Journal of Network and Computer Applications 33 (2010) 183–202

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

On the self-organization of a hybrid peer-to-peer system ☆

Yuh-Jzer Joung [a,*], Zhang-Wen Lin [b]

[a] Department of Information Management, National Taiwan University, Taipei 10617, Taiwan
[b] Acer, Inc., Taipei 106, Taiwan

ARTICLE INFO

Article history:
Received 31 December 2008
Received in revised form
10 April 2009
Accepted 19 August 2009

Keywords:
P2P
Unstructured overlay
Structured overlay
Hybrid overlay
Overlay construction

ABSTRACT

Decentralized peer-to-peer (P2P) systems can be classified into *unstructured* and *structured*. The former is easy to implement, and often simply uses flooding for search, which can be effective only when target objects are popular or nearby. The latter requires peers to cooperate closely to maintain an overlay topology so as to ensure an efficient routing path between any two nodes. Recently, a hybrid use of both paradigms has gained its popularity in several popular file sharing tools to take advantage of each. What is lacking, and thus the purpose of the paper, is a fully decentralized algorithm to build such hybrid systems, as existing methods often require human intervention and some centralized gateway to select peers and guide them to build the structured overlay. The challenges include how to ensure that only one connected overlay is constructed in the lack of any global knowledge, and that only stable peers are selected for the structured overlay so as to reduce its maintenance cost. In addition, the construction must be efficient, scalable, robust, and easy to implement in a highly dynamic environment.

1. Introduction

According to the way overlay networks are organized, existing P2P systems can be roughly classified into four families: (*decentralized*) *unstructured*, (*decentralized*) *structured*, *partially centralized*, and *hybrid systems*.

In unstructured P2P systems (e.g., Gnutella), peers connect to one another at will, as the choice of neighbors is irrelevant to a correct functioning of the major operations (e.g., search) in the systems. Object placement is basically also arbitrary. Due to lack of a mechanism to gather object information and/or to infer correlation between objects and the topology, a blind search process like flooding is inevitable in order to find the interested objects. The flexibility of the topology makes them adapt better to highly dynamic network environments. The blind search process essentially imposes no restrictions on queries, thereby allowing complex search like keyword search or range query to be conducted as well. However, flooding costs too much and is unscalable. Techniques like iterative deepening, directed BFS, routing indices (Yang and Garcia-Molina, 2002), probability indexing (Cheng and Joung, 2006), and random walk (Lv et al., 2002) have all increased scalability and search performance of unstructured P2P systems, but improvements are often limited when retrieving rare and distant objects.

In contrast, structured P2P systems (e.g., CAN, Ratnasamy et al., 2001 and Chord, Stoica et al., 2001) have a well-structured overlay network topology to assist routing and object placement. Among them, *distributed hash tables* (*DHTs*) have emerged as the most popular scheme in this family. In DHTs, each peer acts as a hash table bucket of a globally agreed hash function. Objects are inserted into the network with a unique key. Search in DHTs is a guaranteed and efficient process. Peers take only a small number of messages, which is typically logarithmic of the overlay network size, to locate target objects. DHTs, however, suffer from two fatal problems—robustness and search flexibility. This is because efficient and effective structured P2P systems rely on close cooperation between peers to maintain their somewhat inflexible topologies, but P2P participants are often unstable and unreliable. In addition, because hashing functions wipe out most information of objects, DHTs must find other ways to perform complex search. Extra data overlays have been proposed to facilitate complex search (Reynolds and Vahdat, 2003; Andrzejak and Xu, 2002) but they cost considerable overhead.

In partially centralized systems (e.g., Napster and recent versions of Gnutella), pre-established or elected super peers provide centralized services, typically object indexing for ordinary peers. The use of centralized mechanisms greatly reduces system complexity, and also enables flexible and efficient search. These systems, however, are also vulnerable to attacks if the number of super peers are relatively small. Even if the number is large, the lack of a well-cooperative network structure among super peers

☆This research is supported in part by the National Science Council, Taipei, Taiwan, Grants NSC 95-2221-E-002-058 and NSC 96-2628-E-002-026-MY3.

* Corresponding author. Tel.: +886 2 33661183; fax: +886 2 23627094.
E-mail addresses: joung@ntu.edu.tw (Y.-J. Joung), imtaco@gmail.com (Z.-W. Lin).

1084-8045/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2009.08.002

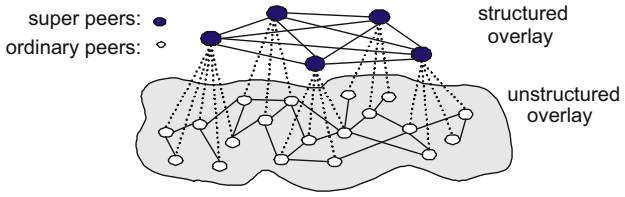

Fig. 1. The Envoy architecture overview.

will reduce their overall utilization and limit their further development into an Internet-scale system.

The final family, hybrid systems, tends to simultaneously adopt various approaches in other families to complement drawbacks of others. For instance, flooding and DHT-based approaches are good, respectively, at searching popular and rare objects. So systems could be made much more efficient by adopting different searching approaches according to the object popularity. This observation has indeed been supported by some live measurements from the Gnutella workload (Loo et al., 2004), and adopted by many popular file sharing tools, e.g., eMule, RevConnect, Overnet, and BitComet.

The synergy of the hybrid systems can be boosted by taking network heterogeneity into account. It is well-understood that P2P nodes have quite different characteristics, e.g., uptime and processing power (Saroiu et al., 2002). Moreover, unstructured networks are more resilient to churns than their structured counterparts. By leveraging node heterogeneity, we can make the structured overlay more robust by selecting only powerful and stable nodes to serve in the overlay (Joung and Wang, 2007).

In this paper we add another system, called *Envoy* (see Fig. 1), to the hybrid family. Our design philosophy is in concord with the above observations. First, we use an unstructured P2P network as our base because of its simplicity in maintenance, robustness in dynamic environments, flexibility and efficiency in search of popular objects, and, most importantly, being practically available. Then we build a DHT overlay over the base to assist search of distant and rare objects, thereby increasing scalability. Search in Envoy can thus be operated in two modes: *flexible search* in the unstructured overlay, and *guaranteed key-based search* in the structured overlay.

The focus of the paper will be on the construction of the overlay, as an efficient and self-organizing mechanism for constructing the Envoy structured overlay without the use of any centralized mechanism and global knowledge is nontrivial and, to our knowledge, has not been proposed in the literature. The challenges include how to automatically elect super peers to form the structured overlay so that the elected super peers are stable enough to reduce the maintenance cost (that is generally high for structured P2P networks), and powerful enough to provide services for the peers that elect it. Other challenges lie in avoiding concurrently elected super peers to form disjointed overlays in the absence of any centralized bootstrap server, and in making the algorithm efficient, scalable, robust, and easy to implement.

2. Related work

There have been several approaches to cope with network heterogeneity. The most popular way is to cluster peers, and select a super peer in each cluster as a local server to manage the cluster as well as to index objects in the cluster. Intra-cluster communication and lookup can therefore be efficiently done via the super peer of a cluster. The super peers also form an overlay to facilitate inter-cluster communication. The overlay is typically unstructured, e.g., KaZaA, Gia (Chawathe et al., 2003), and recent versions of Gnutella.

If ordinary peers are also connected, then the above approach can be classified as building an unstructured overlay over another one. In contrast, Brocade (Zhao et al., 2002) organizes a structured overlay over another structured overlay to speed up routing, as the super peers are chosen from some network access points such as gateways. Chord^2 (Joung and Wang, 2007) selects stable nodes from a Chord overlay to build a second layer of Chord to assist overlay maintenance. Expressway (Xu et al., 2003) builds the auxiliary overlay by taking proximity into account. HONet (Tian et al., 2005) uses a similar approach, but adds some random links between cluster nodes to further improve routing. An extreme approach is taken in Mizrak et al. (2003) to push all the lookup load into super peers and to have each maintain a global index table so as to achieve constant-time lookup.

Rather than building an extra overlay, hierarchical structure has also been used in an overlay to improve search, routing efficiency and scalability, e.g., Kermarrec et al. (2003), Zhang et al. (2004) for unstructured overlays and Andersen et al. (2001), Garcés-Erice et al. (2003), Ganesan et al. (2004) and Lu et al. (2009) for structured overlays.

In the above systems, all layers of a network, whether physically or logically separate, are homogeneous—they are all structured or all unstructured. The combination of structured and unstructured overlays appears in Loo et al. (2004) and Singh and Liu (2004). In Singh and Liu (2004), peers in a structured overlay are grouped together in a Gnutella-like network, and a rendezvous node is selected from each group to provide anonymity service for other members of the group. The work in Loo et al. (2004) is most similar to ours, where, based on live measurements of the Gnutella workload, a structured overlay is suggested over the Gnutella network to improve search quality, in particular, search of rare objects. They also suggest using only stable peers in the structured overlay to reduce maintenance costs. In fact, such an idea has also been experimented by eMule (in combination with Kademlia, Maymounkov and Mazières, 2002) to combine the best of both worlds: flexible search in unstructured overlay and guaranteed search in structured overlay. The idea is gaining its popularity in several other file sharing tools, e.g., RevConnect, Overnet, BitComet, etc.

Building the above hybrid systems involves two steps: (1) select qualified peers from the underlying unstructured network, and (2) organize the selected peers into a structured overlay. In the first step, existing systems often determine the capability of a peer simply by its hardware specification, or let users manually choose whether they wish to be a super peer. The two methods do not necessarily ensure that a selected peer is qualified (for example, a node with good hardware may only be up for a limited amount of time). A more plausible way is to allow peers to compare among themselves which ones are more qualified. In the second step, a hookup gateway is often used to guide nodes to join and form a structured overlay. The main contribution of the paper is therefore to propose a fully decentralized algorithm that can automatically select qualified peers to form the structured overlay without resorting to human intervention or any centralized mechanism.

The following sections present the Envoy system, including an overview, the detailed algorithms, selection of super peers, some theoretical analysis, and experimental results. Conclusions and future work are offered in Section 6.

3. System design

3.1. Overview

Envoy is composed of two overlays: *structured* on the top and *unstructured* at the bottom (see Fig. 1). Logically, the structured

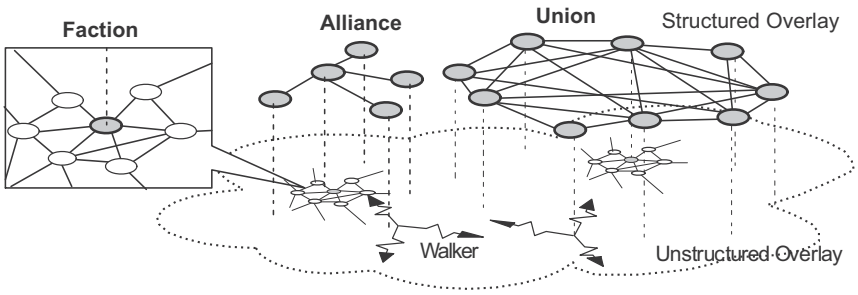

Fig. 2. A closer look at Envoy.

overlay is constructed by peers selected from the unstructured one. However, both overlays may be constructed simultaneously, or we could also defer construction of the structured overlay network after the unstructured one is carried out. The former applies to the case where Envoy is built from scratch as a new P2P network, while the latter applies to the case where Envoy is deployed over an existing unstructured P2P network. For generality, here we assume that both overlays in Envoy are constructed simultaneously.

The unstructured overlay in Envoy is basically a Gnutella-like network. In contrast, there are several choices for building the structured overlay, e.g., Chord, CAN, Tapestry, and Pastry. To increase flexibility, we shall simply assume some procedure for a new peer to join an existing P2P structured network. Such joining procedure determines which specific type of structured P2P overlay one is going to build, and can be chosen independently of our construction.

When peers join Envoy, they follow the Gnutella protocol to establish connections to the unstructured overlay. Some peers may later be elected to join/construct the upper structured overlay. For a distinguishing purpose, the elected peers are called *super peers*. Super peers are elected according to their stability and capability scores. For now, let us assume some scoring function that allows peers to calculate their own scores, and peers faithfully spread their scores. Each super peer manages some ordinary peers in the unstructured overlay, and each peer has exactly one super peer it acknowledges.[1] The relation between super peers and their responsible peers are established through a period of mutual recommendation. A new member to Envoy unstructured overlay will eventually elect some other peer (possibly an existing super peer) as its super peer, or be elected as a super peer by other peers.

A super peer and the peers it manages form a *faction*. A faction might create another faction due to overload, or be absorbed by another faction to form a larger one. A super peer needs to discover other super peers so as to join an existing structured overlay network in Envoy, or to construct a new one. We call the structured overlay network constructed by a set of super peers a *union*. Ideally, only one union should be constructed if the unstructured overlay is connected. However, due to lack of global knowledge, dynamically elected super peers may concurrently organize several disjointed unions before merging into larger ones. Moreover, two unions could never discover each other, resulting in a partition in the structured overlay. Merging two structured overlay networks is generally very complicated and costly (Ganesan et al., 2004). So the challenge is to avoid constructing too many unions, and to allow newly elected super peers to discover existing unions with high probability rather than creating a new one on their own.

To avoid constructing too many unions, when super peers are elected, they will try to discover existing unions and join them. If they cannot discover any union but only isolated super peers, then they form themselves into a temporary, tree-like, loosely coupled structure, called *alliance*. See Fig. 2 for an architectural overview of Envoy. Compared to union mergence, unions and alliances absorb alliances with much lower cost. The smallest alliance consists of just one super peer (i.e., it is simply a faction). An alliance may grow in size as more super peers join into it. When its size is large enough, an alliance will be upgraded to a union.

Although alliances and unions differ in size and structure, it is often convenient to view each of them simply as a *group* consisting of a set of super peers (along with the ordinary peers they manage). A group has an ID, called the group ID, which is the ID of the super peer creating the group. All members of a group know their group ID.

An important procedure in the Envoy construction is to merge groups, which occurs when some super peer discovers a group with a different group ID. There are three cases. First, when an alliance meets a union, we let the members of the alliance leave their group and join into the other, effectively allowing the union to absorb the alliance. This is justified because an alliance has a smaller and looser structure that can be easily broken apart. Second, when two alliances meet, the tree-like structure of an alliance makes it easy to merge them, regardless of which one is merged into the other. The most complicated case is when two unions meet, as both are a structured P2P overlay. However, our design ensures that two large unions are unlikely to coexist (see the discussion below); that is, mergence of unions takes place only in the early stage in which their sizes are still small. Therefore, rather than using a complex mergence algorithm, we simply let the members in one union leave their group and join into the other. Because they are all relatively small, we will not use their sizes to determine which union should be merged into the other, but simply use group ID to break the tie.

Spreading scores and discovering groups are common operations in Envoy. An economical and effective mechanism is essential to perform these operations. For this, we use *random walk*, which has been proven quite effective for searching popular objects in unstructured networks (Lv et al., 2002; Chawathe et al., 2003). When used in discovering groups, peers in a group are viewed as having an identical object representing the identity of the structure. Therefore, the larger the structure, the higher the likelihood for the structure to be discovered. Thus, it is unlikely that some peers will form themselves a new group if some large group already exists. Moreover, recall that an alliance is a temporary loose structure that must grow to a certain size before it can be promoted to a union. Alliances can be merged at a relatively low cost, and when an alliance meets a union, the former can be easily absorbed into the latter. All together, the cost

[1] The requirement that each peer has exactly one super peer can be easily relaxed if applications need.

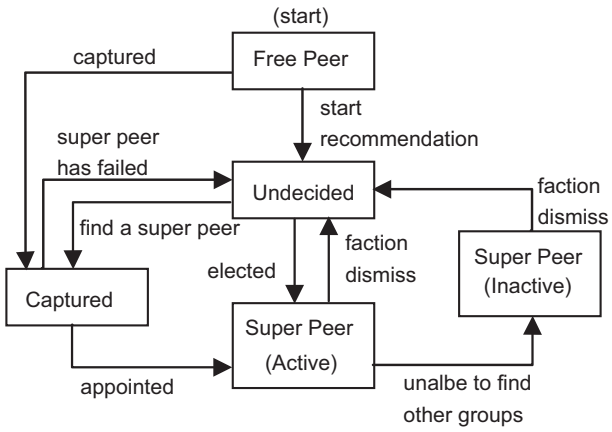

Fig. 3. State transition diagram of peers.

of group mergence is significantly reduced because it is unlikely that two large unions exist simultaneously and need to be merged.

To summarize, random walkers in our system are like "envoys" roaming in the system.[2] Once an unknown homeland lies in the way, the envoys bring the unknown homeland for unification with theirs. As such, the system evolves by letting peers first form into factions, which then form into alliances, which are then upgraded to unions when growing to a certain size, and smaller unions are likely to be absorbed by larger unions, yielding only one final union eventually.

3.2. Detailed algorithms

Our algorithm is represented as a set of (remote) procedure calls. We use *v.func*() to denote a call to *v*'s procedure *func*, and *v.var* to denote a call to access *v*'s variable *var*. A procedure can be triggered by another procedure, periodically invoked by a peer, or invoked by a peer due to the change of state. The state transition diagram is shown in Fig. 3. Initially, every peer is *free*, meaning that it is not yet involved in the construction of Envoy. When a free peer v starts the recommendation process and faction discovery, its state becomes *undecided* until it has discovered a super peer (in which case the peer v is *captured*), or been elected as a super peer. A super peer's state can be further divided into *active* and *inactive*, depending on whether or not it is actively discovering other super peers. A super peer may return to undecided if it loses its captured peers due to failures. Similarly, a captured peer may return to undecided if it loses its super peer.

The construction algorithm can be divided into five parts: peer join, faction discovery and recommendation, update and maintenance within factions, group discovery, and update and maintenance within alliances.

3.2.1. Peer join

A peer v joins into Envoy's unstructured overlay via some *bootstrap* node by invoking *v.joinBase*(*bootstrap*); see Fig. 4. The peer then becomes a free peer until it is ready to start the faction discovery and recommendation process. To start, a peer changes its state to *undecided* and initializes some variables. The use of the variables will be clear later on when we go through the algorithm.

```
// join the unstructured P2P network via a node bootstrap.
v.joinBase(bootstrap)
   if bootstraps ≠ nil
      join Gnutella via bootstrap
   else
      create a Gnutella network
   status := FreePeer      // v becomes a free peer.

// called when a free peer v is ready for recommendation
v.joinEnvoy()
   score := v's stability and capability measure.
   changeStatus(Undecided)

v.changeStatus(newStatus)
   if status = newStatus return
   status := newStatus
   if status = Undecided
      // state initialization
      parent := v     // the peer that v recommended
      parent_score := score
      gid := nil     // v's group id.
      members := ∅     // peers that have recommended v.
      // The set represents v's faction when v becomes a super peer.
      hasRecommended := false // periodically invoke factionDiscovery()
   else if status = SuperPeer
      initSuperPeer()
```

Fig. 4. Peer join.

3.2.2. Faction discovery and recommendation

Faction discovery and recommendation are embedded in the procedure *factionDiscovery*() in Fig. 5. In this process a peer v sends some random walkers with limited TTL in the unstructured overlay. The purpose is twofold: to search an existing super peer, or to recommend one.

When a walker of v discovers a faction (see *visitForFaction*()) at a peer u, u informs v the existence of its faction by writing to *v.SP_discovered* the super peer of the faction. As such, when v's discovery process is timed out, it can choose an arbitrary faction it has discovered to join.

A peer may not be able to find any super peer via faction discovery. So the other purpose of faction discovery is to spread a peer's score, hoping that eventually enough peers will recommend it as a super peer. Basically, the recommendation process works as follows: if a peer v discovers another peer u with a lower score than v has, then u "recommends" v, and waits for v to be elected as a super peer or to recommend a better super peer. The recommendation relationship may result in a heap (see Fig. 6(A)), with an edge (u, v) indicating that u has recommended v. The root of a heap, which has the highest score among the peers in the heap, must continually perform *factionDiscovery*() until either the heap is promoted to faction (when the heap is large enough), or is absorbed by another heap or faction.

To implement recommendation heaps, we use a variable *v.parent* to record v's recommendation. By initializing *v.parent* to v itself (see *v.changeStatus*()), an undecided peer v initializes its heap by setting itself as the only (root) peer in the heap. The execution of *v.factionDiscovery*() may result in two possible outcomes to v's heap (see Fig. 6(B) and (C)): (1) v discovers another undecided peer u, and u has not yet recommended any one, and has a lower score than v's. Then u recommends v by simply setting *u.parent* to v. As a result, u's heap is absorbed by v's.[3] (2) v discovers a faction. Then v joins the faction by setting its

[2] Random walkers are simply passive messages communicated among processes. However, for ease of presentation, they will be viewed as active agents in the paper.

[3] Note that if u's score is higher than v, then u simply passes v's walker to the next random peer. We could speed up the recommendation process by letting u inform v of the existence of a better (undecided) peer so that v may recommend u if it does not find a better peer in an execution of *factionDiscovery*(). This, however,

```
// called periodically by undecided peers until hasRecommended=true
v.factionDiscovery()
   SP_discovered := ∅    // initialize super peers discovered
   select MAX_WALKERS peers randomly from Gnutella neighbors
   for u ∈ selected peers
      u.visitForFaction(v, score, TTL)
   wait until SEARCH_TIMEOUT expires.
   if SP_discovered ≠ ∅ // find some super peer
      parent := a random peer in SP_discovered
      hasRecommended := true


v.visitForFaction(issuer, i_score, ttl)
   if status ∈ {Captured, SuperPeer}
      // issuer's walker has discovered a faction
      issuer.SP_discovered := issuer.SP_discovered ⋃ {parent}
   else if status = Undecided and ¬ hasRecommended and i_score > parent_score
      hasRecommended := true
      parent := issuer
      parent_score := i_score
   if ttl > 0
      w := a random peer in Gnutella neighbors.
      w.visitForFaction(issuer, issuer_score, ttl − 1)
```

Fig. 5. Faction discovery and recommendation.

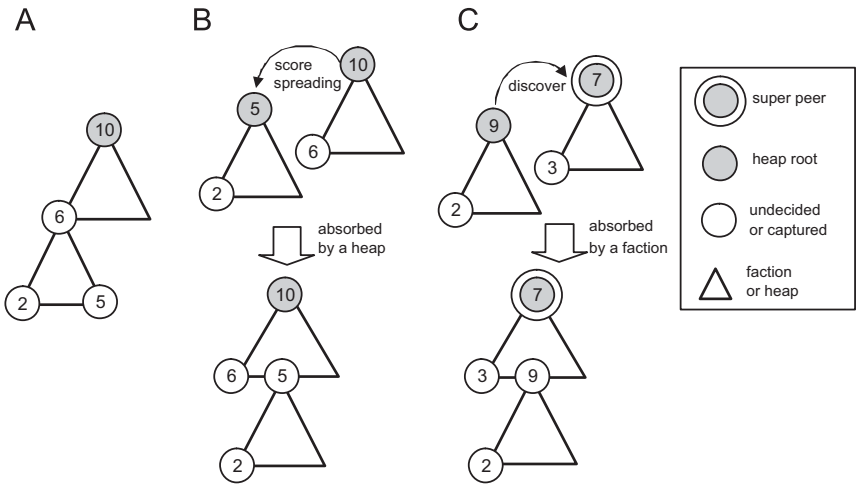

Fig. 6. (A) A recommendation heap. The number in each peer shows the peer's score. (B), (C) Two possible outcomes of faction discovery.

parent to the super peer of the faction. In the next section we will see how the other members of v's heap learn of this result so as to join into the faction.

3.2.3. Update and maintenance within factions

Maintenance of recommendation heaps and factions may become tedious in the presence of failures. For example, assume that the network connection between a heap root v and its recommender u has temporarily failed. This might cause u to seek another heap to join (by restarting its recommendation process and faction discovery), thereby resulting in the situation in which two heap roots both consider the same peer as their recommender. The situation becomes more complicated when one heap is absorbed by another, as the later may have an inconsistent view of the final heap.

Failures in P2P systems are inevitable as peers may come and go at will. So rather than coping with failures in every step of heap/faction construction and impractically maintaining the structure in a tight fashion, we let their members periodically contact their heap root to obtain the state of their structure via *contactParent*() shown in Fig. 7. The heap root returns ⟨NewParent, NewStatus, NewGID⟩ triple via *getFactionInfo*() to its recommenders to update their knowledge about the heap. For example, when a heap root v is absorbed by some other heap/faction, v's recommenders will learn of the new heap root by an inquiry to v, and therefore change its heap root to the new peer. Similarly, if the new peer is promoted to be a super peer, every recommender will also learn of the promotion and change its state from *undecided* to *captured*. Note that the parent-contacting process has the effect of flattening the recommendation heap structure. So if there is an edge (u, v) (meaning that $u.parent = v$) and an edge (v, w), then after u has executed *u.contactParent*(), it will change *u.parent* to w.

On the other hand, the root of a heap can also learn of its members when its children contact it. When the size is large

(*footnote continued*)
is at the cost of more messages and making the faction discovery and recommendation more complex. Our preliminary simulation result indicated that letting a low-score peer u unilaterally set its parent to a high-score peer that discovers u is sufficient.

```
// called periodically by captured and undecided peers.
v.contactParent()
  if parent ≠ v
    ⟨w, w_status, w_gid⟩ = parent.getFactionInfo(v)
    if ⟨w, w_status, w_gid⟩ = nil     // parent has failed
      changeStatus(Undecided)
    else if status ≠ SuperPeer
      ⟨parent, status, gid⟩ = ⟨w, w_status, w_gid⟩

v.getFactionInfo(w)
  if v = parent
    members := members ⋃ { w }
    if status ≠ SuperPeer and |members| > FactionSize
      gid := pid     // assign my id as the group id.
      changeStatus(SuperPeer)
    if status = SuperPeer
      return ⟨v, Captured, gid⟩
    else return ⟨v, Undecided, nil ⟩
  else return ⟨parent, status, gid⟩

// periodically called by a super peer to maintain its faction.
v.factionMaintenance()
  remove overdue peers from members.
  if |members| > 2·FactionSize
    w := the peer with the highest score in members
    M := half of members
    members := members − M
    for u ∈ M
      u.parent := w
    w.appointNewSuperPeer(v, M)
  else if |members| < 1/2·FactionSize
    changeStatus(Undecided)

v.appointNewSuperPeer(appointor, newMembers)
  changeStatus(SuperPeer)
  members := newMembers
  leader := appointor // set the alliance leader
```

Fig. 7. Stabilization and maintenance of factions.

enough, the heap is promoted to a faction, and the root becomes the super peer of the faction.

A super peer v (whether active or inactive) periodically invokes *factionMaintenance*() to keep its faction size in a reasonable range. Faction members that fail to contact v in a certain period of time are removed from the member list. If the faction size becomes too small, the super peer dismisses the faction members by returning its state to undecided (and therefore to restart the recommendation and faction discovery process). On the other hand, when the faction grows too large, the super peer appoints a peer from its faction members that has the highest score as a super peer, and assigns the peer half of its faction members.

3.2.4. Group discovery

In order to unite disjointed groups (alliances and unions), super peers periodically perform group discovery to discover unknown groups until they cannot find a new one. Recall from Fig. 3 that we use *active* and *inactive*, respectively, to distinguish super peers that are still in search of other groups, and super peers that have failed to discover a new one. An inactive super peer remains in the state in the absence of node join and leave. So the Envoy construction algorithm terminates when all super peers become inactive. Note that an inactive super peer still needs to periodically maintain its faction, and node join/leave may cause some peers to enter the undecided state to discover super peers.

Groups are distinguished by their ids, called *gids*, which are the ids of the peers that create the groups. When one group is merged into another, all members of the first group need to update their gid to that of the latter. We shall return to this group information update issue shortly in Section 3.2.5.

As we mentioned in the overview, an alliance is a temporary, tree-like loosely coupled structure to connect a set of super peers that have known each other, but (from an economic point of view) the set has not yet grown large enough to form a well-structured union. An alliance is maintained much like a recommendation heap discussed before. The root of an alliance, called the *alliance leader*, is the super peer that creates the alliance. The smallest alliance is just a faction containing only one super peer. So a new elected super peer is the root of a new alliance, and its id is used as the gid of the alliance (see *getFactionInfo*() in Fig. 7). An appointed super peer joins into the alliance of its appointer (see *appointNewSuperPeer*()) by simply setting its leader to the appointer. The initialization of an alliance leader is given by *initSuperPeer*() in Fig. 8. The procedure is invoked within the procedure *changeStatus*() in Fig. 4 when a peer changes its state to *SuperPeer*.

A group may grow in size due to the generation of appointed super peers or discovery of other groups. Group discovery is done via *groupDiscovery*() in Fig. 8. As group discovery involves mergence of groups of two possible types, the algorithm is somewhat complex. So let us examine the code in more detail below.

First, observe that a new super peer's group type is set to *Alliance*, and its state is initialized to *active* to allow it to discover other groups. An active super peer invokes *groupDiscovery*() periodically before its group type is promoted to *Union*. This avoids an alliance from being isolated and unable to form/join a union. After an active super peer has formed/joined a union, every time it invokes *groupDiscovery*() its state is set to *inactive*. The state can be reset to *active* only by peers of a different group the super peer has discovered in the procedure. Thus, a super peer in a union will stop group discovering unless it is able to discover a new group in its previous discovery process. As a result, all super peers in a union will eventually stop group discovering if no new peer will enter the system.

Second, like faction discovery, a number of random walkers with limited TTL are sent to discover groups. In addition to TTL, each walker carries a triple $\langle issuer, type, gid \rangle$ to represent its group. Type is either *Alliance* or *Union*. The pair $\langle type, gid \rangle$ is used for comparing groups as follows: a union is "better" than an alliance, and if two groups are of the same type, then the one with smaller gid is better. Formally, two pairs $\langle type_1, gid_1 \rangle > \langle type_2, gid_2 \rangle$ if $type_1 = Union$ and $type_2 = Alliance$, or $type_1 = type_2$ and $gid_1 < gid_2$. A group G_1 is better than G_2 if G_1's type-gid pair is greater than G_2's.

During a walker's group discovery journey (see *visitForUnion*() in Fig. 8), if the visited peers are free, they will be invited to join the faction of the issuer. If they are undecided, then they simply pass the walker to the next peer because undecided peers will soon have their own super peers. Otherwise, the visited peer belongs to some group. If the visited peer is not a super peer, it forwards the walker's information to its super peer. Let v denote the super peer that is visited by the walker, or the super peer whose captured peer is visited. Furthermore, let w denote the super peer that issues the walker.

Super peer v now needs to compare which of the two groups (v's and w's) are better. This is done by the procedure *inform*() in Fig. 8, which is called within *visitForUnion*(). The worse group is to

```
v.initSuperPeer()
  parent := v
  leader := v     // v's alliance leader.
  allies := ∅     // v's alliance members if v is a leader.
  type := Alliance     // the type of v's group.
  isActive := true     // super peer status.

// periodically called by an active super peer.
v.groupDiscovery()
  if type = Union then isActive := false // deactivate myself until a new group is discovered

  G_discovered := ∅ // initialize the result of union discovery
  select MAX_WALKERS peers randomly from Gnutella neighbors
  for u ∈ selected peers
    u.visitForUnion(v, gid, type, TTL)
  wait until SEARCH_TIMEOUT expires.
  let ⟨w, w_type, w_gid⟩ be the triple in G_discovered
        with the largest ⟨w_type, w_gid⟩
  if w ≠ ⊥
    for ⟨u, u_type, u_gid⟩ ∈ G_discovered
      u.inform(w, w_type, w_gid)
    if type = Union
      if successfully join a DHT via w
        gid := w.gid
        for u ∈ neighbors in original DHT
          u.inform(w, w_type, gid)
        exit original DHT
    else
      leader := w

v.visitForUnion(issuer, i_gid, i_type, ttl)
  if status = FreePeer
    parent := issuer
    status := Captured
    issuer.isActive = true // activate issuer's status
  else if gid ≠ nil and gid ≠ i_gid
    // visited peer belongs to a different group
    issuer.isActive := true // activate issuer's status
    // inform parent the existence of a new group
    parent.inform(issuer, i_gid, i_type)
  if ttl > 0
    u := a random peer in Gnutella neighbors
    u.visitForUnion(issuer, i_gid, i_type, ttl − 1)

v.inform(w, w_type, w_gid)
  if type = Alliance and v ≠ leader
    leader.inform(w, w_type, w_gid)
  else if ⟨type, gid⟩ > ⟨w_type, w_gid⟩
    w.inform(v, type, gid)
  else if ⟨type, gid⟩ < ⟨w_type, w_gid⟩
    G_discovered := G_discovered ⋃ {⟨w, w_type, w_gid⟩}
```

Fig. 8. Union discovery.

be merged into the other. If v belongs to an alliance, we let v's alliance leader handle the mergence by letting v forward w's group information to its leader. Without loss of generality, let us assume that v is an alliance leader. The two groups are compared by their type-gid pairs. If the visited group (i.e., v's group) is better, v invokes $w.inform(w, type, gid)$, which then causes w (or w's alliance leader) to add v's group information to w's variable *G_discovered* that stores discovered group information. If w's group is better, then v stores w's group information into v's variable *G_discovered*.

The algorithm is designed such that when two groups meet via group discovery, the group information of the better group is written to the other for the latter to merge into the former. Thus a super peer's *G_discovered* may contain only groups better than its group. Also, the procedure *inform*() is carefully designed to cope with temporarily inconsistent group view due to concurrency. To illustrate, consider three groups 1, 2, and 3, where group 3 is better than 2, which is better than 1. Assume that a member x of group 1 has discovered some peer in group 2, and the peer has invoked $y.inform()$ to inform its leader y of the existence of a different group. Consider the following scenario (see Fig. 9):

- y invokes $x.inform()$ as group 2 is better than group 1.
- However, the message arrives *after* group 1 has merged into group 3. So when x receives $x.inform()$ from y, it already has a

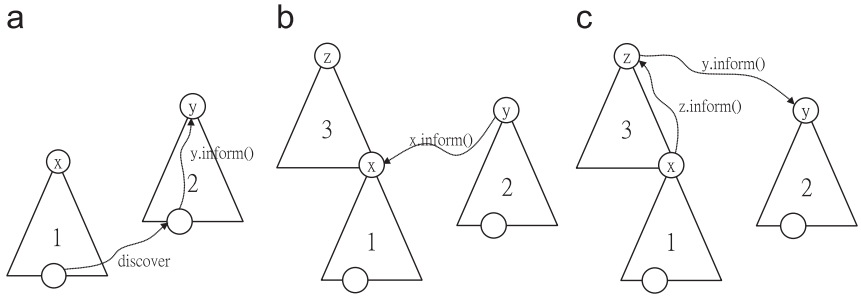

Fig. 9. An illustration of the *inform*() process: (a) a peer in group 1 discovers some peer in group 2, which then informs *y* the existence of group 1, (b) *y* informs *x* the existence of a better group 2 after group 1 has merged into group 3, and (c) *x* informs *z* the existence of group 2, and then *z* informs *y* the existence of a better group 3.

different leader, say *z*, of group 3. So *x* invokes *z.inform*() to pass group 2's information to the new leader.

- *z* finds that its group is better than 2, so it invokes *y.inform*() to send group 3's information to *y*.
- *y* then realizes that there is a better group than its current one, and so adds group 3 to its *G_discovered*.

Alternatively, in the above scenario group 2 may have already been absorbed by another group better than 3 when *z* invokes *y.inform*(). Then a similar sequence of actions will be taken to let *y*'s new leader pass its group information to *z*. Note that this mutual informing actions between two growing groups will eventually stop because of the partial ordering among groups.

After a super peer v has timed out a group discovery process, it selects the best discovered group from *G_discovered*, say G_{best}, for its group to merge into. Before the mergence, v also informs all the other groups in *G_discovered* the existence of G_{best}. As a result, all of them will be merged into G_{best}, thereby boosting the effect of a group discovery.

For v's group to be merged into G_{best}, there are two cases to consider: (1) v's group is a union, and (2) v's group is an alliance. If v belongs to a union, it joins G_{best} (which must be a union too). Then, v leaves its old union (by following the exit procedure in the DHT) and informs its neighboring peers in the old union of the existence of G_{best}. Each of them then follows the same procedure to join G_{best}. In other words, we let each member of a to-be-absorbed union simply leave its union and join into another. This avoids the need of a complicated DHT mergence algorithm, but is practical only if all unions to be absorbed can be kept small. As we shall see, our construction algorithm indeed guarantees such property with high probability.

If v belongs to an alliance, then v simply sets the creator of G_{best} as its new leader. So if G_{best} is also an alliance, then v has effectively switched to G_{best}. v's members will also switch to the new group later on when they contact v for updating their group information (see the following section). If G_{best} is a union, then v will obtain the union information and join into the union later on when it contacts its new leader. All members in v's alliance will also subsequently obtain the new group information from v and join into the new union when they update their group information from v.

One may have noticed that when a super peer v discovers another super peer w of a worse group and sends v's group information to w's *G_discovered*, w may have already become inactive. Similarly, v may discover a better group, but the group information is placed into v's *G_discovered* after v has timed out its group discovery and become inactive. If an inactive super peer ignores information putting into its *G_discovered*, it might lose the chance for its group to merge into another, thereby resulting a partition. So an inactive super peer needs to take the same action as an active one (see *groupDiscovery*()) when it finds that its *G_discovered* is non-empty. However, by our preliminary simulation results, the probability of partition due to this case hardly exists. Therefore, to make the algorithm code more succinct, in Fig. 8 we have deliberately ignored the case for inactive super peers to respond to other super peer's inform actions. Nevertheless, it is not difficult to incorporate the actions into the code to make the algorithm theoretically sound.

Also noteworthy is that when two groups of the same type merge, we use group id to determine which one should be merged into the other. (An alliance is always merged into a union.) Ideally, merging a small group into a large one results in less cost, especially in the case of unions. Considering the dynamics of P2P systems, group size can only be estimated. Several techniques do exist for estimating DHT size (Malkhi et al., 2002; Kostoulas et al., 2005). So, one could further reduce the cost by incorporating them into the mergence. Our main reason for not adopting this alternative here is to keep the algorithm simple and practical, given that it is already very efficient and effective (see the simulation studies in the following section). In practice, merging a large group into a small one hardly occurs because in the presence of a large group, new super peers will discover the large one and join into it before getting a chance to form a new group. Group mergence thus occurs mainly in the initial growing stage during which groups are relatively small.

3.2.5. Update and maintenance within alliances

Recall that an alliance is a tree-like, loosely coupled structure where the leader is in charge of deciding which group to merge into, or when to upgrade itself to a union. In general, an alliance leader remains active until it is absorbed by another group or until it belongs to some union. The other super peers in the alliance learn of the new state of the group by periodically executing *contactLeader*() shown in Fig. 10. The procedure allows them to contact their leader to obtain the most up-to-date information about their group. If they find that the leader has joined a union (i.e., the alliance is to be merged into a union), they will also join the union; and if the leader has joined an alliance (i.e., the alliance is to be merged into another alliance), they will simply change their alliance leader to the new one. This procedure can also easily handle failures of the leader, as if they cannot contact the leader via the procedure, they can just return to an active state to discover a new group.

On the other hand, an alliance leader can also use this procedure to maintain its alliance which may grow or shrink in size due to nodes joining and leaving. If the alliance size exceeds *MinUnionSize*, the leader upgrades the alliance to a union by creating a new structured overlay consisting of the leader itself. Subsequently, all other super peers of the alliance will join the new overlay when they contact the leader via *contactLeader*().

For captured peers in a group, they obtain their group information by periodically contacting their super peers via the procedure *contactParent*() discussed in the previous section. So when a super peer learns of a new gid, this information is effectively pushed down to its faction members when they execute *contactParent*().

```
// called periodically by alliance members.
v.contactLeader()
  ⟨w, w_type, w_gid⟩ := leader.getAllianceInfo(v)
  if ⟨w, w_type, w_gid⟩ = nil    // leader has failed
    initSuperPeer()
  else if w_type = Union
    if successfully join a DHT via w
      type := w_type
      gid := w.gid
  else if type = Alliance
    leader := w
    gid := w_gid

v.getAllianceInfo(w)
  if type = Alliance and v ≠ leader
    return ⟨leader, type, gid⟩
  if type = Alliance
    allies := allies ⋃ {w}
    remove overdue super peers from allies.
    if |allies| ≥ MIN_UNION_SIZE
      type := Union
      create a DHT P2P network.
  else return ⟨v, type, gid⟩
```

Fig. 10. Maintenance of alliances.

3.3. Peer ranking

In Envoy, peers need to elect super peers to serve in the structured overlay. In general, better super peers result in better performance. Two common criteria for selecting super peers are *capability* and *stability*. Capability is a comprehensive evaluation of hardware and bandwidth of a peer. It can be measured by simply mapping hardware specification and/or bandwidth to some predetermined scores.

Stability measures how stable a peer is. Stable peers are unlikely to fail, or to join/leave the system very frequently; i.e., they often have long *session time*—the interval between a peer joining and leaving the system. Stability is closely related to reliability and availability in engineering. Several terms have been defined, e.g., MTBF (mean time between failure), MTTR (mean time to repair), MDT (mean down time), failure rate (1/MTBF), or availability (MTBF/(MTBF+MTTR)), etc. However, none of them alone is able to disclose the number of sessions in a given period, an important factor to the maintenance of structured P2P overlays. For instance, two peers with the same availability do not imply they have the same session time: one could just stay only 1 min and rest for another minute, while the other could stay 1 h and rest for another hour. Although in this case both have the same 50% availability, the former causes higher churn rate Liben-Nowell et al. (2002a, b) to the system, and therefore yields higher overlay maintenance cost.

In general, measurement of stability is very complicated and depends on applications. Here we propose some measurement based on a peer's behavior log. We observe a peer's behavior in some given interval Δ. The peer may have several sessions within this interval (see Fig. 11). Each session τ earns the peer some score $s(\tau)$. Stability of a peer x then is measured by the total scores earned by x in some observed interval Δ:

$$stability_{\Delta}(x) = \sum_{\forall \tau \in \Delta} s(\tau)$$

The key, of course, is how to calculate $s(\tau)$. We provide some clues. First, $s(\tau)$ must be nondecreasing, as longer session time is more welcomed. Second, the length of τ (denoted by $|\tau|$) must exceed certain threshold T, for otherwise it is not worth to elect a peer to the structured overlay. So if $|\tau| \geq T$ then some positive score is rewarded; otherwise a negative score is punished. Third, an uninterrupted session should be much more welcomed than dividing the session into a number of smaller disjointed sessions. For example, the score of a 30-min session has to dominate two sessions of 15-min each. From this we see that $s(\tau)$ is a convex function. The above case concerns the situation where one session has a duration no less than the total session time of a number of disjointed sessions. When the total session time of the latter is larger than the session time of the former, one might prefer the latter case, provided that each session in the latter case is long enough. For example, a peer that has two 20-min sessions might be considered more stable than one that has just one 25-min session. Note that this criterion prevents $s(\tau)$ from growing too fast.

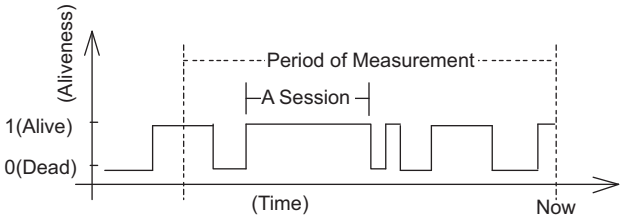

Fig. 11. A peer's behavior in some interval.

We can summarize the above criteria into the following inequalities:

$$\begin{cases} s(\tau) \geq s(\upsilon), & |\tau| \geq |\upsilon| \\ s(\tau) \geq 0, & |\tau| \geq T \\ s(\tau) < 0, & |\tau| < T \\ s(\upsilon) > \Sigma s(\tau_i), & |\upsilon| \geq \Sigma |\tau_i| \\ s(\upsilon) \leq k \cdot s(\tau), & \tau \geq T, \quad |\tau| \geq \alpha |\upsilon| \text{ for some } \alpha > 1/k \end{cases}$$

Based on these inequalities, we can obtain a formula like

$$s(\tau) = (|\tau| - T)^{-\log_{\alpha} k}$$

A peer's final score then is the summation of its capability score and stability score. Both scores can be weighted to reflect different application need. Other criteria, e.g., the number of files shared by a peer, can also be added to let the final score more closely reflect a peer's power.

4. Analysis

In this section we give some formal evaluations to the Envoy construction. We use N to denote the total number of peers and $\overline{F}$ to denote the average faction size. $N/\overline{F}$ thus represents the number of factions in the system.

The fundamental operation in the construction is random walk. Basically, if random walk is used to search a network, and each node has equal probability p to possess the search target, then the success rate of the random walk search is

$$1 - (1 - p)^C \tag{1}$$

where C is the search coverage (number of peers visited). In our construction, the search target could be a peer, a faction, an alliance, or a union. For example, since in the construction faction members are randomly chosen, we may assume that the probability for a random walk to find a target faction is $p = \overline{F}/N$.

4.1. Qualification of super peers

Because better super peers can increase system robustness and object availability as well as reducing the overlay maintenance cost, we begin by evaluating the quality of elected super peers. To avoid dealing with different distributions of scoring functions, we normalize peer scores by defining a *rank* function for each peer v as follows:

$$rank(v) = \frac{\int_{min\_s}^{s(v)} \eta(x)\,dx}{\int_{min\_s}^{max\_s} \eta(x)\,dx}$$

where $\eta(x)$ is the number of peers scored x, and $s(v)$ represents the score of v. As such, rank values are always uniformly distributed between 0 and 1. Thus, a peer of rank 0.95 means that 95% of peers are scored below the peer. Note that rank is used only to analyze how qualified an elected super peer is. The Envoy construction algorithm does not require a peer to know its rank.

Lemma 4.1. *Suppose there are N peers in the system, and σ percent of them have performed exactly one faction discovery, where $0 < \sigma \le 1$. Then, given any $0 \le \beta \le 1$, the percentage of super peers that have rank below β is at most*

$$\max\left\{\frac{e^{(\beta-1)\sigma C} - e^{-\sigma C}}{1 - e^{-\sigma C}}, \beta^{F_{max}-1}\right\}$$

where C is the random walk search coverage and F_{max} is the maximum faction size.

Proof. By our recommendation mechanism, a peer can be elected as super peer only if (i) it is not visited by a walker issued by a higher ranked peer, and (ii) there are enough peers recommending it as super peer. We shall call peers satisfying condition (i) as *qualified* peers, and estimate the probability $Q(\gamma)$ that a peer ranked γ (let us denote the peer by v_γ) is qualified.

There are $\sigma(1-\gamma)N$ peers that have performed one faction discovery, and are ranked higher than γ. The probability that the random walkers sent by any one of these higher ranked peers do not visit v_γ is $(1-1/N)^C$, and so the probability that all the random walkers sent by the higher ranked peers do not visit v_γ is $[(1-1/N)^C]^{\sigma(1-\gamma)N}$. Hence

$$Q(\gamma) = \left[\left(1 - \frac{1}{N}\right)^C\right]^{\sigma(1-\gamma)N} \simeq e^{\sigma(\gamma-1)C}$$

Let $g(\alpha,\beta)$ be the number of qualified peers ranked in $[\alpha,\beta]$. Then

$$g(\alpha,\beta) \simeq \sigma N \int_\alpha^\beta Q(\gamma)\,d\gamma \simeq \sigma N \int_\alpha^\beta e^{\sigma(\gamma-1)C}\,d\gamma = \frac{\sigma N}{C}[e^{\sigma(\beta-1)C} - e^{\sigma(\alpha-1)C}]$$

As commented earlier, a qualified peer can actually be elected as super peer only when there are enough peers recommending it as super peer. Suppose that a peer u's random walkers visit K peers. If L out of the K peers have also been visited by a random walker of a higher rank peer before u's random walkers visit them, then u will only collect $K-L$ votes for recommending it as super peer. Clearly, the higher the rank of u, the more votes u is likely to collect, and so the higher probability u can be elected as super peer. Let $p_{[\alpha,\beta]}$ be the probability that a qualified peer ranked in $[\alpha,\beta]$ is elected as super peer. By the above discussion, $p_{[\alpha_1,\beta_1]} \le p_{[\alpha_2,\beta_2]}$ if $\beta_1 \le \alpha_2$.

Then, the number of peers ranked in $[\alpha,\beta]$ that can be elected as super peer is $p_{[\alpha,\beta]} \cdot g(\alpha,\beta)$. So, given any β, the number of elected super peers that have rank below β is $p_{[0,\beta]} \cdot g(0,\beta)$. Therefore, the percentage of peers that have rank below β is

$$\lambda_{elected} = \frac{p_{[0,\beta]} \cdot g(0,\beta)}{p_{[0,\beta]} \cdot g(0,\beta) + p_{[\beta,1]} \cdot g(\beta,1)} \le \frac{p_{[0,\beta]} \cdot g(0,\beta)}{p_{[0,\beta]} \cdot g(0,\beta) + p_{[0,\beta]} \cdot g(\beta,1)} = \frac{g(0,\beta)}{g(0,1)} = \frac{e^{\sigma(\beta-1)C} - e^{-\sigma C}}{1 - e^{-\sigma C}} \quad (2)$$

In addition to election, super peers can also be appointed by existing super peers. To calculate the quality of appointed super peers, observe first that an appointed super peer can rank below β only if all faction members of the appointer have rank below β before the appointment. So, in an oversized faction of size F_{max}, the probability that all peers other than the super peer have rank below β is $\beta^{F_{max}-1}$. So the probability that an appointed super peer has rank below β is $\beta^{F_{max}-1}$.

Let $h(\sigma)$ be the percentage of super peers that are elected with respect to σ. Then, taking both elected and appointed super peers into account, the percentage of super peers that have rank below β is at most

$$\lambda \le h(\sigma) \cdot \lambda_{elected} + (1 - h(\sigma)) \cdot \beta^{F_{max}-1} \le \max\left\{\frac{e^{(\beta-1)\sigma C} - e^{-\sigma C}}{1 - e^{-\sigma C}}, \beta^{F_{max}-1}\right\}. \quad \square \quad (3)$$

Note that the quality of super peers is independent of the total number of peers in the system. From Eq. (3) it is easy to see that the quality of appointed super peers is very good, as it is determined by $\beta^{F_{max}-1}$. For example, when $\beta = 0.9$ and $F_{max} = 45$, $\beta^{F_{max}-1}$ is less than 1%. To see the quality of elected super peers, Fig. 12 shows the percentage of elected super peers that have rank below a given β. We set $\sigma = 1$, and draw five lines with different coverage C. For example, when $C = 30$ and $\beta = 0.9$, there are at most 5% of super peers that could be ranked less than 0.9, and almost none of them could be ranked below 0.82. That is, 95% of the elected super peers are ranked at the top 10% among all peers in the system, while the rest are ranked no less than top 18%. This is quite appealing as no global knowledge is used in electing super peers.

4.2. Connectedness of super peers

In this section we analyze the connectedness of the overlay, as the main purpose of the Envoy construction is to connect peers to form a single structured overlay.

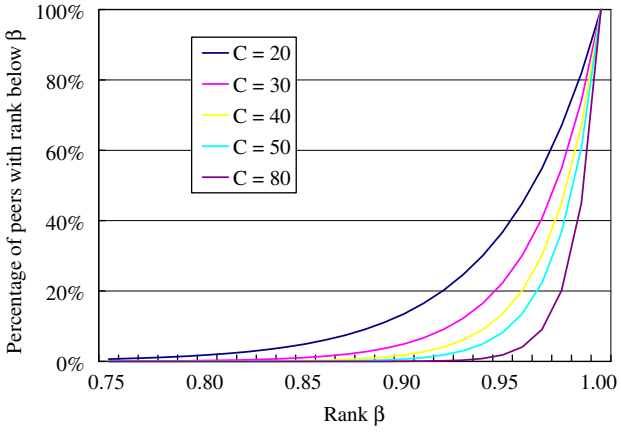

Fig. 12. Qualification of elected super peers.

Observe that initially every peer is free, meaning that it has not connected to any other peer. The construction proceeds by letting peers discover one another to form a faction, an alliance, and a union through the following two types of procedures:

- *Faction discovery*: The procedure is repeatedly executed by a free peer until it has joined a recommendation heap or a faction.
- *Group discovery*: The procedure is repeatedly executed by an active super peer until it becomes inactive.

Although the discovery procedures vary according to a peer's state, the goal is the same: to merge components into a larger one. Thus, we can model the construction as a sequence of state transitions

$$S_0 \to S_1 \to S_2 \cdots$$

where the initial state consists of only singleton components representing each individual peer in the system, and each transition $S_i \to S_{i+1}$ is driven by some discovery procedures to merge at least two components in the previous state. For simplicity, we shall assume that peers do not fail during the construction.

Since discovery procedures are implemented by random walks, connections between peers are established in a randomized manner. So the construction of Envoy resembles some random graph construction. In graph theory, a *random graph* $G(n,p)$ is a graph constructed from n vertices such that the probability that a link exists between two vertices is p (where the probability p can be a function depending on n). So the construction of Envoy resembles some random graph construction. The following lemma (Chung and Graham, 1998) will be used in this section:

Lemma 4.2. *Let $G(n,p)$ be a random graph. If $p > \ln n/n$, then* whp[4] *$G(n,p)$ is connected.*

Lemma 4.3. *Suppose in state S_{n_0} there is a set A of $\gamma\frac{N}{\overline{F}}$ active super peers, where $\gamma \le 1$. If*

$$1 - \left[\left(1 - \frac{\overline{F}}{N}\right)^C\right]^2 \ge \frac{\ln \gamma\frac{N}{\overline{F}}}{\gamma\frac{N}{\overline{F}}} \tag{4}$$

then with high probability there is some state S_n such that all the super peers in A are connected.

Proof. Since each super peer in A is active, there is some state S_n by which every super peer in A has executed at least one instance of group discovery. Let u and v be any two super peers in A. If either one discovers the other, then u and v will be connected. So the probability that u and v will be connected after each one has performed one instance of group discovery is

$$1 - \left[\left(1 - \frac{\overline{F}}{N}\right)^C\right]^2$$

So the probability that every two super peers in A are connected in state S_n is $1 - [(1 - \overline{F}/N)^C]^2$. By Lemma 4.2, if

$$1 - \left[\left(1 - \frac{\overline{F}}{N}\right)^C\right]^2 \ge \frac{\ln \gamma\frac{N}{\overline{F}}}{\gamma\frac{N}{\overline{F}}}$$

then with high probability all super peers in A are connected in S_n. □

Note that $\ln n/n$ is a decreasing function of n for $n \ge e$. So if there is some r satisfying Eq. (4), then for every $r' > r$, r' also satisfies the equation. To illustrate the lemma, suppose $N = 10^6$, $C = 30$, and $\overline{F} = 30$. Then if there are $15\% \times N/\overline{F}$ active super peers, the super peers will be connected whp before they become inactive.

The above lemma says that if there are enough number of super peers simultaneously active, then they are guaranteed whp to form a connected component. Our ultimate goal, however, is to show that once there are some super peers, the construction guarantees whp that eventually all super peers, including late-elected ones, will be connected. To prove this property we need the following lemma.

Lemma 4.4. *Suppose in state S_{n_0} there are a set A of super peers, all are active, where $|A| = \gamma N/\overline{F}$ for some $0 < \gamma \le 1$. Then there exists some state reachable from S_{n_0} such that the expected number of active super peers in S_{n_0} grows to at least*

$$\frac{N}{\overline{F}}(\gamma + (1-\gamma)(1 - e^{-\gamma C/\overline{F}}))$$

while the expected number of inactive super peers is at most

$$\gamma\frac{N}{\overline{F}}\left(\left(1 + \frac{C}{\overline{F}}\right)\gamma\right)^C$$

Proof. Let B be the set of free peers in S_{n_0}. Since each faction has an average size $\overline{F}$, $|B| \simeq N - |A| \times \overline{F} \simeq (1-\gamma)N$. Since all the super peers in A are active, each one must perform at least one group discovery procedure before it becomes inactive. So from S_{n_0} on at least $\gamma N/\overline{F}$ instances of group discovery are to be performed.

Moreover, observe that super peers can be generated in two ways:

- Election among a set of free peers (via the execution of faction discovery).
- Split of a large faction (via faction or group discovery).

So the free super peers in B may generate some super peers, which in turn may also perform some instances of group discovery. So we can assume that from S_{n_0} on there is a sequence of state transitions $S_{n_0} \to S_1 \to \cdots S_n$ driven by a set D of at least $\gamma N/\overline{F}$ instances of group discovery, together with some, possibly none, instances of faction discovery. Let E be the set of new super peers elected by free peers in B in between state S_{n_0} and S_n. Note that each instance in D may be performed by super peers in A, as well as by super peers in E.

As noted above, in addition to the set E of super peers elected by free peers in B, the split of large factions may also generate new super peers. A faction can grow in size when the super peer of the faction captures some free peers while executing group discovery, or when some free peer discovers the faction in faction discovery. Consider just the first case, in which new super peers are generated by existing super peers that have actively participated in the construction. There are $|B| - |E| \times \overline{F}$ free peers in B that could be captured while the discovery instances in D are performed. The probability that a free peer in B is captured by any of the instances is

$$1 - \left(1 - \frac{1}{N}\right)^{C\gamma N/\overline{F}}$$

So the expected number of free peers that could be captured is

$$((1-\gamma)N - |E| \times \overline{F}) \times \left(1 - \left(1 - \frac{1}{N}\right)^{C\gamma N/\overline{F}}\right)$$

[4] A graph has some property Q *with high probability*, abbreviated as *whp*, if the probability it has Q tends to one as n tends to infinity.

On average, every $\overline{F}$ captured free peers will generate one super peer. So at least

$$\frac{1}{\overline{F}}((1-\gamma)N-|E|\times\overline{F})\times\left(1-\left(1-\frac{1}{N}\right)^{C\gamma N/\overline{F}}\right)$$

new super peers may be generated due to the capture of the free peers by the discovery procedures in D.

Together with the $|E|$ super peers elected by free peers in B, the total number of super peers generated in state S_n is

$$\begin{aligned}&|E|+\frac{1}{\overline{F}}((1-\gamma)N-|E|\times\overline{F})\times\left(1-\left(1-\frac{1}{N}\right)^{C\gamma N/\overline{F}}\right)\\&=|E|+\left((1-\gamma)\frac{N}{\overline{F}}-|E|\right)\times\left(1-\left(1-\frac{1}{N}\right)^{C\gamma N/\overline{F}}\right)\\&\geq(1-\gamma)\frac{N}{\overline{F}}\times\left(1-\left(1-\frac{1}{N}\right)^{C\gamma N/\overline{F}}\right)\\&\simeq\frac{N}{\overline{F}}(1-\gamma)\left(1-e^{-\gamma C/\overline{F}}\right)\end{aligned}$$

Taking A into account, the total number of super peers in state S_n is at least

$$\frac{N}{\overline{F}}(\gamma+(1-\gamma)(1-e^{-\gamma C/\overline{F}}))$$

Meanwhile, among these super peers some may become inactive if it cannot discover any peer outside its component while performing group discovery during the transition from S_{n_0} to S_n. Assume first that prior to S_n the largest group that was ever created contains no more than $(1+C/\overline{F})\gamma N/\overline{F}$ super peers. Moreover, without loss of generality assume that S_n is the first state from S_{n_0} during which at least $\gamma N/\overline{F}$ instances of group discovery have been performed. A super peer becomes inactive if it cannot find new group; that is, its random walkers travel only within its group in a discovery process. Since the largest group prior to S_n is of size at most $(1+C/\overline{F})\gamma N$, starting from S_{n_0} the number of super peers that may become inactive in S_n is at most

$$\gamma\frac{N}{\overline{F}}\left(\frac{\left(1+\frac{C}{\overline{F}}\right)\gamma N}{N}\right)^C=\gamma\frac{N}{\overline{F}}\left(\left(1+\frac{C}{\overline{F}}\right)\gamma\right)^C$$

Next, consider the case that some discovery instance in D discovers a group of more than $(1+C/\overline{F})\gamma N/\overline{F}$ super peers. Let S_k, $k<n$, be the first state in which some group U of more than $(1+C/\overline{F})\gamma N/\overline{F}$ super peers has been constructed. We note that

$$\left(1+\frac{C}{\overline{F}}\right)\gamma\geq(\gamma+(1-\gamma)(1-e^{-\gamma C/\overline{F}})),\quad\forall 0\leq\gamma\leq 1.$$

So in S_k there are at least $(N/\overline{F})(\gamma+(1-\gamma)(1-e^{-\gamma C/\overline{F}}))$ super peers, and the largest group that may be discovered prior to S_k contains no more than $(1+C/\overline{F})\gamma N/\overline{F}$ super peers. Then, by the previous argument, in S_k at most $\gamma(N/\overline{F})((1+C/\overline{F})\gamma)^C$ super peers may become inactive. So the lemma is proven. □

Lemma 4.4 says that if there are some active super peers, then the group discovery procedures executed by them guarantee to increase the number of active super peers to a certain amount, while keeping the number of inactive super peers bounded by some value. Let us consider the number of inactive super peers relative to the number of active super peers added:

$$\varepsilon=\frac{\gamma\left(\left(1+\frac{C}{\overline{F}}\right)\gamma\right)^C}{(1-\gamma)(1-e^{-\gamma C/\overline{F}})}\leq\frac{\left(\left(1+\frac{C}{\overline{F}}\right)\gamma\right)^C}{\frac{C}{\overline{F}}(1-\gamma)\left(1-\frac{1}{2}\gamma\frac{C}{\overline{F}}\right)}\tag{5}$$

Note that ε can be made very small by an appropriate setting of C, $\overline{F}$, and γ. For example, $\varepsilon<10^{-5}$ when $C=\overline{F}=30$ and $\gamma=\frac{1}{3}$. Moreover, the smaller the γ, the smaller the ε. In the above example, when $\gamma=\frac{1}{4}$, $\varepsilon\leq 1.4\times10^{-9}$. In other words, by Lemma 4.4 we see that when there are some active super peers, their existence guarantees the number of active super peers to grow to a certain amount before we need to worry about inactive super peers.

Lemma 4.5. *Suppose C and $\overline{F}$ are such that there exists some Γ satisfying the following conditions*:

1. $\dfrac{\Gamma\left(\left(1+\frac{C}{\overline{F}}\right)\Gamma\right)^C}{(1-\Gamma)(1-e^{-\Gamma C/\overline{F}})}\ll 1$
2. $1-\left[\left(1-\dfrac{\overline{F}}{N}\right)^C\right]^2\geq\dfrac{\ln\Gamma\frac{N}{\overline{F}}}{\Gamma\frac{N}{\overline{F}}}$
3. $(1-\Gamma)^C\simeq 0$

Then if in some state S_{n_0} there are some super peers and all are active, then whp *eventually there is a state S_n such that after the state all super peers in the system are connected.*

Proof. Let $\gamma_0 N/\overline{F}$ be the number of super peers in state S_{n_0}. Consider first $\gamma_0<\Gamma$. By Lemma 4.4, there is some state S_{n_1} such that the super peers in S_{n_0} grows to at least

$$\frac{N}{\overline{F}}(\gamma_0+(1-\gamma_0)(1-e^{-\gamma_0 C/\overline{F}}))$$

while the number of inactive super peers is at most

$$\gamma_0\frac{N}{\overline{F}}\left(\left(1+\frac{C}{\overline{F}}\right)\gamma_0\right)^C$$

By Condition 1 of the lemma, we may neglect the number of super peers that may become inactive in the transition from S_{n_0} to S_{n_1}. Let $\gamma_1 N/\overline{F}$ be the number of (active) super peers in S_{n_1}. If $\gamma_1\geq\Gamma$, then by Condition 2 and Lemma 4.3, all the super peers in S_{n_1} will be connected after each has performed one discovery procedure. Otherwise, Lemma 4.4 can be applied again to infer that there is a reachable state S_{n_2} in which the number of active super peers is

$$\gamma_2\frac{N}{\overline{F}}\geq\frac{N}{\overline{F}}(\gamma_1+(1-\gamma_1)(1-e^{-\gamma_1 C/\overline{F}}))$$

while the number of inactive super peers is negligible.

By repeatedly applying the above argument, we see that with high probability there will be a state S_{n_k} such that there are at least $\Gamma N/\overline{F}$ super peers in S_{n_k} and all the super peers are connected.

In the above we considered the case $\gamma_0<\Gamma$. If $\gamma_0\geq\Gamma$, then by Condition 2 and Lemma 4.3 there is some state S_{n_1} such that all super peers in the state are connected.

We have shown that there is some state S_{n_k} such that there are at least $\Gamma N/\overline{F}$ super peers, and all of them are connected. For every free peer in S_{n_k}, when it is activated to join Envoy (by starting to execute faction discovery), the probability that it will discover the

connected component in S_{n_k} is at least

$$p = 1 - \left(1 - \frac{\Gamma \times \frac{N}{\overline{F}} \times \overline{F}}{N}\right)^C = 1 - (1-\Gamma)^C$$

By Condition 3, $p \simeq 1$. So the lemma is proven. □

5. Simulation results

In this section we evaluate Envoy via simulation. As the main point of the paper is on the self-organization of a hybrid P2P system, we focus on the performance evaluation of the construction. For search performance in an Envoy-like hybrid system, one may refer to Loo et al. (2004) for some preliminary results. Moreover, since to our knowledge this is the first such construction to appear in the literature, there is no algorithm for us to compare with. Still, we have designed our simulation so that the results presented in the paper can clearly deliver our message—the effectiveness and efficiency of our construction.

Our simulator is written in Java. It uses a single-process event-driven architecture to simulate concurrent activities of peers, such as join, discovering, and message delivery. The message delay follows exponential distribution with mean 30 time units.

In the simulation, we construct Envoy in two ways: one incrementally by new peers joining to the system, the other by building the Envoy structured overlay from an existing Gnutella-like unstructured network. For the latter, the network we generate has a power-law degree distribution with an average degree of $d = 12$ and up to 300,000 nodes. In both cases, each peer is assigned a random score in the interval $[0, 10,000]$. (Note that we are only concerned with the relative scores among peers; a particular score value does not have any significant meaning.) When Envoy is constructed from an existing unstructured network, a fraction of peers (called the *activation ratio*) in the network will be activated to begin the construction. An activation ratio of 1 thus means that all peers start the construction concurrently. Since most of the worst case performances occur in this scenario (as peers are more likely to create disjointed groups), it is our default ratio. There are also several other parameters in the simulation. Their default values are given in Table 1.

Table 1
Default simulation settings.

# of walkers	5
TTL for discovery messages	6
FactionSize	30 peers
MinUnionSize	30 super peers
Activation ratio	1.0
Discovery cycle	3000 time units
Contact cycle	3000 time units

Moreover, although our construction algorithm does handle churn, we do not model churn in our simulation. This is because the simulation focuses on the Envoy construction, in particular, how the overlays are constructed when peers concurrently enter the network. Allowing a peer to enter and leave the network during the construction stage complicates the simulation setting, but does not help much in understanding the performance of the construction algorithm. Churn would be more suitable for the study of search and routing cost in the overlays; see, e.g. Joung and Wang (2007).

Below we present the simulation results. We begin by evaluating the quality of the elected super peers. Then we evaluate the cost of the overlay construction. This is measured by the number of unions ever created, the size of temporary unions, and the number of union joins and group discoveries performed by a super peer. Finally, we evaluate the effectiveness of the construction by measuring the coverage rate of the elected super peers and the system convergence time.

5.1. Quality of super peers

The results for super peer quality are shown in Fig. 13, where in (a)–(c) the *x*-axis in the charts indicates the rank of super peers

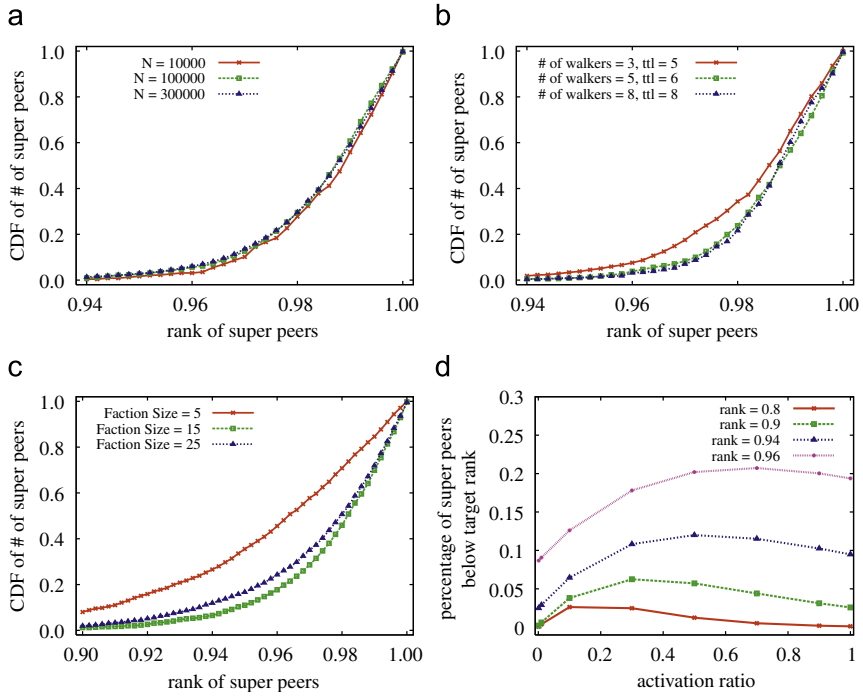

Fig. 13. Quality of super peers with respect to (a) network size, (b) search range, (c) *FactionSize*, and (d) activation ratio.

(with respect to their scores), while the *y*-axis shows the cumulative percentage of super peers having a given rank.

From (a) we see that network size has little affect to super peer quality, which is in concord with our theoretical analysis in Lemma 4.1. We also note that the quality of super peers is very good. For example, only less than 1.8% of super peers are ranked below 94%, and nearly none are ranked below 90%.

From (b) we see that large search range (determined by the number of walkers and their TTL) increases super peer quality. This is also expected from our analysis. Note, however, that the marginal effect for increasing search range is low as large search range consumes more network resources. From (c) we see that *FactionSize* also affects super peer quality: large *FactionSize* yields better quality. That is because a large number of super peers are appointed by existing super peers (due to faction split), rather than elected by ordinary peers. An appointed super peer is the peer with the highest score among the peers captured by its appointer. So its quality depends on the faction size. In the experiment, about 45% of the super peers are elected, and 55% are appointed when the activation ratio is 1.

In our analysis, the percentage of elected super peers increases with the activation ratio. Appointed super peers generally have higher quality than elected ones. So, with a relatively small activation ratio, the quality of super peers decreases as activation ratio increases. However, the percentage of elected super peers ranked below a certain value is a decreasing function of activation ratio. So when the activation ratio grows larger, the quality of super peers will increase again. This analysis is witnessed by our result shown in (d), in which the percentage of super peers ranked below certain value first increases, and then decreases with respect to the growth of activation ratio.

5.2. Number of unions ever created

Union mergence is costly and so Envoy must avoid constructing too many unions, as well as large "transient" unions that are eventually to be absorbed by other unions. In this section we first study the number of unions ever created in the construction. To obtain the data, for each experiment setting we repeat the experiment 30 times. We then count, for each $i > 0$, the number of times x_i in the 30 executions that have i unions been created during the execution. The value $x_i/30$ can then be interpreted as the probability that i unions will be created in an execution of the experiment. We call the distribution of $x_i/30$ the *probability density function* (PDF) of the number of unions created. Fig. 14 shows the results.

In (a) we study the scenario where Envoy is constructed incrementally and directly from new joining peers. We can expect

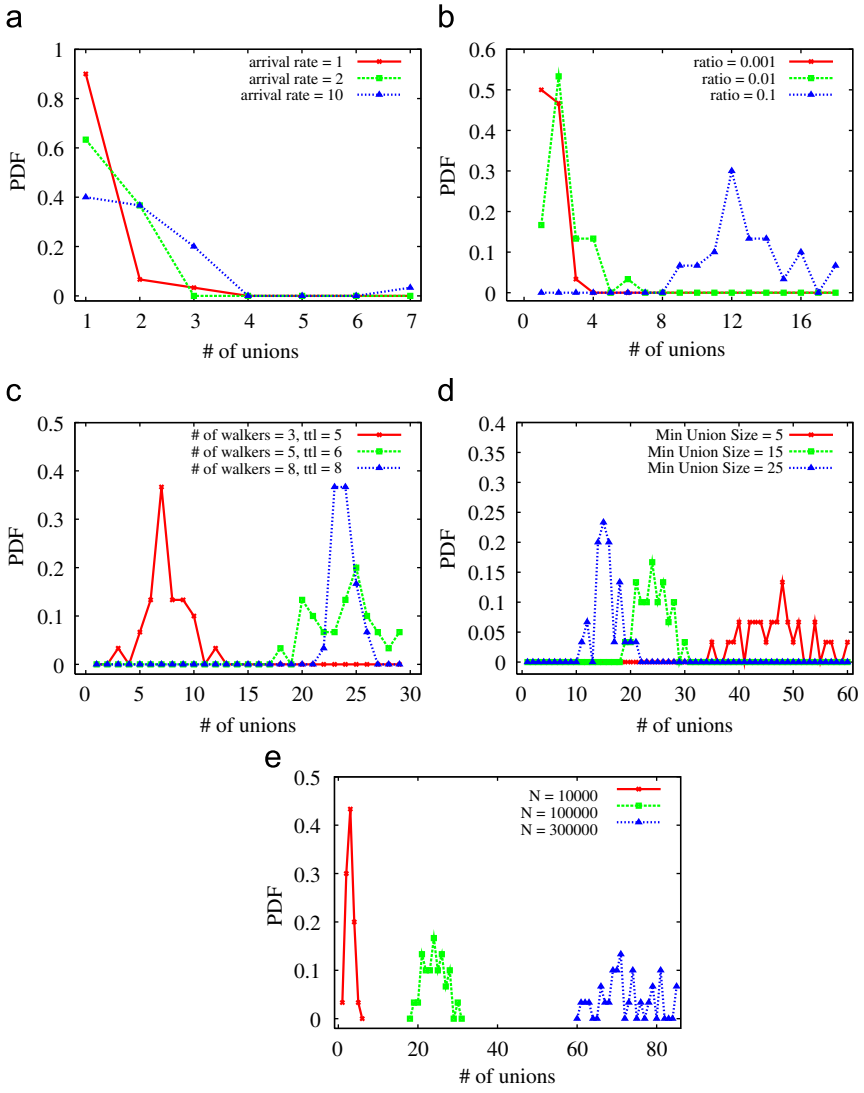

Fig. 14. The PDF of the number of unions ever created versus (a) arrival rate, (b) activation ratio, (c) search range, (d) *MinUnionSize*, and (e) network size.

that with low arrival rate, new peers have high chance to join an existing union instead of creating a new one. So only a very small number of unions (typically two or three) will be created. This is witnessed by the result.

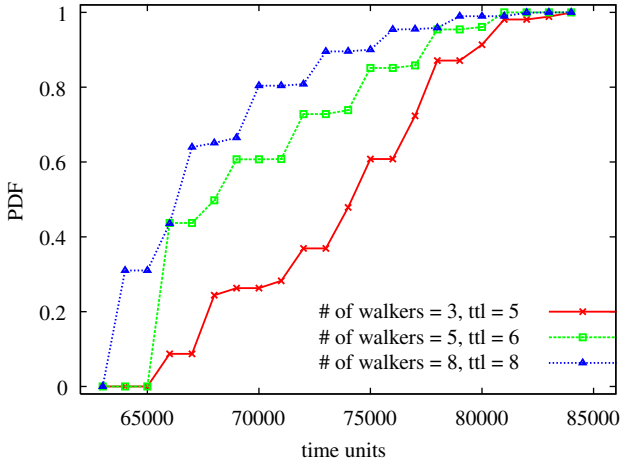

Fig. 15. The percentage of super peers created over time.

In (b) we study the scenario where Envoy is constructed from an existing unstructured network. We vary the activation ratio. Analogous to (a), when activation ratio is low, only a small number of super peers will be elected. The elected super peers will then create and enlarge their unions mainly by capturing free peers. So the number of unions ever created is also small. On the other hand, we can expect that the worst case occurs when activation ratio is 1, i.e., all free peers start simultaneously to discover factions. In the rest of the experiment we evaluate the worst case scenario, and vary other parameters.

In (c), we vary search range. Intuitively, large search range allows two groups to have a higher chance to meet each other, and so results in fewer unions. However, in (c) the smallest search range (of walkers = 3, TTL = 5) performs better than the others. To explain this, we draw the percentage of super peers created over time in Fig. 15. From this figure we see that with large search range (of walkers = 8, TTL = 8), super peers are elected rapidly because peers' scores are spread farther. So a large number of super peers emerge in a short period of time and, therefore, cause more unions to be created.

In (d) we vary *MinUnionSize*. The parameter controls when to upgrade an alliance to a union. So it is easy to see that large *MinUnionSize* causes fewer unions. In (e) we vary network size. It is also easy to see that large network size causes more unions, and

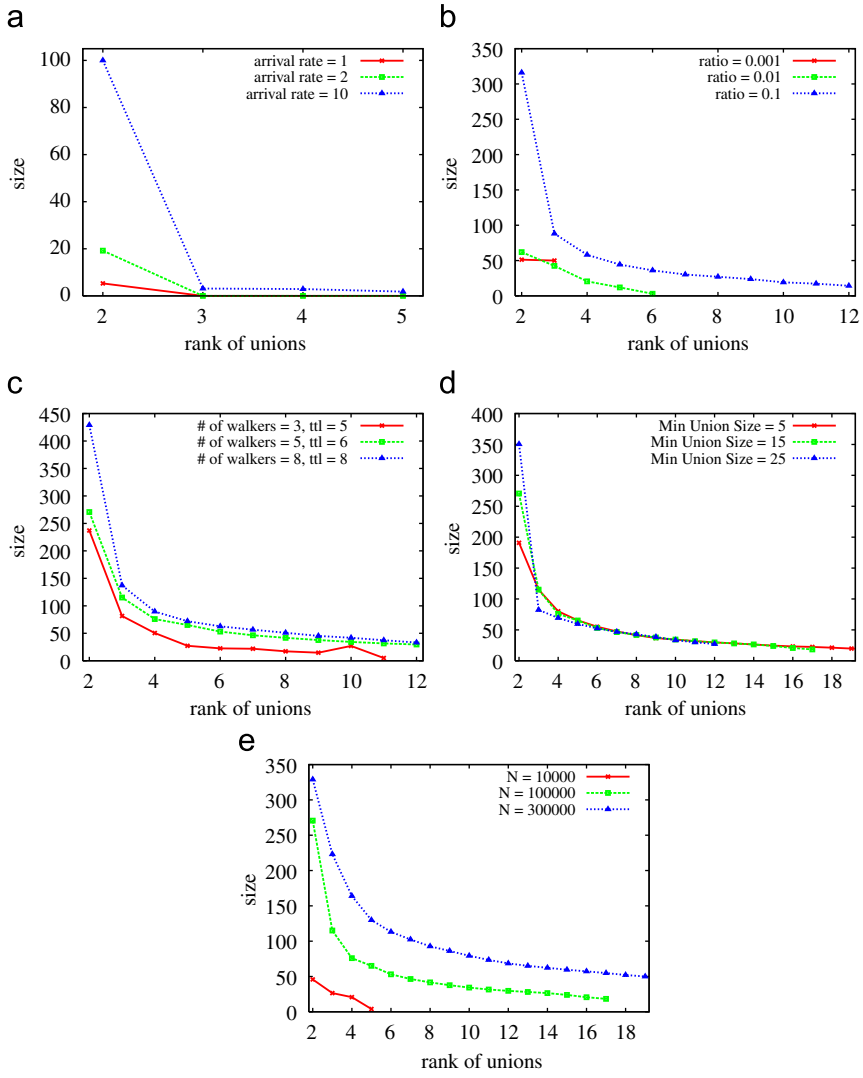

Fig. 16. The size of unions ever created versus (a) arrival rate, (b) activation ratio, (c) search range, (d) *MinUnionSize*, and (e) network size.

the growth rate is linear. Overall, we note that with a network of 300,000 nodes, only a small number (60–80) of unions have ever been created in the worst case scenario where peers start joining Envoy simultaneously.

5.3. Size of unions

The next experiment studies the size of the unions ever created during an Envoy construction. The simulation setting is similar to the previous one. Fig. 16 shows the results. They are generally in concord with what have been observed in Fig. 14. Specifically, in (a) we see that the size of the second largest union ever created decreases as the arrival rate decreases (the largest union is the one that survives after the construction). For example, with arrival rate 10, the largest transient union has size only around 100, and there are merely three remaining transient unions, each of which has size 2 or 3. The reason for transient unions being small both in number and in size is as before: with low arrival rate, new peers have very high chance to join an existing union instead of creating a new one.

Similarly, in (b) we see that higher activation ratio yields worse performance. In (c), (d), and (e), we study the union size in the worst case activation scenario (activation ratio = 1) with respect to different search range, *MinUnionSize*, and network size, respectively. Overall, even with a network of size 300,000, in the worst case the second largest union is around 330, while on average a transient union has size around 43. Together with the previous result that around 60–80 unions were ever created, we conclude that the cost of union mergence in Envoy is very low.

5.4. Number of union joins

The previous two experiments focus on the evolution of unions. The next two focus on the overhead of super peers. We first evaluate the number of times a super peer has performed a union join. Observe that each super peer must perform at least one union join. If it performs more than one join, then its union must be merged into another, and so some extra cost is paid for the super peer to leave its current union and to join the new one.

The results are shown in Fig. 17. In general, we see that a super peer typically joins a union no more than three times. On average, in the worst case where peers start joining Envoy simultaneously, each super peer performs only about 1.6 union joins, and the network size has little affect to this average (see (e)). This indicates that the Envoy construction is not only efficient but also scalable.

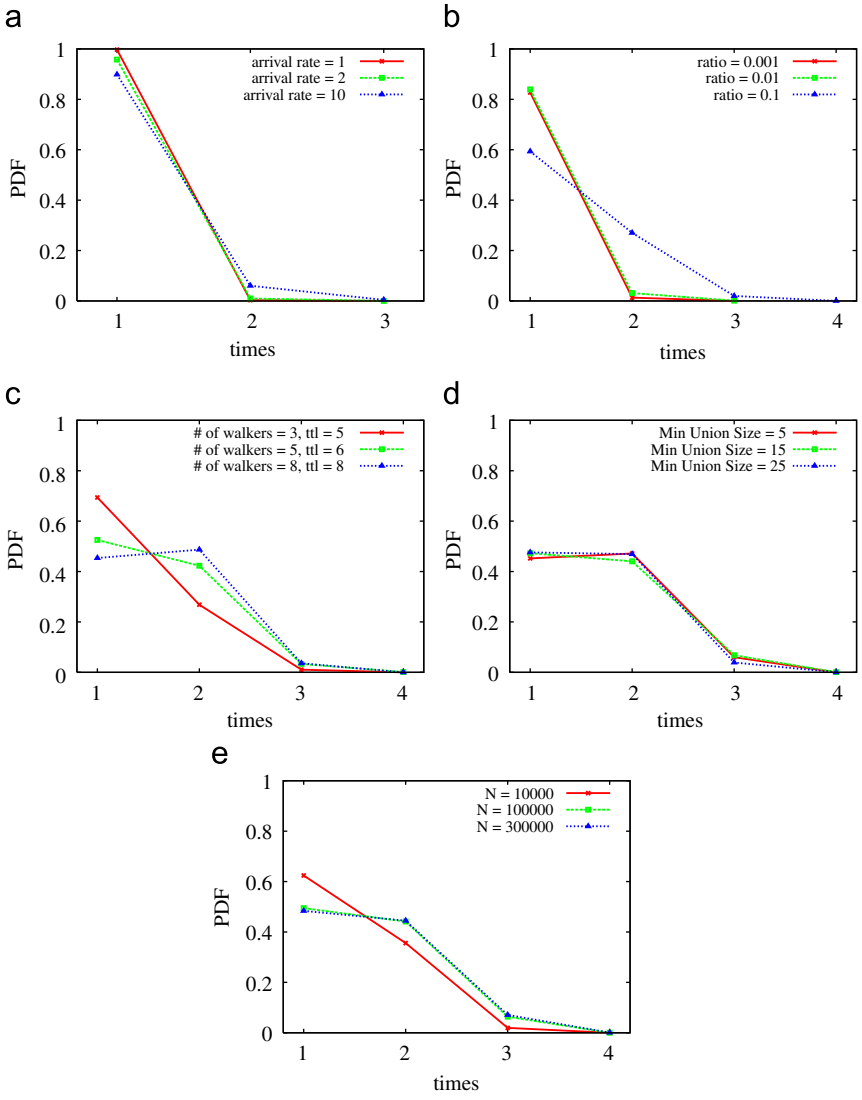

Fig. 17. The PDF of union joins per super peer versus (a) arrival rate, (b) activation ratio, (c) search range, (d) *MinUnionSize*, and (e) network size.

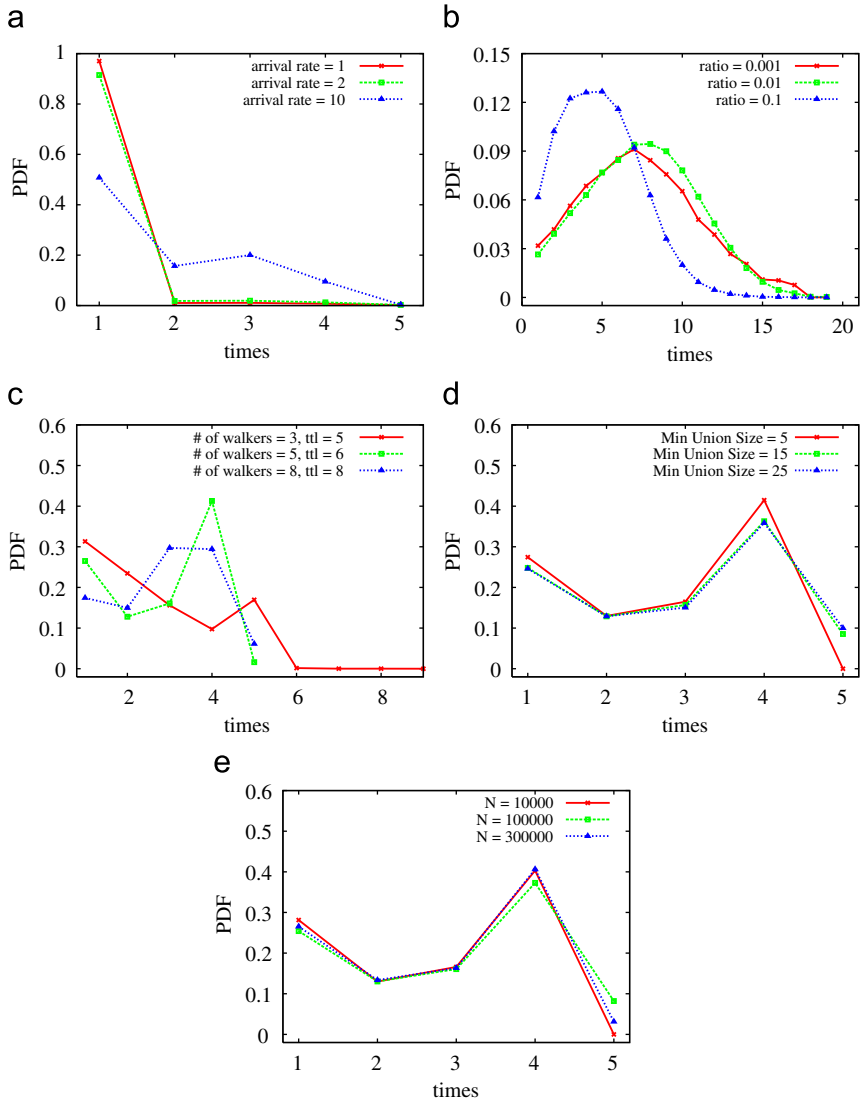

Fig. 18. The PDF of the number of group discoveries executed by each super peer versus (a) arrival rate (b) activation ratio (c) search range (d) *MinUnionSize*, and (e) network size.

5.5. Number of group discoveries

In this experiment we evaluate the number of group discoveries a super peer has performed. The results are shown in Fig. 18. In most cases, each super peer performs no more than five group discoveries. In (b), we see that with activation ratio low as 0.001, a super peer needs to perform around 10 group discoveries on average. This is because in this case there are lots of free peers. So an active super peer has a high chance to find a free peer in each invocation of group discovery, and thus continues to perform group discovery before it becomes inactive. Note that in this case the number of super peers is small. In (e) where activation ratio is 1, on average each super peer performs only less than three group discoveries in the worst case.

5.6. Coverage rate of active super peers

In the experiment we study the coverage rate of the Envoy structured overlay over an existing unstructured network when only a small subset of free peers start the Envoy construction. We wish to see that if other free peers remain free unless captured by active super peers, what percent of the peers (referred to as the *coverage rate*) in the unstructured network will be covered by Envoy. The results are shown in Fig. 19 for different network sizes and search ranges. We see that with only 0.01% of peers activated to start the construction, more than 94% of peers will be covered before all super peers become inactive, and the network size has little affect to the coverage rate. Note that for this coverage, it can be calculated that late activated peers are unlikely to form a separate structured overlay even if all existing super peers have become inactive. That is, the construction guarantees a single connected structured overlay in the end. In fact, for all experiments we have conducted, there is never a case where the structured overlay is partitioned.

5.7. System convergence time

In the final experiment we study the system convergence time. The results are shown in Fig. 20. We use coverage rate to measure system convergence, and draw the coverage rate over time. The time is measured in terms of the number of rounds of faction/group discoveries, where each round takes 3000 simulation time

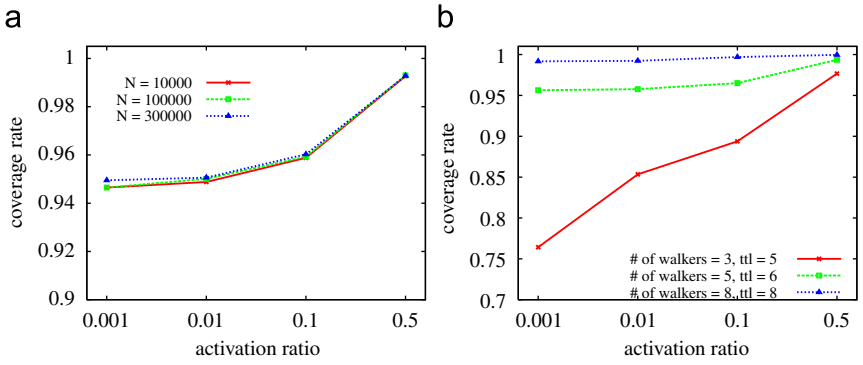

Fig. 19. Coverage rate versus activation ratio for (a) different network sizes, and (b) different search range.

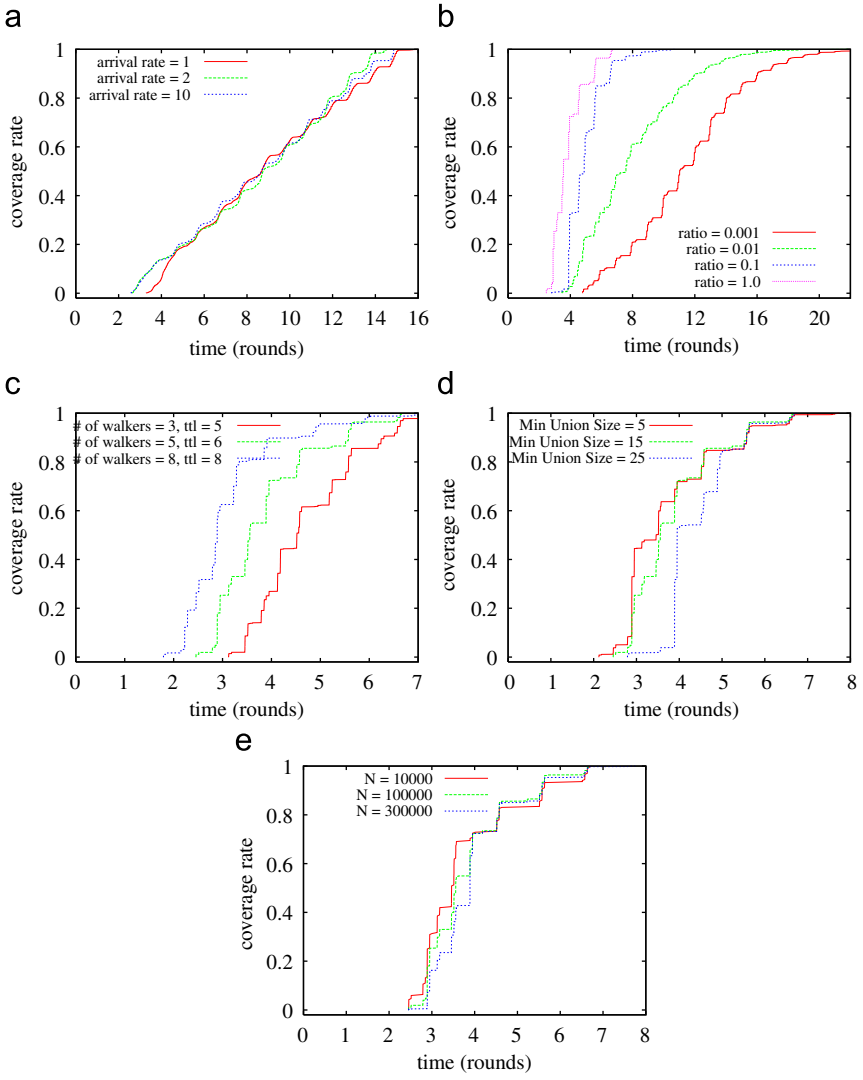

Fig. 20. System convergence time versus (a) arrival rate (b) activation ratio (c) search range (d) *MinUnionSize*, and (e) network size.

units. From the results we see that the Envoy construction is very fast. For example, for just only 0.001 activation ratio, the system takes about 22 rounds to establish the structured overlay over an unstructured network, and when all peers are activated concurrently, about seven rounds of discoveries are sufficient for the construction. The network size again has little affect to the construction time, and therefore the construction is highly scalable.

6. Conclusions and future work

We have presented Envoy, a two-layer P2P network where a structured overlay is built on top of an unstructured one. The purpose of using the two-layer architecture is to combine the advantage of each structure and create synergy. For example, it is known that unstructured P2P overlay is easy to build and maintain, and is quite effective in searching popular and nearby

objects as well as in handling complex search. Structured overlay, on the other hand, guarantees every search to be completed in bounded steps, typically in logarithmic of the network size. Therefore, by combining the two structures, both popular and rare/distant objects can be effectively and efficiently located. Moreover, a structured overlay is usually more difficult to maintain than an unstructured one, especially when system churn rate is high. Therefore, we may carefully select nodes from the unstructured overlay so that only stable nodes are used to build the structured overlay so as to reduce the maintenance cost.

Our focus was on designing an efficient and fully decentralized algorithm to construct the Envoy architecture. As we have shown, peers can work themselves out to select super peers such that only high quality peers will be elected as super peers from the unstructured overlay. Moreover, the super peers can also self-organize themselves to form the structured overlay without resorting to any centralized mechanism. The algorithm is also robust and able to cope with network dynamics where peers can come and go at will. We also provided some theoretical analyses of the algorithm, showing how good the elected super peers are, and, with high probability, the construction is likely to end up with only one single structured overlay, even though super peers may concurrently build the structured overlay and no centralized mechanism is used to assist them.

In addition, we have conducted a thorough simulation study for the Envoy construction. The simulation results confirmed that super peers are of very high quality. For example, only less than 1.8% of super peers are ranked below 94%, and nearly none are ranked below 90%. If a peer's quality is determined based on its stability, then the maintenance cost of the Envoy structured overlay can be significantly reduced due to the high stability of the super peers used to construct the overlay. For measuring peer stability, we have also developed a formula based on peers' session time, as conventional definitions such as MTBF (mean time between failure), MTTR (mean time to repair), MDT (mean down time), failure rate (1/MTBF), or availability (MTBF/(MTBF+MTTR)) are not adequate for the P2P environment.

The simulation results also showed how effective and efficient the construction algorithm is. For example, with just only 0.001 activation ratio, the system takes about 22 rounds of faction/group discoveries to establish the structured overlay over an unstructured network, and when all peers are activated concurrently, about seven rounds of discoveries are sufficient for the construction. In both cases, each super peer performs less than three group discoveries on average. Although the experiment was conducted over a network of 300,000 nodes, in general the network size has little impact to the construction cost, and so the construction is highly scalable.

Due to the distributed nature of the construction algorithm, super peers may concurrently build several disjointed overlays (called unions in our algorithm) during the construction, and then merge into a large one. A good design must ensure that the cost of union mergence is low. Our simulation indeed showed that even with a network of size 300,000, in the worst case around 60–80 unions were ever created, and the largest union to be merged is around 330, while on average a transient union has size around 43. All these results conclude that our construction is effective, efficient and highly scalable.

Finally, although the paper focuses on the construction of a hybrid unstructured and structured P2P system, the performance of the system itself, such as search efficiency and maintenance cost, deserves no less attention. Moreover, the performance of the system depends on a scoring function to rank peers as how qualified they are to serve as super peers. Super peer selection has several concerns. The stability measurement we proposed in Section 3.3 is just one of them. Empirical study is also needed to assess them. Thus our future work will pursue in these directions.

Acknowledgment

The authors would like to thank the anonymous referees of for their invaluable comments and suggestions.

References

Andersen D, Balakrishnan H, Kaashoek F, Morris R. Resilient overlay networks. In: Proceedings of the eighteenth ACM symposium on operating systems principles. ACM Press; 2001. p. 131–45.

Andrzejak A, Xu Z. Scalable efficient range queries for grid information services. In: Proceedings of second international conference on peer-to-peer computing. IEEE Computer Society; 2002. p. 33–40.

Chawathe Y, Ratnasamy S, Breslau L, Lanham N, Shenker S. Making Gnutella-like P2P systems scalable. In: Proceedings of the 2003 conference on applications technologies architectures and protocols for computer communications (SIGCOMM). ACM Press; 2003. p. 407–18.

Cheng A-H, Joung Y-J. Probabilistic file indexing and searching in unstructured peer-to-peer networks. Computer Networks 2006;50(1):106–27.

Chung FRK, Graham RL. Erdos on graphs: his legacy of unsolved problems. AK Peters, Ltd.; 1998.

Ganesan P, Gummadi K, Garcia-Molina H. Canon in G major: designing DHTs with hierarchical structure. In: Proceedings of the twenty-fourth international conference on distributed computing systems. IEEE Computer Society; 2004. p. 263–72.

Garcés-Erice L, Biersack EW, Ross KW, Felber PA, Urvoy-Keller G. Hierarchical P2P systems. In: Proceedings of the ninth European conference on parallel processing science (Euro-Par). Lecture notes in computer science, vol. 2790. Berlin: Springer; 2003. p. 1230–9.

Joung Y-J, Wang J-C. Chord^2: a two-layer chord for reducing maintenance overhead via heterogeneity. Computer Networks 2007;51(3):712–31.

Kermarrec A-M, Massoulié L, Ganesh AJ. Probabilistic reliable dissemination in large-scale systems. IEEE Transactions on Parallel and Distributed Systems 2003;14(3):248–58.

Kostoulas D, Psaltoulis D, Gupta I, Birman K, Demers A. Decentralized schemes for size estimation in large and dynamic groups. In: Proceeding of the 4th IEEE international symposium on network computing and applications (IEEE NCA), July 2005: p. 41–8.

Liben-Nowell D, Balakrishnan H, Karger DR. Analysis of the evolution of peer-to-peer systems. In: Proceedings of the twenty-first annual symposium on principles of distributed computing. ACM Press; 2002. p. 233–42.

Liben-Nowell D, Balakrishnan H, Karger DR. Observations on the dynamic evolution of peer-to-peer networks. In: Proceedings of the first international workshop on peer-to-peer systems. Lecture notes in computer science, vol. 2429. Berlin: Springer; 2002. p. 22–33.

Loo BT, Huebsch R, Stoica I, Hellerstein JM. The case for a hybrid P2P search infrastructure. In: Proceedings of the 3rd international workshop on peer-to-peer systems, February 2004: p. 141–50.

Lu EJ-L, Huang Y-F, Lu S-C. Ml-chord: a multi-layered p2p resource sharing model. Journal of Network and Computer Applications 2009;32(3):578–88.

Lv Q, Cao P, Cohen E, Li K, Shenker S. Search and replication in unstructured peer-to-peer networks. In: Proceedings of the sixteenth international conference on supercomputing. ACM Press; 2002. p. 84–95.

Malkhi D, Naor M, Ratajczak D. Viceroy: a scalable and dynamic emulation of the butterfly. In: Proceedings of the twenty-first annual symposium on principles of distributed computing. ACM Press; 2002. p. 183–92.

Maymounkov P, Mazières D. Kademlia: a peer-to-peer information system based on the XOR metric. In: Proceedings of the first international workshop on peer-to-peer systems. Lecture notes in computer science, vol. 2429. Berlin: Springer; 2002. p. 53–65.

Mizrak AT, Cheng Y, Kumar V, Savage S. Structured superpeers: leveraging heterogeneity to provide constant-time lookup. In: Proceedings of the third IEEE workshop on internet applications (WIAPP). IEEE Computer Society; 2003. p. 104–11.

Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A scalable content-addressable network. In: Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM). ACM Press; 2001. p. 161–72.

Reynolds P, Vahdat A. Efficient peer-to-peer keyword searching. In: Proceedings of the international middleware conference (middleware). Lecture notes in computer science, vol. 2672. Berlin: Springer; 2003. p. 21–40.

Saroiu S, Gummadi PK, Gribble SD. A measurement study of peer-to-peer file sharing systems. In: Proceedings of the 2002 multimedia computing and networking. The International Society of Optical Engineering; January 2002.

Singh A, Liu L. A hybrid topology architecture for P2P systems. In: Proceedings of the international conference on computer communications and networks. IEEE Computer Society; 2004. p. 475–80.

Stoica I, Morris R, Karger DR, Kaashoek MF, Balakrishnan H. Chord: a scalable peer-to-peer lookup service for Internet applications. In: Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM). ACM Press; 2001. p. 149–60.

Tian R, Xiong Y, Zhang Q, Li B, Zhao BY, Li X. Hybrid overlay structure based on random walks. In: Proceedings of the fourth international workshop on peer-to-peer systems. Berlin: Springer; 2005. p. 152–62.

Xu Z, Mahalingam M, Karlsson M. Turning heterogeneity into an advantage in overlay routing. In: Proceedings of twenty-second annual joint conference of the IEEE computer and communications societies (INFOCOM), March 30–April 3, 2003. p. 1499–509.

Yang B, Garcia-Molina H. Improving search in peer-to-peer systems. In: Proceedings of the twenty-second international conference on distributed computing systems. IEEE Computer Society; 2002. p. 5–14.

Zhang XY, Zhang Q, Zhang Z, Song G, Zhu W. A construction of locality-aware overlay network: mOverlay and its performance. IEEE Journal on Selected Areas in Communications 2004;22(1):18–28.

Zhao BY, Duan Y, Huang L, Joseph AD, Kubiatowicz JD. Brocade: landmark routing on overlay networks. In: Proceedings of the first international workshop on peer-to-peer systems. Lecture notes in computer science, vol. 2429. Berlin: Springer; 2002. p. 34–44.