Self-Organizable P2P Document Search Engine for Knowledge Management

Kazuhiro Kojima

AIST, 1-2-1 Namiki, Tsukuba, Ibaraki, Japan
k.kojima@aist.go.jp

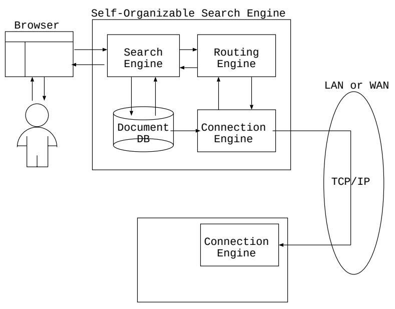
Abstract. Locating contents is an essential function, but it presents a very difficult and challenging problem for large-scale Peer-to-Peer (P2P) systems. Many P2P systems, protocols, architectures, and search strategies are proposed for this problem. In this paper, we focus on the self-organization of a community structure based on user preferences for P2P systems. We propose these methods to improve P2P search performance: 1) Extended Pong, 2) Pong Proxy, 3) QRP with Firework 4) Backward Learning, and 5) Community Self-Organization Algorithm. We evaluate the performance of the self-organized community network through simulations. These results show that the self-organized community network maintains a high query hit rate without overflow. As an example of implementation, we propose the Self-Organizable P2P Search Engine.

1 Introduction

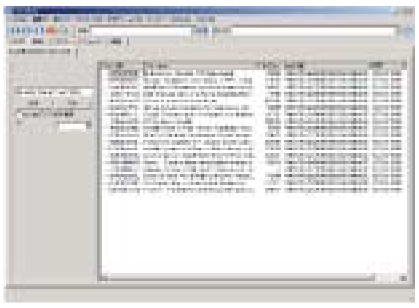
Peer-to-Peer (P2P) protocols and applications such as Gnutella[1] and Freenet[2] extend the power of Internet users. The current major P2P application is sharing of illegal contents. In the future, P2P systems will be applied to distributed search engines, knowledge management, and myriad Internet applications. However, P2P systems present many problems, one of which is *Locating Contents*.

In Gnutella protocol v0.4, a flooding algorithm[1] is used to locate contents. Peer sends a Query tagged with a maximum Time-To-Live (TTL) to its all neighbors on the overlay network. Each peer forwards the Query to its neighbors. When the Query string partially or exactly matches a file name stored locally, the peer replies with the QueryHit. Though this algorithm is very simple and robust even when peers are joining or leaving the system, it lacks scalability. A system comprising a large number of peers is inundated with Queries even if peers do not forward Queries that they have forwarded previously.

This paper proposes an approach to locating contents, a self-organization of communities based on the user's preferences. It is very natural to introduce a structure of community into P2P systems. Communities in the real world and cyberspace are formed by people who have similar preferences. For example, they form mailing lists, chat rooms, and Bulletin Board Systems (BBSs). The members of each community share the common knowledges with each other. We propose a new P2P protocol: TellaGate that can self-organize communities and Self-Organizable P2P Search Engine based on TellaGate.



(a) Architecture of Self-Organizable P2P Search Engine.



(b) ScreenShot of TellaGate Browser.

Fig. 1. Self-Organizable P2P Search Engine and TellaGate Browser. (a) The proposed Self-Organizable P2P Search Engine comprises 4 modules: i. Document DB, ii. Search Engine, iii. Routing Engine, and iv. Connection Engine.

The remainder of this paper proceeds as follows: Section 2 proposes a Self-Organizable P2P Search Engine; Section 3 gives a full detail of TellaGate protocol; Section 4 shows through simulations that a self-organized community network maintains a high query hit rate without overflow; and Section 5 concludes the discussion and proposes future work.

2 Self-Organizable P2P Search Engine

We show the architecture of the Self-Organizable P2P Search Engine in Fig. 1 (a). The proposed system comprises the following 4 modules.

i. Document DB
The shared documents are stored on the Document DB. Each document has a Globally Unique Identifier (GUID).

ii. Search Engine
The Search Engine provides a full text search service. Keywords are extracted from the shared documents using text mining techniques.

iii. Routing Engine
The Routing Engine resolves messages routing according to the Query Routing Protocol with Firework (Sec. 3.2).

iv. Connection Engine
The Connection Engine controls the TCP/IP connections with other peers according to the Community Self-Organization Algorithm (Sec. 3.3).

In Fig. 1 (a), user accesses to the system by browser as shown in Fig. 1 (b). The TCP connections between peers are reconfigured by TellaGate protocol, so that the community structures are self-organized on overlay networks.

To implement the Self-Organizable P2P Search Engine, we propose a new P2P protocol: TellaGate protocol. In the next section, we give a full detail of TellaGate protocol.

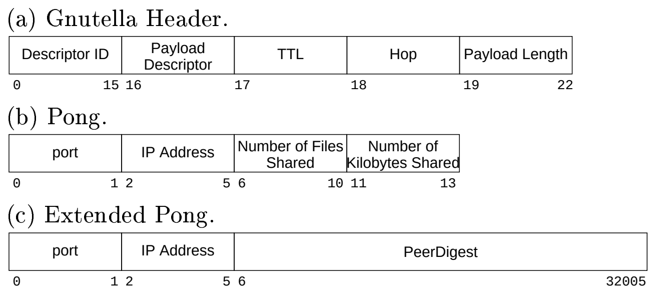


Fig. 2. Structure of message: (a), (b), and (c) are the Gnutella header, Pong, and proposed extended Pong, respectively. (a) In Ping, the Payload Length is 0. That is, Ping comprises only a header. (c) The proposed extended Pong includes a PeerDigest.

3 TellaGate Protocol

There are evident communities in some conventional client-server web applications such as mailing lists, chat rooms, and BBSs. These communities are maintained by centralized servers. On the other hand, decentralized and distributed P2P systems have no such servers. In a social network, a so-called acquaintance network[3], communities are frequently self-organized by local interactions. This study implements local interactions to a decentralized P2P system using 1) Ping-Pong and 2) Bloom filters.[4] Next, we briefly explain these key technologies.

3.1 Key Technologies

Ping-Pong In Gnutella protocol v0.4, a peer uses Ping to probe the network actively for other peers. Gnutella protocol messages comprise a header and payload. A Ping has only a header portion, as shown in Fig. 2(a). A peer receiving a Ping forwards it to its neighbors and responds with a Pong, which contains its own IP address and the Port number and the amount of data it is sharing on the Gnutella network. The payload part of a Pong message is shown in Fig. 2(b). The number of Pings increases exponentially through replication and forwarding, thereby flooding the network with Ping-Pongs.

Bloom Filter and QRP A Bloom filter[4] is a quick and space-efficient data structure for representing a set of N elements to support membership queries. An array of M bits and K independent hash functions is used for a Bloom filter. A Bloom filter may generate false positives. According to the analysis in [5], the false positive rate is given as

$$f = \left(1 - e^{-\frac{K}{r}}\right)^{K}, \qquad (1)$$

where r represents the number of bits per element, M/N. Some exemplary values of false positive rates are

$M/N = 6 \;\; K = 4 \; f = 0.0561$	$M/N = 8 \;\; K = 6 \;\; f = 0.0215$
$M/N = 12 \; K = 8 \; f = 0.00314$	$M/N = 16 \; K = 11 \; f = 0.000458$

We can choose a reasonable value of M/N and K by considering the expected false positive rate.

Rohrs proposed a Query Routing Protocol (QRP)[6] for Gnutella networks. In QRP, keywords are hashed and embedded in a Bloom filter. The Bloom filter of each peer is sent and forwarded to its neighbors so that the Bloom filter is propagated throughout the network. Queries are routed according to the QRP without a flooding algorithm.

3.2 Improvements

The above subsection briefly explained Ping-Pong, Bloom filters, and QRP. We extend these important elements to improve the P2P system performance.

Extended Pong We extend a Pong message so that we can use Bloom filters to self-organize communities. The Extended Pong (ExPong) includes a Bloom filter as shown in Fig. 2(c). In web cache systems[7], a Bloom filter is called a *Summary Cache*[8] or *Cache Digests*[9]. In this paper, a Bloom filter of the ExPong is called a *PeerDigest*. The PeerDigest of Peer i is represented as $digest_i$.

We restrict the shared contents within such document files having extensions as .html, .ps, and .pdf. Shared .pdf or .ps files are transformed into text files by pdftotext or ps2text. Keywords are extracted from these text files using text mining techniques.[10] It is considered that $digest_i$ shows the preference of the Peer i because these keywords are embedded into the PeerDigest.

PeerDigest Cache and Pong Proxy In Gnutella protocol v0.6[1], a hierarchical structure is defined as Ultra Peer and Leaf Peer. The Gnutella backbone is dominated by connections of Ultra Peers because Leaf Peer connects only to Ultra Peers. Ultra Peer serves as a proxy or server like Clip2 Reflector or Napster, thereby reducing redundant Pings and Queries.

In this paper, we propose a *Pong Proxy* instead of a hierarchical structure to reduce Ping-Pongs. Each peer has two *PeerDigest Caches*: $cache_l (l = 1, 2)$. Pairs $(address_j, digest_j)$ are cached into the $cache_l$, where $address_j$ is ip_j:$port_j$ of Peer j. The first and second neighbors of Peer i are represented as $nbr_{i,l} (l = 1, 2)$. The index l of $cache_l$ and $nbr_{i,l}$ indicates the degree of neighbors. If Peer j receives a Ping from Peer i, Peer j responds with an ExPong including $digest_j$. Moreover, Peer j responds with ExPongs including $digest_k (k \in nbr_{j,1})$. Peer i caches $digest_j$ and $digest_k$s into $cache_{i,1}$ and $cache_{i,2}$, respectively. In this manner, Peer j plays the role of Pong Proxy.

If each peer sends Pings to all first neighbor peers, as in Gnutella v0.4, the network load will increase as a result of the ExPongs because the ExPong

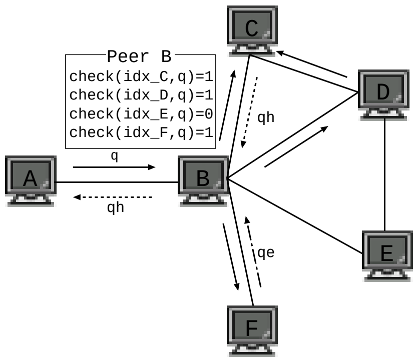


Fig. 3. QRP with Firework and Backward Learning. Peer A sends the Query to Peer B. Query routing is decided by $check(digest_j, q)(j \in nbr_{B,1})$ at Peer B. q, qh, and qe show the Query, QueryHit, and QueryError, respectively.

size is usually very large. We also improve the Ping-ExPong protocol in the following respect. Each peer sends Pings to $MaxCon$ peers that are selected randomly from $cache_1$. The peer responds with $MaxCon$ ExPongs that are selected randomly from $cache_1$. Hence, the upper bound for the total size of ExPongs that a peer will receive is given as

$$\begin{aligned} upper\ bound = MaxCon \times (1 + MaxCon) \\ \times Size_{\mathrm{ExPong}}. \end{aligned} \quad (2)$$

It is fair to add that the Gnutella peer also sends and forwards Pings to $MaxCon$ peers in simulations of Section 4.

QRP with Firework According to the Pong Proxy, each peer has information about the contents shared by its neighbors. We use the PeerDigests for query routing.

In Fig. 3, Peer A sends a Query: q, including search keywords, to Peer B. If Peer B has no content for the Query, Peer B must forward the Query. Routing of the Query is implemented as follows: Peer B checks whether keywords of q are in $digest_j(j \in nbr_{B,1})$ or not. It is given as

$$check(digest_j, q) = \{true, false\}. \quad (3)$$

If keywords exist in $digest_j(j \in nbr_{B,1})$, Peer B forwards the Query to Peer j. In Fig. 3, Peer B forwards the Query to Peers C, D, and F. If $check(digest_j, q)$ is false for all first neighbors, the Query is forwarded to Peer $j(\in nbr_1)$, which is selected randomly. This process is equivalent to a Random Walk Search.

Backward Learning In Fig. 3, Peer B forwards the Query to Peers C, D, and F. If Peer F has no content for the Query and is in *Dead Lock*, Peer F returns QueryError: qe to Peer B. According to the QueryError, Peer B modifies $digest_F$ as

$$digest_F^{(m)} = 0,\ m = h_k(q)\ (k = 1, \cdots, K),$$

where $digest^{(m)}$ is the m-th bit of $digest$ and h is a hash function. On the other hand, if Peer C has content sought by the Query, Peer C replies to Peer B with QueryHit: qh. The QueryHit: qh is backwarded to the Peer A by way of Peer B. According to the QueryHit, each mediate peer and originator routed by random walk strategy modifies $digest$ as

$$digest_j^{(m)} = 1,\ m = h_k(q)\ (k = 1, \cdots, K),$$

where $j(\in nbr_{i,1})$ shows the next or last mediate peer.

3.3 Community Self-Organization Algorithm

This subsection presents the Community Self-Organization Algorithm (CSOA). The above section mentioned that the PeerDigest shows the respective preferences of peers. Therefore, we can use PeerDigest to self-organize communities based on respective users' preferences.

We must define the similarity of the preference between Peer i and Peer j, sim_{ij}. That similarity is defined as

$$sim_{ij} \equiv count(digest_i \wedge digest_j), \tag{4}$$

where $\wedge$ is an AND operator and $count(\cdot)$ is a function that counts bits with 1.

Self-organization is implemented as follows. Firstly, Peer i calculates the similarity $sim_{i,j}$ between itself and the second neighboring Peers $j(\in nbr_{i,2})$. Secondly, Peer i selects the Peer j that has the largest value of $sim_{i,j}$. Thirdly, if the number of current connections Con_i is greater than the maximum connections $MaxCon_i$, then Peer i randomly selects Peer $k(\in nbr_{i,1})$ to disconnect. Finally, Peer i disconnects Peer k and connects to Peer j. The pseudo-code of this self-organization algorithm is shown in Fig. 4. This algorithm is very simple and requires only local information, that is, the PeerDigests of the first and second neighbor peers. Moreover, this local information is obtained by local interactions: Ping-ExPong and Pong Proxy.

4 Simulations

Simulator of complex systems such as Gnutella is not an easy task. We provided a simple setup for the following simulations.

```
procedure _Self-Organization (Peer $i$)

    $calculate\ sim_{ij}$ for all Peers $j \in nbr_{i,2}$
    $nbr_{i,2} = sort\ nbr_{i,2}$ according to $sim_{ij}$
    Peer $j$ =top of $nbr_{i,2}$
    $connect$ to Peer $j$
    $add$ Peer $j$ to $nbr_{i,1}$ and $cache_{i,1}$
    $remove$ Peer $j$ from $nbr_{i,2}$ and $cache_{i,2}$

    if( $Con_i > MaxCon_i$)
        Peer $k = select$ from $nbr_{i,1}$ at random
        $disconnect$ Peer $k$
        $remove$ Peer $k$ from $nbr_{i,1}$ and $cache_{i,1}$
```

Fig. 4. Pseudo code of the proposed Community Self-Organization Algorithm. In the following simulations of Section 4, $MaxCon = 4$.

- *Initial Topology*

  The initial topology of a network is given as a random network, as shown in Fig. 10(a). The network comprises 500 peers. Each peer has four random outgoing links; therefore $MaxCon_i = 4$. The number of peers is constant in the following simulations. We also simulated cases wherein the network is growing, but identical results to these were obtained.
- *Community*

  All peers are classified into five communities. Each peer is assigned randomly to one of these communities. There are no explicit community structures in the initial network topology as shown in Fig. 10(a).
- *Contents and Keywords*

  Each keyword is an integer value for simplicity.[2] The number of keywords, N, is 500. The keywords are divided to five classes: $KC_i (i = 0, \cdots, 4)$. Class KC_i comprises the inherent 100 keywords and 20 keywords which belong to the class $KC_{i+1} (KC_5 = KC_0)$. The content associated with each keyword is an integer equal to the keyword.
- *PeerDigest*

  The hash functions are built by first calculating the MD5 signature of the keyword string, which yields 128 bits, then dividing the 128 bits into four 32-bit unsigned integer values, then finally taking the modulus of each 32-bit unsigned integer value by the PeerDigest size M[8]. MD5 is a cryptographic message digest algorithm that hashes arbitrary length strings to 128 bits. The bits per element M/N is set to four in the following simulations; therefore, the theoretical false positive rate is 0.160.
- *Local Storage*

  In real P2P systems, most users have no shared local storage; they are called *Free Riders*[11]. However, in simulations, peers have the same local storage capacity for simplicity. We define the local storage capacity factor as

$$\alpha \equiv \frac{LSC}{NC}, \tag{5}$$

where LSC and NC are the local storage capacity and the number of contents assigned to a community, respectively. If $\alpha \geq 1$, each peer can store all contents shared by its own community on the local storage. If $\alpha < 1$, LRU cache replacement is used for stored content.

- *Uploader and Downloader*
  All local storage locations are initially empty. Contents are provided by specific peers called Upload-Only Members (UOMs). The remaining peers are Download-Only Members (DOMs). We assumed that the number of UOMs is 10% of all peers. In Fig. 10(a), UOM and DOM peers are shown as big and small circles, respectively.
- *Message Size and Buffer*
  The TTL of Ping and Query are set to $five$. We define the size of each message as shown in Table 1. According to the above assumption, $MaxCon_i = 4$, the upper bound eq.(2) is 0.64 M-byte. The message buffer, that is, the number of messages that Peer i can process in a single time step, is represented as mb_i. Each peer has the same size mb. If the number of messages received is greater than the size of mb, the overflowed messages are dropped.

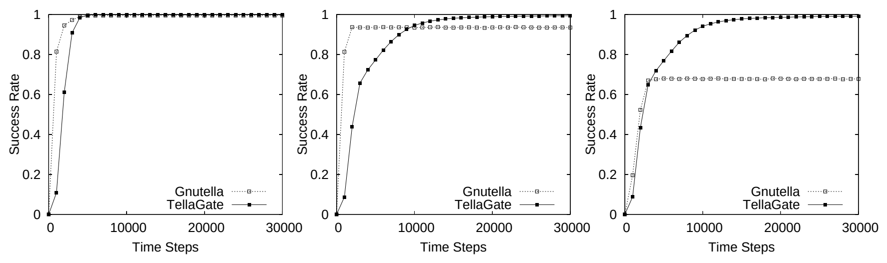
Message Type	Message Size [byte]
Ping	22
Pong	35
ExPong	32027
Query	50
QueryHit	100
QueryError	50

Table 1. Message size.

- *Time Step*
  Each peer generates a Query and Ping every 10 and 1000 time steps. CSOA is applied every 1000 time steps.

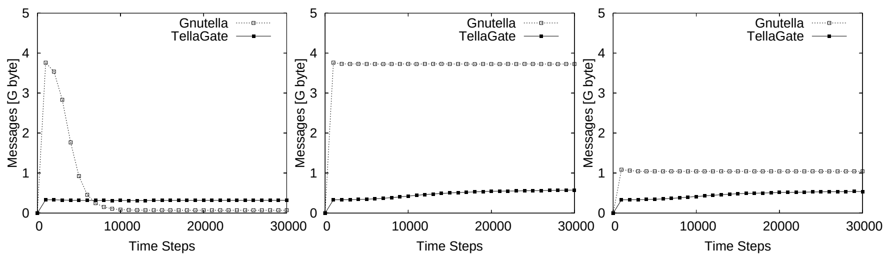
4.1 Performance

We compared the performance of TellaGate with Gnutella protocol v0.4. Respective performances were evaluated by the success rate of search, network load, average HOP of the successful Query, and the average number of QueryHit. We implemented several simulations under various parameter values, that is, the local storage capacity factor α $(0 < \alpha \leq 1.0)$ and the message buffer mb $(10 \leq mb \leq 100)$. Due to limitation space, this paper describes results of simulations under $\alpha = 1.0$ or 0.25 and $mb = 100$ or 20.



(a) $\alpha = 1.0$, $mb = 100$. (b) $\alpha = 0.25$, $mb = 100$. (c) $\alpha = 0.25$, $mb = 20$.

Fig. 5. Performance: success rate.



(a) $\alpha = 1.0$, $mb = 100$. (b) $\alpha = 0.25$, $mb = 100$. (c) $\alpha = 0.25$, $mb = 20$.

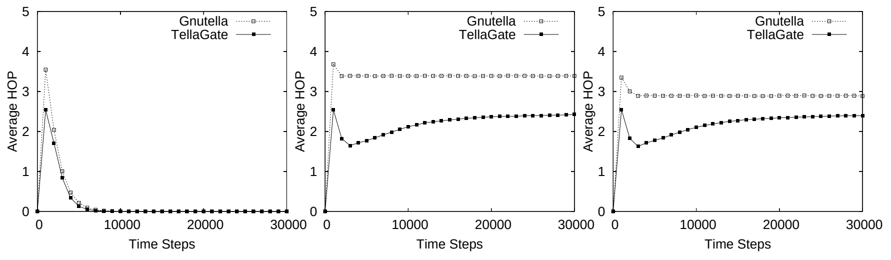
Fig. 6. Performance: network load.

Results represent an average of 10 simulations; they are shown in Figs. 5-8. In those figures, results of Gnutella v0.4 and TellaGate are shown by the broken line with open squares and the solid line with filled squares, respectively.

The success rates converge to 1.0 for Gnutella v0.4 and TellaGate with $\alpha = 1.0$, $mb = 100$, as shown in Fig. 5(a). Moreover, the average HOP converges to 0, as shown in Fig. 7(a) because each peer has local storage of sufficient capacity to store the contents and a message buffer that is sufficient to receive the messages. However, Gnutella v0.4 induces a flood of Ping-Pongs and Queries in an early time step as shown in Fig. 6(a). The network load of Gnutella v0.4 converges to the lower value after the peak. When the size of the local storage is limited, the network load of Gnutella v0.4 converges to the higher value, as shown in Fig. 6(b). When the size of the message buffer is also limited, as many Queries are dropped out; the average number of QueryHit decreases and the success rate converges to the lower value of 0.68, as shown in Fig. 8(c) and Fig. 5(c).

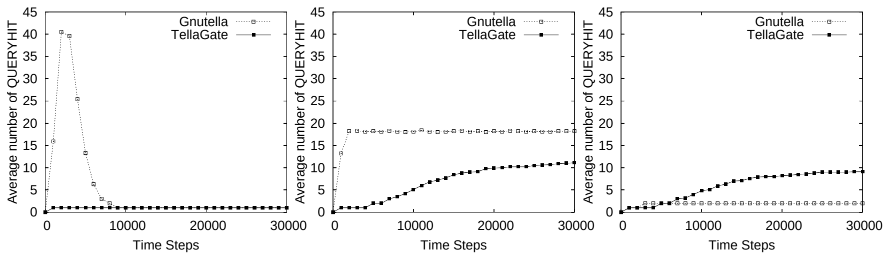
TellaGate does not engender a rapid increase of the network load. Network loads of TellaGate converge to the lower value, as shown in Fig. 6. In Fig. 5, even though the size of local storage and message buffer are limited, the success rate of TellaGate converges to 1.0.

We also simulated TellaGate without the CSOA to clarify the reason why the self-organized community network retains the high success rate value: the network retains the initial topology, as shown in Fig. 10(a) and the routing



(a) $\alpha = 1.0$, $mb = 100$. (b) $\alpha = 0.25$, $mb = 100$. (c) $\alpha = 0.25$, $mb = 20$.

Fig. 7. Performance: average value of HOP.



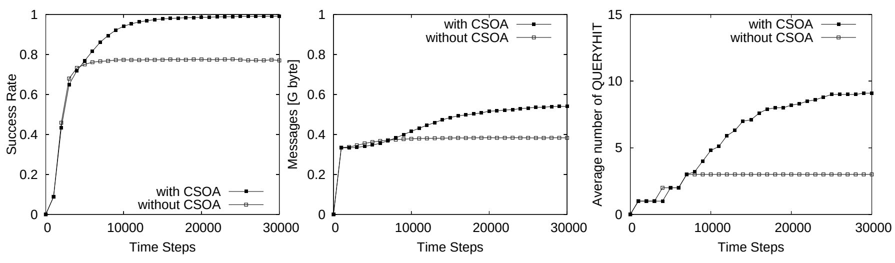
(a) $\alpha = 1.0$, $mb = 100$. (b) $\alpha = 0.25$, $mb = 100$. (c) $\alpha = 0.25$, $mb = 20$.

Fig. 8. Performance: average number of QueryHit.

of Queries is implemented by QRP with Firework. Results of the success rate, network load and average number of QueryHit are shown in Fig. 9. These three values of TellaGate without the CSOA are lower than the values of the original TellaGate.

According to these results, we inferred the following answer to the question above. For TellaGate without the CSOA, the probability that the first neighbors have the same preference is a low value; thereby, the routing strategy is equivalent to a Random Walk Search. If $\alpha \geq 1$, the search strategy is improved gradually to QRP with Firework by backward learning based on QueryError and QueryHit. However, if $\alpha < 1$, as the contents on the local storage are replaced by the LRU replacement policy, the PeerDigest will be inconsistent with the current contents shared by the distant peers. Many Queries are dropped out according to the TTL limitation.

On the other hand, in the self-organized community network shown in Fig. 10(b), each peer has several first neighbors that have identical preferences. As QRP with Firework replicates and forwards a Query based on eq.(3), the probability that the Query is replicated is higher than in the random network. Hence, the probability of locating contents will be a high value. This effect that is induced by the community structure and replication of Query is called the *Community Effect*.



(a) success rate. (b) network load. (c) average number of QueryHit.

Fig. 9. The effect of community structure: CSOA is the Community Self-Organization Algorithm.

5 Conclusion

This paper proposed the Self-Organizable P2P Document Search Engine and a new P2P protocol: TellaGate protocol. Simulation results show that the self-organized community network maintains a high query hit rate without overflow. However, we did not consider that peers often go offline in the above simulations. Therefore, future work will address simulations under such dynamic conditions.

The proposed Community Self-Organization Algorithm is based on local interactions. Recently, Davidsen and co-workers proposed a model of acquaintance networks.[19] In their model, the network is self-organized from an initial random network to a *Small World* network[17][18] characterized as having: 1) short path length, 2) high clustering[17], and 3) scale-free or exponential link distributions[18]. The proposed CSOA in this paper and Davidsen's algorithm are based on similar local interactions; indeed, the Self-Organized Community Network has short path length ($L \simeq 3.0$) and a high clustering ($C \simeq 0.14$) property. However, in Davidsen's model, not every node has an internal state such as PeerDigest. How do Small World properties influence the self-organization processes? Future works will address this question.

References

1. RFC-Gnutella. http://rfc-gnutella.sourceforge.net/
2. I.Clarke, O.Sandberg, B.Wiley, and T.W.Hong: Freenet: A distributed anonymous information storage and retrieval system. Proc. the OCSI Workshop on Design Issues in Anonymity and Unobservability (2000).
3. Stanley Milgram: The Small-World Problem. Psychology Today **1** (1967) 60–67.
4. B.Bloom: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13**(7) (1970) 422–426.
5. M.Mitzenmacher: Compressed Bloom Filters. Proc. of the 20th ACM Symposium on Principles of Distributed Computing (2001).
6. Christopher Rohrs: Query Routing for the Gnutella Network. http://rfc-gnutella. source-forge.net .
7. D.Wessels: Squid. http://squid-cache.org .

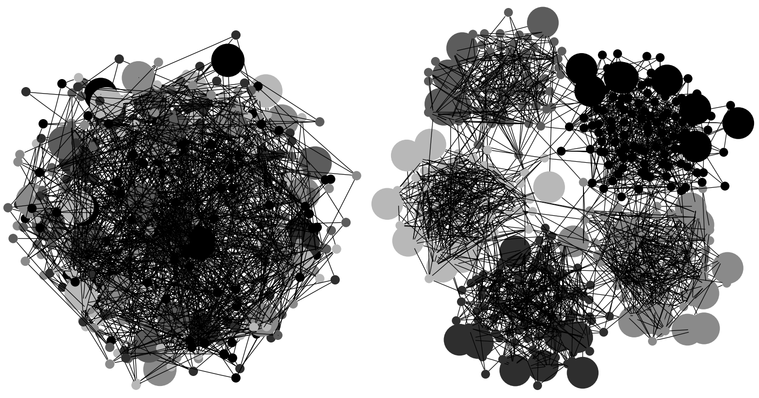

(a) Initial topology. (b) After self-organization.

Fig. 10. Network topology (a) Initial and (b) after self-organization (30 CSOAs were applied). Network topology is illustrated using the spring embedded model[12]. All peers are classified into five communities. The gray scale of each circle represents each community. Big and small circles represent UOMs and DOMs, respectively. In this figure, the number of UOMs is 50 peers and the number of DOMs is 450 peers.

8. L.Fan, P.Cao, J.Almeida, and A.Broder: Summary cache: A scalable wide-area Web cache sharing protocol. IEEE/ACM Transactions on Networking **8**(3) (2000) 281–293.
9. A.Rousskov and D.Wessels: Cache digests. Computer Networks and ISDN Systems **30**(22-23) (1998) 2155–2168.
10. Y.Matsuo and M.Ishizuka: Keyword Extraction from a Document using Word Co-occurrence Statistical Information (in Japanese). Trans. of the Japanese Society for Artificial Intelligence, **17**(3) (2002) 217–223.
11. E.Adar and B.A.Huberman: Free Riding on Gnutella. First Monday **5**(10) (2000).
12. Graphviz. http://www.research.att.com/sw/tools/graphviz .
13. B.Zhao, J.Kubuatowicz and A.Joseph: Tapestry: An Infrastructure for Wide-area fault-tolerant Location and Routing. University of California, Berkeley Computer Science Division, Technical Report, UCB/CSD-01-1141 (2001).
14. A.Rowstron and P.Druschel: Pastry: Scalable, decentralized object location and routing for a large-scale peer-to-peer system. Proc. of the 18th IFIP/ACM International Conference on Distributed System Platforms (2001).
15. I.Stoica, R.Morris, D.Karger, M.F.Kaashoek, and H.Balakrishnan: Chord: A Scalable Peer-to-peer Look-up Service for Internet Application. Proc. of the ACM SIGCOMM Conference (2001).
16. S.Ratnasamy, P.Francis, M.Handley, and R.Karp: A Scalable Content-Addressable Network. Proc. of the ACM SIGCOMM Conference (2001).
17. D.J.Watts: Small-Worlds. Princeton Univ. Press (1999).
18. R.Albert, H.Jeong, and A.-L.Barabási: Error and attack tolerance of complex networks. Nature **406** (2000) 378–381.
19. Jörn Davidsen, Holger Ebel, and Stefan Bornholdt: Emergence of a Small World form Local Interactions: Modeling Acquaintance Networks. Physical Review Letters **88**(12), 128701 (2002).