Multicast over TCP/IP HOWTO

Juan-Mariano de Goyeneche*jmseyas@dit.upm.es* <mailto:jmseyas@dit.upm.es> v1.0, 20 March 1998

김현필*khp@kldp.org* <mailto:khp@kldp.org> v0.9, 10 July 2000

이 문서에서는 TCP/IP 네트웍에서 멀티캐스트와 관련된 전반적인 사항을 설명한다. 따라서 상당부분 리눅스에 국한되는 내용이 아니다. (아직..GNU/리눅스를 쓰지 않을 경우에만) 멀티캐스트는 지금까지도 활발히 연구중인 분야이다. 따라서 이 글의 작성 시점에서는 표준안 대부분이 초안(draft)일 뿐이다. 이점 염두에 두고 읽으시기 바란다.

목 차

제 1 절 소개

나는 이 문서에서 TCP/IP 네트웍에서 멀티캐스트에 대해 가능한 한 광범위하며 최신의 그리고 정확한 정보를 다루도록 노력할 것이다. 여러분의 어떠한 의견이라도 환영한다. 내용 중에 오류가 있거나 조언 및 추가할 내용이 있다면 저자에게 보내주기 바란다.

1.1 멀티캐스트(Multicast)란 무엇인가?

멀티캐스트는 곧 필요성이다. 만일 인터넷상의 여러-전체가 아닌-호스트에게로 전송해야할 상당한량의 정보를 가지고 있다면 바로 멀티캐스트가 그 해답이다. 예를 들어, 인터넷에 분산되어 호스트들로 원격 회의를 하기 위해 실시간으로 영상과 음성을 전송하는 경우, 멀티캐스트를 사용 할 수 있다.

멀티캐스트는 사용자가 자신의 수신기를 조정(관심 있는 채널의 주파수 선택)함으로서 정보를 받는다는 점에서 TV나 라디오와 유사하다. 다른 것은 제쳐두고 자신이원하는 것만 수신한다.

1.2 유니캐스트(Unicast)의 문제점

브로드캐스트(Broadcast)도 아니며 멀티캐스트도 아닌 것을 유니캐스트라 한다. 별로 좋지 않은 정의 같은데...패킷을 전송할 때 송신 프로세스 하나, 수신 프로세스 하나가 존재할 때 이것을 유니캐스트라 한다. TCP는 그 자체가 본질적으로 유니캐스트를 지향한다. 한편, UDP는 훨씬 다양한 형식을 가질 수 있지만 수신하는 프로세스가 단 하나일 때 이 역시 유니캐스트라 할 수 있다.

인터넷 초기에는 유니캐스트로 충분했었다. 1993년 4.4 BSD에서 최초로 멀티캐스트를 구현해서 세상에 내놓았을 때까지는 아무도 그것을 필요로 하는 것 같지 않았다. 그런데 왜 멀티캐스트라 불리는 것이 나오게 된 것일까?

말할 필요도 없이 인터넷은 초기의 모습으로부터 엄청난 변화를 겪었다. 특히 웹의 등장은 상황을 완전히 뒤바꾸어 놓았다. 사람들은 mail, FTP 그 이상의 것을, 자신들의 홈페이지에서 그림을 보기 원했고 더 나아가 음성과 동영상을 원했다.

오늘날의 기술이라면, 당신의 웹을 보고자하는 모든 이에게 유니캐스트 연결을 한다고 해도 충분히 그 "비용(cost)"를 감당할 수 있다. 하지만 음향이나 영상을 보내고자 한다면 엄청난 대역폭이 필요하다. 멀티캐스트를 고려에 넣는다면 두 가지 선택이 있다. 아니, 멀티캐스트가 등장하기 전까지는 두 가지 선택이 존재했었다. 유니캐스트를 통해 각 수신자별로 독립된 연결을 할 것인가, 아니면 브로드캐스트를 할 것인가.

전자는 부적절하다. 음성/영상의 단일 연결조차도 상당한 대역폭을 차지할 터인데 이러한 연결이 수백 또는 수천이라고 상상해 보라. 아마도 당신의 컴퓨터와 네트웍은 붕괴하고 말 것이다.

브로드캐스트가 해답인 듯 하지만 그것도 확신할 수가 없다. 우리 LAN에 있는 모든 호스트가 회의에 참석한다면 브로드캐스트가 확실히 해답이 될 수 있다. 모든 패킷은 한번씩만 전송되며 다른 모든 호스트는 브로드캐스트 주소를 통해 패킷을 수신할 것이다. 하지만, 문제는 일부 호스트만이 이 패킷에 수신하려할 경우다. 게다가, 정말로 회의에 참여하고자하는 단 하나의 호스트가 몇 개의 라우터를 거쳐야 도달 할 수 있는 외부 LAN에 존재한다면 어려운 문제가 된다. 알다시피, 브로드캐스트는 단일 LAN 내부에서만 가능한데, 브로드캐스트 패킷을 라우팅하여 외부 LAN에 전달할 방법이 없다.

최선의 해결책은 패킷을 -마치 TV나 라디오 채널처럼-일부 특정한 주소로만 보내는 것이다. 그러면 회의에 참여하고자하는 모든 호스트는 이 패킷이 네트웍을 통과할 때, 목적지 주소(destination address)를 감지하여 읽어들인 후 자신의 IP계층(layer)으로 보내어 복호화(demultiplexing)한다. 즉, 멀티캐스트는 패킷이 단 한번만 송신되어서 네트웍의 모든 호스트들이 그것을 읽는다는 점에서는 브로드캐스트와 같지만, 이 패킷들이 오직 커널(kernel)에서 원할 경우에만 읽고 처리한다는 점에서 브로드캐스트와 다르다.

이 특별한 패킷들은 IP 패킷이기 때문에 커널 수준(kernel level)에서 라우팅한다. 커널에 라우팅 경로를 알려주는 라우팅 알고리즘에 차이점이 있을 것이다.

제 2 절 멀티캐스트

2.1 Multicast 주소

IP 주소는 32비트 중에서 상위 비트들에 의해서 클래스(class)로 분류된다.

```
Bit -->  0                           31            Address Range:
        +-+----------------------------+
        |0|       Class A Address      |       0.0.0.0 - 127.255.255.255
        +-+----------------------------+
        +-+-+--------------------------+
        |1 0|     Class B Address      |     128.0.0.0 - 191.255.255.255
        +-+-+--------------------------+
        +-+-+-+------------------------+
        |1 1 0|   Class C Address      |     192.0.0.0 - 223.255.255.255
        +-+-+-+------------------------+
        +-+-+-+-+----------------------+
        |1 1 1 0|  MULTICAST Address   |     224.0.0.0 - 239.255.255.255
        +-+-+-+-+----------------------+
        +-+-+-+-+-+--------------------+
        |1 1 1 1 0|     Reserved       |     240.0.0.0 - 247.255.255.255
```

```
+-+-+-+-+-+---------------------+
```

우리가 주의 깊게 할 것은 "D 클래스 주소"다. 목적지 주소가 "1110"으로 시작하는 IP 데이터그램은 멀티캐스트 데이터그램이다.

나머지 28비트는 송신할 데이터그램의 멀티캐스트그룹을 구분하는데 쓰인다. 방송을 듣기 위해 라디오를 어떤 주파수에 맞추는 것과 유사하게 특정한 멀티캐스트 그룹으로 전송되어오는 패킷을 수신하기 위해서 우리의 커널을 특정한 그룹에 맞추어야 한다. 이 과정이 이루어 졌을 때, *호스트가, 지정한 인터페이스를 통해 그룹에 참여했다*고 할 수 있다. 후에 더 자세히 다룰 것이다.

*예약된 멀티캐스트 그룹(well known multicast groups)*이라 불리는 특별한 그룹들이 있는데 이러한 그룹은 다음과 같은 특수한 용도로 쓰이기 때문에 개인적인 프로그램 제작 시에 사용할 수 없다.

- 224.0.0.1 은 전체 호스트 그룹이다. 멀티캐스트 인터페이스는 시동시 이 그룹에 참여해야하므로 이 주소로 ping을 보내면 모든 멀티캐스트 호스트들이 응답할 것이다.
- 224.0.0.2 은 전체 라우터 그룹이다.
- 224.0.0.4 은 전체 DVMRP 라우터 그룹이다.
- 224.0.0.5 은 전체 OSPF 라우터 그룹이다.
- 224.0.013 은 전체 PIM 라우터 그룹이다.
- ...

이 특수 그룹들은 모두 RFC문서 "Assigned Numbers"에 정기적으로 등록된다.

어떠한 경우든지, 224.0.0.0에서 224.0.0.225의 범위는 지역적인 목적(관리나 유지/보수)을 위해 예약되어 있으며 멀티캐스트 라우터들도 이 범위 내의 주소로 목적지로 하는 데이터그램은 포워딩하지 않을 것이다. 이와 유사하게 239.0.0.0에서 239.255.255.255의 범위는"administrative scoping"을 위하여 예약되어 있다. ("administrative scoping")에 관해서는 2.3.1을 참조할 것).

2.2 적응 단계(Levels of Conformance)

호스트가 만족해야하는 멀티캐스트 규격에는 세 가지 적응 단계가 있다.

Level 0 : *the no support for IP Multicasting* IPv4에서는 멀티캐스트가 의무적으로 구현해야하는 것이 아니기 때문에 인터넷에 있는 많은 호스트와 라우터들은 Level 0 상태에 있다. (하지만 IPv6에서는 의무적으로 멀티캐스트를 지원하도록 하고 있다.) 더 이상 설명은 필요 없다. 이 상태에 있는 호스트는 멀티캐스트 패킷을 보내거나 받을 수 없으며, 멀티캐스트 패킷을 완전히 무시한다.

Level 1 : *the support for sending but not receiving multicast IP datagrams* 따라서, 데이터그램을 보내기 위해서 멀티캐스트 그룹에 가입할 필요는 없다. Level 0 호스트를 Level 1 호환 상태로 만들기 위해서는 IP 모듈을 약간 변경해야 한다. 자세한 내용은 2.3에서 다룬다.

Level 2 : *the full support for IP multicasting* Level 2 호스트는 멀티캐스트 데이터그램 송신과 수신이 가능해야 한다. 또, 멀티캐스트 그룹에 참가하고 탈퇴할 수 있어야 하며 새로 갱신된 멀티캐스트 정보를 라우터에 알 릴 수 있어야 한다. 따라서, 호스트의 TCP/IP 스택에 인터넷 그룹관리 프로토콜 Internet Group Management Protocol (IGMP) 이 구현되어 있어야 한다.

2.3 멀티캐스트 데이터그램 전송

이상으로 미루어보아, 멀티캐스트 트래픽은 UDP로 트랜스포트 계층에서 처리하는 것이 분명하다. TCP는 점대점(point-to-point)연결을 제공하는 것이므로 멀티캐스트 트래픽에 적합하지 않다. (새로운 멀티캐스트

지향 트랜스포트 프로토콜 설계와 구현에 대한 연구가 활발히 진행중이다. 9장 "멀티캐스트 트랜스포트 프로토콜"을 참조할 것)

이론상, 응용프로그램에서 단지 UDP 소켓을 열고 class D 멀티캐스트 주소를 목적지로 하는 데이터그램을 부어넣기만 하면 된다. 그렇지만, 송신 프로세스가 통제권을 가지기 위해서 해주어야 할 작업이 있다.

2.3.1 TTL.

IP 헤더의 TTL(Time To Live) 필드는 멀티캐스트에서 중요한 의미를 가진다. 이 필드는 라우팅 에러로 인하여 데이터그램이 네트웍을 영원히 떠돌아다니는 것을 방지한다. 라우터는 네트웍간을 이동하는 데이터그램의 TTL 필드를 감소시키며 TTL 필드가 0이 되는 데이터그램은 버린다(drop). IPv4 멀티캐스트에서 TTL은 문턱값(threshold)의 의미를 지닌다. 다음 예를 보면 그 용도가 분명해진다.

우리 부서의 모든 호스트가 속하는 아주 길고 대역폭을 많이 차지하는 영상회의를 한다고 가정하자. 우리의 LAN에는 엄청난 용량의 트래픽이 발생할 것이며, 아마 우리 부서는 다양한 LAN이 존재하는 큰 네트웍일 것이다. 이 경우 우리는 LAN을 통하여 회의를 열기 원하지만, 우리의 멀티캐스트 트래픽 때문에 인터넷 전체가 붕괴되는 것을 원치 않을 것이다. 따라서, 멀티캐스트 트래픽이 라우터간을 얼마나 *멀리*이동할 수 있도록 할 것인지 제한할 필요가 있다. 이것이 TTL의 용도이다.

라우터는 각각의 인터페이스에 대해 TTL 문턱치(threshold)가 할당하고 있으며 이 문턱치보다 큰 TTL값을 가진 데이터그램만이 포워딩된다. 데이터그램이, 어떤 문턱치가 할당되어있는 라우터를 지날 때, TTL 값이 문턱치만큼 감소되는 것이 아니라는 점에 주의하라. 오직 비교만이 이루어진다 (앞에서 언급했듯이 TTL은 데이터그램이 라우터를 지날 때마다 1씩만 감소된다).

다음 리스트에 TTL문턱치와 그에 해당되는 범위가 표시되어 있다.

```
----------------------------------------------------------------------
TTL     Scope
----------------------------------------------------------------------
   0    호스트 내부로 제한. 인터페이스로 출력되지 않음.
   1    동일 서브넷으로 제한. 라우터는 포워딩하지 않음.
 <32    동일 사이트(site), 단체나 부서로 제한.
 <64    동일 지역(region)으로 제한.
<128    동일 대륙으로 제한.
<255    무제한. 전세계.
----------------------------------------------------------------------
```

"사이트(site)" 나 "지역(region)" 에 대한 정확한 정의는 없다. 그것은 이 제한을 가할 관리자에게 달려있다.

이 TTL기법은 모든 요구에 부응할 만큼 유연성을 제공하지 못하며, 특히 겹쳐지는 지역(overlapping regions)을 다루거나 지리적(geographic)이거나 위상적(topologic)적인 그리고 대역폭 제한적인 연결에는 적합하지 않다. 이러한 문제를 해결하기 위하여 1994년 administratively scoped IPv4 multicast regions 이 제정되었다 (D. Meyer's "Administratively Scoped IP Multicast" Internet draft 참조). 이것은 TTL을 이용하지 않고 IP 주소를 이용하여 범위제한(scoping)을 가한다. 239.0.0.0 에서 239.255.255.255까지의 IP주소가 이 관리용 범위제한을 위해 예약되어 있다.

2.3.2 루프백(Loopback)

전송 호스트가 Level 2 적응 단계에 있으며 데이터그램을 전송하는 그룹의 멤버로 참여하고 있을 때, 패킷 복사본이 루프백 된다. 이것은 네트웍 인터페이스 카드가 해당 네트웍 인터페이스가 자신이 네트웍으로 전송한 패킷을 네트웍으로부터 다시 읽어온다는 것이 아니라, IP 계층이 데이터그램을 인식하여 전송 전에 패킷을 IP 입력 큐에 복사해 넣는 것을 의미하며 이 기능은 기본 값(default)으로 설정되어 있다.

이러한 기능은 필요한 경우도 있고 그렇지 않을 경우도 있다. 따라서 전송 프로세스는 이 기능을 원한다면 켜거나 끌 수 있다.

2.3.3 인터페이스 선택

다수의 네트웍 인터페이스가 부착되어 있는 호스트에서는 응용 프로그램으로 하여금 어떤 인터페이스로 전송을 출력해야하는지 결정할 수 있도록 해 주어야한다. 만일 지정 값이 없다면 커널이 관리자의 설정에 기초하여 기본 값을 정한다.

2.4 멀티캐스트 데이터그램 수신

2.4.1 멀티캐스트 그룹 참여(Join)

브로드캐스트는 멀티캐스트보다 비교적 구현하기 쉽다. 브로드캐스트는 커널에 패킷 처리 규칙을 추가로 알려줄 필요가 없다. 커널은 브로드캐스트 패킷을 읽고 적절한 응용프로그램으로 전달하는 방법을 알고 있다.

그러나 멀티캐스트에서는 우리가 어떤 그룹에 관심이 있는지 커널에 알려줄 필요가 있다. 즉, 커널에 어떤 그룹에 "참여(join)@quot;하도록 요청해야 한다는 것이다. 하드웨어에 따라, 멀티캐스트 데이터그램을 하드웨어가 직접 필터링하거나 아니면 IP 계층에서 하는 경우도 있다. 어떤 경우에는 양쪽 모두에서 이루어지기도 한다. 오직 "참여 (join)"를 통해서 등록된 그룹만이 받아들여진다.

본질적으로 우리가 그룹에 참여한다는 것은 커널에게 다음과 같은 이야기를 하는 것을 의미한다. " *기본 설정으로 멀티캐스트 데이터그램을 처리하지 않는다는 것을 알고 있지만, 내가 이 멀티캐스트 그룹에 참여하고자 한다는 점을 기억해주기 바란다. 그러니 이 네트웍 인터페이스에서 보이는 패킷가운데, 이 멀티캐스트 그룹 주소를 목적지 필드에 포함하고 있는 멀티캐스트 데이터그램을 읽고 (나뿐만 아니라 그것에 관심을 가지고 있는 모든) 프로세스에게 전달해 주기 바란다.* ".

고려할 것 : 우선 그룹에 단지 참여만 하는 것이 모든게 아니라는 점을 주목하라. 우리는 특정한 네트웍 인터페이스 상에서 그룹에 참여한다. 물론, 하나 이상의 인터페이스로 같은 그룹에 참여하는 것이 가능하다. 만일 인터페이스를 확실히 지정하지 않는다면 데이터그램이 전송될 때 라우팅 테이블에 의거하여 커널이 지정할 것이다. 또한, 하나 이상의 프로세스가 동일한 인터페이스로 동일한 멀티캐스트 그룹에 참여하는 것도 가능하다. 모든 프로세스들은 해당 인터페이스로 보내지는 데이터그램을 수신한다.

전에 언급했듯이, 모든 멀티캐스트 호스트들은 시동시에 전체 호스트 그룹에 참여하기 때문에 224.0.0.1 로 ping을 보내면 네트웍 내에서 모든 호스트들이 응답을 보낼 것이다.

끝으로, 프로세스가 멀티캐스트 데이터그램을 받기 위해서는 , 커널에게, 그룹에 참여한 후 데이터그램을 송신하는 포트를 묶도록(bind) 요청해야 한다. UDP 계층은 패킷을 해석(demultiplex)하기 위해 목적지 주소와 포트번호를 같이 사용하며 어떤 소켓으로 혹은 어떤 소켓들로 패킷을 보낼지 결정한다.

2.4.2 멀티캐스트 Group 탈퇴

프로세스가 멀티캐스트 그룹을 떠날 때는 커널에 그 그룹을 떠나고자 한다고 알린다. 이것이 커널이 그 그룹으로 오는 멀티캐스트 데이터그램을 더 이상 받지 않음을 의미하는 것이 아니라는 점을 이해해야 한다. 만일 "multicast join" 신청을 제출한 프로세스가 더 있고 그 그룹에 여전히 관심을 가지고 있는 상태라면 커널은 여전히 패킷을 수신할 것이다. 이러한 경우 호스트는 모든 프로세스가 그 그룹을 떠나기로 결정할 때까지 여전히 그룹에 남아 있게된다.

추가 : 그룹을 떠난다 할지라도, 수신하던 포트에 여전히 연결(bind)된 채로 남아있을 것이며 아직 멀티캐스트 그룹에 참여하고 있는 프로세스가 더 있으면 멀티캐스트 traffic을 계속 수신하게 될 것이다.

멀티캐스트 group 에 참여함에 있어서, 요점은 IP 와 data link 계층에 해당 그룹으로 향하는 멀티캐스트 데이터그램을 받도록 지정하는 것이다. (경우에 따라서는 명시적으로 하드웨어에 지정하는 경우도 있다) 이

것은 프로세스 단위(per-process membership)가 아니라 호스트 단위(per-host membership)이다.

2.4.3 IP 멀티캐스트 주소에서 Ethernet/FDDI 주소로의 사상 (mapping)

Ethernet과 FDDI 에서 프레임(frames)은 48 비트의 목적지 주소공간을 가진다. 멀티캐스트 IP 주소를 ethernet/FDDI로 사상하기 위한 멀티캐스트 ARP 같은 것을 피하기 위해서, IANA는 멀티캐스트를 위해 주소공간을 예약했다. 목적지가, 01-00-5e-00-00-00 에서 01-00-5e-ff-ff-ff (16진수)사이의 모든 ethernet/FDDI 프레임은 멀티캐스트 그룹을 위한 데이터를 가진다. 접두어(prefix) 01-00-5e 는 해당 프레임이 멀티캐스트임을 나타내며, 바로 다음 비트는 항상 0이다. 따라서, 나머지 23 비트만이 멀티캐스트 주소를 위해 사용된다. 그런데, 멀티캐스트 그룹의 IP는 28비트 길이이므로 1대1 사상은 불가능하다. 오직 IP 멀티캐스트 그룹의 23개 하위 비트(Least Significant Bit)만이 프레임에 위치한다. 남는 5개의 상위 비트는 무시되며32개의(2^5=32) 서로 다른 멀티캐스트 그룹이 동일한 ethernet/FDDI address사상된다. 이것은 ethernet 계층이 완전한 필터로 작용하지 못함을 의미하며, IP 계층이 data link 계층을 통과해온 데이터그램을 받을 것인지 버릴 것인지를 결정해야 할 것이다. IP 계층이 최후의 완벽한 필터이다.

FDDI상의 IP 멀티캐스팅에 관한 자세한 내용은 RFC 1390 "Transmission of IP and ARP over FDDI Networks" 에 나와있다. IP Multicast 주소의 ethernet으로의 사상에 관한 더욱 자세한 내용은 draft-ietf-mboned-intro-multicast-03.txt "Introduction to IP Multicast Routing"을 참조하라.

Token-Ring LAN 상에서의 IP 멀티캐스트에 관심이 있다면 RFC 1469를 참조하라.

제 3 절 커널 요구 사항과 설정

Linux는 당연히(설마 의심을?) 완전한 Level 2 멀티캐스트를 지원한다. 모든 송수신 요구사항을 만족하며 멀티캐스트 데이터그램에 관해서 라우터(mrouter)처럼 작동한다.

멀티캐스트 송수신이 하고싶다면 커널 설 정시 "IP: multicasting"에 yes라고 해주기만 하면 된다. Linux box를 멀티캐스트 라우터 (mrouter)로 사용하고 싶다면 "IP: forwarding/gatewaying", "IP: multicast routing" 그리고 "IP: tunneling"을 선택해주면 된다. 후자는 새로운 버전의 mrouted가 멀티캐스트 데이터그램을 유니캐스트 데이터그램에 캡슐화(encapsulation)해서 보내기 위해 IP tunneling 에 의존하기 때문이다. 이것은 유니캐스트 전용 네트웍에 분산되어있는 멀티캐스트 호스트들간의 연결(tunneling)을 위해서 필요하다. (mrouted는 라우팅 알고리즘-라우팅 정책-을 구현하고 커널에 멀티캐스트 데이터그램을 어떻게 라우팅할 것인지 알려주는 daemon이다).

어떠한 버전의 커널들은 멀티캐스트 라우팅을 "EXPERIMENTAL"로 분류하고 있기 때문에, "Code maturity level options"섹션에서 "Prompt for development and/or incomplete code/drivers"옵션을 가능하게 해주어야 한다. 만일 mrouted가 실행중일 때 우리의 Linux box가 속해있는 동일 네트웍에서 생성된 traffic은 정상적으로 다른 네트웍에 포워딩해주는데, 다른 네트웍의 traffic을 볼 수 없다면 ICMP 에러메시지를 받고있지는 않는지 확인해 보라. 거의 모든 경우가 IP tunneling을 활성화 시켜놓지 않은 경우 일 것이다. 일단 알고나면 정말 간단한 것이지만, 정말이지, 모르는 경우에는 시간 꽤나 걸리고 이유도 쉽사리 알기 어렵다. 이런 상황에서는 스니퍼(sniffer)가 정말 유용하다!

(멀티캐스트 라우팅에 대해서는 "Routing Policies and Forwarding Techniques" 섹션에서, mrouted 와 tunnel에 대해서는 "The MBone" 과 "Multicast applications"에서 추가로 설명한다).

일단 커널을 컴파일하고 설치한 후에는 멀티캐스트 트래픽을 위한 기본 라우팅경로를 설정 해야한다. 우리의 목표는 224.0.0.0 네트웍으로의 경로를 추가하는 것이다.

대부분의 사람들이 이 단계에서 겪는 무제는 마스크 값으로 인한 어려움이다. 만일 Terry Dawson의 유명한 NET-3-HOWTO를 읽었다면, 올바른 값을 알아 맞추기 어렵지 않을 것이다. 그 문서에서 설명했듯이, 넷마스크(subnetmask)는 우리의 IP 주소의 네트웍 부분을 모두 1로 채우고 호스트 부분을 모두 0으로 채운 32비트 숫자이다. 섹션 2.1에서 보았듯이 클래스D 멀티캐스트 주소는 네트웍/호스트형식으로 구분되어있지 않다. 대신 18비트의 그룹 구분자와 4비트의 클래스D 구분자로 이루어져있다. 이 4비트가 네트웍 부분에 대

응되고 28비트가 호스트 부분에 대응된다. 따라서 서브넷마스크는 11110000000000000000000000000000 쉽게 읽자면, 240.0.0.0이 된다. 그러면 완전한 명령어는 다음과 같이 될 것이다.

route add 224.0.0.0 netmask 240.0.0.0 dev eth0

route 프로그램의 버전에 따라서 add뒤에 -net 을 붙여야 하는 경우도 있다.

여기서 eth0이 멀티캐스트 가능하다고 가정했으며, 달리 지정되지 않을 경우 멀티캐스트 트래픽이 eth0으로 출력될 것이다. 이것이 우리에게 해당되지 않는다면 dev 변수 값을 다른 값으로 바꾸어 주어야한다.

여기서 다시 한번 /proc 파일시스템의 유용성이 입증된다. 우리는 우리호스트가 참여하고 있는 그룹을 /proc/net/igmp를 통해서 확인해 볼 수 있다.

제 4 절 The MBone.

장점과 함께 단점도 있는 기술이다. 나의 사견이지만, 장점은 분명하다는 점이다. 주된 단점은 아직 수백의 호스트들, 특히 라우터들이 아직 지원하지 않고 있다는 점이다. 결과적으로 멀티캐스트를 사용하는 사람들은 새로운 장비를 구입하고 운영체제를 수정하며 자신의 장소에 멀티캐스트라는 섬을 만든다. 그리고 그들은 다른 곳에서 자신과 비슷한 일을 하고 있는 사람들과 통신할 수없음을 알게된다. 그들 사이의 멀티캐스트를 지원하지 않는 단 하나의 라우터 때문에...

해답은 분명하다. 그들은 가상 멀티캐스트 네트웍을 인터넷 최 상단에 만들기로 결정한다. 즉 멀티캐스트 라우터를 가진 사이트들은 그들간에 직접적으로 통신하는 것이다. 하지만 유니캐스트 라우터를 통해서 연결된 사이트들은 다른 멀티캐스트 섬으로 유니캐스트 갭슐화된 멀티캐스트 트래픽을 보내야 할 것이다. 중간에 걸쳐진 라우터들-유니캐스트전용 라우터들-은 유니캐스트 트래픽을 다루는 것이 되므로 문제가 되지 않는다. 끝으로 수신 측에서는 캡슐화된 트래픽을 다시 원래의 멀티캐스트 트래픽으로 해석할 것이다. 이렇게 멀티캐스트 트래픽을 유니캐스트 트래픽으로 다시 멀티캐스트 트래픽으로 변환하는 것을 멀티캐스트 터널링(tunneling)이라 한다.

즉, MBone 또는 멀티캐스트 백본(Multicast Backbone)은 멀티캐스트 섬들을 멀티캐스트 터널로 이어주는 가상 멀티캐스트 네트웍(virtual multicast network)이다.

이 MBone에 관한 활동들이 활발하다. 그러한 반면 지금도 인터넷에서는 실시간 오디오 및 비디오로 풍부한 원격회의가 이루어지는 중이다. 최근 Linus Torvalds가 Silicon Valley Linux Users Group에서 발표한 것이 라이브로 인터넷 전송된 일이 있다.

MBone에 관한 더 자세한 자료는 다음을 참고하라. `<http://www.mediadesign.co.at/newmedia/more/mbone-faq.html>`

제 5 절 멀티캐스트 응용프로그램들

멀티캐스트를 다루는 대부분의 사람들은 조만 간에 MBone에 연결할 것이며 mrouted를 필요로 할 것이다. 우리 역시 멀티캐스트라우터가 없는 상태에서 다른 서브넷으로 멀티캐스트 트래픽을 전달하기 원한다면 mrouted가 필요하다. mrouted는 멀티캐스트 데이터그램을 유니캐스트 형식으로 캡슐화하여 (IP에서 IP로) 유니캐스트 라우터를 통해 멀티캐스트 트래픽을 전송할 수 있도록 해주며, 이 외에도 몇 가지 기능이 있다. 가장 중요한 것은 멀티캐스트 데이터그램의 발신지와 목적지를 감지하여 어떻게 라우팅을 할 것인지 아니면 하지 않을 것인지 커널에 알려준다는 점이다. 멀티캐스트 라우터를 가지고 있다고 하더라도 mrouted는 커널에 데이터그램으로 무엇을 할지 알려준다. (어떻게가 아니라 무엇이다. mrouted는 "이것을 저 인터페이스가 연결된 네트웍으로 포워딩하라."고 말하지만, 실제로 포워딩은 커널이 수행한다.) 실제 포워딩 작업 부분(어디서 할 것인가)과 포워딩 알고리즘 부분(어떻게 할 것인가)을 구분하면 포워딩 코드를 오직 한번만 작성해서 커널에 넣으면 되기 때문에 아주 유용하다. 포워딩 알고리즘과 정책은 추후에 사용자 공간의 데몬(user space daemon)에서 구현되며, 따라서, 커널을 재 컴파일 할 필요 없이 라우팅 정책을 쉽게 수정할 수 있다.

다음 사이트에서 리눅스로 포팅된 mrouted를 구할 수 있다. <ftp://www.video.ja.net/mice/mrouted/Linux/> 이 사이트의 미러는 다음을 참조하라. <ftp://www.video.ja.net/mice/README.mirrors>

다음으로 리눅스로 포팅된 멀티캐스트 응용프로그램들 가운데 MBone에 연결하도록 작성된 것들에 대해 알아본다. 여기에 있는 리스트는 Michael Esler의 "Linux Multicast Information"에서 가져왔다. <http://www.cs.virginia.edu/~mke2e/multicast/>. 멀티캐스트와 리눅스에 대한 정보가 아주 많아 추천할 만하다.

음성 회의(Audio Conference)

- NeVoT- NetworkVoice Terminal <http://www.fokus.gmd.de/step/nevot>
- RAT -UCL Robust-Audio Tool <http://www-mice.cs.ucl.ac.uk/mice/rat>
- vat - LBL visual audio tool <http://www-nrg.ee.lbl.gov/vat/>

영상 회의(Video Conference)

- ivs -Inria videoconferencing system <http://www.inria.fr/rodeo/ivs.html>
- nv - Network video tool <ftp://ftp.parc.xerox.com/pub/net-research/>
- nv w/ Meteor - Release of nv w/ support for the Matrox Meteor (UVa) <ftp://ftp.cs.virginia.edu/pub/gwtts/Linux/nv-meteor.tar.gz>
- vic - LBL video conferencing tool <http://www-nrg.ee.lbl.gov/vic/>
- vic w/ Meteor - Release of vic w/ support for the Matrox Meteor (UVa) <ftp://ftp.cs.virginia.edu/pub/gwtts/Linux/vic2.7a38-meteor.tar.gz>

기타 유틸리티

- mmphone Multimediaphone service <http://www.eit.com/software/mmphone/phoneform.html>
- wb - LBL shared white board <http://www-nrg.ee.lbl.gov/wb/>
- webcast - Reliable multicast application for linking Mosaic browsers <http://www.ncsa.uiuc.edu/SDG/Software/XMosaic/CCI/webcast.html>

회의 도구(Session Tools)

약간 설명할 필요가 있어서 이것들은 뒤에 놓았다. 회의를 하게되면 회의를 위한 서비스들(음성, 영상, 공유 칠판, 등등...)에 멀티캐스트 그룹과 포트들을 할당한다. 회의를 시작하면 멀티캐스트 그룹과 포트, 프로그램(vic, vat, ...)등에 관한 정보가 주기적으로 MBone으로 전송된다. 회의 도구들은 이러한 정보를 "듣고"우리가 관심 있는 회의가 언제 있는지 (혹은 있을지) 알아보기 쉽게 알려준다. 또한, 회의에 참여하는 과정을 쉽게 해준다. 프로그램을 실행시켜서 참여할 그룹/포트번호를 알려주는 대신 클릭하는 것만으로 세션도구들이 적절한 프로그램을 실행시키고 그 프로그램에 적절한 정보를 제공한다. 회의도구들은 보통 사용자 자신의 회의를 MBone에 알리는 역할도 한다.

- gwTTS - University of Virginia tele-tutoring system <http://www.cs.Virginia.EDU/~gwtts>
- isc - Integrated session controller <http://www.fokus.gmd.de/step/isc>
- mmcc - Multimedia onference control <ftp://ftp.isi.edu/confctrl/mmcc>
- sd - LBL session directory tool <ftp://ftp.ee.lbl.gov/conferencing/sd>
- sd-snoop - Tenet Group session directory snoop utility <ftp://tenet.berkeley.edu/pub/software>
- sdr - UCL's next generation session directory <ftp://cs.ucl.ac.uk/mice/sdr>

제 6 절 멀티캐스트 프로그래밍

멀티캐스트 프로그래밍... 혹은, 우리만의 응용프로그램 작성하기.

멀티캐스트를 지원하기 위해서 몇 가지 확장된 응용 프로그래밍 인터페이스(API)가 필요하다. 그 가운데 대부분은 setsockopt() (커널에 정보를 전달함) 와 getsockopt() (멀티캐스트 관련 정보 구함) 두 가지 시스템 호출을 통해 처리한다. 새로운 시스템 호출 두개가 멀티캐스트를 지원하기 위해 추가되었다는 뜻이 아니다. setsockopt()/getsockopt()는 4.2 BSD 시절부터 있었다. 단지 커널에 전달해 주어야하는 새로운 멀티캐스트 옵션이 추가되었을 뿐이다.

setsockopt()/getsockopt() 함수의 선언이다.

```
int getsockopt(int s, int level, int optname, void* optval, int* optlen);

int setsockopt(int s, int level, int optname, const void* optval, int optlen);
```

첫 번째 변수 s는 시스템 호출을 적용할 소켓이다. 멀티캐스트에서 이 소켓은 AF_INET 계열 이어야하며, 소켓 타입은 SOCK_DGRAM 또는 SOCK_RAW를 쓸 수 있다. 대부분은 SOCK_DGRAM으로 쓰지만, 라우팅 데몬을 제작하거나 수정할 계획이라면 SOCK_RAW 이 필요할지 모른다.

두 번째 변수 level은 옵션이나, 질의(query), 메시지(message)를 처리할 레이어를 의미한다. 소켓 레이어는 SOL_SOCKET, IP 레이어는 IPPROTO_IP, 등등이다. 멀티캐스트 프로그래밍에 level은 항상 IPPROTO_IP 이어야 한다.

optname은 우리가 설정 값을 지정하는 것인지 얻으려 하는 것인지를 의미한다. 설정 값 자체는(커널에 설정하거나 커널로부터 읽어오는 두 경우 모두) optval이다. 멀티캐스트 프로그래밍에서 optnames는 다음과 같다.

optname	setsockopt()	getsockopt()
IP_MULTICAST_LOOP	yes	yes
IP_MULTICAST_TTL	yes	yes
IP_MULTICAST_IF	yes	yes
IP_ADD_MEMBERSHIP	yes	no
IP_DROP_MEMBERSHIP	yes	no

optlen은 optval이 가리키는 데이터구조의 크기를 의미한다. 단, getsockopt()에서는 반환 값이 된다. 커널은 optname 값을 optval이 가리키는 버퍼에 써넣은 후 optlen을 통해 자료의 크기를 알려준다.

setsockopt()/getsockopt()모두 실행 결과 성공하면 0, 에러가 발생하면 -1을 반환한다.

6.1 IP_MULTICAST_LOOP.

프로그래머라면, 데이터를 호스트로 루프백 시킬 것인지 결정해야한다. 만일, 프로세스가 여러 개이거나 데이터를 리스닝"listening"중인 사용자가 있다면 루프백을 가능하게 해야한다. 하지만, 카메라 이미지를 전송하는데 자기 화면에서 이미지를 볼 필요가 없다면, 아마도 루프백이 필요 없을 것이다. 후자의 경우 프로그램은 컴퓨터에 부착된 카메라로부터 영상자료를 이미 가지고 있을 것이기 때문에, 다시 소켓으로부터 자료를 받기 원하는 경우는 드물 것이다. 루프백은 기본 값으로 활성화되어 있다.

optval이 포인터이기 때문에 다음처럼 해서는 안 된다.

setsockopt(socket, IPPROTO_IP, IP_MULTICAST_LOOP, 0, 1);

루프백을 정지시키려면 다음과 같이 하고

u_char loop; setsockopt(socket, IPPROTO_IP, IP_MULTICAST_LOOP, &loop, sizeof(loop));

루프백을 활성화시키려면 loop를 1로, 중지시키려면 0으로 설정한다.

소켓의 현재 루프백상태를 확인하려면 다음과 같이 한다.

u_char loop; int size;

getsockopt(socket, IPPROTO_IP, IP_MULTICAST_LOOP, &loop, &size)

6.2 IP_MULTICAST_TTL.

특별히 지정하지 않는다면, 멀티캐스트 데이터그램은 지역 네트웍 외부로 전송되는 것을 방지하기 위해 TTL 1을 기본 값으로 전송된다. TTL을 원하는 값으로(0에서 255까지) 바꾸기 위해서 TTL값을 변수에 넣고 프로그램에서 다음과 같이 써주면 된다.(여기서는 "ttl"이라고 했다.)

u_char ttl; setsockopt(socket, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(ttl));

IP_MULTICAST_LOOP에서와 흡사하다.

6.3 IP_MULTICAST_IF.

일반적으로 시스템 관리자는 기본 인터페이스를 지정하며, 멀티캐스트 데이터그램은 그쪽으로 보내진다. 프로그래머는 이 옵션을 사용하여 지정된 기본 인터페이스를 무시하고 정해진 소켓에서 사용할 인터페이스를 선택할 수 있다.

struct in_addr interface_addr; setsockopt (socket, IPPROTO_IP, IP_MULTICAST_IF, &interface_addr, sizeof(interface_addr));

이후부터 이 소켓에서 생성된 모든 멀티캐스트 트래픽은 위에서 선택한 인터페이스로 출력된다. 원래 설정을 회복하여 관리자의 설정을 무시하고 커널이 외부 출력 인터페이스를 선택하도록 하려면 인터페이스에 INADDR_ANY를 설정하고 위와 동일한 옵션으로 setsockopt()를 호출한다.

외부 출력 인터페이스를 선택함에 있어서, 다음 ioctl이 유용할 것이다. SIOCGIFADDR (인터페이스 주소 얻기), SIOCGIFCONF (전체 인터페이스 목록 얻기), SIOCGIFFLAGS (인터페이스 상태 플래그(flag)를 얻어서 인터페이스가 멀티캐스트 가능한지 확인 - IFF_MULTICAST 플래그-).

호스트가 멀티캐스트 라우터로 작동한다면, 여러 인터페이스를 가지고 있거나 IP_MULTICAST_IF옵션이 설정되어 있지 않을 경우, 다른 인터페이스에서 멀티캐스트 포워딩이 가능하더라도 멀티캐스트는 기본 인터페이스를 통해 이루어진다.

6.4 IP_ADD_MEMBERSHIP.

커널에 우리가 어떤 그룹에 관심이 있는지 알려주어야 한다고 했던 것을 다시 생각해 보자. 그룹에 관심 있는 프로세스가 하나도 없다면, 그 그룹을 향해서 전송되어오는 패킷은 모두 폐기된다. 따라서 커널에 우리의 관심 그룹을 알리고, 그룹의 회원(member)이 되기 위해서는 다음과정이 필요하다. 우선 ip_mreq 구조체의 내용을 채운다. 그리고, optval을 설정하여 setsockopt()를 호출해서 커널에 이 구조체를 넘겨주면 된다.

ip_mreq 구조체는 (/usr/include/linux/in.h에 있음) 다음과 같다.

```
struct ip_mreq
{
        struct in_addr imr_multiaddr;   /* IP multicast address of group */
        struct in_addr imr_interface;   /* local IP address of interface */
};
```

주의 : 이 구조체의 "물리적" 정의는 위의 파일에 명시되어 있다. 하지만, 코드의 이식성을 위해서 <linux/in.h>을 포함(include)시켜서는 안 된다. 대신 <linux/in.h>를 포함하는 <netinet/in.h>을 포함시켜라.

첫 번째 멤버, imr_multiaddr는 우리가 참여할 그룹주소를 말한다. 회원(membership)은 그룹뿐만 아니라 인터페이스에도 관련이 있다는 점을 명심해야한다. 따라서, 두 번째 멤버, imr_interface값도 알려주어야 한다. 우리가 분산된 호스트에 있더라도 이러한 방식으로 각기 다른 인터페이스를 이용하여 같은 그룹에 참여할 수 있다. imr_interface는 와일드카드 주소(INADDR_ANY)를 이용할 수 있으며 이 경우 인터페이스를 선택하는 일은 커널이 한다.

일단 이 구조체를 채우면 (struct ip_mreq mreq;로 정의 했다고 하면) 다음 방법으로 setsockopt()를 호출하면 된다.

setsockopt (socket, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq));

동일 소켓으로 여러 그룹에 참여 할 수 있다는 점에 주목하라. 그룹 수 제한은 IP_MAX_MEMBERSHIPS에 정의되어 있으며 커널버전 2.0.33에서 20이다.

6.5 IP_DROP_MEMBERSHIP.

이 과정은 그룹에 참여하는 것과 유사하다.

struct ip_mreq mreq; setsockopt (socket, IPPROTO_IP, IP_DROP_MEMBERSHIP, &mreq, sizeof(mreq));

여기서 mreq는 그룹에 참여 할 때 사용했던 것과 동일한 구조체다. imr_interface에 INADDR_ANY를 사용한다면 커널이 선택한 첫 번째 인터페이스가 제거된다.

같은 소켓으로 아주 많은 그룹에 참여했을 경우 연결을 종료하기 위해 모든 회원 관계를 삭제할 필요는 없다. 소켓을 닫으면 연결된 모든 회원 관계는 커널이 의해 삭제된다. 소켓을 연 프로세스를 종료(kill)시킬 때도 동일하다.

끝으로 그룹에서 회원을 제거하는 과정이 호스트로 하여금 그 그룹으로 오는데이터그램을 받지 못하도록 하는 것을 의미하는 것은 아니다. 동일 인터페이스로 그 그룹에 참여한 소켓이 IP_DROP_MEMBERSHIP의 상태가 되더라도 호스트는 그룹 멤버의 목록을 보관한다.

ADD_MEMBERSHIP 과 DROP_MEMBERSHIP 은 성공, 실패 여부를 즉시 반환한다. (nonblocking operations)

제 7 절 멀티캐스트 내부

이 절의 목적은 멀티캐스트가 어떻게 작동하는지를 이해하는데 필요한 정보를 제공하는 것 도 아니고 멀티캐스트 프로그래밍에 대한 정보를 제공하자는 것도 아니다. 하지만 멀티캐스트의 기반을 이루는 프로토콜과 구현을 살펴봄으로서 쉽게 저지르는 실수와 잘못된 이해를 피할 수 있을 것이다.

7.1 IGMP.

IP_ADD_MEMBERSHIP 과 IP_DROP_MEMBERSHIP에서 이 명령으로 커널에 제공한 정보는 어떤 멀티캐스트 데이터그램을 받고 어떤 것을 버릴 것인지를 결정하는데 쓰인다고 했다. 맞는 말이라고 할 수 있지만, 모두 맞는 말은 아니다. 이러한 일반화는 전 세계 모든 멀티캐스트데이터그램이 우리의 호스트로 전달된 후, 호스트에서 프로세스가 발행한 회원관계를 확인 후 데이터그램의 폐기 여부를 결정한다는 말이 된다. 생각할 필요도 없이, 이것은 엄청난 대역폭 낭비임을 알 수 있다.

실제로는 호스트가 자신의 라우터에게, 그 라우터는 상위 라우터에게, 그리고 그 상위 라우터에게....어떤 멀티캐스트 그룹에 관심이 있는지 알려준다. 멀티캐스트 그룹 트래픽을 받을 것인지 말 것인지 결정하는 알고리즘 자체에는 상당히 변화가 심한데, 단 한가지 변하지 않는 것이 있다면, 이 정보를 전달하는 방법이다.

이것은 IGMP(Internet Group Management Protocol을 이용한다. 프로토콜번호 2인 IGMP는 ICMP와 유사한 새로운 프로토콜인데 IP 데이터그램을 이용하며 레벨 2 적응단계의 호스트는 이 프로토콜을 의무적으로 구현해야 한다. 전술한 바와 같이 이것은 라우터에게 회원정보를 전송하는 호스트와 라우터사이의 통신 양쪽 모두 사용한다. 다음 글에서 호스트-라우터간 관계만을 설명하기로 한다. 왜냐하면 mrouted소스 코드를 제외하고는 라우터-라우터간 통신을 해설하는 자료를 찾지 못하였기 때문이다. (RFC 1075-Distance Vector Multicast Routing Protocol은 이제 사용하지 않으며, mrouted는 아직 문서화되지 않은 변형된 DVMRP를 구현하고 있다.)

RFC 988-IGMP 버전 0은 이제 사용하지 않는다.

IGMP 버전 1은 RFC-1112에 정의되어 있고, RFC-2236 (IGMP version 2) 에서 개정되어 현재까지 많이 사용하고 있다. 리눅스 커널에서는 IGMP 버전 1 전체와 버전 2일부가 구현되어 있다.

지금부터 이 프로토콜에 대한 약식설명을 보자. RFC-2236을 펼쳐 놓고 정확한 정식 설명을 보는 것도 좋다.

모든 IGMP 메시지는 다음 구조를 가진다.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Type     | Max Resp Time |           Checksum            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Group Address                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

IGMP 버전 1 (이하 IGMPv1) 는 전송시 "Max Resp Time" 을 "Unused",즉 모두 0 으로 하고 수신시에는 무시한다. "Type"란은 4비트 폭으로 나누어 "Version"과 "Type"으로 사용한다. IGMPv1 에서는 "회원질의(Membership Query)"를 0x11 (version 1, type 1)으로 확인하고, IGMPv2 에서는 0x11 로 확인하기 때문에 8비트를 사실상 동일하게 해석 한다.

IGMPv2 는 주로 추가사항들로 이루어져 있기 때문에 IGMPv1 을 일단 설명한 후 IGMPv2 추가사항을 설명하는 편이 좋을 것 같다.

다음 논의에서 라우터는 모든 IP 멀티캐스트 데이터그램을 받는다는 것을 기억하라.

7.2 IGMP version 1.

라우터는 주기적으로(1분에서 2분에 한번씩) TTL값 1의 IGMP 호스트 회원 질의를 전체 호스트 그룹(224.0.0.1)으로 보낸다. 모든 멀티캐스트 호스트들이 이 메시지를받지만 IGMP 호스트 회원 보고(Host Membership Report)가 폭주(storm)하는 것을 방지하기 위해 즉시 응답하지 않는다. 대신 질의를 수신한 인터페이스에 속하는 각 그룹에 대해, 임의 지연 타이머를 시작한다.

조만 간에 타이머가 완료되면 호스트는 역시 TTL 1의 IGMP 회원 보고를 멀티캐스트 그룹어드레스로 전송한다. 이 메시지는 이미 그룹에 참여하고 있는 모든 호스트들과 타이머가 완료되기를 기다리고 있는 호스트들에게 전달된다. 그러면 각 호스트들은 타이머를 중지하고 더 이상 아무런 보고도 하지 않는다. 하지만 이것으로 라우터는 자신의 서브넷에 그 멀티캐스트 그룹의 멤버가 존재한다는 것만 알면 되기 때문에 회원이 몇이나 있는지 알기 위해 더 이상, 보고 받을 필요가 없다.

몇 번 질의한 후에도 그 그룹에 대해 아무런 회원 보고가 없으면 라우터는 멤버가 없는 것으로 간주하고 해당 그룹의 트래픽을 서브넷으로 포워딩하지 않는다. IGMPv1 에서는 "그룹 탈퇴 메시지(Leave Group messages)"가 없다는 점에 주목하라.

호스트가 새 그룹에 참여하면 커널은 그룹으로 보고를 보내기 때문에 각각의 프로세스들은 새로운 회원질의가 도착할 동안(1,2분 정도) 기다릴 필요가 없다. "IP_ADD_MEMBERSHIP" 정에서 보았겠지만, 이 IGMP 패킷은 IP_ADD_MEMBERSHI명령에 대한 응답으로 커널이 생성한다. "새 그룹"이라는 말을 주의

깊이 보자. 호스트가 이미 참여중인 그룹에 대해 프로세스가 IP_ADD_MEMBERSHIP 명령을 보낼 경우 우리는 이미 그 그룹의 트래픽을 받고 있기 때문에 아무런 IGMP 패킷도 만들어지지 않을 것이다. 대신 그룹 사용 카운터가 증가한다. IGMPv1에서 IP_DROP_MEMBERSHIP 명령은 데이터그램을 생성시키지 않는다.

호스트 회원 질의는 타입 0x11, 호스트 회원 보고는 각각 타입 0x12로 구분한다.

전체 호스트 그룹에는 보고하지 않는다. 이 그룹으로의 회원은 영구적이다.

7.3 IGMP 버전 2.

상기 내용에 대한 추가 사항 중 중요한 것 한가지는 그룹 이탈 메시지(타입0x17)가 포함되었다는 사실이다. 이 메시지는 서브넷에서 마지막 호스트가 그룹을 이탈하는 시점과 질의 시간이 만료되어 라우터가 더 이상 그룹에 남아있는 회원이 없다고 결정하는 시점 사이의 대역폭 낭비(이탈 지연-leave latency)를 줄이기 위해 추가되었다. 그룹 이탈 메시지는 그룹의 다른 멤버들에게는 불필요한 정보이기 때문에 남아있는 그룹보다 전체 라우터 그룹(224.0.0.2)쪽으로 전송해야 한다. (커널 버전 2.0.33 이하에서는 이 메시지를 그룹으로 보냈었다. 호스트로서는 해를 입을 일은 없지만, 쓸데없는 정보이기 때문에 이 정보를 처리하는 것은 시간낭비일 뿐이다.) 언제 이탈 메시지를 보낼 것인가, 언제 보내지 않을 것인가 하는 문제에 대해서는 몇 가지 미묘한 문제가 있다. 관심이 있다면 RFC를 참조할 것.

IGMPv2라우터가 이탈 메시지를 받으면 라우터는 그룹을 지정하여(group-specific) 남아있는 그룹에게 질의를 보낸다. 이것도 IGMPv2 에서 추가된 사항이다. IGMPv1에서 그룹 지정 질의는 없었다. 모든 질의가 전체 호스트 그룹으로 보내는 질의였다. IGMP 헤더에 정의된 타입 값은 변하지 않았지만(전과 동일하게 0x11), "그룹 주소(Group Address)"필드는 남아있는 멀티캐스트 그룹 주소로 채워진다.

IGMPv1에서 전송시 0으로 채우고 수신시 무시하기로 하였던 "Max Resp Time" 필드는 "Membership Query"에서만 의미가 있다. 이 필드는 보고해야할 한계 시간을 설정하는데 쓰인다. 성능을 조절하는 메커니즘으로 사용한다고 볼 수 있다.

IGMPv2에서 새로운 메시지 타입 0x16이 추가되었다. 이것은 "Version 2 Membership Report"로서 IGMPv2 호스트가 IGMPv2 라우터를 발견했을 때 전송한다. (IGMPv2 호스트는 "Max Response" 필드가 0 으로 채워진 패킷을 확인하여 IGMPv1 라우터가 존재한다는 것을 알아낸다).

동시에 여러 라우터가 질의를 할 때, IGMPv2 는 "discussions"를 회피하는 메커니즘을 제공한다. 이 경우 낮은 IP 주소를 가진 라우터가 질의자(querier)가 되고 다른 라우터는 타이머를 설정한다. 낮은 번호의 라우터가 어떤 이유로 제대로 작동하지 못하면 타이머가 만료된 후 다시 질의자를 설정한다.

7.4 커널 관련 사항

이 절에서는 리눅스 커널에서 멀티캐스트 구현을 학습하기 위한 실마리를 보여줄 것이다. 구현 자체를 설명하지는 않는다. 단지 어디서 찾을 것인지만 알려주다.

이 내용은 2.0.32에서의 설명이므로 이 글을 읽는 시점에서 좀 오래된 내용일지도 모른다. (네트웍 코드는 2.1.x에서 많이 변경되었다.

리눅스 커널에서 멀티캐스트 코드는 항상 #ifdef CONFIG_IP_MULTICAST / #endif 쌍으로 둘러싸여 있기 때문에 필요하다면 언제든지 커널에 포함(inclusion/exclusion)시키거나 배제시킬 수 있다. (이 포함/배제는 컴파일 시에 행해지며 #ifdef 구문은 선행처리기가 하는 작업이라는 것을 알고 있을 것이다. 포함여부 결정은make config, make menuconfig 또는 make xconfig를 실행시킬 때 할 수 있다.)

멀티캐스트 라우터 기능을 사용하고 싶다면 #ifdef CONFIG_IP_MROUTE / #endif쌍 속의 코드들을 활성화 시켜야 한다.

커널 소스는 보통 /usr/src/linux에 있다. 하지만 위치는 변할 수도 있는 것이기 때문에 커널 소스의 위치를 간단 명료하게 LINUX라고 가정하자. 이제 커널소스를 /usr/src/linux 에 풀었을 경우, LINUX/net/ipv4/udp.c 는 /usr/src/linux/net/ipv4/udp.c 를 의미한다.

사용자 프로그램에서의 멀티캐스트 인터페이스는 멀티캐스트 프로그래밍을 설명한 절에서 setsockopt()/ getsockopt()를 통해 모두 보여 주었다. 이 두 함수는 전달받은 변수의 유효성을 검사한 다음, 다른 몇 가지 함수를 호출하여 추가 적으로 검사하고, 또 다른 함수를 호출하는 식으로 구현되었다. (이 모든 함수 호출에 관심이 있다면 LINUX/net/socket.c (함수 sys_socketcall() 와 sys_setsockopt()), LINUX/net/ipv4/af_inet.c (함수 inet_setsockopt()) 그리고 LINUX/net/ipv4/ip_sockglue.c (함수 ip_setsockopt()) 를 참조하라.)

LINUX/net/ipv4/ip_sockglue.c도 주의 깊게 보아야 한다. 여기에는 함수 ip_setsockopt() 와 ip_getsockopt() 가 들어 있는데 대부분은 (어떤 에러 체크 후에) 가능한 optname을 검사하는 스위치같은 것이다. 유니캐스트 옵션과 함께, IP_MULTICAST_TTL, IP_MULTICAST_LOOP, IP_MULTICAST_IF, IP_ADD_MEMBERSHIP, IP_DROP_MEMBERSHIP같은 모든 멀티캐스트 옵션을 처리한다. 이 스위치 이전에 옵션을 검사하여 그것이 멀티캐스트 라우터 지정 옵션이라면 함수 ip_mroute_setsockopt() 와 ip_mroute_getsockopt() 로 라우팅 된다. (파일 LINUX/net/ipv4/ipmr.c 에 있음).

LINUX/net/ipv4/af_inet.c 에서 이전 절에서 논의했던 소켓 생성 기본 값을 볼 수 있다. (loopback enabled, TTL=1, 함수 inet_create() 중에서)

```
#ifdef CONFIG_IP_MULTICAST
        sk->ip_mc_loop=1;
        sk->ip_mc_ttl=1;
        *sk->ip_mc_name=0;
        sk->ip_mc_list=NULL;
#endif
```

다음 코드가 ”소켓을 닫으면 커널은 이 소켓에 있던 모든 회원을 탈퇴시킨다”는 내용을 단정적으로 보여준다. (위와 동일 파일의 함수 inet_release()에서)

```
#ifdef CONFIG_IP_MULTICAST
                /* Applications forget to leave groups before exiting */
                ip_mc_drop_socket(sk);
#endif
```

링크 계층의(Link Layer)의 장치 독립적인(Device independent) 연산들은 LINUX/net/core/dev_mcast.c 에 있다.

아직 두 가지 중요한 함수들을 설명하지 않았다. 입력 처리 함수와 멀티캐스트 데이터그램 출력 함수이다. 다른 데이터그램들과 마찬가지로 입력 데이터그램은 장치 드라이버로부터 ip_rcv() 함수(LINUX/net/ipv4/ip_input.c)로 전달된다. 이 함수에서 하위 계층으로부터 전해진 멀티캐스트 패킷을 완벽하게 필터링한다(하위레이어에서 최대한 필터링하지만, IP 계층에서 우리가 관심을 가지고 있는 그룹에 대해 100% 알고 있기 때문에 패킷을 완전하게 필터링한다고 했던것을 기억하자.) 호스트가 멀티캐스트 라우터로 작동중이라면 역시 이 함수가 패킷 포워딩 여부를 결정하고 적절하게 ipmr_forward()를 호출한다. (ipmr_forward() 은 LINUX/net/ipv4/ipmr.c에 구현되어 있다.).

패킷 출력 임무를 맡은 코드는 LINUX/net/ipv4/ip_output.c에 담겨져 있다. 이곳에서 패킷을 루프백 할지 안 할지(안한다면 ip_queue_xmit()호출) 검사하기 때문에IP_MULTICAST_LOOP 옵션이 효력을 발휘하는 곳이다. 또한, 외부로 향하는 패킷이 멀티캐스트인지 유니캐스트인지에 의거하여 TTL을 정하는 곳이기도 하다. 전자의 겨우 IP_MULTICAST_TTL 로 전달된 변수를 사용한다(함수 ip_build_xmit()).

mrouted(커널에 멀티캐스트 데이터그램 라우팅 방법을 알려주는 프로그램)로 작업하는 동안, 우리는 멀티캐스트 라우터로 작동하는 우리의 리눅스 박스에서 만들어진 패킷을 제외한, 로컬 네트웍에서 생성된 멀티캐스트 데이터그램은 모두 적절하게 라우팅되는 것을 발견했다. ip_input.c 는 제대로 작동하고 있는것 같았지만 ip_output.c 는 그렇지 않았다. 출력 프로그램의 소스 코드를 읽는 동안 외부로 나가는 데이터그램들이 라우팅 여부를 결정하는 함수 ipmr_forward()를 거치지 않는 것을 발견했다. 패킷들은 로컬 네트웍으로 출력되었지만 네트웍 카드는 자신이 전송하는 데이터그램을 읽을 수 없었고, 이런 데이터그램들은 라우

팅되지 않았다. 우리가 ip_build_xmit() 함수에 적절한 코드를 추가하자 모든 것이 올바로 작동하였다. (소스 수정은 사치를 떠는 것도 유식한척하는 것도 아니다; 필요다!)

ipmr_forward() 는 수 차례 언급했었다. ipmr_forward() 는 우리가 흔히 잘못 이해하고 있는 것을 풀어줄 수 있기 때문에 아주 중요한 함수이다. 멀티캐스트 트래픽을 라우팅할 때, 패킷 사본을 만들고 그것을 올바른 수신자에게 전달하는 것은 mrouted자체가 아니다. mrouted는 모든 멀티캐스트 트래픽을 수신하여, 그 정보에 근거하여 , 멀티캐스트 라우팅 테이블을 계산해 내고 커널에 라우팅 방법을 알려준다. "이 인터페이스에서 들어오는 저 그룹의 데이터그램은 저 인터페이스로 보내라."

이 라우팅 정보는 mrouted 데몬(raw 소켓 생성시 지어된 프로토콜은 IPPROTO_IGMP이어야 한다.)이 만든 raw 소켓에서 setsockopt()를 호출하여 커널에 설정한다. 이 옵션은 LINUX/net/ipv4/ipmr.c 의 ip_mroute_setsockopt()함수에서 처리한다. 소켓에서 발행한 첫번째 옵션(would be better to call them commands rather than options)은 MRT_INIT이어야 한다. 첫 번째 발행한 옵션이 MRT_INIT이 아니라면 다른 모든 명령은 무시한다. (-EACCES를 반납) 한 호스트 상에는 mrouted의 인스턴스가 꼭 하나만 존재할 수 있다. 이것을 추적하기 위해 첫 번째 MRT_INIT 을 받았을 때 구조체 sock* mroute_socket이 MRT_INIT을 수신한 소켓을 가리키게 된다. 만일 MRT_INIT 을 발행했을때 mroute_socket이 널(null)값을 가지지 않는다면 이미 다른 mrouted가 작동중임을 의미하므로 -EADDRINUSE 을 반납한다. 나머지 다른 명령들(MRT_DONE, MRT_ADD_VIF, MRT_DEL_VIF, MRT_ADD_MFC, MRT_DEL_MFC and MRT_ASSERT)은 mroute_socket과 다른 소켓에서 올 경우 -EACCES 를 반납한다.

라우팅된 멀티캐스트 데이터그램은 물리적 인터페이스 혹은 (가상적인) 터널을 통해 전송하기 때문에, 일반적인 추상화(abstraction) 단계를 VIFs, 즉, 가상 인터페이스 (Virtual InterFaces)고안해냈다. mrouted는 물리 혹은 터널 인터페이스를 가리키는 vif 구조체를 라우팅 테이블에 추가하기 위하여 커널에 넘겨주고, 데이터그램을 어디로 포워딩할지 알려주는 멀티캐스트 포워딩 목록(multicast forwarding entries)에도 알려준다.

VIFs 는 MRT_ADD_VIF 로 추가하고 MRT_DEL_VIF 로 삭제한다.두 함수 모두 vifctl 구조체를 커널로 넘긴다. (/usr/include/linux/mroute.h) 에서 다음과 같이 정의하고 있다.

```
struct vifctl {
        vifi_t  vifc_vifi;              /* Index of VIF */
        unsigned char vifc_flags;       /* VIFF_ flags */
        unsigned char vifc_threshold;   /* ttl limit */
        unsigned int vifc_rate_limit;   /* Rate limiter values (NI) */
        struct in_addr vifc_lcl_addr;   /* Our address */
        struct in_addr vifc_rmt_addr;   /* IPIP tunnel addr */
};
```

이 정보를 통해 vif_device 구조체를 만든다.

```
struct vif_device
{
        struct device   *dev;                   /* Device we are using */
        struct route    *rt_cache;              /* Tunnel route cache */
        unsigned long   bytes_in,bytes_out;
        unsigned long   pkt_in,pkt_out;         /* Statistics */
        unsigned long   rate_limit;             /* Traffic shaping (NI) */
        unsigned char   threshold;              /* TTL threshold */
        unsigned short  flags;                  /* Control flags */
        unsigned long   local,remote;           /* Addresses(remote for tunnels)*/
};
```

이 구조체에서 dev 항목을 주목하라. device 구조체는 /usr/include/linux/netdevice.h 에서 정의하고 있다. 아주 큰 구조체이지만 우리는 다음 필드에만 관심을 두면 된다.

struct ip_mc_list* ip_mc_list; /* IP multicast filter chain */

ip_mc_list 구조체는 /usr/include/linux/igmp.h에서 다음과 같이 정의하고 있다. :

```
struct ip_mc_list
{
        struct device *interface;
        unsigned long multiaddr;
        struct ip_mc_list *next;
        struct timer_list timer;
        short tm_running;
        short reporter;
        int users;
};
```

dev 구조체의 ip_mc_list 는 ip_mc_list 형 구조체의 연결리스트를 가리키는 포인터이다. ip_mc_list 는 해당 네트웍 인터페이스가 멤버로 참여하고 있는 멀티캐스트 그룹을 담고있다. 여기서 다시 한번, 회원은 인터페이스와 관계가 있음을 알 수 있다. LINUX/net/ipv4/ip_input.c 는 데이터그램이 (데이터그램을 수신한 인터페이스가 속하는 그룹 가운데) 어떤 그룹에 속한 것인지 결정하기 위해 이 연결리스트를 순회한다.

```
#ifdef CONFIG_IP_MULTICAST
                if(!(dev->flags&IFF_ALLMULTI) && brd==IS_MULTICAfST
                   && iph->daddr!=IGMP_ALL_HOSTS
                   && !(dev->flags&IFF_LOOPBACK))
                {
                        /*
                         *      Check it is for one of our groups
                         */
                        struct ip_mc_list *ip_mc=dev->ip_mc_list;
                        do
                        {
                                if(ip_mc==NULL)
                                {
                                        kfree_skb(skb, FREE_WRITE);
                                        return 0;
                                }
                                if(ip_mc->multiaddr==iph->daddr)
                                        break;
                                ip_mc=ip_mc->next;
                        }
                        while(1);
                }
#endif
```

ip_mc_list구조체에서 users 필드는 "IGMP 버전 1"에서 설명했던 것을 구현하기 위해 사용한다. 프로세스가 그룹에 참여 할때, 인터페이스가 이미 그 그룹의 회원이라면 (다시 말해서, 동일 인터페이스로 동일 그룹에 참여한 프로세스가 이미 있으면) 멤버 수(users)만 증가시킨다. IGMP 메시지를 보내지 않음을, 다음 코드에서 볼 수 있다. (ip_mc_inc_group() 에서 가져왔으며, ip_mc_join_group() 와 LINUX/net/ipv4/igmp.c에서 호출한다.)

```
        for(i=dev->ip_mc_list;i!=NULL;i=i->next)
        {
                if(i->multiaddr==addr)
                {
                        i->users++;
                        return;
                }
        }
```

멤버가 탈퇴할 때 카운터를 감소하며, 카운터가 0에 다다랐을 때에만 부가적인 작업을 수행한다. (ip_mc_dec_group()).

MRT_ADD_MFC 와 MRT_DEL_MFC 는 멀티캐스트 라우팅 테이블에서 포워딩 항목을 성정하거나 삭제한다. 둘 다 mfcctl 구조체를 다음 정보와 함께 커널로 넘긴다. (/usr/include/linux/mroute.h 에서 정의)

```
struct mfcctl
{
        struct in_addr mfcc_origin;             /* Origin of mcast      */
        struct in_addr mfcc_mcastgrp;           /* Group in question    */
        vifi_t  mfcc_parent;                    /* Where it arrived     */
        unsigned char mfcc_ttls[MAXVIFS];       /* Where it is going    */
};
```

이상의 정보를 가지고, ipmr_forward() 는 VIFs를 "돌아다닌다". 그리고 일치하는 포워딩 항목을 발견하면 데이터그램을 복사하여 ipmr_queue_xmit()을 호출한다. 그러면, ipmr_queue_xmit() 은 패킷을 터널을 통해 보낼 경우 라우팅 테이블에 명시 된 출력장치와 목적지 주소를 이용해 전송한다. (즉, 반대편 터널 종단의 유니캐스트 주소) 함수 ip_rt_event() 는(출력과 직접적인 관련은 없지만 ip_output.c 에 들어 있다.) 장치 활성화 메시지 같은 네트웍 장치 관련된 이벤트(event)를 받는다. 이 함수는 장치가 전체 호스트 그룹에 참여하는 것을 보증한다.

IGMP 함수들은 LINUX/net/ipv4/igmp.c 에서 구현한다. 이 함수들에 관한 중요한 정보는 /usr/include/linux/igmp.h 와 /usr/include/linux/mroute.h 에 있다. /proc/net 의 IGMP 관련 항목은 LINUX/net/ipv4/ip_output.c에 있는 ip_init() 에서 만든다.

제 8 절 라우팅 정책과 포워딩 기술

별로 좋지 않은 알고리즘은 멀티캐스트 데이터 수신 의사와 상관없이 전 세계로 트래픽을 내보낸다. 이런 최적화 되지 않은 알고리즘을 대신할 몇 가지 포워딩 기술과 라우팅 알고리즘이 개발되었다.

거리 벡터 멀티캐스트 라우팅 프로토콜 DVMRP (Distance Vector Multicast Routing Protocol) 는 아마도 오늘날 멀티캐스트 라우터 대부분이 사용하고 있는 알고리즘일 것이다. 이것은 고밀도형(dense mode) 라우팅 프로토콜, 즉, 높은 대역폭에 멤버들이 고밀도로 분산되어있는 있는 환경에서 잘 작동한다. 하지만, 멤버들이 낮은 밀도로 흩어져있는 경우 문제(scalability problem)가 있다.

DVMRP 같은 고밀도형 라우팅 프로토콜로 멀티캐스트 확장형 최단거리 우선 프로토콜 MOSPF (Multicast Extensions to OSPF -Open Shortest Path First-) , 프로토콜 독립형 고집적 멀티캐스트 PIM-DM (Protocol-Independent Multicast Dense Mode) 그리고 중심 기반 트리 CBT (Core Based Trees) 가 있다.

OSPF 버전 2 는 RFC 1583 , MOSPF 는 RFC 1584 , PIM-SM 과 CBT 는 각각 RFC 2117 과 2201 이 규격을 명시하고 있다.

이 모든 라우팅 프로토콜들은 flooding, Reverse Path Broadcasting (RPB), Truncated Reverse Path Broadcasting (TRPB), Reverse Path Multicasting (RPM) or Shared Trees 같은 멀티캐스트 포워딩 기술을 사용한다.

내용이 너무 길어지므로 이 모든 것들을 여기서 설명하지는 않겠다. 이것들에 대한 해설은 쉽게 찾아볼 수 있을 것이다. draft-ietf-mboned-in.txt 문서를 읽어 볼 것을 권한다. 위의 RFC 들이 있는 곳에 같이 있을 것이며, 상기한 기술과 정책들에 대한 약간 자세한 내용을 담고있다.

제 9 절 멀티캐스트 트랜스포트 프로토콜

지금까지 UDP를 이용한 멀티캐스트 전송에 대해 알아보았다. TCP에서는 불가능하기 때문에 이상이 일반적인 내용이다. 그렇지만, 좀더 새로운 멀티캐스트 트랜스포트 프로토콜 (Multicast Transport Protocols)을 만들어내기 위해 지난 몇 년간 집중적인 연구가 계속되고 있다.

새로 개발된 프로토콜 가운데 몇 가지는 구현 단계를 거쳐 테스트 단계까지 이르고 있다. 모든 형태의 응용에 적합한 일반적인 멀티캐스트 트랜스포트 프로토콜은 존재하지 않는다는 것이, 연구를 통해 밝혀지는 것 같다.

아주 복잡하고 조정하기 어려운 프로토콜에서 다음과 같은 문제들을 생각해 보자. 지연(멀티미디어 회의에서), 자료 손실, 패킷 정렬(ordering), 재전송, 흐름 및 혼잡도 조정, 그룹 관리 등.....게다가 수신자가 하나가 아니라 수백 수천 개의 저밀도 분산 호스트들이라면...이런 경우에 조정에 큰 곤란을 겪을 것이다. 따라서, 수신한 모든 패킷에 대해 수신 확인(ACKs)을 보내는 대신, 수신하지 못한 패킷에 대해서만 수신 불량 신호(NACKs)를 보내는 식의 알고리즘을 개발한다. RFC 1458 이 멀티캐스트 프로토콜 요구사항을 제안하고 있다.

멀티캐스트 프로토콜에 대한 자세한 해설은 문서의 범위를 넘기 때문에 더 이상 설명하지 않을 것이지만, 대신, 프로토콜 이름과 자료를 찾아 볼 수 있는 요점만 짚고 넘어가도록 하겠다. Real-Time Transport Protocol (RTP) 은 분산형 멀티미디어 회의(multi-partite multimedia conferences)와 관련 있으며, Scalable Reliable Multicast (SRM) 은 wb (the distributed White-Board tool, see section “Multicast applications”)가 사용한다. Uniform Reliable Group Communication Protocol (URGC) 은 중앙 집중식 제어에서 자료 처리(transaction) 가 신뢰적이며 정연히 이루어질 수 있도록 도와준다. Muse 는 MBone 에서 뉴스를 전송할 수 있도록 특정 프로그램용으로 고안된 것이며, Multicast File Transfer Protocol (MFTP) 은 문자 그대로 파일 전송과정 회의에 참여하듯이 참여 할수 있도록 해주는 프로토콜이다. Log-Based Receiver-reliable Multicast (LBRM) 은 기록 서버(logging server) 로 보내지는 모든 패킷을 추적하여 전송측에 데이터를 다시 전송해야하는지 아니면 이상 없이 수신되었는지 알려주는 특이한 프로토콜이다. 또 한가지, 멀티캐스트 프로토콜이면서 STORM (STructure-Oriented Resilient Multicast)이라는 재미있는 이름을 가진 프로토콜도 있다. 웹을 찾아보면 이 외에다 많은 멀티캐스트 프로토콜이 있으며 멀티캐스트를 이용해 새로운 일들을 벌이려는 재미있는 내용의 글들도 많다.(예를 들어, 멀티캐스트를 이용한 웹 페이지 자동 전송 같은 것)

멀티캐스트 프로토콜간의 신뢰성비교는 `<http://www.tascnets.com/mist/doc/mcpCompare.html>`을 참고하라.

멀티캐스트와 관련한 흥미로운 링크들과 최신 정보(인터넷 초안, RFC, 논문, 링크)가 가득한 아주 좋은 사이트다. `<http://research.ivv.nasa.gov/RMP/links.html>`

여기도 좋은 정보가 많다. `<http://hill.lut.ac.uk/DS-Archive/MTP.html>`

Katia Obraczka의 "Multicast Transport Protocols: A Survey and Taxonomy" 라는 글은 각 프로토콜에 대한 간단한 설명을 하고 기능상 차이점으로 분류하고 있다. 이 글은 IEEE Communications magazine, January 1998, vol. 36, No. 1. 에도 실려있다.

제 10 절 참고 자료

10.1 RFC 문서

- RFC 1112 "Host Extensions for IP Multicasting". Steve Deering. August 1989.
- RFC 2236 "Internet Group Management Protocol, version 2". W.Fenner. November 1997.
- RFC 1458 "Requirements for Multicast Protocols". Braudes, R and Zabele, S. May 1993.
- RFC 1469 "IP Multicast over Token-Ring Local Area Networks". T. Pusateri. June 1993.
- RFC 1390 "Transmission of IP and ARP over FDDI Networks". D. Katz. January 1993.
- RFC 1583 "OSPF Version 2". John Moy. March 1994.
- RFC 1584 "Multicast Extensions to OSPF". John Moy. March 1994.
- RFC 1585 "MOSPF: Analysis and Experience". John Moy. March 1994.
- RFC 1812 "Requirements for IP version 4 Routers". Fred Baker, Editor. June 1995
- RFC 2117 "Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification". D. Estrin, D. Farinacci, A. Helmy, D. Thaler; S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. July 1997.
- RFC 2189 "Core Based Trees (CBT version 2) Multicast Routing". A. Ballardie. September 1997. RFC 2201 "Core Based Trees (CBT) Multicast Routing Architecture". A. Ballardie. September 1997.

10.2 인터넷 초안(draft)

- "Introduction to IP Multicast Routing". draft-ietf-mboned-intro- multicast- 03.txt. T. Maufer, C. Semeria. July 1997.
- "Administratively Scoped IP Multicast". draft-ietf-mboned-admin-ip-space-03.txt. D. Meyer. June 10, 1997.

10.3 웹 페이지

- Linux Multicast Homepage. `<http://www.cs.virginia.edu/~mke2e/multicast.html>`
- Linux Multicast FAQ. `<http://andrew.triumf.ca/pub/linux/multicast-FAQ>`
- Multicast and MBONE on Linux. `<http://www.teksouth.com/linux/multicast/>`
- Christian Daudt's MBONE-Linux Page. `<http://www.microplex.com/~csd/linux/mbone.html>`
- Reliable Multicast Links `<http://research.ivv.nasa.gov/RMP/links.html>`
- Multicast Transport Protocols `<http://hill.lut.ac.uk/DS-Archive/MTP.html>`

10.4 참고 서적

- "TCP/IP Illustrated: Volume 1 The Protocols". Stevens, W. Richard. Addison Wesley Publishing Company, Reading MA, 1994
- "TCP/IP Illustrated: Volume 2, The Implementation". Wright, Gary and W. Richard Stevens. Addison Wesley Publishing Company, Reading MA, 1995
- "UNIX Network Programming Volume 1. Networking APIs: Sockets and XTI". Stevens, W. Richard. Second Edition, Prentice Hall, Inc. 1998.
- "Internetworking with TCP/IP Volume 1 Principles, Protocols, and Architecture". Comer, Douglas E. Second Edition, Prentice Hall, Inc. Englewood Cliffs, New Jersey, 1991

제 11 절 저작권과 책임 한계 고지

제 12 절 감사의 글

이곳은 내가 감사드릴 사람을 밝힐 좋은 기회인것 같다. 이 장의 내용이 다소 커질것 같다..... 어쨌든 이곳은-적어도 내게있어서-이 문서에서 가장 중요한 부분이다.

우선, Elena Apolinario Fernandez de Sousa (Elena가 이름,나머지가 성이다 ;-))에게 감사드리고 싶다. 우리 부서를 MBone과 연결하는 과정, 그리고 우리가 멀티캐스트 터널용으로 제작한 CSCV 소프트웨어를 디버깅하는 과정에서 그녀와 함께 일하며 얻은 모든 지식을 이 문서에 반영하려고 노력하였다. 그녀는 네트웍 문제를 찾아내 해결하고 커널의 버그를 수정하는데 많은 도움을 주었으며, 계속되는 버그와의 싸움에서도 유머감각을 잃지 않았다. 또, 그녀는 이문서의 초안을 읽고 수정해 주었으며 중요한 제안을 해 주었다. 이 문서가 사람들에게 도움이 되었다면 상당부분 그녀 덕분일 것이다. 고마워, Elena!

얼마 길지 않은 인생이지만 내 생애에서 정말 운 좋다고 생각하는 것은, 자기 시간을 쪼개서 이런 새로운 것을 배우는 남을 돕고 그 자체를 좋아하는 사람들을 만났다는 것이다. 그분들은 자신이 이미 아는 것을 설명해주었으며 그 뿐만 아니라 책을 빌려주고 자료와 설비를 공유해 주었다. 나는 이런 분들을 많이 알고 있으며 그분들에게 감사의 뜻을 전하고자 한다.

Pablo Basterrechea는 내가 인터넷 초보자일때 많은 문서들을 빌려주었다. 그의 책으로 어셈블리를 공부하고 구조적 프로그래밍으로 나아 갈 수 있었다. 고맙네, Pablo.

내가 대학 신입생때 Pepe Mas씨가 많은 도움을 주었다. 그는 컴퓨터 프로그래밍을 가르치는 분이었는데, 나는 곧 그분의 서가에 홀려버렸다. 내가 책을 제때에 제대로 돌려줄 것이라는 보증이 없었는데도 이름

조차 물어보지 않고 조건없이 책을 빌려주었다. 그 분의 책 즉, Comer의 "Internetworking with TCP/IP, Volume 1"을 여름 내내 보며 처음으로 TCP/IP를 접했다. 이 책은 내게 많은 영향을 미쳤으며 그 여름 이후로 TCP/IP는 주 분야가 되었다.

내가 가장 감사드려야 할 분이 있다면, 알파벳 순서로 ;-), Jos Manuel 과 Paco Moya 이 두명이다. 이 두 명에게 나는 궁금한 것을 가장 많이 물었다.(C, C++, Linux, security, Web, OSs, signals & systems, electronics, ... 그외의 것들 모두!) 나의 계속되는 질문에도 불구하고 언제나 친절하게 알려주었다.내가 GNU/Linux를 쓰는 것도 그들 덕이며 정말로 감사드리는 분들이다.

Igo Mascaraque 역시 초보 시절 나를 도와 주고 포기하지 않도록 용기를 북돋아 주었다.

제일 중요한 부모님께 감사드리고 싶다. 언제나 내게 좋은 기회를 주셨다. 정말로 감사드린다.

내가 처음 시스템 관리를 독학 할때 root 패스워드를 내게 줄 정도로 나를 믿어준 Joaqu Seoane에게도 감사드린다. 또, 역시 이곳 DIT에서 내게 좋은 기회를 만들어준 Santiago Pav에게도 감사드린다.

W. Richard Stevens씨의 책은 내게 정말 계시와도 같은 책이었다.(비싼게 흠이긴 하지만) 그 분이 혹시라도 이 글을 보신다면 정말로 감사드리고 싶다. 또, 계속 집필해 주시기를 부탁드린다. 우리에게 많은 도움이 될 것이다.

끝으로 Richard Stallman, Linus Torvalds, Alan Cox, 그 외에, 우리에게 이런 훌륭한 운영체제를 안겨준 Linux 커널 및 free software 개발자들에게 감사드린다.

누군가 잊은것 같긴 하지만 ..죄송...내가 그 분들에게 감사해하고 있다는 것을 알고 계시리라 확신한다.

제 13 절 역자 한마디

보기보다 만만치 않군요. 특히 용어 선택이 어려웠습니다. 잘못된 부분이 있거나 다른 하고 싶은 말씀이 있다면 메일 부탁드립니다. 조속한 시일 내에 개정하도록 노력하겠습니다. 도움이 되었기를 바라면서....