멀티 홉 무선 애드혹 네트워크에서 P2P 응용을 위한 이웃 캐싱

(Neighbor Caching for P2P Applications in Multi-hop Wireless Ad Hoc Networks)

조 준 호 † 오 승 택 † 김 재 명 †† 이 형 호 ††† 이 준 원 ††††
(Joonho Cho) (Seungtaek Oh) (Jaemyoung Kim) (Hyeongho Lee) (Joonwon Lee)

요 약 애드혹 네트워크 상의 노드들이 서로의 분산된 데이타를 주고 받는 P2P 응용은 멀티 홉 무선 통신의 오버헤드로 인하여 효율성이 떨어진다. 이것을 극복하기 위해서 본 논문은 이웃 캐싱(neighbor caching) 기법을 제안하고, 이 방법이 노드들의 독립적인 캐싱 방법보다 효율적이라는 것을 보이고 있다. 이웃 캐싱 기법은 쉬고 있는 이웃 노드의 저장 공간을 잠시 빌려 씀으로써 캐싱 공간을 확대하고 먼 거리에서 데이타를 가져오는 멀티 홉 무선 통신의 단점을 극복하는 방법이다. 모의 실험의 결과에 따르면 이웃 캐싱은 망의 크기가 커질 때, 노드들의 쉬는 시간이 길 때, 그리고 노드들의 캐시 크기가 작을 때 좋은 성능을 나타낸다. 이와 함께 본 논문에서는 이웃 캐싱을 할 때 노드들 중에서 최적의 이웃 노드를 선별해 내는 우선순위에 근거한 예측기법(ranking based prediction)을 제안하였다. 우선순위에 근거한 예측 기법을 통해 데이타가 가장 오랫동안 보관될 가능성이 높은 이웃 노드를 선별해내고 우선순위가 낮은 데이타를 이웃 캐싱 하지 않을 수 있어서 이웃 캐싱의 효율성을 높일 수 있다. 모의 실험을 통해 이 방법이 노드들의 상황에 따라 이웃 캐싱의 횟수를 적절히 조절하여 성능향상을 가져올 뿐만 아니라 노드들이 분주한 상황에서도 이웃 캐싱이 유연하게 동작하도록 하는 것을 알 수 있다.

키워드 : 이웃 캐싱, 멀티 홉, 애드혹 네트워크, P2P, 데이타 캐싱, 클라이언트 캐시, 분산 컴퓨팅

Abstract Because of multi-hop wireless communication, P2P applications in ad hoc networks suffer poor performance. We propose neighbor caching strategy to overcome this shortcoming and show it is more efficient than self caching that nodes store data in theirs own cache individually. A node can extend its caching storage instantaneously with neighbor caching by borrowing the storage from idle neighbors, so overcome multi-hop wireless communications with data source long distance away from itself. We also present the ranking based prediction that selects the most appropriate neighbor which data can be stored in. The node that uses the ranking based prediction can select the neighbor that has high possibility to keep data for a long time and avoid caching the low ranked data. Therefore the ranking based prediction improves the throughput of neighbor caching. In the simulation results, we observe that neighbor caching has better performance, as large as network size, as long as idle time, and as small as cache size. We also show the ranking based prediction is an adaptive algorithm that adjusts times of data movement into the neighbor, so makes neighbor caching flexible according to the idleness of nodes

Key words : neighbor caching, multi-hop, ad hoc network, P2P, data caching, client cache, distributed computing

† 학생회원 : 한국과학기술원 전자전산학과
jhcho@camars.kaist.ac.kr
stoh@camars.kaist.ac.kr
†† 비 회 원 : 한국전자통신연구원 네트워크기술연구소 연구원
jaemkim@etri.re.kr
††† 종신회원 : 한국전자통신연구원 네트워크기술연구소 부장
holee@etri.re.kr
†††† 종신회원 : 한국과학기술원 전자전산학과 교수
joon@camars.kaist.ac.kr
논문접수 : 2002년 12월 11일
심사완료 : 2003년 6월 20일

1. 서 론

오늘날은 인터넷(Internet)과 셀룰라 네트워크(cellular network)와 같은 무선통신의 발달로 인해 언제 어디서든지 데이타를 주고 받을 수 있게 되었다. 그러나 셀룰라 네트워크와 같은 인프라스트럭처 기반 네트워크(infrastructured network)가 통신에 드는 비용이 높다는 점에서 상대적으로 저비용인 애드혹 네트워크(ad

hoc network)의 유용성이 주목을 받고 있다.

애드혹 네트워크는 인프라스트럭쳐 없이 노드(node)들이 필요에 따라 즉각적이고 임시적으로 네트워크를 형성하여 패킷 데이타(packet data)를 주고 받는 망의 한 형태이다. 중앙의 어떠한 통제장치도 없이 모든 노드들이 자율적으로 구성되기 때문에 순수한 분산컴퓨팅의 형태를 띠고 있고 새로운 노드의 참여, 탈퇴, 자유로운 이동이 가능하다는 특징이 있다. 망상에 존재하는 모든 노드들이 다른 노드의 통신을 위해서 라우터의 역할을 하기도 하고 통신의 주체가 되어 데이타를 주고 받기도 한다[1].

애드혹 네트워크에 대한 연구는 전혀 새로운 것이 아니다. 그러나 최근에 블루투스(Bluetooth)[2]나 IEEE 802.11의 애드혹 모드(ad hoc mode)[3]와 같은 기반 기술을 통해서 애드혹 네트워크는 더 많은 사람들의 관심을 갖게 되었다. 아직까지도 애드혹 네트워크에 대한 많은 연구들[4][5]이 네트워크 하부 계층에 관심을 가지고 있는 것이 현실이나 최근 들어서 몇몇 연구들은[6][7] Peer-to-peer 데이타 모델의 적용과 함께 애드혹 네트워크를 응용이나 정보, 서비스의 공유와 같은 상위계층의 입장에서의 분석을 주된 연구로 삼고 있으며, 이러한 상위 계층에 관한 연구들에서는 데이타를 얼마나 효율적으로 가져올 수 있느냐는 것이 매우 중요하다. 또한 최근의 P2P 시스템에서는 자신이 가지고 있는 하드웨어 자원을 공유함으로써 전체 시스템의 성능을 향상시키는 방법에 대해서도 연구되고 있다[8]. 따라서 본 논문에서는 이를 실현하기 위하여 애드혹 네트워크에서 각 노드들이 완전히 분산된 데이타에 대한 액세스를 하는 P2P 응용을 위한 데이타의 배치와 재사용에 관한 새로운 제안을 하고자 한다.

멀티 홉 애드혹 네트워크(multi-hop ad hoc network)는 망의 특성상 통신에 따르는 오버헤드(overhead)가 높아 통신 환경이 좋지 못하다. 이것은 크게 다음과 같은 이유로 나타나는 현상이다.

첫째, 무선 통신 채널 자체의 문제점이다. 무선 통신은 유선 구간에 비해서 연결성이 약하고 채널의 용량도 작다. 더군다나 전체 채널 용량을 주변의 여러 노드가 공유함으로써 개별적이고 순간적인 채널의 용량은 이것보다 더욱 낮아지게 된다[9]. 또, TCP와 같은 전송계층의 프로토콜이 무선구간의 효율성을 더욱 떨어뜨리는 역할을 하기도 한다[5].

둘째, 분산된 컴퓨팅 환경에 따른 문제점을 들 수가 있다. 애드혹 네트워크에서는 노드들이 모두 분산된 형태로 서로의 정보를 공유하고 있기 때문에 통신 계층이 높아짐에 따라 많은 오버헤드가 따른다. 예를 들면 라우팅 정보 유지나 데이타와 서비스를 찾고 가져오는데 드는 비용도 많이 든다.

셋째, 멀티 홉 애드혹 네트워크에서 노드의 숫자가 증가할수록 중간에 데이타를 전달(relay)해주는 노드에 가해지는 부담이 커지게 된다. 애드혹 네트워크에서는 모든 노드들은 라우터의 역할을 한다. 그런데 망의 노드 수가 늘어나고 망의 크기가 커지면 그만큼 중계해주는 데이타가 늘어나게 되고, 이 것은 노드들이 데이타를 중계하기 위한 채널의 사용이 증가하게 되는 결과를 가져와서 전체적으로 채널의 용량을 떨어뜨리는 효과를 발생시킨다.

애드혹 네트워크의 통신에 따른 오버헤드를 줄이는 방법 중에서 중요한 한가지가 데이타 캐싱(caching)이다. 그러나 휴대용 단말장치들은 비교적 저장공간이 부족하기 때문에 충분한 량의 데이타를 저장하여 재사용할 수 없으므로 캐싱능력이 떨어진다고 할 수 있다.

본 연구에서 이러한 점을 극복하기 위하여 다음과 같은 알고리즘을 제안한다. 첫째, 이웃 노드와 캐싱 공간을 공유하는 방법을 제안하였다. 이것은 대치 알고리즘(replacement algorithm)을 이용하여 자기자신의 저장공간과 이웃 노드의 저장공간에 대한 다단계 캐싱을 사용하고 관리하는 방법이다. 본 논문에서는 이러한 캐싱 기법을 이웃 캐싱(neighbor caching)이라고 명하였다. 그러나 여기서는 순수한 분산 환경을 고려하고 있기 때문에 전체적으로 최적화된 공유 방법을 찾기는 어렵다. 노드는 이웃 노드들이 앞으로 얼마만큼의 기간 동안 일하거나 쉴 것인지 예측할 수 없으므로 특정순간에 이웃에 저장된 데이타가 이후에 얼마 동안 이웃에 남아있을 것이라는 것도 예측하기 힘들다. 따라서 둘째, 데이타를 저장할 수 있는 이웃 노드들 중에서 캐싱된 데이타가 가장 오랫동안 남아있을 가능성이 높은 이웃 노드를 선별하는 방법을 제안하였다.

제안된 방법은 실제에서 일어날 수 있는 가능성이 높은 시나리오를 바탕으로 제시된 데이타 액세스 패턴, 노드의 쉬는 시간의 길이와 빈도에 관한 패턴을 가지고 실험되었다. 그 결과, 다양한 변수와 환경에서도 대체로 성능 향상이 이루어 짐을 알 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구에서 제안하고 있는 이웃 캐싱에 대한 전반적인 동작방식과 알고리즘을 제안한다. 3장에서는 실험 환경에 대해 설명하고 구현된 알고리즘을 바탕으로 자기 캐싱과 이웃 캐싱을 비교한 모의실험의 결과를 다양한 측면에서 분석한다. 마지막으로 4장에서 본 논문의 결론과 향후 연구 과제에 대해 기술한다.

2. 이웃 캐싱(Neighbor Caching)

2.1 이웃 캐싱(Neighbor Caching) 기법

이웃 노드라는 용어의 정확한 의미는 어떤 노드에서 물리적으로 1홉 거리에 있는 노드들을 의미한다. 이웃 캐싱은 모든 노드들이 일을 하는 시간, 즉 데이타를 액세스하는 시간이 있고 또 아무 일도 하지 않고 단지 주변 노드들의 데이타를 중계만 하는 시간도 있다라고 하는 가정에서 출발한다. 이렇게 자기 자신을 위한 일을 하지 않는 '쉬는 시간(idle time)'에 이웃 노드를 위해서 자기의 캐싱 공간을 제공하는 것이다.

기본적인 동작 방식은 다음과 같다. 우선 현재 데이타 액세스를 하고 있는(active time) 어떤 노드는 가져온 데이타를 나중에 다시 사용하기 위해 자기의 캐싱 공간에 저장 한다. 그러면 대치 알고리즘에 의해 자신의 캐싱 공간에서 퇴출된 데이타는 이웃 노드들 중에 하나의 노드에 저장된다. 나중에 그 데이타에 대한 요구가 발생했을 때 그 노드는 비록 자기 자신에게는 데이타가 없지만 이웃 노드에게 이전에 맡겨두었기 때문에 원래의 근원지로부터 데이타를 가져오는 대신 이웃 노드로부터 데이타를 가져 오는 것이다.

이웃 캐싱은 중앙에서 전체적으로 통제하는 통제장치가 있는 것이 아니라 개별적이고 지역적인 운영으로 동작한다. 따라서 이웃 노드에 캐싱 하는 과정에서 어떻게 하는 것이 최적의 경우인지를 알기 힘들다. 더구나 한 노드의 캐싱 공간은 주변 이웃 노드들이 모두 경쟁하면서 사용하고 있어서 최적의 상황을 알기 위해서는 모든 주변 노드들의 상황을 알아야 하는 어려움이 있다. 어떤 경우에는 이웃 노드에 데이타를 저장했는데 그 이웃이 굉장히 분주하게 작업하고 있다면 저장된 데이타가 곧 그 이웃 노드로부터 버려질 것이고 어떤 경우에는 이웃 노드에 저장한 데이타가 매우 낮은 우선순위를 가지고 있어서 쉽게 퇴출 당할 수도 있을 것이다. 그렇게 재사용하지 못하고 버려지는 데이타가 많다면 저장된 데이타의 적중률이 낮아지게 되고 이웃 캐싱으로 얻어지는 이득보다는 손해가 많을 것이다.

이웃 노드에 저장되어 있는 데이타의 적중률을 높이는 것이 이웃 캐싱의 성능에 매우 중요한 영향을 미친다. 적중률을 높이기 위해서는 첫째, 가장 데이타를 오래 보존할 수 있는 최적의 이웃 노드를 찾아야 한다. 둘째, 오래 저장될 수 있을 가능성이 높은 데이타만 이웃 캐싱 해야 된다.

본 논문에서는 이를 해결하기 위해서 우선순위에 근거한 예측기법(raking based prediction)을 제안한다. 이 방법은 주변의 정보를 모으고 유지하는 데 많은 노력을 들이지 않고 가지고 있는 정보를 활용하여 캐싱하고자 하는 데이타가 가장 오래 유지될 가능성이 높은 이웃을 선택한다. 그리고 무계획적으로 데이타가 이웃에 저장되는 것을 막고 재사용될 가능성이 높은 데이타만을 저장하도록 하여 데이타의 적중률을 높인다. 따라서 이웃 캐싱에 드는 오버헤드는 줄이고 효율을 극대화 하도록 고안되었다.

2.2 우선순위에 근거한 예측 기법(Ranking Based Prediction)

이 절에서는 우선순위에 근거한 예측 기법을 제안하고 그 기법을 중심으로 이웃 캐싱의 세부적인 동작방식을 설명하도록 하겠다. 전체는 크게 다섯 개의 단계로 나뉘어져 설명된다.

그림 1에서 그림 3까지 우선순위에 근거한 예측 기법과 이웃 캐싱의 세부적인 동작방식을 설명하고 있다. 그림의 용례는 다음과 같다. 그림에서는 노드 A와 이웃 노드들인 X, Y, Z로 구성되어 있다. 그림에서 여러 개의 층으로 이루어진 상자는 각 노드들의 캐싱 공간을 의미한다. 그곳에는 데이타가 들어있고 우선순위 값에 의해서 정렬된 순서이다. 캐시는 10개의 데이타를 저장할 수 있는 크기이며 각각의 데이타는 고유한 식별자, 데이타의 소유자, 우선순위 값으로 표시되었다. 예제에서는 대치 알고리즘이 LRU(Least Recently Used)값이므로 우선순위 값은 최근 사용된 시간으로 나타내어 진다. 데이타를 이웃에 저장할 때 기준이 되는 우선순위 기준 값(ranking threshold)은 어둡게 칠해진 상자에 따로 명시되었다. 예제에서는 우선순위 기준 값이 전체 캐싱 공간의 50%에 해당하는 데이타의 우선순위 값이다.

첫 번째 단계는 자기의 캐시에서 퇴출되는 데이타를 이웃 노드에게 보내기 위해 우선순위에 근거한 예측 기법으로 최적의 이웃 노드를 선택하는 단계이다. 여기서 우선순위 기준 값을 이용하는데 우선순위 기준 값은 모든 노드가 가지고 있는 값으로 그 값보다 우선순위가 낮은 데이타는 이웃 캐싱 되지 않도록 하는 기준이 되는 값이다. 우선순위 기준 값을 정하는 이유는 이웃 노드에 저장되는 데이타의 적중률을 높이기 위해서이다. 만일 우선순위가 아주 낮은 데이타가 이웃 노드에 캐싱된다면 시간이 얼마 지나지 않아서 그 데이타가 이웃 노드로부터 퇴출될 가능성이 높다. 따라서 오랫동안 이웃노드에게 남아있을 수 있는 우선순위 높은 데이타만 캐싱 하여 적중률을 높인다.

어떤 노드가 외부에서 새로운 데이타를 가져오면 자기의 캐싱 공간에 그 데이타를 집어넣게 된다. 그렇게 하면 기존에 있던 데이타 중에 가장 우선순위가 낮은 데이타는 퇴출 당하게 된다. 퇴출 당하는 데이타의 우선순위 값이 특정 이웃의 우선순위 기준 값보다 높다면 그 이웃에 저장될 가능성이 있게 된다. 만일 퇴출 당하는 데이타의 우선순위 값이 모든 이웃들의 우선순위 기준 값보다 낮다면 퇴출 당하는 데이타는 이웃 캐싱 될 수 없으므로 그냥 버려진다. 그리고 우선순위 기준 값이

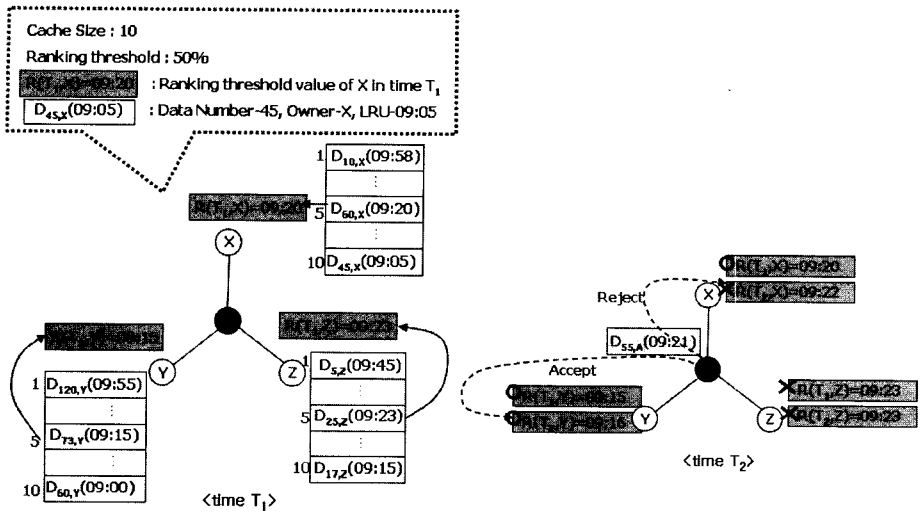


그림 1 우선순위에 근거한 예측기법을 통해 최적의 이웃을 선택하는 과정

퇴출 당하는 데이타의 우선순위 값보다 낮은 이웃 노드들이 하나보다 많다면 그러한 이웃 노드들 중에서 하나를 고르는 과정이 따르게 된다.

여러 개의 후보들 중에 어떤 이웃 노드를 선정하는가에 대한 기준은 이웃 캐싱된 데이타가 얼마나 오랫동안 해당 이웃 노드에서 퇴출 당하지 않고 오랫동안 살아있을 가능성이 높은 가이다. 이것 역시 오랫동안 남아있는 데이타가 적중률이 높기 때문이다. 여기서는 이웃 캐싱 될 데이타의 우선순위 값과 비교했을 때 우선순위 기준 값이 가장 차이가 많이 나는 이웃을 그런 노드로 본다. 즉, 두 값의 차이가 많이 난다는 것은 이웃 캐싱 될 데이타가 그 이웃 노드로 이동이 될 경우 그곳에서 보다 높은 순위를 차지하게 되고 이는 앞으로 퇴출 당할 시기가 늦어질 가능성이 가장 높다는 것을 의미하기 때문이다.

그림 1에서 보면 시간 T1일 때 노드 A의 이웃 노드들인 X, Y, Z의 우선순위 기준 값은 각각 09:20, 09:15, 09:23이다. 이 값들을 가지고 시간 T2일 때 노드 A는 자기에게서 퇴출 당하는 데이타 D55를 이웃 노드들 중에 하나에 캐싱하려고 한다. 그림에서 나타난 예에서는 우선순위 값이 LRU값이므로 우선순위 값이 높다는 것은 더 최근에 참조된 데이타 값이라는 의미이다. 시간 T2일 때 데이타 D55의 우선순위 값이 09:21이므로 시간 T1일 때의 이웃 노드 X, Y의 우선순위 기준 값보다 높다. 시간 T1과 T2의 차이로 인해 시간 T2때에는 노드 X, Y, Z의 우선순위 기준 값들이 조금씩 상승할 수 있다. 따라서 이웃 노드 X, Y, Z의 시간 T2일 때 우선순위 기준 값은 시간 T1일 때 값보다 적어도 같거나 크다. 즉 노드 A는 시간 T2일 때 노드 X, Y가 이웃 캐시를 할 수 있는 노드라고 확신할 수는 없지만 적어도 노드 Z는 데이타 D55를 저장할 수 없는 노드라는 것을 확신할 수 있다. 가능성 있는 두 개의 이웃 노드들 중에서 어떤 것을 선택할 것인지는 노드 X, 노드 Y의 우선순위 기준 값과 데이타 D55의 우선순위 값 차이를 보고 판단한다. 예에서는 노드 Y의 우선순위 기준 값과 데이타 D55의 우선순위 값 차이가 더 많이 나므로 노드 Y가 최종적으로 데이타 D55를 저장할 가장 최적의 후보로 선정된다.

두 번째 단계는 첫 번째 단계에서 선정된 이웃 노드와 협상하는 단계이다. 데이타를 이웃에게 보내고자 하는 노드는 예측에 의해 선정된 최적의 이웃과 협상을 한다. 보내고자 하는 데이타의 우선순위 값을 포함한 협상용 패킷을 이웃 노드에게 보내면 이웃 노드는 그 데이타가 자기에 캐싱 공간에서 우선순위 기준 값보다 높은지 확인하여 만일 저장 가능하다면 긍정적인 응답을 보낼 것이고 그렇지 않다면 거부를 한다. 만일 협상이 결렬되면 데이타를 보내고자 하는 노드는 다음으로 적당한 이웃 노드와 협상을 하게 된다.

그림 1을 보면 노드 A는 시간 T1일 때 수집된 이웃 노드의 우선순위 기준 값을 근거로 최적의 이웃 노드를 노드 Y로 선정하고 데이타 D55를 노드 Y에게 보내기 위해 시간 T2때 협상을 하게 된다. 협상을 통해 노드 Y의 우선순위 기준 값이 변한걸 알 수 있지만 노드 Y의 우선순위 기준 값은 여전히 데이타 D55의 우선순위 값보다 낮다. 따라서 긍정적인 응답을 할 것이고 이렇게 하면 협상은 이루어 지는 것이다. 만일 최적의 이웃 노

드인 노드 Y가 없고 대신 X가 최적의 이웃 노드라고 가정하면 노드 A는 노드 X와 협상을 하게 될 것이다. 시간 T1때는 노드 X가 캐싱 가능한 노드였지만 협상을 통해 시간 T2때는 노드 X의 우선순위 기준 값이 데이타 D55의 우선순위 값보다 크다는 걸 알게 될 것이다. 따라서 요청은 거부되며 노드 A는 데이타 D55를 노드 X에 캐싱 할 수 없게 된다. 이렇게 협상에서 실패하면 다음으로 가능성 있는 노드와 협상을 해야 한다. 그러나 우선순위 기준 값이 아주 엄격한 값이 아니므로 이런 시간적 차이에 의한 오차는 허용하고 캐싱 가능하도록 하면 이웃 캐싱을 좀 더 유연하고 간단하게 만들 수 있다.

세 번째 단계는 협상이 이루어진 후 실제로 데이타를 이웃 노드에 저장하는 단계이다. 데이타가 이동하여 이웃 노드의 캐싱 공간에 저장되면 그 데이타의 소유자는 그 데이타를 맡긴 노드가 되고 데이타에 소유자 정보가 같이 저장된다. 만일 우연히 저장하려는 데이타가 이웃 노드에 존재한다면 굳이 데이타를 이동시키는 것이 아니라 그 데이타의 소유자 리스트에 데이타를 맡기고자 하는 노드를 추가하면 된다.

네 번째 단계는 이웃 캐싱 된 데이타에 대해 적중(hit)이 되었을 때이다. 노드는 사용자로부터 데이타에 대한 요청이 발생할 때 우선적으로 자기의 캐시를 살펴본다. 만약 자기의 캐시에 요청된 데이타가 있다면 그 데이타를 가져다가 사용하면 되고, 요청된 데이타가 없다면 이웃 노드들에 저장되어 있는 자기 소유의 데이타들을 살펴본다. 앞에서 설명된 세 번째 단계에 의하여 이웃 노드들에 저장되어 있는 데이타들의 정보는 데이타의 소유자에 의해서 파악되고 있으므로 추가적인 검색은 필요 없다. 만일 이웃 노드 중에 요청된 데이타가 있다면 멀티 홉 통신을 이용하여 원래의 근원지로부터 데이타를 가져오는 것이 아니라 이웃 노드로부터 데이타를 가져온다. 가져와서 사용된 데이타는 우선순위 값이 재조정되기 때문에 우선순위 값에 따라 다시 자기 캐시에 저장될 수 있다.

그림 2에서의 예를 한번 살펴보자. 노드 A가 사용자로부터 데이타 D55에 대한 요청을 받고 우선적으로 자기의 캐싱 공간을 본다. 이전에 사용한 데이타들이 저장된 자기 캐시에는 데이타 D55가 없기 때문에 노드 A는 이웃 노드들의 캐싱 공간을 살펴 보게 된다. 이웃 노드들 중에 노드 Y에 데이타 D55가 존재하므로 노드 A는 데이타 D55를 원래의 근원지에게 요청하는 것이 아니라 노드 Y에게 요청한다.

다섯 번째 단계는 이웃 캐싱된 데이타가 버려질 때 그 데이타를 저장하고 있던 이웃 노드가 데이타의 소유자에게 이 사실을 알리는 단계이다. 이웃 캐싱으로 말미

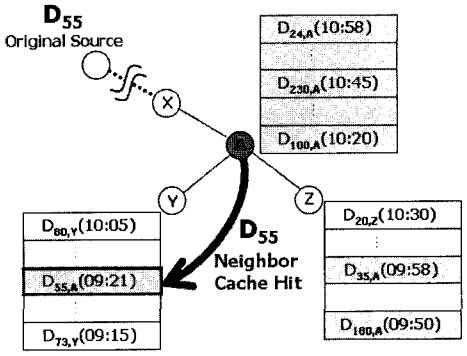


그림 2 이웃 캐싱 된 데이타의 적중(hit)

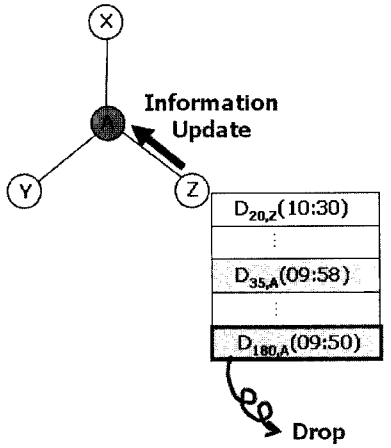


그림 3 업데이트(update)

암아 한 노드의 캐싱 공간에는 자기소유의 데이타뿐만 아니라 이웃에 의해 맡겨진 데이타도 공존한다. 버려지는 데이타가 자신의 데이타라면 그 데이타를 자신의 이웃 노드에게 이웃 캐싱을 할 수 있는 가능성이 존재하므로 첫 번째 단계부터 시작하여 이웃 캐싱의 절차를 밟는다. 그러나 버려지는 데이타가 자신의 데이타가 아니라 이웃 노드에 의해 이웃 캐싱 된 데이타라면 반복적으로 이웃 캐싱 되는 일 없이 데이타를 버리고, 이 데이타를 가지고 있던 노드는 데이타의 소유자에게 이 사실을 알려준다. 이것은 한번 이웃 캐싱 된 데이타가 또 다시 다른 노드로 이웃 캐싱 되는 절차를 여러 번 밟음으로써 데이타가 원래의 소유자에게서부터 1홉 이상의 거리에 있는 노드로 흘러 들어가게 되고 원래의 소유자는 자신의 데이타에 대한 정보를 유지하기 위해 점점 복잡한 과정을 겪게 되는 것을 막기 위함이다.

그림 3의 예에서 노드 Z에 저장된 데이타 중에서 우선순위가 가장 낮은 데이타 D180가 버려지는 상황에 처했다. 이 데이타의 소유자가 노드 Z라면 첫 번째 단계

부터 시작하여 이웃 노드에 데이타를 캐싱 하기 위한 절차를 밟을 것이다. 그러나 데이타 D180는 노드 A가 소유자인 이웃 캐싱된 데이타이다. 따라서 데이타 D180는 그냥 버려지고 이 사실을 노드 Z가 노드 A에게 알린다. 이렇게 이웃 노드로부터 받은 정보를 가지고 노드 A는 자기 소유의 데이타들에 대한 정보를 갱신하게 된다. 만일 소유자가 하나가 아니라 여럿이라면 이 과정을 모든 소유자에게 알려야 한다.

3. 실험

3.1 실험환경

3.1.1 시뮬레이터 및 설정사항

본 연구에서는 시뮬레이션을 통해서 성능 평가를 하였다. 본 실험에서는 ns2 시뮬레이터를 이용하였는데, 우리는 이를 통하여 데이타를 전송하고 중계하는 것을 포함하는 모든 네트워크 전송량과 본 논문에서 제시한 방법에 따른 부하를 측정하였다. 실험에서 사용된 시뮬레이터 환경, 자세한 설정사항, 변수들, 측정값(metric), 비교 대상은 표 1에 나와 있는 것과 같다.

3.1.2 사용자의 데이타 액세스 패턴과 사용 시간 패턴

본 논문에서는 실험에 필요한 몇 가지의 패턴을 제시한다. 이러한 패턴을 정확하게 유추해내기는 쉽지 않으나 실제 환경에 좀더 근접하고 실험하기 용이한 방향으로 설정을 하였다. 실험에서 필요한 패턴들은 아래와 같다.

첫 번째 데이타 액세스 패턴이다. 본 실험에서 사용된 데이타의 액세스 패턴은 일반적으로 널리 알려진 것 중의 하나인 Zipf-like distribution[11]을 사용하였다. Zipf-like distribution에 따르면 i번째 인기 있는 데이타는 $1/i^{\alpha}(\alpha \approx 1)$의 비율로 액세스 된다.

두 번째는 쉬는 시간의 길이이다. 즉 노드들이 얼마 동안 쉴 것인지에 대한 패턴이 있어야 한다. 이러한 패턴은 첫 번째의 경우보다 더욱더 일반화하기 어렵다. 실제 세계에서는 매우 분주한 노드와 계속해서 쉬는 노드들 사이에 다양한 패턴이 있을 수 있으나 실험에서는 일반적인 상황을 고려하여 모든 노드가 데이타 액세스에 참여하여 일한 시간에 대해 특정 비율로 쉬도록 하였다.

세 번째는 쉬는 시간의 빈도 이다. 두 번째에서 언급된 가정에 따라 일정 비율의 시간 동안 노드가 쉬는 시간을 가진다 하더라도 얼마의 빈도로 쉬는 지에 대한 정의가 있어야 한다. 즉, 쉬는 빈도가 잦다면 노드는 조금 일하고 조금 쉬는 것을 자주 반복할 것이며 쉬는 빈도가 드물다면 노드는 많은 일을 하고 또 그만큼 많은 시간 동안 쉬게 될 것이다. 실험에서는 쉬는 시간의 빈도를 고정시켰다.

3.2 실험 결과

3.2.1 노드 개수 변화에 따른 결과

그림 4는 노드 개수를 변화시켜 실험을 수행했을 때 모든 노드들이 실험을 끝내는데 걸린 평균 수행시간과 그때 망의 전체 트래픽을 측정한 그래프이다. 이 때 사용된 캐시의 크기는 5이고 우선순위 기준 값은 0.6이다. 쉬는 시간의 빈도는 5%이고 쉬는 시간의 길이는 일한 시간의 10배를 쉬게 하였다.

그림 4를 보면 노드의 개수가 증가하면서 이웃 캐싱의 성능 개선이 뚜렷해짐을 알 수 있다. 왼쪽 그래프에

표 1 실험에 적용된 세부설정들

Simulator	√ ns2 (2.1b9a) - UC Berkeley √ MAC : IEEE 802.11 √ 1 hop cell coverage : 250 m √ Routing protocol : DSR[10] √ Transport layer : TCP (tao-TCP), UDP (for control packet)
Constants	√ Number of job : 100 √ Total number of data : number of nodes * 10 √ Sampling data set : 100 √ Data size : 2K, Control packet size : 16 √ User access pattern (Zifp-like distribution $1/i^{\alpha}$, with $\alpha \approx 1.0$) √ eplacement algorithms : LRU (Least Recently Used)
Parameters	√ Cache size √ Ranking threshold √ Network size (number of nodes) √ Idle time
Metrics	√ Data access time (average ending time) √ Traffic (total number of packets)
Comparison	√ Self caching (caching in its own storage) vs. Neighbor caching

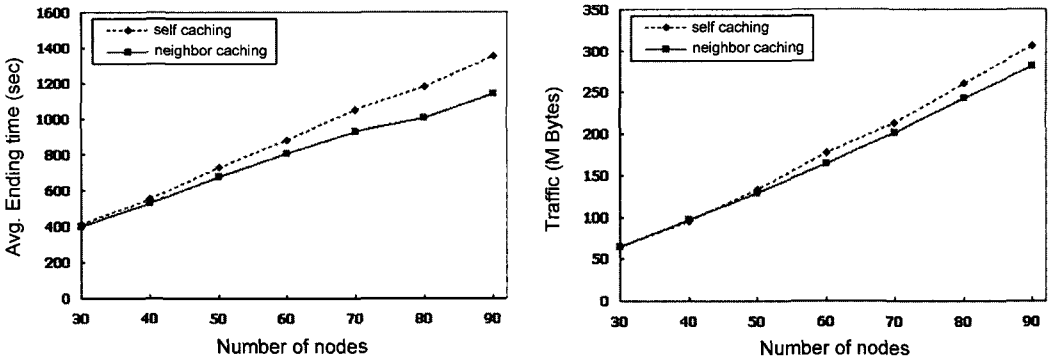


그림 4 노드 개수의 변화에 따른 성능 변화

서는 노드 개수가 30개일 때에는 자기 캐싱과 이웃 캐싱의 성능 차이가 거의 없지만 노드 개수가 90개일 때는 약 15%정도 수행시간이 단축되는 것을 볼 수 있다. 이는 노드 수가 작을 때는 데이타를 가져오는 평균 홉 수가 작기 때문에 이웃 캐싱을 통해 데이타를 이웃에서 가져오는 것이 근원지로부터 데이타를 가져오는 것에 비해 크게 효율적이지 않기 때문이다. 그러나 노드 수가 증가하면서 상대적으로 이웃 캐싱으로 데이타를 가져오는 것의 효율성이 증가하면서 성능에서 뚜렷한 개선을 보인다. 망 전체의 트래픽을 나타내는 오른쪽 그래프에서는 수행시간에서 보다 성능향상이 좀 낮지만 역시 노드 수 증가에 따라 성능 향상의 폭이 커짐을 알 수 있다. 이 때는 90개 일 때는 약 8%의 트래픽 절감이 이루어진다.

3.2.2 캐시 크기 변화에 따른 결과

그림 5는 캐시 크기 변화에 따른 성능향상의 변화를 나타낸다. 이 실험에서는 노드의 개수가 50개로 고정되었으며 우선순위 기준 값은 0.5(50%)이다. 쉬는 시간의 길이와 빈도는 앞의 실험과 동일하다.

그림 5에서 이웃 캐싱은 캐시의 크기가 작으면 작을수록 성능향상이 많이 되고 캐시의 크기가 커지면서 자기 캐싱만 하는 경우와의 성능 차이가 줄어드는 것을 알 수 있다. 이것은 캐시의 크기가 커짐으로 해서 자주 액세스 되는 데이타의 대부분은 자기 캐시에서 적중이 되고 캐시에서 퇴출 당하는 데이타의 수가 줄어들게 되기 때문이다. 따라서 이웃 캐싱되기 위해 이웃 노드로 이동하는 데이타의 수와 적중되는 데이타의 수가 줄어들게 되어 결국 이웃 캐싱으로 인한 성능 개선은 줄어들게 되는 것이다.

실험에서 적용된 최소의 캐시 크기는 5이다. 이것은 하나의 노드가 액세스하는 전체 데이타의 집합이 100개이므로 이중에 5%의 데이타를 저장할 수 있는 크기이다. 전체 데이타의 집합에 5%의 데이타를 저장할 수 있다는 것은 실제로는 상당히 큰 저장공간이 될 수 있으나 실험의 편리성을 위해서 캐시의 최소 크기로 설정된 값이다. 따라서 전체 데이타의 집합이 커지고 캐시의 크

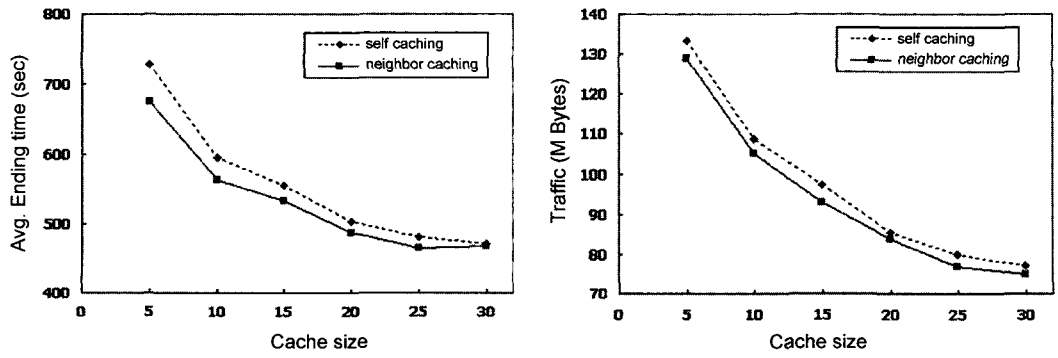


그림 5 캐시 크기의 변화에 따른 성능 변화

기가 상대적으로 작아진다면 이웃 캐싱을 통한 성능향상이 더욱 뚜렷해질 것이라는 것을 알 수 있다.

3.2.3 쉬는 시간과 우선순위 기준 값의 변화에 따른 결과

그림 6은 쉬는 시간의 길이 변화에 따른 이웃 캐싱과 자기 캐싱의 성능 비교와 함께 이웃 캐싱에서도 우선순위 기준 값이 성능에 어떠한 영향을 미치는 지 동시에 보여주는 그래프이다. 이 그래프에서 X축은 우선순위 기준 값(ranking threshold)인데 이 값은 이웃 캐싱일 때 0.1~0.9 사이에 있다. 그리고 우선순위 기준 값이 0인 지점은 이웃 캐싱을 하지 않는 다는 의미로 자기 캐싱을 나타내고 1인 지점은 이웃 캐싱은 하되 우선순위 값에 근거한 예측기법을 사용하지 않고 이웃 노드에게 차례로 데이타를 캐싱 하는 라운드 로빈(round robin) 방식의 이웃 캐싱을 나타낸다. Y축은 성능 비교를 위해 자기 캐싱의 결과 값에 의해 정규화(normalization)하였고 각각의 꺾은선들은 쉬는 시간의 변화에 따른 성능 변화를 나타낸다. 실험에서 적용된 노드의 개수는 50개이고 캐시의 크기는 10이다.

먼저 쉬는 시간의 길이 변화가 성능상에 미치는 영향을 살펴보면 쉬는 시간의 길이가 길어질수록 그래프가 아래쪽으로 이동하는 것을 볼 수 있는데 이는 이웃 캐싱으로 인한 성능 개선이 커진다는 것을 의미한다.

다음으로 우선순위 기준 값의 변화가 성능상에 미치는 영향을 살펴보기 위해서는 하나의 꺾은 선에서 각 점들의 값을 살펴보면 된다. 대체로 이웃 캐싱에서 우선순위 기준 값이 0.3~0.5일 때 최적의 성능을 나타내는 것을 알 수 있다. 이 수치는 노드들이 분주해질수록 더욱 앞당겨지는 경향이 있다. 즉, 노드들이 분주하게 동작하고 있을 때일수록 이웃 캐싱의 횟수를 줄여서 이웃 캐싱으로 인하여 발생하는 오버헤드를 최소로 하고 얻는 이익을 최대로 하는 것이 좋다는 것을 의미한다. 반대로 노드들이 한가해질수록 이웃 캐싱을 좀 더 많이 하여서 이웃 노드에게 가져오는 데이타의 수를 늘리는 것이 바람직하다.

이와 함께 그림 6에서는 우선순위에 근거한 예측 기법을 통한 이웃 캐싱과 예측을 하지 않고 단순히 주변 노드들에게 돌아가면서 데이타를 저장하는 라운드 로빈 방식과의 비교도 볼 수 있다. 그래프에서 우선순위 기준 값이 1인 경우가 라운드 로빈 방식인데 대체적으로 예측기법을 적용한 경우보다 성능이 나쁨을 알 수 있다. 실험에서는 모든 노드들이 동일한 패턴을 가지고 동작하기 해서 특별히 더 바쁘거나 한가한 노드는 없는 상황이다. 이 때문에 라운드 로빈 방식에 의한 이웃 캐싱과 우선순위에 근거한 예측기법을 적용한 이웃 캐싱과의 성능 차이가 크지 않다. 그러나 만일 이웃 노드들의 패턴이 매우 다양해서 분주한 노드와 한가한 노드의 차이가 명확하다면 라운드 로빈 방식에 의한 이웃 캐싱은 상당히 비효율적으로 나타날 것이다.

3.2.4 쉬는 시간의 길이 변화에 따른 이웃 캐싱의 적중률 변화

그림 6에서 우선순위 기준 값과 쉬는 시간의 길이가 변화했을 때 성능 변화를 살펴보았다. 이 실험에서 우선순위 기준 값이 0.5일 때를 기준으로 쉬는 시간에 따른 이웃 캐싱을 위한 데이타의 이동과 1홉 적중 횟수, 적중률을 분석한 것이 그림 7이다. 이 그림에서 X축은 쉬는 시간의 길이를 나타낸다. 왼쪽의 Y축은 막대 그래프의 값을 나타내는데 1홉 데이타의 이동횟수와 1홉 적중을 표시하기 위함이고 오른쪽의 Y축은 꺾은 선 그래프의 값으로 1홉 적중률을 나타내고 있다.

이 그래프에서는 우선순위에 근거한 예측기법을 적용하면 쉬는 시간의 길이 변화에 따라 이웃 노드로 이동하는 데이타 수가 변화한다는 것을 알 수 있다. 즉, 노드들이 분주해질수록 이웃으로 이동되는 데이타 수는

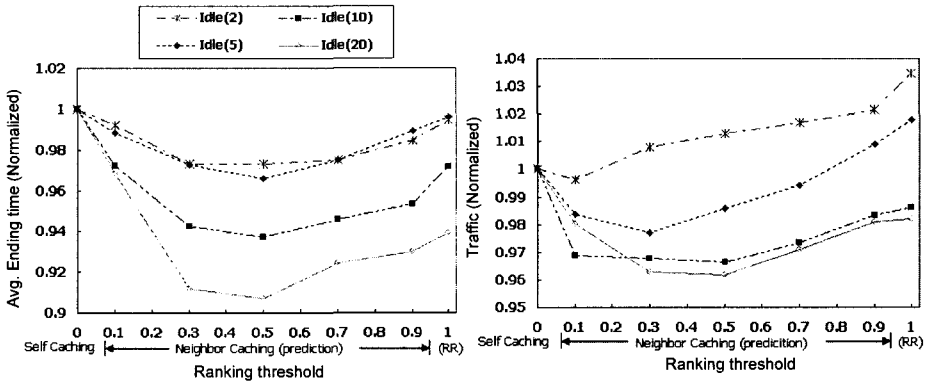


그림 6 쉬는 시간의 길이와 우선순위 기준 값의 변화

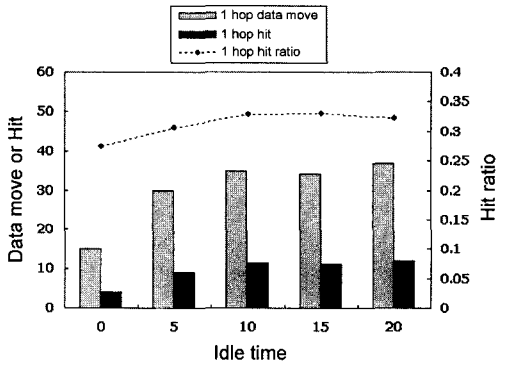


그림 7 쉬는 시간의 길이 변화에 따른 이웃 캐싱의 적중률 변화

줄어들게 되는데 이것은 분주한 노드들 사이의 트래픽 증가에 따른 성능 저하를 막을 수 있게 해준다.

또, 그림 7을 통해 노드들이 분주해지면 우선순위에 근거한 예측기법에 의해 이웃 노드로 이동되는 데이타의 수는 줄어들지만 1홉 적중률은 일정 정도로 유지가 되는 것을 볼 수 있다. 따라서 우리는 우선순위에 근거한 예측기법이 망의 분주함에 대해서 어느 정도 탄력적(adaptive)으로 반응하는 방식임을 알 수 있다.

4. 결론 및 향후 연구 과제

본 논문에서는 애드혹 네트워크를 기반으로 하여 분산된 데이타를 서로 주고 받는 실험을 수행하였을 때 이웃 캐싱 기법을 통하여 효율성과 확장성을 증대시키는 방법을 제안하였다. 이웃 캐싱(neighbor caching) 기법은 쉬고 있는 이웃 노드의 저장 공간을 잠시 빌려 씀으로써 캐싱 공간을 확대하고 먼 거리에서 데이타를 가져오는 멀티 홉 무선 통신의 단점을 극복하는 방법이다. 이와 함께 이웃 캐싱에서 주변 노드들 중에서 최적의 이웃 노드를 선별해 내는 우선순위에 근거한 예측기법(ranking based prediction)을 제안하였다. 우선순위에 근거한 예측 기법을 통해 데이타가 가장 오랫동안 보관될 가능성이 높은 이웃 노드를 선별해내고 우선순위가 낮은 데이타를 이웃 캐싱하지 않을 수 있어서 효율성을 높일 수 있다.

다양한 변수들을 가지고 실험된 결과 일반적으로 노드의 개수가 늘어나 망의 크기가 커질 때, 노드들의 쉬는 시간이 길 때, 그리고 노드들의 캐시 크기가 작을 때 좋은 성능을 나타낸다. 그러나 노드들의 쉬는 시간이 비교적 짧은 경우에도 자기 캐싱(self caching)에 비해 나쁘지 않은 성능을 나타내며 우선순위에 근거한 예측기법을 통해 성능을 더욱 좋게 할 수 있다. 그리고, 우선순위에 근거한 예측기법은 노드들의 바쁜 정도에 따라 이웃으로 캐싱 하는 데이타의 수를 조절함으로써 이웃 캐싱의 적중률을 높이는 매우 적응력 있는 방식임을 알 수 있다.

앞으로의 연구에서는 노드의 다양한 작업 패턴에 대해서 이웃 캐싱과 우선순위에 근거한 예측기법이 얼마나 성능 개선을 가져 올 것인지에 대한 실험을 좀 더 할 필요가 있다. 또 이웃 캐싱의 영역을 좀 더 확대하여 지역적으로 저장 공간이나 데이타를 공유하는 방법을 연구하는 것도 필요할 것이다. 이와 함께 유선 네트워크에서 P2P 응용을 실행할 때 이웃 캐싱 기법을 적용하는 것도 연구해 볼만한 가치가 있다.

참 고 문 헌

[1] M. Frodigh, P. Johansson, and P. Larsson, "Wireless ad hoc networking-The art of networking without a network," Ericsson Review No. 4, 2000.

[2] J. Haartsen, "The Bluetooth Radio System," IEEE Personal Communications, February, 2000.

[3] Y. Lin, Y. Hsu, K. Oyang, T. Tsai, and D. Yang, "Multihop Wireless IEEE 802.11 LANs: A Prototype Implementation," IEEE International Conference on Communications 1999, 1999.

[4] E. Royer and C. Toh, "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks," IEEE Personal Communications, April, 1999.

[5] J. Liu and S. Singh, "ATCP: TCP for mobile ad hoc networks," IEEE Journal on Selected Areas in Communications, Vol. 19, Issue 7, July, 2001.

[6] T. Hara, "Effective replica allocation in ad hoc networks for improving data accessibility," Proceedings of INFOCOM 2001, Vol. 3, 2001.

[7] G. Kortuem, J. Schneider, D. Pruitt, T. Thompson, S. Fickas, and Z. Segall, "When Peero-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad hoc Networks," Proceedings of the 1st International Conference on Peer-to-Peer Computing, 2002.

[8] Krishna Kant, Ravi Iyer, and Vijay Tewari, "A Framework for Classifying Peer-to-Peer Technologies," Proceedings of CCGRID, May, 2002.

[9] J. Li, C. Blake, D. Couto, H. Lee, and R. Morris, "Capacity of Ad Hoc Wireless Networks," Proceedings of the 7th Annual International Conference on Mobile Computing and Networking, pp. 61-69, 2001.

[10] D. B. Johnson et al., "The Dynamic Source Routing Protocol for Mobile Ad hoc Networks," IETF Internet Draft. http://www.ierf.org/internet-drafts/draft-ietf-manet-dsr-02.txt, 1999.

[11] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions:

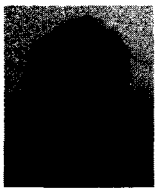
Evidence and implications," Proceedings of INFOCOM, April 1999.

조 준 호

2001년 2월 경북대학교 컴퓨터공학과 졸업. 2003년 2월 한국과학기술원 전산학과 졸업. 2003년 3월~현재 삼성전자 연구원. 관심분야는 홈네트워크 미들웨어, Wireless PAN, P2P

오 승 택

1974년 10월 19일생. 1997년 한국과학기술원 전산학과 졸업. 1999년 한국과학기술원 전산학 석사. 1999년 3월~현재 한국과학기술원 전기전산학과 박사과정. 관심분야는 병렬 컴퓨터 구조, 운영체제 등

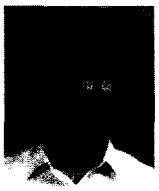


김 재 명

1983년 부산대학교 계산통계학과. 1985년 한국과학기술원 전산학 석사. 1985년~현재 ETRI 네트워크기술연구소 홈서비스관리팀(책임연구원). 관심분야는 임베디드시스템, 분산 컴퓨팅, 통신 프로토콜, 홈네트워킹

이 형 호

1977년 서울대학교 공업교육학과. 1979년 한국과학기술원 전자공학 석사. 1983년 한국과학기술원 전자통신 박사. 1983년~현재 ETRI 네트워크기술연구소 광가입자망연구 부장(책임연구원). 관심분야는 고속 라우터 기술, 이동 교환 기술, 초고속 광가입자 기술



이 준 원

1983년 서울대학교 계산통계학과 졸업(학사). 1990년 Georgia Tech. 전산학과(석사). 1991년 Georgia Tech. 전산학과(박사). 1983년~1986년 (주)유공 근무 1991년~1992년 IBM 근무. 1992년~현재 한국과학기술원 전산학과 부교수. 관심분야는 운영체제, 병렬처리 등