순수 P2P 네트워크 환경을 위한 효율적인 피어 연결 기법

(An Efficient Peer Connection Scheme for Pure P2P Network Environments)

김 영 진 † 엄 영 익 ††
(Young-jin Kim) (Young Ik Eom)

요 약 각 사용자들이 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, 피어들간 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 피어들간의 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 제안한다. 이 제안 기법을 이용한 P2P 응용 프로그램은 최초 실행시 하나 이상의 IP 주소를 입력받아야 하지만, 실행 이후에는 사용자로부터 어떠한 정보도 받지 않으면서 네트워크 그룹에 지속적으로 연결될 수 있다. 제안기법의 평가를 위해 시뮬레이션을 통해 연결되어진 피어들이 종료할 경우 네트워크 그룹에 재연결할 때까지 걸리는 시간을 측정하였다.

키워드 : P2P, 분산 컴퓨팅, 네트워크 고립, 피어 연결

Abstract 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 : P2P, Distributed Computing, Network Isolation, Peer Connection

1. 서 론

각 사용자들이 자료를 직접 송수신하기 위한 P2P 네트워크 환경에는 하이브리드 P2P 네트워크 환경과 순수 P2P 네트워크 환경이 있다. 하이브리드 P2P 네트워크 환경에서 모든 피어들은 서버에 연결한 후 일괄적인 서비스를 제공받기 때문에[1], 피어들간에 메시지를 전달

· 이 논문은 2002년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2002-041-D00420)
† 비 회 원 : 성균관대학교 정보통신공학부
door21c@tjssm.co.kr
†† 종신회원 : 성균관대학교 정보통신공학부 교수
yieom@ece.skku.ac.kr
논문접수 : 2003년 5월 7일
심사완료 : 2003년 10월 13일

할 수 없는 네트워크 고립 상태가 발생되지 않는다. 하지만 서비스를 제공하는 서버가 운영되지 않는 경우에는 각 피어들이 더 이상의 서비스를 제공받을 수 없게 된다는 문제점이 발생한다. 이러한 문제점을 해결하기 위해 순수 P2P 네트워크 환경 기반의 응용 프로그램들이 개발되기 시작하였다. 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 피어들간의 통신을 중재하는 특정 피어가 종료할 경우 대상 피어들간의 통신이 단절되는 네트워크 고립 현상이 발생될 수 있다[2].

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 제안한다. 이 제안 기법을 이용한 P2P 응용 프로그램에서는 최초 실행시 하나 이상의 IP 주소를 입력받아야 하지만, 이후의 동작 과정에서는 각 피어들이 다른 피어의 IP 주소를 능동적으로 확보하여 관리함으로써 네트워크 그룹에 지속적으로 연결할 수 있게 된다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 P2P 네트워크 환경에 대한 관련 연구를 소개한다. 3장에서는 순수 P2P 네트워크 환경 유지를 위한 소켓 연결 분산 기법의 동작과정과 시나리오를 제시한다. 4장에서는 제안한 시스템의 시뮬레이션 과정을 보이고, 5장에서는 제안 기법에 대한 성능을 분석한다. 마지막으로 6장에서는 결론을 맺고 향후 필요한 연구를 제시한다.

2. 관련연구

본 장에서는 순수 P2P 방식, 하이브리드 P2P 방식 등 P2P 네트워크 환경의 여러 가지 기존 방식들에 대해 알아보도록 한다.

2.1 Napster 방식

이 방식은 하이브리드 P2P 방식으로, 각 피어들이 서비스를 제공하는 특정 서버에 접속한 후 사용자가 공유한 파일에 대한 정보를 서버에 전송하게 되며, 이 정보를 받은 서버는 각 피어들의 파일 목록을 관리하게 된다. 이후 다른 사용자가 특정 파일을 찾기 위해 검색 메시지를 서버에 전송할 경우 서버는 각 피어들이 보낸 파일 목록 중 사용자가 원하는 파일을 보유한 피어 주소와 파일명 등의 정보를 요청한 피어에게 전달하게 되

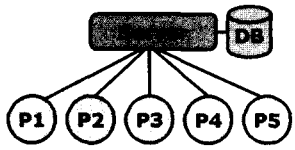


그림 1 Napster 방식의 네트워크 구조

며, 이 메시지를 받은 피어는 파일을 보유한 피어와 직접 연결을 맺음으로써 파일을 수신할 수 있게 된다[3,4]. 그림 1에서는 Napster 방식의 네트워크 구조를 보인다.

Napster 방식은 기존 클라이언트/서버 구조에서 사용자들이 특정 파일을 서버에서 받아감으로써 발생하는 네트워크 트래픽을 일반 피어들에게 분산시킴으로써 자원을 빠른 속도로 공유할 수 있다는 장점을 가지고 있다. 그러나 각 피어들은 자원의 검색을 위해 항상 서버와 연결되어 있어야 하고 공유한 파일 목록을 서버로 전송해야 하기 때문에 서버로 네트워크 트래픽이 집중되는 특성을 갖는다. 또한 피어들은 이러한 목록을 관리하는 서버가 종료할 경우 서비스를 제공받을 수 없게 된다는 단점을 갖는다[5-7].

2.2 소리바다 방식

이 방식 또한 하이브리드 P2P 방식이지만 기존의 Napster 방식과 다른 점은 피어들이 파일에 대한 어떠한 정보도 미리 서버에게 보내지 않는다는 것이다. 그림 2에서는 소리바다 방식의 네트워크 구조를 보인다.

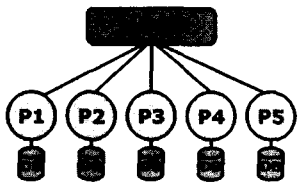


그림 2 소리바다 방식의 네트워크 구조

서버에 접속한 각 피어들이 서버에게 파일 검색 메시지를 전송할 경우 메시지를 수신한 서버는 현재 연결된 모든 피어들에게 검색 요청 메시지를 전송하게 된다. 이 메시지를 받은 피어들 중 지정된 파일을 가지고 있는 피어들은 파일 정보를 서버에게 전송하게 되며, 이 정보를 받은 서버는 최초 자료 검색 요청을 한 피어에게 파일 정보가 담긴 응답 메시지를 전송하는 방식으로 진행된다[8]. 그러나 이러한 방식 또한 서버가 모든 요청과 답변을 중재함으로써 네트워크 트래픽이 여전히 한 호스트로 집중되는 문제가 발생하게 되며, 클라이언트/서버 방식에서처럼 서버 의존적인 연결 방식이기 때문에 서버가 종료되면 서비스를 제공받을 수 없게 된다는 단점을 갖는다.

2.3 Gnutella 방식

이 방식은 순수 P2P 방식으로 특정 서버를 필요로 하지 않으며, 각 피어들로 하여금 서버와 클라이언트의 기능을 모두 가지게 하는 방식이다[9,10]. 그림 3에서는 Gnutella 방식의 네트워크 구조를 보인다.

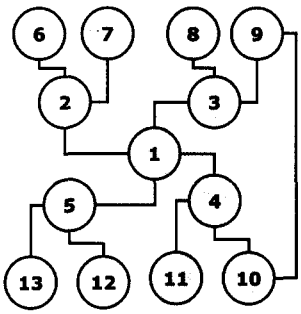


그림 3 Gnutella 방식 네트워크 구조

한 피어에서 Gnutella 방식을 이용하는 프로그램이 처음 실행될 시에는 Gnutella 네트워크에 참여하고 있는 다른 하나 이상의 피어 IP 주소를 입력/연결해 주어야 하며, 이러한 방식으로 모든 피어들은 피라미드식으로 연결되어 무제한의 자료를 공유하게 된다. 그러나 Gnutella 방식은 각 피어들이 자원을 검색하기 위해 다른 피어에게 메시지를 전송하는 경우 Gnutella 네트워크에 참여하고 있는 수많은 피어들에게 메시지들이 전송되기 때문에 자원 검색 속도의 저하를 일으키는 단점이 있다. 또한 프로그램에 입력된 IP 주소와의 연결이 모두 끊어지는 경우 더이상의 서비스를 제공받을 수 없다는 단점을 갖는다.

3. 제안기법

본 장에서는 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 항상 참여할 수 있게 하는 기법을 소개한다.

3.1 시스템 환경

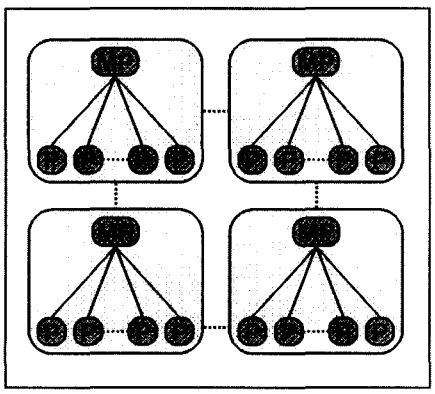
본 기법을 이용한 모든 피어는 동일한 포트를 개방하고 있으며, 프로그램의 최초 실행시 네트워크 그룹에 참여하고 있는 하나 이상의 피어 IP 주소가 이미 입력되어 있다고 가정한다. 일반적인 P2P 네트워크 환경과 본 논문에서 제안하는 기법에 적용될 네트워크 환경을 그림 4에서 보인다.

그림 4(a)에서, 피어 P들은 최초 입력된 피어 MP에게 접속하여 네트워크 그룹에 참여하게 된다. 그러나 본 논문에서 제안하는 피어들의 연결은 그림 4(b)에서 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. 피어간 연결되어 있는 선들은 피어들간의 네트워크 소켓 연결을 의미한다. 본 논문에서는 네트워크 그룹의 일부인 그림 4(b)의 A 부분을 모델링 하도록 한다.

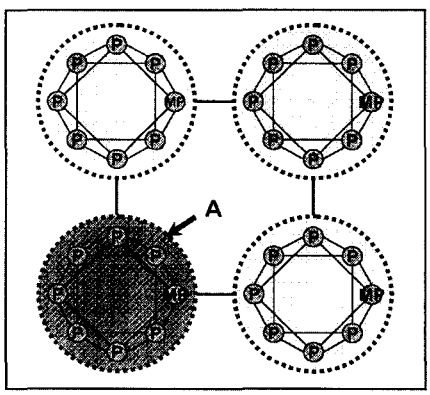
3.2 기본 동작

본 논문에서 제시하는 기법은 다른 피어로 네트워크 소켓 연결을 시도하여 상대방 피어에게 연결된 또다른 피어들의 IP 주소 목록을 요청하는 **연결/요청 단계**와 다른 피어들로부터 수신된 요청 메시지를 처리하는 **메시지 응답 단계**로 나뉜다.

연결/요청 단계에서는 프로그램이 실행할 경우 저장



(a) 순수 P2P 네트워크 환경



(b) 본 논문 네트워크 환경

그림 4 네트워크 환경

되어 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 단계**에서는 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.

3.3 패킷 형식

본 논문에서 제안하는 기법은 IP 주소 요청 패킷(REQ_AD), IP 주소 응답 패킷(REP_AD), 그리고 연결 포기 패킷(RESET)을 사용하며, 그 형식은 그림 5에서 보인다.

그림 5에서, 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. 이 때 표 1에서 보이는 바와 같은 여러 가지 필드들을 함께 저장한다.

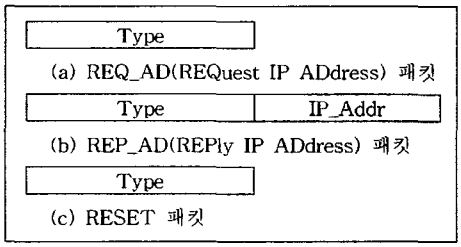


그림 5 패킷 형식

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 재시도를 못하도록 한다.

3.4 알고리즘

본 논문에서 제안하는 기법은 각 피어들이 다른 피어에게 연결한 후 IP 주소 목록을 요청하는 연결/요청 단계와 다른 피어로부터 수신된 메시지들을 처리하는 메시지 응답 단계로 구성되며, 본 절에서는 각 단계별로 세부 알고리즘을 설명한다.

(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.

표 1 IP 주소 테이블의 구성

명 칭	내 용
IP_Addr	피어 IP 주소
Try	접속 시도한 적이 있는지를 표시하는 플래그
KeepSock	소켓 연결 종료 가능 여부를 표시하는 플래그
S_Handle(socket handle)	연결된 소켓 핸들
NumSuccess	연결 성공 횟수
NumFail	연결 접속 실패 횟수
NumAttempt	연결 시도 횟수
FwNAT	Firewall/NAT 유무를 표시하는 플래그

```
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;
    }
}
```

알고리즘 1 연결/요청 단계

연결/요청 단계에서는 IP 주소 테이블 내의 모든 IP 주소에 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.

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.

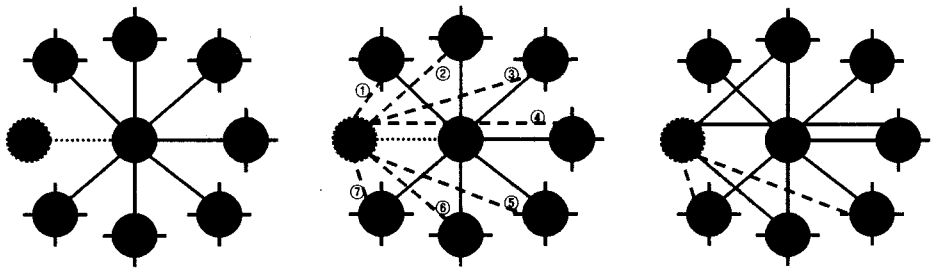
3.5 시나리오

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.

그림 6(a)에서, 각 피어들을 연결하는 선은 피어들간 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

```
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;
    }
}
```

알고리즘 2 메시지 응답 단계



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

그림 6 접속 후 연결 3단계(N_Sockets:5)

생성하고 있는 모습을 보인다. 피어 P8은 실행시 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 6(b)에서, 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 6(c)에서는 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 6(c)에서 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 재거한다.

4. 시뮬레이션

본 논문에서 제안하는 기법을 시뮬레이션하기 위해 Windows 2000 플랫폼을 사용하였으며, 툴은 MS사의 Visual C++ 6.0을 사용하였다. 시뮬레이션 프로그램의 화면 구성은 그림 7에서 보인다.

그림 7에서의 "피어 생성" 버튼은 테스트 하고자 하는 피어의 개수를 입력한 후 피어를 생성하는 버튼이며, "Stop" 버튼은 실행중인 시뮬레이션을 정지시키는 버튼이다. 화면 좌측 상단에 있는 Avg와 P의 값은 피어들

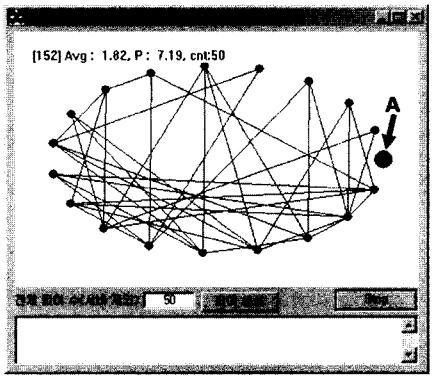


그림 7 시뮬레이션 중인 프로그램의 화면

에 의해 생성된 네트워크 소켓 수의 평균과 표준편차를 나타내며, 변수 cnt의 값은 테스트되고 있는 피어의 개수를 나타낸다. 피어 "A"는 처음 실행되는 각 피어들로부터 접속되는 피어이다. 그 외의 작은 원들은 일반 피어들로써, 최초 실행시 피어 A에게 연결하도록 설정되어 있다. 그림 7에서, 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 8에서 보인다.

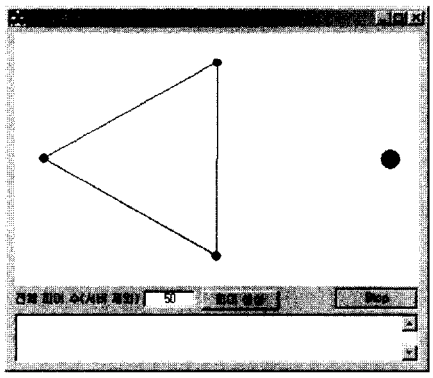


그림 8 피어 3개만이 남아 있을 경우의 시뮬레이션 동작 모습

그림 8은 연결을 중재하는 피어들이 종료하여 네트워크 소켓 연결이 끊어져도 남아있는 피어들은 목록으로 유지된 IP 주소로 연결하여 항상 네트워크 그룹에 참여함으로써 서비스를 제공하거나 받을 수 있다는 것을 보여준다.

5. 비교 분석

본 시뮬레이션은 24시간동안 동작되었으며, 각 피어들이 다른 피어에게 접속함으로써 네트워크 그룹에 참여하는데 걸리는 시간을 측정하였다. 시뮬레이션에 사용한 수치 데이타는 표 2에서 보인다.

표 2 시뮬레이션에서 사용한 인자 값

피어 IP 상태	개 수	
	(a)	(b)
방화벽/NAT 환경 내에 있는 피어 수	10	20
일반 고정 IP를 사용하고 있는 피어 수	30	60
유동 IP를 보유하고 있는 피어 수	10	20
합 계	50	100

피어 실행 시간	백분율	
	(a)	(b)
1시간~2시간	30%	30%
30분~1시간	30%	30%
10분~30분	40%	40%

시뮬레이션에 사용한 피어의 수는 각각 50개와 100개로 하였다. 이 수치를 기반으로 한 시뮬레이션의 결과는 그림 9에서 보인다.

그림 9에서, 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 피어들간의 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.

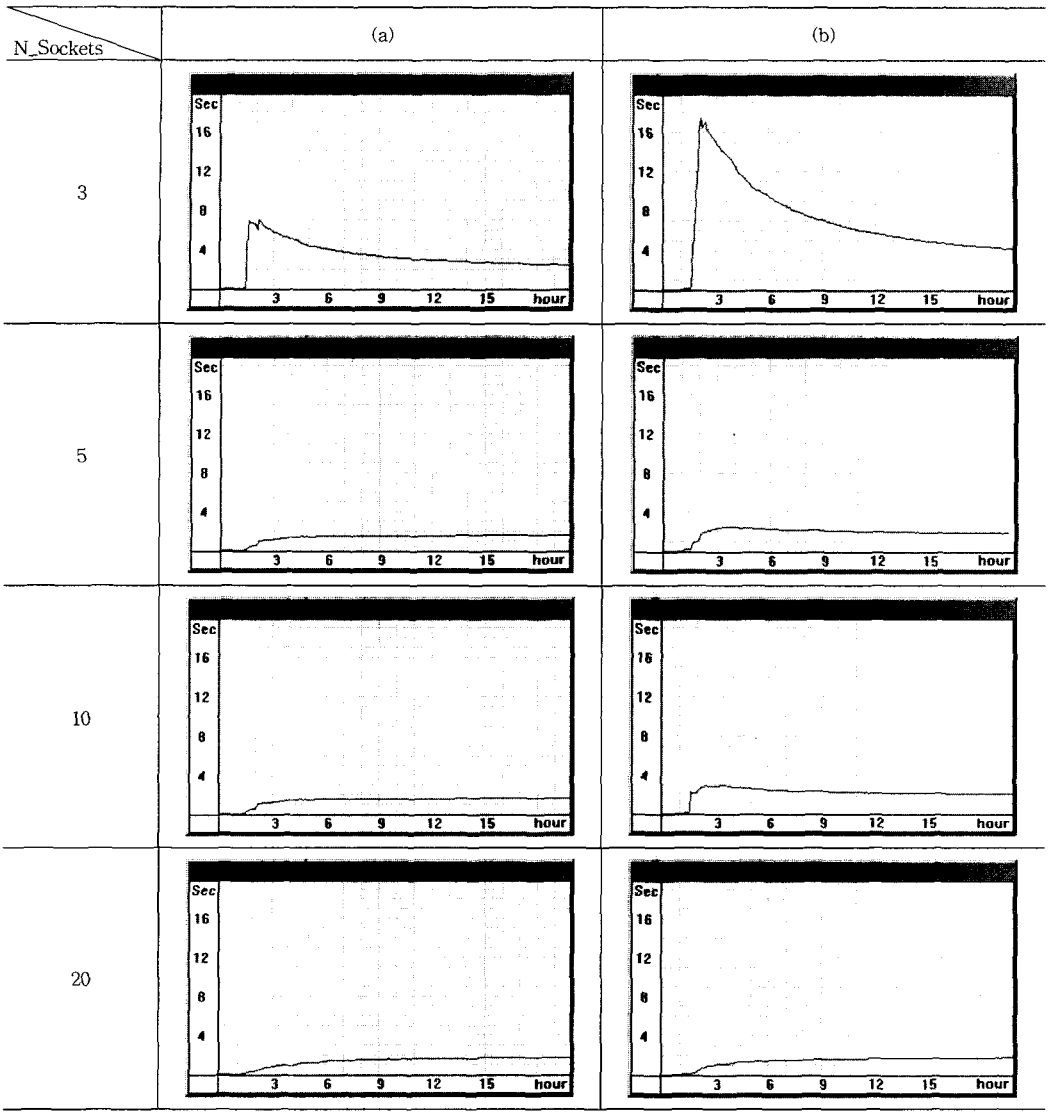


그림 9 N_Sockets와 네트워크 그룹 참여를 위해 소요되는 시간과의 관계

6. 결론 및 향후 연구과제

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 목록으로 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 플랫폼(agent platform)으로 하여금 에이전트를 이주(migration)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.

참 고 문 헌

[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, 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.
[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," 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.

김 영 진

2002년 대전대학교 컴퓨터공학과 졸업(학사). 2002년~현재 성균관대학교 대학원 정보통신공학부 석사과정. 관심분야는 P2P, 이동 에이전트, 분산 컴퓨팅 등

엄 영 익

1983년 2월 서울대학교 계산통계학과 졸업(학사). 1985년 2월 서울대학교 대학원 전산과학과 졸업(석사). 1991년 8월 서울대학교 대학원 전산과학과 졸업(박사) 2000년 9월~2001년 8월 Dept. of Info. and Comm. Science at UCI 방문교수 1993년 3월~현재 성균관대학교 정보통신공학부 교수. 관심분야는 분산 컴퓨팅, 이동 컴퓨팅 시스템, 이동 에이전트, 시스템 소프트웨어 등