社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－DPS－127 (11)
2006／6／6

Standalone Overlay Reconfiguration Algorithm for Unstructured Peer-to-Peer Networks

Naohiro Hayashibara Makoto Takizawa

Department of Computers and Systems Engineering,
School of Science and Engineering,
Tokyo Denki University (TDU)
Ishizaka, Hatoyama, Saitama 350-0394, Japan
E-mail: {haya,taki}@takilab.k.dendai.ac.jp

Abstract

Recent years, peer-to-peer (P2P) overlay networks attract great interests because of its scalability and capability to share resources. However, unstructured P2P overlay networks (e.g., Gnutella) can cluster links. In such a situation, the performance of applications, such as flooding search, running on the network can be reduced because of the clustering. In the view point of reliability of the network, there exist some vulnerable points (i.e., hub peers) on crashing peers. It leads network partitions.

In this paper, we propose an algorithm of self-reconstructing unstructured peer-to-peer overlays into the desired topology with some constant degree D or $D+1$ in each peer. This algorithm does not require additional services (i.e., membership protocols, etc.). Moreover it does not divide the given overlay network during its execution.

1 Introduction

The development of peer-to-peer applications is one of the main stream in the area of networking. The growth of it is getting faster with the spreading of high speed network infrastructure. This movement motivate interest in the notion of overlay network separated from physical one.

On such a network, processes can connect to another process as they want without considering the physical topology and the routing information. Thus, applications can realize the robustness to node and link failures. Scalability is also one of the advantages of peer-to-peer overlay networks since there is no central control.

However, links sometimes cluster on one side of the network. It makes some hubs which mean that there exist performance bottleneck even in this type of network. Therefore it would reduce the performance of algorithms for overlay networks (e.g., flooding search).

There are several approaches [3–7] that constitute a desired form of overlay topology from arbitrary state. However they need some additional services such as group membership, address lookup, etc. These services help to modify the overlay structure though they cause overhead and make the algorithm complex.

In this paper, we propose an algorithm that each peer autonomously constructs the topology with the desired degree D or $D+1$ from arbitrary initial state. More precisely, the resulting topology is shown as an undirected connected graph G such that the difference between the maximum degree and the minimum degree is at most one. The proposed algorithm guarantees that it does not isolate the whole system during its execution. We call the graph with such a property *quasi-regular* graph. Moreover, this algorithm does not need any other service.

The topology modified by the proposed algorithm can keep connectivity after random failures on peers or links. It also have some advantages for boosting distributed algorithms, for example, search algorithms based on flooding such as Gnutella, CBF [8], etc. Suppose the large tree structure (or star structure), it might

take many hops to reach the leaf node that has the target object (e.g., files, multimedia contents) from another leaf node. Therefore, flooding search algorithms spend many TTLs in this case.

2 System Model

Distributed system consists of a set of peers $\Pi = \{p_1, p_2, ..., p_i\}$ which communicate only by sending and receiving messages. We assume that every pair of peers is connected by two unidirectional *quasi-reliable channel* [1], which holds the following conditions as the channel condition in the system.

- No Creation: If a process q receives a message m from process p, then p sends m to q.
- No Duplication: q receives no more than one message m from p.
- No Loss: If p and q are correct and p sends m to q, q eventually receiving m.

Quasi-reliable channels can be implemented over unreliable channels, using error detecting or correcting codes, sequence numbers and retransmission in cases of message loss. The TCP protocol is a good approximation to quasi-reliable channels. Thus, quasi-reliable channels ensure that messages are not lost in transit. We use

We consider that peers can only fail by crashing, and that crashed peers never recover. Every peer has a local failure detector that eventually detects all crashed peers (e.g., $\diamond\mathcal{P}$ defined in [2]).

2.1 Problem Description

The topology of the system is represented as a undirected graph $G = (P, L)$ consisting of a set of peers P and a set of links $L \subseteq P \times P$. In the graph theory, peers and links are simply replaced into vertexes and edges, respectively.

We assume each peer is assigned a unique identifier (e.g., IP address + port number). Since the proposed algorithm is fully decentralize, each peer executes independently. Thus every peer involved in the algorithm has the following restrictions: (i) each peer only knows its neighbors which are directly connected, (ii) peers can communicate only to their neighbors by message passing. Since the algorithm does not rely on membership protocols, the assumption is much rigor than other related works introduced above. This assumption means that it is impossible to connect again two or more sets of peers partitioned while the construction of overlay links is modified. Our approach has to take network partitions caused by modifying the overlay structure into consideration.

Links of the given topology is initially connected among peers arbitrary (i.e., unstructured overlay network) without no redundant link in any pair of peers. This assumption is not so strong because it is not difficult to remove such links. The algorithm aims to reconstruct the initial topology into the desired one that holds the following properties.

Definition 1 (Fairness). *In the given graph G, the difference of $\Delta(G)$ and $\delta(G)$ is at most one. Let $\Delta(G)$ be the maximum degree and $\delta(G)$ be the minimum degree among all peers in G.*

Definition 2 (Connectivity). *Any pair of peers in the graph G has at least one path (i.e., connected graph).*

Definition 3 (Non-redundancy). *Any pair of peers has at most one link.*

In this paper, the desired degree D is given as a parameter. Each peer could be converge its degree into D or $D + 1$ by connecting and disconnecting links.

The algorithm allows to join and leave peers during its execution. However, we assume time after which no such operation is occurred until the properties described above are satisfied, otherwise, the topology would gradually converge but the algorithm can not guarantee to hold Fairness property.

Although join/leave operations are often discussed with overlay formation algorithms, we do not discuss these operations in this paper. Because, we consider that the new topology is given and the algorithm starts again from the topology when such operations were occurred. Normally, these operations require a look-up service for the address of the peer or a partial view of the group membership. However, the algorithm does not require such services. Thus, we should discuss on it separated from join/leave operations though we assume them developed in the context of group communication protocols.

3 Reconfiguration Algorithm

In this section, we describe the detail of the proposed algorithm for reconfiguration of unstructured

overlays. The algorithm constitutes balanced topology from given arbitrary initial one. Any peer i maintains a set of one hop neighborhoods (i.e., directly connected ones) N_i. Since the given initial topology must be a connected graph, N_i must not be empty ($N_i \neq \emptyset$).

Moreover, this algorithm does not rely on additional protocols such as membership management protocols and look-up services for peer IDs. Every peer only knows its neighbors on the initial topology at the beginning. So, peers can only communicate with peer $j \in N_i$. Then, each peer requires its neighbors to have candidates for adding and removing links.

Thus, changing links would be done locally. It means that the resulting structure would not insure the property of the initial one. For example, if the initial overlay topology coincide with the physical structure, the overlay topology reconstructed by the algorithm could be close to the physical one.

Every peer executes the same procedure independently. The procedure contains two operations: (i) the *connect* procedure, which establishes a link, is called by peer i if the current degree of i is $|N_i| < D$ (ii) the *disconnect* procedure which removes a link of peer i if the current degree of i is $|N_i| > D$.

3.1 Connecting links

First, each peer checks its current degree (i.e., the number of neighbors) and then it calls the procedure for connecting links if the degree is less than desired degree D. Now we consider three peers: any peer i that wants to , one hop neighbor j of i (i.e., $j \in N_i$), two hops neighbor k of i (i.e., $k \in N_j$). Let d_j^i be a local variable that stores the degree of i at j. We also define the following message types that are used for connecting links.

- **GATHER_CANDIDATES**: a message to all neighborhoods for having their neighbors
- **PROPOSE_CANDIDATES**: a response to GATHER_CANDIDATES including own set of neighbors
- **CONNECT_REQUEST**: a request message for connecting a link
- **CONNECT_ACK**: an acknowledgement of CONNECT_REQUEST
- **UPDATE_DEGREE**: a message to notify all neighbors own degree that has been updated

Algorithm 1 shows the procedures for connecting a link. Suppose three peers i, j, k: peer i wants to connect a link, j is a neighbor of i and k is a neighbor of j. First, i collects sets of neighbors N_j of each i's neighbor j, then it makes a set of candidates $Cand_i$ for making a link (line 2-6). In the connection procedure, $Cand_i$ is constructed by the following way.

$$Cand_i = (\bigcup_{j \in N_i} N_j) \cap \overline{N_i}$$

Note that i can only communicate with its neighbors and connect a link to a two-hop neighboring peer of i. Thus, any peer $k \in Cand_i$ must be also a two-hop neighbor of i.

If $Cand_i$ is not an empty set, i selects a peer k with minimum degree among peers in $Cand_i$ and sends it a CONNECT_REQUEST message (line 7 and 8). The way of selecting an eligible link is one of the important factors to constitute the desired topology. There are some possibilities for selecting such a link and the corresponding peer, such as taking account of transmission delay, geometrical distance, etc. For instance, if you focus on the transmission time, i has to select a link of the peer that have low transmission delay. Although we choose the way that each peer selects a peer that have the minimum degree among its neighbors in order to make the algorithm simple, it is capable to take such factors into account.

We assume that k accepts any connection request in this algorithm if it is alive. Thus, k replies CONNECT_ACK to i and adds i into N_k when k received CONNECT_REQUEST message (line 15-17). Then, i adds k into N_i and sends UPDATE_DEGREE messages with i's degree $|N_i|$ to all neighbors (line 9-12). Finally, each peer in N_i updates i's degree which is the local value of the peer after receiving a UPDATE_DEGREE message (line 18 and 19).

We also show an example of the interaction in the link connection procedure in Fig. **??**. Now, we focus on the peer p_0 with the given topology as shown in the figure. In this topology, p_0 has two neighboring peers, p_1 and p_2 (i.e., $N_{p_0} = \{p_1, p_2\}$). Thus, p_0 sends GATHER_CANDIDATES messages to p_1 and p_2 and then gets $Cand_{p_0}$ including four candidates, p_3, p_4, p_5, p_6, to connect a link.

In Algorithm 1, p_0 would select the peer which has the minimum degree among candidates. If p_0 still have several choices after filtering with the minimum degree, p_0 has to select one of them in another way such

as the minimum ID number, similarity of IP address, hop count, etc. We now consider p_0 selects the peer which has the minimum ID number, thus p_3 is selected as the destination of new link. Therefore, p_0 sends a CONNECT_REQUEST message to p_3. After receiving the response of p_3, p_0 adds p_3 into N_{p_0} and sends UPDATE_DEGREE messages with its degree to all peers in N_{p_0}. Peer p_1, p_2, p_3 update their local variable $d_{p_1}^{p_0}$, $d_{p_2}^{p_0}$, $d_{p_3}^{p_0}$ respectively.

Peer p_0 periodically check the status of peer p_3 through the local failure detector when p_0 makes the link to p_3. If p_0 notifies that p_3 has crashed during the procedure, p_3 is removed from $Cand_{p_0}$ and then p_0 continues the procedure from line 7.

3.2 Disconnecting links

Peers must reduce their degree by disconnecting their links if their degree is more than $D + 2$. For removing links, each peer must consider to keep connectivity in whole structure. Therefore, it has to select the eligible link which is not a bridge. In this task, each peer also considers that its degree is not less than D.

The procedure for disconnecting links shown in Algo. 2 uses the following message types together with ones defined in Sect. 3.1.

- **DISCONN_REQUEST**: a request message for disconnecting a link
- **REPLACE_REQUEST**: a request message for replacing a link
- **DISCONN_ACK**: a positive acknowledgement of DISCONN_REQUEST or REPLACE_REQUEST
- **DISCONN_NACK**: a negative acknowledgement of DISCONN_REQUEST
- **DISCONN_CONFIRM**: a confirmation message for disconnecting the link that is going to removed

Peer i starts the disconnection procedure by executing the exactly same procedure (line 2-4) as the connection procedure (line 2-4) shown above. Because i also requires information on two-hop neighborhoods to have a set of eligible links and corresponding peers (i.e., $Cand_i$) that can be removed. In order to

Algorithm 1 Procedure for connecting a link

Initialization:

1: $Cand_i \leftarrow \{\emptyset\}$;

Link connection procedure at peer i:

2: **for all** j such that $j \in N_i$ **do**
3: send GATHER_CANDIDATES msg. to j;
4: **upon** receive PROPOSE_CANDIDATES msg. containing N_j from j **do**
5: $Cand_i \leftarrow Cand_i \cup (N_j \cap \overline{Cand_i})$; {avoid to add the same peer redundantly}
6: $Cand_i \leftarrow Cand_i \cap \overline{N_i}$; {remove already connected ones}
7: **if** $Cand_i \neq \{\emptyset\}$ **then**
8: send CONNECT_REQUEST msg. to k where $k \in Cand_i \wedge k = min(Cand_i)$;
9: **upon** receive CONNECT_ACK from k **do**
10: $N_i \leftarrow N_i \cup \{k\}$; {add k into N_i}
11: **for all** j such that $j \in N_i$ **do**
12: send UPDATE_DEGREE with $|N_i|$ to j;

Message handling at recipient i:

13: **upon** receive GATHER_CANDIDATES msg. from j **do**
14: send PROPOSE_CANDIDATES msg. with N_i to j;
15: **upon** receive CONNECT_REQUEST msg. from j **do**
16: $N_i \leftarrow N_i \cup \{j\}$
17: send CONNECT_ACK msg. to j;
18: **upon** receive UPDATE_DEGREE msg. with $|N_j|$ from j **do**
19: $d_i^j \leftarrow |N_j|$; {update d_i^j at peer i}

make $Cand_i$, i must avoid to select the bridges that can occur partitions by removing themselves. However, it is difficult to find such links, because each peer can only have the local view of two-hop distance. We consider this fact from the opposite side, that is, i constructs $Cand_i$ in the following way.

$$Cand_i = \bigcup_{j \in N_i} (N_i \cap N_j)$$

We take an intersection of N_i and sets of neighbors of every peer j in N_i to lead $Cand_i$. It means that elements of $Cand_i$ does not have a bridge to i. Thus, one of them and its link can be removed without network partitioning. Line 6 in the disconnection procedure makes $Cand_i$ in this way.

If a peer i has $Cand_i = \emptyset$, it tries to replace one of its links. In this case, i sends a REPLACE_REQUEST message to its neighborhood $j \in N_i$. j makes a link with one of the neighbors of i (i.e., N_i) and then sends a DISCONN_ACK message to i. Upon receiving the DISCONN_ACK message, i can remove the link (i,j) and reduce its degree.

3.3 Properties of the algorithm

We now discuss on the properties of the algorithm.

Connectivity We show that the algorithm guarantee that any pair of peers has at least one path. On the other words, any peer is never separated from others in the overlay network.

To prove the fact, first, we show that any pair of peers can not be partition away by removing links in the algorithm.

Lemma 1 (Local connectivity). *If there exist a link (a, b) between peer a and peer b, both peers can not be reachable.*

Proof. For adding a link, each peer a finds a candidate b that have the link to the peer c where $c \in N_a$. Thus, after the connection procedure, b has two paths to a reaching within two hops, and a is reachable to b if at most one link of b is removed. When a removes the link (a, b) and b is not reachable from a without (a, b), b makes a link to a certain peer d in N_a before removing (a, b). Therefore, it is guarantee that b has some neighboring peer x where $x \in N_a$ whenever a removes the link (a, b). It means that connectivity of

Algorithm 2 Procedures for disconnecting and replacing links

Initialization:
1: $Cand_i \leftarrow \{\emptyset\}$;
Link disconnection procedure at peer i
2: **for all** j such that $j \in N_i$ **do**
3: send GATHER_CANDIDATES msg. to j;
4: **upon** receive PROPOSE_CANDIDATES msg. containing N_j from j **do**
5: **if** $N_i \cap N_j \neq \{\emptyset\}$ **then**
6: $Cand_i \leftarrow Cand_i \cup (N_i \cap N_j)$;
7: **if** $Cand_i = \emptyset \vee \forall k \in Cand_i.(d_i^k \leq D)$ **then**
8: send REPLACE_REQUEST msg. to $l \in N_i$;
9: **else**
10: send DISCONN_REQUEST msg. to k where $k \in Cand_i \wedge d_i^k > D$;
11: **upon** receive DISCONN_ACK from k **do**
12: $N_i \leftarrow N_i \cap \overline{\{k\}}$
13: send DISCONN_CONFIRM msg. to k;
14: **upon** receive DISCONN_NACK from k **do**
15: send REPLACE_REQUEST msg. to $l \in N_i$;
16: **for all** j such that $j \in N_i$ **do**
17: send UPDATE_DEGREE msg. with $|N_i|$ to j;
Message handling at recipient i
18: **upon** receive DISCONN_REQUEST msg. from j **do**
19: **if** $|N_i| > D$ **then**
20: send DISCONN_ACK msg. to j;
21: **else**
22: send DISCONN_NACK msg. to j;
23: **upon** receive DISCONN_CONFIRM msg. from j **do**
24: $N_i \leftarrow N_i \cap \overline{\{j\}}$; {remove j from N_i}
25: **upon** receive REPLACE_REQUEST msg. from j **do**
26: $Cand_i \leftarrow N_j$;
27: send CONNECT_REQUEST msg. to k where $k \in Cand_i \wedge k = min(Cand_i)$;
28: **upon** receive CONNECT_ACK msg. from k **do**
29: $N_i \leftarrow N_i \cup \{k\}$;
30: **for all** l such that $l \in N_i$ **do**
31: send UPDATE_DEGREE msg. with $|N_i|$ to l;
32: send DISCONN_ACK msg. to j; {Note that j is a sender of REPLACE_REQUEST msg.}

the overlay network can not be lost in the disconnect procedure. □

Lemma 2 (Inter-subnet connectivity). *Suppose two sub-networks A and B in the overlay network. They are connected with the link (a, b) where $a \in A$ and $b \in B$. A and B always keep connected.*

Proof. If A and B are connected by the link (a, b), both sub networks are never separated by any procedure in the algorithm. Because the local connectivity shows that a is reachable to b even if (a, b) is removed. □

Theorem 3 (Global connectivity). *Neither any peer nor any sub-network in the overlay network are partitioned away in the algorithm if the overlay is connected.*

Proof. According to Lemma 1 and Lemma 2, this theorem can be proven. □

Non-redundancy It is obvious that the proposed algorithms (Algo. 1 and Algo 2) never make redundant links.

4 Conclusion

In this paper, we presented an algorithm which transforms an unstructured overlay network into the structured one with constant degree in each node. This algorithm does not require any other service. Thus, it has to consider network partition caused by operations of modifying overlay structure. We have proven that network partition is never happened in the proposed algorithm. We'll make an experiment to measure convergence time of our algorithm.

References

[1] A. Basu, B. Charron-Bost, and S. Toueg. Solving problems in the presence of process crashes and lossy links. Technical Report TR96-1609, Cornell University, USA, Sept. 1996.

[2] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[3] L. Massoulié, A. M. Kermarrec, and A. J. Ganesh. Network awareness and failure resilience in self-organising overlay networks. In *Proc. IEEE Int'l Symp. on Reliable Distributed Systems (SRDS'03)*, pages 47–55, Florence, Italy, Oct. 2003.

[4] R. Melamed and I. Keidar. Araneola: A scalable reliable multicast system for dynamic environments. In *Proc. IEEE Int'l Symp. on Network Computing and Applications (NCA'04)*, pages 5–14, Cambridge, MA, USA, aug 2004.

[5] K. Shen. Structure management for scalable overlay service construction. In *Proc. First Symp. on Networked Systems Design and Implementations (NSDI'04)*, San Francisco, California, USA, March 2004.

[6] A. Stavrou, D. Rubenstein, and S. Sahu. A lightweight, robust p2p system to handle flash crowds. *IEEE Journal on Selected Areas in Communications*, 22(1):6–17, 2004.

[7] S. Voulgaris, D. Gavidia, and M. van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):1143–1152, 2005.

[8] K. Watanabe, N. Hayashibara, and M. Takizawa. CBF: Look-up protocol for distributed multimedia objects in peer-to-peer overlay networks. *Journal of Interconnection Networks*, 6(3):323–344, 2005.