에드 혹 네트워크를 위한 계층적인 메쉬 기반 멀티캐스트 라우팅 프로토콜

(Hierarchical Mesh-based Multicast Routing Protocol for Ad-Hoc Networks)

김 예 경 † 이 미 정 ††
(Yekyung Kim) (Meejeong Lee)

요 약 본 논문에서는 에드 혹 망을 위한 메쉬 기반의 새로운 멀티캐스트 라우팅 프로토콜인 HMMRP (Hierarchical Mesh-based Multicast Routing Protocol)를 제안한다. HMMRP는 송신원 중 일부를 코어 송신원으로 두고 타 송신원들이 코어 송신원 중 하나에 반드시 연결되도록 한다. 그리고 송신원별 트리의 합집합으로 구성되는 메쉬에 의해 송신원과 수신원이 연결되도록 한다. HMMRP는 이들 연결 경로 상의 노드들로써 데이타 전달 메쉬를 형성하고 이를 정기적으로 재구성한다. 특히, 일반 송신원으로부터 코어 송신원에 이르는 경로와 코어 송신원으로부터 수신원에 이르는 트리에 해당하는 메쉬에 대해서는 정기적인 재구성 기간보다 훨씬 짧은 기간마다 국부적으로 메쉬 단절 가능성을 감시하고 복구하도록 함으로써, 임의의 송신원 수신원간에 최소한 송신원-코어-수신원을 경유하는 메쉬를 통해 데이타를 전달할 수 있도록 한다. 이렇게 함으로써 이동성이 높은 에드 혹 네트워크일지라도 정기적인 메쉬 재구성 기간을 짧게 잡지 않고 높은 데이타 전달율을 제공할 수 있다. 시뮬레이션을 통한 성능 분석 결과, HMMRP는 이동성에 대한 성능저하가 상대적으로 적으며, 특히 멀티캐스트 그룹의 규모가 커질수록 이동성에 대한 성능저하가 더욱 경미해짐을 알 수 있었다.

Abstract We propose a mesh based multicast routing protocol referred to as HMMRP for ad-hoc networks. In HMMRP, a limited number of sources are selected as core sources, and the rest of the sources of a multicast group are connected to one of those core sources. The sources and the receivers of a multicast group are also connected through per source trees. In HMMRP, the data delivery mesh of a multicast group are composed of the nodes on these paths, and are reconfigured at regular intervals. Furthermore, each mesh member that lies on the paths between the sources and the core sources as well as between the core sources and the receivers keeps checking if there is a symptom of mesh separation around itself. When a mesh member finds such symptom, it tries to patch itself to the mesh with a local flooding. As a result, the part of the data delivery mesh on those paths are kept connected with a lot higher probability than the rest of the data delivery mesh. That is, for a certain source receiver pair, it is very likely that at least there exists a data delivery path that route from the source to a core source and then to the receiver. Therefore, HMMRP may provide very high data delivery ratio without frequent entire data delivery mesh reconfiguration even when the nodal mobility is high. Simulation results show that HMMRP shows relatively little performance degradation with respect to mobility. Furthermore, the performance degradation with respect to mobility is even smaller when the size of the multicast group becomes larger.

· 본 연구는 한국학술진흥재단의 2000년도 BK21 특화사업 지원에 의하여 수행되었음.
† 학생회원 : 이화여자대학교 컴퓨터학과
yeki@nuri.net
†† 정 회 원 : 이화여자대학교 컴퓨터학과 교수
lmj@ewha.ac.kr
논문접수 : 2001년 2월 19일
심사완료 : 2001년 7월 5일

1. 서 론

에드 혹 망은 고정된 네트워크 기반이 없이 이동하는 호스트들로 구성되는 다중 홉 무선 네트워크이다. 이러한 에드 혹 망의 대표적인 응용으로는 재난 시 이를 해결하기 위한 긴급 대책위원회 멤버들간의 통신이나 전

투시 전투요원들간의 통신, 컨퍼런스나 야외 오락에서 멤버들 간의 통신 등을 들 수 있다. 이러한 응용들은 대부분 다중점 대 다중점의 통신을 요구하기 때문에 이를 효율적으로 지원하기 위해서는 멀티캐스트 라우팅 기술이 필요하다. 그런데, 에드 혹 망은 망의 기반 구조가 예측할 수 없이 동적으로 변해 나가며, 호스트의 프로세싱 능력 및 메모리 등의 지원이 매우 제한적이라는 특성이 있어 이러한 환경에 효율적인 멀티캐스트 라우팅 프로토콜을 설계하는 것은 매우 도전적이다.

최근 에드혹 망의 효율적인 멀티캐스팅을 위해 많은 새로운 프로토콜들이 다양하게 연구되고 있다. IETF에서도 MANET 워킹그룹이 조성되어 이러한 프로토콜에 대한 연구가 이루어지고 있다. 제안된 프로토콜들을 크게 분류하자면, 송신원으로부터 각 수신원에 대해 유일한 최단 경로가 결정되어 이를 통해 데이타를 전달하는 트리 기반(Tree-based) 방식과 하나 이상의 경로를 통하여 데이타를 전달하는 메쉬 기반(mesh-based) 방식으로 나누어 볼 수 있다. 트리 기반 방식의 경우 네트워크에 발생하는 패킷의 복사본 수를 최소화한다는 장점이 있지만 노드의 이동 정도가 높아 네트워크의 링크 단절이 자주 발생하는 환경에서는 트리 링크 단절로 인한 데이타 패킷 손실이 발생할 수 있다. 또한, 트리를 유지하기 위해 발생하는 제어 패킷으로 인해 채널 오버헤드가 매우 커질 수 있다는 문제점이 있다. 한편, 최단 경로이외에 여분의 경로(redundant path)를 제공하는 메쉬 기반 방식은 최단 경로에 링크 단절이 발생하여도 데이타가 전달될 수 있다는 가능성이 있으나 트리 기반 방식에 비하여 발생되는 패킷 복사본이 많아지게 된다. 이들 프로토콜들의 성능을 비교한 최근 논문에 의하면, 전반적으로 메쉬 기반의 프로토콜이 트리 기반의 프로토콜에 비해 데이타 전달율이 높으며, 제어 트래픽의 오버헤드도 낮은 것으로 나타났다[1].

메쉬 기반의 프로토콜로 제안된 것 중 대표적인 것으로는 ODMRP (On-Demand Multicast Routing Protocol)와 [2],[3],[4], CAMP (Core-Assisted Mesh Protocol) [5],[6] 등이 있다. ODMRP와 CAMP는 모두 멀티캐스트그룹의 데이타 전달을 담당하는 메쉬에 속하는 노드들만이 멀티캐스트그룹에 속하는 데이타를 플러딩하는 제한적인 플러딩을 사용한다. 트리 기반 방식의 경우 트리상의 각 노드들을 자신의 상위 노드로부터 온 패킷만을 전달하는데 반해, 메쉬 기반 방식에서는 멀티캐스트그룹의 데이타 전달 메쉬에 속하는 노드들은 어느 노드로부터 온 데이타이든지 상관하지 않고 해당 멀티캐스트그룹의 데이타는 모두 플러딩한다. 멀티캐스트 그룹을 위한 데이타 전달 메쉬의 구성을 보면, ODMRP는 각 송신원으로부터 수신원에 이르는 최단 경로상에 있는 노드들로 이를 형성하고, CAMP의 경우에는 한 개 이상의 코어 노드가 있어 이들간에 완전 메쉬(full mesh)가 형성되고, 멀티캐스트그룹의 각 멤버는 최단 경로를 통해 코어 노드 중 하나에 연결된다. 또한, CAMP에서는 데이타 전송이 진행됨에 따라 송신원과 수신원 사이의 최단 경로 상의 노드들도 데이타 전달 메쉬의 멤버로 선택되어 나간다.

그런데, ODMRP와 CAMP 프로토콜의 데이타 전달 메쉬 형성 및 유지 방식에는 각각 서로 다른 약점이 있다. ODMRP의 경우에는 정기적인 제어 패킷의 플러딩에 의해 데이타 전달 메쉬를 재구성한다. 따라서 노드들의 이동성이 커지는 경우에는 데이타 전달 메쉬의 단절을 줄이기 위해 메쉬 재구성 인터벌을 짧게 잡아야 하기 때문에, 플러딩으로 인한 오버헤드가 매우 커지게 된다. CAMP의 경우에는 알려진 코어 노드를 향해 데이타 전달 메쉬에 연결하므로 메쉬 재구성을 위해 플러딩이 필요하지 않지만, 코어 메쉬를 중심으로 메쉬가 더 두텁게 형성되고 중심에서 멀어질수록 메쉬가 성기게 형성되어 메쉬의 외곽에 위치한 멤버의 경우 데이타 전달율이 떨어지고 중심부에서의 제어 데이타 발생 오버헤드가 크다는 문제가 있다. [7]에서 제시된 성능 비교에 의하면 CAMP의 데이타 전달율이 ODMRP에 비하여 낮음을 볼 수 있다.

본 논문에서는 ODMRP와 CAMP의 장점/단점을 수용/보완한 새로운 메쉬 기반 에드 혹 네트워크 멀티캐스트 라우팅 프로토콜로서 HMMRP(Hierarchical Mesh-based Multicast Routing Protocol)를 제안하였다. HMMRP는 송신원 중 일부를 코어 송신원으로 두고 타 송신원들이 코어 송신원 중 하나에 연결되도록 한다. 그리고 ODMRP와 유사한 송신원별 트리의 합집합으로 송신원과 수신원이 연결되도록 한다. HMMRP는 이들 연결 경로 상의 노드들로서 데이타 전달 메쉬를 형성하고 이를 정기적으로 재구성하는데 특히 데이타 전달 메쉬의 일부 즉, 일반 송신원으로부터 코어 송신원에 이르는 경로와 코어 송신원으로부터 수신원에 이르는 경로에 해당하는 메쉬에 대해서는 정기적인 재구성 기간보다 훨씬 짧은 기간마다 국부적으로 메쉬 단절 가능성을 감시하고 복구하도록 한다. 이렇게 함으로써 임의의 송신원 수신원은 최소한 송신원과 코어 송신원 사이의 경로 그리고 그 코어 송신원으로부터 수신원에 이르는 경로를 통해서는 거의 항상 연결되어 있는 상태이므로 이동성이 높은 경우라도 정기적인 메쉬 재구성 기간을 짧게 잡지 않고

높은 데이타 전달율을 제공할 수 있다. 또한, HMMRP에서는 하나의 서버를 두고 모든 송신원이 이 서버에 등록하도록 함으로써 이 서버가 모든 송신원의 ID를 모아 이들 중 코어 송신원을 선출하고 정기적으로 송신원 목록을 플러딩하도록 한다.

HMMRP는 코어 노드간에 메쉬를 형성하지 않도록 함으로써 CAMP와 같이 코어 주변에 메쉬 밀도가 높아지고 제어 메시지 교환 오버헤드가 커지는 현상을 피할 수 있다. 또한, 송신원 중 일부를 코어로 사용하여 효과적인 추가의(redundant) 경로를 제공할 뿐 아니라 국부적으로 메쉬 단절을 감시/복구함으로써 ODMRP보다 이동성에 더 효율적으로 대처할 수 있다. 그리고, 서버의 도입으로 정기적 메쉬 재구성에 소요되는 플러딩 오버헤드를 줄인다. 시뮬레이션을 통하여 제안하는 HMMRP가 적은 프로토콜 오버헤드로 멀티캐스트그룹의 규모와 노드의 이동 정도에 상관없이 높은 데이타 전송률을 일정하게 유지함을 볼 수 있었다.

본 논문은 다음과 같은 순서로 구성되어 있다. 1장의 서론에 이어서 2장에서는 본 논문에서 제안하는 HMMRP의 데이타구조 및 작동을 자세히 설명한다. 3장에서는 제안하는 HMMRP의 성능 평가를 위한 시뮬레이션 모델을 설명하고 그 결과를 분석한다. 마지막으로 4장에서는 본 논문의 결론과 향후 연구 방안에 대하여 기술한다.

2. HMMRP

HMMRP는 일반 IP 멀티캐스트에서 멀티캐스트그룹 주소를 제공하는 도메인 네임 시스템(Domain Name System : DNS)의 매핑 서비스[8]와 임의의 에드 혹 유니캐스트 라우팅 프로토콜 사용을 가정한다. HMMRP의 구성요소로는 서버 노드를 비롯하여 송신원, 코어 송신원, 수신원, 데이타 전달 메쉬 멤버 노드(이후로 메쉬 멤버라 함) 등이 있다. 서버는 멀티캐스트그룹의 멤버 여부와는 관계가 없고, 에드 혹 네트워크 상에서 모든 노드들에게 알려져 있다고 가정한다. 에드 혹 네트워크를 인터넷에 연결하는 역할을 하는 라우터가 있다면 이러한 라우터가 서버의 역할을 담당하도록 한다. 일반적으로 이러한 라우터는 고정적이며 여러 가지 이유로 인해 에드 혹 네트워크 전체에 알려져 있기 때문이다 [11][12].

어떤 노드가 특정 멀티캐스트그룹에 데이타를 보내기 원하거나 더 이상 그 그룹에 보낼 데이타가 없을 때, 그 노드는 서버에게 이 사실을 보고함으로써 멀티캐스트그룹의 송신원으로 등록/탈퇴하게 된다. 송신원의 등록/탈퇴를 통하여 멀티캐스트그룹에 속하는 송신원들의 정보를 모은 서버는 이를 정기적으로 에드 혹 네트워크 상에 플러딩함으로써 해당 멀티캐스트그룹의 수신원들이 그 멀티캐스트그룹의 송신원을 파악하여 메쉬를 재구성할 수 있도록 해 준다. ODMRP에서는 송신원이 각자 정기적으로 JOIN REQUEST의 플러딩을 통해 자신을 알리고 수신원들이 이렇게 플러딩되는 정보를 모아 멀티캐스트그룹의 송신원 정보를 모두 파악한 것에 비해[2] HMMRP는 서버가 송신원 정보를 모으고 서버만이 이를 정기적으로 플러딩함으로써 에드 혹 네트워크의 규모나 멀티캐스트그룹의 규모가 커지는 경우 송신원 존재를 파악하도록 하기 위한 플러딩 오버헤드를 줄일 수 있다.

서버는 해당 멀티캐스트그룹에 등록한 송신원들 중 일부를 코어 송신원으로 선정한다. 서버는 코어 송신원을 추가/제거하기 위하여 먼저 해당 코어 송신원에게 이를 알리고 승인을 받은 후 변경된 코어 송신원 정보를 네트워크 전체에 알린다. 서버는 멀티캐스트그룹의 송신원 목록과 함께 코어 송신원 목록도 정기적으로 플러딩한다.

HMMRP의 수신원들은 서버의 정기적인 플러딩을 통해 송신원을 파악하면, ODMRP에서 수신원이 송신원으로 JOIN TABLE을 전달하는 것과 유사한 방법에 의해 수신원으로부터 송신원에 이르는 최단 경로 상에 있는 노드들을 메쉬 멤버로 선정한다[2]. 단, ODMRP에서는 송신원의 JOIN REQUEST가 플러딩되는 과정에서 네트워크 노드들이 송신원으로의 경로를 학습하기 때문에 수신원의 JOIN TABLE이 JOIN REQUEST가 전달된 최단 역경로를 통해 송신원으로 전달되는데 반해, HMMRP에서는 하부의 유니캐스트 라우팅을 이용해 수신원에서 송신원에 이르는 최단 경로 상의 노드들이 메쉬 멤버로 선출된다. 이렇게 형성된 데이타 전달 메쉬를 송신원-수신원 메쉬라 부르기로 한다. [그림 1]은 HMMRP의 메쉬 구성을 개념적으로 보여주고 있는데, 송신원-수신원 메쉬는 모두 수신원에 이르는 송신원별 트리의 합집합으로 [그림 1]의 가는 실선 부분이 이에 해당한다.

서버의 정기적인 플러딩을 통해 코어 송신원도 파악한 수신원은 자신이 이 정보를 유지할 뿐 아니라 자신으로부터 코어 송신원에 이르는 코어-수신원 메쉬 상에 있는 메쉬 멤버에게도 이를 알려 이들 메쉬 멤버들이 자신이 코어-수신원 메쉬 상에 있는 메쉬 멤버라는 사실과 자신이 연결하고 있는 코어 송신원이 누구이며 자신을 코어 송신원으로 연결하고 있는 다음 홉 메쉬 멤

버가 누구인지 파악하도록 한다. 앞으로 코어 송신원으로 연결하는 경로상에 있는 다음 홉 메쉬 멤버를 상위 메쉬 멤버라 부르기로 한다. [그림 1]에서 송신원-수신원 메쉬에 속하는 송신원별 트리들 중 두 개의 코어 송신원을 루트로 하는 트리들에 속하는 메쉬가 코어-수신원 메쉬이다. 서버의 정기적인 플러딩으로 코어 송신원을 파악한 일반 송신원은 코어 송신원들 중 자신과 가장 가까운 코어 송신원에게 최단 경로를 통해 Reserve Request를 보낸다. 코어 송신원이 이에 대한 응답으로 Reserve Ack을 보내는 과정을 통해 일반 송신원과 코어 송신원 사이의 송신원-코어 메쉬에 속하는 멤버가 선출된다. [그림 1]에서 점선으로 표시된 송신원으로부터 코어 송신원에 이르는 경로 상에 있는 메쉬가 이에 해당한다.

HMMRP에서는 송신원-수신원 메쉬에 대해서는 ODMRP나 CAMP와 마찬가지로 정기적으로 메쉬를 재구성하는 반면 송신원-코어 메쉬와 코어-수신원 메쉬에 대해서는 정기적인 인터벌마다 메쉬를 재구성할 뿐 아니라, 항상 그 연결이 유지되도록 정기적인 재구성 시점 사이에 계속해서(즉, 정기적인 메쉬 전체 재구성 인터벌보다 훨씬 짧은 인터벌마다) 국부적으로 메쉬 단절 가능성을 조사하고 국부적으로 이를 연결하는 작업을 해나간다. 즉, [그림 1]에서 굵은 선으로 표시된 부분은 다른 메쉬 부분에 비하여 훨씬 높은 확률로 메쉬 연결이 유지되고 있는 것이다. 이렇게 함으로써 HMMRP의 모든 송신원과 수신원은 적어도 코어 송신원을 경유하는 우회 경로를 통해서는 항상 연결되어 있기 때문에 어떤 송신원으로부터의 최단 경로 상의 메쉬가 단절되어도 그 송신원에서 코어 송신원을 경유하여 전달되는 경로를 통해 데이타를 전달받게 된다.

[그림 1]의 굵은 실선 부분에 해당하는 메쉬의 국부적인 감시 및 재구성은 [7]에서 제안된 방식을 개선하여 사용한다. [7]의 PatchODMRP은 ODMRP를 확장하여 국부적인 메쉬 재연결을 시도하는 프로토콜이다. PatchODMRP는 비콘(Beacon)을 이용해 이웃 메쉬 멤버와의 연결 상실을 탐지하고 만약 사라진 이웃 메쉬 멤버가 어떤 송신원으로 연결되는 경로 상의 상위 메쉬 멤버라면 국부적인 플러딩을 이용해 자신보다 그 송신원에 더 가까운 곳에 위치한 메쉬 멤버를 발견하여 연결을 시도한다. HMMRP나 ODMRP는 모두 메쉬에 기반한 방식이므로 상위 메쉬 멤버 중 하나가 사라진 것이 항상 전달 경로의 단절을 의미하는 것은 아니다. 그러나, 메쉬가 두텁게 형성되지 않은 경우에는 [그림 2]에서와 같이 M_3가 오른쪽으로 이동해 감으로써 M_2는

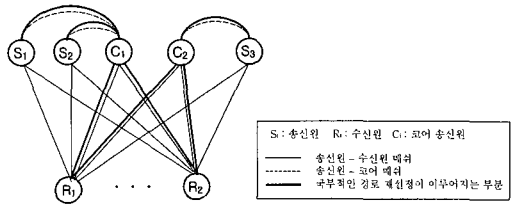

그림 1 HMMRP의 개념적인 메쉬 구성도

송신원 S_2로의 경로상에 있는 다음 홉 메쉬 멤버가 없어지게 되고 이로 인해 메쉬가 단절되어 S_2부터의 데이타를 받지 못하게 될 수 있다. 그런데, 실제로 이러한 경우 M_2가 메쉬를 떠나 멀리 이동한 것이 아니라면 국부적인 플러딩으로 자신에게 가까이 있으면서 S_2에 이르는 경로 상에 있는 다른 메쉬 멤버를 발견할 수 있다. 이를 위해 PatchODMRP에서는 각 메쉬 멤버가 자신이 연결하고 있는 송신원 각각에 대해 어느 메쉬 멤버가 그 송신원으로의 다음 홉에 해당하는 지를 파악하고 있도록 한다. 그러나 이러한 정보는 멀티캐스트그룹의 크기가 큰 경우 특히 송신원의 수가 많다면 매우 양이 많아질 수 있어 각 메쉬 멤버 노드가 이러한 멀티캐스트 그룹의 크기에 의해 그 양이 결정되는 정보를 유지하는 것은 프로토콜 확장성에 문제가 된다. HMMRP는 제한된 수의 코어 송신원에 대해서만 이러한 정보를 유지하기 때문에 멀티캐스트그룹의 크기 및 송신원 수에 대하여 확장성이 있다. 또한 HMMRP는 PatchODMRP에서 국부적인 메쉬 재구성시 재구성하는 노드 주변에 중복적으로 메쉬 멤버가 많이 선출되는 현상을 피할 수 있도록 프로토콜을 개선하였다.

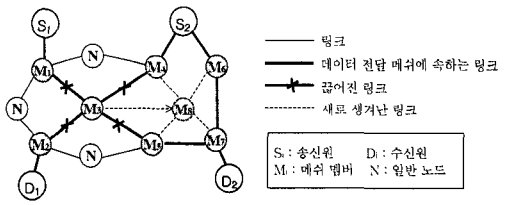

그림 2 메쉬 멤버의 이동으로 인한 메쉬 단절 예

먼저, 2.1절에서는 HMMRP의 각 엔터티가 유지해야 하는 정보 및 타이머들과 제어 패킷의 종류들을 소개하고 2.2절은 멀티캐스트그룹 가입/탈퇴와 정기적인 데이타 전달 메쉬 재구성 과정을, 2.3절은 국부적으로 메쉬 단절 가능성을 감지하고 국부적인 플러딩으로 이를 재구성하는 프로토콜을 설명한다.

2.1 HMMRP 데이타구조와 제어 패킷 및 타이머

HMMRP의 엔터티들은 [그림 3]과 같은 정보를 유지한다. 이중 라우팅 테이블과 메시지 캐쉬 테이블은 모든 메쉬 멤버와 송·수신원이 유지해야 하는 정보이다. 라우팅 테이블은 유니캐스트 라우팅 프로토콜에 의해 제공되며, 노드에서 각 목적지까지의 다음 홉 주소와 목적지까지의 거리가 명시되어 있다고 가정한다. 메시지 캐쉬 테이블은 [그림 3]의 (a)와 같이 최근에 받은 패킷의 송신원 주소, 목적지 주소, 패킷 종류와 패킷 순번을 일정시간동안 저장한다. 이것의 주된 역할은 패킷을 받을 때마다 중복된 패킷인지 체크를 함으로써, 플러딩으로 인해 유발되는 중복 패킷의 양을 줄이고 패킷 루핑을 방지하기 위한 것이다.

서버에 의해 코어 송신원으로 선출된 노드는 [그림 3]의 (b)와 같이 어느 멀티캐스트 그룹의 코어 송신원으로 작동하는지를 표시하기 위해 코어 테이블을 둔다. 코어 테이블은 멀티캐스트그룹 ID에 대하여 코어 송신원인지 여부를 표시하는 플래그를 둔다. 코어 송신원으로 선출된 노드는 서버가 코어 송신원에서 제외되었다고 알리거나 멀티캐스트 그룹에서 탈퇴하게 될 때까지 코어 송신원으로 작동하게 된다.

메쉬 멤버들은 [그림 3]의 (c)와 같이 어느 멀티캐스트그룹의 메쉬 멤버로 언제까지 작동해야 하는지를 표시하기 위해 메쉬 멤버 테이블을 둔다. 메쉬 멤버 테이블에는 멀티캐스트그룹 ID에 대하여 그 멀티캐스트그룹의 메쉬 멤버인지 여부를 표시하는 플래그와 재활성되지 않는 경우 메쉬 멤버 역할을 종료하게 될 시간을 표시하는 타이머를 둔다.

송신원-코어 메쉬와 코어-수신원 메쉬 상의 메쉬 멤버 및 송·수신원은 코어 송신원으로의 경로 상에 있는 메쉬 단절 가능성을 국부적으로 인지하고 메쉬를 재구성하기 위하여 [그림 3]의 (d)와 같이 경로 테이블을 유지한다. 경로 테이블에는 자신이 연결되어 있는 코어 송신원의 주소 및 그 코어 송신원에 대한 상위 메쉬 멤버 주소를 유지한다. 송신원-코어 혹은 코어-수신원 메쉬상의 노드는 비콘에 의해 이웃 노드 중 하나가 없어졌다는 것을 알게되면 경로 테이블을 점검하여 그 없어진 이웃 노드가 자신의 상위 메쉬 멤버중 하나인지 확인한다. 그렇다면 그 없어진 상위 메쉬 멤버를 통해 연결되던 코어 송신원을 향해 국부적으로 새로운 연결을 찾아본다. 어떤 메쉬 멤버가 수신원(들)으로부터 n 개의 코어 송신원에 이르는 경로 상에 있다면 이 메쉬 멤버는 해당 멀티캐스트 그룹에 대하여 n 개의 코어 송신원에 대한 상위 메쉬 멤버 정보를 경로 테이블에 유지하고 있어야 한다. 노드는 어떤 멀티캐스트그룹의 송신원-코어 혹은 코어-수신원 메쉬 상의 메쉬 멤버로 선출되면 경로 테이블의 해당 멀티캐스트그룹 ID 엔트리에 해당 코어 송신원에 관한 상위 메쉬 멤버 정보를 추가하고 재활성되지 않아 타이머가 만료될 때까지 국부적인 메쉬 감시 및 재구성을 담당한다.

송신원-코어 메쉬를 구성할 때 실제 멀티캐스트 데이타가 전달되는 방향으로 메쉬 멤버를 선출하기 위하여 [그림 3]의 (e)와 같은 펜딩 테이블을 사용한다. 일반 송신원은 코어 송신원에 이르는 경로를 설정하기 위해 Reserve Request 제어 패킷을 만들어 대상이 되는 코어 송신원으로 전송한다. 이 제어 패킷을 받은 각 노드는 이들 패킷을 전해준 전 홉 노드에 대한 정보를 [그림 3]의 (e)와 같이 일정시간 동안 펜딩 테이블에 기록해 둔다. 펜딩 테이블의 Reserve Request 패킷 ID 필드는 Reserve Request에 대한 응답인 Reserve Ack 패킷을 받을 경우, 해당하는 Reserve Request 패킷을 유일하게 식별할 수 있도록 Reserve Request에 표시된 멀티캐스트그룹의 ID(MG ID), Reserve Request를 보낸 송신원의 주소(SrcAddr) 및 Reserve Request의 순번(SrcSeq)을 기록한다. 노드가 일정 시간 안에 Reserve Request의 목적지였던 코어 송신원으로부터 Reserve Ack을 받으면, 노드는 이 Reserve Ack에 대응하는 Reserve Request를 자신에게 전해 주었던 '전 홉'을 Reserve Request 패킷 ID를 통해 펜딩 테이블에서 찾아 그 노드에게 Reserve Ack을 전달하고, 메쉬 멤버가 된다. 즉 Reserve Request가 전달된 역경로를 통해 Reserve Ack을 전달하면서 메쉬 멤버가 선출되는 것이다. 한편, 펜딩 테이블에서 타이머가 만기된 엔트리는 삭제한다.

서버와 수신원은 [그림 3]의 (f)와 같이 진행 중인 각 멀티캐스트그룹에 대하여 등록된 송신원들의 리스트를 송신원 목록 테이블에 유지한다. 송신원은 멀티캐스트그룹에 참여/탈퇴시마다 그 사실을 서버에게 보고하고 이를 통해 서버는 송신원 목록 테이블을 유지해 나간다. 서버는 정기적으로 송신원 목록을 플러딩하여 해당 멀티캐스트그룹의 수신원들이 송신원 목록을 파악하도록 한다.

[그림 3]의 (g)는 서버와 송·수신원에서 유지하는 코어 송신원 목록 테이블이다. 서버는 송신원 목록 테이블에서 해당 멀티캐스트그룹에 속하는 송신원들 중 적절한 수의 송신원을 어떤 기준에 따라 혹은 임의로 코어 송신원으로 결정하고 코어 송신원 목록을 구성한다. 가령 서버가 송신원 노드들의 이동성 정도를 알 수 있다면 이를 반영하여 이동 정도가 적은 송신원을 코어 송신원으로

선택할 수 있다. 코어 송신원의 수도 멀티캐스트그룹의 크기 및 송신원 수에 따라 적절하게 결정하는 것이 필요하다. 이와 같이 서버는 송신원 목록 테이블의 송신원 목록으로부터 중앙 집중적으로 각 멀티캐스트그룹의 코어 송신원을 선출하여 송신원 목록과 함께 이 정보를 정기적으로 플러딩함으로써 송·수신원들이 코어 송신원을 파악하도록 한다. 서버로부터 코어 송신원 목록을 받은 송신원은 정기적으로 코어 송신원 목록의 여러 코어 송신원 중에서 자신과 가장 가까운 코어 송신원으로 연결을 시도하고 계속적으로 이 연결이 유지되는지를 국부적으로 감시/재구성한다. 또한, 수신원도 코어 송신원이 어떤 노드들인지 파악하여 이들 코어 송신원으로의 연결이 항상 유지되도록 국부적으로 감시/재구성한다.

송신원 혹은 수신원으로부터 코어 송신원으로의 메쉬 상에 메쉬 단절 우려가 있는 경우에는 ADVT 패킷과 PATCH 패킷을 사용하여 국부적으로 메쉬를 재구성하는데 각 노드는 ADVT나 PATCH 패킷을 받으면 이들 패킷이 전달된 경로를 [그림 3]의 (h)와 같은 패치 테이블에 저장한다. 패치 테이블의 목적지 주소 필드에는 ADVT나 PATCH 패킷을 발생시킨 근원지 노드의 주소를 표시하며 다음 홉 필드에는 ADVT 혹은 PATCH 패킷을 자신에게 전해준 노드의 주소를 표시한다. 패치 테이블에 저장된 정보는 ADVT와 PATCH 패킷에 대한 응답으로 발생하는 PATCH와 PATCH Ack 패킷을 전달할 때 다음 홉을 결정하기 위한 라우팅 정보로서 사용된다.

국부적인 메쉬 재구성을 위해 ADVT 패킷을 발생시킨 메쉬 멤버는 ADVT에 대한 응답으로 가까이에 있는 메쉬 멤버들로부터 PATCH 패킷을 받게 되는데, 받은 PATCH 패킷 중 자신이 연결되기 원하는 코어 송신원에 이르는 가장 가까운 경로 정보를 제공한 PATCH 패킷의 정보를 패치 캐쉬에 저장한다. 패치 캐쉬에는 [그림 3]의 (i)와 같이 멀티캐스트그룹 ID에 대하여 해당 멀티캐스트그룹의 코어 송신원 주소, 코어 송신원까지의 거리를 나타내는 홉 카운트, PATCH 패킷을 발생시킨 메쉬 멤버의 주소, PATCH 패킷을 발생시킨 노드로 향한 다음 홉 주소를 저장한다.

HMMRP는 데이타 전달 메쉬를 관리하기 위해 여러 제어 패킷들을 사용한다. 송신원의 서버 등록을 위한 제어 패킷들로는 Join Request/Join Ack/Join Notify/Join Notify Ack/Leave Request/Leave Ack 등이 있으며, 송신원-코어 메쉬를 정기적으로 재구성하기 위한 제어 패킷들은 Reserve Request/Reserve Ack 이 있다. 또한, 송신원-수신원 메쉬를 정기적으로 재구성하기 위한 제어 패킷들은 Mesh Refresh/Mesh Reply가 있다. 서버가 새로운 코어 송신원을 선출했거나 기존의 코어 송신원을 제외시키는 경우, 서버가 해당 코어 송신원에게 이를 알리고 동의를 얻는데 사용하는 제어 패킷으로 Core Notify/Core Notify Ack이 있다. 그리고 국부적인 메쉬 재구성에 사용되는 제어 패킷으로 ADVT/PATCH/PATCH Ack 패킷이 있다. 이 패킷들의 처리에 관해서는 2.2절에 자세히 설명되어 있으며, 다음의 [표 1]과 [표 2]는 각각 패킷 필드와 패킷 종류에 대한 설명이다. 그리고 [그림 4]는 HMMRP에서 어떤 엔터티 간에 어떤 제어 패킷이 교환되는지를 요약하여 보여주고 있다.

HMMRP는 메쉬 멤버의 탈퇴를 소프트 상태 (soft-state)에 의해 처리한다. 메쉬 멤버는 MM-LIFETIME 동안 MRP나 RA를 받지 못하면 메쉬 멤버 테이블의 타이머가 만기되어 자동으로 메쉬 멤버의 역할을 중단한다. 또한, 송신원-코어나 코어-수신원 메쉬 상의 메쉬 멤버들은 MM-LIFETIME 동안 특정 코어 송신원으로의 메쉬 상에 있다고 알리는 MRP나 RA를 받지 못하면 경로 테이블에서 해당 코어 송신원에 대한 정보를 삭제한다. MM-LIFETIME은 송신원-수신원 메쉬 및 송신원-코어 메쉬를 재구성하는 주기인 Mesh-Refresh-Interval을 고려하여 이들 값의 몇 배가 되도록 해야 한다. 서버와 송신원은 Mesh-Refresh-Interval 마다 한 번씩 각각 MRF와 RR 패킷을 발생시켜야 하고 수신원과 코어 송신원은 이에 대한 응답으로 역시 정기적으로 각각 MRP와 RA 패킷을 발생시킨다. 이와 같이 정기적으로 발생되는 MRP와 RA가 전달되는 과정에서 송신원-수신원 메쉬와 송신원-코어 메쉬가 각각 정기적으로 재구성된다.

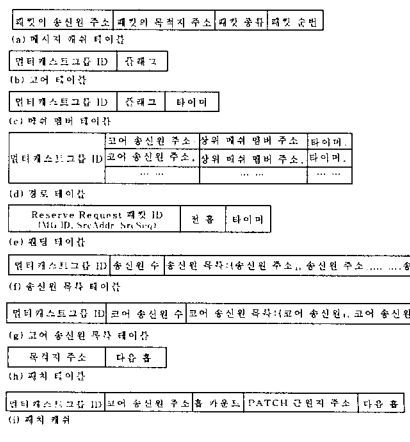

그림 3 HMMRP 데이타 구조

표 1 HMMRP의 제어 패킷 필드

필드명	설 명
PacketType	패킷 구분
MG ID	멀티캐스트그룹 ID
SrcAddr	패킷을 발생시킨 노드 주소
NewSrcAddr	해당 멀티캐스트그룹에 새롭게 참여하고자 하는 송신원 주소
DestAddr	패킷의 목적지 노드 주소
CoreAddr	코어 송신원 주소
NextHop	목적지(DestAddr)로 향한 다음 홉 주소
PreHop	패킷을 건네주는 노드 자신의 주소
SrcSeq/AckSeq	요구 패킷의 순번/요구 패킷에 대한 승인 패킷의 순번. 서로 대응하는 질의/요청 패킷과 응답/승인 패킷을
HopCount	플러딩되는 제어 패킷의 경우 경유할 수 있는 최대 홉수
CoreFlag	코어 송신원임을 표시하는 플래그
CoreHopCount	코어까지의 거리(홉 수)
SrcList	해당 멀티캐스트그룹의 모든 송신원 리스트, {송신원1, 송신원2, 송신원3, ..., 송신원n}
CoreList	해당 멀티캐스트그룹의 모든 코어 송신원 리스트, {코어 송신원1, 코어 송신원2, ..., 코어 송신원n}

표 2 HMMRP의 제어 패킷

패킷 종류 (줄임표기)	설 명
Join Request (JR)	새롭게 멀티캐스트그룹에 참여하려는 송신원이 서버에게 보내는 참여 요청
	{PacketType, MG ID, SrcAddr,DestAddr, NextHop, SrcSeq}
Join Ack (JA)	JR에 대한 응답으로 서버나 이미 등록된 송신원이 새로운 송신원에게 코어 송신원 목록을 실어 보냄
	{PacketType, MG ID, SrcAddr, DestAddr, NextHop, AckSeq, CoreList}
Join Notify (JN)	이미 등록된 송신원이 JR을 받은 경우 JR을 중단시키고 대신 JN을 만들어 새 송신원 참여를 서버에게 알림
	{PacketType, MG ID, SrcAddr, DestAddr, NextHop, SrcSeq, NewSrcAddr}
Join Notify Ack (JNA)	JN에 대한 응답으로 서버가 JN을 보낸 송신원에게 보냄.
	{PacketType, MG ID, SrcAddr, DestAddr, NextHop, AckSeq, NewSrcAddr}
Leave Request (LR)	멀티캐스트그룹 탈퇴하려는 송신원이 서버에게 보내는 탈퇴 요청
	{PacketType, MG ID, SrcAddr, DestAddr, NextHop, SrcSeq}
Leave Ack (LA)	LA에 대한 응답으로 서버가 LR을 보낸 송신원에게 보냄.
	{PacketType, MG ID, SrcAddr, DestAddr, NextHop, AckSeq}
Reserve Request (RR)	송신원이 자신과 가장 가까운 코어 송신원에게 정기적으로 보내는 연결 요청
	{PacketType, MG ID, SrcAddr, DestAddr, NextHop, SrcSeq, PreHop}
Reserve Ack (RA)	RR에 대한 응답으로, RR을 받은 코어 송신원이 RR을 보낸 송신원에게 보냄. RA가 전달되는 과정에서 송신원-코어 메쉬를 구성하는 메쉬 멤버가 선출됨.
	{PacketType, MG ID, SrcAddr, DestAddr, NextHop, AckSeq, PreHop}
Mesh Refresh (MRF)	서버가 정기적으로 송신원 및 코어 송신원 목록 알리기 위해 플러딩하는 패킷
	{PacketType, MG ID, SrcAddr, HopCount, SrcList, CoreList}
Mesh Reply (MRP)	송신원-수신원 메쉬를 재구성하기 위해 정기적으로 수신원이 송신원들을 향해 보내는 패킷. MRP가 전달되는 과정에서 송신원-수신원 메쉬를 구성하는 메쉬 멤버가 선출됨
	{PacketType, MG ID, SrcAddr, (DestAddr, CoreFlag, NextHop) 리스트}
Core Notify (CN)	서버가 코어 송신원을 추가/제거 해야하는 경우 해당 코어 송신원에게 이를 알림
	{PacketType, MG ID, SrcAddr, DestAddr, NextHop, SrcSeq}
Core Notify Ack (CA)	새로이 선출된 코어 송신원이 CN에 대한 응답으로 서버에게 보냄
	{PacketType, MG ID, SrcAddr, DestAddr, NextHop, AckSeq}
ADVT	코어 송신원으로의 연결에 대하여 상위 메쉬 멤버를 손실한 메쉬 멤버가 주변에 도움을 청하기 위해 국부적으로 이 사실을 알리는데 사용하는 패킷
	{PacketType, (MG ID, CoreAddr, CoreHopCount) 리스트, SrcAddr, SrcSeq, HopCount}
PATCH	코어 송신원으로의 경로상에 있는 메쉬 멤버로서 ADVT의 근원지 노드보다 해당 코어 송신원에 더 가까운 메쉬 멤버가 ADVT에 대한 응답으로 보냄
	{PacketType, (MG ID, CoreAddr, CoreHopCount) 리스트, SrcAddr, DestAddr, NextHop, AckSeq}
PATCH Ack	ADVT의 근원지 노드가 PATCH 메시지의 근원지 노드 중 원하는 코어 송신원으로의 가장 가까운 연결을 제공하는 노드에게 PATCH 이 대한 응답으로 보냄. PATCH Ack이 전달되는 과정에서 국부적으로 코어 송신원으로의 연결을 재구성하는 메쉬 멤버들이 새로 선출됨
	{PacketType, {MG ID, CoreAddr}, SrcAddr, DestAddr, NextHop}

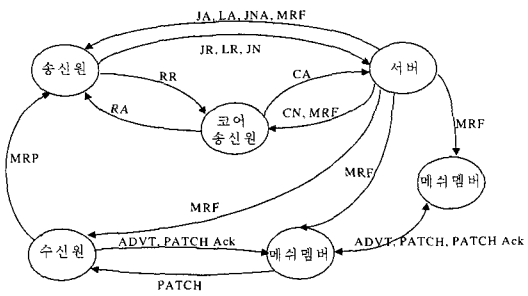

그림 4 HMMRP 구성 엔터티와 엔터티 간에 주고받는 제어 패킷

2.2 데이타 전달 메쉬 형성 및 정기적인 재구성

이 절에서는 HMMRP의 송·수신원이 멀티캐스트그룹에 참여하는 과정, 서버가 코어 송신원을 선출하고 관리하는 방법, 정기적으로 데이타 전달 메쉬가 재구성되는 과정 등에 대하여 차례로 설명한다.

2.2.1 멀티캐스트그룹 가입/탈퇴

▶ 송신원의 멀티캐스트그룹 참여/탈퇴

새롭게 멀티캐스트그룹에 참여하려는 송신원은 먼저 서버의 멀티캐스트그룹 송신원 목록에 등록해야 한다. 서버는 모든 멀티캐스트그룹의 송신원 목록을 유지하여 코어 송신원을 선출하고 정기적으로 코어 송신원과 송신원 목록을 애드 혹 네트워크 전체에 알리는 역할을 하기 때문이다. 멀티캐스트그룹의 송신원으로 참여(탈퇴)하려는 노드는 JR(LR) 패킷을 서버에게 전송한다.

JR(LR)을 받은 노드는 자신의 주소와 패킷의 NextHop 필드 주소를 비교한다. 주소가 같다면, 자신의 유니캐스트 라우팅 테이블에서 서버로 가는 다음 홉을 찾아 받은 패킷의 NextHop 값을 수정하고 이를 브로드캐스트한다. 만약 JR(LR)을 받은 노드의 주소가 JR(LR)의 NextHop 필드에 기록된 주소와 다르면 노드는 그 패킷을 버린다. 이와 같은 방법으로 JR/LR 패킷은 서버를 향해 전달되어 나간다. JR(LR) 이외에도 목적지를 두고 전송되는 패킷에는 항상 목적지로의 NextHop 필드를 두고 위와 같은 방법에 의하여 전송한다.

드디어 서버가 JR(LR)을 받으면, 서버는 송신원 목록 테이블에서 JR(LR)의 MG ID에 표시된 멀티캐스트그룹에 해당하는 엔트리를 찾아 그 엔트리의 송신원 목록에 JR(LR) 패킷의 SrcAddr에 기록된 주소(JR(LR)을 발생시킨 송신원)를 추가(제거)한다. 그리고, JR(LR)에 대한 응답으로 JR(LR)을 보낸 송신원에게 JA(LA)를 전송한다. SrcSeq와 AckSeq는 어느 JR(LR)에 대한 JA(LA) 인지를 확인하는데 사용된다. JR(LR)과 JA(LA) 이외에도 질의 혹은 요청과 그에 대응하는 응답 혹은 승인이 이루어지는 경우에는 항상 이와 같이 SrcSeq와 AckSeq를 사용하여 어느 질의/요청에 대한 응답/승인인지를 확인할 수 있도록 한다.

JR의 경우, 서버는 참여를 요청한 송신원에게 이를 승인할 뿐만 아니라, 해당 멀티캐스트그룹의 코어 송신원들을 가르쳐주어야 한다. 이를 위해 서버는 코어 송신원 목록 테이블에서 JR의 MG ID에 표시된 멀티캐스트그룹에 해당하는 엔트리를 찾아, 그 엔트리의 코어 송신원 목록을 JA 패킷의 CoreList 필드에 명시한다. 송신원은 자신이 발생한 JR(LR)에 대하여 일정 시간 안에 JA(LA)를 받지 못하면 JA(LA)를 받을 때까지 혹은 최대 재전송 횟수만큼 JR(LR)을 재전송해야 한다. JR(LR) 이외에도 질의/요청에 대하여 응답/승인이 이루어지는 경우에는 항상 이와 같이 발생시킨 질의/요청에 대하여 응답/승인이 이루어질 때까지 최대 재전송 횟수가 허용하는 한도 내에서 재시도 한다. [그림 5]의 (a)는 이와 같은 송신원의 가입/탈퇴 요청 및 서버의 승인 과정을 요약하여 도식화한 것이다.

만일, [그림 5]의 (b)처럼, 멀티캐스트그룹에 참여하려는 송신원(송신원 B)의 JR 패킷이 서버에게로 전달되는 중 해당 멀티캐스트그룹에 이미 등록된 송신원(송신원 A)을 거치게 되면, 송신원의 가입이 신속하게 이루어지도록 하기 위해 서버대신 송신원 A가 JR을 보낸 송신원 B에게 JA패킷을 보내준다. 이때 JA의 SrcAddr에는 송신원A의 주소를 적고, CoreList에는 송신원A가 가지고 있는 코어 송신원 목록을 싣게 된다. 송신원A는 JR을 보낸 송신원B에게 JA를 보냄과 동시에, 서버에게 새로운 송신원의 참여를 알리기 위해 JN 패킷을 전송한다. 이 때 JN의 NewSrcAddr 필드에는 새롭게 멀티캐스트그룹에 참여하기 위해 JR을 보낸 송신원 B의 주소를 기입한다. JN을 받은 서버는 JR을 받았을 때와 마찬가지로 자신의 송신원 목록 테이블을 갱신하고, 송신원 A에게로 송신원 B의 참여를 승인하는 JNA를 보낸다.

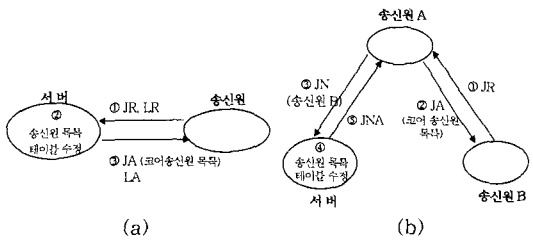

그림 5 송신원의 멀티캐스트그룹 참여/탈퇴

JR, JN, LR 등을 통해 서버의 송신원 목록에 변화가 생기면 서버는 MRF를 발생시켜 이를 네트워크 전체에 알린다. MRF의 처리 및 MRF로 인해 시작되는 데이타 전달 메쉬 재구성에 관해서는 2.2.2절에서 자세히 설명하기로 한다.

▶ 수신원의 멀티캐스트그룹 참여/탈퇴

HMMRP에서 수신원은 송신원과 달리, 자신의 존재를 서버에게 알리지 않는다. 특정 멀티캐스트그룹으로부터 데이타를 받고 싶은 노드는 [그림 6]에서와 같이 서버가 각 멀티캐스트그룹에 대하여 정기적으로 발생시키는 MRF 패킷을 기다린다. 원하는 멀티캐스트그룹의 MRF를 받으면, MRF로부터 해당 멀티캐스트그룹의 송신원 목록과 코어 송신원 목록을 알아내고 송신원들에게로 MRP를 전송한다. MRP는 ODMRP의 JOIN TABLE과 유사한 정보를 담고 있으며 역시 JOIN TABLE과 유사한 방법으로 각 송신원으로 전달되면서 그 과정에서 수신원으로부터 각 송신원들에 이르는 데이타 전달 메쉬의 멤버 노드들을 선출한다. MRP가 송신원을 향하여 진행하면서 송신원-수신원 메쉬가 형성되는 과정에 대해서는 2.2.2절에서 자세히 설명하기로 한다.

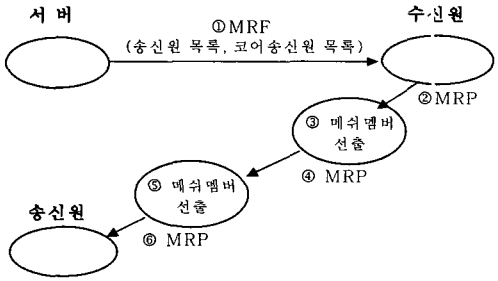

그림 6 수신원의 멀티캐스트그룹 참여/탈퇴

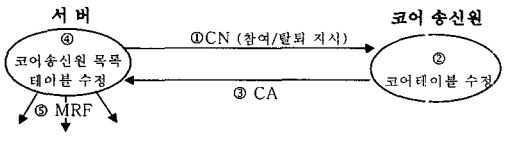

그림 7 코어 송신원 선출 및 관리

▶ 서버의 코어 송신원 선출 및 관리

코어 송신원은 서버에 의해 선출된다. 서버는 멀티캐스트그룹의 송신원 수 및 멀티캐스트그룹의 크기 등을 반영하여 적절하게 코어 송신원 수를 결정할 수 있다. 서버에게 송신원의 이동성 정도에 관한 정보가 있다면 위치변화가 적은 송신원을 코어 송신원으로 선출하고, 가능한 한 코어 송신원들이 네트워크에 고르게 배치되도록 선출하는 것이 유리할 것이다.

코어 송신원이 참여/탈퇴 할 경우에, [그림 7]과 같이 서버는 먼저 CN의 PacketType에 코어 송신원의 참여시 IN을, 탈퇴시 OUT을 표시하여 해당 노드에게 참여 혹은 탈퇴해줄 것을 요청한다. 이 CN을 받은 노드는 CN의 PacketType이 참여를 요청하는 것이면 자신의 코어 테이블의 해당 멀티캐스트그룹 엔트리의 플래그를 세트시키고, 반대로 탈퇴를 요청하는 것이면 플래그를 오프 시킨다. 자신의 코어 테이블의 플래그에 대한 설정이 끝난 후 CN에 대한 응답으로 CA를 서버에게 전달한다. 이 CA의 PacketType에는 CN과 마찬가지로 참여시 IN을, 탈퇴시 OUT을 표시한다. CA를 받은 서버는 PacketType의 IN 혹은 OUT 표시에 따라, 자신의 코어 송신원 목록을 변경하고 MRF를 플러딩하여 전체 네트워크에 변경된 코어 송신원 목록을 알린다.

2.2.2 데이타 전달 경로 형성 및 정기적인 재구성 과정

HMMRP에서의 데이타 전달 메쉬는 송신원과 코어 송신원 사이의 송신원-코어 메쉬, 송신원과 수신원 사이의 송신원-수신원 메쉬로 구성된다. 코어-수신원 메쉬는 송신원-수신원 메쉬의 부분집합으로 코어 송신원을 루트로 하는 트리들에 해당한다. 이 절에서는 이들 각각의 메쉬가 형성되는 과정을 살펴보기로 한다.

▶ 송신원-코어 메쉬

송신원은 처음 멀티캐스트그룹에 참여할 때에는 JR에 대한 응답으로 받은 JA를 통하여 코어 송신원들을 파악하게 되고, 그 이후로는 서버가 플러딩하는 MRF를 통하여 변경되는 코어 송신원 목록을 계속해서 파악하게 된다. JA나 MRF를 받은 송신원은 유니캐스트 라우팅 테이블을 참조하여 JA 혹은 MRF에서 알려준 코어 송신원들 중 자신과 가장 가까운 코어 송신원을 결정하고,

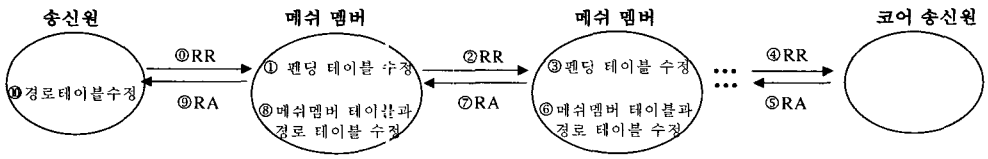

그림 8 송신원-코어 메쉬 형성 및 정기적 재구성

그 코어 송신원에게 RR 패킷을 보낸다. 이 때 RR의 PreHop 필드에는 RR을 만드는 송신원 자신의 주소를 표시한다.

RR 패킷을 받은 노드는 자신이 RR의 NextHop에 표시된 노드라면(즉, RR이 목적지 코어 송신원까지 가는 경로 상의 노드라면), 자신의 펜딩 테이블에 RR에 표시된 (MG ID, SrcAddr, SrcSeq)를 'RR 패킷 ID'로 하는 엔트리를 만들고, 그 엔트리의 전 홉 필드에 RR의 PreHop을 기록한다. 그리고 이 엔트리를 일정 시간 동안만 유지하기 위해 엔트리의 타이머 필드를 세팅한다. RR이 거쳐가는 경로 상의 각 노드는 RR의 PreHop 필드를 자신의 주소로 수정한다. 이렇게 함으로써 각 홉에서는 펜딩 테이블에 저장할 RR의 전 홉 정보(RR을 자신에게 전해 준 노드 주소)를 RR의 PreHop 필드에서 알아낼 수 있다.

RR의 DestAddr에 표시된 코어 송신원이 RR을 받으면, 그 코어 송신원은 RA 패킷을 만들어 이에 응답한다. 이 때 RA의 PreHop 필드에는 코어 송신원 자신의 주소를 기록한다. RA를 받은 노드는 RA의 NextHop에 표시된 노드가 자신인지 체크하고, 만일 그렇다면 자신의 메쉬 멤버 테이블에서 RA의 MG ID에 표시된 멀티캐스트그룹에 해당하는 엔트리를 찾아 그 엔트리의 플래그를 세트시키고, 그 엔트리의 타이머를 현재 시간으로부터 MM-LIFETIME 만큼 경과한 시각으로 갱신한다. 그리고 노드는 타이머가 만기될 때까지 해당 멀티캐스트그룹의 메쉬 멤버로 작동하게 된다.

또한, 이렇게 메쉬 멤버가 된 노드는 송신원-코어 메쉬 상의 멤버이므로, 자신이 받은 RA를 보낸 코어 송신원으로의 경로를 국부적으로 감시/재구성하기 위해 사용하는 경로 테이블에 필요한 정보를 저장한다. 즉, 경로 테이블에서 RA의 MG ID에 표시된 멀티캐스트그룹에 해당하는 엔트리를 찾아 그 엔트리의 코어 송신원 주소 필드와 상위 메쉬 멤버 필드에 RA의 SrcAddr에 표시된 주소(RA를 보낸 코어 송신원)와 RA의 PreHop에 표시된 주소(RA를 전해 준 전 홉 노드, 즉, 코어 송신원으로의 다음 홉)를 각각 추가한다.

그리고 자신의 펜딩 테이블에서 RA의 (MG ID, DestAddr, AckSeq)를 키로 하여 RA를 발생시킨 RR에 대한 엔트리를 찾아 그 엔트리의 '전 홉' 필드로부터 RR을 전달해 주었던 전 홉 노드를 알아내어 RA의 NextHop을 그 값으로 수정한다. 이렇게 함으로써 RA는 RR이 전달되었던 역경로를 통해 전송된다. 또한 RA의 PreHop 필드에는 자신의 주소를 적어 RA를 받게 될 다음 홉에서 코어 송신원으로의 상위 메쉬 멤버가 누구임을 알 수 있도록 한다. 이처럼 RA가 펜딩 테이블의 정보를 이용하여 RR이 경유한 경로를 역으로 거슬러 전달되는 과정이 진행되면서 메쉬 멤버들이 선출된다.

마침내 RA의 DestAddr에 표시된 송신원이 RA를 받게 되면, 송신원은 메쉬 멤버들과 마찬가지로 자신의 경로 테이블을 수정한다. 송신원은 멀티캐스트그룹에 참여하는 동안 MRF를 받을 때마다, 즉 Mesh-Refresh-Interval 마다 가장 가까운 코어 송신원을 결정하고 그 코어 송신원에게 RR 패킷을 보냄으로써, 송신원-코어 메쉬를 재구성하고 그 재구성된 메쉬 상에 있는 노드들의 경로 테이블을 갱신한다. [그림 8]은 이와 같은 송신원-코어 메쉬의 정기적 재구성 과정을 요약하여 보여준 것이다.

▶ 송신원-수신원 메쉬

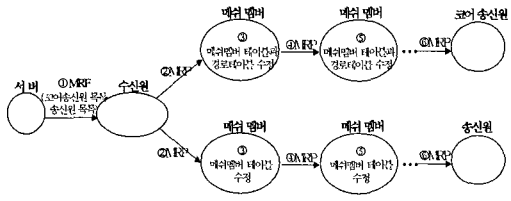

그림 9 송신원-수신원 메쉬 형성 및 재구성

멀티캐스트그룹의 송신원과 수신원을 최단 경로로 잇는 송신원-수신원 메쉬 형성 및 재구성은 서버의 정기적인 MRF 플러딩에 의해 시작된다. 서버는 멀티캐스트그룹 송신원이 존재하는 한, Mesh-Refresh-Interval마다 MRF를 플러딩한다. 해당 멀티캐스트그룹의 수신원 중 하나에게 MRF 패킷이 도달하면 수신원은 경로 테이블에서 MRF의 MG ID에 표시된 멀티캐스트그룹에 해당하는 엔트리를 찾아, 그 엔트리의 코어 송신원 필드에 MRF의 CoreList에 있는 모든 코어 송신원들을 저장하고, 각 코어 송신원에 대한 상위 메쉬 멤버 필드에는 그 코어 송신원으로의 경로 상에 있는 다음 홉을 기록한다.

그리고, MRP 패킷을 만들어 이웃 노드들에게 브로드캐스트하는데, MRP에는 수신원이 송신원 목록 테이블에 기록하고 있는 모든 송신원에 대하여 송신원 주소(MRP 패킷의 SrcAddr 필드), 코어 송신원인지의 여부를 표시하는 플래그(MRP 패킷의 CoreFlag 필드), 그리고 그 송신원으로의 다음 홉이 어느 노드인지(MRP 패킷의 NextHop 필드)를 표시한다. 이 MRP를 받은 노드는 자신이 받은 MRP 패킷의 NextHop 필드에 표시된 다음 홉 노드 중 하나인지 체크하여, 만일 아니라면 받

은 패킷을 버리고 다음 홉 노드 중 하나라면 메쉬 멤버가 된다. 또한, 자신이 NextHop으로 표시되어 있는 송신원이 코어 송신원이라면 코어-수신원 메쉬 상에 있는 메쉬 멤버이므로 코어 송신원 정보를 유지하기 위하여, 자신의 경로 테이블에 그 코어 송신원과 해당 상위 메쉬 멤버(그 코어 송신원으로의 다음 홉)정보를 기록한다. 이렇게 MRP가 송신원까지 전달되는 과정을 통해 수신원과 서버에 등록한 송신원(코어 송신원 포함)들 사이에 송신원-수신원 메쉬가 (재)구성되고, 각 메쉬 멤버는 자신을 통해 연결되고 있는 코어 송신원의 주소와 그에 해당하는 상위 메쉬 멤버 정보를 경로 테이블에 저장한다. [그림 9]는 이와 같이 MRF와 MRP에 의해 송신원-수신원 메쉬가 (재)구성되는 과정을 요약한 것이다.

[그림 10]은 MRP가 포워딩되는 과정에 관한 예를 보여주고 있다. 노드 S_1, S_2, S_3은 멀티캐스트 송신원이며 그 중 S_2는 코어 송신원이고, R_1, R_2, R_3은 멀티캐스트 수신원이다. R_1, R_2, R_3이 서버로부터 MRF를 받으면, 송신원들을 향해 MRP를 보낸다. MRP의 (DestAddr, CoreFlag, NextHop)리스트에는 MRP의 목적지인 송신원 주소, 코어 송신원인지의 여부, 해당 송신원으로의 다음 홉 주소가 각각 기록되며, 이러한 정보는 각각 송신원 목록 테이블과 코어 송신원 목록 및 유니캐스트 라우팅을 이용해 얻는다. 노드 R_1이 발생시킨 MRP는 I_1을 통해 S_1에게 전달되고, I_2를 통해 S_2, S_3에게 전달된다. 노드 R_2가 발생시킨 MRP는 I_2를 통해 S_1, S_2, S_3에게 전달되며, 노드 R_3이 발생시킨 MRP는 I_2를 통해 S_1, S_2에게 전달되고, I_3을 통해 S_3에게 전달된다. I_1은 R_1이 발생한 MRP를 받았을 때, 이 MRP에 자신이 S_1에 대한 NextHop으로 지정되어 있으므로 메쉬 멤버가 되고, 자신의 MRP를 만들어 브로드캐스트 하게 된다. S_1은 코어 송신원이 아니므로 경로 테이블은 갱신할 필요가 없다. R_1, R_2, R_3으로부터 MRP를 받은 I_2는 이들 MRP에서 자신을 S_1, S_2, S_3에 대한 NextHop으로 지정하고 있으므로 역시 메쉬 멤버가 된다. 또한 자신이 연결을 담당하게 되는 송신원 S_1, S_2, S_3중에 S_2에 대해서는 CoreFlag가 세트되어 있으므로 I_2는 자신이 코어-수신원 메쉬 상의 노드임을 인지하고 경로 테이블에 코어 송신원 S_2에 대한 정보를 삽입하게 된다. 즉, 경로 테이블의 해당 멀티캐스트그룹 엔트리에 S_2의 주소와 S_2로의 다음 홉 정보를 각각 코어 송신원 주소 필드와 상위 메쉬 멤버 필드에 저장한다. 그런 다음, I_2는 자신의 MRP를 만들어 브로드캐스트 하게 된다. 한편, I_3은 R_3이 보낸 MRP에 자신이 S_3에 대한 NextHop으로 표시되어 있으므로 역시 메쉬 멤버가 되고 자신의 MRP를 만들어 브로드캐스트한다. S_3은 코어 송신원이 아니므로 경로 테이블은 갱신할 필요가 없다.

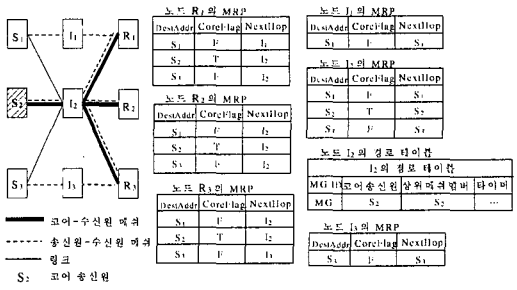

그림 10 MRP의 포워딩 과정 예

여기서 주의해 볼 점은 첫째로, I_1이 만드는 MRP 패킷의 (DestAddr, CoreFlag, NextHop)리스트에는 S_1에 대한 정보만 기록되고 S_2와 S_3에 대한 정보는 제거된다는 것이다. 이것은 I_1이 S_2와 S_3에 대해서는 다음 홉으로 지정되어 있지 않기 때문이다. 둘째로, I_2의 경우 R_1, R_2, R_3으로부터 각각 MRP를 받게 되는데 I_2는 이에 대하여 각각 자신의 MRP를 만드는 것이 아니고 어느 일정 시간 안에 이들 MRP를 모두 받게 되는 경우 이들의 정보를 종합하여 [그림 10]에서 보듯이 하나의 MRP를 만들어 브로드캐스트한다는 것이다.

2.3 국부적으로 손실된 경로를 재구성하는 방법

이 절에서는 HMMRP의 송신원-코어 메쉬와 코어-수신원 메쉬에 대한 국부적인 메쉬 단절 가능성 감지와 재구성에 관하여 설명한다. 일반적으로 에드 혹 노드는 IEEE 802.11에서 제공하는 MAC계층의 비콘 신호[9]를 주기적으로 발생시킴으로써, 현재 자신의 전송 범위 내에 어떤 이웃 노드가 있는지를 파악한다. 만일 전송 범위를 벗어난 이웃 노드가 있다면, HMMRP의 메쉬 멤버는 이웃 노드가 현재 경로 테이블에 있는 상위 메쉬 멤버 중 하나인지를 체크한다.

만약 그렇다면, 메쉬 멤버는 경로 테이블에서 그 상위 메쉬 멤버에 대응하는 코어 송신원으로의 경로가 끊어졌을지도 모른다고 가정하고 코어 송신원으로의 새로운 데이타 경로를 복구하기 위한 작업을 시작한다. 이 작업은 새로운 상위 메쉬 멤버가 필요함을 알리는 요구 단계와 그에 대해 가까이에 있는 메쉬 멤버들이 답하는 응답 단계, 그리고 그 응답 중에서 해당 코어 송신원으로의 최적의 경로를 제공하는 응답을 보내온 메쉬 멤버로의 연결을 시도하는 연결 단계로 진행된다. 먼저, 요

구 단계에서는 코어 송신원으로의 상위 메쉬 멤버를 잃어버린 메쉬 멤버가 가까이에 있는 메쉬 멤버들에게 자신을 알리기 위해 ADVT 패킷을 만들어 플러딩한다. ADVT 패킷에는 잃어버린 상위 메쉬 멤버를 통해 연결되어 있던 코어 송신원의 주소와 자신이 알고 있던 그 코어 송신원까지의 거리(홉 수) 등의 정보를 실어 보내는데 이러한 정보는 경로 테이블과 라우팅 테이블에서 찾아 기록한다. 만일 잃어버린 상위 메쉬 멤버가 지원하던 코어 송신원이 하나 이상이었다면 이 내용을 리스트로 작성한다. ADVT 패킷의 HopCount 필드에는 ADVT 패킷이 경유할 수 있는 최대 홉 수를 기록하는데, ADVT의 최대 경유 홉 수는 2 ~ 3으로 제한하여 국부적으로만 플러딩되도록 한다. 이것은 ADVT 패킷을 발생하는 메쉬 멤버가 크게 멀티캐스트 지역을 벗어난 것이 아닌 경우(이 경우에만 해당 메쉬 멤버가 메쉬에 연결되는 것이 필요함) 대부분 짧은 거리 내에서 다른 메쉬 멤버를 발견할 수 있을 것이라는 가정에 따라 불필요한 플러딩 오버헤드를 줄이기 위함이다. ADVT의 PreHop 필드에는 ADVT를 만드는 메쉬 멤버 자신의 주소를 표시한다.

ADVT 패킷을 전송 받은 노드는, 패치 테이블의 목적지, 다음 홉 필드에 각각 ADVT 패킷의 근원지 주소(ADVT 패킷의 ScrAddr 필드)와 ADVT 패킷을 자신에게 전달해준 전 홉 주소(ADVT 패킷의 PreHop 필드)를 기록함으로써 ADVT의 근원지인 메쉬 멤버로의 라우팅 정보를 남겨둔다. 패치 테이블을 갱신한 후, 노드는 자신이 ADVT에 표시된 멀티캐스트그룹을 담당하고 있으며, ADVT에 표시된 코어 송신원으로의 경로상에 있는 메쉬 멤버인지를 메쉬 멤버 테이블과 경로 테이블에서 각각 확인한다. 또한 자신으로부터 그 코어 송신원까지의 홉 수가 ADVT 패킷에 표시된 Core Hcount보다 작은지 검사한다. 마지막 조건은 재구성 이전의 메쉬에서 ADVT를 생성한 메쉬 멤버보다 해당 멀티캐스트그룹의 코어 송신원으로부터 더 가까운 메쉬 멤버만이 ADVT에 응답하도록 제한하기 위함이다. 이 조건이 모두 맞으면, 노드는 자신이 ADVT를 생성한 메쉬 멤버를 그 메쉬 멤버가 원하는 코어 송신원으로 연결해 줄 수 있다고 판단하여 더 이상 ADVT를 전송하지 않고 이에 대한 응답단계에 들어가게 된다. 한편, 앞의 세 조건 중 하나라도 만족되지 않으면, ADVT 패킷의 HopCount를 1 감소시키고, PreHop 필드를 자신의 주소로 수정하여 플러딩한다. 이렇게 함으로써 각 홉에서는 패치 테이블에 저장할 ADVT의 전 홉 정보를 ADVT의 PreHop 필드에서 알아낼 수 있다. ADVT 패킷의 HopCount가 0이 되면 더 이상 전달하지 않는다.

응답 단계에서는 ADVT를 발생한 메쉬 멤버에게 ADVT 패킷에 대한 응답 패킷인 PATCH 패킷을 전송한다. PATCH 패킷에는 ADVT 패킷에 표시되었던 코어 송신원들 중 자신이 연결해 줄 수 있는 코어 송신원의 주소를 기록하고, PATCH 패킷의 CoreHcount 필드에 자신으로부터 그 코어 송신원까지의 홉 수를 표시한다. 만일 ADVT에서 요청한 코어 송신원들 중 노드가 연결 가능한 코어 송신원이 하나 이상이라면, 이 정보를 리스트로 작성한다. 그리고 ADVT 패킷을 생성한 메쉬 멤버로 향한 다음 홉을 패치 테이블로부터 알아내어 PATCH 패킷의 NextHop 필드에 기록한다.

PATCH 패킷을 받은 노드는 자신의 주소와 패킷의 NextHop 필드 주소를 비교한다. 주소가 같다면 패치 테이블에 PATCH 패킷의 근원지 주소로의 다음 홉 정보를 기록한다. 이것 역시, PATCH 패킷이 지나간 경로를 기록해 두었다가 PATCH 패킷에 대한 응답으로 발생하는 제어 패킷인 PATCH Ack이 전달될 때 라우팅 정보로 사용하기 위함이다. 노드는 ADVT 경유시에 기록하였던 패치 테이블 엔트리를 참조하여 ADVT가 전달된 역경로를 통해 PATCH를 전달한다. 각 홉에서는 PATCH 패킷의 CoreHcount 필드를 1씩 증가시키는데, 이렇게 함으로써 PATCH 패킷이 목적지에 도착했을 때 CoreHcount는 재구성된 경로에서 코어 송신원까지의 거리가 얼마나 되는지를 표시하게 된다.

ADVT를 발생시킨 메쉬 멤버가 PATCH를 받으면, 받은 PATCH 패킷의 정보를 현재 패치 캐쉬에 보관되어 있는 정보와 비교한다. 만일 받은 PATCH 패킷의 CoreHcount값과 패치 캐쉬의 홉 카운트를 비교하여 CoreHcount가 더 적은 경우 PATCH 패킷의 정보로 현재의 패치 캐쉬 엔트리를 대치시킨다. 이렇게 함으로써 패치 캐쉬 안에는 이제까지 받아온 PATCH 패킷 중 CoreHcount값이 가장 작은 PATCH 패킷이 제공하는 정보만을 유지한다. 즉 코어 송신원으로의 가장 빠른 경로에 대한 정보만이 저장된다.

메쉬 멤버가 ADVT를 발생시킨 후 일정 시간이 지나면, 패치 캐쉬의 다음 홉 필드에 기록되어 있는 노드를 해당 코어 송신원에 대한 새로운 상위 메쉬 멤버로 결정하고, 코어 송신원으로 연결되는 새로운 경로 상에 있는 노드들을 메쉬 멤버로 만드는 **연결 단계**에 들어서게 된다. 메쉬 멤버는 먼저 자신의 경로 테이블에서 해당 멀티캐스트그룹 ID와 코어 송신원 주소 엔트리를 찾아 그 엔트리의 상위 메쉬 멤버 주소 필드를 패치 캐쉬의 해당 멀티캐스트그룹 ID, 코어 송신원 주소 엔트리

에 표시되어 있는 다음 홉으로 갱신한다. 그리고 패치 캐쉬에 저장된 PATCH 패킷을 보내준 메쉬 멤버(패치 캐쉬의 PATCH 근원지 주소 필드)에 이르는 경로를 설정하기 위해 그 메쉬 멤버에게 PATCH Ack패킷을 만들어 보낸다. PATCH Ack은 PATCH 패킷 전달시 작성된 패치 테이블 엔트리가 제공하는 라우팅 정보를 이용해 PATCH 패킷이 전달된 역경로를 따라 PATCH 패킷의 근원지로 전달된다.

PATCH Ack을 받은 노드는 자신이 PATCH Ack의 NextHop에 표시된 노드라면 해당 멀티캐스트그룹의 메쉬 멤버로 작동하게 된다. 단, 이때 메쉬 멤버 테이블의 타이머는 일반적인 메쉬 멤버 라이프 타임인 MM_LIFETIME의 반으로 설정하는데, 이것은 임시로 선출된 메쉬 멤버가 지나치게 많아져 중복적으로 데이타가 플러딩되는 것을 막기 위함이다. 또한, 이렇게 메쉬 멤버가 된 노드는 코어 송신원으로 향한 메쉬 멤버이므로, PATCH Ack에 기록된 정보를 이용하여 자신의 경로 테이블도 수정한다. 이와 같이 PATCH Ack이 전달되는 과정에서 메쉬 멤버들이 선출되며 마침내 PATCH Ack이 PATCH 패킷의 근원지에 도착하면, 국부적 경로 재 설정 과정이 완료된 것이다. [그림 11]은 이와 같은 국부적 경로 재구성 과정을 요약하여 도식화한 것이다.

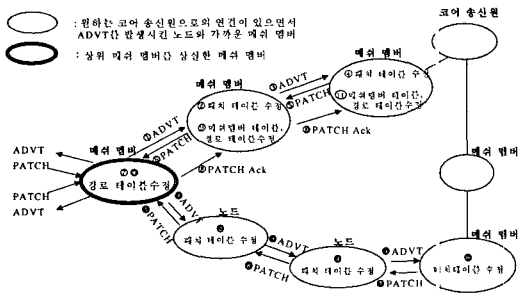

그림 11 국부적인 경로 재설정 과정

[12]에서 제안한 방식에서는, PATCH 패킷이 전달되면서 모든 PATCH 패킷에 의해 메쉬 멤버가 선출됨에 따라 ADVT를 발생한 노드 주변에 불필요한 메쉬 멤버가 많이 선출되어 불필요한 데이타 복사본이 많이 발생하는 단점이 있었다. 이를 수정하기 위해 HMMRP에서는 PATCH Ack을 사용하여 코어로의 최단의 연결을 제공하는 PATCH 패킷에 대해서만 PATCH Ack을 보내어 메쉬 멤버를 선출함으로써 불필요한 메쉬 멤버 증가를 막고 중복 데이타 발생 오버헤드를 줄였다.

3. 시뮬레이션 결과 및 분석

본 장에서는 제안하는 HMMRP 성능을 평가하기 위한 시뮬레이션 모델을 설명하고 그 결과를 분석한다. 시뮬레이션은 WindowsNT 4.0 시스템에서 Microsoft Visual C++6.0으로 구현하였으며 UCLA에서 개발한 Global Mobile Simulation(GloMoSim) 라이브러리를 사용하였다. GloMoSim는 무선 네트워크 시뮬레이션을 위해 개발한 도구로서, 라이브러리 기반으로 순차적이고 병렬 처리가 가능한 시뮬레이터이다[10]. GloMoSim에서는 각 계층마다 기존에 연구된 여러 네트워크 프로토콜들이 라이브러리로 구현되어 있다. 이 라이브러리는 C 기반 병렬 시뮬레이션 언어인 PARSEC으로 개발되었기 때문에 새로운 프로토콜과 모듈들은 PARSEC을 사용하여 프로그램되거나 수정할 수 있다. 3.1절에서는 시뮬레이션 모델을 제시하며, 3.2절에서는 실험 결과를 토대로 본 논문에서 제안하는 프로토콜의 성능에 대하여 분석하여 본다.

3.1 시뮬레이션 모델

시뮬레이션 모델은 다음과 같다. 우선, 1000m×1000m 지역에 노드 50개를 무작위로 배치하였다. 그리고 이 노드들 중에서 멀티캐스트그룹의 송신원과 수신원을 다시 무작위로 선정한다. 선정된 송신원과 수신원은 시뮬레이션 시작 시 결정되어 끝날 때까지 멀티캐스트그룹의 멤버로 남아있게 된다. 멀티캐스트그룹의 수신원 수는 실험 시나리오에 따라 최소 2에서 최대 30개이며 송신원 수는 6개이고 그 중 코어 송신원의 수는 2개로 가정한다. 송신원은 1초당 1개의 패킷을 내보내며, 그 페이로드 크기는 512bytes 이다. 각 노드의 전파 전송 범위는 250 미터이며, 채널 용량은 2Mbits/sec 이다. 각 실험에 대해 시뮬레이션 시간은 700초이다. [표 3]은 본 시뮬레이션에서 사용한 계층별 프로토콜을 보여준다.

노드의 움직임 모델로는 Random Waypoint를 사용하였다. 이 모델에 의하면 노드는 이동할 목적지를 무작위로 선택하여 일정 속도로 그 목적지를 향해 움직여간다. 그리고 일단, 그 목적지에 도착하면, 노드는 그 자리에서 일정 시간동안 머물러 있게 된다. 다시 일정 시간이 경과하면, 다시 노드는 다른 방향의 목적지를 향해 위의 동작을 반복한다. 따라서 일정한 방향성 없이, 주어진 속도와 멈춤 시간에 의해서 노드의 움직임이 발생한다. 시뮬레이션에서는 노드의 속도를 다양하게 0 km/hr 에서 72km/hr 까지 실험하였다. 단, 코어 송신원은 모든 경우에 이동 속도를 3.6km/hr라 가정하였다.

이러한 시뮬레이션 모델하에서 HMMRP 성능을 평가

표 3 시뮬레이션에서 사용한 계층별 프로토콜

Layer:	Models;
Physical (Radio propagation)	Free space
Data Link (MAC)	802.11
Network (Routing)	DSR, HMMRP
Transport	TCP, UDP
Application	CBR

하기 위해, 총 프로토콜 오버헤드로서 제어 패킷 발생양과 함께 불필요한 데이타 패킷 복사본의 양을 측정하였으며, 제어에 관련된 오버헤드로서 제어 패킷 수만을 측정하였다. 제어 패킷 별 발생하는 오버헤드가 크게 차이가 나지 않는다고 가정한다면, 제어 패킷 발생양에 따라 대역폭 소모뿐 아니라 각 제어 패킷을 처리하기 위한 프로세싱 오버헤드도 커지므로 "발생한 제어 패킷 수"라는 메트릭에 의해 제어 복잡도 정도를 대략적으로 측정할 수 있는 방법이라고 생각하여 제어 패킷 수만을 측정한 것이다. 다음은 IETF MANET 워킹그룹에서 프로토콜 평가를 위해 제안한 기준[11]을 참고하여 결정한 4가지 HMMRP 성능 평가 기준이다.

▶ **평균 데이타 패킷 전달율(A):**

평균 데이타 패킷 전달율은 다음 식에 의하여 구한다. 각 수신원이 받은 데이타 패킷 수를 계수하는데 있어 중복적으로 도착한 데이타 패킷은 한번만 계수하였다. 평균 데이타 패킷 전달율은 송신원에서 수신원으로 보낸 데이타 패킷이 평균적으로 얼마나 제대로 전달되었는지를 나타내며, 이 값으로 프로토콜의 유효 정도를 가늠할 수 있다.

$$A=\frac{\frac{\sum_{\text{수신원}} \text{각 수신원이 받은 데이타 패킷수}}{\text{수신원수}}}{\sum_{\text{송신원}} \text{각 송신원에서 보낸 총 데이타 패킷수}}$$

▶ **데이타 패킷 당 네트워크 전체에 발생한 제어 패킷의 수(B):**

데이타 패킷 당 네트워크 전체에 발생한 제어 패킷의 수는 다음 식과 같이 구하는데 역시, 각 수신원에 도착한 데이타 패킷 수를 계수하는데 있어 중복적인 패킷은 계수하지 않았다. 이것은 하나의 데이타 패킷을 전달하기 위해 발생되는 제어 패킷이 몇 개나 되는지를 보여준다. 시뮬레이션 시작시 이미 송신원과 코어 송신원이 결정되어 있다고 가정하므로, 계수된 제어 패킷은 RR, RA, MRF, MRP, ADVT, PATCH, PATCH Ack 등이다.

$$B=\frac{\text{네크워크에 발생한 총 제어 패킷수}}{\sum \text{각 수신원에 도착한 데이타 패킷수}}$$

▶ **데이타 패킷 당 네트워크 전체에 발생한 중복된 데이타 패킷의 수(C):**

메쉬 구조에서는 데이타 패킷이 여러 경로를 통해 전달되어 결과적으로 수신원이 동일한 데이타 패킷을 중복적으로 받을 수 있다. 따라서 제어 패킷 이외에도 이러한 중복된 데이타 패킷이 채널 오버헤드가 될 수 있다. 이 오버헤드를 측정하기 위해, 다음과 같이 데이타 패킷 하나 당 발생된 데이타 오버헤드를 계산하였다.

$$C=\frac{\text{네트워크에 발생한 총 데이타 패킷수}}{\sum \text{각 수신원에 도착한 데이타 패킷수}}$$

▶ **총 오버헤드(D):**

네트워크 전체에서 해당 멀티캐스트를 위해 시뮬레이션 기간동안 발생된 모든 제어 패킷 오버헤드와 중복적으로 전달된 데이타 패킷 오버헤드의 합으로써 전체 오버헤드를 측정해 보았다.

$$D=B+C$$

3.2 시뮬레이션 결과 및 분석

▶ 평균 데이타 패킷 전송률

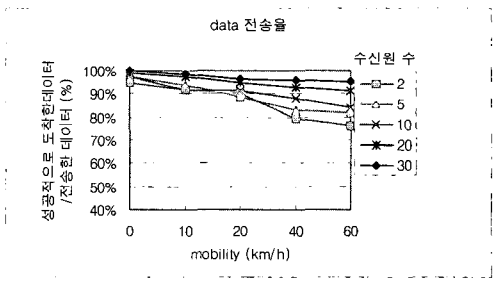

그림 12 데이타 패킷 전송률

송신원 수는 고정해 놓고 수신원의 수를 2 ~ 30까지 변화시켜 보면서 HMMRP의 성능을 살펴보았다. [그림 12]에서 보듯이, 대부분의 결과가 80%이상의 전송률을 보이기는 하지만, 수신원 수에 관계없이 노드의 이동 속도가 빨라질수록 성능이 약간씩 감소함을 볼 수 있다. 그러나 이것은 이동성에 대한 다른 프로토콜들의 성능저하 정도에 비하면[7] 정도가 매우 경미한 것으로 HMMRP가 이동성에 강함을 보여주고 있다. 그리고 전반적으로 멀티캐스트 멤버 수가 작아질수록 전송률이 저조해지고 이동성에도 더 민감하게 성능이 저하하는 현상을 볼 수 있는데, 이것은 수신원의 수가 적으면 송신원에서 수신원까지의 메쉬가 두텁게 형성되지 못하여 충분한 다중 경로를 제공하지 못하기 때문이다. 이러한 성능 저하 경향도 대부분의 다른 멀티캐스트 라우팅 프로토콜

에서 볼 수 있는 것이다[1]. 그러나 HMMRP에서는 수신원의 수가 10 이상인 경우에는 이미 이동성이 따른 성능저하 정도가 상당히 둔화되는 것을 볼 수 있다.

▶ 데이타 패킷 당 발생한 제어 패킷 오버헤드

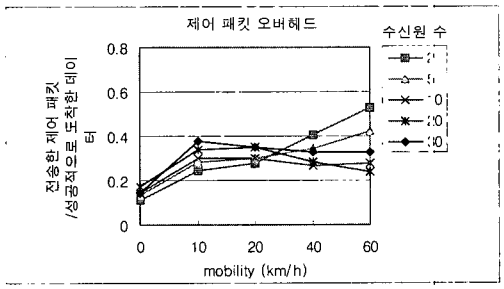

그림 13 제어 패킷 오버헤드

[그림 13]는 한 데이타 패킷당 발생한 제어 패킷 수를 나타낸 것이다. [그림 13]에 의하면, 수신원 수가 2나 5와 같이 적은 경우에는 노드의 이동이 커질수록 데이타 패킷당 발생하는 제어 패킷 수가 증가한다. 제어 패킷중 MRF와 MRP, RR과 RA들은 고정된 인터벌마다 발생하므로, 노드의 이동 정도와 상관없이 거의 일정한 반면, ADVT, PATCH, PATCH Ack 패킷은 코어 송신원으로의 경로가 손실된 경우 발생하는 제어 패킷들로서, 노드의 이동 정도와 비례하여 증가된다. 따라서, 경로 재설정 작업으로 인한 ADVT, PATCH, PATCH Ack 발생은 노드의 이동 정도와 비례하여 증가될 것이다. 특히, 수신원수가 2나 5와 같이 적은 경우는 수신원까지 성공적으로 도달하는 데이타 패킷 수가 노드 속도 증가에 따라 현저히 줄어들기 때문에 데이타 패킷 하나 당 발생하는 제어 패킷의 수가 더욱 현저히 커지게 된다.

그런데, 수신원수가 10 이상인 경우에서는 제어 패킷 수가 노드의 속도 증가에 별로 영향을 받지 않고 일정한 비율을 보인다. 그 이유는 수신원에 도착한 데이타 패킷의 수가 노드의 이동 정도에 영향을 받지 않고 일정하며, 멤버수가 적을 때에 비해 메쉬가 두터우므로 경로 재 설정작업인 플러딩이 가까이에서 매우 효율적으로 이루어져 국부적 경로 재구성을 위한 패킷이 적게 발생하기 때문이다. 따라서, 멀티캐스트 멤버의 수가 클수록 제어 패킷 오버헤드가 감소됨을 알 수 있다.

[그림 14]은 데이타 패킷당 발생한 데이타 패킷 수를 나타낸 것이다. 메쉬 구조에서는 하나의 데이타 패킷을 여러 경로로 전달하게 되기 때문에 제어 패킷에 의한 오버헤드 뿐 아니라, 중복된 데이타를 전송하기 위한 오

▶ 데이타 패킷 당 발생한 데이타 패킷 오버헤드

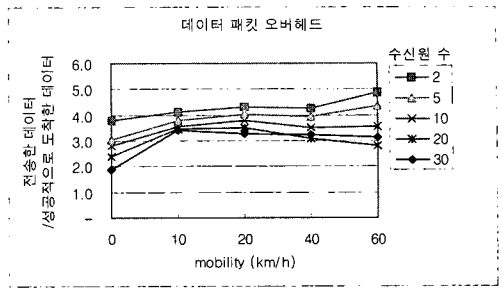

그림 14 데이타 패킷 오버헤드

버헤드가 발생한다. [그림 14]을 보면, 데이타 전송률이 좋지 않았던 수신원수 2의 경우를 제외하고는 노드의 이동성과 관계없이 거의 일정한 비율을 유지한다. 한편, 수신원수에 따른 결과를 비교해 보면, 수신원수가 많아질수록 중복된 데이타 패킷 오버헤드가 줄어드는 것을 볼 수 있다. 그 이유는, 수신원수가 많을수록 공유되어 선출되는 데이타 경로가 증가하며, 따라서 이 경로를 통과하는 데이타 패킷은 한번에 여러 수신원에게 전달되기 때문이다. 이외에도 이 결과로 유추할 수 있는 것은, HMMRP에서의 국부적인 링크 단절을 재 설정하기 위해 메쉬 멤버를 재 선출하는 방식이 필요이상으로 데이타 전달 경로를 중복하여 발생하지 않는다는 것이다.

▶ 총 오버헤드

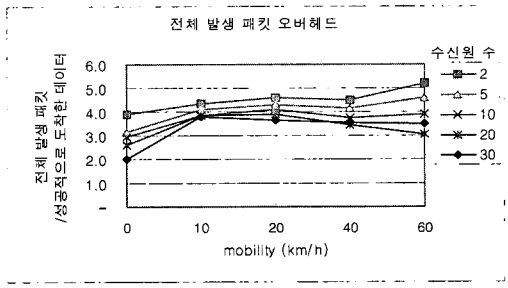

그림 15 총 오버헤드

[그림 15]는 에드 혹 네트워크 망에 발생한 전체 오버헤드 패킷의 양을 보여주고 있다. 제어 패킷 오버헤드와 중복된 데이타 패킷 오버헤드 중 후자가 더 비중이 훨씬 더 크기 때문에 [그림 15]는 데이타 패킷 오버헤드를 보여주는 [그림 14]과 유사한 경향을 보이고 있다. [그림 15]를 보면, 수신원수가 많을수록 더 적은 오버헤드 패킷

을 발생시키며, 이러한 경향은 노드의 이동 속도가 증가하더라도 크게 변화가 없다. 수신원수가 2나 5와 같이 적은 경우에 속도가 60km 일 때 비율이 약간씩 증가되는 이유는 데이타 패킷이 수신원에 도착하지 못하고 중간에 손실되는 경우가 많이 발생하기 때문이다. 수신원수가 10 이상에서는 속도가 증가해도 비율이 일정하게 유지되고 있다. 이는 이동성이 높아져도 성공적으로 수신원에 배달된 데이타 패킷 수가 거의 일정하고 중복 데이타나 제어 패킷의 오버헤드가 매우 제한적이기 때문이다.

4. 결 론

본 논문에서는 에드 혹 망의 이동성에 효율적으로 대처할 수 있는 메쉬 기반의 멀티캐스트 라우팅 프로토콜인 HMMRP를 제안하였다. HMMRP는 송신원 중 일부를 코어 송신원으로 두고 타 송신원들이 코어 송신원 중 하나에 연결되도록 한다. 그리고 ODMRP와 유사한 송신원별 트리의 합집합으로 송신원과 수신원이 연결되도록 한다. HMMRP는 이들 연결 경로 상의 노드들로서 데이타 전달 메쉬를 형성하고 이를 정기적으로 재구성하는데 특히 일반 송신원으로부터 코어 송신원에 이르는 경로와 코어 송신원으로부터 수신원에 이르는 경로에 해당하는 메쉬에 대해서는 정기적인 재구성 기간보다 훨씬 짧은 기간마다 국부적으로 메쉬 단절 가능성을 감시하고 복구하도록 한다. 이렇게 함으로써 임의의 송신원 수신원은 최소한 송신원과 코어 송신원 사이의 경로 그리고 그 코어 송신원으로부터 수신원에 이르는 경로를 통해서는 항상 연결되어 있는 상태이므로 이동성이 높은 경우라도 정기적인 메쉬 재구성 기간을 짧게 잡지 않고 높은 데이타 전달율을 제공할 수 있다.

HMMRP의 성능을 분석하기 위해 시뮬레이션을 수행한 결과, 타 논문들을 통해 발표된 기존의 메쉬 기반 에드 혹 멀티캐스트 프로토콜들의 이동성에 대한 성능 저하에 비해 HMMRP의 이동성에 대한 성능 저하는 매우 적고, 특히 멀티캐스트그룹의 멤버수가 어느 정도 이상이 되면 HMMRP의 이동성에 대한 성능 저하가 경미함을 볼 수 있었다.

참 고 문 헌

[1] Sung-Ju Lee, William Su, Mario Gerla, and Rajive Bagrodia, "A Performance Comparison Study of Ad Hoc Wireless Multicast Protocols," In Proceeding of Inforcom'2000.

[2] Sung-Ju Lee, Mario Gerla, Ching-Chuan Chiang, "On-Demand Multicast Routing Protocol," In Proceeding of IEEE WCNC'99. New Orleans, LA, Sep. 1999.

[3] Sung-Ju Lee, Willian Su, Mario Gerla, "On-Demand Multicast Routing Protocol (ODMRP) for Ad Hoc Networks," Internet Draft, draft-ietf-manet-odmrp-02.txt, July. 2000.

[4] Sung-Ju Lee, Willian Su, Mario Gerla, "Ad hoc Wireless Multicast with Mobility Prediction," In Proceeding of IEEE ICCCN'99, New Orleans, LA, Sep.1999.

[5] J.J. Garcia-Luna-Aceves, E.L. Madruga, "A Multicast Routing Protocol for Ad-Hoc Networks," In Proceeding of IEEE INFOCOM '99, New York, NY, June, 1999.

[6] J.J. Garcia-Luna-Aceves, E.L. Madruga, "The Core-Assisted Mesh Protocol," IEEE Journal on Selected Areas in Communications, vol. 17, no.8, Aug. 1999.

[7] Meejeong Lee, Ye Kyung Kim, "PatchODMRP : An Ad-hoc Network Multicast Routing Protocol" In Proceeding of ICOIN-15, Beppu, Japan, 2001년 2월.

[8] S. Deering, "Host Extensions for IP multicasting," RFC-1112, August, 1989.

[9] IEEE Computer Society LAN MAN Standards Committee, Wireless LAN Medium Access Protocol (MAC) and Physical Layer (PHY) Specification, IEEE Std 802.11-1997. The Institute of Electrical and Electronics Engineers, New York, 1997.

[10] UCLA Computer Science Department Parallel Computing Laboratory and Wireless Adaptive Mobility Laboratory, GloMoSim: A Scalable Simulation Environment for Wireless and Wired Network Systems. http://pcl.cs.ucla.edu/projects/domains/glomosim.html

[11] M.S. Corson and J. Macker, "Mobile ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations," Request For Comments 2501, Internet Engineering Task Force, Jan. 1999.

[12] M.Ohta and J.Crowcroft, "Static Multicast,"Internet Draft, http://www.ietf.org/internet-drafts /draft-ohta-static-multicast-01.txt, Octorber, 1998.

김 예 경
정보과학회논문지 : 정보통신
제 28 권 제 1 호 참조

이 미 정
정보과학회논문지 : 정보통신
제 28 권 제 1 호 참조