순수 P2P 네트워크 환경에서의 효율적인 피어 고립 방지 기법

김 영 진† · 엄 영 익††

요 약

각 사용자들이 자료를 직접 주고받을 수 있는 P2P 네트워킹 방식에는 하이브리드 P2P 네트워킹 방식과 순수 P2P 네트워킹 방식이 있다. 하이브리드 P2P 네트워킹 방식에서는 각 피어들로 하여금 서버에 연결하여 서비스를 제공받게 하기 때문에, 서버가 운영되고 있는 한 각 피어들은 서버로부터 지속적인 서비스를 받을 수 있으며 다른 피어들간의 통신이 단절되는 네트워크 고립 현상이 발생되지 않는다. 그러나 순수 P2P 네트워킹 방식에서는 각 피어들이 서버가 아닌 다른 피어에게로 연결하여 서비스를 제공받게 함으로써, 피어들간의 통신을 중재하는 특정 피어가 종료할 경우 피어들간에 통신을 할 수 없는 네트워크 고립 현상이 발생될 수 있다. 본 논문에서는 이러한 문제점을 해결하기 위해 각 피어들로 하여금 인접한 피어들의 연결 상태를 관리하게 함으로써, 연결된 피어가 종료할 경우 종료한 피어의 인접 피어에게 연결하는 기법과 이때 특정 피어에게 집중될 수 있는 연결들을 제거하거나 다른 피어로 분산하는 기법을 제시한다. 그에 따라 이 제안기법은 각 피어들로 하여금 연결된 특정 피어가 종료되어도 다른 피어에게 연결하여 네트워크 그룹에 지속적으로 연결을 유지하게 함으로써, 특정 피어가 네트워크 그룹으로부터 고립되어 서비스를 제공받지 못하는 현상을 방지할 수 있다. 시뮬레이션을 통하여 각 피어들이 네트워크 그룹으로의 고립이 발생하지 않는 상태에서 네트워크 연결을 분산시키는 것을 확인하였고, 피어 하나당 권장 연결 수에 따른 전체 네트워크 트래픽 양과 권장 연결 수에 따른 최대 홉 수를 측정하여 비교하였다.

An Efficient Peer Isolation Prevention Scheme in Pure P2P Network Environments

Young-jin Kim† · Young Ik Eom††

ABSTRACT

According to the arbitration mechanism among the peers in the network, the P2P networking environments can be classified into hybrid P2P networking environments and pure P2P networking environments. In hybrid P2P networking environments, each peer gets continual services from the servers that are operational most of the time, and so, network isolation does not occur because every peer can always keep connection to the server. In pure P2P networking environments, however, every peer directly connects to another peer without server intervention, and so, network isolation can occur when the peer mediating the connection is terminated. In this paper, we propose a scheme for each peer to keep connection information of other peers by maintaining IDs of its neighbor peers, to reconnect to another peers when the mediating peer fails to work, and, for efficiency, to balance the number of connections that should be maintained by each peer. With our mechanism, each peer in the network can continuously maintain connection to the network and get seamless services from other peers. Through the simulation, we ascertained that network isolation does not occur in the pure P2P network adopting our mechanism and that our mechanism distributes and balances the connections that are maintained by each peer. We also analyzed the total network traffic and the mean number of hops for the connections made by each peer according to the recommended number of connections that is established at system setup time.

키워드 : P2P 네트워크(P2P Network), 분산 컴퓨팅(Distributed Computing), 네트워크 고립(Network Isolation), 분산 연결(Distributed Connection)

1. 서 론

P2P 네트워킹 방식은 네트워크 그룹 유지를 위한 서버의 유무에 따라 하이브리드 P2P(hybrid P2P) 네트워킹 방식과 순수 P2P(pure P2P) 네트워킹 방식으로 구분될 수 있다. 하이브리드 P2P 네트워킹 방식에서 각 피어들은 서비스를 제공받기 위해 서버와의 네트워크 연결을 유지하며 운영되기 때문에 다른 피어들의 ID 목록을 관리할 필요가 없다. 또한 이 방식은 모든 클라이언트들이 서버에 연결되어 있기 때문에 특정 피어에게만 서비스가 제공되지 않는 네트워크 고립 현상이 발생하지 않는다. 그러나 서버는 연결되어 있는 모든 클라이언트들의 메시지를 중재하기 때문에 많은

※ 이 논문은 2003년도 한국학술진흥재단의 지원에 의하여 연구되었음(KRF 2003 041 D20420).
† 준 회 원 : 삼성전자 소프트웨어센터 연구원
†† 종신회원 : 성균관대학교 정보통신공학부 교수(교신저자)
논문접수 : 2003년 11월 13일, 심사완료 : 2004년 8월 31일

네트워크 대역폭을 필요로 하게 된다. 또한 각 피어들은 단 하나뿐인 서버와의 연결이 끊어지게 될 경우 다른 피어들로부터 어떠한 서비스도 제공받지 못하게 되는 문제점을 가지고 있다[1-3]. 반면 순수 P2P 네트워킹 방식에서 각 피어들은 다른 피어들에게 연결하여 서비스를 제공받기 때문에 별도의 서버를 필요로 하지 않는다. 이러한 방식은 각 피어들로 하여금 하이브리드 P2P 네트워킹 방식에서 서버에게 발생되었던 네트워크 트래픽을 여러 피어들에게 분산시키게 함으로써, 특정 피어에게 네트워크 트래픽이 집중되지 않게 한다. 그러나 각 피어들은 다른 피어들의 통신을 중재하는 역할을 하고 있기 때문에 피어들의 통신을 중재하는 특정 피어가 종료할 경우 대상 피어들간의 통신이 단절되는 네트워크 고립 현상이 발생될 수 있다[4].

본 논문에서는 각 피어들에게 발생할 수 있는 네트워크 고립 현상을 해결하기 위해 각 피어들로 하여금 인접한 피어로부터 ID를 확보하여 목록으로 관리하게 함으로써, 연결된 피어가 종료할 경우 종료한 피어에 연결되어 있던 다른 피어로 연결하게 하는 기법을 제시한다. 이와 더불어 종료된 피어에 인접해 있던 피어들이 다른 피어에게 연결함으로써 발생하는 많은 네트워크 연결들을 효율적으로 정리하는 기법과 연결이 집중될 경우 주위 피어에게 연결을 분산하는 기법을 제안하고자 한다. 이 제안기법을 이용한 P2P 응용 프로그램은 다른 피어의 ID를 확보한 후 연결된 피어가 종료할 시 그 피어에 인접하였던 피어로 연결하기 때문에 네트워크 그룹으로부터 고립되지 않고 여러 가지 서비스를 지속적으로 제공받을 수 있게 된다.

본 논문의 구성은 다음과 같다. 2장에서 기존의 P2P 네트워킹 방식에 대한 관련연구를 소개한다. 3장에서는 제안기법의 동작과정과 시나리오를 제시한다. 4장에서는 제안한 시스템의 시뮬레이션 과정을 보이고, 마지막으로 5장에서 결론 및 향후 연구를 살펴보도록 한다.

2. 관련 연구

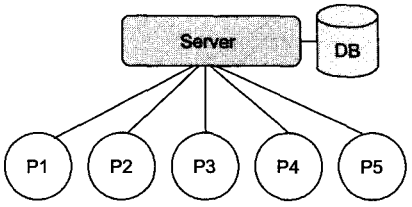
P2P 네트워킹 방식에서 실질적인 자원의 송수신은 피어들간 직접적인 연결을 통해 이루어지며, 여러 가지 정보를 제공하는 서버의 유무에 따라 하이브리드 P2P 네트워킹 방식과 순수 P2P 네트워킹 방식으로 나뉜다. 본 장에서는 이러한 P2P 네트워킹 방식들과 문제점들을 알아보도록 한다.

2.1 하이브리드 P2P 네트워킹 방식

하이브리드 P2P 네트워킹 방식에서 피어들간 자원의 송수신은 다른 피어로 직접 연결하여 이루어지지만, 각 피어들의 운영에 필요한 정보들은 서버로부터 제공받는다. 이 방식은 각 피어들이 보유한 파일들의 정보를 서버가 가지고 있는 Napster 방식과 실행되어 있는 피어들과의 연결상태만을 유지하는 소리바다 방식으로 나뉜다.

2.1.1 Napster 방식

각 피어들은 특정 서버에 접속한 후 사용자가 지정한 공유 파일의 정보를 서버에 전송하게 되며, 이를 받은 서버는 각 피어들의 파일 목록을 DB에 저장하여 관리하게 된다. 이후 임의의 사용자가 특정 파일을 찾기 위해 검색 메시지를 서버에 전송하면, 서버는 DB에 저장된 파일 목록에서 사용자가 원하는 파일을 검색한 후 검색된 파일의 파일명, 크기, 소유자 등 여러 정보를 자료를 요청한 피어에게 전달하게 된다. 따라서 서버로부터 검색 결과를 받은 피어는 메시지 내에 지정된 피어에게 직접 연결함으로써 원하는 파일을 수신할 수 있게 된다[5-7]. (그림 1)에서는 Napster 방식의 네트워크 구조를 보인다.



(그림 1) Napster 방식의 네트워크 구조

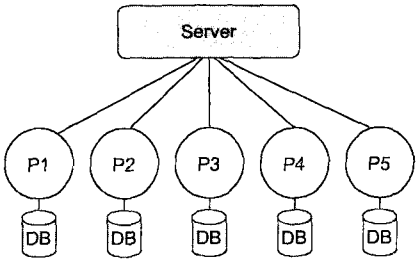
이러한 방식에서 서버는 각 피어들이 보유한 파일들의 목록만을 관리하고 있기 때문에, 기존 클라이언트/서버 구조에서 사용자들이 특정 파일을 서버에서 받아감으로써 서버에 집중되는 네트워크 트래픽을 일반 피어들에게 분산시켜 자원을 빠른 속도로 공유할 수 있다. 또한 서버가 모든 피어들과 연결되어 있기 때문에, 서버가 종료되지 않는다면 특정 피어가 다른 피어들과 통신을 하지 못하는 네트워크 고립 현상이 발생되지 않는다. 그러나 모든 피어들은 자원을 검색하기 위해 서버와의 연결을 항상 유지해야 하고 피어들이 공유하고 있는 파일 목록을 서버로 전송해야 하기 때문에, 전형적인 클라이언트/서버 방식에서처럼 서버에게 네트워크 트래픽이 집중되는 문제가 여전히 발생하게 된다. 또한 피어들은 파일 목록을 관리하는 서버가 종료된다면 서비스를 제공받을 수 없다는 단점을 가지고 있다.

2.1.2 소리바다 방식

소리바다 방식이 Napster 방식과 다른 점은 각 피어들로 하여금 파일에 대한 어떠한 정보도 서버에게 보내지 않게 한다는 것이다. (그림 2)에서는 소리바다 방식의 네트워크 구조를 보인다.

파일 검색을 원하는 피어가 서버에게 파일 검색 메시지를 전송하면 서버는 연결되어 있는 피어들 모두에게 검색 요청 메시지를 전송하게 된다. 이 메시지를 받은 피어들 중 해당 파일을 가지고 있는 피어는 파일 정보를 서버에게 전송하게 되며, 이 정보를 받은 서버는 최초 자료 검색을 요청한 피어에게 파일 정보가 담긴 응답 메시지를 전송하게

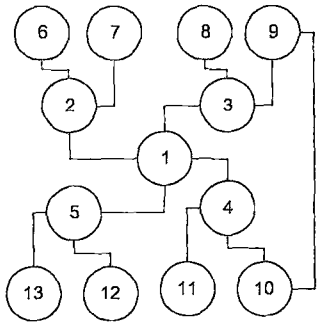
된다[8]. 그러나 이러한 방식 또한 서버가 피어들의 모든 요청과 답변을 중재하기 때문에 네트워크 트래픽이 여전히 서버에게 집중되게 된다. 그리고 기존의 방식에서처럼 서버 의존적인 연결 방식이어서, 서버가 종료되면 피어들은 서비스를 제공받을 수 없게 된다.



(그림 2) 소리바다 방식의 네트워크 구조

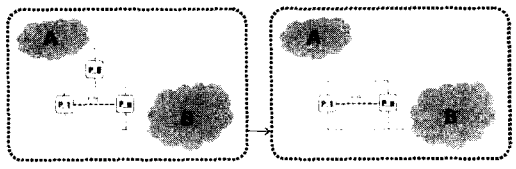
2.2 순수 P2P 네트워킹 방식

이 방식은 피어들에게 서비스를 제공하는 특정 서버를 두지 않고, 모든 피어들로 하여금 서버와 클라이언트의 기능을 동시에 가지게 하는 방식으로, Gnutella가 대표적이다[9-11]. (그림 3)에서는 Gnutella 방식의 네트워크 구조를 보인다.



(그림 3) Gnutella 방식 네트워크 구조

Gnutella 방식을 사용한 프로그램들은 최초 실행시 Gnutella 네트워크 그룹에 이미 참여하고 있는 피어에게 연결하게 되며, 이러한 방식으로 모든 피어들이 피라미드식으로 연결되어 무제한의 자료를 공유하게 된다. 그러나 Gnutella 네트워크 환경에서는 피어가 종료될 경우 네트워크 고립 현상이 발생할 수 있으며, 이러한 현상을 (그림 4)에서 보인다.



(a) 피어 종료 전 (b) 피어 종료 후

(그림 4) 피어 종료로 인한 네트워크 그룹 분리 상태

(그림 4)(a)에서 모든 피어는 하나의 네트워크 그룹으로 연결되어져 있다. 그러나 (그림 4)(b)에서와 같이, 피어 P0이 종료되면서 단일 네트워크 그룹이 두개의 네트워크 그룹으로 나눠지게 되었다. 따라서 특정 피어의 종료로 인해 나눠진 두 네트워크 그룹 내의 피어들간에는 어떠한 메시지도 전달할 수 없게 된다.

3. 피어 고립 방지 기법

본 장에서는 순수 P2P 네트워크 환경에서 네트워크 연결이 특정 피어에게 집중되지 않고 여러 피어에게 분산되게 함으로써 각 피어들로 하여금 네트워크 그룹에 안정적으로 참여할 수 있게 하는 기법을 소개한다.

3.1 시스템 모델

순수 P2P 네트워크 환경에서 동작하는 각 피어들이 다른 피어들과 원활한 연결을 유지하며 동작하기 위해서는 여러 가지 기법들이 필요하게 되며, 이와 관련하여 본 논문에서는 다음과 같은 가정을 한다.

- 모든 피어는 동일한 포트를 개방하고 있다.
- 각 피어에서 동작하는 프로그램에는 하나 이상의 피어 ID 정보가 이미 존재한다.
- 모든 피어는 방화벽/NAT 네트워크 환경 내에 있지 않다.
- 두 피어가 서로 연결할 경우 어느 피어가 먼저 접속을 시도하였는지 알 수 있다.
- 두 피어가 종료할 경우 어느 피어가 먼저 종료하였는지 알 수 있다.

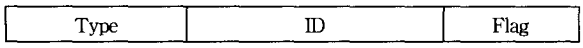
본 논문에서는 이러한 가정을 기반으로 하여 각 피어들의 연결과정을 3단계, 즉, 다른 피어에게 연결한 후 ID들을 수신하는 **연결/수신 단계**, 다른 피어로부터 네트워크 접속(connect) 또는 단절(close)이 발생할 때마다 인접한 피어들에게 알리는 **연결정보 전송 단계**, 그리고 생성된 연결 수가 인접한 피어의 평균 연결 수보다 많을 경우 인접 피어들에게 연결을 분산하는 **연결 분산 단계**로 나누어 설명한다.

연결/수신 단계에서는 지정된 ID를 사용하는 피어로 연결한 후 해당 피어로부터 새로운 ID들을 수신하게 된다. 수신된 ID들은 특정 형태로 ID 테이블 내에 저장되며, ID를 수신한 피어의 연결 수가 권장 연결 수(N_Connections)보다 작을 경우 수신된 ID로 네트워크 연결을 시도하여 네트워크 연결 수가 권장 연결 수만큼 증가되도록 한다. **연결정보 전송 단계**에서는 다른 피어로의 접속, 다른 피어로부터의 접속, 다른 피어의 종료로 인한 연결 끊어짐, 그리고 연결되어 있는 상대방 피어의 요청에 의한 연결 끊어짐 등 4가지 사건이 발생할 경우 각각에 대해 연결되어 있는 피어들에게 사건 정보를 전달하게 된다. 마지막으로 연결 분산 단계에서는 생성된 연결 수가 N_Connections보다 많고

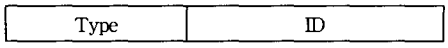
연결된 주위 피어들의 평균 연결 수와 1 이상 차이가 나는 경우 해당 피어는 연결되어 있는 피어 중 연결 수가 가장 많은 피어에게 연결 수가 가장 적은 피어로 접속하도록 메시지를 전달하여 연결되어 있는 주위 피어들의 네트워크 연결을 분산시킨다.

3.2 패킷 형식 및 알고리즘

본 논문에서 제안하는 기법은 연결/끊어짐 이벤트 패킷(CEP), 연결 지시 패킷(COP)을 사용하며, 그 형식은 (그림 5)에서 보인다.



(a) CEP(Connect/close Event Packet)



(b) COP(Connect Order Packet)

(그림 5) 패킷 형식

(그림 5)에서 각 패킷들은 Type 필드 값에 의해 메시지의 형식이 구분된다. CEP 메시지의 ID 필드는 연결되거나 끊어진 다른 피어의 ID를 갖게 되며, Flag의 값에는 0과 1이 설정되어져 각각 끊어짐과 연결됨의 ID 상태 정보를 표현하게 된다. COP 메시지의 ID 필드는 새롭게 연결해야 할 피어의 ID가 담겨져 있다.

CEP 메시지를 수신한 피어는 메시지 내의 ID를 피어 관리 버퍼(BMIP : Buffer for Managing Information of Peers)에 저장 또는 제거하며, 저장되는 요소는 <표 1>에서 보인다.

〈표 1〉 피어 관리 버퍼의 구성

명 칭	내 용
NeighborID	이웃한 피어의 ID
IDs	NeighborID의 피어와 연결되어 있는 피어들의 ID

다른 피어로부터 CEP 메시지를 수신한 피어는 메시지 내의 Flag 값이 1로 설정되어 있다면 메시지 내의 ID를 BMIP에 저장하게 되며, Flag 값이 0으로 설정되어 있다면 BMIP 내에 있는 정보 중 CEP 메시지 내의 정보와 관련된 엔트리를 삭제하게 된다.

본 논문에서 제안하는 기법은 지정된 ID를 가진 피어에게 연결한 후, 연결된 피어로부터 ID들을 수신하는 **연결/수신 단계**, 다른 피어와 연결 및 끊어짐 이벤트가 발생할 경우 연결된 피어에게 CEP 메시지를 전송하는 **연결정보 전송 단계**, 그리고 생성된 연결 수가 연결된 피어들의 평균 연결 수보다 많을 경우 연결을 분산하는 **연결 분산 단계**로 구성되며, 본 절에서는 각 단계별로 세부 알고리즘을 설명한다.

3.2.1 연결/수신 단계

각 피어에서 프로그램이 실행될 경우 해당 프로그램은 지정된 ID로 접속을 시도하게 되며, 접속 요청을 받은 피어는 자신에게 연결되어 있는 피어들의 ID를 접속 시도한 피어에게 전달하게 된다. 다른 피어로부터 접속 요청을 받은 피어의 실행 과정을 (알고리즘 1)에서 보인다.

```
input : IDR(ID of the peer that requests the ID list)
output : none
{
   CEP cep ; /* CEP message type */
   ENTRY ety ; /* BMIP entry type */

   cep.Flag = 1 ;
   cep.ID = ID of the connected peers
   send the cep to the peer that requested the ID list ;

   cep.ID = IDR ;
   send the cep to the connected peers ;

   ety = new ENTRY ;
   ety.NeighborID = IDR ;
   add the ety into the BMIP ;
}
```

(알고리즘 1) 다른 피어로부터 접속 요청을 받은 피어의 실행 과정

피어들은 다른 피어로부터 네트워크 접속 요청을 받았을 경우 연결되어 있는 피어들의 ID를 CEP 메시지에 포함시킨 후 접속한 피어에게 전송하여 준다. 또한 이미 연결되어 있는 피어들에게는 접속한 피어의 ID를 CEP 메시지에 포함시켜 전송한다. CEP 메시지를 수신한 피어의 실행 과정은 (알고리즘 2)에서 보인다.

```
input : cep(CEP message), IDS(ID of the peer that sends the cep message)
output : none
{
   ENTRY ety=get an entry that (NeighborID field value of the
   entry in BMIP == IDS) ;
   switch(cep.Flag) {
   case 0 :
      if (ety)
         remove cep.ID from IDs in the ety ;
      break ;
   case 1 :
      if (ety == NULL) {
         ety = new ENTRY ;
         ety.NeighborID = IDS ;
         add the ety into the BMIP ;
      }
      if (the ety doesn't include cep.ID)
         insert cep.ID into the ety ;
      if (the number of the connected peers<N_Connections &&
         not connected with the peer that uses cep.ID) {
         SOCKET sock ;
         sock = connect(cep.ID) ;
         if (sock != NULL )
            send the cep to the connected peers ;
      }
      break ;
   }
}
```

(알고리즘 2) CEP 메시지를 수신한 피어의 실행 과정

다른 피어로부터 CEP 메시지를 받은 피어는 메시지를 전송한 피어의 ID에 해당하는 엔트리를 얻은 후, CEP 메시지 내의 Flag 값이 0일 경우 엔트리의 IDs 필드에 포함된 ID들 중 메시지 내에 포함된 ID와 일치하는 값을 제거한다. 그리고 Flag 값이 1일 경우 메시지 내에 설정되어 있는 ID를 해당 엔트리의 IDs 필드에 추가하게 되며, 이후 연결되어 있는 연결 수가 N_Connections보다 작고 CEP 메시지 내에 있는 ID로 연결되어 있지 않을 경우 CEP 메시지 내에 있는 ID로 연결을 시도하게 된다. 각 피어들은 CEP 메시지를 수신한 후 필요시 다른 피어에게 연결함으로써 항상 N_Connections 이상의 네트워크 연결을 유지할 수 있게 되어 네트워크 그룹으로부터 고립되지 않게 된다.

3.2.2 연결정보 전송 단계

다른 피어로부터 연결이 끊어진 피어의 실행 과정을 (알고리즘 3)에서 보인다.

```
input : ID of the disconnected peer
output : none
{
    CEP cep ;
    cep.ID = ID of the disconnected peer ;
    cep.Flag = 0 ;
    send the cep to the connected peers
    remove an entry that (NeighborID) of the entry in BMIP == ID
    of the disconnected peer) ;
}
```

(알고리즘 3) 네트워크 연결이 끊어진 피어의 실행 과정

특정 피어와 다른 피어와의 연결이 끊어지게 되는 경우, 끊어진 피어의 ID가 포함된 CEP 메시지를 연결된 모든 피어들에게 전송한다. 또한, 연결이 끊어진 피어의 ID에 해당하는 엔트리를 BMIP 내에서 삭제한다.

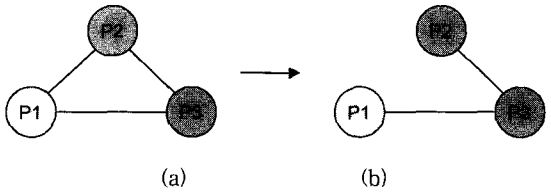
3.2.3 연결 분산 단계

연결 분산 단계는 각 피어들이 연결 수를 줄일 경우 발생할 수 있는 네트워크 고립 현상을 방지하면서 각 피어들이 보유한 연결 수를 평준화하는 과정이다. 네트워크 고립 상태가 발생하지 않도록 하면서 피어의 연결 수를 줄이는 과정은 (알고리즘 4)에서 보인다.

```
input : none
output : none
{
    for (a = (the number of the connected peers)-1 ; a >= 0 ; a--) {
    ID_id = get ID of a th connected peer ;
    if (_id is in IDs of the connected peer &&
      (the number of the connected peers) > N_Connections) {
       disconnect the peer that uses_id ;
      }
    }
}
```

(알고리즘 4) 피어간 연결 수를 줄이는 과정

이 작업은 다른 피어의 종료로 인하여 연결이 끊어질 경우 또 다른 피어들에게 연결하는 알고리즘이다. (알고리즘 4)의 동작 과정에 대한 예를 (그림 6)에서 보인다.



(a) (b)

(그림 6) 피어간 네트워크 연결 정리 단계(N_Connections : 1)

(그림 6)(a)에서는 피어 P1, P2, P3이 서로 다른 피어에게 모두 연결되어 있음으로써 각 피어들이 2개의 연결을 보유하고 있는 모습을 보여주고 있다. 이 피어들은 N_Connections의 값이 1로 설정되어 있기 때문에 네트워크 연결 수를 1로 유지하는 작업을 수행하게 된다. (그림 6)(a)에서, 피어 P1이 피어 P2에게 데이터를 전송하는 방법에는 직접 전송하는 방법과 피어 P3을 경유하여 전송하는 방법이 있다. 이때, 피어 P1은 피어 P3으로 하여금 피어 P1의 데이터를 피어 P2에게 전송하도록 지시할 수 있음을 고려하여 피어 P2와의 연결을 끊게 된다. 이후 (그림 6)(b)에서 피어 P2는 연결 수가 N_Connections를 만족하기 때문에 네트워크 연결을 더 이상 끊지 않는다. 피어 P3은 비록 네트워크 연결 수가 2이지만 피어 P1과 피어 P2에게 데이터를 전송할 있는 다른 경로가 존재하지 않기 때문에 어떠한 피어와도 연결을 끊지 않는다. 피어에 생성된 연결 수가 연결된 인접 피어들의 평균 연결 수보다 클 경우 네트워크 연결 분산 작업을 하게 되며, 그 과정은 (알고리즘 5)에서 보인다.

```
input : none
output : none
{
    LBMIP = the number of entries in BMIP ;
    sort BMIP by the number of IDs in entry ;
    total = the number of the connected peers ;
    for (index = 0 ; index < LBMIP; index++) {
       ENTRY ety = get index-th entry of BMIP ;
       total += the number of IDs in ety ;
    }
    avg = total/((size of BMIP)+1) ;
    dsn = (int)((the number of the connected peers) - avg) ;
    for (index = 0 ; index < dsn ; index++) {
         int reverseIndex = LBMIP - index - 1 ;
         ENTRY ety1 = get index-th entry from BMIP ;
         ENTRY ety2 = get reverseIndex-th entry from BMIP ;
         COP cop ;
         cop.ID = ety2.NeighborID ;
         send the cop to the peer that uses ety1.NeighborID ;
         disconnect the peer that uses ety1.NeighborID ;
         remove the ety1 from BMIP ;
    }
}
```

(알고리즘 5) 네트워크 연결의 분산 과정

피어는 연결 분산 단계를 처리하기 위해 BMIP 내의 정보들을 IDs에 포함된 ID 개수 순으로 정렬한다. 이후 인접한 임의 피어의 연결 수가 인접한 피어들의 평균 연결 수보다 많을 경우, 피어는 연결 수가 적은 피어로 네트워크 연결을 할 수 있도록 연결 수가 많은 피어에게 COP 메시지를 전달하게 된다. COP 메시지를 받은 피어의 동작 과정을 (알고리즘 6)에서 보인다.

```
input : IDS(ID of the peer that sends the message), cop(COP message)
output : none
{
    disconnect the peer that sends COP message ;
    remove the entry that NeighborID field at BMIP is IDS ;

    CEP cep ;
    cep.ID = IDS ;
    cep.Flag = 0 ;
    send the cep to the connected peers ;

    connect to the peer that uses cop.ID ;

    cep.ID = cop.ID ;
    cep.Flag = 1 ;
    send the cep to the connected peers ;
}
```

(알고리즘 6) COP 메시지를 수신한 피어의 동작 과정

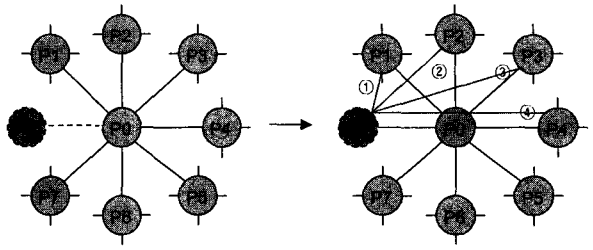
COP 메시지를 받은 피어는 COP 메시지를 보낸 피어와의 연결을 끊은 후 COP 메시지 내에 지정되어 있는 ID로 연결을 시도한다. 또한, 주위에 연결되어 있는 모든 피어들에게 끊어진 피어의 ID와 새로 연결한 피어의 ID를 각각 CEP 메시지에 저장하여 전달한다.

3.3 시나리오

본 시나리오는 임의의 피어가 다른 피어로 접속하는 경우, 연결되어 있는 피어와 연결이 끊어지는 경우, 그리고 네트워크 연결이 특정 피어에게 집중되는 경우로 분류될 수 있다.

3.3.1 임의의 피어가 다른 피어로 접속하는 경우

임의 피어가 다른 피어로 접속할 경우의 모습을 (그림 7)에서 보인다. (그림 7)(a)에서, 각 피어들을 연결하는 선은 피어들간 네트워크 연결을 의미하며, 피어 P8이 피어 P0에게 접속하기 전에 피어 P0은 7개의 연결, 피어 P1, P3, P5, P7은 5개의 연결, 피어 P2, P4, P6은 4개의 연결을 생성하고 있는 모습을 보인다. 피어 P8은 실행시 프로그램 내에 미리 지정되어 있던 피어 P0으로 네트워크 연결을 시도한다. 피어 P8과 연결된 피어 P0은 이미 연결되어 있는 다른 피어들의 ID들을 피어 P8에게 전송하게 되고, 피어 P8은 피어 P0이 송신한 피어 P1, P2, P3, P4, P5, P6, P7의 ID들을 수신하게 된다. (그림 7)(b)에서, 피어 P0으로부터 주소 정보를 수신한 피어 P8은 네트워크 연결 수가 N_Connections가 될 때까지 네트워크 연결을 시도하게 되며, 본 그림에서는 ①~④ 순서로 각 피어들에게 접속함을 보인다.

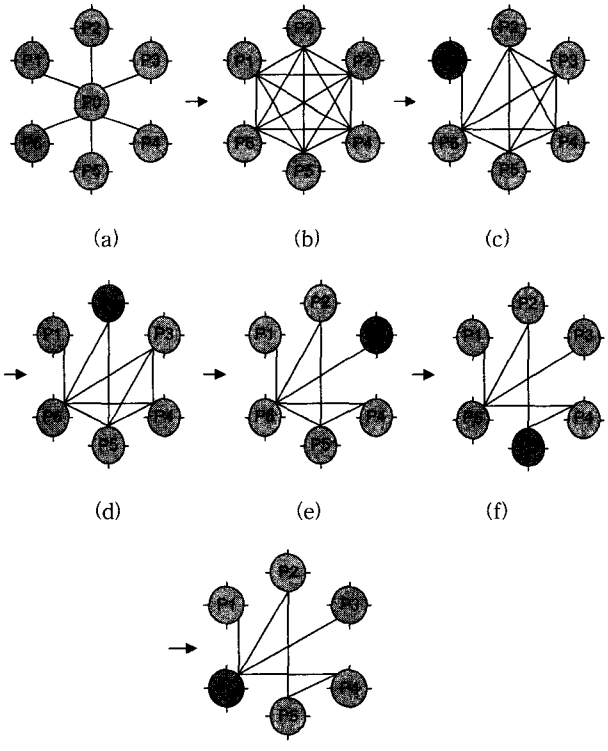


(a) 최초 연결 단계 (b) 피어 연결단계

(그림 7) 다른 피어로 접속시의 네트워크 연결 생성 단계 (N_Connections : 5)

3.3.2 연결되어 있는 피어와 연결이 끊어지는 경우

다른 피어로부터 네트워크 연결이 단절된 후 연결을 복원하는 과정을 (그림 8)에서 보인다. (그림 8)(a)에서, 피어 P0은 피어 P1, P2, P3, P4, P5, P6과 연결되어 있다. 그러나 (그림 8)(b)에서, 피어 P0의 종료로 인해 피어 P1, P2, P3, P4, P5, P6들은 피어 P0에 연결되어 있던 모든 피어들에게 접속을 시도하게 된다. (그림 8)(c)~(그림 8)(g)에서 피어 P0에 연결되어 있던 피어들은 연결 수를 줄이기 위해 각각 연결 분산 단계를 거치게 되며, 편의상 P1→P2→P3→P4→P5→P6 순서로 진행하도록 한다.



(그림 8) 피어 종료로 인한 네트워크 연결 단계 (N_Connections : 5)

(그림 8)(b)에서, 피어 P1은 피어 P0의 종료로 인해 피어 P2, P3, P4, P5, P6과 연결한 상태를 보인다. 피어 P1이 피어 P2에게 데이터를 송신하기 원할 경우 이 데이터는 피어 P3, P4, P5, P6 중 한 피어를 경유하여 전달될 수 있기 때문에 피어 P1은 피어 P2와의 네트워크 연결을 끊을 수 있다. 또한, 피어 P2와의 네트워크 연결을 끊은 후 피어 P1이 피어 P3에게 데이터를 송신하기 원할 경우 피어 P4, P5, P6 중 한 피어를 통해 데이터를 전송할 수 있기 때문에 피어 P1은 피어 P3과의 네트워크 연결을 끊을 수 있다. 이와 동일한 방법으로 피어 P1은 피어 P4에게 데이터를 송신하기 원할 경우 피어 P5, P6 중 한 피어를 통해, 피어 P5에게 데이터를 송신하기 원할 경우 피어 P6을 통해 데이터를 전송할 수 있다. 결국 (그림 8)(c)에서 보이는 바와 같이, 피어 P1은 연결 수가 N_Connections보다 작아지지 않게 유지하면서 제거 가능한 네트워크 연결을 끊게 된다.

(그림 8)(c)에서, 피어 P2는 피어 P3, P4, P5, P6과 연결되어 있다. 이 경우도 피어 P2가 피어 P3에게 데이터를 송신하기 원할 경우 P4, P5, P6 중 한 피어를 통해, 피어 P4에게 데이터를 전송하기 원할 경우 피어 P5, P6 중 한 피어를 통해 데이터를 전송할 수가 있다. 따라서 (그림 8)(d)와 같이, 피어 P2는 피어 P3과 피어 P4와의 네트워크 연결을 끊는다. 비록 피어 P2가 피어 P5에게 데이터를 전송하기 원할 경우 피어 P6을 경유하여 데이터를 전송할 수 있지만, 피어 P5와의 연결을 끊는다면 피어 P2의 연결 수가 N_Connections수인 5보다 작아지기 때문에 더 이상 연결을 끊을 수 없다.

(그림 8)(d)에서, 피어 P3은 피어 P4, P5, P6과 연결되어 있다. 이 경우도 피어 P3이 피어 P4에게 데이터를 송신하기 원할 경우 피어 P5, P6 중 한 피어를 통해, 피어 P5에게 데이터를 전송하기 원할 경우 피어 P6을 통해 데이터를 전송할 수 있다. 따라서 (그림 8)(e)와 같이, 피어 P3은 피어 P4와 피어 P5와의 네트워크 연결을 끊는다.

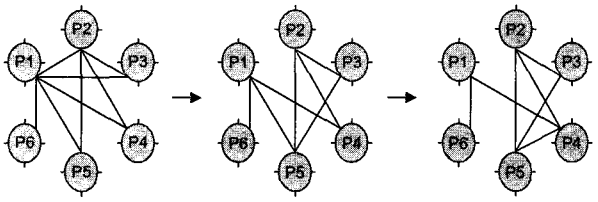
(그림 8)(e)에서, 피어 P4는 연결 수가 N_Connections인 5와 동일하기 때문에 피어 P5와 피어 P6과의 네트워크 연결을 끊을 수 없다. 피어 P5는 피어 P6에게 데이터를 송신하기 원할 경우 피어 P2를 통해 데이터를 전송할 수 있기 때문에 피어 P6과의 네트워크 연결을 끊는다.

(그림 8)(f)에서 피어 P6은 더 이상 연결을 끊을 피어가 없게 되며, (그림 8)(g)에서 네트워크 연결 최소화 단계를 모두 거치게 된 피어간의 네트워크 연결 모습을 보인다.

3.3.3 네트워크 연결이 특정 피어에게 집중되는 경우

각 피어들은 생성된 연결 수가 연결된 인접 피어들의 평균 연결 수보다 많아지게 되면 네트워크 연결을 다른 피어로 분산하는 작업을 수행하게 된다. 네트워크 연결을 다른 피어로 분산하는 과정을 (그림 9)에서 보인다.

(그림 9)(a)에서의 각 피어별 연결 정보는 <표 2>에서 보인다.



(그림 9) 네트워크 연결을 주위 피어로 분산하는 과정 (N_Connections : 5)

<표 2> 피어별 연결 정보

피어	연결된 피어	연결 수
P1	P2, P3, P4, P5, P6외 4개	9
P2	P1, P3, P4, P5외 3개	7
P3	P1, P2외 4개	6
P4	P1, P2외 3개	5
P5	P1, P2외 3개	5
P6	P1외 4개	5

피어 P1은 N_Connections보다 많은 연결 수를 가지고 있기 때문에, 연결 분산 단계를 실행하게 된다. 피어 P1을 포함하여 피어 P1에 연결된 인접 피어들의 연결 수 평균은 약 6.16((9+7+6+5+5+5)/6)이며, 피어 P1은 연결 수 평균보다 9-6.16인 약 2개의 연결이 더 연결되어 있다. 따라서 피어 P1은 가장 많은 연결 수를 가지고 있는 피어 P2, P3이 연결 수가 가장 적은 피어 P4, P5로 연결하도록 피어 P2, P3에게 COP 메시지를 전송한 후, 피어 P2와 피어 P3과의 네트워크 연결을 끊는다.

(그림 9)(b)에서는 COP 메시지를 받은 피어 P2와 피어 P3이 다른 피어에게 연결을 재설정한 모습을 보인다. 피어 P2는 피어 P4에게 연결을 시도하라는 COP 메시지를 피어 P1로부터 받지만, 이미 피어 P4와 연결되어 있기 때문에 연결을 시도하지 않는다. 피어 P3은 피어 P5와의 연결을 시도하라는 COP 메시지를 피어 P1로부터 받은 후, 피어 P5에게 연결을 시도하게 된다. (그림 9)(b)에서의 각 피어별 연결 정보는 <표 3>에서 보인다.

<표 3> 네트워크 연결 분산 후 피어별 연결 정보

피어	연결된 피어	연결 수
P1	P4, P5, P6외 4개	7
P4	P1, P2외 3개	5
P5	P1, P2, P3외 3개	6
P6	P1외 4개	5

피어 P1을 포함하여 피어 P1에 연결된 인접 피어 P4, P5, P6의 연결 수 평균은 (7+5+6+5)/4인 6이 되었지만, 아직까지 피어 P1은 평균보다 1만큼 연결 수가 더 많다. 따라서

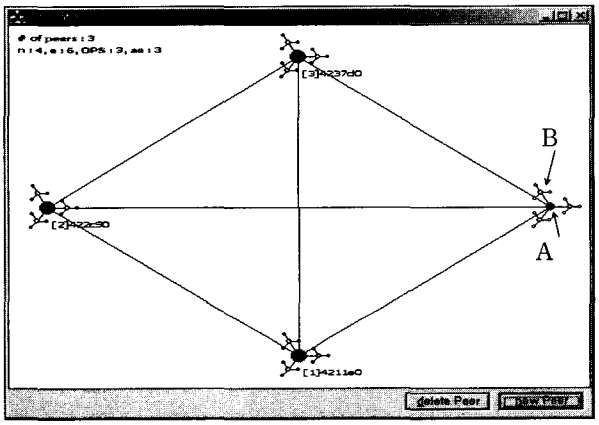
피어 P1은 연결 수가 가장 많은 피어 P5에게 연결 수가 가장 적은 피어 P4와 연결하라는 COP 메시지를 피어 P5에게 전송한 후 피어 P5와의 네트워크 연결을 끊는다. (그림 9)(c)는 피어 P5가 피어 P4에게 연결한 상태를 보인다.

4. 피어 고립 방지 기법 시뮬레이션

지금까지 순수 P2P 네트워크 환경에서, 피어들이 네트워크 그룹으로부터 고립됨을 방지하기 위한 피어 고립 방지 기법을 소개하였다. 본 장에서는 시뮬레이션 모델을 정의하여 이러한 알고리즘들의 성능을 측정하고 그 결과를 비교/분석한다.

4.1 시뮬레이션 개요 및 과정

본 논문에서 제안하는 기법을 테스트하기 위해 Windows 2000 플랫폼을 사용하였으며, 툴은 Microsoft사의 Visual C++ 6.0을 사용하였다. 본 기법을 이용한 시뮬레이션 프로그램의 화면 구성은 (그림 10)에서 보인다.



(그림 10) 피어 4개만이 남아 있을 경우의 피어별 네트워크 연결 모습(N_Connections : 3)

(그림 10)에서, "new Peer" 버튼은 새로운 피어를 실행하는 버튼이며, "delete Peer" 버튼은 이미 실행된 피어를 종료하는 버튼이다. 화면 좌측 상단에 있는 수치 중 n, e, OPS, ae는 각각 피어 수, 노드 edge 수, 권장 연결 수(N_Connections), 피어들의 평균 연결 수를 의미한다. 피어 A는 실행되는 피어들이 접속하는 피어이며, B는 피어 A에 연결된 인접 피어들의 연결 상태를 도식화한 것이다. (그림 10)은 N_Connections를 3으로 설정하였을 때의 실행 화면이며, 4개의 피어만이 남은 네트워크 연결 모습을 보인다. 본 시뮬레이션에서는 100개 이상의 피어들을 실행/종료시킴과 동시에 각 피어들로 하여금 연결 정리 및 연결 분산 기법을 적용함으로써 동작하였다. 이 과정 중, 연결된 인접 피어의 종료로 인해 네트워크 연결이 끊어질 경우 네트워크 연결을 잃은 피어는 새로운 피어와 연결하며, 특정 피어로의 네트워크 연결이 집중되었을 경우 해당 피어는 네트워크 연결을 인접 피어로 분산시키게 된다. 결국 본 시뮬레이션은 각 피어들에게 네트워크 고립 현상이 발생되지 않으며, 어느 특정 피어로 네트워크 연결이 집중되지 않는다는 것을 보여준다.

4.2 시뮬레이션 실행 결과

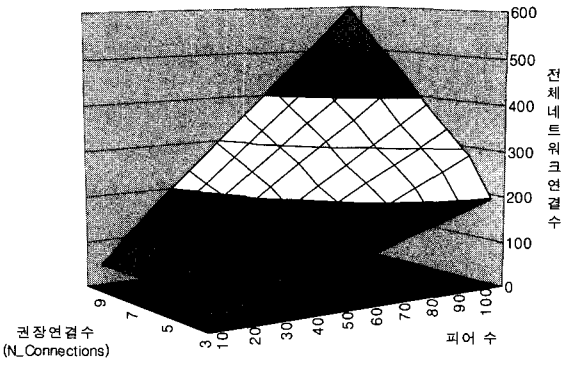
전체 피어 수 n과 피어간 연결되어 있는 전체 연결 수 e에 대한 패킷의 발생 수는 다음에서 보이는 바와 같다.

$$T_n = E_n - 1 \tag{a}$$

$$e = E_1 + E_2 + \cdots E_n = \sum_{k=1}^{n} E_k \tag{b}$$

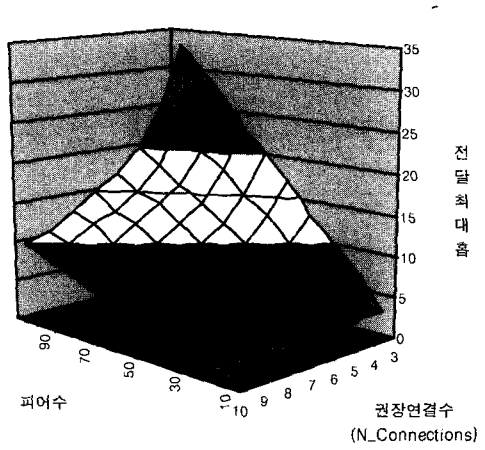
$$T = \sum_{k=1}^{n} T_k + 1 = \sum_{k=1}^{n} E_k - n + 1 = e - n + 1 \tag{c}$$

각 피어들은 메시지를 수신할 경우 메시지가 들어온 피어를 제외한 나머지 피어로 메시지를 전송(broadcast)하기 때문에, 식 (a)에서 보이는 바와 같이 n번째 피어에서 발생하는 패킷 양 T_n은 자신에게 연결된 연결 수보다 1이 작은 수가 된다. 메시지를 발생시키는 피어는 인접한 모든 피어로 메시지를 전송함을 고려할 경우, 전체 네트워크 내에서 생성되는 패킷 양은 식 (c)와 같게 된다. 이 식에서 노드수인 n은 실행된 전체 피어 수이기 때문에 프로그램 내에서 결정할 수 있는 수치가 아니다. 따라서 네트워크 그룹 상에서 발생되는 패킷의 수를 줄이기 위해서는 연결 수를 줄여야 한다는 것을 알 수 있다. 결국 N_Connections의 수치를 작게 할 경우 피어 하나당 연결되어 있는 네트워크 연결 수가 줄어들게 되어 전체 네트워크 그룹에서 발생하는 패킷의 수가 줄어들게 된다. 그러나 N_Connections의 수치가 너무 작으면 패킷이 특정 피어에게 전달되어지는 시간이 오래 걸릴 수 있다. 시뮬레이션 결과로써, 전체 피어의 수와 N_Connections 값에 따른 전체 네트워크 연결 수의 변화를 (그림 11)에서 보인다.



(그림 11) 피어 수, N_Connections의 변화에 따른 전체 네트워크 연결 수

피어 수는 네트워크 그룹에 참여하고 있는 사용자의 수에 의해 결정되기 때문에, 알고리즘에 영향을 미치는 수치는 N_Connections가 된다. (그림 11)에서, N_Connections가 증가하면 전체 연결 수도 증가하는 것을 볼 수 있다. 전체 피어의 수와 N_Connections 값에 따른 전달 최대 홉 수 변화를 (그림 12)에서 보인다.



(그림 12) 피어 수, N_Connections의 변화에 따른 전체 최대 홉 수

(그림 12)에서 N_Connections의 수치가 작을수록 모든 피어에게 전달하기 위한 최대 홉 수가 증가하는 것을 볼 수 있다. 따라서 (그림 11)과 (그림 12)에서 보이는 바와 같이 N_Connections의 수치가 작아지면 전체 연결 수가 줄어들어 전체 네트워크에서 발생되는 트래픽양이 적어지지만, 패킷을 전달하기 위해 거쳐야 하는 피어수가 증가하는 것을 볼 수 있다. 또한, N_Connections의 수치가 커지면 패킷을 전달하기 위해 거쳐야 하는 피어수가 감소하지만, 전체 네트워크 연결 수가 많아져 네트워크 트래픽양이 커지게 된다.

5. 결론 및 향후 연구과제

본 논문에서는 인접한 피어로부터 얻은 ID를 목록으로 관리함으로써, 연결된 피어가 종료할 경우 새로운 피어로 연결하여 네트워크 그룹에 항상 연결되어 있게 하고, 특정 피어에게 네트워크 연결이 집중됨을 방지할 수 있는 기법을 제안하였다. 기존의 순수 P2P 프로그램들은 서버와 클라이언트의 기능을 모두 제공한다는 점에서 순수 P2P 네트워크 환경기반에서 구현되었다고 할 수 있지만, 미리 입력된 ID에게만 네트워크 연결이 집중되기 때문에 입력된 ID를 사용하는 피어들이 종료될 경우 새로운 ID를 입력하지 않는 이상 네트워크 그룹에 참여할 수 없다는 문제를 가지고 있다. 또한, 이러한 피어들은 다른 피어로 연결이 되어 있다 하더라도 특정 피어의 연결 종료로 인해 네트워크 고립 현상이 발생될 수도 있다. 따라서 이 기법은 다양한 피어들의 ID를 목록으로 관리함으로써, 각 피어들로 하여금 연결되어 있는 피어가 종료하여도 새로운 피어에게 연결하여 네트워크 그룹에 지속적으로 참여할 수 있게 한다.

차후 에이전트 플랫폼(agent platform) 개발시 에이전트를 이주(migration)시키기 위해 필요한 플랫폼들의 ID 목록 유지에 적용될 수 있으며, 여러 가지 순수 P2P 응용 프로그램에 적용하여 각 피어들이 네트워크 그룹에 항상 연결될 수 있도록 유지하게 할 수 있다. 또한, 기존의 하이브리드 P2P 네트워크 환경에서 특정 서버를 재가동시키게 되면 각 클라이언트들이 서버로부터 일정동안 서비스를 제공받지 못하는 점, 그리고 사용자들이 많아질 경우 서버의 성능이 저하되거나 제한된 동시 접속자만을 처리할 수밖에 없는 문제점들이 발생하게 되었지만, 본 논문에서 제안된 기법을 이용하는 경우 서버로의 네트워크 연결을 다른 피어로 분산시킴으로써 기존보다 더 많은 클라이언트들을 관리할 수 있게 될 것이다.

참 고 문 헌

[1] David Barkai, "An Introduction to Peer-to-Peer Computing," Developer Update Magazine, Intel Corporation, Feb., 2000.

[2] White Paper, "Introducing Peer-to-Peer," Endeavors Technology Inc., 2002.

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

[4] JXTA, "Project JXTA Virtual Network," Sun Microsystems, Inc., Feb., 2002.

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

[6] 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.

[7] 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.

[8] D. C. Hyde, "How New Peer to Peer Developments May Effect Collaborative Systems," Department of Computer Science, Bucknell University, Jan., 2002.

[9] CLIP2, "The Gnutella Protocol Specification v0.4," http://www.clip2.com.

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

[11] DS. Milojicic, et al., "Peer-to-Peer Computing," HP Laboratories Palo Alto, Mar., 2002.

김 영 진

e-mail : yj21c.kim@samsung.com
2002년 대전대학교 컴퓨터공학과(학사)
2004년 성균관대학교 대학원 정보통신 공학부(석사)
현재 삼성전자 소프트웨어센터 연구원
관심분야 : P2P, 이동 에이전트, 분산 컴퓨팅 등

엄 영 익

e-mail : yieom@ece.skku.ac.kr
1983년 서울대학교 계산통계학과(학사)
1985년 서울대학교 대학원 전산과학과 (석사)
1991년 서울대학교 대학원 전산과학과 (박사)
2000년~2001년 Dept. of Info. and Comm. Science at UCI 방문 교수
현재 성균관대학교 정보통신공학부 교수
관심분야 : 분산 시스템, 이동 컴퓨팅, 이동 에이전트, 시스템 소프트웨어, 시스템 보안 등