2006년 한국산학기술학회 춘계 학술발표논문집

P2P 네트워크 환경의 효과적인 피어(Peer) 연결 기법

최 성, 박대효
남서울대학교 컴퓨터학과
jellers@naver.com

An Efficient Peer Connection Scheme for Pure P2P Network Environments

Choi Sung, Park dae-hyo
NamSeoul University

요 약

사용자가 자료를 직접 송수신하기 위한 P2P 네트워크 환경에는 하이브리드 P2P 네트워크 환경과 순수 P2P 네트워크 환경이 있다. 하이브리드 P2P 네트워크 환경에서 모든 피어들은 서버에 연결되어 있기 때문에, 피어들 간 메시지를 전달할 수 없는 네트워크 고립 상태가 발생되지 않는다. 그러나 순수 P2P 네트워크 환경에서 각 피어들은 서버 없이 서로 직접 연결하여 다른 피어로부터 서비스를 제공받기 때문에, 각 피어들 간의 통신을 중재하는 특정 피어가 종료할 경우 대상 피어들과의 통신이 단절되는 네트워크 고립 현상이 발생될 수 있다. 본 논문에서는 이러한 문제점을 해결하기 위해 각 피어들로 하여금 인접한 피어로부터 IP 주소를 얻어 목록으로 관리하게 함으로써, 연결된 피어가 종료할 경우 목록으로 유지된 IP 주소로 연결하여 네트워크 그룹에 지속적으로 참여할 수 있게 하는 기법을 연구하였다.

1. 서론

순수 P2P 네트워크 환경에서 각 피어들은 서버 없이 서로 직접 연결하여 다른 피어들로부터 서비스를 제공받는다. 따라서 각 피어들은 여러 호스트로부터 서비스를 제공받기 때문에, 하이브리드 P2P 네트워크 환경에서의 클라이언트들보다 안정적인 서비스를 제공받을 수 있다. 그러나 순수 P2P 네트워크 환경은 각 피어 간의 통신을 중재하는 특정 피어가 종료할 경우 대상 피어 간의 통신이 단절되는 네트워크 고립 현상이 발생된다.

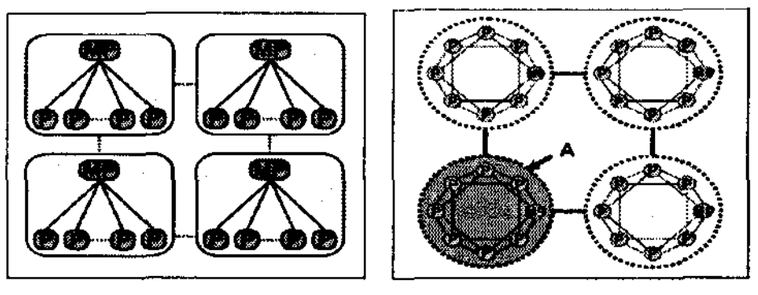
본 논문에서는 순수 P2P 네트워크 환경에서 발생될 수 있는 네트워크 고립 현상을 해결하기 위해 각 피어들로 하여금 인접한 피어로부터 IP 주소를 얻어 목록으로 관리하게 함으로써, 연결된 피어가 종료할 경우 목록으로 유지된 IP 주소로 연결하여 네트워크 그룹에 지속적으로 참여할 수 있게 하는 기법을 연구한다. 이 기법을 이용한 P2P 응용 프로그램에서는 최초 실행 시 하나 이상의 IP 주소를 입력받아야 하지만, 이후의 동작과정에서는 각 피어들이 다른 피어의 IP 주소를 능동적으로 확보하여 관리함으로써 네트워크 그룹에 지속적으로 연결할 수 있다.

2. 제안기법

본 논문에서는 기존의 Gnutella 방식과 같은 순수 P2P 네트워크 환경에서, 다른 피어와의 연결이 끊어졌을 경우 특정 피어에게 네트워크 소켓 연결이 집중되지 않도록 여러 피어에게 소켓 연결을 분산시킴으로써 각 피어들이 네트워크 그룹에 참여가 가능하도록 연구한다.

2.1 시스템 환경

이 기법을 이용한 모든 피어는 동일한 포트를 개방하고 있으며, 프로그램의 최초 실행 시 네트워크 그룹에 참여하고 있는 하나 이상의 피어 IP 주소가 이미 입력되어 있다고 가정한다. 일반적인 P2P 네트워크 환경과 본 논문에서 제안하는 기법에 적용될 네트워크 환경을 그림 1에서 보인다.



(a) 순수 P2P 네트워크 환경 (b) 본 논문 네트워크 환경

그림 1. 네트워크 환경

피어 P들은 최초 입력된 피어 MP에게 접속하여 네트워크 그룹에 참여하게 된다. 그러나 본 논문에서 제안하는 피어들의 연결은 그림에서 보이는 바와 같이 피어 P들로 하여금 피어 MP에 연결되어 있는 다른 피어에게 연결하게 하는 연결 구조를 가진다. 피어 MP는 다른 피어들이 최초 접속하는 피어이며, 피어 MP를 제외한 피어 P들은 피어 MP로 접속하도록 지정된 피어들이다. 피어 간 연결되어 있는 선들은 피어 간의 네트워크 소켓 연결을 의미한다. 본 논문에서는 네트워크 그룹의 일부인 그림 1-a 부분을 모델링 하도록 한다.

2.2 기본 동작

본 논문에서 제시하는 기법은 다른 피어로 네트워크 소켓 연결을 시도하여 상대방 피어에게 연결된 또 다른 피어들의 IP 주소 목록을 요청하는 연결/요청 단계와 다른 피어들로부터 수신된 요청 메시지를 처리하는 메시지 응답 단계로 나뉜다. 연결/요청 단계에서는 프로그램이 실행할 경우 저장되어 있는 IP 주소로 접속한 후 상대방 피어에게 IP 주소 목록을 요청하게 된다. 이 때, 연결 요청을 받은 피어는 소켓 수가 권장 소켓 수(N_Sockets)보다 많아질 경우 오랫동안 연결되어 있는 소켓 순으로 연결 포기 패킷을 전달한다. 메시지 응답 단계에서는 다른 피어로부터 IP 주소 요청 패킷, IP 주소 응답 패킷, 그리고 연결 포기 패킷을 수신하였을 경우 해당 루틴을 처리하게 된다. IP 주소 요청 패킷이 수신되었을 경우 네트워크 소켓으로 연결된 피어들의 IP 주소 목록을 요청한 피어에게 전송하여 준다. IP 주소 응답 패킷이 수신되었을 경우에는 상대방 피어가 보낸 IP 주소 목록을 자신이 유지하고 있는 IP 주소 테이블(IP address table)에 추가하고, 이 때 연결된 네트워크 소켓 수가 N_Sockets보다 작을 경우 수신된 IP 주소로 연결/요청 단계를 실행한다. 마지막으로 연결 포기 패킷이 수신되었을 경우에는 연결되어 있는 네트워크 소켓의 수를 측정하여 N_Sockets보다 많을 경우 연결 포기 패킷을 보낸 피어와의 소켓 연결을 끊는다.

2.3 패킷 형식

본 논문에서 제안하는 기법은 IP 주소 요청 패킷(REQ_AD), IP 주소 응답 패킷(REP_AD), 그리고 연결 포기 패킷(RESET)을 사용한다. 각 패킷들의 메시지 형식은 Type 필드의 값으로 구분된다. REP_AD 메시지의 IP_Addr에는 피어의 IP 주소들이 저장되며, REP_AD 메시지를 수신한 피어는 메시지 내의 IP 주소들을 자신의 IP 주소 테이블에 저장한다. 각 피어들이 다른 피어로부터 수신된 IP 주소를 IP 주소 테이블 내에 기록할 경우 수신된 IP 주소들이 네트워크 그룹에 참여하고 있는 피어들의 IP 주소임을 고려하여 해당 엔트리의 NumAttempt 값을 1만큼 증가시키고, NumFail 값은 0으로 세팅하며 NumSuccess 값을 1만큼 증가시킨다. 그리고 이미 연결되어 있는 네트워크 소켓 연결이 끊어지거나 IP 주소 테이블 내에 있는 IP_Addr로의 연결 접속이 실패하는 경우에는 해당 엔트리의 NumFail 값을 1만큼 증가시킨다. 또한 이미 연결되어 있는 IP 주소로의 연결 시도가 실패하는 경우에는 해당 엔트리의 FwNAT 값을 TRUE로 체크하여 상대방 피어가 방화벽/NAT 환경 내에 있는 것으로 표시한다. 본 기법에서는 이러한 연결 시도 후 특정 엔트리의 연속 접속 실패 횟수(NumFail)가 특정 수치에 도달하게 되면 IP 주소 테이블 내에서 제거하여 해당 IP 주소로 연결 재 시도를 못하도록 한다.

2.4 알고리즘

(1) 연결/요청 단계

순수 P2P 네트워크 환경에서 각 피어들은 다른 피어에게 소켓 연결을 한 후 IP 주소 목록을 요청하게 되며, 이 때, 다른 피어의 소켓 연결 시도로 인해 연결된 피어는 소켓 수가 권장 소켓 수(N_Sockets)보다 많아질 경우 오랫동안 연결되어 있는 소켓 순으로 연결 포기 패킷을 전송한다. 또한 네트워크 소켓 수가 N_Sockets보다 작은 피어는 소켓 수가 N_Sockets에 도달하도록 하기 위해 IP 주소 테이블 내에서 아직 연결을 시도하지 않은 IP 주소로 소켓 연결을 시도한다. 각 피어에게 연결되어 있는 네트워크 소켓 연결 수가 N_Sockets보다 작을 경우 실행되는 과정을 알고리즘 1에서 보인다.

알고리즘 1. 연결/요청 단계

```
input : none
output : none
{
    ENTRY ety;
    SOCKET sock;
    ety = an entry that (Try == FALSE);
    if ( ety == NULL ) {
        set the Try flags of all entries in the IP address table to FALSE;
        return;
    }
    sock = connect(ety.IP_Addr, PUBLIC_PORT);
    ety.NumAttempt++;
    ety.Try = TRUE;
    if ( sock == NULL ) {
        ety.NumFail++;
        if ( ety.NumFail == MAX_FAIL )
            remove ety in the IP address table;
        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_Sockets )
        return;
    ety = an entry that (KeepSock == TRUE);
    sock = ety.S_Handle;
    if ( sock != NULL ) {
        ety.KeepSock = FALSE;
        make a RESET message and send it to the socket handle;
    }
}
```

연결/요청 단계에서는 IP 주소 테이블 내의 모든 IP 주

소에 이미 연결 시도를 하였을 경우 지금까지 연결을 시도했던 모든 IP 주소들에 대해 연결 시도가 없었던 것으로 설정한다. 그러나 IP 주소 테이블 내에 연결 시도를 하지 않은 IP 주소가 존재한다면 연결/요청 단계 중인 피어는 연결 시도를 하지 않은 것으로 표시된 엔트리를 획득한 후 엔트리 내의 IP 주소로 연결을 시도하게 되며, 해당 엔트리의 Try 플래그 값을 TRUE로 설정한다. 또한 각 피어들은 다른 피어로의 접속이 실패할 경우 해당 엔트리의 NumFail 플래그 값을 1만큼 증가시키지만, 접속이 성공할 경우에는 접속된 피어에게 REQ_AD 메시지를 전송하고, 연결된 소켓 핸들을 해당 엔트리의 S_Handle에 저장한다. 각 피어들은 연결된 소켓의 수가 N_Sockets보다 많아질 경우 KeepSock 플래그 값이 TRUE로 설정되어 있는 엔트리를 얻어 온 후 FALSE로 세팅하며 상대방 피어에게 소켓 연결을 끊어도 좋다는 RESET 메시지를 전송한다. 이 때 상대방 피어가 소켓 연결을 종료할 경우 연결이 끊어지게 된 피어는 해당 엔트리의 KeepSock 플래그 값을 확인하여 FALSE로 설정되어 있다면 연결/요청 단계를 재실행하지 않는다. 알고리즘 1에서 MAX_FAIL은 연결이 실패할 경우의 최대 연결 시도 횟수를 의미하며, 시스템 파라메타로 미리 주어지는 것으로 한다. 또한 PUBLIC_PORT는 모든 피어들이 동일하게 개방하고 있는 포트를 의미한다.

(2) 메시지 응답 단계

REQ_AD, REP_AD, 그리고 RESET 메시지를 수신한 피어의 작업 절차는 알고리즘 2에서 보인다.

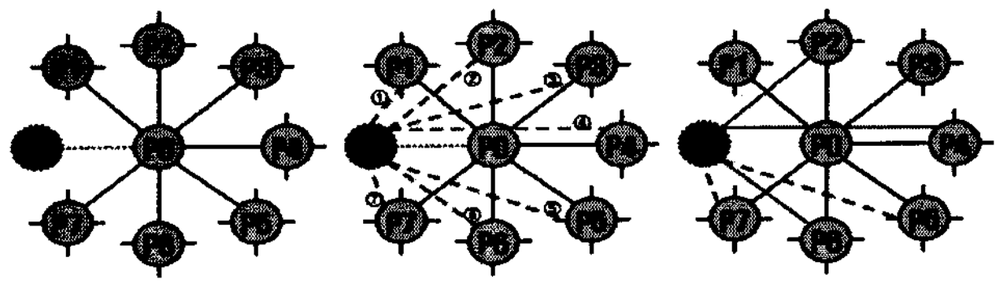
알고리즘 2. 메시지 응답 단계

```
input : message, socket handle
output : none
{
    switch (the type of the message) {
    case MESSAGE_REQ_AD:
        REP_AD repad;
        repad.IP_Addr = IP addresses in the IP address tables;
        send the repad message to the socket handle;
        break;
    case MESSAGE_REP_AD:
        make new entries with IP addresses in the message;
        add the entries to the IP address table;
        break;
    case MESSAGE_RESET:
        if ( number of connected sockets > N_Sockets )
            disconnect socket handle;
        break;
    }
}
```

REQ_AD 메시지를 받은 피어는 메시지를 송신한 피어에게 연결된 피어들의 IP 주소들이 포함된 REP_AD 메시지를 전송하며, REP_AD 메시지를 받은 피어는 메시지 내의 IP 주소들을 IP 주소 테이블에 추가한다. RESET 메시지를 받은 피어는 연결되어 있는 네트워크 소켓의 수가 N_Sockets보다 많을 경우 RESET 메시지를 보낸 피어와의 소켓 연결을 끊는다.

2.5 시나리오

본 시나리오에서 각 피어들은 연결 시도 주체에 따라 연결을 시도하는 피어와 다른 피어로부터 연결되는 피어로 분류할 수 있으며, 특정 피어가 다른 피어로 접속할 경우의 모습을 그림 6에서 보인다.



(a) 최초 연결 단계 (b) 피어 연결 단계 (c) 소켓 정리 단계

그림 2. 접속 후 연결 3단계(N_Sockets:5)

그림 2-a에서, 각 피어들을 연결하는 선은 피어들 간 네트워크 소켓 연결을 의미하며, 피어 P8이 피어 P0에게 접속하기 전에 피어 P0은 7개의 소켓, 피어 P1, P3, P5, P7은 5개의 소켓, 피어 P2, P4, P6은 4개의 소켓을 생성하고 있는 모습을 보인다. 피어 P8은 실행 시 프로그램 내에 미리 지정된 피어 P0으로 소켓 연결을 시도한다. 피어 P8과 연결된 피어 P0은 이미 연결되어 있는 다른 피어들의 IP 주소들을 피어 P8에게 전송하게 되고, 피어 P8은 피어 P0이 송신한 피어 P1, P2, P3, P4, P5, P6, P7의 IP 주소들을 수신하게 된다. 그림 2-b에서, 피어 P0으로부터 주소 정보를 수신한 피어 P8은 피어 P1부터 피어 P7까지 네트워크 소켓 연결을 시도하게 되며, 본 그림에서는 피어 P8이 ①~⑦ 순서로 각 피어들에게 접속함을 보인다. 이후 네트워크 소켓 연결 수가 N_Sockets보다 많아질 경우 각 피어들은 가장 오래된 네트워크 소켓 연결 순으로 RESET 메시지를 보내게 된다. 그림 2-c에서는 RESET 메시지를 받은 피어들의 소켓 수가 N_Sockets보다 많을 경우 연결을 끊은 상태를 보이며, 각 피어들은 소켓의 수가 N_Sockets보다 작거나 또는 RESET 메시지가 오지 않은 피어와의 소켓 연결만을 유지하게 된다. 또한 RESET 메시지를 수신한 피어 P1, P3은 연결 소켓의 수가 N_Sockets 수보다 많기 때문에 피어 P0과의 네트워크 소켓 연결을 끊게 된다. 그림 2-c에서 실선은 피어 P2, P4, P6의 소켓 수가 N_Sockets이기 때문에 더 이상의 소켓 수를 줄일 수 없다는 의미를 나타내며, 점선은 피어 P5, P7의 소켓 수가 N_Sockets보다 많기 때문에 피어 P5, P7이 피어 P8에게 RESET 메시지를 보낸 상태를 나타낸다. 이후 피어 P8의 네트워크 소켓 수가 N_Sockets보다 많아지게 된다면 피어 P8은 피어 P5와 P7과의 연결을 우선적으로 제거한다.

3. 시뮬레이션

3.1 시뮬레이션 개요

시뮬레이션 프로그램은 Visual C++ 6.0을 사용하였다.

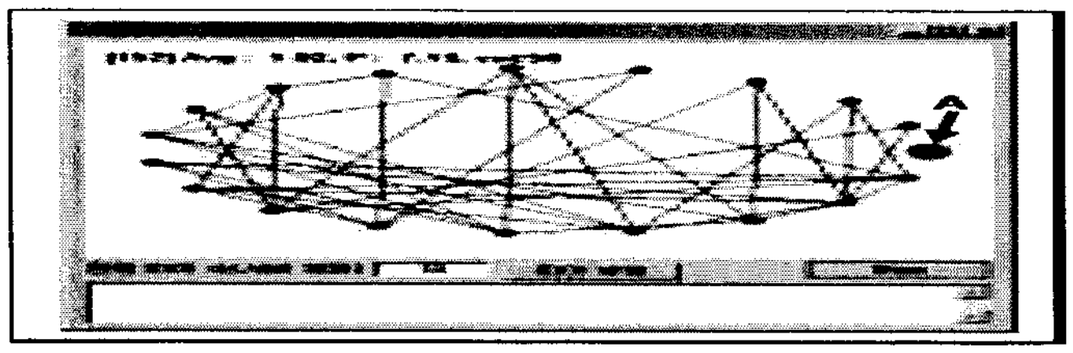


그림 3. 시뮬레이션 중인 프로그램의 화면

그림 3에서의 "피어 생성" 버튼은 테스트 하고자 하는 피어의 개수를 입력한 후 피어를 생성하는 버튼이며, "Stop" 버튼은 실행중인 시뮬레이션을 정지시키는 버튼이다. 그림 3서, 각 피어들은 피어 A에게 연결한 후 IP 주소들을 수신하여 다른 피어로 소켓 연결을 시도한다. 또한 피어 A가 종료할 경우 각 피어들은 유지된 IP 주소 목록을 이용하여 다른 피어에게 접속함으로써 네트워크 그룹에 지속적으로 참여하게 된다. 3개의 피어를 제외한 나머지 피어들이 모두 종료하였을 경우에 대한 네트워크 소켓 연결 모습을 그림 4에서 보인다.

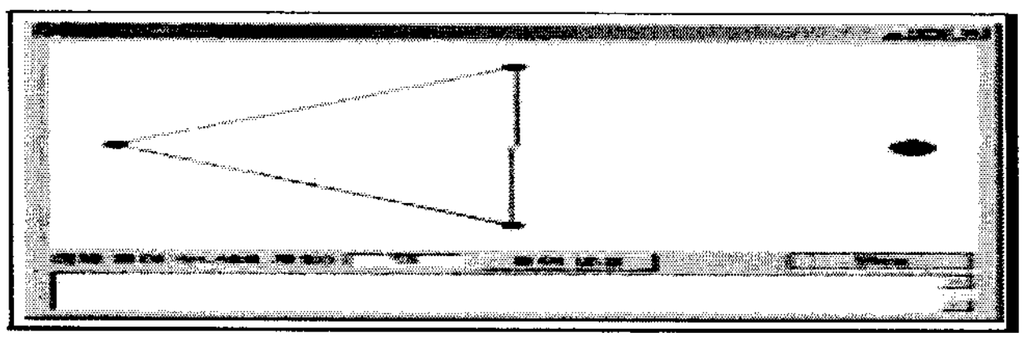


그림 4. 피어 3개만이 남아 있을 경우

그림 4는 연결을 중재하는 피어들이 종료하여 네트워크 소켓 연결이 끊어져도 남아있는 피어들은 목록으로 유지된 IP 주소로 연결하여 네트워크 그룹에 참여함으로써 서비스를 제공하거나 받을 수 있다는 것이다.

4. 비교 분석

본 시뮬레이션은 24시간동안 동작되었으며, 각 피어들이 다른 피어에게 접속함으로써 네트워크 그룹에 참여하는데 걸리는 시간을 측정하였다. 시뮬레이션에 사용한 피어의 수는 각각 50개와 100개로 하였다. 이 수치를 기반으로 한 시뮬레이션의 결과는 그림 9에서 보인다.

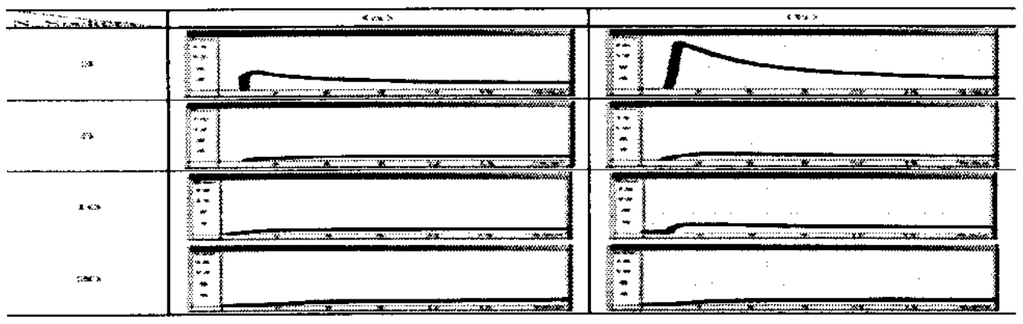


그림 5. N_Sockets와 네트워크 그룹 참여를 위해 소요되는 시간과의 관계

그림 5에서, X축은 시뮬레이션이 진행된 시간을 나타내며, Y축은 피어들이 네트워크 그룹에 참여할 때까지 소요되는 평균 시간을 나타낸다. 시뮬레이션의 결과에서와 같이 N_Sockets의 수치는 피어가 네트워크 그룹에 참여할 때까지 소요되는 시간에 영향을 미치는 것을 알 수 있다. N_Sockets 값이 작은 상태에서 연결된 피어들이 모두 종료할 경우 네트워크 그룹으로부터의 단절이 많아지기 때문에 새로운 피어로 접속하기 위해서는 많은 시간이 소요되며 접속 빈도수도 많아지게 된다. 그러나 각 피어들은 N_Sockets 값이 커질수록 네트워크 그룹으로부터 단절되는 가능성이 줄어들지만 각 피어들이 유지해야 하는 네트워크 소켓 수도 많아지게 됨으로써 각 피어 간의 통신을 위한 패킷의 발생 빈도수가 많아지게 된다. 차후 피어가 유동 IP 주소를 사용하는지의 여부를 파악하여 각 피어들로 하여금 순수 고정 IP 주소를 사용하는 피어들의 IP 주소 목록만을 관리하게 한다면, 네트워크 그룹으로부터 고립된 피어들은 실행시마다 바뀔 수 있는 피어들의 IP 주소를 목록에서 제외함으로써 보다 빠른 시간 내에 네트워크 그룹에 참여할 수 있게 된다. 본 시뮬레이션의 결과는 피어들로부터 연결이 끊어질 경우 서비스를 전혀 받지 못하였던 기존의 문제점이 해결되었음을 보인다.

5. 결 론

본 제안 기법을 이용한 피어들은 순수 P2P 네트워크 환경을 기반으로 하여, 연결된 피어로부터 얻은 또다른 피어들의 IP 주소들을 목록으로 관리함으로써, 연결되어 있는 피어가 종료하여도 목록으로 유지된 다른 IP 주소로 연결을 유지할 수 있게 된다. 기존의 순수 P2P 프로그램들은 서버와 클라이언트의 기능을 모두 가지고 있다는 점에서 순수 P2P 네트워크 환경을 기반으로 구현되었다고 할 수 있지만, 연결된 피어들이 모두 종료되는 경우 다른 피어의 IP 주소를 입력하지 않는다면 네트워크 그룹에 참여할 수 없게 된다는 단점을 갖는다. 따라서 본 제안 기법을 이용하여 다양한 피어들의 IP 주소를 목록

으로 관리한다면, 각 피어들은 연결되어 있는 피어들이 종료할 경우에도 목록 내의 IP 주소로 연결하여 네트워크 그룹에 지속적으로 참여할 수 있게 되는 것이다. 차후 이 기법은 각 에이전트 플랫폼으로 하여금 에이전트를 이주 시키기 위해 필요한 플랫폼의 IP 주소를 관리하게 함으로써 플랫폼간의 에이전트 이주를 원활하게 지원할 수 있으며, 기존의 순수 P2P 응용 프로그램에 적용하여 네트워크 그룹에 항상 연결될 수 있도록 유지할 수 있다. 또한 지금까지 하이브리드 P2P 네트워크 환경에서 서버를 재가동시킬 경우 각 피어들이 서버로부터 서비스를 제공받지 못하였던 여러 가지 서비스들의 중단 현상까지도 해결할 수 있을 것으로 생각된다.

참고문헌

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

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

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