이웃 캐싱을 이용한 애드혹 망의 성능향상

조준호[+] 오승택[O++] 이준원[++]
[+]삼성전자 [++]한국과학기술원 전자전산학과
{jhcho, stoh, joon}@camars.kaist.ac.kr

Performance Increment of Ad Hoc Networks using Neighbor Caching

Joonho Cho[+] Seungtaek Oh[O++] Joonwon Lee[++]
[+]Samsung Electronics [++]Dept. of EECS, KAIST

요 약

애드혹 네트워크 상의 노드들이 서로의 분산된 데이터를 주고 받는 어플리케이션은 멀티 홉 무선 통신의 오버헤드로 인하여 효율성이 떨어진다. 이것을 극복하기 위해서 본 논문은 이웃 캐싱기법을 제안하고, 이 방법이 노드들의 독립적인 캐싱 방법보다 효율적이라는 것을 보이고 있다. 이웃 캐싱은 쉬고 있는 이웃 노드의 저장 공간을 잠시 빌려 씀으로써 캐싱 공간을 확대하고 먼 거리에서 데이터를 가져오는 멀티 홉 무선 통신의 단점을 극복하는 방법이다. 이와 함께 본 논문에서는 이웃 캐싱을 할 때 노드들 중에서 최적의 이웃 노드를 선별해 내는 우선순위에 근거한 예측기법을 제안하였다. 우선순위에 근거한 예측 기법을 통해 데이터가 가장 오랫동안 보관될 가능성이 높은 이웃 노드를 선별해내고 우선순위가 낮은 데이터를 이웃 캐싱 하지 않을 수 있어서 이웃 캐싱의 효율성을 높일 수 있다.

1. 서 론

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[1]. 또, TCP와 같은 전송계층의 프로토콜이 무선구간의 효율성을 더욱 떨어뜨리는 역할을 하기도 한다[2]. 둘째, 애드혹 네트워크에서는 노드들이 완전히 분산된 컴퓨팅을 하기 때문에 라우팅이나 데이터 검색과 같이 통신 계층의 상위 영역으로 갈수록 오버헤드가 증가한다. 셋째, 멀티 홉 애드혹 네트워크에서는 노드의 숫자가 증가할수록 중간에 데이터를 전달(relay)해주는 노드에 가해지는 부담이 커지게 되어 전체적으로는 채널의 용량을 떨어뜨리는 효과를 발생시킨다.

애드혹 네트워크의 통신에 따른 오버헤드를 줄이는 방법 중에서 중요한 한가지가 데이터 캐싱이다. 그러나 이동 기기들은 비교적 저장공간이 부족하기 때문에 충분한 량의 데이터를 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 방법인 이웃 캐싱(neighbor caching)을 제안하였다. 이것은 대치 알고리즘(replacement algorithm)을 이 용하여 자기자신의 저장공간과 쉬고 있는 이웃 노드의 저장공간에 대한 다단계 캐싱을 사용하고 관리하는 기법 이다. 둘째, 데이터를 저장할 수 있는 이웃 노드들 중에서 캐쉬 된 데이터가 가장 오랫동안 남아있을 가능성이 높은 이웃 노드를 선별하는 방법을 제안하였다.

모의실험을 통해 제안된 방법이 다양한 변수와 환경에서도 대체로 성능 향상이 된다는 것을 알 수 있다.

2. 이웃 캐싱(Neighbor Caching)

2.1. 이웃 노드와 저장 공간의 공유

이웃 노드는 물리적으로 1홉 거리에 있는 노드들을 의미한다. 이웃 캐싱은 모든 노드들이 데이터를 액세스하는 시간이 있고 또 아무 일도 하지 않고 단지 주변 노드들의 데이터를 중계만 하는 ' 쉬는 시간(idle time)' 도 있다라고 하는 가정에서 출발한다. 따라서 빈번한 데이터 액세스 작업을 하는 노드는 쉬고 있는 이웃 노드의 캐싱 공간을 낮은 우선순위의 계층적인 캐쉬로 사용하는 것이다.

이웃 캐싱의 기본적인 동작 방식은 다음과 같다. 우선 현재 데이터 액세스를 하고 있는(active time) 어떤 노드는 가져온 데이터를 나중에 다시 사용하기 위해 자기의 캐싱 공간에 저장 한다. 그러면 대치 알고리즘에 의해 자신의 캐싱 공간에서 퇴출된 데이터는 이웃 노드들 중에 하나의 노드에 저장된다. 나중에 그 데이터에 대한 요구가 발생했을 때 그 노드는 비록 자기 자신에게는 데이터가 없지만 이웃 노드에게 이전에 맡겨두었기 때문에 원래의 근원지로부터 데이터를 가져오는 대신 이웃 노드로부터 데이터를 가져 오는 것이다. 참고로 본 논문에서는 대 치 알고리즘을 LRU(Least Recently Used)로 고정하여 서술하겠다.

이웃 캐싱은 완전히 분산되고 지역적인 운영으로 동작하기 때문에 최적의 경우를 찾기 힘들다. 한 노드의 캐싱 공간은 주변 이웃 노드들이 모두 경쟁하면서 사용하고 있어서 최악의 경우 분주한 경쟁으로 인하여 이웃에 저장된 데이터가 적중되기 전에 퇴출 당할 수 있다. 또 이웃 노드로 데이터를 옮기는 과정에서 연쇄적인 데이터의 이동이 발생할 수도 있다. 이렇게 재사용하지 못하고 버려지는 데이터가 많거나 무조건적으로 데이터가 이동한다면 저장된 데이터의 적중률이 낮아지게 되고 이웃 캐싱으로 얻어지는 이득보다는 손해가 많을 것이다.

이웃 노드에 저장되어 있는 데이터의 적중률을 높이기 위해서는 첫째, 가장 데이터를 오래 보존할 수 있는 최적의 이웃 노드

를 찾아야 한다. 둘째, 오래 저장될 수 있을 가능성이 높은 데이터만 이웃 캐싱 해야 된다.

본 논문에서는 이를 해결하기 위해서 우선순위에 근거한 예측기법(raking based prediction)을 제안한다. 이 방법은 최소한의 이웃 노드에 대한 정보만을 활용하여 캐싱 하고자 하는 데이터가 가장 오래 유지될 가능성이 높은 이웃을 선택한다. 그리고 무분별적으로 데이터가 이웃으로 저장되는 것을 막고 재사용될 가능성이 높은 데이터만을 저장하는 근거를 제시하여 데이터의 적중률을 높인다. 따라서 이웃 캐싱에 드는 오버헤드는 줄이고 효율을 극대화 하도록 고안되었다.

2.2. 우선순위에 근거한 예측 기법

분주한 노드에서 쫓겨나는 데이터는 비록 그 노드의 캐쉬에서 가장 낮은 우선순위(ranking)를 가지고 있지만 한가한 이웃 노드로 이동 된다면 거기서 높은 우선순위를 가질 수 있다. 따라서 우선순위에 근거한 예측 기법은 가장 한가한 노드를 예측하여 선별하기 위하여 데이터들이 특정 캐쉬에서 차지하는 우선순위와 노드들의 우선순위 기준(ranking threshold)을 이용한다. 또한 노드들의 우선순위 기준에 대응되는 LRU값을 비교함으로써 상대적으로 이웃 노드들의 분주한 정도를 파악할 수 있다. 우선순위에 근거한 예측 기법을 이용한 이웃 캐싱의 동작방식은 다음과 같다.

첫 번째 단계는 자기의 캐쉬에서 퇴출되는 데이터를 이웃 노드에게 보내기 위해 우선순위에 근거한 예측 기법으로 최적의 이웃 노드를 선택하는 단계이다. 먼저 이웃 노드들 중에서 캐싱하고자 하는 데이터의 LRU값보다 우선순위 기준에 대응되는 LRU 값이 높은 이웃 노드들을 제외한다. 왜냐하면 이러한 노드에는 해당 데이터가 저장될 수 없기 때문이다. 그런 다음 나머지 캐싱 가능한 이웃 노드들 중에서 데이터의 LRU값과 가장 많은 차이를 나타내는 우선순위 기준 LRU 값을 가진 노드를 가장 한가한 최적의 노드로 선별한다. 이 방법을 통해서 모든 이웃 노드들의 우선순위 기준보다 낮은 우선순위를 가진 데이터는 이웃 캐싱 하지 않을 수 있고 현재 가장 한가한 이웃 노드에 데이터를 저장할 수 있기 때문에 이웃 캐싱 된 데이터 의 적중률을 높일 수 있다.

두 번째 단계는 첫 번째 단계에서 선정된 이웃 노드와 협상하는 단계이다. 노드가 이웃 노드들의 우선순위 기준LRU값을 수집한 시간과 실제 예측된 결과를 바탕으로 데이터를 저장할 때의 시간적 차이가 존재할 수 있다. 이는 무시할 수 있으나 협상을 통해서 좀 더 정확한 판단을 내릴 수 있다. 또 협상은 이웃 노드에게 데이터를 보내기 위한 사전작업의 역할도 한다. 캐싱하고자 하는 노드가 보내고자 하는 데이터의 우선순위 값을 포함한 협상용 패킷을 선정된 이웃 노드에게 보내면 이웃 노드는 그 데이터의 LRU값이 자기의 우선순위 기준 LRU값보다 높은지 확인하여 만약 저장 가능하다면 긍정적인 응답을 보낼 것이고 그렇지 않다면 거부를 한다. 만일 협상이 결렬되면 데이터를 보내고자 하는 노드는 다음으로 적당한 이웃 노드와 협상을 하게 된다.

그림 2.1은 우선순위에 근거한 예측기법을 통한 최적의 노드 선별과정과 협상과정을 나타내고 있다. 이 그림에서 노드 A는 시간 T_1에서의 이웃 노드 X, Y, Z의 우선순위 기준 LRU값을 알고 있고 이 값을 근거로 데이터 D_{55}를 이웃 캐싱 하려고 한다. 노드 Z는 우선순위 기준 LRU값이 데이터 D_{55}의 LRU 값보다 높기 때문에 제외되고 캐싱 가능한 노드 X, Y중에서 D_{55}의 LRU값과 가장 많은 차이를 보이는 우선순위 기준 LRU값을 가지고 있는 노드 Y가 최적의 노드로 선택된다.

그리고 시간 T_2에서 노드 Y와 협상을 한다. 그러나 시간적 차이로 인하여 노드 Y에서 우선순위 기준 LRU값이 상승되어 D_{55}를 캐싱할 수 없게 되었다. 따라서 캐싱 요청은 거부된다. 따라서 다시 다음 후보인 노드 X와 협상을 하게 되고 시간 T_2에서의 우선순위 기준 LRU값이 D_{55}의 LRU값보다 여전히 낮기 때문에 캐싱 요청은 수락된다.

협상이 이루어진 후 실제로 데이터를 이웃 노드에 저장하게 되며 저장된 데이터의 소유자는 그 데이터를 맡긴 노드가 되고 데이터에 소유자 정보가 같이 저장된다. 그 후에 사용자가 요청한 데이터가 자기의 캐쉬에 없고 이웃 노드에 있다면 이웃 노드로부터 데이터를 가져온다. 이웃 캐쉬에 저장된 데이터 중 오랫동안 사용되지 않은 데이터는 퇴출되게 되며 이 데이터를 가지고 있던 노드는 데이터의 소유자에게 이 사실을 알려준다.

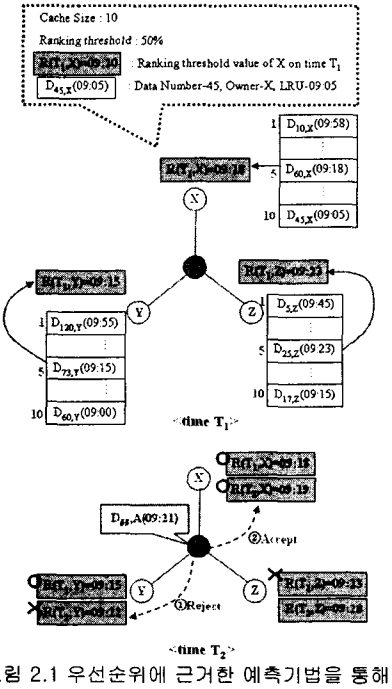

그림 2.1 우선순위에 근거한 예측기법을 통해 최적의 이웃을 선택하는 과정

3. 실험

3.1. 실험 환경

본 연구에서는 시뮬레이션을 통해서 성능 평가를 하였다. 실험에서 사용된 시뮬레이터 환경, 자세한 설정사항은 표 3.1에 나와 있는 것과 같다. 실험에 사용된 어플리케이션은 임의의 노드로 흩어져있는 분산 데이터를 100개 추출하여 특정 패턴을

Simulator	* ns2 - extended to support the simulation of app. * MAC : IEEE 802.11 * 1 hop cell coverage : 250 m * Routing protocol : DSR [4] * Transport layer : 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 * Data access pattern : Zifp-like distribution $1/i^{\alpha}$, $\alpha \approx 1.0$ * Replacement algorithms : LRU (Least Recently Used)
비교	* Self caching (individual caching) & Neighbor caching

표 3.1: 실험에 적용된 세부설정들

가지고 100번 액세스한다. 노드는 데이터 액세스를 하다가 얼마간 쉬고 다시 데이터 액세스를 하는 작업을 반복하고 주어진 데이터를 모두 액세스하면 종료한다.

실험에서 데이터 액세스를 할 때 사용된 패턴은 일반적으로 웹 어플리케이션의 데이터 액세스 패턴으로 널리 알려진 Zipf-lik e distribution[3]을 사용하였다. Zipf-like distribution에 따르면 i번째 인기 있는 데이터는 $1/i^{\alpha}(\alpha \approx 1)$의 비율로 액세스 된다.

3.2. 실험 결과

첫 번째로 노드 개수의 변화에 따른 성능 변화를 살펴보자.

그림 3.1은 노드 개수를 변화시켜 어플리케이션을 수행했을 때 모든 노드들이 어플리케이션을 끝내는데 걸린 평균 수행시간과 그때 망의 전체 트래픽을 측정한 그래프이다. 이 때 사용된 캐쉬의 크기는 5 - 추출된 데이터 집합의 5%를 저장할 수 있음 - 이고 우선순위 기준 값은 0.6이다. 쉬는 시간의 빈도는 0.05의 확률로 나타나고 쉬는 시간의 길이는 일한 시간의 10배를 쉬게 하였다.

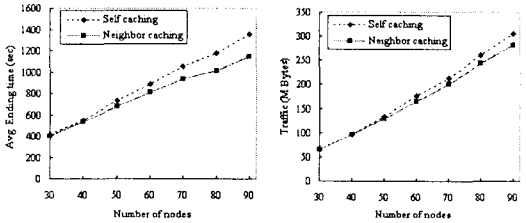

그림 3.1: 노드 개수의 변화에 따른 성능 변화

그림 3.1을 보면 노드의 개수가 증가하면서 이웃 캐싱의 성능 개선이 뚜렷해짐을 알 수 있다. 실험에서는 노드들의 망 형태가 격자구조로 밀도가 일정하기 때문에 노드 수의 증가는 곧 망의 크기가 커지는 것을 의미한다. 따라서 노드 수가 증가하면서 근원지로부터 데이터를 가져오는 평균 홉 수가 커지기 때문에 이웃 캐싱의 효율성이 상대적으로 더욱 커지게 된다. 망 전체의 트래픽을 나타내는 오른쪽 그래프에서는 수행시간에서 보다 성능향상이 좀 낮지만 역시 노드 수 증가에 따라 성능 향상의 폭이 커짐을 알 수 있다.

두 번째로 쉬는 시간과 우선순위 기준의 변화에 따른 결과를 살펴보자. 그림 3.2는 쉬는 시간의 길이 변화에 따른 이웃 캐싱과 자기 캐싱(self caching)의 성능 비교와 함께 이웃 캐싱에서도 우선순위 기준이 성능에 어떠한 영향을 미치는 지 동시에 보여주는 그래프이다. 이 그래프에서 X축은 우선순위 기준인데 이 값은 이웃 캐싱일 때 0.1 ~ 0.9 사이에 있다. 그리고 우선 순위 기준이 0인 지점은 이웃 캐싱을 하지 않는 다는 의미로 자기 캐싱을 나타내고 1인 지점은 이웃 캐싱은 하되 우선순위에 근거한 예측기법을 사용하지 않고 모든 데이터를 이웃 노드에게 차례로 저장 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 캐쉬의 크기는 100이다.

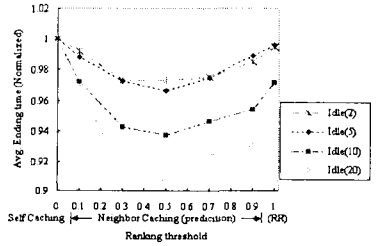

그림 3.2: 쉬는 시간의 길이와 우선순위 기준의 변화

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

다음으로 우선순위 기준의 변화가 성능상에 미치는 영향을 살펴보기 위해서는 하나의 꺾은 선에서 각 점들의 값을 살펴보면 된다. 대체로 이웃 캐싱에서 우선순위 기준이 0.3 ~ 0.5 일 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 이웃에게 가져오는 데이터의 수를 늘리는 것이 바람직하다.

표 3.2에서는 우선순위 기준이 0.5일 때 쉬는 시간에 따른 이웃 캐싱을 위한 데이터의 이동과 이웃 캐쉬의 적중률을 분석하였다. 이 표를 통해 우선순위에 근거한 예측기법을 적용하면 노드들이 분주해질수록 이웃으로 이동되는 데이터 수는 줄어들게 된다는 것을 알 수 있는데 이것이 분주한 노드들 사이의 트래픽 증가에 따른 성능 저하를 막을 수 있게 해준다. 아울러 1홉 적중률은 일정 정도로 유지함으로써 우리는 우선순위에 근거한 예측기법이 망의 분주함에 대해서 어느 정도 탄력적(adaptive)으로 반응하는 방식임을 알 수 있다.

Idle time	0	5	10
Data move	15.06	29.81	34.78
Neighbor hit ratio	0.275	0.305	0.328

표 3.2: 데이터 이동과 이웃 캐쉬의 적중률

4. 결론 및 향후 연구 과제

본 논문에서는 애드혹 네트워크를 기반으로 하여 분산된 데이터를 서로 주고 받는 어플리케이션을 수행할 때 이웃 캐싱 기법과 우선순위에 근거한 예측기법을 통하여 효율성과 확장성을 증대시키는 방법을 제안하였다.

다양한 변수들을 가지고 실험된 결과 이웃 캐싱은 일반적으로 노드의 개수가 늘어나 망의 크기가 커질 때, 노드들의 쉬는 시간이 길 때, 그리고 노드들의 캐쉬 크기가 작을 때 좋은 성능을 나타낸다. 그리고, 우선순위에 근거한 예측기법은 노드들의 바쁜 상황에 따라 이웃으로 캐싱 하는 데이터의 수를 조절함으로써 이웃 캐싱의 적중률을 높이는 매우 적응력 있는 방식임을 알 수 있다.

5. 참고 문헌

[1] 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

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

[3] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "*Web caching and Zipf-like distributions: Evidence and implications*", In Proceedings of INFOCOM, April 1999

[4] 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