Gnutella 기반의 균형 잡힌 트리형태 토폴로지

김철민° 박재현
중앙대학교 컴퓨터공학과
cheal72@hsc.cau.ac.kr°, hyunie@cau.ac.kr

Balanced Tree Topology in Gnutella

Cheal-Min Kim°, Jae Hyun Park
Dept. of Computer Science and Engineering, Chung-Ang University

요 약

이 논문에서 제안하는 Gnutella기반의 Balanced tree topology는 Unstructured P2P이므로 탐색 질의가 hash key에 제한되지 않고 동적인 네트워크 환경에도 적합하다. 또한 topology형태가 tree이므로 Gnutella의 큰 문제점인 질의 탐색 시 중복된 패킷을 제거하고, tree형태가 balanced tree이므로 일정 홉 수 안에 모든 노드를 방문 할 수 있다. Balanced tree topology는 일정 홉 수 안에 모든 노드를 방문하기 때문에 Gnutella기반의 P2P가 가지는 희귀 자료에 대한 검색의 문제점 또한 해결하였다. Gnutella기반의 확장 ping과 확장 local cache를 사용하여 balanced tree topology유지 및 문제 해결에 필요한 추가 비용을 줄였다.

1. 서 론

P2P 네트워크는 자원의 분배 방식, 노드 조인 방식 및 네트워크 형태에 따라 크게 unstructured, structured 네트워크로 구분된다.[1][2] structured P2P시스템은 DHT기반 P2P라고도 하며, CAN[3], Chord[4], Tapestry[5], Pastry[6], Kademlia[7], Viceroy[8], Koorde[9], Structured SuperPeers[10]등이 DHT구현 방식을 사용한다.

DHT방식은 hash key index를 균등하게 분배하여 logN비용으로 원하는 데이터를 얻으므로 중복된 패킷이 없다. 자료에 대한 정보를 가지고 있으므로 희귀한 데이터도 같은 검색 비용으로 찾을 수 있다. 하지만 라우팅 테이블을 유지해야 하므로 주기적인 정보 교환하는 오버헤드가 있고, 노드의 참여와 이탈이 빈번한 동적인 네트워크에는 알맞지 않다. 또한 검색이 hash key 값에 기반 하므로 쿼리 검색이 index 값에 제한되고, 복잡한 쿼리나, 범위로 찾는 쿼리 또한 제한되므로 이에 대한 연구는 아직도 진행되고 있다.

unstructured P2P에는 Napster[11], Freenet[12], Gnutella[13], FastTrack/KaZaA[14], BitTorrent[15], eDonkey2000[16]등이 있다. unstructured P2P는 노드의 조인과 이탈이 간단하고, 특정 topology를 유지하기위해 주기적으로 정보를 교환할 필요가 없다. 또한 DHT기반이 아니므로 데이터 검색이 index에 제한되지 않는다. 하지만 데이터에 대한 정보가 없으므로 최소한의 hop안에 최대한 많은 노드를 방문하여 데이터를 검색하고자 한다. 따라서 unstructured P2P방식은 flooding방식을 기본 방식으로 사용한다. flooding방식은 일정 홉 안에 많은 노드를 방문하는 목적에는 적합하지만 중복된 패킷생성이라는 큰 문제를 가지고 있다. 또한 홉 수가 모든 네트워크에 다다를 만큼 크지 않으면 희귀한 데이터 검은 실패할 것이다.

DHT기반의 P2P는 데이터 검색이 index에 제한되고, flooding 방식을 사용하는 unstructured P2P는 중복된 패킷 생성이라는 큰 문제점을 가지고 있다. 이 논문에서는 이러한 문제점을 해결하기위해 gnutella 기반의 balanced tree topology를 제안한다.

2. 관련 연구

DHT기반의 P2P에는 index key를 기반으로 key 분배에 사용되어지는 알고리즘에 따라 여러 형태의 topology가 있다. Tree 형태에는 PRR[17]를 기본적으로 따르는 Tapestry[5], Pastry[6], Kadlemia[7]등이 있고, B+tree와 CHORD을 접목한 P-Tree[18], 각 노드가 가상의 이진 prefix 탐색 tree를 유지하는 P-Grid[19], B tree를 p2p 환경에 맞게 변형시킨 PB-link tree[20]가 있다. BATON[21]는 height-balanced tree를 형성하여 logN의 탐색 시간을 보장하는 알고리즘을 제안하였고, SCALLOP[22]는 balanced tree를 형성하여 load balance를 유지한다. 그 외에도 다양한 tree형태의 혹은 balanced tree형태의 방법을 제시되고 있지만, DHT를 기반으로 하고 있기 때문에 DHT방식의 가장 큰 문제인 쿼리 탐색이 index key에 제한되는 문제점을 여전히 가지고 있다.

그 외에도 DHT기반의 P2P에는 Ring형태의 Chord[4], Tori형태의 CAN[3], Butterfly형태의 Viceroy[8], de Bruijn Graphs형태의 Koorde[9], Skip Lists를 사용하는 Skip Graphs[23]등이 있다.

structured P2P와 달리 unstructured P2P는 topology구성이 비교적 자유롭다. unstructured P2P의 가장 큰 문제인 중복 패킷을 제거하기 위해 여러 가지 방법들이 제시되었다.

Iterative deepening[24] 는 TTL을 늘리면서 원하는 자료를 검색하여 중복된 패킷을 제거한다. 이 방법은 수효가 많은 파일 검색에는 유리 하지만 비인기 파일 검색에는 적합하지 않다. Directed BFS[24]는 쿼리 발생 노드가 좀 더 높은 결과 값을 보장 할만한 몇 개의 이웃 노드집합에게만 쿼리를 전송하고 그 이후 hop부터는 기존의 BFS방법을 따른다. 이 방법은 한 hop 안에서 쿼리를 전송할 이웃 노드 집합선택이 어렵고, 2 hop부터는 기존의 flooding을 사용하므로 큰 이득이 없다. 그 외에도 k-walker random walk[25], Intelligent search[26], Local indices based search[24], Routing indices based search[27], Attenuated bloom filter based search[28], Adaptive probabilistic search[29]등이 있다. 이러한 방법들은 중복 패킷을 줄이기 위해서 경험적 방법으로 혹은 확률적인 방법으로 또는 랜덤으로 이웃 노드를 선택하여 질의를 보낸다. 이러한 방법들은 예측 값이 틀리거나 이웃 노드를 잘못 선택할 경우 원하는 데이터를 찾는데 많은 시간이 소요되고 새로운 종류의 데이터를 찾을 때에도 많은 시간이 걸린다. Super-peers 방식은 super peers와 super peers를 부모 노드로 가지는 노드들 간의 주기적인 meta-data교환하는 오버헤드가 있고, super-peers가 죽었을 경우 자식 노드들은 통신이 두절되는 문제점이 있다.

LightFlood[30]과 FloodTrail[31]는 tree를 사용하여 중복된 패킷을 제거했다. 하지만 LightFlood는 비 연결 sub-tree가 생성되어 모든 노드들의 완전한 연결성을 보장하지 못하고, balanced tree가 아니므로 hop를 예측할 수 없다. FloodTrail는 모든 노드마다 각각의 tree를 생성 및 유지해야 하므로 엄청난 오버헤드를 감수해야하고, 또한 balanced tree가 아니므로 쿼리 검색 시 hop수도 보장하지 않는다.

이 논문에서 제안하는 gnutella 기반의 balanced tree topology는 unstructured P2P기반이므로 DHT기반의 P2P의 문제점인 index 제한적인 검색 문제점이 없다. 또한 topology를 tree형태로 형성하여, unstructured P2P의 가장 큰 문제점인 중복 패킷 문제를 해결하고 tree를 balanced tree로 유지하여 질의응답 시간을 보장한다. 기존의 unstructured 방식은 몇몇의 이웃 노드만 선택하기 때문에 query hit이 낮은 반면 이 논문에서 제안하는 방법은 중복 패킷 없이 모든 노드를 방문하기 때문에 같은 hop수 안에 query hit이 높다. balanced tree 형태의 topology 이므로 N개의 노드를 검색 할 때 총 N-1개의 패킷이 사용되어 지며, $2 \times \log_a N$ 개 hop으로 모든 노드를 방문 할 수 있다.

3. 알고리즘

이 논문에서 제안하는 balanced tree topology는 Gnutella기반으로 최대한 중복된 패킷을 줄이고 기존의 unstructured P2P와 같은 검색 효과를 가지는 것을 목표로 한다. 또한 Gnutella 0.6 Protocol[33]과 중복되는 내용은 설명하지 않는다.

노드 5개가 <그림 1>과 같이 모든 노드들이 서로 연결되거나

< 그림 1 topology structure >

tree 형태로 연결되었고, 연결된 노드들 간의 거리와 응답 속도가 같고, Gnutella기반의 질의 탐색 방식을 따른다고 가정하자. 임의의 노드에서 질의를 시작하면 왼쪽 네트워크에서 사용된 총 패킷 수는 4+3+3+3+3개 즉 16개이고, 오른쪽 네트워크는 4개이다. 따라서 Gnutella기반의 토폴로지에서 한 개의 탐색 질의가 모든 노드들 방문하는데 사용되는 총 패킷 수는 최소 (N-1)개서 최대$(N-1)^2$ 개임을 알 수 있다. 이 논문에서 제안하는 balanced tree topology는 tree형태 이므로 하나의 질의 문에 사용되는 총 패킷 수는 (N-1)개이고, 실제 실험에서는 68%의 중복 패킷을 제거하는 효율을 보였다.

N개의 노드가 참여하는 balanced tree 형태의 높이는$\log_a N$이다. 이것은 임의의 노드에서 질의 탐색을 시작했을 때$2 \times \log_a N$번의 홉 수 안에 모든 노드를 방문 하게 해 준다. a의 크기를 적절히 조절하면 Gnutella에서 90% 노드를 방문하는 평균 홉 수 인 7홉에 근접 할 수 있다. 실제 실험에서는 a를 30으로 두어 평균 6~7홉으로 모든 노드를 방문 가능하게 하였다.

모든 노드는 balanced tree 형태를 유지하기 위해서 최신의 <표 1> 정보를 각각 가지고 있으며 확장 ping을 사용하여 필요한 정보를 교환한다. 확장 ping은 기존 ping헤더에 myLevelChecking, parentID, myLevel를 첨부한 것 이고, <표 2>는 이를 설명하고 있다. ping에 실려 온 유효한 myLevel중 가장 작은 값이 leastLevel이 된다.

<표 1>

myLevel	부모 노드 myLevel에 1를 더한 값.
leastLevel	자식을 더 가질 수 있는 노드들의 myLevel중 가장 작은 값.
childNum	부모 노드가 현재 가지고 있는 자식들 수.

<표 2>

myLevelChecking	자식을 더 가질 수 있으면 1로 세팅되어 myLevel값이 유효함을 알림.
parentID	부모 노드의 ID로 보통 32bits의 IP를 사용함.
myLevel	Topology tree에서의 자신의 level.

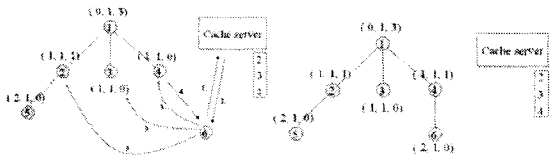
3.1 node join

노드가 처음 Gnutella네트워크에 참여하고자 할 땐 기존의 방법처럼 Gnutella cache server통해서 참여 가능한 노드 주소를 받아 연결을 시도한다.[32] <표 3>는 연결 가능한 노드들의 조건이며 이러한 조건을 만족하는 노드들의 주소가 cache server에 저장되어 있다. 부모 노드는 <표 3>의 조건을 만족할 때에만 연결을 허락 할 수 있다.

<표 3>

childNum < maxChildNum
&&
myLevel <= leastLevel

<그림 2> 는 노드6이 bootstrapping 하는 과정을 보여주고 있다. 노드는 자식을 최대한 3개 가질 수 있다고 가정한다. Gnutella 에서는 주기적으로 노드와 Cache server간의 정보 교환이 있고, Cache server는 <표 3>을 만족하는 노드들을 cache에 저장해 놓는다. 따라서 Cache server는 노드2,3,4의 주소를 저장해 놓고 있다. 노드1은 자식의 level이 0이고, topology에의 leastLevel이 1이고, 자식을 3개 가지고 있으므로 (0,1,3)의 값을 가지고 있다. 노드 2,3,4,5 도 각각에 해당하는 값을 가지고 있다. 노드6은 Cache server을 통해서 연결 가능한 노드들 즉 cache 에 저장된 노드2,3,4의 주소를 받고 자신의 parentLevelCache에 저장한다. 그리고 연결을 시도한다. <표 3>를 계속 만족하는 노드들은 대답을 할 것이고, 그 중 응답이 빠른 노드를 자신의 부모로 선정하고 연결한다. <그림 2>에는 노드4가 가장 먼저 응답을 하였고 노드 6은 노드 4를 자신의 부모로 선정한다. <그림 3>은 노드 6이 연결된 이후의 모습이다.



<그림 2 node join>　　<그림 3 node join>

```
GET_BYE_OR_LEVELUP {
  IF(bye message is sent by parent node or level up message)
    IF(has a neighbor node)
      do LEVEL_UP_PROCESS;
    ELSE {
      IF(message is a level up message)
        maintain connection with level up node;
      ELSE
        connect to grand parent node;
    }
  IF(bye message is sent by child node)
    IF(grandChild flag value in bye message is 1)
      preserve one child place for a specific time;
    ELSE
      decrease childNum;
}
LEVEL_UP_PROCESS {
  IF(this node is not the oldest node)
    for a specific period wait for neighbor's level up notification;
  notify neighbor nodes that this node levels up;
  IF(has children nodes)
    LEVEL_UP_TO_CHILD_NODE;
}
LEVEL_UP_TO_CHILD_NODE {
  send children nodes level up message;
  preserve one child place for a specific time;
}
```

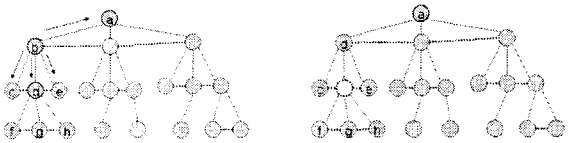
<그림 4 node leave and level up 알고리즘>

3.2 node leave and level up

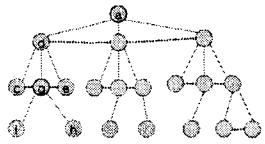
node leave란 topology에 참여하고 있는 노드가 정상적으로 주위 노드에게 자신이 나감을 알리고 떠나는 것을 말한다. 떠나는 node는 주위 1 hop 노드에게 bye ping메시지를 보낸다. bye메시지는 Gnutella spec 0.6[33] 에 사용되는 bye메시지로 ttl값이 1인 메시지다. 부모에게 bye ping을 받은 자식 노드는 level up할 준비를 한다. 여기서 level up이란 myLevel이 1감소하여 부모 위치의 level로 상승하는 것을 의미한다. bye ping을 받은 부모 노드는 level up할 경우에 대비하여 자식들 자리중 하나를 비워둔다. neighbor list중에서 가장 오래된 노드 순으로 자신이 level up 하겠음을 알린다. 먼저 알린 노드가 level up하게 된다. 이웃 노드는 neighbor list cache에 저장되며 ping에 실려 온 parentID가 자신의 parentID와 같으면 ping을 보낸 노드가 이웃 노드이므로 neighbor list cache에 저장한다. <그림 5>에서 점선으로 연결된 노드들은 서로 이웃 노드이므로 c, d, e와 f, g, h는 이웃 노드이다.

임의의 노드가 떠나거나 갑자기 사라졌을 경우 자식들 중 가장 오래된 노드가 새로운 부모가 된다. 새롭게 부모가 된 노드를 level up했다고 한다. 이 과정은 leaf node를 만날 때까지 실행된다. 즉 노드가 떠날 때 자식 노드를 가지고 있었다면 node level up을 항상 수반하게 된다.

<그림 5>는 노드 b가 node leave하고 노드 d, g가 node level up하는 과정을 보여주고 있다. 떠나는 노드 b는 bye message에 부모 노드 a의 주소와 자식이 있음을 알리는 flag를 첨부한 이후 1 hop안의 노드들에게 bye message를 전송한다. 노드 a는 bye message의 flag를 확인하여 b의 자식 노드가 있음을 알고 일정 시간 동안 자식 자리 하나를 비워두어 level up 할 노드를 다른 새로운 노드보다 연결 우선순위를 높게 둔다. bye message를 받은 노드 c, d, e는 level up할 준비를 한다. 이때 neighbor list cache중에서 가장 오래된 노드 d가 이웃 노도들과 자식 노드들에게 자신이 level up함을 알린다. 노드 d는 bye message에 실려 온 노드a 에게 level up연결을 요청하고, 이웃 노드 c, e와 새로운 부모 자식 관계를 맺는다. level up 메시지를 받은 자식 노드 f, g, h는 새로운 level up 과정을 시작한다. 가장 오래된 노드 g가 level up하게 되고, 기존의 부모 노드 d에게 자신이 level up했음을 알린 이후 부모 자식 관계를 유지하고, 이웃 노드 f, h와 새로운 부모 자식 관계를 맺는다. 이러한 level up 과정은 leaf node 까지 반복된다. <그림 4>는 bye message혹은 level up message를 받을 경우 level up과정, 새로운 부모 자식 노드 연결 과정 및 level up할 자식노드를 위해서 일정 시간 노드 자리를 비워두는 과정, childNum값 갱신 과정 등을 보여주고 있다.



<a>　　<b>



<c>

<그림 5 node leave, level up>

3.3 node break

노드가 갑자기 사라지거나 죽었을 경우를 node break라 한다. 예기치 못한 상황에서 연결이 끊어지면 기본적으로 local Cache를 사용하여 새로운 연결을 시도한다. 부모가 갑자기 사라질 경우 parentLevelCache를 사용하여 기존의 부모 노드의 level과 같은 노드와 연결을 시도한다. 이는 전제 topology를 balanced tree형태가 되게 한다. 이러한 node break는 level up과정에서도 나타날 수 있다. 일반 적으로 level up한 노드는 level up하지 못한 이전의 이웃 노드에게 연결을 시도한다. 하지만 일정 시간동안 level up한 노드의 연결 요청이 없으면 새로운 level up 주기가 시작되어 다른 노드가 level up하게 된다. 모든 시도가 실패하면 각각의 노드는 parentLevelCache를 통하여 새로운 부모와 연결을 시도한다. 최악의 경우 Gnutella Cache Server에 접속하여 새롭게 bootstrapping한다. <그림 6>은 node break되었을 때 혹은 childNum값이 maxChildNum을 초과하여 부모 노드로부터 버림을 받았을 때 parentLevelCache를 사용하여 재 연결하는 과정과, 연결 이후의 초기화 과정을 보여주고 있다.

```
JOIN {
  WHILE(not connected) {
    IF(parent node break || discard message from parent node)
      send connection request message to nodes which are in parentLevelCache:
    IF(fail in connection request with nodes which are in parentLevelCache)
      get nodes whose level is same with parent node's level from Cache Server:
    IF(first join)
      bootstrapping by using Cache Server:
  }
  myLevel = myLevel of connected node+1:
  get localCache information from connected node:
}
```

<그림 6 JOIN 알고리즘>

4 overhead

전체 네트워크를 balanced tree형태로 유지하기 위해서 확장 ping, pong, local cache를 사용한다. <표 4>는 기존의 ping 크기와 확장 ping에 추가된 myLevelChecking, parentID, myLevel의 크기를 각각 보여주고 있다. 추가된 내용의 총 크기는 36bits로 이것은 전체 ping의 크기가 24Bytes인 것과 비교하면 큰 부담이 되지 않는다. 여기서 myLevel는 ping을 보낸 노드의 level이며 평균 Gnutella에서 90% 노드를 방문하는 걸리는 홉 수가 7홉인 것을 감안하면 leaf node에서 root node까지 3홉 안에 도달해야 하고 이때의 level값은 2bits로 충분히 나타낼 수 있다. 따라서 level값은 3bits면 모든 level을 충분히 나타낼 수 있다. pong은 ping과 달리 header만으로 구성되어 있지 않으므로 확장 pong은 큰 문제가 되지 않는다.

<표 4>

Ping header	23 Bytes
myLevelChecking	1 bit
parentID	32 bits
myLevel	3 bits

<표 5>는 이 논문에서 사용하고 있는 local cache를 보여 주고 있으며 기존 Gnutella의 local cache를 용도에 따라 확장한 것이다. Gnutella는 노드들 간의 중복된 패킷 교환이 큰 이슈이지 이러한 local cache확장 및 변경은 큰 문제가 되지 않는다. 또한 개인 컴퓨터 하드웨어의 발달로 이러한 local cache확장은 큰 문제가 되지 않는다. 또한 local Cache 크기를 적절히 조절하여 원하는 성능을 얻으면서 메모리 사용도 절약할 수 있다.

<표 5>

ParentLevelCache	부모 노드와 level이 같은 노드들이 저장됨. Node break시 사용.
NeighborListCache	같은 부모와 연결된 이웃 노드들이 저장됨. level up시 사용.
LeastLevelCache	myLevelChecking값이 1인 myLevel값이 저장됨. leastLevel 값 설정 때 사용.

5 실험

본 논문은 Gnutella기반 P2P의 큰 문제인 중복 패킷을 제거하는 방법으로 balanced tree 형태의 topology를 제안하였고 기존의 Gnutella 방식의 질의 검색 성공률을 유지하면서 질의 검색에 사용된 총 패킷을 줄이는 것을 목표로 하였다.

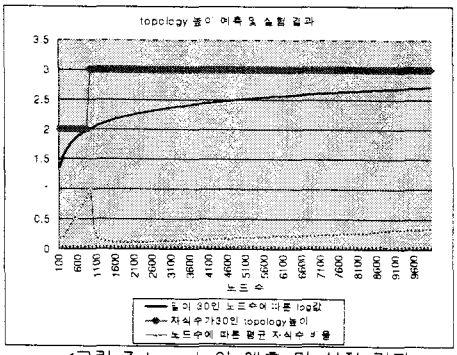
Topology에 참여하는 노드수가 10000개가 될 때 까지 랜덤하게 노드들이 참여하고 일정한 비율로 노들이 topology를 떠나며 모든 실험 그래프는 노드 수가 10000개가 된 이후에 측정한 것이다. Gnutella기본적인 topology형태인 random, power law topology와 비교 하였다. 시뮬레이션을 위한 파라미터는 <표 6>과 같다.

<그림 7>는 N값의 증가에 따른 $\log_{30} N$ 예측 및 실제 실험에서 maxChildNum값이 30일 때의 topology의 높이의 결과이다. topology의 높이가 3을 유지하는 것은 10000개의 노드가 balanced tree를 유지하고 있음을 보여준다. <그림 8>은 노드 수에 따른 전체 노드를 방문하는데 사용된 총 패킷을 보여 주

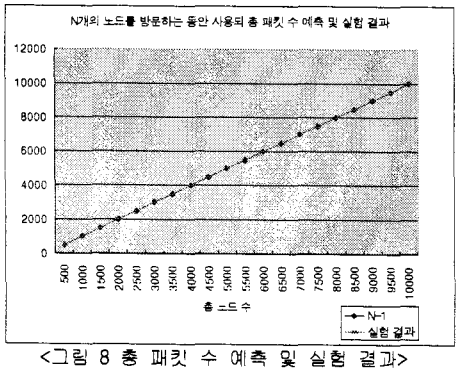
<표 6>

Simulator	PeerSim P2P Simultor(BISON)
topology	Balanced Tree, Power-Law, Random
# of child in Balanced Tree	30
# of maximum node	10,000
Node leave rate	20%
Data distribution	Zipf distribution
Query distribution	Zipf distribution
Max query	10
Keywords width	10
LeastLevelCache size	160
ParentLevelCache size	60
NeighborListCache size	40
Sending protocol	flooding
Maximum TTL	10

고 있다. Balanced tree topology는 앞에서 예측한데로 노드수가 N개이면 총 사용된 패킷 수는 N-1이라고 예측하였다. <그림 8>의 실험 결과는 예측한 데로 사용된 총 패킷이 N-1임을 보여주고 있고 이것은 실험 결과 노드들이 tree형태를 띄고 있음을 나타낸다.

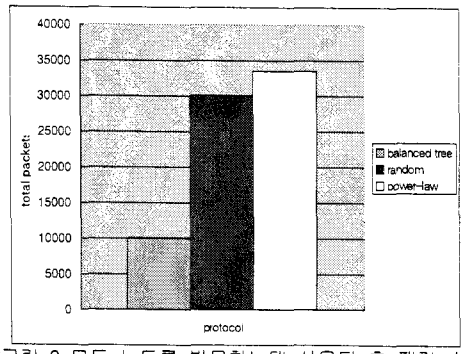


<그림 7 tree 높이 예측 및 실험 결과>



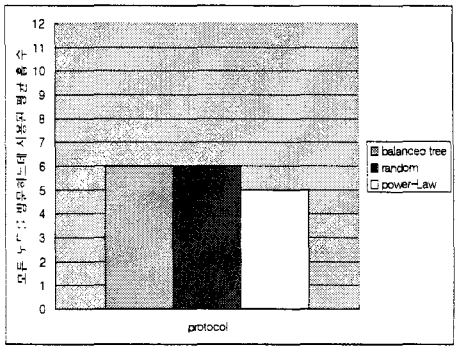
<그림 8 총 패킷 수 예측 및 실험 결과>

<그림 9>은 balanced tree topology, random topology, power law topology의 노드가 각각 10000개인 실험 환경에서 한 개의 탐색 질의가 모든 노드를 방문 할 때 사용된 총 패킷 수를 나타내고 있다. balanced tree topology에서는 다른 topology에 비해 약 68%의 중복 패킷이 제거됨을 알 수 있다.



<그림 9 모든 노드를 방문하는데 사용된 총 패킷 수>

<그림 10>은 모든 노드를 방문하는데 사용된 평균 홉 수를 보여 주고 있다. Balanced tree topology에서 사용되는 홉 수는 random, power law topology와 비슷함을 알 수 있다. 또한 Gnutella에서 90%의 노드를 방문하는데 평균 7홉인 것과 비교하여도 Balanced tree topology의 6홉은 만족할 만한 결과이다. 또한 탐색 질의가 모든 노드를 방문 하므로 hit률은 세 개의 topology모두 같음 을 알 수 있다.



<그림 10 모든 노드를 방문하는데 사용된 총 홉 수>

6 결론 및 향후 과제

structured P2P는 탐색 질의가 hash index에 제한되고, unstructured P2P는 질의 탐색 시 중복된 패킷이 많다. 이러한 두 문제를 해결하기 위해 Gnutella기반의 balanced tree topology를 제안하였다. Gnutella기반이므로 탐색 질의가 hash key에 제한되지 않고, tree 형태이므로 중복된 패킷 없이 모든 노드를 방문 할 수 있다. 또한 tree형태가 balanced tree이므로 일정 홉 수 안에 모든 노드를 방문하여 원하는 질의 탐색 시 원하는 결과 hit을 준다. 실제 실험에서 N개의 노드들을 방문하는데 총 N-1개의 패킷이 사용되었다. 이것은 topology 형태가 random 이거나 powerLaw인 것과 비교하여 중복된 패킷이 약 68%제거된 것이다. 부모 노드가 가질 수 있는 최대 자식 수가 a이면 $2\times\log_a N$안에 모든 노드들을 방문 할 수 있고 개인 하드웨어의 용량을 고려하여 a를 적절히 조절하면 원하는 tree의 높이를 얻을 수 있다. 실제 실험에서 maxChildNum이 30이면 $2\times\log_{30} N$값이 6이 되어 Gnutella에서 90%의 노드들을 방문하는데 평균 7홉이 걸리는 것과 비교하여 질의 응답속도 또한 적절함을 보였다. 또한 다른 topology와의 질의 hit비율도 같음을 보였다.

향후 질의 탐색 시 일어나는 node break에서의 패킷 예측과 질의 전송에 대한 방법 제안 및 local cache크기 변화에 따른 전체 네트워크 형태 변화 예측과 local cache크기 최상화가 필요하다. 또한 balanced tree topology라는 새로운 형태의 unstructured P2P에 기존의 다른 P2P기술을 접목하여 보다 효율적인 질의 탐색을 할 수 있다.

7 참고문헌

[1] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peerto- peer networks" , Proc. of the

16th ACM International Conference on Supercomputing(ACM ICS' 02), 2002.

[2] Xiuqi Li, Jie Wu, "Searching Techniques in Peer-to-Peer Networks", Handbook of Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks, CRC press, 2005

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content addressable network," in Processings of the ACM SIGCOMM, 2001, pp. 161-172.

[4] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," IEEE/ACM Transactions on Networking, vol. 11, no. 1, pp. 17-32, 2003.

[5] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," IEEE Journal on Selected Areas in Communications, vol. 22, no. 1, pp. 41-53, January 2004.

[6] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in Proceedings of the Middleware, 2001.

[7] P. Maymounkov and D. Mazi`eres, "Kademlia: A peer-to-peer information system based on the xor metric," in Processings of the IPTPS, Cambridge, MA, USA, February 2002, pp. 53-65.

[8] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: a scalable and dynamic emulation of the butterfly," in Processings of the ACM PODC' 02, Monterey, CA, USA, July 2002, pp. 183-192.

[9] Kaashoek, M. F. and Karger, D. 2003. "Koorde: A simple degree-optimal distributed hash table." , In Proceedings of the 2nd International Peer To Peer Systems Workshop (IPTPS 2003). SanDiego, California, USA.

[10] My' zrak, A. T., Cheng, Y., Kumar, V., and Savage, S. 2003. "Structured superpeers: Leveraging heterogeneity to provide constant-time lookup." , In WIAPP ' 03: Proceedings of the The Third IEEE Workshop on Internet Applications. IEEE Computer Society, Washington, DC, USA, 104.

[11] Napster 2001. Napster. http://www.napster.com/.

[12] Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. 2001. "Freenet: a distributed anonymous information storage and retrieval system." In International workshop on Designing privacy enhancing technologies. Springer-Verlag New York, Inc., 46-66.

[13] Gnutella 2005. Gnutella. http://rfc-gnutella.sourceforge.net/.

[14] KazaaMediaDesktop 2005. KaZaA media desktop. http://www.kazaa.com/.

[15] BitTorrent 2005. BitTorrent. http://www.bittorrent.com/.

[16] edonkey2000 2005. eDonkey2000 - Overnet. http://www.edonkey2000.com/.

[17] Plaxton, C. G., Rajaraman, R., and Richa, A. W. 1997. Accessing nearby copies of replicated objects in a distributed environment. In SPAA ' 97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures. ACM Press, New York, NY, USA, 311- 320.

[18] Adina Crainiceanu, Prakash Linga, Hohannes Gehrke, Jayavel Shanmugasundaram, "P-Tree: A P2P Index for Resource Discovery Applications", ACM 1-58113-912-8/04/0005.

[19] http://www.p-grid.org/

[20] Ming Li, Dongsheng Wang, Weimin Zhen, Jinfeng Hu, and Yongquan Ma "PB-link tree: a numeric range query algorithm in peer-to-peer architecture", China National Computer Conference (CNCC 2003)

[21] H.V. Jagadish , Beng Chin Ooi , Quang Hieu Vu, "BATON: A Balanced Tree Structure for Peer-to-Peer Networks",the 31st VLDB Conference, Trondheim, Norway, 2005

[22] Jerry C.-Y. Chou, Student Member, IEEE, Tai-Yi Huang, Member, IEEE, Kuang-Li Huang, and Tsung-Yen Chen, "SCALLOP: A Scalable and Load-Balanced Peer-to-Peer Lookup Protocol", IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 17, NO. 5, MAY 2006

[23] Harvey, N., Jones, M., Saroiu, S., Theimer, M., and Wolman, A. 2003. "Skipnet: A scalable overlay network with practical locality properties.", In In Proc. 4th USENIX Symp. on Internet Tech. and Syst. (USITS), 2003. ACM Press.

[24] B. Yang, H. Garcia-Molina, "Improving Search in Peer to Peer Systems" , Proceedings of the 22nd International Conference on Distributed Computing Systems, July 2002, pp. 5-14.

[25] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer to peer networks" , Proc. of the 16th ACM International Conference on Supercomputing (ACM ICS' 02), 2002.

[26] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-yazti, "A local search mecha nism for peer to peer networks" , Proc. of the 11th ACM Conference on Information and Knowledge Management (ACM CIKM' 02), 2002.

[27] A. Crespo, and H. Garcia-Molina, "Routing indices for peer-to-peer systems" , Proc. of the 22nd International Conference on Distributed Computing (IEEE ICDCS' 02), 2002.

[28] S. C. Rhea, and J. Kubiatowicz, "Probabilistic location and routing" , Proc. of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM' 02), 2002.

[29] D. Tsoumakos, and N. Roussopoulos, "Adaptive probabilistic search in peer-to-peer networks" , Proc. of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS' 03), 2003.

[30] Song Jiang, Lei Guo and Xiaodong Zhang, "LightFlood: an Efficient Flooding Scheme for File Search in Unstructured Peer -to-Peer Systems", ICPP'03 0190-3918/03

[31] Song Jiang, Xiaodong Zhang, "FloodTrail: an Efficient File Search Technique in Unstructured Peer-to-Peer Systems", IEEE Globecom Conference, San Fransisco, California, USA, December 1-5 2003.

[32] Pradnya Karbhari, Mostafa Ammar, Amogh Dhamdhere, Himanshu Raj, George Riley, Ellen Zegura, "Bootstrapping in Gnutella: A Preliminary Measurement Study", GIT-CC-03-35, GA-30332

[33]http://rfc-gnutella.sourceforge.net/developer/testing/index.html

[34] Daniel Stutzbach, Reza Rejaie, Subhabrata Sen, "Characterizing Unstructured Overlay Topologies in Modern P2P File-Sharing Systems." , Technical Report CIS-TR-2005-01