Chord 에서 낮은 Lookup 지연을 위한 RTT 정보 기반의 다음 홉 선택 기법

박소영, 이호진, 권태경, 최양희
서울대학교 컴퓨터공학부
{sypark, lumiere, tk, yhchoi}@mmlab.snu.ac.kr

Next-hop Selection in Chord for Low Lookup Latency Based on RTT Information

Soyoung Park, Hojin Lee, Taekyoung Kwon, Yanghee Choi
School of Computer Science and Engineering
Seoul National University, Seoul, Korea

요 약

규모성 있고 강건하며 효율적인 lookup 서비스를 위해서 Chord 가 제안되었다. Chord 는 노드의 수가 N 일 때 $\log_2 N$ 이내의 Chord 노드를 거쳐가면 임의의 키를 책임지는 노드를 찾는 것이 보장된다. 하지만 Chord 는 Chord 노드를 거쳐가는 회수(논리적 홉 수)를 줄이는데 초점을 맞출 뿐 실제 물리적 네트워크를 반영하지 않는다. 그렇기 때문에 실제로 논리적 홉 수가 줄어든다고 해서 지연이 줄어드는 것을 보장하지 않는다. 본 논문에서, 우리는 Chord 의 finger table 에 RTT 를 추가하고 다음 홉 선택 시 RTT 정보를 반영하는 간단한 방법을 제안한다. RTT 는 Chord 의 안정화를 위해 주기적으로 사용하는 메시지를 확장해서 쉽게 구현할 수 있다. 제안하는 기법은 물리적 네트워크의 정보를 반영하여 근처의 노드를 다음 홉으로 선택함으로써 논리적 홉 수는 다소 증가할 수 있지만 lookup 시 지연을 줄일 수 있다. 시뮬레이션 결과 제안하는 기법이 Chord 보다 10% 정도 성능향상이 있다.

I. 서론

Chord[1]는 Distributed Hash Table(DHT)을 구현한 대표적인 lookup 프로토콜로 규모성(scalable)이 있고 강건(robust)하며 동작이 단순하다. 또한, 이를 기반으로 한 다양한 응용이 제안되었다[2][3][4].

Chord 의 각 노드는 M 비트의 식별자를 지니며 이 노드들은 식별자 고리(ring)에서 시계 방향으로 식별자가 커지도록 위치한다. 각 노드는 선임 노드(predecessor)와 자신의 식별자 사이(자신의 식별자는 포함)에 위치하는 키 값에 대해서 책임을 지며, 후임 노드(successor)의 위치를 알고 있다. 그렇게 함으로써 임의의 키 값에 대한 질의를 받았을 때, 그 키 값을 자신이 책임지지 않는 경우 후임 노드들을 따라가서 키 값을 책임지는 노드를 찾을 수 있다. 하지만, 이 경우 노드의 개수가 N 일 때 검색을 위해서 O(N)의 논리적 홉(Chord 노드를 거치는 회수)이 필요하다.

논리적 홉을 줄이기 위해서 Chord 는 finger table 을 이용한다. finger table 은 M 개의 레코드를 지니고 있으며 i 번째 레코드는 [자신의 식별자 + 2^{i-1}, 자신의 식별자 + 2^i) 영역의 키를 담당하며, 자신의 식별자 + 2^{i-1} 이상인 노드들 중 최소 식별자를 지닌 노드를 다음 홉으로 저장한다. 특정 키 lookup 시, finger table 의 마지막부터 첫번째까지 검색하면서 그 키가 포함된 레코드의 노드로 질의를 라우팅한다. 이렇게 함으로써 후임 노드를 따라가는 경우와 달리 한 홉마다 검색하는 키와 현재 노드와의 식별자간 거리를 최소 반 이상 줄일 수 있으며, 그 결과 $\log_2 N$ 이내로 lookup 을 수행한다.

Chord 라우팅시, 키 값이 i 번째 레코드 담당 영역에 속한 경우에 i-1 번째 레코드를 택해도 라우팅이 동작한다는 사실에 주목할 필요가 있다. 즉, Chord 고리 상에서 식별자가 키 값을 시계 방향으로 지나치지 않은 노드는 그 노드의 finger table 을 이용해서 라우팅이 가능하다. i 번째 레코드를 선택하는 것은 키와 다음 노드의 식별자간 거리를 최대한 줄여서 논리적 홉을 줄이기 위한 탐욕적(greedy) 방법이다.

우리는 논리적 홉이 적은 것이 반드시 물리적 홉 또는 지연(latency)을 줄이는 것이 아니라는 사실에 영감을 얻었다. Chord 노드는 오버레이 네트워크(overlay network)상에 존재하기 때문에 Chord 노드 간 거리가 논리적으로 한 홉이라 하더라도 실제 물리적 홉과 지연은 다양할 수가 있다. 따라서 우리는 특정 경우에 i 번째 레코드 노드 대신 i-1 번째 레코드 노드를 선택하게 하여, 논리적 홉은 약간 증가하더라도 실제 성능의 척도인 지연을 줄이는 방법을 제안한다. 우리가 제안한 방법은 단순히 finger table 에 RTT 정보를 추가하면 된다. RTT 정보는 Chord 의 안정화를 위해서 주기적으로 교환하는 메시지를 확장해서 쉽게 구현할 수 있다. 실험 결과 제안하는 기법은 단순한 수정으로 10 % 정도 성능향상을 보인다.

본 논문은 다음과 같이 구성된다. 우선 제안하는 기법을 설명하기에 앞서 2 장에서는 기본이 되는 Chord 에 대해 간략히 살펴본다. 3 장에서는 Chord 의 문제점을 알아본 후, 이 문제점을 해결하기 위하여 본 논문에서 제시하는 기법에 대해서 설명한다. 4 장에서 제안기법과 Chord 를 비교 분석한 실험 결과를 제시한 뒤에, 5 장에서 본 논문을 정리하고 추가 연구를 제안하며

마친다.

II. Chord

Chord 는 DHT 프로토콜 중의 하나로, 가상의 고리를 형성하여 이 고리를 통해 오버레이 네트워크를 구축한다. 각 노드는 M 비트 식별자 공간에서 무작위하게 식별자를 할당받으며, 시계 방향으로 식별자가 커지도록 배치되어 식별자 고리를 형성한다. 키도 노드와 같은 식별자 공간에서 식별자를 할당받으며, 할당된 식별자 이상인 노드들 중 최소 식별자를 지닌 노드가 해당 키를 담당한다. Chord 고리에서의 거리(이를 Chord 거리라고 하겠다.)는 시계방향으로의 거리만을 의미한다. 따라서 Chord 거리가 가장 가까운 노드는 고리에서 시계방향으로 자신과 가장 가까운 노드이다.

각 노드는 후임 노드의 위치를 알고 있다. 임의의 키 값에 대한 질의를 받았을 때, 그 키 값을 자신이 담당하지 않는 경우 후임 노드를 따라간다. 이 과정을 반복함으로써 키 값을 담당하는 노드를 찾아갈 수 있다. 하지만 이런 방식은 노드의 개수가 N 인 상황에서 검색을 위해 최악의 경우 O(N)의 논리적 홉이 필요하다.

논리적 홉을 줄이기 위해서 Chord 의 각 노드는 finger table 을 유지한다. finger table 은 M 개의 노드에 대한 레코드를 가지고 있다. i 번째 레코드는 자신의 식별자 + 2^{i-1} 이상인 노드들 중 최소 식별자를 지닌 노드를 가리키며, 이 노드는 [자신의 식별자 + 2^{i-1}, 자신의 식별자 + 2^{i}) 영역의 키를 담당한다. 임의의 키 값에 대한 질의를 받으면, 키의 식별자가 속하는 영역의 레코드가 가리키는 노드를 다음홉으로 지정한다. 이 노드를 레코드 노드라고 부르겠다.

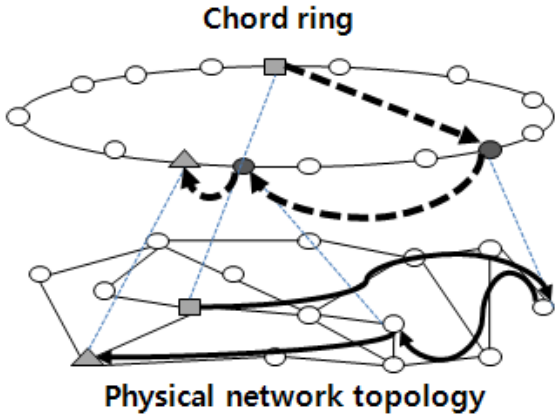
각 노드는 finger table 을 통해 자신의 식별자 + 2^{i-1} ($1 \le i \le M$)와 가장 Chord 거리가 가까운 노드들에 대한 정보를 유지한다. finger table 을 이용하면, 한 번의 lookup 마다 현재 노드와 검색하는 키의 Chord 거리를 최소 반 이상 줄일 수 있어 $\log_2 N$ 이내로 lookup 을 수행한다.

lookup 을 수행하는 노드는 레코드의 담당 범위가 자신과 Chord 거리가 가장 먼 레코드부터 가까운 레코드 순으로 (M 번째 레코드에서 1 번째 레코드 순으로) 자신의 finger table 을 살펴보면서, 키의 식별자가 해당 레코드의 담당 범위에 속하면 그 레코드 노드를 다음 홉으로 선택한다. 이렇게 finger table 내의 레코드 노드 중 목적 노드까지의 Chord 거리가 가장 가까운 노드로 탐욕적으로 전달하는 과정을 반복함으로써 최종적으로 원하는 키를 찾아가게 된다.

III. Chord+

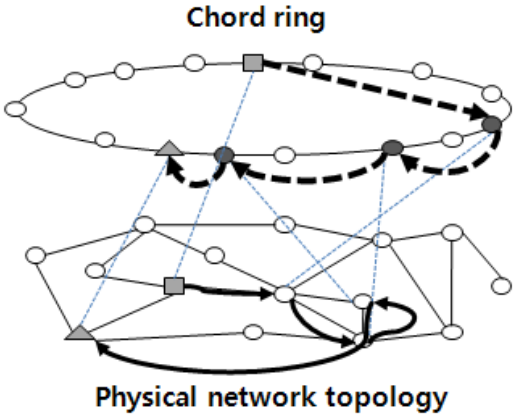
3.1. 기존 Chord 의 문제점

II 에서 언급한 바와 같이 Chord 는 식별자 공간에서 무작위하게 식별자를 할당받고, 이를 이용하여 고리를 구성한다. 이 때 고리 상의 위치는 식별자로 정해지기 때문에 실제 물리적 네트워크는 전혀 반영되지 않는다. 고리 상에서 가까워도 실제 물리적 거리는 멀 수 있고, 반대로 고리 상에서 멀어도 실제 물리적 거리는 가까울 수 있다. 따라서 물리적 네트워크에서 가까이 위치한 노드인데도 불구하고, Chord 고리를 따라 검색하면 물리적 네트워크에서 많이 돌아가는 상황이 발생한다. 즉, 논리적 홉이 적다고 물리적 홉 또는 지연이 적다고 할 수 없다. [그림 1]을 예로 들어 설명하겠다. 위는 Chord 고리이고 아래는 실제 물리적 네트워크 토폴로지다. 네모 노드가 임의의 키 검색에 대한 질의를 시작하고, 세모 노드가 그 키를 담당하는 상황이다. 물리적 네트워크에서 네모 노드와 세모 노드는 1 홉 거리이지만, Chord 거리는 멀리 떨어져있어서 두 개의 검은 동그라미 노드를 거쳐 전달되어야 한다. 이를 물리적 네트워크에 대응시켜보면, 검색에 총 11 홉이 소요된다. 이를 통해 논리적 홉이 적더라도 물리적 홉이 적은 것이 아니라는 사실을 알 수 있다.



[그림 1] 물리적 홉이 짧아도 논리적 홉이 긴 경우

3.2. Chord+ 알고리즘



[그림 2] 논리적 홉이 증가해도 물리적 홉이 짧아진 경우

[그림 2]는 [그림 1]과 동일한 상황에서 논리적 홉이 늘어나더라도 물리적 홉이 줄어드는 경우가 있음을 보여준다. 논리적 홉은 4 홉으로 [그림 1]보다 1 홉 증가하였으나, 물리적 홉은 6 홉으로 5 홉 감소하였다. 본 논문은 Chord 에서 적은 논리적 홉이 적은 물리적 홉 또는 지연을 의미하지 않는다는 사실을 착안하여, 논리적 홉이 다소 증가하더라고 지연을 줄일 수 있는 방법을 제안한다. 제안 방법 Chord+는 물리적 네트워크 정보 중 하나인 지연을 Chord 에 반영하여, 특정 경우에 i 번째 레코드 노드 대신 i-1 번째 레코드 노드를 선택하게 한다.

1) finger table 에 속한 각 레코드 노드까지의 지연 시간 측정.

2) 검색하는 키의 식별자가 담당 범위에 속하는 finger

table 레코드를 찾음(기존 Chord 와 동일).

3) i 번째 레코드에 속할 경우, (i 번째 레코드 노드까지의 지연시간 > *a* × i-1 번째 레코드 노드까지의 지연시간) 이면 i-1 번째 레코드 노드 선택.

[표 1] Chord+ 알고리즘

Chord+ 알고리즘은 [표 1]에 나타난 바와 같이 크게 세 단계로 나누어 진다.

먼저 각 노드는 finger table 에 속해 있는 레코드 노드들까지의 지연 정보를 유지한다. Chord 에서는 참여하는 노드가 변화하여도 lookup 이 바르게 되는 것을 보장하기 위해서 finger table 에 있는 노드들과 주기적으로 메시지를 교환한다(Chord 의 안정화 프로토콜). 메시지를 주고 받을 때 RTT 를 측정하여, 그 값의 반을 지연으로 사용한다. 따라서 지연 정보를 알아내는데 추가적인 패킷 오버헤드가 들지 않는다. 이렇게 할 때 정확한 RTT 는 측정하지 못할 수 있지만, Chord+에서는 충분하다. Chord+는 정확한 지연을 필요로 하지 않고, 두 노드의 RTT 가 현저히 다를 때 그 비교를 위해서 RTT 를 사용하기 때문이다.

lookup 을 수행하는 노드는 자신과 Chord 거리가 가장 먼 레코드부터 가까운 레코드 순으로(M 번째 레코드부터 1 번째 레코드 순으로) finger table 을 살펴본다. 이 부분은 기존 Chord 와 동일하다.

Chord+는 다음 홉을 선택할 때 Chord 와 상이한 방식을 사용한다. 키의 식별자가 i 번째 레코드의 담당 범위에 속했을 경우, Chord 는 i 번째 레코드 노드를 다음 홉으로 선택한다. 하지만 Chord+는 첫 번째 단계에서 구한 지연을 이용하여 다음 홉을 i 번째 레코드로 할지, i-1 번째 레코드로 할지 결정한다. i-1 번째 레코드 노드까지의 지연이 i 번째 레코드 노드까지의 지연보다 일정 비율 작으면 i-1 번째 레코드 노드를 선택한다. 그렇지 않은 경우는 Chord 와 동일하게 i 번째 레코드 노드를 다음 홉으로 결정한다.

다음 홉으로 i-1 번째 레코드 노드를 선택할 경우, 목적 노드까지의 Chord 거리가 기존보다 덜 줄어들기 때문에, 논리적 홉이 Chord 보다 증가할 확률이 높다. 하지만 i 번째 레코드 노드보다 물리적 거리가 일정 비율 가까운 i-1 번째 레코드 노드를 선택함으로써 얻는 지연 이득이 늘어난 논리적 홉으로 인해 증가되는 지연보다 평균적으로 크다면, 실제 물리적 네트워크에서의 지연은 줄어들게 된다. IV 장의 실험을 통해서 이를 보인다.

i-1 번째 레코드 노드까지의 지연이 i 번째 레코드 노드까지의 지연보다 일정 비율 작을 때를 i-1 번째 레코드 노드를 다음 홉으로 선택하는 기준으로 삼는다. 여기서 일정 비율을 정하는 매개 변수는 [표 1]의 a 이다. a 값이 2 라면 i-1 번째 레코드 노드까지의 지연이 i 번째 레코드 노드까지의 지연보다 절반 미만인 경우, i-1 번째 레코드 노드를 선택한다. a 값이 변하면 i-1 번째 레코드 노드를 선택하게 되는 기준이 변하기 때문에 Chord+의 성능도 변한다. a 값에 따른 성능 변화도 IV 장의 실험에서 살펴본다.

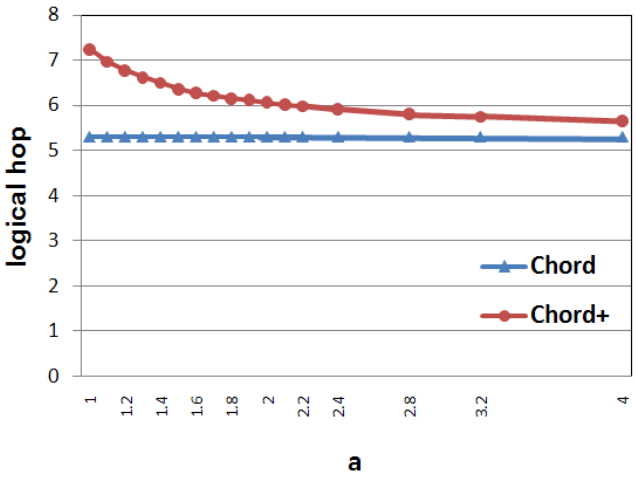
IV. 실험

4.1. 실험 환경

JAVA 를 이용하여 Chord 와 Chord+를 구현하였다. 2000 개의 노드들이 있는 토폴로지를 생성했으며, 두 노드 사이의 지연 시간은 [1ms, 1000ms] 범위에서 무작위한 값을 뽑아 대칭적으로 설정하였다. 식별자는 32bits 길이로 [0, 2^{32}-1] 범위에서 각 노드에게 무작위 할당된다. Chord 고리를 형성한 후, 출발 노드와 목적 노드 쌍을 1000 번 정하여 실험하는 과정을 2 번 실행하였다. a 값에 따른 Chord+ 성능 변화를 보기 위하여 a 값을 1.0 에서 2.2 까지는 0.1 단위로, 그 후에는 2.2, 2.4, 2.8, 3.2, 4.0 로 변화시켜가면서 측정하였다.

4.2. 논리적 홉 (logical hop)

Chord 는 finger table 의 레코드 노드 중 검색하는 키까지의 Chord 거리가 가장 가까운 노드(i 번째 레코드 노드)에게 질의를 라우팅한다. lookup 마다 키를 관리하고 있는 목적 노드까지의 Chord 거리가 줄어들게 된다. Chord+는 경우에 따라 검색하는 키까지의 Chord 거리가 두 번째로 가까운 노드(i-1 번째 레코드 노드)를 다음 홉으로 선택한다. 이 경우 i 번째 레코드 노드를 다음 홉으로 정했을 때보다 목적 노드까지의 Chord 거리가 적게 줄어들게 된다. 따라서 i-1 번째 레코드 노드를 많이 선택 할수록 논리적 홉이 증가할 확률이 높아진다. 이를 알아보기 위해 출발 노드에서 목적 노드까지의 평균 논리적 홉 수를 측정했다.

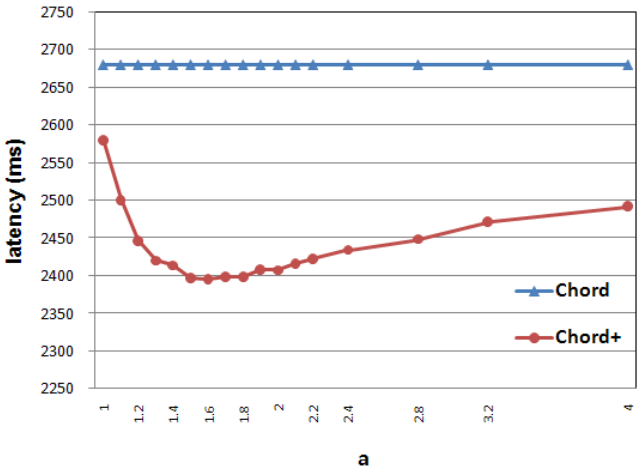


[그림 3] a 값 변화에 따른 논리적 홉 수

[그림 3] 에서 보면 전체적으로 Chord+가 Chord 보다 평균 논리적 홉 수가 큰 것을 볼 수 있다. 또한 a 값과 논리적 홉이 반비례함을 알 수 있다. a 값이 작으면 i-1 번째 레코드 노드를 다음 홉으로 선택하는 기준이 낮아지기 때문에, i-1 번째 레코드 노드로 질의를 라우팅 하는 경우가 많아져 평균 논리적 홉이 증가한다. 반대로 a 가 커지면 i-1 번째 레코드 노드를 선택하는 기준이 높아져 Chord 와 유사하게 동작하기 때문에, 평균 논리적 홉 수가 점점 감소하여 Chord 의 논리적 홉과 비슷해진다.

4.2. 지연 (latency)

Chord+의 성능 향상을 보기 위해 출발 노드에서 목적 노드까지의 평균 지연을 Chord 와 비교하였다. 또한 a 값 변화에 따른 평균 지연의 변화를 살펴보았다.



[그림 4] a 값 변화에 따른 지연

[그림 4]에서 Chord+의 평균 지연이 기존 Chord 의 평균 지연보다 적어, Chord+가 Chord 보다 성능향상 되었음을 알 수 있다.

a 값에 따른 지연 변화도 알 수 있다. 먼저 이론적으로 a 값과 성능과의 상관관계를 생각해보자. a 값이 1 이면, i-1 번째 레코드 노드까지의 지연이 i 번째 레코드 노드까지의 지연보다 작을 경우 무조건 i-1 번째 노드를 다음 홉으로 선택하므로 논리적 홉이 많이 증가한다. 이는 [그림 3]을 통해서 다시 확인할 수 있다. 논리적 홉이 증가하면 논리적 홉 증가에 따른 지연이 늘어나지만, i-1 번째 레코드 노드까지의 지연이 i 번째 레코드 노드까지의 지연시간보다 많이 작지 않기 때문에 i-1 번째 레코드 노드를 선택함으로써 감소하는 지연이 적다. 그러므로 최적 성능보다 더 낮은 성능을 보인다. a 값이 커지면 평균 논리적 홉이 적게 증가하고 i-1 번째 레코드 노드를 선택함으로써 얻는 지연 감소가 커져 성능이 좋아진다. 하지만 a 가 일정 지점(최적 성능을 보이는 a)보다 커지면, i-1 번째 레코드 노드를 선택하는 기준이 엄격해져 Chord 와 같이 동작하는 경우가 많아지기 때문에 성능이 다시 감소한다. 따라서 적당한 a 값 설정이 필요하다.

실험 결과를 보면 이론적 예측과 맞게 a 값이 1 부터 증가함에 따라 지연이 감소하다가 다시 증가하는 것을 볼 수 있다. a 가 1.6 일 때 최적 성능을 보였다. 이 때 Chord+의 평균 지연이 Chord 보다 약 10.6% 적다.

V. 결론

Chord 는 실제 물리적 네트워크 정보를 전혀 반영하지 않은 가상의 고리를 사용하기 때문에, 논리적 홉과 물리적 홉과의 괴리가 존재한다. 따라서 출발 노드부터 목적 노드까지의 실제 물리적 홉보다 많은 물리적 홉을 거쳐 lookup 하는 경우가 발생한다. 이는 검색 지연을 길게 하고, 네트워크 자원의 효율성을 떨어뜨린다. 이 문제를 완화하기 위해 본 논문은 Chord 에 물리적 네트워크 정보인 지연을 반영한 Chord+를 제안하였다. Chord+는 Chord 의 안정화 프로토콜을 이용하여 지연을 알아내기 때문에 추가적인 패킷 오버헤드가 필요 없으며, Chord 와 비교했을 때 지연을 감소시켜 더 좋은 성능을 보였다. a 값에 따라 Chord+의 성능 향상 정도가 변하는데, 최적 성능을 내는 a 값을 찾기 위한 수학적 분석을 앞으로 진행할 예정이다.

감사의 글

본 연구는 지식경제부 및 정보통신연구진흥원의 IT 신성장동력핵심기술개발사업의 일환으로 수행하였음. [2007-F-038-02, 미래 인터넷 핵심기술 연구]

이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다.

참고문헌

[1] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. frans Kaashoek, frank Dabek, Hari Balakrishnan, “Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications,” *IEEE/ACM Transaction on Netoworking*, vol. 11, no. 1, 2003.

[2] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, Sonesh Surana, “Internet Indirection Infrastructure,” *IEEE/ACM Transactions on networking*, vol. 12 no. 2, 2004

[3] Karthik Lakshminarayanan, Ananth Rao, Ion Stoica, Scott Shenker, “End-host Controlled Multicast Routing,” *Elsevier Computer Networks*, vol. 50, no. 6, 2006.

[4] Shelley Zhuang, Kevin Lai, Ion Stoica, Randy Katz, Scott Shenker, “Host Mobility Using an Internet Indirection Infrastructure,” in *Proc. ACM MobiSys ’03*, 2003.