.17%, respectively. The reason for the superior performance of Stir under high churn rates is that peers who stay longer in the session are likely to have a certain number of friends and exchange data to each other. In addition, the social-based acceptance check gives friends higher priority in data delivery. Consequently, even when a large number of users who are not interested in the session leave, the communities of friends are not seriously affected. For example, we calculate the traffic (number of packets exchanged) between friends and that between non-friends in the protocols for the high dynamic case: ~ 60.6% of traffic in Stir is among friends, while, without social knowledge, up to ~ 65.8% of traffic in CoolStreaming (~ 64.2% in NCStream) is between ‘strangers’. Although the percentages depend on how dense the friendship network is, this experiment indicates the ability of exploiting social knowledge in Stir.

One may be concerned that the skip rate also depends on the choice of segment sizes in Stir and CoolStreaming, and NC block size in NCStream. Generally, the larger the segment size is, the higher the skip rate is for CoolStreaming and Stir. As shown in [5], there is a tradeoff in choosing NC block size. Small block sizes are more robust to peer churn, but cause more overhead for coding coefficients. In the above experiments, we set the segment size to 2 seconds of playback in CoolStreaming and Stir, and the NC block size to 1 KB. Although not shown here, with the same segment size Stir always achieves much better performance than CoolStreaming. Since the use of social knowledge and NC can be combined, we expect that the combination even offers better performance.

2) **On The Size of Neighbor List:** In CoolStreaming and NCStream, the neighbor list is the local view of peers to the network. Therefore, the streaming quality a peer receives totally depends on its neighbors. It has been shown that the size of the neighbor list should be from 50 to 60 regardless of the network size. In Stir, in addition to the neighbor list, a peer has the friend list, which is expected to include ‘good’ peers. It should be noted that peers in the friend list can
also appear in the neighbor list because the neighbor list is updated by gossiping and independently from the social network. We would like to answer the two following questions: how important is the neighbor list in Stir? and How big should it be? We plot the skip rate of the protocols with the size of the neighbor list ranging from 20 to 60 for Stir, and 40 to 80 for CoolStreaming and NCStream in Figure 7. The reason for CoolStreaming and NCStream having bigger neighbor lists is to have a ‘fair’ comparison between them and Stir because peers in Stir may also have a large friend list.

![Graph showing skip rates with different sizes of the neighbor list](image)

Fig. 7. Skip rates with different sizes of the neighbor list. The numbers in parentheses denote the size of the neighbor list in CoolStreaming and NCStream.

It can be concluded from Figure 7 that (1) using a large neighbor list actually does not always improve the performance in CoolStreaming and NCStream, and (2) with the existence of the friend list, the size of the neighbor list in Stir can be smaller than in traditional systems. The counter effect of larger neighbor lists in CoolStreaming and NCStream can be explained as follows. The larger the neighbor list is, the higher the probability of more than one peers choosing the same set of high capacity peers, i.e., bottlenecks at high capacity peers are likely to occur. The problem does not occur in Stir due to the acceptance check that guarantees that a peer ‘reserves’ resources for its friends.

C. Insights of Stir

Convinced that by exploiting social knowledge Stir deals with peer churn much better than previous work, we now turn our attention to the insights of Stir. We experiment with different values of $\alpha$ and $\beta$ in the partner selection and the acceptance check to answer the following questions:

- Is the role of friendship, network capacity, and social capacity in choosing partners different or similar?
- Between network capacity and social factors (including friendships and social capacity), which one is more important to the system performance?
- Between friendship and social capacity, which one is more important to the system performance?

1) On the Partner Selection: We fix the value of $\alpha$ and $\beta$ in the acceptance check, while using wide ranges of values for the coefficients in the partner selection. Figure 8 shows the skip rate of Stir when $\alpha + \beta = 0.6$ and $\alpha = 0, 0.05, ..., 0.55$ to understand effects of social capacity and friendship in choosing partners (Figure 8b).

![Graphs showing effects of alpha and beta](image)

Fig. 8. The effect of $\alpha$ and $\beta$ in partner selection.

Figure 8a indicates that network capacity is important in the partner selection. If a peer connects to others based heavily on social factors (high values of $\alpha + \beta$), it may suffer playback skips because (1) high social capacity does not mean high network capacity, (2) high social capacity peers have more friends to serve, and (3) its friends may not have sufficient bandwidth. However, a peer should also not depend only on the network capacity (high values of $1 - \alpha - \beta$) because of peer churn: high network capacity peers may only stay in the session for a short period of time. Very low skip rates can be achieved if peers consider network capacity as important as, or slightly less important than, the social factors ($1 - \alpha - \beta = 0.4$, or 0.5).

Between social capacity and friendship, from Figure 8b, we can see that preferring high social capacity peers gives better results than preferring friends for data requests as high values of $\alpha (> 0.4)$ increase the skip rate significantly. Although this phenomenon is somewhat contradict to the idea of connecting friends with each other, it is reasonable from peers’ point of view because the higher social capacity is, the more durable the peer is. However, friendship has a certain importance as setting $\alpha$ to 0 does not achieve best performance. The reason is that if a peer chooses its friends as partners, it will have certain priority at the friend side in their bandwidth allocation. Therefore, if friends of a peer have sufficient bandwidth, the peer will receive higher quality. In addition, being a partner of a high social capacity peer does not guarantee that it will be served. In a nutshell, network capacity, social capacity and friendship have its own role in partner selection. However, different from existing work, when a peer chooses partners, friendship is not the only important factor, but bandwidth and social capacity.

2) On the Acceptance Check: Since peers have the highest priority to be in the acceptance list of their friends, the role of $\alpha$ no longer exists in the utility function. For non-friends, the acceptance check is based on their social capacity and network capacity. On one hand, a peer could prefer high social capacity peers (by setting high values for $\beta$) to reward them as they are ‘famous’ in the social network. On the other hand, high network capacity peers could have high priority for the reason that when they can receive packets quickly, they can deliver
them quickly to other peers. However, our experiments with different values of $\beta$ from 0, 0.1, ... to 1 show no significant differences to the overall system performance (the graph is not shown here). The main reason probably is that peers which have a certain number of friends are served well by their friends, so they do not need to be partners of non-friends. For those, who have very few friends or no friends, their social capacity is quite similar. Therefore, the role of $\beta$ is minor. In other words, the case of high social capacity peers compete with low social capacity ones to be in an acceptance list seldom occurs in our experiments. However, we believe that keeping the utility function at the acceptance check is useful in practice because there still be the case that a peer has a number of friends but some of them may not join the session.

3) The value of being famous in Stir: It has been demonstrated so far that exploiting social knowledge improves the overall system performance. However, since each peer communicates with a small number of other peers at a time without global knowledge, one question remains: *what are system behaviors caused by the protocol run at each peer?* Especially, it is interesting to know for the whole user population of a session: *does high social capacity peers generally receive better quality than low social capacity ones?* In Figure 9, we plot the average quality of peers having the same number of friends.

![Fig. 9. In Stir: The more famous you are, the higher quality you receive.](image)

Generally, Figure 9 shows that the average skip rates reflect the design objective of Stir: the more friends a user has, the higher the quality she is likely to receive. This encourages users to join the system, and creates a friendly collaborative network among users. In details, there are two noteworthy points. Firstly, the skip rate of those who do not have any friend is less than 0.5%, which is still acceptable in live P2P streaming, which means Stir does not discriminate against low social capacity peers. They still can receive benefits from the system. Secondly, for those whose have fewer than 20 friends, higher social capacity does not always bring lower skip rates. If looking back the CDF of the number of friends in Figure 4, we can understand the reason. There are many users (~ 80% of the population) having fewer than 20 friends. As a result of this, many of those users may not receive enough packets from friends, and have to compete with others. As shown in the experiments of the acceptance check, the social capacity of those peers does not have a strong impact.

VI. SOCIAL TRAFFIC COSTS

In Stir, a certain amount of user bandwidth is consumed by social activities. If social traffic costs are expensive, the overall performance of the system can be seriously affected because of insufficient bandwidth for video data. Therefore, justification on social traffic costs for using spontaneous social networking inside a P2P streaming is necessary.

For a user $U$, social traffic includes: friend list downloading when $U$ joins the system, friend list updating when new friendship is made, instant messaging with friends, and Twitter-like commenting. Important information about a friend in the list includes user ID, number of friends, and IP address. It should be noted that not all friends have their IP address available when $U$ downloads the list from the IM server because some friends may not join the session (yet). However, the IM server does not need to update the presence of $U$ to her friends because $U$ will later contact them for partnership requests. This ‘embedded’ status notification is one benefit the streaming protocol brings to the social network because it reduces the cost of status updates for IM. If we use 20 bytes (20 characters) for a user ID, 4 bytes for the number of friends, and 4 bytes for an IP address, a user with 50 friends that are already in the system will download a friend list of negligible 1.4 KB. When $U$ has a new friend, she only needs to send the user ID of the new friend to the server. During streaming, $U$ can send Twitter-like comments to a forum, as described in Section III, to share with others. Such comments are usually very short as those in Twitter or YouTube. So, the most expensive cost comes from IM among friends. Xiao et al. have shown that the overall IM traffic is about 8.9 Kbps, in which chat messages constitute only a small percentage [16]. However, in Stir, there may be more chat messages exchanged among users because they are watching a real time stream. Although it depends on user characteristics and the content they are watching, let us consider the case of watching an interesting football game to imagine how much traffic IM could cause. As shown in [23], a football game has an average of 300 highlights, including goals, shots, etc. Assuming that a user sends 3 IMs to 5 friends at the same time about every highlight during 90 minutes of the game, and each message has 1000 bytes in length (the biggest message length observed in MSN by [16]), she would consume a bit rate of ~ 6.67 Kbps for chatting, which is equal to ~ 1.67% of a 400 Kbps stream.

We believe that the consumption in practice is much smaller than the above synthetic case.

 Altogether, we can conclude that the social traffic costs caused by social networks in Stir are negligible.

VII. RELATED WORK

This section summarizes notable studies in live P2P streaming, and pioneers of applying social networking to P2P systems.
A. Approaches to Peer Churn in Traditional P2P Streaming

Kumar et. al. introduces a stochastic fluid model that accounts for many essential features of a P2P streaming system, including peer churn [3]. The model helps understand the fundamental characteristic and limitations of a P2P streaming system, especially in a dynamic one with churn.

Vu et. al. have a different approach to understand P2P streaming systems in general, and peer dynamics in particular. They measure and model the large-scale P2P overlay graphs of PPLive [4]. Their work gives a better understanding on IPTV users’ dynamics, which are stated to be helpful in improve the system performance.

Wang and Li are successful in applying NC to live P2P streaming [5]. Thanks to NC, the perfect collaboration of peers is achieved. As a result of this, peer bandwidth is better utilized and the system is more robust to peer churn.

Having the same objective of improving the performance of P2P streaming in dynamic environments, we believe that exploiting social network could further reduce the impact of peer churn to the system.

B. Social-based P2P

There have been few studies on applications of social networking to P2P systems, especially P2P file sharing systems. Generally, social networking can be exploited in two ways.

Firstly, P2P systems can mimic how people form a social network and how they query, by preference, their friends or acquaintances to construct overlays, which can achieve more efficient routing and data locating. A notable study following this approach is of Lin et. al. [10]. They introduce SocioNet, a social-based multimedia access system for unstructured P2P networks. SocioNet clusters peers based on their similar interests. In addition, peers also maintain a certain number of connections to others with different interests as “shortcuts” that connect them to other parts of the network. They show that such social-based overlay construction create small-world networks, and achieves a higher success ratio than non-social-based overlays.

Secondly, P2P systems can import social graphs from other social networks, and peers establish connections with those they have relationships with in the social graphs. TRIBLER [9] is such a system. It (1) facilitates the formation and maintenance of social networks by importing existing user contacts from other social networks, e.g., MSN, and (2) exploits the social networks to create connections among peers. Implemented as a set of extensions to BitTorrent, TRIBLER has been demonstrated to be able achieve fast, trusted content discovery and recommendation, and a significant improvement in download performance.

Similar to TRIBLER in using existing social graphs, Altmann and Bedane present an algorithm for topology formation in a P2P file sharing system [24]. They exploit the fact that peers are willing to contribute their resources to a P2P network if they know that their resources directly benefit their friends and family to search and relay shared files between network leafs that are located behind Firewalls/NATs.

More recently, Liu et. al. present a new incentive paradigm, Networked Asynchronous Bilateral Trading (NABT), based on social networking. The idea of NABT is that each peer has a set of friends, which can be potentially derived from other social networks, and each pair of friends keeps track of a credit balance between them. Credits can be used for ‘buying’ services, e.g. a file, and can be exchanged via friends-of-a-friend relations. Simulations show that NABT can have high trading efficiency, provide service differentiation, and discourage free-riders.

While the above systems are designed for P2P file sharing, the idea of establishing connections among peers based on social relationships can be used in other P2P applications, including live P2P streaming. However, we are not aware of the existence of any social P2P streaming system. The major obstacle could be live P2P streaming poses a stringent timing requirement, and using existing social data does not bring much benefits, as discussed in Section I.

VIII. Conclusion

In this paper, we present Stir, a new framework towards a tightly integrated spontaneous social networking in P2P streaming, as well as a social-based streaming protocol exploiting social relationships of the social network. We show that by offering cheap, yet efficient, communication means to users who join the same streaming session, the P2P streaming system is not only able to provide more entertaining services from the perspective of users, but also achieves much better performance compared to previous systems, especially when dealing with high peer dynamics. Simulation with real social network data and real peer dynamic traces demonstrates our approach. We believe that forming social networks spontaneously on top of P2P overlays would bring significant benefits for both users and the P2P systems. This will be a trend in the future, and we hope that our study on Stir will shed light on how such a system could be built in practice.

REFERENCES


