A Reliable Peer Connection Scheme for Pure P2P Network Environments

Gu Su Kim, and Young Ik Eom

School of Information and Communication Eng., Sungkyunkwan University,
300 Cheoncheon-dong, Jangan-gu, Suwon, Gyeonggi-do 440-746, Korea

Summary
P2P network environments provide users with direct data transmission and sharing facilities and those environments can be classified into hybrid P2P network environments and pure P2P network environments according to the arbitration mechanism among the peers in the network. In hybrid P2P network environments, there exists a server that maintains index information for the data to be shared and network isolation does not occur because every peer always keeps connection to the server. In pure P2P network environments, however, each peer directly connects to another peer and gets services without server intervention, and so, network isolation can occur when the mediating peer fails to work. In this paper, we propose a scheme for each peer to keep connection to other peers continuously by maintaining IP addresses of its neighbor peers and connecting to the peers when the mediating peer fails to work. Although the P2P application that uses our proposed framework should obtain one or more IP addresses of the neighbor peers manually, after instantiation, the application can do its job while maintaining connection to the network continuously and automatically. To evaluate our proposed scheme, we measured and analyzed the time for a peer to reconnect to the network when the mediating peer fails and the network isolation occurs.

Key words:
pure P2P, Peer Connection, Network Isolation, reliable connection, .

1. Introduction

P2P network environments for sharing resources among the peers in the network can be classified into two categories: hybrid and pure. In hybrid P2P network environments, as each peer connects with a server and then receives services from the server [1], the network isolation, which is a situation that a peer cannot connect to any other peer, does not occur. When the server fails to work, however, all the peers cannot get services from the server. To solve this problem, the pure P2P network framework and the applications based on the framework have been developed. In the pure P2P network environments, a peer can get services directly from another peer without going through the servers. Therefore, the peers in the pure P2P environments can get services more reliably and efficiently than in hybrid P2P environments. In the pure P2P environments, however, the network isolation can occur when the mediating peer fails to work [2].

In this paper, we propose a scheme that prevents network isolation in pure P2P network environments. With our scheme, all the peers can continuously join the network group by maintaining a list of neighbor peer IDs and reconnecting to another peer in the list when the mediating peer fails to work. At the initial stage, the P2P application that uses our proposed framework should manually obtain one or more IDs of the neighbor peers. After instantiation, the application can do its job while continuously and automatically maintaining connection to the network.

The rest of the paper is organized as follows. In Section 2, we give a brief introduction of the related work. The detailed algorithms and scenarios of the proposed scheme are described in Section 3. In Section 4, we present and analyze the simulation results. Finally, Section 5 discusses future works and concludes this paper.

2. Related Work

In this section, we give an overview of the P2P network environments and discuss the most typical 3 types of P2P environments that are being used practically.

2.1 Napster type

The P2P networking method in Napster environment is hybrid [3]. After connecting to a server in the network, each peer sends the information on the shared files that it has, and the server then maintains the shared file list for each peer.

When a peer sends a search message to the server for searching a file F, the server searches the DB of shared file information and sends the search result (host address, filename, etc.) to the peer. Eventually, the peer that receives the reply message from the server can connect

Manuscript received July 25, 2006.
Manuscript revised July 30 , 2006.

directly with the peer that contains the file F and obtain the file [3,4]. Figure 1 shows the network structure of the Napster environment.

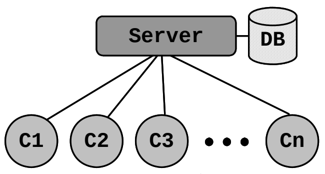


Fig. 1 The network structure of Napster environment

The Napster system has a merit that it can distribute the network traffic among the peers in the network and does not burden the server with the file service as in client-server environments. Even though the network traffic is distributed among the peers in the system, each peer should send index information for its shared files at connection time. Furthermore, each peer must always connect to the server to get index information whenever it wants file service. In these points of view, the Napster system has a drawback that the network traffic is still intensive around the server. Also, there is a reliability problem in that all the peers cannot get services when the server fails to work [5~7].

2.2 Soribada type

The P2P networking method in Soribada system, the P2P application system that provides sharing environment of MP3 files made by a company in Korea, is also hybrid. However, Soribada system is different from Napster system in that each peer does not send any information on shared files to the server at connection time. Figure 2 shows the network structure of the Soribada environment. As shown in the figure, Soribada system does not maintain any index information on shared files.

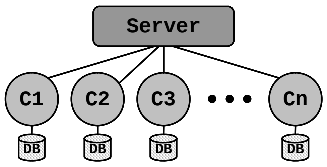


Fig. 2 The network structure of Soribada environment

In this environment, when a peer sends a search message to the server, the server analyzes the message and sends a request message to all the peers in the network. When a peer receives the request message, it checks its directory for the requested file and provides the server with the file information. Then the server summarizes the information from each peer, and sends the summary to the original requestor [8].

Soribada system has drawbacks, too. The network traffic concentrates around the server because the peers should contact the server for file service. It also has the reliability problem in that the overall system can stop when the server fails to work.

2.3 Gnutella type

The P2P networking method in Gnutella environment is taken as pure. In this system, all the peers have both roles of client and server [9,10]. Figure 3 shows the network structure of Gnutella system.

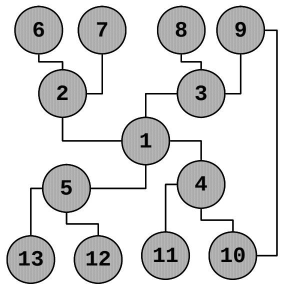


Fig. 3 The network structure of Gnutella environment

When an application is initially executed at a peer, it should get an ID of the peer that has already joined the Gnutella network and should connect to it. With this process, eventually, the peers in the system establish a mesh network and can share their resources in the system. The Gnutella system has a drawback of high network traffic and slow file access speed as the system basically uses broadcast mechanism for communication among the peers. Also, when a peer is disconnected from all the adjacent peers (probably due to peer failure, etc.) it cannot get any service from another peer in the system.

3. Reliable peer connection scheme

In this section, we propose a scheme that (1) reduces the latency for a peer to join a pure P2P network, (2) increases the chance that a peer can join the network group, and (3) distributes the connections so that a peer does not have too many connections.

3.1 System environments

We assume that all peers in the network open and listen to the same connection port and that each peer has at least one address (e.g. IP address) of another peer. This address can be input manually to the peer. With our proposed peer connection scheme, each peer P, after initial join to the network group, gradually connects to other peers. This paper describes the peer connection scheme based on a single network group.

3.2 Basic operation

The proposed scheme consists of the two phases: (1) ***the connect/request phase*** that a peer P_i connects to another peer P_j and requests the addresses of the peers that are known to B, and (2) ***the reply phase*** that a peer processes the messages obtained during the connect/request phase.

In the connect/request phase, after a peer P_i connects to the peer P_j whose address is known a priori to P_i, P_i requests the list of peer addresses that P_j has in its KPT(Known Peers Table). At this time, if the number of connections of P_j becomes greater than the number of recommended connections(N_Conns, to be discussed soon), P_j tries to close the oldest connections.

In the reply phase, the peer processes the messages such as *address request message*, *address reply message*, and *disconnection message*. When a peer receives the address request message, it sends the address list in its KPT to the requestor. When address reply message is received, the peer inserts the addresses in the list into its KPT. Then the connect/request phase is reinitiated with the new peers if the number of connections is still less than N_Conns. When disconnection message is received, the peer checks the number of connections and closes some connections if it is greater than N_Conns. The detailed message format is described in the next section.

3.3 Message formats and structures

Our scheme uses the REQ_AD message for address request, the REP_AD message for address reply, and the RESET message for disconnection. Figure 4 shows the detailed formats of the messages.

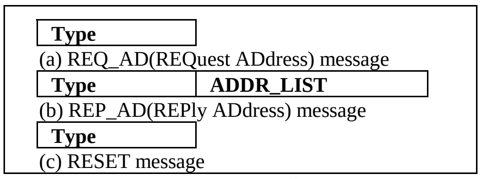

Type
(a) REQ_AD(REQuest ADdress) message
Type
ADDR_LIST
(b) REP_AD(REPly ADdress) message
Type
(c) RESET message

Fig 4. Message format

There are three kinds of messages for our peer connection scheme and they are identified by the Type field as shown in Figure 4. The ADDR_LIST field in the REP_AD message contains the addresses of the peers. The peer that receives this message inserts the addresses in the ADDR_LIST into its KPT along with the fields shown in Table 1.

Table 1: Field description of the KPT(Known Peers Table)

FieldName	Remarks
P_ADDR	Peer address
Try	TRUE if the peer has tried to connect with other peers, FALSE otherwise
KeepCon	TRUE if the connection should be maintained, FALSE otherwise
S_Handle	Handle of the connection(socket)
NumSuccess	Number of the connections succeeded
NumFail	Number of the connections failed
NumAttempt	Number of connection trials
FwNAT	TRUE if the peer exists in Firewall/NAT area, FALSE otherwise

When a peer modifies (or inserts) an entry for another peer in its KPT (upon the receipt of REP_AD message), the peer increases the value of NumAttempt of the entry by 1. Also, the value of NumFail is reset to 0 and the value of NumSuccess is incremented by 1 at this time. On the other hand, when a peer closes a connection with other peer or fails to connect to another peer, the value of NumFail field of the corresponding entry is increased by 1. When a peer cannot communicate with another peer that has already been connected to itself, the value of the FwNAT field of the corresponding entry is set to TRUE in order to indicate that the peer is in the firewall/NAT environments. In our scheme, when the value of NumFail field of an entry in KPT reaches a threshold value, the peer removes the entry from its KPT in order not to try connection with the peer.

3.4 Algorithms

In this section, we discuss the detailed algorithms for the connect/request phase and the reply phase.

(1) The connect/request phase

With our scheme, a peer P_i tries to connect with another peer when the number of connections with other peers is less than N_Conns. Also, peer A selects its partner from its KPT. Any peer that has not been connected before is selected as the connection partner. Let us call it P_j. In the connection process, peer P_i requests the addresses of the peers that the peer P_j has in its KPT, stores the address information into its KPT, and repeats this procedure until the number of connections becomes greater than or equal to N_Conns.

In the meantime, the number of connections of peer P_j can be increased. So peer B checks the number of connections. If the number is greater than N_Conns, peer B sends a disconnection message to the oldest connection and tries to reduce the number of connections. The detailed algorithm of the connect/request phase is described in Figure 5.

```
input: none
output: none
{
    ENTRY ety = NULL;
    SOCKET sock;
    while (ety == NULL) {
        ety = a KPT entry such that (Try == FALSE);
        if (ety != NULL) break;
        set the Try flags of all the entries in the KPT to FALSE;
    }
    sock = connect(ety.P_ADDR, PUBLIC_PORT);
    ety.NumAttempt++;
    ety.Try = TRUE;
    if (sock == NULL) {
        ety.NumFail++;
        if (ety.NumFail == MAX_FAIL)
            remove the ety from the KPT;
        return;
    }
    ety.NumSuccess++;
    ety.NumFail = 0;
    ety.S_Handle = sock;
    make a REQ_AD message and send it to sock;
    if (number of connected sockets <= N_Conns)
        return;
    ety = an entry such that (KeepCon==TRUE);
    sock = ety.S_Handle;
    if (sock != NULL) {
        ety.KeepCon = FALSE;
        make a RESET message and send it to sock;
    }
}
```

Fig. 5 The algorithm of the connect/request phase

In the connect/request phase, when the `Try` fields of all entries in the KPT are TRUE, the peer marks all the `Try` values to FALSE in order to make the peer try to reconnect to all the peers in the KPT. When the `Try` field of an entry in the KPT is FALSE, however, the peer tries to connect to the peer that the entry represents, and marks the `Try` field of the entry with TRUE. Also, when a peer fails to connect to another peer, the peer increases the value of the `NumFail` field of the corresponding entry by 1. When a peer succeeds in connecting to another peer, the peer sends a REQ_AD message to the connected peer and stores the socket handle of the connected peer in the `S_Handle` field of the corresponding entry.

When the number of connections of the peer becomes larger than `N_Conns`, the peer retrieves an entry that the `KeepCon` field is TRUE and marks the field of the entry to FALSE. Now, the peer sends a RESET message to the corresponding peer to inform that it allows disconnection. When the peer gets disconnected from another peer *and* the `KeepCon` field of the entry that contains the information of the disconnected peer has the value FALSE at this time, the peer will not execute the connect/request phase any more. In Figure 5, the `MAX_FAIL` parameter denotes the maximum number of connection trials permitted, and the `PUBLIC_PORT` is the common port that all the peers listen to.

```
input: message, socket handle
output: none
{
    switch (the type of the message) {
    case REQ_AD:
        REP_AD repad;
        repad.P_ADDR = addresses in the KPT;
        send the repad message
            to the socket handle;
        break;
    case REP_AD:
        make new entries or modify the entries
            in the KPT with the addresses in the message;
        break;
    case RESET:
        if (number of connections > N_Conns)
            close the connection;
        break;
    }
}
```

Fig. 6 The algorithm of the reply phase

(2) The reply phase

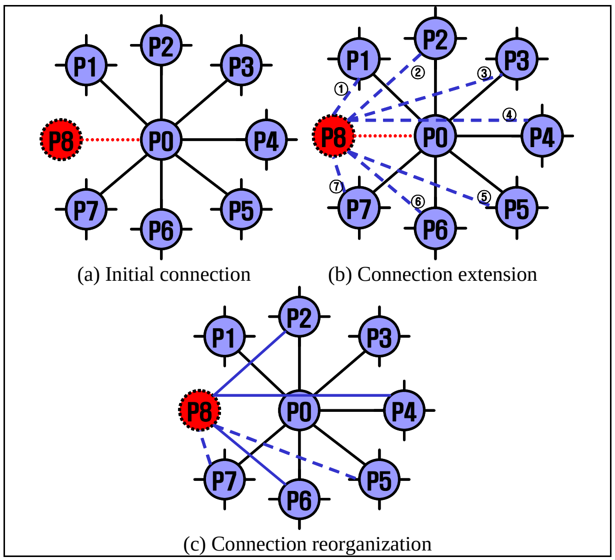
Figure 6 shows the procedure that should be executed when a peer receives REQ_AD, REP_AD, and RESET messages.

A peer that has received REQ_AD message sends REP_AD message to the requestor. This REP_AD message includes the addresses of the peers connected. A peer that has received REP_AD message inserts the addresses in the REP_AD message into its KPT. A peer that has received RESET message closes the connection with the peer that has sent RESET message when the number of connections becomes greater than `N_Conns`.

3.5 Scenario

In this section, we will describe a scenario that shows the peer connection process. In the scenario, each peer can be classified into two types: (1) a peer that tries to connect to another peer and (2) a peer that is *passively* connected by another peer. Figure 7 shows a process where peer `P8` gets connected to other peers.

In Figure 7-(a), an edge between two peers indicates that they are connected. Before peer `P8` opens a connection with peer `P0`, peer `P0` already has 7 connections, and each of `P1`, `P3`, `P5`, and `P7` has 5 connections, respectively. Similarly, each of `P2`, `P4`, and `P6` has 4 connections. After peer `P8` connects to the peer `P0`, `P8` sends the REQ_AD message to `P0`. Peer `P0` replies with the REP_AD message that includes the addresses of the peers `P1`, `P2`, `P3`, `P4`, `P5`, `P6`, and `P7`.



(a) Initial connection (b) Connection extension

(c) Connection reorganization

Fig. 7 A scenario of connection process (N_Conns: 5)

Now, peer P8 stores the address information into its KPT and tries to connect to the peers P1 through P7 as shown in Figure 7-(b). In the meantime, when the number of connections of a peer in the system becomes greater than N_Conns, the peer sends the RESET message to the oldest connection. Figure 7-(c) shows the connection reorganization step. The peer that has received a RESET message, if the number of connections becomes greater than N_Conns, closes the connection with the peer that has sent the RESET message. Eventually, a connection is maintained between a pair of peers only when the number of connections (of each peer) is less than or equal to N_Conns or each peer has not sent the RESET message. Also, peers P1 and P3 close the connection with peer P0 when they receive a RESET message from P0 because the number of connections of P1 and P3 became greater than N_Conns. In figure 7-(c), the solid edge indicates that the connection is maintained among the peers. In contrast, the dotted edge indicates that peers P5 and P7 sent a RESET message to peer P8 because the number of connections of P5 and P7 is greater than N_Conns. Hereafter, when the number of connections of P8 becomes greater than N_Conns, P8 will close the connection with P5 and P7.

4. Performance Evaluation and Analyses

We have simulated our peer connection scheme using Visual C++ 6.0 on Windows 2000 platform. In the simulation, a special peer is chosen, and all other peers in the network group are initially connected to the peer. Let us name it *P*. Black circles in Figure 7 are normal peers that tries to connect to *P* at boot time. After each peer connects to *P*, it additionally tries to connect to the peers that *P* knows so that it can continuously get services from the network group even when *P* terminates.

Our simulation has been performed for the period of 24 hours. In our simulation, we measured and analyzed the time to reconnect to the network group when the mediating peer fails to work. The environments for our simulation are shown in Table 2 and Table 3.

Table 2: Environments of the simulation

The attributes of the peers	**Number**	
	(a)	**(b)**
Number of peers in the firewall/NAT network area	10	20
Number of peers that use fixed address	30	60
Number of peers that use dynamic address	10	20
Total	50	100

Table 3: Environments of the simulation

Peer execution time	**Percentage**	
	(a)	**(b)**
1 hour – 2 hours	30%	30%
30 minutes – 1 hour	30%	30%
10 minutes – 30 minutes	40%	40%

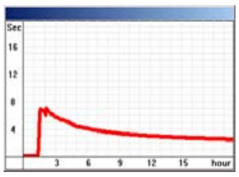
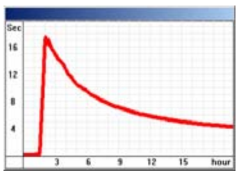
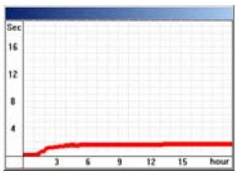
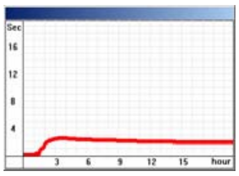
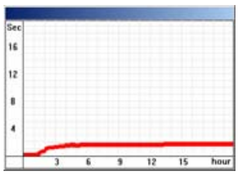
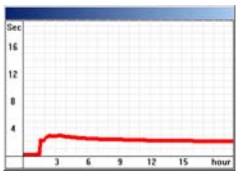
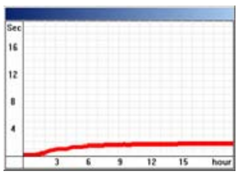
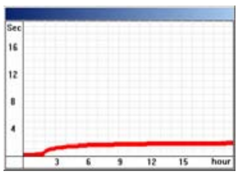
N_Conns	**Case (a)**	**Case (b)**
3		
5		
10		
20		

Fig. 8 The time to join the network group

Our simulation was performed with the numbers of peers (a) 50 and (b) 100. Figure 8 shows the results of the simulation for the cases (a) and (b). In Figure 8, the X-axis represents the time elapsed with the simulation, and the Y-axis represents the time consumed for a peer to join the network group. The simulation results show that `N_Conns` affects the time for a peer to join the network group. When many peers are disconnected from the network group while `N_Conns` is small, the time to join the network group increases. On the other hand, when `N_Conns` is large, the time for a peer to join the network group decreases. In this case, however, the connection trials generated by the peers increase because the number of connections maintained by a peer is large.

If we assume that all the peers have the knowledge about which peer uses fixed address and maintain this information in their KPTs, the time to join the network group can be further reduced by eliminating the information of the peers that use dynamic addresses from their KPTs.

The simulation results show how the time to join the network group can be reduced and how the proposed peer connection scheme resolves the network isolation problem.

5. Conclusion

Traditional pure P2P environments have the problem that a peer can be isolated from its network group and cannot get services from the peers in the network group when the mediating peer fails to work. Within the proposed scheme, the peers in pure P2P environments can have seamless connections with others in the network group even if the mediating peer fails to work. It has been achieved by efficiently and reliably maintaining the information about neighbor peers in each peer, and redirecting the connection when the mediating peer fails. We have evaluated our scheme with the simulation and showed that the time to join a network group depends on the threshold value for the number of connections that should be maintained in each peer. It has been validated in two experiments: the first experiment has been done with the number of peers set to 50, and the second experiment with the number of peers set to 100.

Our scheme may support smooth migration of mobile agents in the pure P2P network environments. By maintaining the addresses of other platforms, a mobile agent platform can actively migrate its agents to another platform and let the agent get seamless services thereafter. Also, our scheme can solve the problem that a client cannot get services from the server when the server in the hybrid P2P network environments fails to work.

References

[1] A. Oram, *Peer-To-Peer*, O'Reilly, Mar. 2001.

[2] B. Traversat, et. al., "Project JXTA Virtual Network," *Technique Report*, Sun Microsystems, Inc., Feb. 2002.

[3] B. Yang and H. Garcia-Molina, "Comparing Hybrid Peer-to-Peer Systems," *Proc. of the 27th International Conference on Very Large Databases*, Rome, Italy, Sep. 2001.

[4] B. Yang and H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks," *Proc. of the 22nd International Conference on Distributed Computing Systems*, IEEE, Vienna, Austria, Jul. 2002.

[5] David Barkai, "An Introduction to Peer-to-Peer Computing," *Developer Update Magazine*, Intel Corp., Feb. 2000.

[6] Endeavors Tech., "Introducing Peer-to-Peer," *White Paper*, Endeavors Technology Inc., 2002.

[7] Groove Networks Inc., "Why Peer-to-Peer," *White Paper*, Groove Networks Inc., 2002.

[8] D. C. Hyde, "How New Peer to Peer Developments May Effect Collaborative Systems," *Technical Report*, Dept. of Computer Science, Bucknell Univ., Jan. 2002.

[9] CLIP2, "The Gnutella Protocol Specification V0.4," *http://www.clip2.com*, Mar. 2001.

[10] Sandvine Inc., "Peer-to-Peer File Sharing: The Effects of File Sharing on a Service Provider's Network," *White Paper*, Sandvine Inc., Jul. 2002.

Gu Su Kim received his B.S., M.S. and Ph.D. degrees from the School of Electrical and Computer Engineering at Sungkyunkwan University in 1994, 1996, and 2007 respectively. He is currently in the post doc. course of the Department of Electrical and Computer Engineering, Sungkyunkwan University. He is currently studying P2P and mobile agent systems.

Young Ik Eom received his B.S, M.S. and Ph.D. degrees from the Department of Computer Science and Statistics of Seoul National University in Korea, in 1983, 1985 and 1991, respectively. From 1986 to 1993, he was an associate professor at Dankook University in Korea. He was also a visiting scholar in the Department of Information and Computer Science at the University of California, Irvine from Sep. 2000 to Aug. 2001. Since 1993, he is a professor at Sungkyunkwan University in Korea. His research interests include distributed computing, mobile computing, mobile agent systems and system securities.