Linux Networking-HOWTO (Previously the Net-3 Howto)

현 저자: {Poet} poet@linuxports.com
v1.5, August 1999

역자: 정하녕 alita@kldp.org
2000년 2월

원 저자: Terry Dawson (주 저자), VK2KTJ; Alessandro Rubini (관리자)

목 록

1 소개.

이것은 LinuxPorts가 이 문서의 저자가 된 이후 첫 번째 릴리즈이다. 우선 나는 앞으로 몇 달 동안 여러분이 이 문서가 유용하다는 것을 발견하고 우리가 리눅스의 네트워킹 이슈들에 관한 정확하고 신속한 정보를 재공할 수 있기를 바란다는 것을 말하고 싶다.

이 문서는 곧 Net-3(4) Howto가 아닌 Networking-HOWTO가 될 것이다. 우리는 PPP, VPN 등의 아이템들도 포함할 것이다.

2 문서 역사

최초의 NET-FAQ는 Linux Documentation Project가 공식적으로 시작된기 전에 Linux의 네트워킹에 관하여 흔히 질문되는 것들에 대한 답변으로 Matt Welsh와 Terry Dawson에 의해 쓰여졌다. 그 문서는 리눅스 네트워킹 커널의 매우 초기 버전을 다루고 있었다. NET-2-HOWTO는 NET-FAQ를 이어받았으며 최초의 LDP HOWTO 중 하나였고 후에 버전 3이라 불린 리눅스 네트워킹 소프트웨어의 버전 2를 다루고 있었다. 이 문서는 차례로 그 문서를 이어받았으며 오직 버전 4의 리눅스 네트워킹 커널, 특히 커널 2.x와 2.2.x와 관련이 있다.

이 문서의 이전 버전들은 그 안에 들어있는 막대한 많은 양의 내용들 때문에 크기가 매우 커졌다. 이 문제를 해결하는 것을 돕기 위해 특정 네트워킹 주제를 다루는 많은 HOWTO들이 만들어 졌다. 이 문서는 알맞은 곳에서 그 문서들로의 포인터를 제공하고 아직 다른 문서들에 의해 다뤄지지 않는 분야를 다루게 될 것이다.

2.1 피드백 `poet@linuxports.com`.

우린 항상 피드백을 반기고 있다. `poet@linuxports.com` 을 통해 우리에게 연락을 주기 바란다.

또한, 잘못된 내용이나 추가되었으면 하는 것들이 있을 경우에도 연락주기 바란다.

(역자주: 역시나, 오역이나 어색한 부분에 대해서는 역자에게 alita@kldp.org 을 통해 연락주기 바랍니다. 후반부의 잘 모르는 부분의 번역이 상당히 이상하거나 잘못되었을 수 있습니다. 기타 어떠한 comment도 환영합니다.)

3 이 HOWTO를 사용하는 방법

이 문서는 하향식으로 짜여져 있다. 첫 부분은 정보 제공 수준의 내용을 담고 있으며 원치 않으면 읽지 않아도 된다. 다음으로 네트워킹 문제에 관란 일반적인 문제들을 다루고 있으며 좀 더 상세한 부분으로 들어가기 위해서 반드시 이해를 해야 한다. 나머지 특정 기술과 관련된 내용들은 이더넷과 IP 관련 내용, 일반적인 PC 하드웨어에 적합한 기술들, 흔히 쓰 이지 않는 기술들 이렇게 세 개의 주 섹션으로 구분되어 있다.

이 문서를 읽는데 다음의 순서를 추천한다.

일반적인 내용이 있는 부분을 읽어라

이 부분은 뒤에 나올 거의 모든 기술들에 적용되며 이 기술들을 이해하기 위해서 매우 중요하다. 그러나 아마 많은 독자들은 이미 이 내용에 익숙할 것이다.

자신의 네트웍을 고려하라

자신의 네트웍이 현재 어떻게 구성되어 있는지 혹은 앞으로 어떻게 구성될 것이며 어떤 하드웨어와 기술을 사용할 것인지 정확히 알아야 한다.

만약 LAN이나 인터넷에 직접 연결된다면 "이더넷과 IP" 부분 을 읽어라.

이 섹션은 기본적인 이더넷 설정법과 Linux가 제공하는 파이어월이나 라우팅 같은 다양한 IP 네트웍 관련 특징들을 설명하고 있다.

저가의 로칼 네트웍이나 전화접속 연결에 관심이 있다면 그 다음 부분 을 읽어라.

이 부분은 PLIP와 PPP, SLIP, ISDN 등의 개인용 워크스테이션 에서 널리 사용되는 기술들을 설명하고 있다.

자신의 요구 사항과 관련된 특정 기술을 다루는 부분을 읽어라

만약 자신이 필요한 것이 IP나 일반적인 하드웨어와 다른 것이라면 마지막 부분이 non-IP 프로토콜과 특정 통신 하드웨어에 대해 자세히 설명해줄 것이다.

설정 작업을 하여라

직접 네트웍 환경 설정을 해 봐야 하며 발생하는 문제들을 신중히 검사하여야 한다.

필요하다면 다른 도움을 찾아보아라

만약 이 문서가 도움을 주지 못할 문제가 발생한다면 도움을 찾는 법과 버그를 보고하는 법에 관한 부분을 읽어라.

즐겨라!

네트워킹은 재미있다. 그걸 즐겨라.

3.1 이 문서에서 사용된 관행

어떤 특정 관행도 여기선 사용되지 않았으나 명령문이 표시되는 방식에 주의해야 한다. 전통적인 유닉스 문서화에 따라 쉘에서 입력해야 하는 모든 명령문 앞에는 프롬프트가 나온다. 이 HOWTO는 "user%"를 슈퍼유저 권한이 필요하지 않는 명령문의 프롬프트로, "root#"를 루트로 실행하는 명령문의 프롬프트로 사용한다. 주석을 나타낼 때 "#" 표시를 쓰는 쉘 스크립트를 인용한 부분과의 혼란을 막기 위해 "#" 대신 "root#"을 선택하였다.

“커널 컴파일 옵션들”을 나타낼 땐 *menuconfig*에 의해 사용되는 형식으로 표시된다. (저자 처럼) *menuconfig*에 익숙치 않다 하더라도 이해할 수 있을 것이다. 만약 옵션들의 nesting에 의문점이 있다면 한번 프로그램을 실행시켜 보라.

이 문서의 html 버전을 사용한다면 다른 HOWTO들 처럼 이 문서의 복사본 을 가지고 있는 것이 매우 도움이 될 것이다. 만약 문서 전체를 가지고 있지 않다면, 모든 HOWTO는 `metalab.unc.edu(/pub/Linux/HOWTO` 디렉토리)와 셀 수 없이 많은 미러 사이트에서 구할 수 있다.

4 리눅스 네트워킹에 대한 일반적인 내용

4.1 리눅스 네트워킹 커널 개발의 간략한 역사

기존의 것들 만큼 좋은 성능을 지닌 tcp/ip 프로토콜 스택을 새로운 커널에서 개발 하는 것은 쉽지 않은 일이었다. U.S.L의 판례로 인해 제한적인 저작권으로 기존의 것이 제한을 받는지가 불확실했고 기존의 것보다 더 좋고 완전히 다르게 만들려는 열정이 있을 당시에 기존의 것을 포팅하지 않으려는 결정이 내려졌다.

커널 네트웍의 개발을 주도한 초기 자원자는 Ross Biro `<biro@yggdrasil.com>` 였다. Ross는 WD-8003 네트웍 인터페이스 카드용 드라이버 위에서 간단하고 불완전하지만 매우 유용한 루틴들을 구현하였다. 이것은 많은 사람들로 하여금 그 소프트웨어를 가지고 실험해 보도록 하기엔 충분했고 몇몇 사람들은 머신을 인터넷에 연결시키기 위해 이 설정을 사용하기도 하였다. 리눅스 개발 공동체 안에서 네트워킹 지원에 대한 압력이 발생하였고 결국 Ross에 대한 일부 부당한 압력과 그의 개인적인 책임들이 그가 이룩한 성과보다 우세해져서 그는 개발 리더 자리에서 물러나게 되었다. 그런 논쟁만을 좋아하는 환경에서 프로젝트가 시작되도록 하고 어떤 유용한 것을 만들어 내는 책임을 받아들인 Ross의 노력은 이후의 작업에 촉매 역할을 하였고 결국 지금의 성공에 근본적인 요소가 되었다.

Orest Zborowski `<obz@Kodak.com>` 은 최초로 리눅스 커널용 BSD 소켓 프로그래밍 인터페이스를 만들었다. 이것은 수많은 기존의 네트웍 프로그램들이 큰 수정 없이 리눅스로 포팅될 수 있도록 해줬다는 데에서 매우 큰 진전이었다.

이즈음 Laurence Culhane `<loz@holmes.demon.co.uk>` 은 SLIP 프로토콜을 지원하는 리눅스용 드라이버를 처음 개발하였다. 이것은 이더넷 네트워킹을 할 수 없었던 많은 사람들에게 새로운 네트워킹 소프트웨어를 실험할 수 있게 해주었다. 또한 어떤 사람들은 이 드라이버를 이용하여 인터넷 접속을 하는데 사용하기도 하였다. 이것은 많은 사람들에게 리눅스 네트워킹 지원이 완성되었을 때 실현될 가능성들을 맛보게 하였고 기존의 네트워킹 소프트웨어를 적극적으로 사용하고 실험하는 사용자 수를 늘어나게 했다.

네트워킹 지원을 만드는 일에 적극적으로 참여해 온 사람중 하나가 Fred van Kempen `<waltje@uwalt.nl.mugnet.org>` 이었다. Ross가 개발 리더 자리에서 물러난 후 얼마동안 Fred는 그의 시간과 노력을 들였고 기본적으로 반대하는 사람이 없는 역할을 받아들였다.Fred는 리눅스 네트워킹 소프트웨어를 이끌 방향에 대한 야심찬 계획을 가지고 있었고 그렇게 일을 진행하였다. Fred는 NET-2 커널 코드라 불리는 (Ross의 것을 NET이라 부른다) 일련의 네트워킹 코드를 만들었고 많은 사람들이 유용하게 이용 할 수 있었다. Fred는 다이나믹 디바이스 인터페이스, 아마추어 라디오 AX.25 프로토콜 지원, 더욱 모듈화된 네트워킹 구현 같은 많은 혁신적인 것들을 개발 계획에 집어넣었다. Fred의 NET-2 코드는 열광적인 수많은 사람들에 의해 사용되었고 그 소프트웨어가 돌아간다는 말이 퍼지면서 그 숫자는 더욱 늘어났다. 이 당시 네트워킹 소프트웨어는 표준 커널 코드의 수많은 패치에 불과했으며 보통의 릴리즈에는 포함되지 않았었다. NET-FAQ와 이어 나온 NET-2-HOWTO는 그 모든 것을 작동하도록 하기 위한 매우 복잡한 작업들에 대해 설명하고 있었다. Fred의 초점은 표준 네트워킹 구현에 대한 기술 혁신에 있었으며 이는 시간이 걸리는 작업이었다. 사용자들은 믿을만하게 동작하고 80%의 사용자를 만족시킬 수 있는 것을 조급해 했고 Ross에게처럼 개발 리더로서의 Fred에 대한 압력이 생겨났다.

Alan Cox `<iialan@www.uk.linux.org>` 는 이러한 상황을 해결하기 위한 해결책을 제시했다. 그는 Fred의 NET-2 코드를 가져다 디버깅하고 안정적으로 만들어서 조급한 사용자들을 만족시키는 동시에 Fred에 대한 압력을 덜어줘서 그에게 자신의 일을 계속할 수 있도록 해 주려 하였다. ‘Net-2D(debugged)’라 불리는 리눅스 네트워킹 코드의 첫번째 버전의 성공과 함께 이 일을 시작하였다. 그 코드는 많은 대표적인 환경 하에서 안정적

으로 작동하였고 사용자들은 만족하였다. 확실히 Alan은 그 프로젝트에 공헌할만한 아이디어와 기술을 가지고 있었고 NET-2 코드가 나아갈 방향에 대한 많은 논의들이 계속해서 발생하였다. 그 중 리눅스 네트워킹 사회 속에 서로 다른 두 파가 생겨났는데 한 쪽은 '우선 동작하도록 만든 후에 향상시키자'라는 철학을 가지고 있었고 다른 한 쪽은 '처음부터 잘 만들자'라는 철학을 가지고 있었다. 결국 Linus가 중재를 하였고 Alan의 개발 노력에 대한 지지를 표명하면서 표준 커널 소스 배포본에 Alan의 코드를 포함시켰다. 이것은 Fred를 난처한 입장에 놓이게 하였다. 개발이 계속되었음에도 그의 코드를 사용하고 시험해 볼 사용자층이 부족하였고 이는 개발의 진행을 더디고 어렵게 만들었다. 결국 Fred는 잠시 개발을 계속하다 중단했으며 Alan이 리눅스 네트워킹 커널 개발 노력의 새 리더가 되었다.

곧 Donald Becker <becker@cesdis.gsfc.nasa.gov> 가 네트워킹의 로우 레벨 분야에서의 그의 재능을 들어내었고 막대한 양의 이더넷 드라이버들을 만들어서 현재 커널에 포함된 거의 모든 드라이버들이 Donald에 의한 것일 정도이다. 많은 중요한 공헌을 한 다른 사람들이 있었지만 Donald의 업적은 그 양이 매우 방대하여 특별한 언급의 가치가 있다.

Alan은 'TODO' 리스트에 언급되지 않은 일들을 진행해 나가는 동시에 NET-2-Debugged 코드를 개량하는 일을 계속하였다. 리눅스 커널 1.3.* 소스가 모습을 들어낼 때 즈음에 커널 네트워킹 코드는 현재 버전의 기초가 되는 NET-3로 바뀌어 있었다. Alan은 네트워킹 코드의 많은 다른 방향에서 작업을 해 나갔고 리눅스 네트워킹 사회의 많은 다른 재능있는 사람들과 함께 코드를 모든 방향에서 향상시켜 갔다. Alan은 동적 네트워킹 장치를 만들었고 최초로 표준 AX.25와 IPX를 구현하였다. Alan은 그 코드를 계속해서 고치고 천천히 개조하면서 현재의 모습으로 발전시켜왔다.

PPP의 지원은 Michael Callahan <callahan@maths.ox.ac.uk> 과 Al Longyear <longyear@netcom.com> 에 의해 추가되었는데 이는 네트워킹을 위해 리눅스를 사용하는 사람들의 수를 증가시키는대 매우 결정적인 것이었다.

Jonathon Naylor <jsn@cs.nott.ac.uk> 는 Alan의 AX.25 코드를 개량하고 NetRom 과 Rose 프로토콜을 추가함으로써 많은 공헌을 해왔다. AX.25/NetRom/Rose의 지원은, 리눅스 외의 어떤 다른 운영체제도 이 프로토콜들에 대한 표준적인 지원을 하지 못하고 있다는 이유 때문에 그 자체만으로도 매우 중요하다.

물론 리눅스 네트워킹 소프트웨어의 개발에 중요한 공헌을 한 수많은 다른 사람들이 있어왔다. 이들 중 일부는 특정 기술에 관한 내용에서 나올 것이고 다른 사람들도 모듈이나 드라이버, 버그 수정, 제안, 테스트 리포팅 그리고 정신적 지원 등에 공헌하였다. 어느 경우에나 모두가 일정한 역할을 하였고 그들이 할 수 있는 것을 제공하였다. 리눅스 커널 네트워킹 코드는 무정부주의적인 리눅스 스타일로부터 나올 수 있는 결과의 훌륭한 한 예이며 만약 이가 아직 놀랍지 않다면 곧 충분히 놀랍게 될 것이다. 개발은 아직도 끝나지 않았다.

4.2 리눅스 네트워킹 관련 자료들.

리눅스 네트워킹에 관한 좋은 정보들을 찾을 수 있는 곳이 많이 있다.

많은 컨설턴트들이 있으며 그 목록은 LinuxPortsConsultantsDatabase에서 찾을 수 있다.

리눅스 커널 네트워킹 코드의 현 관리자인 Alan Cox는 리눅스 네트워킹의 지금 혹은 새롭게 개발되는 것들 중 주요한 것을 담고 있는 웹 페이지를 www.uk.linux.org에서 운영하고 있다.

다른 좋은 것은 Olaf Kirch가 쓴 Network Administrator's Guide.란 책이다. 이는 LinuxDocumentationProject 의 결과물이며 NetworkAdministratorsGuideHTMLversion에서 언제라도 읽을 수 있고 metalab.unc.eduLDPftparchive에서 다양한 양식의 것을 구할 수 있다. Olaf의 책은 매우 포괄적이며 리눅스 하에서의 네트워킹 구성에 대한 수준 높은 개관을 제공해 준다.

리눅스 뉴스 계층에는 네트워킹 관련 뉴스그룹도 있다.: comp.os.linux.networking

리눅스 네트워킹과 관련된 질문을 할 수 있는 메일링 리스트도 있다. 이에 가입하기 위해서는 다음과 같이 메일을 보내야 한다.

```
To: majordomo@vger.rutgers.edu
Subject: anything at all
Message:
```

```
subscribe linux-net
```

많은 IRC 네트워크 상에서는 리눅스 네트워킹 관련 질문을 할 수 있는 `#linux` 채널이 열린다.

어떤 문제를 보고할 때에는 가능한한 그 문제에 대해 자세히 기술하는 것을 상기하라. 특히, 커널이나 *pppd* 혹은 *dip* 등의 툴의 버전 같은 사용하는 소프트웨어의 버전과 겪고 있는 문제의 정확한 특성을 명시해야 한다. 이는 실행한 명령문과 나타난 에러메시지를 정확히 기술함을 의미한다.

4.3 리눅스에 국한되지 않은 네트워크 정보를 얻을 수 있는 곳.

만약 tcp/ip 네트워킹에 대한 일반적이고 기본적인 학습 정보를 원한다면 다음 문서들을 볼 것을 추천한다.

tcp/ip introduction

this document comes as both a `textversion` and a `postscriptversion`.

tcp/ip administration

this document comes as both a `textversion` and a `postscriptversion`.

만약 tcp/ip 네트워킹에 대한 보다 자세한 정보를 원한다면 다음을 강력히 추천한다:

> *Internetworking with TCP/IP, Volume 1: principles, protocols and architecture*, by Douglas E. Comer, ISBN 0-13-227836-7, Prentice Hall publications, Third Edition, 1995.

만약 유닉스 같은 환경에서 네트워킹 어플리케이션을 만드는 법을 배우고 싶다면 다음을 강력히 추천한다.:

> *Unix Network Programming*, by W. Richard Stevens, ISBN 0-13-949876-1, Prentice Hall publications, 1990.

이 책의 두번째 판을 서점에서 찾을 수 있으며 세 권으로 이루어져 있다. 더 자세히 알고 싶으면 `Prenice-Hall`를 방문해 보라.

또한 `comp.protocols.tcp-ip` 뉴스그룹을 이용할 수도 있다.

인터넷과 tcp/ip 프로토콜과 관련된 특정 기술 정보의 중요한 소스는 RFC들이다. RFC는 'Request For Comment'의 줄임말이며 인터넷 프로토콜 표준을 문서화하고 제안하는 표준적인 방법이다. 많은 RFC 저장소가 있으며 대부분은 ftp 사이트이고 일부는 특정 키워드로 RFC 데이타베이스를 검색할 수 있도록 검색 엔진이 달린 웹 억세스도 제공한다.

RFC의 소스중 하나는 `NexorRFCdatabase`이다.

5 일반적인 네트워크 설정 관련 정보

여러분은 아래에 나오는 부분들을 알아야 할 필요가 있으며 실제로 여러분의 네트워을 설정해보기 전에 이해해야 한다. 여러분이 사용할 네트워의 정확한 특성에 관계없이 작동하는 기본적인 원리들이다.

5.1 내가 시작하려는 것은 무엇인가 ?

네트웍을 설정하거나 꾸미기 전에 여러분은 몇가지 필요한 것이 있다. 그 중 가장 중요한 것들은 다음과 같다.

5.1.1 최신의 커널 소스(선택사항)

주의사항:

대부분의 최신 리눅스 배포판들은 네트워킹 지원이 가능한 상태로 나온다. 따라서 커널을 다시 컴파일할 필요는 없다. 만약 여러분이 3COM NIC나 NE200 NIC, 인텔 NIC 등의 잘 알려진 하드웨어를 사용한다면 아무 문제가 없을 것이다. 그러나 커널을 업데이트 해야 하는 상황에 놓여있다면 다음의 정보가 제공된다.

지금 돌리고 있는 커널이 여러분이 사용하고자 하는 네트웍 형식이나 카드에 대한 지원을 가지고 있지 않기 때문에 여러분은 알맞은 옵션으로 커널을 다시 컴파일하기 위해 커널 소스가 필요할 것이다.

Redhat이나 Caldera, Debian, Suse 같은 주요 배포본의 사용자들에겐 더이상 적용되지 않는다. 일반적으로 많이 사용하는 하드웨어를 사용한다면 매우 특별한 기능을 원치 않는 한 커널을 다시 컴파일할 필욘 없다.

여러분은 언제나 `ftp.cdrom.com`로부터 최신의 커널 소스를 얻을 수 있다. 이곳이 공식 사이트는 아니지만 훨씬 큰 bandwidth와 더 많은 동시사용자 수를 가지고 있다. 공식 사이트는 kernel.org 이지만 가능한한 위의 것을 이용하라. ftp.kernel.org 는 매우 붐빈다는 것을 상기하고 미러들을 이용하라.

보통 커널 소스는 `/usr/src/linux` 디렉토리에 풀려진다. 패치를 적용시키고 커널을 빌드하는 방법에 대한 정보를 얻기 위해선 `Kernel-HOWTO`를 읽어라. 커널 모듈 설정에 관한 정보를 위해선 "Modules mini-HOWTO"를 읽어라. 또한 커널 소스 안의 `README`와 `Documentation` 디렉토리도 용감한 독자들에겐 매우 유익할 것이다.

특별히 따로 언급되지 않는 한 나는 여러분들이 안전을 위해 안정 커널 릴리즈 (버젼 번호의 두 번째 숫자가 짝수인 것)을 고수할 것을 추천한다. 개발 커널 릴리즈(버전 번호의 두 번째 숫자가 홀수인 것)은 여러분의 시스템의 다른 소프트웨어들과 함께 작동하는 데 있어 문제를 일으킬지도 모르는 구조적 혹은 여타 변화된 사항을 가지고 있을 수 있다. 여러분이 앞으로 있을 지 모를 소프트웨어 에러를 포함해 이 문제들을 해결할 수 있다고 확신하지 않는 한 이를 사용하지 말아라.

5.1.2 최신 네트워크 툴들

네트워크 툴들은 리눅스 네트웍 장치들을 설정하는 데 사용하는 프로그램들이다. 예를 들어 이 툴들을 이용하여 장치들에 주소를 할당하고 라우팅 정보를 설정할 수 있다.

대부분의 현대 리눅스 배포본은 네트웍 툴과 함께 제공되므로 만약 배포본을 깐 후 아직 네트워크 툴들 인스톨하지 않았다면 인스톨 해야 한다.

만약 배포본으로부터 인스톨하지 않았다면 소스를 구해서 직접 툴들을 컴파일해야 한다. 이는 어렵지 않다.

네트웍 툴들은 현재 Bernd Eckenfels에 의해 관리되고 있으며 `ftp.inka.de`에서 구할 수 있고 `ftp.uk.linux.org`에서 미러링되고 있다.

또한 `net-tools-1.51-3.i386.rpm`에서 최신의 RedHat 패키지를 구할 수 있다.

여러분이 사용하려 하는 커널에 알맞는 버젼을 선택하도록 주의하고 설치하기 위해선 패키지 안의 지시를 따른다.

이 글을 쓰는 현재의 최신 버전을 설정하고 설치하기 위한 방법은 아래와 같다.

```
user% tar xvzf net-tools-1.33.tar.gz
user% cd net-tools-1.33
user% make config
user% make
root# make install
```

혹은 Redhat 패키지를 이용하기 위해선

```
root# rpm -U net-tools-1.51-3.i386.rpm
```

추가적으로 방화벽을 설정하거나 IP 마스커레이드 기능을 사용하길 원한다면 *ipfwadm* 프로그램이 필요하다. 최신 버전은 `ftp.xos.nl` 에서 구할 수 있다. 역시 많은 버젼이 존재하므로 여러분이 사용하는 커널에 가장 알맞는 것을 구하도록 한다. 리눅스의 방화벽 기능은 2.1의 개발동안 바뀌었고 커널 v2.2에서는 *ipchains*에 의해 계승되었다. *ipfwadm*은 커널 버젼 2.0에만 적용된다. 다음은 2.0 이하 버젼의 커널이 들어있는 배포본들이다.

```
Redhat 5.2 or below
Caldera pre version 2.2
Slackware pre version 4.x
Debian pre version 2.x
```

이 글을 쓰는 시점의 최신 버전을 설정하고 설치하기 위해선 `TheLinuxDocumentationProject` 에 있는 IPChains howto를 읽을 필요가 있다.

만약 커널 버전 2.2(혹은 2.1 이후)를 사용한다면 *ipfwadm* 은 방화벽 설정에 알맞은 툴이 아니다. 이 NET-3-HOWTO는 현재 새 방화벽 구성을 다루지 않고 있다. ipchain에 대한 자세한 정보를 원한다면 위의 문서를 참고하여라.

5.1.3 네트웍 응용 프로그램들

네트웍 응용 프로그램들은 *telnet*과 *ftp* 그리고 그에 관련된 서버 프로그램들이다. David Holland는 이것들의 가장 대표적인 배포본을 관리해 왔고 현재 `netbug@ftp.uk.linux.org`에서 관리되고 있다. 이 배포본은 `ftp.uk.linux.org`에서 구할 수 있다.

5.1.4 IP 주소의 설명

인터넷 프로토콜 주소(IP Address)는 네 바이트로 이루어져 있으며 편의상 'dotted decimal notation' - 직역하자면 점으로 구분되는 10진수 표기법 - 으로 표기된다. 이 형식에서 각 바이트는 0부터 255까지의 십진수로 바뀌며 각 바이트는 '.'으로 구분되어진다. 편의상 호스트나 라우터의 각각의 인터페이스는 IP 주소를 갖는다. 한 머신의 서로 다른 인터페이스들이 같은 IP 주소를 갖을 수도 있으나 보통은 각각의 인터페이스는 자신만의 주소를 갖는다.

인터넷 프로토콜 네트웍은 IP 주소들의 끊임없는 연속이다. 한 네트웍 안의 모든 주소들은 많은 숫자들을 공통적으로 가지고 있다. 그 네트웍 안의 모든 주소들 중에 공통적인 부분을 주소의 'network portion' 이라 부르고 나머지 숫자들을 'host portion' 이라 부른다. 한 네트웍 안의 모든 주소들에 의해 공유되는 bit 수를 넷마스크라 부르고 어느 주소가 해당 네트웍에 속해있는지 아닌지를 판단하는 것이 넷마스크의 역할이다. 예를 들어 아래를 보자.

```
-----------------  ---------------
Host Address       192.168.110.23
Network Mask       255.255.255.0
Network Portion    192.168.110.
Host portion                  .23
-----------------  ---------------
Network Address    192.168.110.0
Broadcast Address  192.168.110.255
-----------------  ---------------
```

해당 넷마스크와 'bitwise anded'된 모든 주소들은 자신이 속한 네트웍 주소를 나타낸다. 따라서 네트웍 주소는 항상 그 네트웍의 주소 범위에서 최소값을 갖는 주소이며 host portion은 항상 모두 0이다.

브로드캐스트 주소는 그 네트웍 상의 모든 호스트들이 자신의 주소외에도 들을 수 있는 특별한 주소다. 네트웍 상의 모든 호스트가 정보를 받도록 하고 싶을 때 그 정보를 이 주소로 보낸다. 라우팅 정보나 경고 메시지 같은 특정 종류의 데이타그램들은 브로드캐스팅 주소로 보내어져서 네트웍 상의 모든 호스트들이 동시에 받을 수 있도록 한다. 브로드캐스트 주소에 대해선 널리 사용되는 두 가지 표준이 있다. 가장 많이 쓰이는 것은 한 네트웍상에서 가능한 제일 큰 주소를 브로드캐스트 주소로 사용하는 것이다. 위의 예에선 192.168.110.255가 될 것이다. 몇몇 이유 때문에 다른 사이트들은 네트웍 주소를 브로드캐스트 주소로 사용하는 방법을 이용한다. 실제로 어느 것을 사용하는 지는 별 문제가 되지 않지만 네트웍 상의 모든 호스트들이 같은 브로드캐스트 주소를 사용하도록 주의하여야 한다.

관리상의 이유 때문에 IP 프로토콜의 발전 초기에 임의의 주소들의 그룹들이 네트웍을 형성하게 되었고 이 네트웍들은 클래스라 불리우는 것으로 구분되어졌다. 이 클래스들은 할당 가능한 네트웍의 표준적인 크기를 규정하고 있다. 할당된 범위는 아래와 다음과 같다.

Network Class	Netmask	Network Addresses
A	255.0.0.0	0.0.0.0 - 127.255.255.255
B	255.255.0.0	128.0.0.0 - 191.255.255.255
C	255.255.255.0	192.0.0.0 - 223.255.255.255
Multicast	240.0.0.0	224.0.0.0 - 239.255.255.255

여러분이 사용해야할 주소들은 여러분이 하려는 것에 따라 달라진다. 필요한 모든 주소들을 얻기 위해선 아래의 작업들을 수행해야 할 것이다.

이미 존재하는 IP 네트웍 상에 리눅스 머신을 설치하는 경우.

만약 여러분이 이미 존재하는 IP 네트웍 상에 리눅스 머신을 설치하고자 한다면 네트웍 관리자에게 가서 다음 정보들을 물어봐야 한다.

- 호스트의 IP 주소
- IP 네트웍 주소
- IP 브로드캐스트 주소
- IP 넷마스크
- 라우터 주소
- 도메인 네임 서버 주소

그리곤 여러분의 리눅스 네트웍 장치들을 위의 값들을 가지고 설정한다. 이 값들을 임의로 만들어 낸 후에 설정들이 동작하리라고 기대해선 안된다.

인터넷에 연결되지 않을 새로운 네트웍을 구성하는 경우.

만약 여러분이 개인적인 네트웍을 구성하고 인터넷에 연결하지 않을 계획이라면 여러분이 좋아하는 어떤 주소라도 선택할 수 있다. 그러나 안전성과 일관성의 이유 때문에 이런 목적을 위해 특별히 예비되어 있는 IP 네트웍 주소들이 있다. RFC1597에 자세히 나와있으며 다음의 것들이다.

RESERVED PRIVATE NETWORK ALLOCATIONS		
Network Class	Netmask	Network Addresses

```
|    A    | 255.0.0.0     | 10.0.0.0    - 10.255.255.255  |
|    B    | 255.255.0.0   | 172.16.0.0  - 172.31.255.255  |
|    C    | 255.255.255.0 | 192.168.0.0 - 192.168.255.255 |
-----------------------------------------------------------
```

여러분은 우선 여러분의 네트웍의 크기를 결정해야 하고 그 다음에 필요한 만큼의 주소를 고른다.

5.2 어디에 설정 명령을 넣어야 하나?

리눅스 시스템의 부트 과정에는 몇 개의 다른 방식이 있다. 커널이 부팅된 후 항상 '*init*'라 불리는 프로그램이 실행된다. *init* 프로그램은 /etc/inittab라는 자신의 설정 파일을 읽어들인 후 부트 과정을 시작한다. 모든 사람들이 Miguel van Smoorenburg에 의해 개발된 System V (Five) 방식으로 수렴하고 있지만 몇개의 서로 다른 *init* 방식이 있다.

init 프로그램은 항상 동일하다는 사실에도 불구하고 각각의 배포본에서 시스템 부트의 구성은 다른 방식으로 설정된다.

일반적으로 /etc/inittab 파일은 아래와 같은 엔트리들을 포함하고 있다.

```
si::sysinit:/etc/init.d/boot
```

이 줄은 부트 과정을 실제적으로 처리하는 쉘 스크립트 파일의 이름을 명시한다. 이 파일은 MS-DOS의 AUTOEXEC.BAT 파일과 어떤 면에서 같은 것이다.

부트 스크립트라 불리는 다른 스크립트들도 일반적으로 존재하며 종종 이 많은 것들 중 하나 속에서 네트웍이 설정된다.

아래의 표는 여러분의 시스템에 대해 알려준다.

배포본	Interface Config/Routing	Server Initialization
Debian	/etc/init.d/network	/etc/rc2.d/*
Slackware	/etc/rc.d/rc.inet1	/etc/rc.d/rc.inet2
RedHat	/etc/rc.d/init.d/network	/etc/rc.d/rc3.d/*

Debian과 RedHat이 시스템의 서비스를 시작하는 호스트 스크립트들에 디렉토리 전체를 사용하고 있다는 것을 주목하라. (보통 정보는 이 파일들 안에 들어있지 않다. 예를 들어 RedHat 시스템은 모든 시스템 설정들을 /etc/sysconfig 안의 파일들에 보관한다.) 만약 부트 과정의 자세한 사항을 알고 싶다면 */etc/inittab*과 *init*에 동반하는 문서를 들여다 볼 것을 권한다. 리눅스 저널 또한 시스템 초기화에 대한 글을 내놓을 예정이며 본 문서는 그 글이 웹에 올라오는 데로 링크시킬 것이다.

대부분의 요즘의 배포본은 많은 일반적인 네트웍 인터페이스들을 설정할 수 있는 프로그램들을 포함하고 있다. 이들중 하나를 가지고 있다면 손으로 직접 설정을 하기 전에 이 것들이 여러분이 원하는 것을 할 수 있는지를 먼저 보라.

배포본	네트웍 설정 프로그램
RedHat	/usr/bin/netcfg

```
Slackware | /sbin/netconfig
----------------------------------------
```

5.3 자신의 네트웍 인터페이스 만들기

많은 유닉스 운영체제에서는 네트웍 장치를 */dev*에서 볼 수 있지만 리눅스에서는 아니다. 리눅스에서 네트웍 장치는 소프트웨어에서 동적으로 많들어 지며 장치 파일이 존재할 필요가 없다.

대부분의 경우에 네트웍 장치는 하드웨어의 위치를 정하고 초기화하는 동안 장치 드라이버에 의해 자동으로 생성된다. 예를 들어 이더넷 장치 드라이버는 `eth[0..n]` 인터페이스를 이더넷 하드웨어의 위치를 정하면서 순차적으로 생성한다.

그러나 *slip*과 *ppp* 같은 몇몇 경우에는 네트웍 장치는 유저 프로그램의 작동에 의해 만들어진다. 역시 순차적인 장치 번호가 메겨지지만 장치들이 부팅 시에 자동으로 생성되진 않는다. 이러한 이유는 이더넷 장치와는 달리 동작중인 *slip* 이나 *ppp* 장치들의 수가 머신의 가동 시간 동안 바뀔 수 있기 때문이다.

5.4 네트웍 인터페이스 설정하기

여러분이 필요한 프로그램과 주소, 네트웍 정보를 모두 가지고 있다면 네트웍 인터페이스를 설정할 수 있다. 네트웍 인터페이스 설정에 관해 말할 때 우리는 네트웍 장치에 알맞은 주소를 할당하고 다른 설정 변수들에 알맞은 값들을 할당하는 과정에 대해 말하고 있는 것이다.

전형적으로 여러분은 아래와 비슷한 명령어를 사용한다.

```
root# ifconfig eth0 192.168.0.1 netmask 255.255.255.0 up
```

이 경우에 나는 '`192.168.0.1`'의 IP 주소와 '`255.255.255.0`'의 네트웍 마스크를 가지고 이더넷 인터페이스 '`eth0`'를 설정하고 있다. 명령문의 마지막에 오는 '*up*'은 인터페이스에게 작동하도록 말을 하는 것이지만 기본 명령이므로 생략할 수 있다. 인터페이스의 작동을 멈추기 위해선 단순히 "`ifconfig eth0 down`"라고 명령을 내려주면 된다.

인터페이스를 설정할 때 커널은 어떤 기본값을 가정한다. 예를들어 여러분이 인터페이스에 네트웍 주소와 브로드캐스트 주소를 설정할테지만 만약 하지 않는다면 위의 내 예제처럼 커널은 여러분이 설정한 넷 마스크에 기초하여 적당한 추정값을 적용할 것이며 넷마스크를 설정하지 않았다면 설정된 IP 주소의 네트웍 클래스에 기초해서 값들을 추정할 것이다. 내 예제에서 커널은 인터페이스에 설정되는 네트웍이 C 클래스라 가정하고 네트웍 주소를 '`192.168.0.0`'으로, 브로드캐스트 주소를 '`192.168.0.255`'로 설정하였다.

이 외에도 *ifconfig* 명령어에는 많은 다른 옵션들이 있다. 이 것들 중 가장 중요한 것들은 다음과 같다.

up

이 옵션은 인터페이스를 작동시킨다(그리고 기본 설정이다).

down

이 옵션은 인터페이스의 동작을 멈춘다.

[-]**arp**

이 옵션은 이 인터페이스에서 address resolution protocol 의 사용을 키거나 끈다.

[-]**allmulti**

이 옵션은 모든 hardware multicast packet들의 수신 기능을 키거나 끈다. Hardware multicast는 호스트의 그룹들이 특정 목적지로 보내진 패킷을 받을 수 있도록 해준다. desktop videoconferencing 같은 응용 프로그램을 사용할 때 중요할지 모르나 일반적으론 사용하지 않는다.

mtu N

이 변수는 장치의 *MTU* 값을 설정할 수 있게 해준다.

netmask <addr>

이 변수는 장치가 속한 네트웍의 네트웍 마스크를 설정할 수 있도록 해준다.

irq <addr>

이 변수는 특정 종류의 하드웨어에서만 작동하며 장치 하드웨어의 IRQ 값을 설정할 수 있도록 해준다.

[-]**broadcast** [**addr**]

이 변수는 broadcast 주소로 보내지는 데이타그램들의 수신을 기능을 설정하고 키거나 끈다.

[-]**pointopoint** [**addr**]

이 변수는 *slip*이나 *ppp*같은 point to point 연결의 원격지에서 머신의 주소를 설정할 수 있도록 해준다.

hw <type> <addr>

이 변수는 어떤 종류의 네트웍 장치들의 하드웨어 주소를 설정할 수 있도록 해준다. 이 기능은 이더넷에는 그다지 쓸모가 없지만 AX.25 같은 다른 종류의 네트웍에서는 유용하다.

여러분은 어떤 네트웍 인터페이스에도 *ifconfig* 명령을 사용할 수 있다. *pppd*나 *dip*같은 어떤 프로그램들은 네트웍 장치들을 만드는 동시에 설정 하므로 *ifconfig*을 직접 사용할 필요는 없다.

5.5 Name Resolver의 설정

'*Name Resolver*'는 리눅스의 표준 라이브러리의 일부이다. 기본 기능은 사용자에게 편한 '`ftp.funet.fi`'같은 호스트명을 기계에게 편한 `128.214.248.6`같은 IP 주소로 바꿔주는 서비스를 재공하는 것이다.

5.5.1 name이란 무엇인가?

여러분은 아마 인터넷 호스트명에 익숙할 것이나 그것들이 어떻게 생성되고 없어지는지는 이해하지 못할 것이다. 인터넷 도매인명은 근본적으로 계층적 형태이며 다시 말해 tree와 같은 구조를 가진다. '*domain*'은 name들의 묶음 혹은 집단이다. '*domain*'은 '*subdomain*'으로 다시 쪼개어 질 수 있다. '*toplevel domain*'은 subdomain이 아닌 domain이다. Top Level Domain은 RFC-920에 명시되어 있다. 가장 흔히 쓰이는 top level domain에 대한 예는 다음과 같다.

COM

상업 기관

EDU

교육 기관

GOV

정부 기관

MIL

군사 기관

ORG

기타 기관

NET

인터넷 관련 기관

Country Designator

특정 국가를 나타내는 두 글자의 코드이다. (역자주: 예를 들어 한국은 .kr, 일본은 .jp)

역사적인 이유 때문에 미국이 '`.us`'라는 국가 domain을 가지고 있음에도 불구하고 국가 의존적이지 않은 top level domain(ex. .com, .edu 등)에 속한 domain의 거의 대부분은 미국 내의 기관들에 의해 사용되었다. 그러나 `.com` 과 `.org`에 있어선 이제 그렇지 않으며 미국 외의 나라에서도 흔히 쓰인다.

이 top level domain들 각각은 subdomain들을 가지고 있다. 국가 이름에 기초한 top level domain들은 흔히 다시 `com`과 `edu`, `gov`, `mil`, `org` domain들에 기초한 subdomain들로 쪼개진다. 예를 들어 오스트레일리아의 상업 기관과 정부기관을 위해 각각 `com.au`와 `gov.au`를 사용할 수 있다. 그러나 이는 일반적인 규칙은 아니며 각 domain 명명권한을 가진 쪽의 정책에 달려있다.

다음 레벨의 domain은 대부분 기관의 이름을 나타낸다. 그 이상의 subdomain은 매우 다양해서 흔히 기관의 부서 구조에 기초하거나 기관의 네트웍 관리자에 의해 중요하게 생각되는 기준에 따른다.

name의 최 좌측 부분은 항상 호스트 머신에 할당된 유일한 이름이며 '*hostname*'이라 불린다. hostname의 오른쪽에 있는 name의 부분은 '*domainname*'이라 불리고 전체의 name은 '*Fully Qulified Domain Name*'이라 부른다.

일례로 Terry의 호스트를 사용하기 위해선 fully qulified domain name은 '`perf.no.itg.telstra.com.au`'이다. 이것은 host name이 '`perf`'이고 domain name이 '`no.itg.telstra.com.au`'라는 것을 나타낸다. domain name은 그(Terry)의 국가, Australia에 기초한 top level domain에 기반을 두고 있으고 그의 email 주소는 상업 기관에 속해 있으며 '`.com`'이 다음 level의 domain 으로 쓰였다. 회사의 이름은 '`telstrs`'이며(였으며) 내부 명명 구조는 기관 구조에 기초하고 있는데 이 경우에 그 머신(Terry의 머신)은 Network Operations section의 Information Technology Group에 속해있다.

일반적으로 name은 매우 짧다. 예를 들어 내 ISP는 "`systemy.it`"라 불리고 내 비 영리 기관은 "`linux.it`"이라 불리며 `com`이나 `org` subdomain을 사용하지 않아서 내 호스트는 "`morgana.systemy.it`"이고 `rubini@linux.it`은 가능한 email 주소이다. domain의 소유자가 subdomain 뿐 아니라 hostname을 등록할 권리를 가진다는 것을 명심해라. 예를 들어 `linux.it`의 소유자가 LUG를 위한 subdomain을 만드는 것에 찬성했기 때문에 내가 속한 LUG는 `pluto.linux.it`의 domain을 사용한다.

5.5.2 여러분이 필요한 정보

여러분은 여러분의 호스트의 이름이 속할 domain을 알아야 한다. name resolver software는 '*Domain Name Server*'에 요청을 하여 이 name 변환 서비스를 있으므로 여러분이 사용할 수 있는 local nameserver의 IP주소를 알 필요가 있을 것이다.

또 수정해야 할 파일이 세 개 있으며 이를 차례대로 다루겠다.

5.5.3 /etc/resolv.conf

`/etc/resolv.conf`는 name resolver code에 대한 주 설정 파일이다. 형식은 매우 간단해서 줄당 한 개의 키워드를 갖는 텍스트 파일이다. 특별히 쓰이는 세 개의 키워드가 있으며 아래와 같다.

domain

이 키워드는 local domain name을 명시한다.

search

이 키워드는 hostname을 찾기 위한 다른 domain의 리스트를 명시한다.

nameserver

이 키워드는 name resolving을 위한 쿼리를 보낼 domain name server의 IP 주소를 명시하며 여러번 쓰일 수 있다. (역자주: hostname을 해당 IP 주소로 변환하는 것은 name resolving이라 한다. 이름 변환 등의 우리 말을 쓸 수도 있겠지만 정확한 의미 전달을 위해 name resolving을 이하 그대로 사용하겠다.)

`/etc/resolv.conf`의 예는 아래와 같다.

```
domain maths.wu.edu.au
search maths.wu.edu.au wu.edu.au
nameserver 192.168.10.1
nameserver 192.168.12.1
```

이 예는 unqualified name(domain이 없는 hostname)의 뒤에 붙을 기본 domain name이 `maths.wu.edu.au`이며 이 domain에서 호스트를 찾지 못했으면 `wu.edu.au` domain을 직접 찾아볼 것을 명시하고 있다. 두 개의 nameserver 엔트리가 있는데 각각은 name을 resolve하기 위해 name resolver code에서 보낸 요구를 받을 것이다.

5.5.4 /etc/host.conf

`/etc/host.conf` 파일은 name resolver code의 행동을 관리하는 몇몇 항목들을 설정하는 곳이다. 이 파일의 형식은 '`resolv+`' 맨 페이지에 자세히 나와있다. 거의 모든 환경에선 아래의 예가 잘 작동할 것이다.

```
order hosts,bind
multi on
```

이 설정은 name resolver에게 nameserver에게 쿼리를 보내기 전에 `/etc/hosts` 파일을 먼저 검사하고 처음 발견되는 하나 말고도 `/etc/hosts` 파일 안에서 발견된 호스트에 해당하는 모든 주소를 돌려주도록 하고 있다.

5.5.5 /etc/hosts

`/etc/hosts` 파일은 로컬 호스트들의 name과 IP 주소들을 넣어두는 곳이다. 이 파일 안에 호스트를 넣어 놓으면 IP 주소를 얻으려 할 때 domain name server를 검색할 필요가 없다. 그러나 호스트의 IP 주소가 바뀌면 여러분 스스로가 이 파일을 최신으로 계속 갱신해야 한다는 것이 단점이다. 잘 정비된 시스템에서는 이 파일 안에 나오는 hostname들은 오직 loopback 인터페이스나 로컬 호스트들의 name이다.

```
# /etc/hosts
127.0.0.1       localhost loopback
192.168.0.1     this.host.name
```

첫 번째 엔트리처럼 한 라인에 둘 이상의 host name을 명시할 수 있으며 첫 엔트리는 loopback 인터페이스에 대한 표준 엔트리이다.

5.5.6 name server 돌리기

로컬 nameserver를 돌리기를 원한다면 쉽게 할 수 있다. `DNS-HOWTO`와 여러분이 가진 *BIND* (Berkeley Internet Name Domain) 버전에 포함된 문서들을 참조하라.

5.6 loopback 인터페이스 설정하기

'loopback' 인터페이스는 여러분 자신에게 연결할 수 있도록 해주는 특별한 인터페이스이다. 이것이 필요한 여러 이유가 있는데 한 예로 네트웍의 다른 사람을 방해하지 않으면서 어떤 네트웍 소프트웨어를 시험해볼 수 있다. 편의상 '127.0.0.1'의 IP 주소가 loopback용으로 할당되어 왔다. 따라서 어떤 머신으로 가더라도 127.0.0.1로 telnet 연결을 연다면 로컬 호스트로 접속될 것이다.

loopback 인터페이스의 설정은 간단하며 반드시 해야 한다(그러나 이 작업은 보통 표준 초기화 스크립트에서 행해진다).

```
root# ifconfig lo 127.0.0.1
root# route add -host 127.0.0.1 lo
```

다음 부분에선 *route* 명령에 대해 말하겠다.

5.7 라우팅.

라우팅은 중요한 문제이며 그것에 대한 두꺼운 책들을 쓰는 것도 어렵지 않다. 여러분의 대부분은 매우 간단한 라우팅 조건을 가지고 있을 것이고 일부는 그것 조차 없을 것이다. 나는 라우팅의 일부 기본적인 기초에 대해서만 다룰 것이다. 만약 좀 더 자세한 내용에 관심이 있다면 이 문서의 처음에 나온 리퍼런스들을 볼 것을 권한다.

우선 정의부터 시작한다. IP 라우팅이란 무엇인가? 아래는 내가 사용하고 있는 정의이다.

> IP 라우팅은 여러 개의 네트웍 연결을 가진 호스트가 자신이 받은 데이타그램을 전달할 곳을 결정하는 과정이다.

이를 간단한 예와 함께 설명하는 것이 유용할지 모른다. 전형적인 사무용 라우터를 생각해보면 그것은 인터넷으로의 PPP연결과 워크스테이션들에 연결된 많은 이더넷 부분들, 다른 사무실로의 PPP 연결을 가지고 있을 것이다. 라우터는 자신의 네트웍 연결 중 한 곳으로부터 데이타그램을 받으면 다음으로 그 데이타그램을 전달할 자신의 인터페이스를 결정하는데 이 메카니즘이 라우팅이다. 단순한 호스트들도 라우팅이 필요다. 모든 인터넷 호스트는 두 개의 네트웍 장치를 가지고 있는데 하나는 위에서 설명한 loopback 인터페이스이고 다른 하나는 이더넷이나 PPP, SLIP 같은 자신이 속한 네트웍과 통신하기 위해 사용하는 인터페이스이다.

그래서 라우팅은 어떻게 작동하는가? 각각의 호스트들은 라우팅 테이블이라 불리는 라우팅 규칙들의 특별한 리스트를 가지고 있다. 이 테이블은 일반적으로 세 개 이상의 필드를 갖는 열(row)들을 가지고 있다. 첫 번째는 목표 주소이고 두 번째는 데이타그램이 발송될 인터페이스의 이름이며 세 번째는 네트웍 상에서 데이타그램을 그 다음 단계에서 전달할 다른 머신의 IP 주소를 가지고 있을 수 있다. 리눅스에서 여러분은 다음과 같은 명령을 통해 이 테이블의 내용을 볼 수 있다.

```
user% cat /proc/net/route
```

혹은 다음 명령을 쓸 수도 있다.

```
user% /sbin/route -n
user% netstat -r
```

라우팅 과정은 매우 단순하다. 데이타그램이 받아지면 목적지 주소가 검사되고 테이블의 각 엔트리들과 비교된다. 그 주소에 가장 알맞는 엔트리가 선택되고 해당 인터페이스로 데이타그램이 전달된다. 게이트웨이 필드

에 값이 존재한다면 데이타그램은 명시된 인터페이스를 통해 그 호스트로 전송될 것이고 그렇지 않다면 목적지 주소가 그 인터페이스가 속한 네트웍 상에 있다고 여겨진다.

이 테이블을 다루기 위해선 특별한 명령이 필요하다. 이 명령은 명령행 인자들을 받아들여 이를 커널에게 라우팅 엔트리를 추가, 삭제 혹은 수정하도록 요구하는 커널 시스템 콜로 변경한다. 이 명령은 '*route*'다.

간단한 예를 보자. 여러분이 이더넷 네트웍상에 있다고 하자. 주소가 192.168.1.0인 class-C 네트웍이라는 것을 알고 있고 192.168.1.10의 IP 주소를 사용하도록 배정받았으며 인터넷에 연결된 라우터가 192.168.1.1 이라는 것을 알고 있다.

먼저 할 일은 위에 나온 인터페이스를 설정하는 것이다. 다음과 같은 명령을 쓸 것이다.

```
root# ifconfig eth0 192.168.1.10 netmask 255.255.255.0 up
```

주소가 192.168.1.*에 해당하는 호스트들에게 가는 데이타그램들이 위의 이더넷 장치로 보내지도록 커널에 알려주기 위해선 라우팅 테이블에 엔트리를 삽입할 필요가 있다.

```
root# route add -net 192.168.1.0 netmask 255.255.255.0 eth0
```

이 엔트리가 네트웍 라우트라는 것을 라우트 프로그램에 알려주기 위해서 '-net' 인자를 사용한 것에 주의한다. 이외에도 '-host'를 사용하여 특정한 한 IP 주소로의 라우팅을 설정할 수 있다.

이 라우팅은 여러분에게 여러분의 이더넷 네트웍 위의 모든 호스트들에 IP 접속을 할 수 있도록 해줄 것이다. 그러나 이 이더넷 네트웍 위에 있지 않은 IP 호스트들은 어떠한가?

모든 가능한 목적지 네트웍으로의 라우팅 정보를 추가하는 것은 매우 힘든 일이며 따라서 이 작업을 단순화하기 위해 사용되는, 'default' 라우트 라 불리는 특별한 기법이 있다. default 라우트는 모든 가능한 목적지에 적용되나 매우 빈약해서 해당 주소에 적용되는 다른 엔트리가 있다면 이것이 default 라우트 대신 쓰인다. default 라우트의 개념은 단순히 "다른 모든 것들은 이곳으로 가야한다"고 말하도록 하는 것이다. 이 예에서 나는 여러분이 다음과 같은 엔트리를 사용하도록 했다.

```
root# route add default gw 192.168.1.1 eth0
```

'gw' 인자는 라우트 명령에게 다음 인자가 이 엔트리에 적용되는 모든 데이타그램이 다른 라우팅을 위해 보내져야 할 게이트웨이나 라우터 기계의 이름 혹은 IP주소라는 것을 말해준다.

따라서 완성된 설정은 아래와 같다.

```
root# ifconfig eth0 192.168.1.10 netmask 255.255.255.0 up
root# route add -net 192.168.1.0 netmask 255.255.255.0 eth0
root# route add default gw 192.168.1.1 eth0
```

여러분의 네트웍 'rc' 파일을 자세히 들여다본다면 이와 비슷해 보이는 하나 이상의 것들을 발견할 것이다. 이는 매우 일반적인 설정이다.

좀 더 복잡한 라우팅 설정을 보도록 하자. 앞에서 나왔던, 인터넷으로 PPP 연결을 지원하고 사무실 안의 워크스테이션들간의 랜 연결을 지원하는 라우터의 설정을 한다고 생각해 보자. 이 라우터가 세 개의 이더넷 부분과 한 개의 PPP 연결을 가지고 있다고 가정하자. 우리의 라우팅 설정은 아래와 같을 것이다.

```
root# route add -net 192.168.1.0 netmask 255.255.255.0 eth0
root# route add -net 192.168.2.0 netmask 255.255.255.0 eth1
root# route add -net 192.168.3.0 netmask 255.255.255.0 eth2
root# route add default ppp0
```

각각의 워크스테이션들은 위의 것보다 간단한 형식을 이용할 것이고 라우터만이 각 내트웍의 라우팅을 명시할 필요가 있다. 왜냐하면 워크스테이션들의 default 라우트 메카니즘은 라우팅 정보를 알맞게 분배하는 문제를 라우터에게 남겨둔 채로 이 정보들을 받아들이기 때문이다. 여러분은 위의 default 라우트가 'gw'를 명시하지 않은 것을 이상히 여길 것이다. 이유는 간단하다. PPP나 slip같은 시리얼 연결 프로토콜은 그 네트웍 상에 양 끝에 단 두 개의 호스트만을 가지고 있다. 연결의 반대 끝 호스트를 게이트웨이로 명시하는 것은 무의미하며 선택의 여지가 없기 때문에 중복적인 것이다. 따라서 이런 종류의 네트웍 연결에 대해선 게이트웨이를 설정해 줄 필요가 없다. 이더넷이나 아크넷, 토큰 링 같은 다른 종류의 네트웍들은 많은 수의 호스트들을 지원하기 때문에 게이트웨이가 명시되어야 한다.

5.7.1 그래서, *routed* 프로그램이 하는 일이 뭔가 ?

위에 설명된 라우팅 설정은 목적지까지 단순한 하나의 경로만이 존재하는 간단한 네트웍 환경에 가장 알맞다. 네트웍이 좀 더 복잡하다면 모든 것들이 더 복잡해진다. 운좋게도 여러분중 대부분에게 이것은 중요치 않을 것이다.

설명된것 같은 '수동 라우팅(manual routing)'이나 '고정 라우팅(static routing)' 이 갖는 큰 문제점은 네트웍 상의 연결이 끊어지거나 기계가 죽었을 때 데이타그램을 다른 경로 - 다른 경로가 있다면 - 로 돌리는 유일한 방법은 중간에 직접 끼어들어서 적당한 명령을 실행시키는 것이다. 당연히 이것은 불편하고 느리고 비현실적이며 문제를 일으킬 소지가 많다. 네트웍 문제 발생시 다른 경로가 존재할 때 라우팅 테이블을 자동으로 수정해주는 많은 기법들이 발전되어 왔고 이 모든 기술들은 '동적 라우팅 프로토콜(dynamic routing protocols)'이라는 말로 묶여진다.

여러분은 몇몇 일반적인 동적 라우팅 프로토콜에 대해 들어봤을 것이다. 가장 많이 사용되는 것은 RIP (Routing Information Protocol)와 OSPF (Open Shortest Path First Protocol)이다. RIP는 중소기업의 네트웍이나 빌딩 내부의 네트웍 같은 작은 크기의 네트웍에 매우 많이 이용된다. OSPF는 RIP보다 최근에 나왔고 규모가 큰 네트웍의 설정을 다룰 수 있고 네트웍 안에 가능한 경로의 수가 매우 많은 환경에 보다 더 적합하다. 이 프로토콜들의 대표적인 구현이 '*routed*' - RIP 와 '*gated*' - RIP, OSPF 등이다. '*routed*' 프로그램은 대부분의 리눅스 배포본에 기본으로 제공되며 혹은 위에서 자세히 설명한 'NetKit' 안에 포함되어 있다.

여러분이 동적 라우팅 프로토콜을 사용하는 곳과 방법의 한 예가 아래에 있다.

```
    192.168.1.0 /                         192.168.2.0 /
       255.255.255.0                         255.255.255.0
     -                                      -
     |                                      |
     |   /-----\                 /-----\    |
     |   |     |ppp0   //    ppp0|     |    |
eth0 |---|  A  |------//---------|  B  |---| eth0
     |   |     |     //          |     |    |
     |   \-----/                 \-----/    |
     |      \ ppp1             ppp1 /       |
     -       \                     /        -
              \                   /
               \                 /
                \               /
                 \             /
                  \           /
                   \         /
                    \       /
                     \     /
                  ppp0\   /ppp1
                     /-----\
                     |     |
                     |  C  |
```

```
         |      |
         \-----/
            |eth0
            |
       |---------|
       192.168.3.0 /
          255.255.255.0
```

우리는 A, B, C 이렇게 세 개의 라우터를 가지고 있다. 각각은 C 클래스 (넷마스크 255.255.255.0) 의 이더넷 네트웍 부분을 지원한다. 또한 각 라우터는 다른 라우터로의 PPP 연결을 가지고 있다. 네트웍은 삼각형 형태를 띤다.

라우터 A의 라우팅 테이블은 아래와 같아야 한다.

```
root# route add -net 192.168.1.0 netmask 255.255.255.0 eth0
root# route add -net 192.168.2.0 netmask 255.255.255.0 ppp0
root# route add -net 192.168.3.0 netmask 255.255.255.0 ppp1
```

라우터 A와 B간의 연결이 끊어지기 전까진 잘 작동하고 있었다. 위의 라우팅 엔트리에서 이 연결(A와 B간의)이 끊어지면 A의 이더넷 부분 위의 호스트들은 B의 이더넷 부분 위의 호스트들과 연결될 수 없다. 모든 데이타그램들이 끊어져 있는 A의 ppp0 연결로 보내지기 때문이다. A상의 호스트들은 여전히 C의 이더넷 부분 상의 호스트들과 통신이 가능하고 C 상의 호스트들도 B 상의 호스트들과 통신이 가능하다. B와 C 사이의 연결은 살아있기 때문이다.

그러나 잠시 생각해보자. A와 C가 통신할 수 있고 C와 B가 통신할 수 있다면 왜 A 라우터는 B로 갈 데이타그램들을 C를 통해서 보내고 C에게 그들을 B로 보내도록 하지 않는가? RIP같은 동적 라우팅 프로토콜(dynamic routing protocols)은 바로 이런 문제를 해결하기 위해 만들어졌다. 라우터 A, B, C가 각각 라우팅 데몬을 돌리고 있다면 많약 네트웍 상의 연결들 중 하나가 끊겼을 경우 새로운 네트웍 상태를 반영하기 위해 라우팅 테이블들이 자동으로 수정될 것이다. 이런 네트웍을 설정하는 것은 간단해서 각 라우터에 딱 두 가지 일만 해주면 된다. 라우터 A의 경우

```
root# route add -net 192.168.1.0 netmask 255.255.255.0 eth0
root# /usr/sbin/routed
```

'*routed*' 라우팅 데몬은 시작시에 가능한 모든 네트웍 포트들을 자동으로 찾고 그 호스트의 라우팅 테이블을 결정하고 갱신하기 위해 각 네트웍 장치로 메시지를 보내거나 받는다.

지금까지 동적 라우팅의 매우 간단한 설명이었고 이를 사용하는 예를 보았다. 만약 더 많은 정보를 원한다면 문서 맨 앞에 열거된 참조문서들을 볼 것을 권한다.

동적 라우팅과 관련된 중요한 점은

1. 여러분은 여러분의 리눅스 머신이 목적지까지 많은 경로중에서 선택할 수 있는 입장일 때만 동적 라우팅 프로토콜 데몬을 돌릴 필요가 있다.
2. 동적 라우팅 데몬은 네트웍 변화에 맞추기 위해 라우팅 테이블을 자동으로 수정한다.
3. RIP는 작은 규모의 네트웍에 알맞다.

5.8 네트웍 서버와 서비스의 설정.

네트웍 서버와 서비스는 원격지의 사용자에게 여러분의 리눅스 머신을 사용하도록 해 주는 프로그램이다. 서버 프로그램은 네트웍 포트를 통해 메시지를 받는다. 네트웍 포트는 특정 호스트의 특정 서비스를 지칭하는 수

단이며 서버가 요청이 들어온 telnet 연결과 ftp 연결을 구분하는 수단이다. 원격 사용자는 여러분의 머신에 네트웍 연결을 만들고 서버 프로그램인 네트웍 데몬 프로그램은 그 포트를 듣고 있다가 연결을 받아들이고 작업을 수행한다. 네트웍 데몬이 작동하는 두 가지 방식이 있는데 둘 다 실제로 흔히 사용된다. 두 가지는

standalone

네트웍 데몬 프로그램이 해당 네트웍 포트를 듣고 있다가 연결이 들어오면 그 자신이 네트웍 연결을 관리해서 서비스를 제공한다.

slave to the *inetd* server

inetd 서버는 들어오는 네트웍 연결을 전문적으로 다루는 특별한 네트웍 데몬이다. 들어온 연결이 받아들여졌을 때 실행해야 할 프로그램을 알려주는 설정 파일을 가지고 있다. 어떤 서비스 포트도 tcp나 udp 혹은 둘 다를 사용하도록 설정할 수 있다. 포트는 곧 말할 다른 파일에 기술된다.

설정할 필요가 있는 두 개의 중요한 파일이 있다. 그것은 포트 번호에 이름을 부여하는 `/etc/services`파일과 *inetd* 네트웍 데몬의 설정 파일인 `/etc/inetd.conf`이다.

5.8.1 `/etc/services`

`/etc/services` 파일은 사람에게 친숙한 이름을 기계가 친숙한 서비스 포트에 부여하는 간단한 데이타베이스다. 형식은 매우 단순하다. 한 줄에 데이타베이스 엔트리가 나오는 텍스트 파일이다. 각 엔트리는 whitespace(tab이나 space) 문자로 구분되는 세 개의 필드를 가지고 있는데 그 필드는 다음과 같다.

```
name        port/protocol        aliases     # comment
```

name

설명되는 서비스를 타나내는 한 단어 길이의 이름

port/protocol

이 필드는 두 개의 서브필드로 나뉜다.

port

명명된 서비스가 제공될 포트 번호를 지시하는 숫자. 대부분의 일반적인 서비스들은 서비스 번호를 이미 부여받았다. 이는 `RFC-1340`에 명시되어 있다.

protocol

이 서브필드는 `tcp`나 `udp` 로 설정된다.

`18/tcp`와 `18/udp`의 엔트리들은 매우 다르며 같은 서비스를 둘 다(tcp와 udp)에서 제공하는지에 대한 기술적 이유는 없다. 보통 상식이 통하며 여러분이 둘 다에 대한 엔트리를 보는 경우는 특정 서비스가 `tcp`와 `udp` 둘 다를 통해 가능한 때 뿐이다.

aliases

이 서비스 엔트리를 참조하기 위해 사용되는 다른 이름들

'`#`' 문자 다음에 나오는 줄은 주석으로 여겨지며 무시된다.

/etc/services 파일의 예 모든 요즘의 리눅스 배포본은 괜찮은 /etc/services 파일을 제공한다. 단지 여러분이 머신을 처음부터 다시 만드는 경우를 위해 오래된 Debian 배포본에서 제공되는 /etc/services 파일이 여기 있다.

```
# /etc/services:
# $Id: services,v 1.3 1996/05/06 21:42:37 tobias Exp $
#
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a single well-known
# port number for both TCP and UDP; hence, most entries here have two entries
# even if the protocol doesn't support UDP operations.
# Updated from RFC 1340, ``Assigned Numbers'' (July 1992).  Not all ports
# are included, only the more common ones.

tcpmux          1/tcp                           # TCP port service multiplexer
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp          quote
msp             18/tcp                          # message send protocol
msp             18/udp                          # message send protocol
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp-data        20/tcp
ftp             21/tcp
ssh             22/tcp                          # SSH Remote Login Protocol
ssh             22/udp                          # SSH Remote Login Protocol
telnet          23/tcp
# 24 - private
smtp            25/tcp          mail
# 26 - unassigned
time            37/tcp          timserver
time            37/udp          timserver
rlp             39/udp          resource        # resource location
nameserver      42/tcp          name            # IEN 116
whois           43/tcp          nicname
re-mail-ck      50/tcp                          # Remote Mail Checking Protocol
re-mail-ck      50/udp                          # Remote Mail Checking Protocol
domain          53/tcp          nameserver      # name-domain server
domain          53/udp          nameserver
mtp             57/tcp                          # deprecated
bootps          67/tcp                          # BOOTP server
bootps          67/udp
bootpc          68/tcp                          # BOOTP client
bootpc          68/udp
tftp            69/udp
gopher          70/tcp                          # Internet Gopher
gopher          70/udp
rje             77/tcp          netrjs
```

```
finger          79/tcp
www             80/tcp          http            # WorldWideWeb HTTP
www             80/udp                          # HyperText Transfer Protocol
link            87/tcp          ttylink
kerberos        88/tcp          kerberos5 krb5  # Kerberos v5
kerberos        88/udp          kerberos5 krb5  # Kerberos v5
supdup          95/tcp
# 100 - reserved
hostnames       101/tcp         hostname        # usually from sri-nic
iso-tsap        102/tcp         tsap            # part of ISODE.
csnet-ns        105/tcp         cso-ns          # also used by CSO name server
csnet-ns        105/udp         cso-ns
rtelnet         107/tcp                         # Remote Telnet
rtelnet         107/udp
pop-2           109/tcp         postoffice      # POP version 2
pop-2           109/udp
pop-3           110/tcp                         # POP version 3
pop-3           110/udp
sunrpc          111/tcp         portmapper      # RPC 4.0 portmapper TCP
sunrpc          111/udp         portmapper      # RPC 4.0 portmapper UDP
auth            113/tcp         authentication tap ident
sftp            115/tcp
uucp-path       117/tcp
nntp            119/tcp         readnews untp   # USENET News Transfer Protocol
ntp             123/tcp
ntp             123/udp                         # Network Time Protocol
netbios-ns      137/tcp                         # NETBIOS Name Service
netbios-ns      137/udp
netbios-dgm     138/tcp                         # NETBIOS Datagram Service
netbios-dgm     138/udp
netbios-ssn     139/tcp                         # NETBIOS session service
netbios-ssn     139/udp
imap2           143/tcp                         # Interim Mail Access Proto v2
imap2           143/udp
snmp            161/udp                         # Simple Net Mgmt Proto
snmp-trap       162/udp         snmptrap        # Traps for SNMP
cmip-man        163/tcp                         # ISO mgmt over IP (CMOT)
cmip-man        163/udp
cmip-agent      164/tcp
cmip-agent      164/udp
xdmcp           177/tcp                         # X Display Mgr. Control Proto
xdmcp           177/udp
nextstep        178/tcp         NeXTStep NextStep       # NeXTStep window
nextstep        178/udp         NeXTStep NextStep       # server
bgp             179/tcp                         # Border Gateway Proto.
bgp             179/udp
prospero        191/tcp                         # Cliff Neuman's Prospero
prospero        191/udp
irc             194/tcp                         # Internet Relay Chat
irc             194/udp
smux            199/tcp                         # SNMP Unix Multiplexer
smux            199/udp
at-rtmp         201/tcp                         # AppleTalk routing
at-rtmp         201/udp
at-nbp          202/tcp                         # AppleTalk name binding
```

```
at-nbp          202/udp
at-echo         204/tcp                          # AppleTalk echo
at-echo         204/udp
at-zis          206/tcp                          # AppleTalk zone information
at-zis          206/udp
z3950           210/tcp         wais             # NISO Z39.50 database
z3950           210/udp         wais
ipx             213/tcp                          # IPX
ipx             213/udp
imap3           220/tcp                          # Interactive Mail Access
imap3           220/udp                          # Protocol v3
ulistserv       372/tcp                          # UNIX Listserv
ulistserv       372/udp
#
# UNIX specific services
#
exec            512/tcp
biff            512/udp         comsat
login           513/tcp
who             513/udp         whod
shell           514/tcp         cmd              # no passwords used
syslog          514/udp
printer         515/tcp         spooler          # line printer spooler
talk            517/udp
ntalk           518/udp
route           520/udp         router routed    # RIP
timed           525/udp         timeserver
tempo           526/tcp         newdate
courier         530/tcp         rpc
conference      531/tcp         chat
netnews         532/tcp         readnews
netwall         533/udp                          # -for emergency broadcasts
uucp            540/tcp         uucpd            # uucp daemon
remotefs        556/tcp         rfs_server rfs   # Brunhoff remote filesystem
klogin          543/tcp                          # Kerberized ‘rlogin’ (v5)
kshell          544/tcp         krcmd            # Kerberized ‘rsh’ (v5)
kerberos-adm    749/tcp                          # Kerberos ‘kadmin’ (v5)
#
webster         765/tcp                          # Network dictionary
webster         765/udp
#
# From ‘‘Assigned Numbers’’:
#
#> The Registered Ports are not controlled by the IANA and on most systems
#> can be used by ordinary user processes or programs executed by ordinary
#> users.
#
#> Ports are used in the TCP [45,106] to name the ends of logical
#> connections which carry long term conversations.  For the purpose of
#> providing services to unknown callers, a service contact port is
#> defined.  This list specifies the port used by the server process as its
#> contact port.  While the IANA can not control uses of these ports it
#> does register or list uses of these ports as a convenience to the
#> community.
#
```

```
ingreslock      1524/tcp
ingreslock      1524/udp
prospero-np     1525/tcp                # Prospero non-privileged
prospero-np     1525/udp
rfe             5002/tcp                # Radio Free Ethernet
rfe             5002/udp                # Actually uses UDP only
bbs             7000/tcp                # BBS service
#
#
# Kerberos (Project Athena/MIT) services
# Note that these are for Kerberos v4 and are unofficial.  Sites running
# v4 should uncomment these and comment out the v5 entries above.
#
kerberos4       750/udp         kdc     # Kerberos (server) udp
kerberos4       750/tcp         kdc     # Kerberos (server) tcp
kerberos_master 751/udp                 # Kerberos authentication
kerberos_master 751/tcp                 # Kerberos authentication
passwd_server   752/udp                 # Kerberos passwd server
krb_prop        754/tcp                 # Kerberos slave propagation
krbupdate       760/tcp         kreg    # Kerberos registration
kpasswd         761/tcp         kpwd    # Kerberos "passwd"
kpop            1109/tcp                # Pop with Kerberos
knetd           2053/tcp                # Kerberos de-multiplexor
zephyr-srv      2102/udp                # Zephyr server
zephyr-clt      2103/udp                # Zephyr serv-hm connection
zephyr-hm       2104/udp                # Zephyr hostmanager
eklogin         2105/tcp                # Kerberos encrypted rlogin
#
# Unofficial but necessary (for NetBSD) services
#
supfilesrv      871/tcp                 # SUP server
supfiledbg      1127/tcp                # SUP debugging
#
# Datagram Delivery Protocol services
#
rtmp            1/ddp                   # Routing Table Maintenance Protocol
nbp             2/ddp                   # Name Binding Protocol
echo            4/ddp                   # AppleTalk Echo Protocol
zip             6/ddp                   # Zone Information Protocol
#
# Debian GNU/Linux services
rmtcfg          1236/tcp                # Gracilis Packeten remote config server
xtel            1313/tcp                # french minitel
cfinger         2003/tcp                # GNU Finger
postgres        4321/tcp                # POSTGRES
mandelspawn     9359/udp        mandelbrot      # network mandelbrot

# Local services
```

실제론 새 서비스가 생겨나면 파일도 커진다. 여러분의 현재 파일이 불완전하다고 생각되면 최신 배포본에서 /etc/services 파일을 복사할 것을 권한다.

5.8.2 /etc/inetd.conf

/etc/inetd.conf 파일은 *inetd* 서버 데몬용 설정 파일이다. 그 기능은 *inetd*에게 특정 서비스에 대한 연결 요청을 받았을 때 무엇을 해야 하는지를 알려주는 것이다. 여러분이 연결을 받아들일 각 서비스에 대해 여러분은 *inetd* 에게 어떤 네트워크 서버 데몬을 어떻게 실행할지를 알려줘야 한다.

이 파일의 형식도 매우 간단하다. 제공하려는 서비스를 설명하는 라인들로 이루어진 텍스트 파일이다. '#'로 시작하는 줄의 모든 텍스트는 주석으로 간주되어 무시된다. 각 줄은 whitespace(tab이나 space) 문자로 구분되는 일곱개의 필드로 구성되며 whitespace의 개수는 관계없다. 일반적인 형식은 다음과 같다.

```
service socket_type proto flags user server_path server_args
```

service

/etc/services 파일에서 가져온 이 설정에 해당하는 서비스다.

socket_type

현재 엔트리에 알맞는 소켓 종류를 나타나낸 필드이며 가능한 값은 stream, dgram, raw, rdm, seqpacket 들이다. 원랜 상당히 기술적인 것이지만 편의상 거의 모든 tcp 기반 서비스는 stream을, udp 기반의 서비스들은 dgram을 사용한다. 다른 값을 쓰는 서버 데몬들은 매우 특별한 종류 뿐이다.

proto

이 엔트리에 적합한 프로토콜. 이 값은 /etc/services 파일 안의 해당 엔트리와 일치해야 하며 전형적으로 tcp이거나 udp이다. 선의 RPC (Remote Procedure Call) 기반의 서버들은 rpc/tcp나 rpc/udp를 사용한다.

flags

이 필드에 오는 값은 딱 두 개 뿐이다. 이 필드는 *inetd*에게 네트웍 서버 프로그램이 시작된 후에 소켓을 놔 주는지를 알려주며 결과적으로 *inetd*가 다른 연결 요청이 들어올 때 서버를 또 실행할 지 아니면 이미 실행 중인 서버 데몬이 새 연결 요청을 다루도록 해야할지를 알려준다. 정확한 해결방법이 아니겠지만 편의상 모든 tcp 서버들은 이 필드값이 nowait가 되도록 하고 udp 서버들은 wait값을 갖는다. 이에 대한 예외가 존재함을 주의해야 하며 여러분이 확실치 못할 경우 예제를 따라 하도록 해라.

user

이 필드는 /etc/passwd 안의 어떤 사용자 계정이 새로 시작되는 네트웍 서버의 소유자가 될 것인지를 나타낸다. 여러분이 보안 위첨에서 벗어나고자 할 경우 이는 매우 유용하다. 네트웍 서버 보안이 깨졌을 때 피해를 최소화하도록 엔트리의 사용자를 nobody로 할 수도 있다. 그러나 보통 이 필드는 root로 설정되는데 이는 많은 서버들이 정상적인 작동을 위해 루트 권한을 필요로 하기 때문이다.

server_path

이 필드는 이 엔트리에서 실행시키는 실제 서버 프로그램의 경로이다.

server_args

이 필드는 줄의 남은 부분으로 이루어지며 선택사항이다. 이 필드는 서버 데몬 프로그램이 실행될 때 넘겨줄 인자들을 넣어놓는 곳이다.

/etc/inetd.conf의 예 /etc/services 파일처럼 요즘의 모든 배포본들은 그냥 쓰기에 꽤 좋은 /etc/inetd.conf 파일을 포함하고 있다. 아래는 Debian 배포본에 들어있는 /etc/inetd.conf 파일이다.

```
# /etc/inetd.conf:  see inetd(8) for further informations.
#
# Internet server configuration database
```

```
#
#
# Modified for Debian by Peter Tobias <tobias@et-inf.fho-emden.de>
#
# <service_name> <sock_type> <proto> <flags> <user> <server_path> <args>
#
# Internal services
#
#echo           stream  tcp     nowait  root    internal
#echo           dgram   udp     wait    root    internal
discard         stream  tcp     nowait  root    internal
discard         dgram   udp     wait    root    internal
daytime         stream  tcp     nowait  root    internal
daytime         dgram   udp     wait    root    internal
#chargen        stream  tcp     nowait  root    internal
#chargen        dgram   udp     wait    root    internal
time            stream  tcp     nowait  root    internal
time            dgram   udp     wait    root    internal
#
# These are standard services.
#
telnet  stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.telnetd
ftp     stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.ftpd
#fsp    dgram   udp     wait    root    /usr/sbin/tcpd  /usr/sbin/in.fspd
#
# Shell, login, exec and talk are BSD protocols.
#
shell   stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rshd
login   stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rlogind
#exec   stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rexecd
talk    dgram   udp     wait    root    /usr/sbin/tcpd  /usr/sbin/in.talkd
ntalk   dgram   udp     wait    root    /usr/sbin/tcpd  /usr/sbin/in.ntalkd
#
# Mail, news and uucp services.
#
smtp    stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.smtpd
#nntp   stream  tcp     nowait  news    /usr/sbin/tcpd  /usr/sbin/in.nntpd
#uucp   stream  tcp     nowait  uucp    /usr/sbin/tcpd  /usr/lib/uucp/uucico
#comsat dgram   udp     wait    root    /usr/sbin/tcpd  /usr/sbin/in.comsat
#
# Pop et al
#
#pop-2  stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.pop2d
#pop-3  stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.pop3d
#
# `cfinger' is for the GNU finger server available for Debian.  (NOTE: The
# current implementation of the `finger' daemon allows it to be run as `root'.)
#
#cfinger stream tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.cfingerd
#finger stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.fingerd
#netstat        stream  tcp     nowait  nobody  /usr/sbin/tcpd  /bin/netstat
#systat stream  tcp     nowait  nobody  /usr/sbin/tcpd  /bin/ps -auwwx
#
# Tftp service is provided primarily for booting.  Most sites
# run this only on machines acting as "boot servers."
```

```
#
#tftp   dgram   udp     wait    nobody  /usr/sbin/tcpd  /usr/sbin/in.tftpd
#tftp   dgram   udp     wait    nobody  /usr/sbin/tcpd  /usr/sbin/in.tftpd /boot
#bootps dgram   udp     wait    root    /usr/sbin/bootpd        bootpd -i -t 120
#
# Kerberos authenticated services (these probably need to be corrected)
#
#klogin         stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rlogind -k
#eklogin        stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rlogind -k -x
#kshell         stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/in.rshd -k
#
# Services run ONLY on the Kerberos server (these probably need to be corrected)
#
#krbupdate      stream tcp      nowait  root    /usr/sbin/tcpd  /usr/sbin/registerd
#kpasswd        stream  tcp     nowait  root    /usr/sbin/tcpd  /usr/sbin/kpasswdd
#
# RPC based services
#
#mountd/1       dgram   rpc/udp wait    root    /usr/sbin/tcpd  /usr/sbin/rpc.mountd
#rstatd/1-3     dgram   rpc/udp wait    root    /usr/sbin/tcpd  /usr/sbin/rpc.rstatd
#rusersd/2-3    dgram   rpc/udp wait    root    /usr/sbin/tcpd  /usr/sbin/rpc.rusersd
#walld/1        dgram   rpc/udp wait    root    /usr/sbin/tcpd  /usr/sbin/rpc.rwalld
#
# End of inetd.conf.
ident           stream  tcp     nowait  nobody  /usr/sbin/identd        identd -i
```

5.9 다른 잡다한 네트워 관련 설정 파일들

리눅스에는 여러분이 관심있어 할 네트워 관련 잡다한 설정 파일들이 많이 있다. 여러분은 이 파일을 수정할 필요가 없을지도 모르나 안에 들어있는 것이 무언지 알려주기 위해서 설명할 가치가 있다.

5.9.1 /etc/protocols

/etc/protocols 파일은 프로토콜 id 번호를 프로토콜 이름과 연결짓는 데이타베이스다. 이 파일은 프로그래머들이 자신의 프로그램 안에서 프로토콜을 이름을 가지고 명시하기 위해 사용되고 또는 *tcpdump*같은 프로그램에서 결과에 프로토콜 번호 대신 이름을 사용하기 위해 쓰인다. 일반적인 문법은 다음과 같다.

```
protocolname  number  aliases
```

Debian 배포본에 들어있는 /etc/protocols 파일은 아래와 같다.

```
# /etc/protocols:
# $Id: protocols,v 1.1 1995/02/24 01:09:41 imurdock Exp $
#
# Internet (IP) protocols
#
#       from: @(#)protocols     5.1 (Berkeley) 4/17/89
#
# Updated for NetBSD based on RFC 1340, Assigned Numbers (July 1992).

ip      0       IP              # internet protocol, pseudo protocol number
icmp    1       ICMP            # internet control message protocol
```

```
igmp    2       IGMP            # Internet Group Management
ggp     3       GGP             # gateway-gateway protocol
ipencap 4       IP-ENCAP        # IP encapsulated in IP (officially ``IP'')
st      5       ST              # ST datagram mode
tcp     6       TCP             # transmission control protocol
egp     8       EGP             # exterior gateway protocol
pup     12      PUP             # PARC universal packet protocol
udp     17      UDP             # user datagram protocol
hmp     20      HMP             # host monitoring protocol
xns-idp 22      XNS-IDP         # Xerox NS IDP
rdp     27      RDP             # "reliable datagram" protocol
iso-tp4 29      ISO-TP4         # ISO Transport Protocol class 4
xtp     36      XTP             # Xpress Tranfer Protocol
ddp     37      DDP             # Datagram Delivery Protocol
idpr-cmtp       39      IDPR-CMTP       # IDPR Control Message Transport
rspf    73      RSPF            # Radio Shortest Path First.
vmtp    81      VMTP            # Versatile Message Transport
ospf    89      OSPFIGP         # Open Shortest Path First IGP
ipip    94      IPIP            # Yet Another IP encapsulation
encap   98      ENCAP           # Yet Another IP encapsulation
```

5.9.2 /etc/networks

/etc/networks 파일은 /etc/hosts 파일과 비슷한 역할을 한다. 이 파일은 네트웍 주소를 네트웍 이름과 연관 짓는 간단한 데이타베이스를 제공한다. 한 줄에 단 두 개의 필드만을 갖는다는 점이 /etc/hosts와 다르며 각 필드는 아래처럼 사용된다.

```
networkname networkaddress
```

다음은 한 예이다.

```
loopnet    127.0.0.0
localnet   192.168.0.0
amprnet    44.0.0.0
```

여러분이 *route* 명령을 내렸을 때 목적지가 네트웍이고 /etc/networks 파일 안에 그 네트웍에 해당하는 엔트리가 있다면 route 명령은 네트웍을 주소 대신 이름으로 출력할 것이다.

5.10 네트웍 보안과 접근 제어

여러분의 머신과 네트웍을 악의에 찬 공격으로부터 안전히 보호하는 일은 매우 복잡한 기술이라는 것을 먼저 주지시키면서 이 내용을 시작코자 한다. 나는 내 자신을 이 분야에서 전문가라 생각치 않으며 내가 말할 아래의 기법들이 도움이 될 지라도 여러분이 보안에 더욱 신중하다면 이 분야를 좀 더 공부할 것을 추천한다. 이 분야에 관한 매우 좋은 참조자료들이 인터넷 상엔 매우 많으며 그 중 하나는 Security-HOWTO이다.

중요한 기본법칙은 '**사용하지 않을 서버는 돌리지 말아라.**' 이다. 많은 배포본들이 모든 서비스가 설정되어 자동으로 실행되도록 되어 있다. 최소한의 보안 등급을 만족시키려면 여러분은 /etc/inetd.conf 파일로 가서 사용하지 않을 서비스에 해당하는 엔트리엔 주석 처리 (*줄의 맨 앞에 '#'를 붙인다*) 를 해라. shell, login, exec, uucp, ftp와 finger, netstat, systat같은 정보 제공용 서비스들이 그런 예들이다.

보안과 접근 제어 기법이 많이 있으며 그 기본적인 사항만을 설명할 것이다.

5.10.1 /etc/ftpusers

/etc/ftpusers 파일은 특정 사용자들이 ftp를 통해 여러분의 머신에 들어오는 것을 제한하도록 하는 간단한 기법이다. 이 /etc/ftpusers 파일은 새 ftp 연결이 들어온 후 ftp 데몬 프로그램 (*ftpd*)이 실행될 때 읽혀진다. 이 파일은 들어오지 못하도록 할 사용자들의 간단한 목록이며 아래처럼 생겼다.

```
# /etc/ftpusers - users not allowed to login via ftp
root
uucp
bin
mail
```

5.10.2 /etc/securetty

/etc/securetty 파일은 root가 로그온이 가능한 tty 장치들을 명시한다. /etc/securetty 파일은 로긴 프로그램 (보통 */bin/login*) 에 의해 읽혀진다. 형식은 허가될 tty 장치 이름들의 목록이며 다른 장치들에선 root의 로긴이 불허된다.

```
# /etc/securetty - tty's on which root is allowed to login
tty1
tty2
tty3
tty4
```

5.10.3 *tcpd*의 호스트 접근 제어 기법

/etc/inetd.conf 안에서 많이 봤을 *tcpd* 프로그램은 이 프로그램이 보호하도록 설정된 서비스로의 로긴과 접근을 제어한다.

*inetd*프로그램이 실행될 때 *tcpd*이 보호하는 서버로의 접근을 허가할지 불허할지에 대한 규칙을 담고 있는 두 개의 파일을 읽어들인다.

서비스에 해당하는 룰이 처음 나타날 때까지 파일들을 검색하고 알맞는 규칙이 없으면 접근은 누구에게나 허가된다. 파일을 검색하는 순서는 /etc/hosts.allow , /etc/hosts.deny 순이다. 이 두 개를 차례대로 설명할 것이다. 이 기능을 완벽히 쓰기 위해선 적당한 *man* 페이지를 참조해야 한다 (hosts_access(5)는 좋은 시작점이다).

/etc/hosts.allow /etc/hosts.allow 파일은 */usr/sbin/tcpd* 프로그램의 설정 파일이다. hosts.allow 파일은 여러분의 머신의 서비스에 연결이 *허용되*는 호스트들을 나타내는 규칙을 포함하고 있다.

이 파일의 형식은 매우 간단하다.

```
# /etc/hosts.allow
#
# <service list>: <host list> [: command]
```

service list

이 규칙이 적용되는 서버 이름을 콤마로 구분해 놓은 목록이다. 서버 이름은 ftpd, telnetd나 fingerd 같은 것들이다.

`host list`

콤마로 구분된 호스트 이름들의 목록이다. 여기에 IP 주소를 쓸 수도 있다. 추가적으로 호스트 이름을 명시하거나 한 그룹의 호스트을 나타내기 위해 와일드카드 문자를 사용한 주소를 쓸 수도 있다. 예를 들어 `gw.vk2ktj.ampr.org`는 특정 호스트를, `.uts.edu.au`는 이 문자열로 끝나는 모든 호스트 이름들을, `44.`는 이 숫자로 시작하는 모든 IP 주소를 나타낸다. 설정을 간편하게 하기 위한 특정 기호가 있는데, `LOCAL`은 '`.`'를 이름 안에 포함하지 않는 모든 호스트 다시말해 여러분의 호스트와 같은 도메인 상에 있는 모든 호스트를 나타내며 `PARANOID`는 이름과 주소가 일치하지 않는 모든 호스트 (name spoofing)를 나타낸다. 매우 유용한 마지막 기호가 있다. `EXCEPT`는 예외 목록을 나타낼 수 있도록 해준다. 이는 뒤의 예제에서 다시 다뤄질 것이다.

`command`

이 매개변수는 선택사항이다. 이 변수는 이 규칙이 적용될 때마다 실행될 명령의 완전한 경로이름이다. 예를 들어 호스트에 접속한 사람을 판별하기 위한 명령을 실행할 수도 있고 누군가 접속을 시도할 때 관리자에게 경고 등의 메시지나 메일을 보내도록 할 수도 있다. 사용 가능한 많은 확장 기능이 있으며 흔히 쓰이는 것들은 접속한 호스트의 이름이나 주소를 나타내는 `%h`와 불려지는 데몬 이름을 나타내는 `%d` 등이다.

예제

```
# /etc/hosts.allow
#
# Allow mail to anyone
in.smtpd: ALL
# All telnet and ftp to only hosts within my domain and my host at home.
telnetd, ftpd: LOCAL, myhost.athome.org.au
# Allow finger to anyone but keep a record of who they are.
fingerd: ALL: (finger @%h | mail -s "finger from %h" root)
```

/etc/hosts.deny `/etc/hosts.deny` 파일은 */usr/sbin/tcpd* 프로그램의 설정 파일이다. `hosts.deny` 파일은 여러분의 머신의 서비스에 접속을 *불허하*는 호스트들을 나타내는 규칙을 가지고 있다.

간단한 예는 아래와 같다.

```
# /etc/hosts.deny
#
# Disallow all hosts with suspect hostnames
ALL: PARANOID
#
# Disallow all hosts.
ALL: ALL
```

다른 엔트리가 모든 경우에 모든 것을 불허하기 때문에 사실상 `PARANOID` 엔트리는 중복으로 쓰인 것이다. 이 둘중 하나는 여러분의 특별한 요구에 따라서 합리적인 기본 설정이 될 것이다.

`/etc/hosts.deny` 안에 `ALL: ALL`를 기본으로 넣어 놓고 원하는 서비스와 호스트들을 `/etc/hosts.allow` 파일 안에서 명시적으로 허가하는 것이 가장 안전한 설정법이다.

5.10.4 /etc/hosts.equiv

`hosts.equiv` 파일은 특정 호스트와 사용자에게 암호 없이 여러분의 머신의 계정에 접속할 권한을 주기 위해 사용된다. 이는 여러분이 모든 머신을 제어하는 안전한 환경에선 유용하지만 그렇지 않다면 보안의 헛점이 될

수 있다. 여러분의 머신은 여러분이 신뢰하는 호스트들 중 가장 안전하지 못한 것 만큼만 안전하다. 보안을 최대화하기 위해선 이 메카니즘을 사용하지 말고 사용자들에게 .rhosts 파일 또한 쓰지 말 것을 권하라.

5.10.5 *ftp* 데몬을 알맞게 설정하기

많은 사이트들이 다른 사람들에게 특정 사용자 ID 없이 파일을 올리고 받도록 하기 위해 어노니머스(anonymous) *ftp* 서버를 돌리는 것에 관심이 있다. 만약 여러분이 이를 하기로 했다면 어노니머스 접근에 대해 *ftp* 데몬을 알맞게 설정하도록 주의해야 한다. ftpd(8)에 대한 대부분의 *man* 페이지들이 이에 대한 방법을 어느 정도 설명하고 있으며 항상 거기에 나온 지시사항들을 따라야 한다. 중요한 팁은 어노니머스 계정의 /etc 디렉토리 안에 /etc/passwd 파일의 복사본을 넣어놓지 말라는 것과 정말 사용해야 하는 계정을 제외한 나머지 것들을 다 빼버리라는 것이다. 그렇지 않으면 brute force 패스워드 크랙 기법에 노출될 위험이 있다.

5.10.6 네트웍 방화벽

데이타그램이 여러분의 머신이나 서버에 도달조차 하지 못하도록 하는 것은 매우 훌륭한 보안 방법이다. 이는 Firewall-HOWTO와 이 문서의 뒷부분에서 좀 더 자세히 다뤄질 것이다.

5.10.7 다른 사항들

고려해야 할 다른 문제들.

sendmail

널리 쓰임에도 불구하고 *sendmail* 데몬은 보안 위험 보고에 매우 자주 등장한다. 모든건 여러분에게 달려 있지만 나는 이 데몬을 돌리지 않기로 했다.

NFS and other Sun RPC services

매우 주의깊게 이들을 살펴봐야 한다. 이 서비스들에는 가능한 모든 종류의 보안 구멍이 있다. NFS 같은 서비스의 옵션들을 찾는 것이 어렵지만 이를 설정하게 된다면 누구에게 mount 권한을 부여할 것인지 매우 신중하라.

6 IP 와 이더넷 관련 정보

이 부분은 이더넷과 IP 에 관련된 정보를 다룰 것이다. 여기의 소 분류들은 그것들이 소위 "특정 기술에 대한" 부분에서 가장 흥미있는 것들이라 생각했기 때문에 나누었다. LAN을 사용하는 사람은 누구나 이 부분들에서 얻을 것이 있을 것이다.

6.1 Ethernet

이더넷 장치의 이름은 'eth0', 'eth1', 'eth2' 등이다. 커널에 의해 처음 발견되는 카드가 'eth0'을 부여받고 나머지는 발견되는 순서대로 하나씩 부여받는다.

기본적으로 커널은 하나의 이더넷 장치만을 찾으며 다른 장치를 찾기 위해선 직접 특정 명령을 커널에 내려줘야 한다.

리눅스 상에서 이더넷 카드를 작동하도록 하는 법을 알고 싶으면 Ethernet-HOWTO를 참조하라.

커널이 이더넷 카드를 지원하도록 알맞게 만들어진 후에는 카드 설정은 쉽다.

보통 여러분은 아래와 같은 것을 쓰며 대부분의 배포본은 이더넷 카드를 지원하도록 설정을 했다면 이를 이미 해준다.

```
root# ifconfig eth0 192.168.0.1 netmask 255.255.255.0 up
root# route add -net 192.168.0.0 netmask 255.255.255.0 eth0
```

대부분의 이더넷 드라이버들은 Donald Becker, becker@CESDIS.gsfc.nasa.gov에 의해 만들어졌다.

6.2 EQL - multiple line traffic equaliser

EQL 장치 이름은 'eql'이다. 표준 커널 소스로는 머신당 한개의 EQL 장치를 가질 수 있다. EQL은 PPP나 slip, plip같은 point-to-point 다중 연결을 한 개의 논리적인 tcp/ip 연결로 사용할 수 있도록 해준다. 보통은 하나의 고속 연결을 만드는 것보다 저속의 다중 연결을 사용하는 편이 비용이 싸다.

Kernel Compile Options:

```
Network device support  --->
    [*] Network device support
    <*> EQL (serial line load balancing) support
```

이 기술을 사용하기 위해선 연결의 반대편에 있는 머신도 EQL을 지원해야 한다. 리눅스와 Livingstone Portmasters, 새로운 dial-in 서버들은 호환 가능한 기능을 지원한다.

EQL을 설정하기 위해선 metalab.unc.edu 에서 얻을 수 있는 eql 툴들이 필요할 것이다.

설정은 매우 직설적이다. eql 인터페이스의 설정으로 시작한다. eql 인터페이스는 다른 네트웍 장치들과 비슷하다. *ifconfig*를 이용하여 IP 주소와 mtu를 설정하며 아래와 같다.

```
root# ifconfig eql 192.168.10.1 mtu 1006
```

그다음 여러분이 사용할 각 줄들을 직접 시작할 필요가 있다. 이는 point-to-point 네트웍 장치의 경우와 마찬가지이다. 연결을 시작하는 방법은 연결의 종류에 따라 다르며 자세한 사항은 문서의 해당 부분을 참조하여라.

마지막으로 serial 연결을 EQL 장치와 연결시켜야 한다. 이는 'enslaving'이라 불리며 아래와 같은 *eql_enslave* 명령으로 수행된다.

```
root# eql_enslave eql sl0 28800
root# eql_enslave eql ppp0 14400
```

*eql_enslave*에 넘겨주는 '*estimated speed*' 매개변수는 직접은 아무 것도 하지 않는다. 이는 EQL 드라이버가 장치가 받을 데이타그램의 할당을 결정하며 이 값을 가지고 연결들의 밸런스를 조정할 수 있다.

연결을 EQL 장치로부터 떼어내기 위해선 아래처럼 *eql_emancipate* 명령을 쓴다.

```
root# eql_emancipate eql sl0
```

다른 point-to-point 연결처럼 라우팅을 추가해주는데 실제 직렬 장치 대신 eql 장치를 라우터가 참조하는 것이 다르다. 보통 아래처럼 쓴다.

```
root# route add default eql
```

EQL 드라이버는 Simon Jones, simon@ncm.com 에 의해 개발되었다.

6.3 IP Accounting (for Linux-2.0)

리눅스 커널의 IP accounting 기능은 네트웍을 사용하는 자료들을 모아서 분석할 수 있도록 해준다. 모아진 자료들은 초기화 이후 누적된 패킷과 바이트의 수들로 이루어진다. 사용 목적에 따라 이들을 분류할 수 있는 다양한 규칙들을 사용할 수 있다. 이 옵션은 이전의 ipfwadm 기반의 방화벽 기능이 "ipfwchains"로 대체되면서 커널 2.1.102때부터 빠졌다.

Kernel Compile Options:

```
Networking options  --->
    [*] IP: accounting
```

IP accounting을 설정하기 위해선 커널을 컴파일하여 설치한 후 *ipfwadm* 명령을 사용해야 한다. 선택한 accounting 정보를 자세히 분류하는 많은 방법들이 있다. 여기에선 유용하게 사용할 수 있는 간단한 예제를 골랐으며 더 자세한 정보를 위해선 *ipfwadm* 맨 페이지를 읽어보길 바란다.

시나리오: 여러분은 PPP 연결을 통해 인터넷에 연결된 이더넷 네트웍을 가지고 있다. 이더넷 상에서 많은 서비스를 하는 머신을 가지고 있으며 전체적인 tcp, udp 트래픽 뿐 아니라 ftp나 www 등의 각 서비스에 의한 트래픽도 알고 싶다.

여러분은 아래의 쉘 스크립트 처럼 보이는 명령들을 사용해야 한다.

```
#!/bin/sh
#
# Flush the accounting rules
ipfwadm -A -f
#
# Set shortcuts
localnet=44.136.8.96/29
any=0/0
# Add rules for local ethernet segment
ipfwadm -A in  -a -P tcp -D $localnet ftp-data
ipfwadm -A out -a -P tcp -S $localnet ftp-data
ipfwadm -A in  -a -P tcp -D $localnet www
ipfwadm -A out -a -P tcp -S $localnet www
ipfwadm -A in  -a -P tcp -D $localnet
ipfwadm -A out -a -P tcp -S $localnet
ipfwadm -A in  -a -P udp -D $localnet
ipfwadm -A out -a -P udp -S $localnet
#
# Rules for default
ipfwadm -A in  -a -P tcp -D $any ftp-data
ipfwadm -A out -a -P tcp -S $any ftp-data
ipfwadm -A in  -a -P tcp -D $any www
ipfwadm -A out -a -P tcp -S $any www
ipfwadm -A in  -a -P tcp -D $any
ipfwadm -A out -a -P tcp -S $any
ipfwadm -A in  -a -P udp -D $any
ipfwadm -A out -a -P udp -S $any
#
# List the rules
ipfwadm -A -l -n
#
```

"ftp-data"와 "www' 같은 이름들은 `/etc/services` 안의 줄들을 가리킨다. 마지막 명령은 각 Accounting 규칙들과 모아진 결과를 출력한다.

IP accounting을 분석할 때 주의해야 할 것은 **적용되는 모든 규칙의 결과가 합해진다는 것**이다. 따라서 원하는 값을 얻기 위해선 적절한 산술 연산이 필요하다. 예를 들어 ftp도 www도 아닌 자료의 양을 알길 원한다면 각 결과들을 모든 포트에 적용되는 결과에서 뺄 것이다.

```
root# ipfwadm -A -l -n
IP accounting rules
 pkts bytes dir prot source               destination          ports
    0     0 in  tcp  0.0.0.0/0            44.136.8.96/29       * -> 20
    0     0 out tcp  44.136.8.96/29       0.0.0.0/0            20 -> *
   10  1166 in  tcp  0.0.0.0/0            44.136.8.96/29       * -> 80
   10   572 out tcp  44.136.8.96/29       0.0.0.0/0            80 -> *
  252 10943 in  tcp  0.0.0.0/0            44.136.8.96/29       * -> *
  231 18831 out tcp  44.136.8.96/29       0.0.0.0/0             * -> *
    0     0 in  udp  0.0.0.0/0            44.136.8.96/29       * -> *
    0     0 out udp  44.136.8.96/29       0.0.0.0/0            * -> *
    0     0 in  tcp  0.0.0.0/0            0.0.0.0/0            * -> 20
    0     0 out tcp  0.0.0.0/0            0.0.0.0/0            20 -> *
   10  1166 in  tcp  0.0.0.0/0            0.0.0.0/0            * -> 80
   10   572 out tcp  0.0.0.0/0            0.0.0.0/0            80 -> *
  253 10983 in  tcp  0.0.0.0/0            0.0.0.0/0            * -> *
  231 18831 out tcp  0.0.0.0/0            0.0.0.0/0            * -> *
    0     0 in  udp  0.0.0.0/0            0.0.0.0/0            * -> *
    0     0 out udp  0.0.0.0/0            0.0.0.0/0            * -> *
```

6.4 IP Accounting (for Linux-2.2)

새 accounting 코드는 "IP Firewall Chains"를 통해 구현된다. `IPchains`를 보면 더 많은 정보를 얻을 수 있다. 무엇보다도 여러분의 필터를 설정하기 위해선 `ipfwadm` 대신에 *ipchains*를 사용하게 될 것이다. (최신 커널 소스의 `Documentation/Changes`로부터)

6.5 IP Aliasing

한 네트웍 장치에 둘 이상의 IP 주소를 부여할 수 있는 프로그램들이 있으며 이는 꽤 유용하다. ISP (Internet Service Provider)들은 종종 사용자들에게 특화된 World Wide Web과 ftp를 제공하기 위해 이 기능을 사용한다. 여러분은 여기에 나온 것보다 더 많은 정보를 "IP-Alias mini-HOWTO"에서 얻을 수 있다.

Kernel Compile Options:

```
        Networking options  --->
            ....
            [*] Network aliasing
            ....
            <*> IP: aliasing support
```

커널을 IP_Alias 지원과 함께 컴파일해서 설치한 다음엔 설정은 매우 간단하다. alias들은 실제 네트웍 장치와 연관된 가상의 네트웍 장치들에 추가된다. 이 장치들에는 단순한 명명법이 쓰이는데 `<devname>:<virtual dev num>` 과 같은 식으로 예를 들어 `eth0:0`, `ppp0:10` 등이다. 이 ifname:number 장치는 주 인터페이스가 설정된 *후에만* 설정될 수 있다는 것을 명심해라.

예를 들어 여러분이 두 개의 서로 다른 IP 서브네트웍을 동시에 지원하는 이더넷 네트웍을 가지고 있고 여러분의 머신이 양 쪽을 모두 직접 접근할 수 있도록 하고 싶다면 다음과 같이 해주면 된다.

```
root# ifconfig eth0 192.168.1.1 netmask 255.255.255.0 up
root# route add -net 192.168.1.0 netmask 255.255.255.0 eth0

root# ifconfig eth0:0 192.168.10.1 netmask 255.255.255.0 up
root# route add -net 192.168.10.0 netmask 255.255.255.0 eth0:0
```

alias를 지우기 위해선 단순히 이름 뒤에 '-'를 붙여주면 된다.

```
root# ifconfig eth0:0- 0
```

alias에 관련된 모든 route 정보들도 자동으로 지워질 것이다.

6.6 IP Firewall (for Linux-2.0)

IP Firewall과 방화벽 문제는 `Firewall-HOWTO`에서 좀 더 깊게 다뤄질 것이다. IP Firewalling은 여러분이 지정한 IP 주소로 가는 혹은 그 주소에서 오는 데이타그램들을 걸러내거나 통과시킴으로써 허가되지 않은 네트웍 접근으로부터 여러분의 머신을 보호할 수 있도록 해준다. 세 개의 다른 규칙 등급이 적용되는데 incoming filtering과 outgoing filtering, forwarding filtering이다. Incoming rule은 네트웍 장치가 받아들이는 데이타그램에 적용되며 Outgoing rule은 네트웍 장치를 통해 나가는 데이타그램에, Forwarding rule은 받아들여지긴 하나 이 머신으로 오지 않은, 다시말해 이 머신을 거쳐 다른 곳으로 가는 데이타그램에 적용된다.

Kernel Compile Options:

```
Networking options  --->
    [*] Network firewalls
    ....
    [*] IP: forwarding/gatewaying
    ....
    [*] IP: firewalling
    [ ] IP: firewall packet logging
```

IP firewall 규칙들의 설정은 *ipfwadm* 명령을 사용하여 이루어진다. 앞에서 언급한 것처럼 나는 보안 전문가가 아니며 따라서 보안이 여러분에게 중요한 문제라면 내가 간단한 예제를 보여주는 것과는 별도로 여러분 스스로 규칙들을 공부하고 개발해야 한다.

IP firewall을 쓰는 대부분의 경우는 아마도 리눅스 머신을 라우터나 방화벽의 게이트웨이(gateway)로 사용하여 여러분의 로칼 네트웍을 외부의 허가되지 않은 접근으로부터 보호하고자 하는 경우일 것이다.

아래의 설정은 Arnt Gulbrandsen, <`agulbra@troll.no`>의 도움에 기반하고 있다.

아래의 예제는 밑의 그림에 나온 것처럼 리눅스 firewall/router 머신 위에 방화벽을 설정했을 경우이다.

```
-                                   -
 \                                  | 172.16.37.0
  \                                 |   /255.255.255.0
   \                 ---------      |
    |  172.16.174.30 | Linux |      |
NET =================|  f/w  |------|    ..37.19
```

```
       |    PPP        | router|      |  --------
      /                ---------      |--| Mail |
     /                                |  | /DNS |
    /                                 |  --------
   -                                  -
```

아래 명령들은 보통 rc 파일 안에 들어가서 시스템이 부팅될 때마다 자동으로 시작되도록 되어있다. 보안을 최대로 하기 위해선 인터페이스가 설정된 후에, 그러나 방화벽 머신이 리부팅되는 동안 누군가 접근 권한을 얻는 것을 막기 위해 인터페이스가 실제로 작동되기 이전에 아래 명령들이 실행되야 한다.

```
#!/bin/sh

# Flush the 'Forwarding' rules table
# Change the default policy to 'accept'
#
/sbin/ipfwadm -F -f
/sbin/ipfwadm -F -p accept
#
# .. and for 'Incoming'
#
/sbin/ipfwadm -I -f
/sbin/ipfwadm -I -p accept

# First off, seal off the PPP interface
# I'd love to use '-a deny' instead of '-a reject -y' but then it
# would be impossible to originate connections on that interface too.
# The -o causes all rejected datagrams to be logged. This trades
# disk space against knowledge of an attack of configuration error.
#
/sbin/ipfwadm -I -a reject -y -o -P tcp -S 0/0 -D 172.16.174.30

# Throw away certain kinds of obviously forged packets right away:
# Nothing should come from multicast/anycast/broadcast addresses
#
/sbin/ipfwadm -F -a deny -o -S 224.0/3 -D 172.16.37.0/24
#
# and nothing coming from the loopback network should ever be
# seen on a wire
#
/sbin/ipfwadm -F -a deny -o -S 127.0/8 -D 172.16.37.0/24

# accept incoming SMTP and DNS connections, but only
# to the Mail/Name Server
#
/sbin/ipfwadm -F -a accept -P tcp -S 0/0 -D 172.16.37.19 25 53
#
# DNS uses UDP as well as TCP, so allow that too
# for questions to our name server
#
/sbin/ipfwadm -F -a accept -P udp -S 0/0 -D 172.16.37.19 53
#
# but not "answers" coming to dangerous ports like NFS and
# Larry McVoy's NFS extension.  If you run squid, add its port here.
#
/sbin/ipfwadm -F -a deny -o -P udp -S 0/0 53 \
```

```
        -D 172.16.37.0/24 2049 2050

# answers to other user ports are okay
#
/sbin/ipfwadm -F -a accept -P udp -S 0/0 53 \
        -D 172.16.37.0/24 53 1024:65535

# Reject incoming connections to identd
# We use 'reject' here so that the connecting host is told
# straight away not to bother continuing, otherwise we'd experience
# delays while ident timed out.
#
/sbin/ipfwadm -F -a reject -o -P tcp -S 0/0 -D 172.16.37.0/24 113

# Accept some common service connections from the 192.168.64 and
# 192.168.65 networks, they are friends that we trust.
#
/sbin/ipfwadm -F -a accept -P tcp -S 192.168.64.0/23 \
        -D 172.16.37.0/24 20:23

# accept and pass through anything originating inside
#
/sbin/ipfwadm -F -a accept -P tcp -S 172.16.37.0/24 -D 0/0

# deny most other incoming TCP connections and log them
# (append 1:1023 if you have problems with ftp not working)
#
/sbin/ipfwadm -F -a deny -o -y -P tcp -S 0/0 -D 172.16.37.0/24

# ... for UDP too
#
/sbin/ipfwadm -F -a deny -o -P udp -S 0/0 -D 172.16.37.0/24
```

좋은 방화벽 설정은 복잡하다. 이 예제는 여러분에게 어느 정도의 시작점만을 보여준다. *ipfwadm* 메뉴얼 페이지가 이 툴을 사용하는 데에 도움이 될 것이다. 방화벽을 설정하고자 한다면 믿을만한 곳에 물어보고 많은 충고를 받아라. 또한 외부에서 여러분의 설정을 검사하도록 해라.

6.7 IP Firewall (for Linux-2.2)

새로운 방화벽 코드가 "IP Firewall Chains"를 통해 구현된다. 더 많은 정보를 위해선 IPchains를 보라. 지금 여러분의 필터를 설정하기 위해선 ipfwadm 대신 *ipchains*를 사용해야 한다는 것을 알아야 한다. (최신 커널의 Documentation/Changes 참조)

우리도 이 내용이 최신과 거리가 멀다는 것을 알고 있으며 좀 더 최신 내용을 다루기 위해 현재 작업중이다. 아마 1999년 8월의 새 버전에선 다뤄질 것이다. (역자주: 현재 번역중인 이 문서가 1999년 8월 버젼이다-_-;)

6.8 IPIP Encapsulation

왜 여러분은 IP 데이타그램 내에 IP 데이타그램을 집어넣기를 원하나? 이런 방식을 이전에 본 적이 없다면 이상한 것으로 보일 것이다. Mobile-IP와 IP-Multicast가 이 방식이 사용되는 흔한 예이다. 또한 잘 알려져 있진 않지만 이 방식을 제일 많이 쓰는 곳은 Amateur Radio이다.

Kernel Compile Options:

```
Networking options  --->
    [*] TCP/IP networking
    [*] IP: forwarding/gatewaying
    ....
    <*> IP: tunneling
```

IP tunnel 장치는 'tunl0', 'tunl1' 등으로 불린다.

일반적인 IP 라우팅 규칙은 IP 네트웍이 네트웍 주소와 네트웍 마스크로 이루어져 있도록 요구한다. 이는 한 라우팅 엔트리를 통해 일련의 연속된 주소들이 모두 라우팅되도록 해준다. 이는 매우 편리하지만 여러분이 네트웍의 특정 부분에 연결되어 있는 동안 특정 IP 주소만을 사용해야 한다는 것을 의미한다. 대부분의 경우 문제가 없으나 여러분의 mobile netizen이라면 계속 한 장소에 접속되어 있을 순 없을 것이다. IP/IP encapsulation (IP tunneling)은 여러분의 IP 주소로 오도록 되어 있는 데이타그램이 한 번 더 쌓여져서 다른 IP 주소로 돌려지도록 하므로써 제한을 극복할 수 있게 해준다. 만약 여러분이 다른 IP 네트웍에서 작업을 할 예정이라면 원래 네트웍의 머신이 여러분의 IP 주소로 오는 데이타그램을 받아서 임시로 현재 사용하는 주소로 돌려주도록 설정할 수 있다.

6.8.1 A tunneled network configuration.

```
 192.168.1/24                          192.168.2/24

    -                                     -
    |      ppp0 =            ppp0 =       |
    |  aaa.bbb.ccc.ddd  fff.ggg.hhh.iii   |
    |                                     |
    |   /-----\                /-----\    |
    |   |     |       //       |     |    |
    |---|  A  |------//---------|  B  |---|
    |   |     |     //         |     |    |
    |   \-----/                \-----/    |
    |                                     |
    -                                     -
```

이 그림은 IPIP encapsulation을 쓰는 다른 마땅한 이유, 가상 개별 네트워킹을 나타낸다. 이 예제는 여러분이 기본적인 다이얼업 인터넷 연결을 가진 두 머신을 가지고 있다고 가정한다. 각 호스트는 한 개의 IP 주소를 부여받았다. 이 머신들 뒤에는 예약된 IP 네트웍 주소로 설정된 로칼 네트웍이 존재한다. 네트웍 라우터로 인터넷에 연결되는 것처럼 A 네트웍에 있는 호스트들이 B 네트웍 위의 호스트들에 접속할 수 있도록 하길 원한다고 해보자. IPIP encapsulation이 이를 가능하게 해준다. 주의할 것은 encapsulation이 A와 B 네트웍 상의 호스트들이 인터넷 상에서 서로 통신할 수 있도록 하는 문제를 해결하는 것이 아니며 여전히 이를 위해선 IP Masquerade 같은 기법을 사용해야 한다는 것이다. encapsulation은 보통 라우터로써의 머신 기능에 의해 구현된다.

리눅스 라우터 'A'는 아래와 같은 스크립트로 설정될 것이다.

```
#!/bin/sh
PATH=/sbin:/usr/sbin
mask=255.255.255.0
remotegw=fff.ggg.hhh.iii
#
# Ethernet configuration
ifconfig eth0 192.168.1.1 netmask $mask up
route add -net 192.168.1.0 netmask $mask eth0
#
```

```
# ppp0 configuration (start ppp link, set default route)
pppd
route add default ppp0
#
# Tunnel device configuration
ifconfig tunl0 192.168.1.1 up
route add -net 192.168.2.0 netmask $mask gw $remotegw tunl0
```

리눅스 라우터 'B'는 아래와 같은 스크립트로 설정될 것이다.

```
#!/bin/sh
PATH=/sbin:/usr/sbin
mask=255.255.255.0
remotegw=aaa.bbb.ccc.ddd
#
# Ethernet configuration
ifconfig eth0 192.168.2.1 netmask $mask up
route add -net 192.168.2.0 netmask $mask eth0
#
# ppp0 configuration (start ppp link, set default route)
pppd
route add default ppp0
#
# Tunnel device configuration
ifconfig tunl0 192.168.2.1 up
route add -net 192.168.1.0 netmask $mask gw $remotegw tunl0
```

명령:

```
route add -net 192.168.1.0 netmask $mask gw $remotegw tunl0
```

이는 '192.168.1.0/24로 향하는 데이타그램들을 IPIP 캡슐화 데이타그램 속에 넣어서 aaa.bbb.ccc.ddd'의 주소로 보낸다'는 것을 뜻한다.

설정은 양 측에서 일치해야 한다. tunnel 장치는 라우팅시에 자신이 라우팅하도록 받은 데이타그램을 집어넣을 IP 데이타그램의 *목적지*로 'gw'를 사용한다. 그 머신 역시 IPIP 데이타그램을 decapsulate하는 법을 알아야 하며 다시말해 tunnel 장치가 설정되어 있어야 한다.

6.8.2 A tunneled host configuration.

라우팅 하는 것이 전체 네트웍일 필욘 없으며 단 하나의 IP 주소만을 라우팅할 수 있다. 이럴 경우 '원격' 머신의 tunl 장치는 머신의 IP 주소로 성절해야 하며 A 쪽에선 tunnel 장치를 통한 네트웍 라우팅 보다는 호스트 라우팅(그리고 Proxy Arp)를 사용해야 한다. 이에 맞도록 설정을 조정해서 알맞게 다시 그려보자. 이제 우리는 완전히 인터넷에 연결되어 있고 호스트 'A'에 의해 지원되는 원격 네트웍의 일부인 것처럼 작동하려는 호스트 'B'를 가지고 있다.

```
192.168.1/24

 -
 |       ppp0 =                 ppp0 =
```

```
|  aaa.bbb.ccc.ddd      fff.ggg.hhh.iii
|
|   /-----\                /-----\
|   |     |       //       |     |
|---|  A  |------//---------|  B  |
|   |     |     //         |     |
|   \-----/                \-----/
|                      also: 192.168.1.12
-
```

리눅스 라우터 'A'는 다음과 같이 설정된다.

```
#!/bin/sh
PATH=/sbin:/usr/sbin
mask=255.255.255.0
remotegw=fff.ggg.hhh.iii
#
# Ethernet configuration
ifconfig eth0 192.168.1.1 netmask $mask up
route add -net 192.168.1.0 netmask $mask eth0
#
# ppp0 configuration (start ppp link, set default route)
pppd
route add default ppp0
#
# Tunnel device configuration
ifconfig tunl0 192.168.1.1 up
route add -host 192.168.1.12 gw $remotegw tunl0
#
# Proxy ARP for the remote host
arp -s 192.168.1.12 xx:xx:xx:xx:xx:xx pub
```

리눅스 호스트 'B'는 다음과 같이 설정된다.

```
#!/bin/sh
PATH=/sbin:/usr/sbin
mask=255.255.255.0
remotegw=aaa.bbb.ccc.ddd
#
# ppp0 configuration (start ppp link, set default route)
pppd
route add default ppp0
#
# Tunnel device configuration
ifconfig tunl0 192.168.1.12 up
route add -net 192.168.1.0 netmask $mask gw $remotegwtunl0
```

이런 종류의 설정은 Mobile-IP 분야에서 더욱 전형적이다. 이 분야에서는 하나의 호스트가 인터넷 내부를 돌아다니면서도 내내 하나의 IP 주소를 사용하고자 한다. 실제로 이를 다루는 것에 대한 더 많은 정보를 원한다면 Mobile-IP 부분을 참고하라.

6.9 IP Masquerade

많은 사람들은 인터넷에 접속하기 위해 단순한 다이얼업 계정을 가지고 있을 뿐이다. 이런 종류의 설정을 쓰는 거의 모든 사람들은 ISP (Internet Service Provider) 로부터 단지 하나의 IP 주소만을 할당받는다. 이는 단 하나의 호스트가 네트웍에 접속할 경운 일반적으로 충분하다. IP Masquerade는 여러 호스트들이 다이얼업 연결을 하는 머신처럼 보이도록 하므로써 한 IP 주소를 여러 머신들이 쓸 수 있도록 하는 일종의 트릭이다. (역자주: masquerade는 가장 무도회, 가장하다. 의 뜻이다.) 한 작은 제한점은 masquerade 기능을 한 방향으로만 작동한다는 것이다. 다시 말해 이 기능을 사용하는 호스트는 접속을 하도록 요청을 할 수는 있지만 원격 호스트로부터의 접속을 받아들이지는 못한다는 것이다. *talk*같은 일부 네트웍 서비스는 작동하지 않으며 *ftp*같은 서비스들은 수동 (PASV) 모드로 작동하도록 설정되어야 한다는 것을 의미한다. 운이 좋게도 WWW나 *irc*, *telnet* 같은 대부분의 네트웍 서비스는 잘 작동한다.

Kernel Compile Options:

```
Code maturity level options  --->
    [*] Prompt for development and/or incomplete code/drivers
Networking options  --->
    [*] Network firewalls
    ....
    [*] TCP/IP networking
    [*] IP: forwarding/gatewaying
    ....
    [*] IP: masquerading (EXPERIMENTAL)
```

보통 여러분은 리눅스 머신에 단독(standalone) 머신일 때처럼 slip이나 ppp 연결을 지원하도록 만든다. 이에 추가로 예약된 네트웍 주소로 설정이 된 이더넷 같은 또 다른 네트웍 장치를 가지고 있을 것이다. masquerade 될 호스트들은 이 두 번째 네트웍 상에 존재한다. 각각의 이 호스트들은 위 리눅스 머신의 이더넷 포트 주소를 기본 gateway나 라우터로 설정한다.

전형적인 설정은 아래처럼 보인다.

```
-                                   -
 \                                  | 192.168.1.0
  \                                 |   /255.255.255.0
   \                 ---------      |
    |                | Linux | .1.1 |
NET =================| masq  |------|
    |    PPP/slip    | router|      |  --------
   /                 ---------      |--| host |
  /                                 |  |      |
 /                                  |  --------
-                                   -
```

IPFWADM을 이용한 Masquerading

이 설정에 가장 적절한 명령은 아래와 같다.

```
# Network route for ethernet
route add -net 192.168.1.0 netmask 255.255.255.0 eth0
#
# Default route to the rest of the internet.
route add default ppp0
#
# Cause all hosts on the 192.168.1/24 network to be masqueraded.
```

```
ipfwadm -F -a m -S 192.168.1.0/24 -D 0.0.0.0/0
```

*IPCHAINS*을 이용한 Masquerading

IPFWADM을 쓰는 경우와 비슷하나 명령 구조가 바뀌었다.

```
# Network route for ethernet
route add -net 192.168.1.0 netmask 255.255.255.0 eth0
#
# Default route to the rest of the internet.
route add default ppp0
#
# Cause all hosts on the 192.168.1/24 network to be masqueraded.
ipchains -A forward -s 192.168.1.0/24 -j MASQ
```

여러분은 IPMasqueradeResourcePage로부터 Linux IP Masquerade에 대한 더 많은 정보를 얻을 수 있다. 또한 masquerading에 대한 *매우* 자세한 문서로 "IP-Masquerade mini-HOWTO"가 있다(여기에는 다른 OS들을 Linux masquerade 서버와 함께 작동하도록 설정하는 법도 나와있다).

6.10 IP Transparent Proxy

ip transparent proxy는 다른 머신을 향하는 서비스나 서버를 이 머신의 그 서비스들로 돌릴 수 있도록 해주는 기능이다. 일반적으로 이 기능은 리눅스 머신을 라우터로 쓰는 동시에 proxy 서버로 쓰는 곳에서 유용하다. 여러분은 각 서비스들로 오는 모든 연결들을 로칼 proxy 서버로 돌릴 수 있다.

Kernel Compile Options:

```
Code maturity level options  --->
        [*] Prompt for development and/or incomplete code/drivers
Networking options  --->
        [*] Network firewalls
        ....
        [*] TCP/IP networking
        ....
        [*] IP: firewalling
        ....
        [*] IP: transparent proxy support (EXPERIMENTAL)
```

transparent proxy 특징의 설정은 *ipfwadm* 명령을 사용해서 한다.

쓸만한 예가 아래에 있다.

```
root# ipfwadm -I -a accept -D 0/0 telnet -r 2323
```

이 예는 어떤 호스트의 `telnet` 포트(23)에 들어오는 연결들을 모두 이 호스트의 2323 포트로 돌려준다. 만약 그 포트에서 서비스를 돌리고 있다면 이를 telnet 연결로 돌리거나 log를 남기거나 혹은 원하는 무엇이든 할 수 있다.

더욱 흥미있는 예는 모든 `http` 연결들을 local cache를 통하도록 돌리는 것이다. 그러나 proxy 서버가 사용하는 protocol은 보통의 http와는 다르다. 클라이언트가 http를 이용할 경우 `www.server.com:80`에

연결되어 /path/page를 요청하지만 local cache에 연결될 때는 proxy.local.domain:8080에 접속해서 www.server.com/path/page를 요청한다.

http 요청을 local proxy를 통해 걸러내기 위해선 transproxy라는 작은 서버를 삽입해서 그 프로토콜을 받아들일 필요가 있다 (이는 웹에서 쉽게 구할 수 있다). 이 transproxy를 8081 포트에서 돌도록 결정한 다음 아래 명령을 내린다.

```
root# ipfwadm -I -a accept -D 0/0 80 -r 8081
```

그러면 transproxy 프로그램은 외부 서버로 가는 모든 연결을 받은 후 프로토콜간의 상이점을 보정한 후 local proxy로 그 연결들을 보낼 것이다.

6.11 IPv6

여러분이 막 IP 네트워킹을 이해하기 시작하는 순간 규칙이 변형되었다. IPv6는 Internet Protocol의 version 6에 대한 줄임말이다. IPv6는 할당할 IP 주소의 부족 문제를 극복하기 위해 개발되었다. IPv6 주소는 16바이트(128비트)를 사용한다. IPv6는 많은 다른 수정 사항들을 포함하는데 그 중엔 IPv4 네트웍 보다 IPv6 네트웍을 다루기 쉽도록 하는 단순화도 있다.

리눅스는 이미 완전하지는 않지만 작동은 하는 IPv6를 2.2.* 버전의 커널들에서 구현해 놨다.

여러분이 이 새로운 인터넷 기술을 경험해 보고 싶거나 필요하다면 www.terra.net에서 구할 수 있는 IPv6-FAQ를 읽어보라.

6.12 Mobile IP

”IP mobility”라는 말은 한 호스트가 IP 주소의 변화나 연결 중단 없이 인터넷의 한 지점에서 다른 지점으로 연결을 이동시킬 수 있는 능력을 말한다. 일반적으로 IP 호스트가 연결 지점을 바꾸는 경우 IP 주소도 바꿔야만 한다. IP Mobility는 mobile 호스트에 고정 IP 주소를 할당하고 이 주소로 오는 데이타그램들을 실제 사용되는 IP 주소로 돌리는 자동 라우팅과 함께 IP encapsulation (tunneling)을 사용하여 이 문제를 극복한다.

리눅스에서 완전한 IP mobility 도구를 제공하기 위한 프로젝트가 진행중이다. 이 프로젝트와 도구들의 상태는 LinuxMobileIPHomePage에서 볼 수 있다.

6.13 Multicast

IP Multicast는 다른 IP 네트웍 상의 많은 IP 호스트들이 그들에게 동시에 보내지는 IP 데이타그램을 받도록 해준다. 이 기술은 오디오와 비디오 전송 등의 인터넷을 통한 방송을 위해 개발되었다.

Kernel Compile Options:

```
Networking options  --->
        [*] TCP/IP networking
        ....
        [*] IP: multicasting
```

약간의 도구와 작은 네트웍 설정이 필요하다. 리눅스 상에서의 Multicast 지원에 대한 추가 정보는 Multicast-HOWTO를 보라.

6.14 NAT - Network Address Translation

IP Network Address Translation 기능은 리눅스 IP Masquerade 기능의 훨씬 표준화된 기술이다. RFC-1631에 보다 자세히 기술되어 있다. NAT는 IP-Masquerade가 지원하지 못하는 기능들을 제공하며 이는 NAT가 방화벽이 있는 라우터나 보다 큰 규모의 환경에 더 적합하도록 해준다.

Linux 2.0.29 커널에 대한 NAT 구현의 알파 버전이 Michael.Hasenstein, `Michael.Hasenstein@informatik.tu-chemnitz.de`에 의해 개발되어 왔다. Michael의 문서와 구현은 `LinuxIPNetworkAddressWebPage` 에서 구할 수 있다.

새로운 Linux 2.2.x 커널들은 몇몇 NAT의 기능을 라우팅 알고리즘에 포함하고 있다.

6.15 Traffic Shaper - 변환 가능한 대역폭

traffic shaper는 새 인터페이스 장치를 만드는 드라이버이다. 그 장치들은 사용자의 정의에 따라 트래픽이 제한될 수 있고 실제 전송에 있어선 실제의 네트웍 장치에 의존하면서 네트웍 트래픽을 밖으로 내보내는 장치로 쓰일 수 있다.

shaper는 Linux-2.1.15에서 소개되었고 Linux-2.0.36으로 거꾸로 적용되었다 (shaper 장치의 제작자이며 Linux-2.0를 관리하는 Alan Cox가 배포한 `2.0.36-pre-patch-2`에 포함되어 있다).

traffic shaper는 모듈로만 컴파일될 수 있으며 *shapecfg* 프로그램으로 아래와 같은 방식으로 설정된다.

```
shapecfg attach shaper0 eth1
shapecfg speed shaper0 64000
```

shaper 장치는 패킷들이 라우팅 테이블에 따라 shaper를 통과하는 경우에 밖으로 나가는 트래픽의 대역폭만을 조절할 수 있다. 따라서 "출발지 주소에 의한 라우팅" 기능은 리눅스 라우터를 사용하는 특정 호스트들의 전반적인 대역폭을 제한하는 데 도움을 준다.

Linux-2.2는 이미 그런 라우팅을 지원하며 Linux-2.0에서 이 기능이 필요하다면 `ftp.invlogic.com`에서 Mike McLagan이 만든 패치를 점검해 보라. shaper에 대한 더 많은 정보는 `Documentation`networking/shaper.txt에서 얻을 수 있다.

들어오는 패킷에 대한 shaping을 시험해 보고 싶다면 `ftp.systemy.it`로부터 새 버전인 `rshaper-1.01`를 시도해 볼 수 있다.

6.16 Linux-2.2에서의 라우팅

Linux의 최신 버전인 2.2는 라우팅 정책에 있어 많은 유연성을 제공한다. 불행히도 여러분은 이 하우투 문서의 다음 버전을 기다리거나 커널 소스를 읽으러 가야한다.

7 일반적인 PC 하드웨어 사용하기

7.1 ISDN

Integrated Services Digital Network (ISDN)은 일반적인 스위칭 방식의 디지탈 데이타 네트웍을 지칭하는 일련의 표준이다. ISDN '통화'는 목적지까지의 동기식 점대점(point-to-point) 연결을 생성한다. 보통 ISDN은 여러 개의 다른 채널로 불리되는 고속 연결 상에서 전달된다. 채널에는 서로 다른 두 형식이 있는데 'B 채널'은 실제로 사용자의 자료를 전달하며 'D 채널'은 ISDN의 연결을 만들거나 다른 기능들을 위해 교환하는 제어 정보들을 보내기 위해 사용된다. 한 예로 호주에서는 ISDN이 2Mbps의 연결 상에서 사용되는데 이 연결은 30개의 64kbps B 채널과 하나의 64kbps D 채널로 나뉜다. 한 번에 몇 개의 채널이든 쓰일 수 있으며 어떤 조합으로도

쓰일 수 있다. 예를 들어 30개의 서로 다른 목적지에 각 64kbps로 30개의 다른 연결을 만들거나 15개의 목적지에 각 128kbps(연결 하나당 두 채널)로 15개의 연결을 만들 수도 혹은 적은 수의 연결을 만들고 나머지는 안쓰는 채로 둘 수도 있다. 채널은 나가는 연결이나 들어오는 연결 어느 쪽으로도 쓰일 수 있다. ISDN의 본 목적은 전화 회사들에게 사용자들이 특정한 설정의 변경 없이 집이나 사무실에서 전화(디지탈 전화)와 데이타 통신을 모두 사용할 수 있는 서비스를 제공하도록 하는 것이었다.

컴퓨터를 ISDN 서비스에 연결하는 방법은 몇 가지가 있다. 이중 하나는 'Terminal Adaptor'라 불리는 장치를 사용하는 것인데 이 장치는 여러분이 ISDN 서비스를 신청할 때 전화회사에서 설치해 주는 Network Terminating Unit에 끼워지며 많은 직렬 인터페이스를 가지고 있다. 이 인터페이스들 중 하나가 연결을 만들고 설정을 하는 명령을 내리기 위해 사용되며 나머지는 연결이 되었을 때 데이타 써킷을 쓸 네트웍 장치들에 실제로 연결된다. 리눅스는 별다른 수정 없이도 이런 설정에서 작동하며 여러분은 Terminal Adaptor의 포트를 다른 직렬 장치 다루듯 다룰 수 있다. 다른 방법은 커널이 ISDN을 지원하는 방법인데 ISDN 카드를 리눅스 머신에 설치할 수 있도록 설계되었으며 리눅스 소프트웨어가 프로토콜을 직접 다루어서 연결을 스스로 만들도록 되어있다.

Kernel Compile Options:

```
ISDN subsystem  --->
        <*> ISDN support
        [ ] Support synchronous PPP
        [ ] Support audio via ISDN
        < > ICN 2B and 4B support
        < > PCBIT-D support
        < > Teles/NICCY1016PC/Creatix support
```

Linux에서 구현된 ISDN은 많은 종류의 내장형 ISDN 카드들을 지원한다. 이들은 커널 설정 옵션에서 볼 수 있다.

- ICN 2B and 4B
- Octal PCBIT-D
- Teles ISDN-cards and compatibles

이 카드들중 일부는 작동하기 위해서 다른 소프트웨어를 받아야 할 필요가 있다. 이를 위해 별도의 다른 유틸리티가 존재한다.

Linux의 ISDN 지원의 설정에 대한 자세한 사항은 `/usr/src/linux/Documentation/isdn/` 디렉토리에서 볼 수 있으며 *isdn4linux*를 위한 FAQ를 `www.lrz-muenchen.de`에서 볼 수 있다.

PPP에 대한 주의점. PPP 프로토콜 관련 프로그램들은 동기식 직렬 연결과 비동기식 직렬 연결 모두에서 작동한다. 일반적으로 배포되는 Linux의 PPP 데몬인 '*pppd*'는 비동기 방식만 지원한다. ISDN 상에서 PPP를 사용하기 위해서는 특별히 수정된 버전이 필요하다. 이를 찾을 수 있는 곳에 대한 자세한 정보도 위에 언급한 문서에 나와있다.

7.2 Linux-2.0에서의 PLIP

PLIP 장치 이름은 '`plip0`'과 '`plip1`', '`plip2`' 등이다.

Kernel Compile Options:

```
Network device support  --->
    <*> PLIP (parallel port) support
```

plip (Parallel Line IP)는 직렬 포트 대신에 머신에 있는 병렬 프린터 포트를 쓴다는 점을 제외하곤 두 머신 간의 *점대점(point to point)* 네트웍 연결을 제공하기 위해 사용된다는 점에서 SLIP과 비슷하다(이 문서 뒷부분의 케이블 부분에서 케이블 그림이 제공된다). 병렬 포트에선 한 번에 하나 이상의 비트 전송이 가능하기 때문에 *plip* 인터페이스로 표준 직렬 장치보다 고속의 전송을 하는 것이 가능하다. 추가로 직렬 포트를 위해 비교적 비싼 16550AFN UART 포트를 사는 대신 간단한 병렬 포트인 프린터 포트를 사용할 수 있다. PLIP는 직렬 연결에 비해 CPU를 많이 사용하며 만약 값싼 이더넷 카드가 있다면 좋은 선택이 아니겠지만 달리 방법이 없을 경우 매우 잘 작동할 것이다. 연결이 잘 작동한다면 초당 20 킬로바이트의 전송률을 기대할 수 있다.

PLIP 장치 드라이버는 병렬 포트 하드웨어용 병렬 장치 드라이버와 충돌한다. 만약 두 드라이버를 모두 사용하고 싶다면 PLIP용 포트와 프린터 드라이버용 포트를 선택할 수 있도록 두 드라이버 모두 모듈로 컴파일해야 한다. 커널 모듈 설정에 대한 더 많은 정보를 위해선 "Module mini-HOWTO"를 참조하라.

일부 랩탑 컴퓨터들은 PLIP가 사용하는 특정 신호의 조합을 지원하지 않기 때문에 PLIP가 작동하지 않을 수 있다는 것을 명심하라. 그러나 프린터는 그런 조합을 쓰지 않는다.

리눅스의 *plip* 인터페이스는 *Crynwyr Packet Driver PLIP*와 호환이 되며 이른 여러분이 여러분의 리눅스 머신을 *plip*를 통해 tcp/ip 소프트웨어를 돌리고 있는 DOS 머신에 연결할 수 있다는 것을 의미한다.

커널 2.0.* 에선 plip 장치가 i/o port와 IRQ에 아래처럼 매핑된다.

```
device  i/o     IRQ
------  -----   ---
plip0   0x3bc   5
plip1   0x378   7
plip2   0x278   2
```

여러분의 병렬 장치가 위의 조합에 맞지 않는다면 *ifconfig* 명령의 'irq' 파라메타를 이용하여 포트의 IRQ를 바꿀 수 있다(ROM BIOS가 지원한다면 프린터 포트의 IRQ를 작동하도록 설정한다). 다른 방법으로 여러분이 모듈을 사용할 경우 *insmod* 명령에 "io="와 "irq=" 옵션을 사용할 수도 있다. 예를 들어

```
root# insmod plip.o io=0x288 irq=5
```

PLIP의 작동은 두 개의 제한시간에 의해 조절되는데 모두 대부분의 경우 기본값이 ok이다. 여러분이 매우 느린 컴퓨터를 가지고 있다면 이 값들을 증가시킬 필요가 있을 것이며 이럴 경우 증가시킬 타이머는 실제론 **상대** 컴퓨터에 있다. 커널을 다시 컴파일 하지 않고 이 타이머 설정을 바꿀 수 있도록 *plipconfig* 라는 프로그램이 존재한다. 이는 많은 리눅스 배포본에 포함되어 있다.

To configure a *plip* interface, you will need to invoke the following commands (or **add** them to your initialization scripts):

plip 인터페이스를 설정하기 위해선 아래 명령을 내린다(혹은 이 명령들을 초기화 스크립트에 추가할 수 있다).

```
root# /sbin/ifconfig plip1 localplip pointopoint remoteplip
root# /sbin/route add remoteplip plip1
```

이제 사용하는 포트는 I/O 주소가 0x378인 것이며 *localplip*와 *remoteplip*는 PLIP 케이블 상에서 사용될 IP 주소나 이름들이다. 나는 이 값을 /etc/hosts에 넣어 두었다.

```
# plip entries
192.168.3.1   localplip
192.168.3.2   remoteplip
```

pointtopoint 파라메타는 SLIP에서와 같은 의미를 가지는데 연결의 다른 끝에 있는 머신의 주소를 지정한다.

In almost all respects you can treat a *plip* interface as though it were a *SLIP* interface, except that neither *dip* nor *slattach* need be, nor can be, used.

*dip*와 *slattach* 모두 쓰이지 않을 경우를 제외하곤 대부분의 경우 여러분은 *plip* 인터페이스를 *SLIP* 인터페이스처럼 다룰 수 있다.

PLIP에 대한 더 많은 정보는 "PILP mini-HOWTO"에서 얻을 수 있다.

7.3 Linux-2.2에서의 PLIP

커널 2.1 버전의 개발동안 병렬 포트에 대한 지원이 더욱 좋게 변경되었다.

Kernel Compile Options:

```
        General setup  --->
            [*] Parallel port support
        Network device support  --->
            <*> PLIP (parallel port) support
```

PLIP의 새 코드는 이전의 것처럼 작동한다 (앞의 부분에서처럼 *ifconfig*와 *route* 명령을 똑같이 사용한다. 그러나 향상된 병렬 포트 지원 때문에 장치 초기화는 약간 다르다).

"첫번째" PLIP 장치는 항상 "plip0"라 불리며 이더넷 장치에서처럼 이 첫번째 장치는 시스템에 의해 잡히는 첫번째 장치이다. 실제로 사용되는 병렬 포트는 /proc/parpot에 나오는 것처럼 사용 가능한 포트들 중 하나이다. 예를 들어 한 개의 병렬 포트만 있다면 /proc/parport/0라는 단 한개의 디렉토리만 존재한 것이다.

커널이 포트의 IRQ를 찾지 못한다면 "insmod plip"는 실패한다. 이런 경우 /proc/parport/0/irq에 알맞은 값을 직접 쓰고 *insmod*를 다시 실행한다.

병렬 포트 관리에 대한 완전한 정보는 Documentation/parport.txt에 있으며 이는 커널 소스의 일부이다.

7.4 PPP

PPP 장치 이름은 'ppp0', 'ppp1' 등이다. 이 장치들은 첫 장치를 '0' 으로 시작해서 순서대로 번호가 부여된다.

Kernel Compile Options:

```
        Networking options  --->
            <*> PPP (point-to-point) support
```

PPP 설정은 PPP-HOWTO에서 자세히 다루어진다.

7.4.1 *pppd*를 이용해서 네트워에 영구적인 연결을 유지하기

여러분이 네트웍에 반 영구적인 연결을 할 수 있을 만큼 운이 좋고 연결이 끊길 경우 자동으로 PPP 연결을 다시 만들도록 하고 싶다면 여기에 간단한 트릭이 있다.

아래 명령을 수행해서 PPP가 root 유저에 의해 시작될 수 있도록 설정한다.

```
    # pppd
```

/etc/ppp/options 파일 안에 '-detach' 옵션을 설정하는 것을 주의한다. 그리고 아래 문장을 /etc/inittab 파일의 *getty* 정의 아래에 삽입한다.

```
pd:23:respawn:/usr/sbin/pppd
```

이는 *init* 프로그램으로 하여금 *pppd*를 실행 후 감시하면서 죽었을 경우 자동으로 재실행 하도록 해준다.

7.5 SLIP client

SLIP 장치는 'sl0', 'sl1' 등으로 이름이 붙여지며 설정되는 첫 장치가 '0'을 부여받고 나머지가 설정되는 순서대로 일련의 번호를 부여받는다.

Kernel Compile Options:

```
Network device support  --->
    [*] Network device support
    <*> SLIP (serial line) support
    [ ]  CSLIP compressed headers
    [ ]  Keepalive and linefill
    [ ]  Six bit SLIP encapsulation
```

SLIP(Serial Line Internet Protocol)은 직렬 연결 상에서 tcp/ip를 사용하도록 해 주며 이 연결은 모뎀을 이용한 전화선 혹은 비슷한 류의 다른 임대선일 수 있다. 물론 SLIP을 쓰기 위해선 근처의 *SLIP-server*에 접근할 수 있어야 한다. 전 세계의 수많은 대학과 기업들이 SLIP 연결을 지원한다.

Slip은 IP 데이타그램의 전송을 위해 직렬 포트를 사용한다. 이를 위해 직렬 장치에 대한 제어권을 획득해야 한다. Slip 장치는 *sl0*, *sl1* 등으로 이름이 붙여진다. 이 이름들이 어떻게 직렬 장치로 할당될까? 네트워킹 코드는 직렬 장치를 SLIP 장치로 바꾸기 위해 *ioctl* (i/o control) 이라 불리는 것을 사용한다. 이를 할 수 있는 두 개의 프로그램이 제공되는데 *dip*과 *slattach*이다.

7.5.1 dip

dip (Dialup IP)는 직렬 장치의 속도를 조절하고 연결의 다른 쪽으로 전화를 걸도록 모뎀에게 명령을 내리고 원격 서버에 자동으로 로긴하며 서버에서 보내진 메시지를 자동으로 찾아서 그 안에서 현재 장치의 IP 주소 같은 정보를 뽑아내고 직렬 포트를 SLIP 모드로 바꾸기 위해 필요한 *ioctl* 명령을 실행하는 등의 일을 하는 매우 유용한 프로그램이다. *dip*은 강력한 스크립팅 기능을 가지고 있으며 이를 이용해서 로긴 과정을 자동화할 수 있다.

`metalab.unc.edu`에서 이 프로그램을 구할 수 있다.

설치하기 위해선 아래와 같이 한다.

```
user% tar xvzf dip337o-uri.tgz
user% cd dip-3.3.7o
user% vi Makefile
root# make install
```

`Makefile`은 *uucp*라는 그룹이 존재한다고 가정하나 여러분의 설정에 따라 *dip*이나 *SLIP*으로 이를 변경할 수 있다.

7.5.2 slattach

*dip*과는 반대로 *slattach*은 매우 단순한 프로그램으로 사용하기엔 매우 쉽지만 *dip*과 같은 복잡성은 없다. 스크립팅 기능도 없으며 이를 이용해선 직렬 장치를 SLIP 장치로 설정하는 것만 할 수 있다. *slattach*은 여러분이 필요한 모든 정보를 알고 있고 이 프로그램 실행 전에 직렬 연결이 성립되어 있다고 가정한다. *slattach*은 물리적인 케이블이나 임대 선 같은 서버로의 영구적 연결을 사용하는 경우에 쓰기에 좋다.

7.5.3 어느 경우에 어떤 것을 사용하나 ?

여러분은 SLIP 서버로 머신을 연결하는 데에 다이얼업 모뎀이나 임시 선을 사용하는 경우엔 *dip*을 쓰는 것이 좋다. 머신과 서버간에 지속적인 연결이 가능하며 연결을 만들 별다른 일이 필요하지 않을 경우엔 *slattach*을 쓰는 것이 좋다. 자세한 사항은 'Permanent Slip connection' 부분을 참고하라.

SLIP 설정은 이더넷 인터페이스 설정과 매우 비슷하다 (위의 '이더넷 장치 설정' 부분을 읽어라). 그러나 중요한 몇 가지 차이점이 있다.

무엇보다도 연결의 양 끝에 한 개씩, 네트웍 상에 총 두 개의 호스트만이 있다는 점에서 SLIP 연결은 이더넷 네트웍과 다르다. 케이블 연결만으로도 금방 사용이 가능한 이더넷과는 달리 SLIP은 연결의 종류에 따라 특정한 방법으로 네트웍 연결을 초기화 해줘야 한다.

*dip*을 사용하는 경우 초기화는 일반적으로 부팅시가 아닌 연결을 사용할 수 있게 된 후의 어느 시점에 행해진다. 이 과정을 자동화 하는 것도 가능하다. *slattach*을 사용한다면 *rc.inet1* 안에 초기화 부분을 추가할 수 있으며 이에 대해선 곧 설명할 것이다.

SLIP 서버에는 크게 두 종류가 있는데 동적 IP 주소 서버와 정적 IP 주소 서버이다. 거의 모든 SLIP 서버는 여러분에게 연결 시에 사용자 이름과 암호를 써서 로긴하도록 할 것이다. *dip*은 이런 로긴 과정을 자동으로 다룰 수 있다.

7.5.4 전화선과 DIP을 이용한 정적 SLIP 서버로의 연결.

정적 SLIP 서버는 여러분에게 고정되며 다른 사람과 겹치지 않는 하나의 IP 주소를 제공한다. 서버로 연결될 때마다 여러분의 SLIP 포트는 이 주소로 설정된다. 정적 SLIP 서버는 모뎀 연결에 응답하면서 사용자 이름과 암호를 물어볼 것이고 곧 여러분의 IP 주소로 가는 데이타그램들을 그 연결을 통해 보내줄 것이다. 정적 서버를 사용한다면 여러분의 호스트명과 IP 주소에 대한 엔트리를 `/etc/hosts` 안에 넣어 둘 수 있다. 또한 `rc.inet2`, `host.conf`, `resolv.conf`, `/etc/HOSTNAME`, `rc.local` 같은 파일들도 설정할 수 있다. SLIP 연결시에 인터페이스의 설정에 대한 모든 일은 *dip*이 해주므로 `rc.inet1` 을 설정할 때 SLIP 연결에 대한 별다른 사항을 넣을 필요가 없다는 것을 명심하라. 여러분은 *dip*에 알맞은 값들을 넘겨줘야 하며 이 프로그램은 모뎀이 연결을 만들도록 하고 SLIP 서버에 로긴한 후에 인터페이스를 자동으로 설정할 것이다.

여러분이 사용하는 서버가 위와 같이 동작한다면 *dip*을 알맞게 설정하기 위해 'Dip 사용하기' 부분으로 넘어가도 된다.

7.5.5 전화선과 DIP을 이용한 동적 SLIP 서버로의 연결.

동적 SLIP 서버는 여러분이 로긴할 때마다 수많은 주소들 중에 임의의 IP 주소를 여러분에게 할당해 준다. 따라서 여러분이 매번 특정한 주소를 가진 다는 것을 보장할 수 없으며 현재 받은 주소도 여러분이 연결을 끊은 후에는 다른 누군가가 사용할 수도 있다. 이런 SLIP 서버를 설정하는 네트웍 관리자는 SLIP 서버가 사용할 주소 풀(pool)을 가지고 있으며 새로운 연결이 들어올 때마다 사용되지 않는 주소중 하나를 고르고 접속자를 로긴 과정으로 이끈 후에 할당된 IP 주소를 포함하는 환영 메시지를 보내주고 그 IP 주소를 연결이 되어 있는 동안 사용하도록 한다.

이런 종류의 서버에 대한 설정도 정적 서버의 경우와 비슷하나 서버가 할당해 준 IP 주소를 받을 경우 그 값으로 SLIP 장치를 설정해 주는 과정이 추가되어야 한다.

역시 *dip*이 필요한 힘든 일들을 모두 해주며 새 버전은 로긴 과정뿐만 아니라 자동으로 환영 메시지 안의 IP 주소를 읽어서 SLIP 장치를 이 값으로 설정해 주는 것을 자동으로 해줄 만큼 쓸만한다.

여러분의 SLIP 서버가 이와 같이 동작한다면 *dip*을 알맞게 설정하는 법을 보기 위해 'Dip 사용하기' 부분으로 이동해도 된다.

7.5.6 DIP 사용하기.

앞에서 설명한 것처럼 *dip*은 SLIP 서버로 연결 요청과 로긴, 연결 수립, 알맞은 *ifconfig*과 *route* 명령으로의 SLIP 장치 설정을 자동화 하고 둔산화 해주는 강력한 프로그램이다.

근본적으로 *dip*을 사용하기 위해선느 'dip script'를 만들어야 하는데 이 스크립트는 여러분이 *dip*이 하길 원하는 작업들을 수행하는 방법을 *dip*에게 알려주는 명령들의 단순한 목록이다. 작동하는 방식을 보기 위해선 *dip*과 같이 제공되는 `sample.dip`을 보면 된다. *dip* 은 많은 옵션을 가진 매우 강력한 프로그램이다. 여기에서 모든 것을 다루는 대신에 여러분은 *man* 페이지나 *dip*과 같이 제공되는 샘플 파일들과 README를 볼 것은 권한다.

`sample.dip` 스크립트는 여러분이 정적 SLIP 서버를 사용한다고 가정하고 있다. 따라서 여러분의 IP 주소가 무엇인지 먼저 알고 있어야 한다. 동적 SLIP 서버인 경우를 위해 *dip*의 새 버젼들은 동적 서버가 할당해 준 IP 주소를 자동으로 읽어들여 SLIP 장치를 설정해 주는 데에 사용할 수 있는 명령을 제공한다. 아래의 예제는 *dip337j-uri.tgz*에 들어있는 `sample.dip`을 약간 수정한 것이며 여러분에게 좋은 시작점이 될 것이다. 이 예제를 `/etc/dipscript`로 저장한 후에 여러분의 설정에 맞도록 수정할 수도 있다.

```
#
# sample.dip    Dialup IP connection support program.
#
#               This file (should show) shows how to use the DIP
#       This file should work for Annex type dynamic servers, if you
#       use a static address server then use the sample.dip file that
#       comes as part of the dip337-uri.tgz package.
#
#
# Version:      @(#)sample.dip  1.40    07/20/93
#
# Author:       Fred N. van Kempen, <waltje@uWalt.NL.Mugnet.ORG>
#

main:
# Next, set up the other side's name and address.
# My dialin machine is called 'xs4all.hacktic.nl' (== 193.78.33.42)
get $remote xs4all.hacktic.nl
# Set netmask on sl0 to 255.255.255.0
netmask 255.255.255.0
# Set the desired serial port and speed.
port cua02
speed 38400

# Reset the modem and terminal line.
# This seems to cause trouble for some people!
reset

# Note! "Standard" pre-defined "errlevel" values:
#  0 - OK
#  1 - CONNECT
#  2 - ERROR
#
# You can change those grep'ping for "addchat()" in *.c...

# Prepare for dialing.
send ATQ0V1E1X4\r
wait OK 2
if $errlvl != 0 goto modem_trouble
```

```
dial 555-1234567
if $errlvl != 1 goto modem_trouble

# We are connected.  Login to the system.
login:
sleep 2
wait ogin: 20
if $errlvl != 0 goto login_trouble
send MYLOGIN\n
wait ord: 20
if $errlvl != 0 goto password_error
send MYPASSWD\n
loggedin:

# We are now logged in.
wait SOMEPROMPT 30
if $errlvl != 0 goto prompt_error

# Command the server into SLIP mode
send SLIP\n
wait SLIP 30
if $errlvl != 0 goto prompt_error

# Get and Set your IP address from the server.
#   Here we assume that after commanding the SLIP server into SLIP
#   mode that it prints your IP address
get $locip remote 30
if $errlvl != 0 goto prompt_error

# Set up the SLIP operating parameters.
get $mtu 296
# Ensure "route add -net default xs4all.hacktic.nl" will be done
default

# Say hello and fire up!
done:
print CONNECTED $locip ---> $rmtip
mode CSLIP
goto exit

prompt_error:
print TIME-OUT waiting for sliplogin to fire up...
goto error

login_trouble:
print Trouble waiting for the Login: prompt...
goto error

password:error:
print Trouble waiting for the Password: prompt...
goto error

modem_trouble:
print Trouble occurred with the modem...
error:
```

```
print CONNECT FAILED to $remote
quit

exit:
exit
```

위의 예제는 여러분이 *동적* SLIP 서버에 연결한다고 가정하고 있으며 만약 *정적* SLIP 서버를 사용한다면 *dip337j-uri.tgz*에 들어있는 sample.dip가 잘 작동할 것이다.

*dip*이 *get $local* 명령을 만나면 연결에서 들어오는 텍스트 안에서 IP 주소처럼 보이는, 다시 말해 '.' 문자로 분리된 숫자열들을 찾는다. 이 수정사항은 *동적* SLIP 서버와 동작하기 위해서 들어가며 이로 인해 서버에 의해 할당된 IP 주소를 읽어들이는 과정이 자동화될 수 있다.

위의 예제는 SLIP 연결을 통하는 기본 라우트를 자동으로 만들 것이며 이를 원치 않고 기본 라우터로 사용할 이더넷 연결이 있다면 스크립트에서 *default* 명령을 삭제해야 한다. 스크립트의 실행이 끝난 후 *ifconfig* 명령을 수행시켜 보면 *sl0*이라는 장치를 볼 수 있을 것이다. 이것이 SLIP 장치이다. *dip* 명령이 수행된 후라도 *ifconfig*와 *route* 명령을 사용해서 설정을 수동으로 바꿀 수 있다.

*dip*은 mode 명령을 통해서 여러 개의 프로토콜 중에서 사용할 것을 선택할 수 있도록 해주며 가장 많이 쓰이는 것은 압축 기능이 있는 SLIP에 쓰이는 *cSLIP*이다. 중요한 것은 연결의 양 쪽이 모두 동일한 것을 사용해야 하며 따라서 여러분이 사용하는 서버와 같은 것을 사용해야 한다.

위의 예제는 매우 안정적이며 대부분의 오류들에 대처할 수 있다. 추가 정보를 위해선 *dip*의 man 페이지를 참고하라. 예를 들어 여러분은 미리 정해진 일정 시간동안 연결 수립이 안되면 서버로 다시 전화를 걸도록 하거나 하나 이상의 서버를 차례대로 시도 하도록 하는 등의 일을 스크립트로 만들 수 있다.

7.5.7 임대선과 slattach를 이용한 영구적인 SLIP 연결.

여러분이 두 머신간에 케이블 연결을 사용하거나 임대선을 쓰는 등의 영구적인 직렬 연결을 사용할 수 있다면 직렬 연결을 만들기 위해서 *dip*을 사용하며 생기는 모든 어려움들을 감수할 필요는 없다. *slattach*은 단지 연결을 설정하는 기능만을 갖춘 매우 간단한 프로그램이다.

연결이 영구적일 것이므로 rc.inet1 파일 안에 명령들을 추가할 수도 있다. 영구적인 연결에 있어 근본적으로 필요한 것은 직렬 장치를 알맞은 속도로 설정하고 직렬 장치를 SLIP 모드로 바꾸는 것 뿐이다. *slattach*은 이러한 것을 하나의 명령으로 가능하도록 해준다. 아래의 코드를 rc.inet1 파일에 **추가하라**.

```
#
# Attach a leased line static SLIP connection
#
#  configure /dev/cua0 for 19.2kbps and cslip
/sbin/slattach -p cslip -s 19200 /dev/cua0 &
/sbin/ifconfig sl0 IPA.IPA.IPA.IPA pointopoint IPR.IPR.IPR.IPR up
#
# End static SLIP.
```

설명 :

IPA.IPA.IPA.IPA

여러분의 IP 주소를 나타낸다.

IPR.IPR.IPR.IPR

반대편 끝의 IP 주소를 나타낸다.

*slattach*은 할당되지 않은 첫번째 SLIP 장치를 지정된 직렬 장치에 할당한다. *slattach*은 *sl0*에서부터 시작한다. 따라서 맨 처음 *slattach*은 SLIP 장치 *sl0*을 지정된 직렬 장치에 할당하고 다음엔 *sl1*을 할당한다.

*slattach*은 -p 인자를 가지고 여러 가지의 프로토콜을 설정할 수 있도록 해준다. 압축을 사용할 것인지 아닌지에 따라 *SLIP*이나 *cSLIP* 중 하나를 사용할 것이며 연결의 양 끝이 모두 같은 것을 사용해야 한다는 것을 명심한다.

7.6 SLIP 서버.

여러분이 네트웍에 연결되어 있고 다른 사람이 전화를 걸고 들어와서 네트웍 서비스를 사용 가능하도록 하고 싶은 머신이 있다면 그 머신을 서버로 설정할 필요가 있다. 직렬 연결 프로토콜로 SLIP을 쓰길 원한다면 여러분의 리눅스 머신을 SLIP 서버로 설정하는 데에는 크게 세 가지 옵션이 있다. 내가 좋아하는 것은 처음 설명되는 *sliplogin*을 사용하는 것인데 이는 이 방법이 이해하고 설정하기 가장 쉽기 때문이다. 그러나 여기에선 모든 방법을 간단히 쉶여할 것이며 여러분이 알맞은 것을 선택할 수 있다. –>

7.6.1 *sliplogin*을 사용하는 SLIP 서버.

*sliplogin*은 터미날 연결을 SLIP 연결로 바꿔 주어 SLIP 사용자 용으로 일반 로긴 쉘 대신에 사용할 수 있는 프로그램이다. 이 프로그램은 여러분의 리눅스 머신을 사용자가 접속할 때마다 같은 주소를 받게 하는 *정적 주소 서버*나 접속할 때마다 가능한 주소를 할당해서 쓰게 하는 *동적 주소 서버*로 설정할 수 있다.

접속자는 표준 로긴 과정처럼 사용자 이름과 암호를 입력할 것이나 로긴 이후에 쉘을 받는 대신 *sliplogin*이 실행되어서 접속한 사용자의 로긴 이름에 해당하는 엔트리를 설정 파일(`/etc/slip.hosts`)에서 검색한다. 엔트리가 찾아지면 연결을 8비트 클린 연결로 설정한 후 SLIP 연결로 변환하기 위해 *ioctl*을 사용한다. 이 과정이 끝나면 설정의 마지막 단계가 진행되는데 이 단계에서 *sliplogin*은 쉘 스크립트를 실행하여 SLIP 인터페이스를 알맞은 IP 주소와 netmask로 설정하고 적당한 라우팅 세팅을 한다. 이 스크립트는 일반적으로 `/etc/slip.login`이나 특정 접속자에게 특정한 초기화가 필요하다면 *getty*에서처럼 `/etc/slip.login.loginname`의 설정 스크립트를 만들어서 기본 스크립트 대신 실행되도록 할 수도 있다>

*sliplogin*이 여러분에게 알맞게 작동하도록 하기 위해 설정해야 할 파일은 3 4개가 있다. 여기에서 소프트웨어를 구하는 방법과 장소, 그리고 각각을 설정하는 방법을 자세히 설명할 것이다. 필요한 파일들은 아래와 같다.

- `/etc/passwd`, 접속 사용자 계정을 위해.
- `/etc/slip.hosts`, 각 접속 사용자에게 특화된 정보를 담고 있다.
- `/etc/slip.login`, 사용자를 위해 필요한 라우팅 설정을 관리한다.
- `/etc/slip.tty`, 서버를 *동적 주소 할당* 방식으로 설정하는 경우에만 필요하며 할당할 주소들의 테이블을 담고 있다.
- `/etc/slip.logout`, 사용자가 접속을 끊고 나간 후의 클린-업을 위한 명령을 담고 있다.

***sliplogin*을 얻는 곳** 여러분은 이미 배포본의 일부로 *sliplogin* 패키지를 가지고 있을 수도 있으나 그렇지 않다면 `metalab.unc.edu` 에서 *sliplogin*을 구할 수 있다. 이 tar 파일은 소스와 이미 컴파일 된 바이너리, *man* 페이지를 모두 포함하고 있다.

허가된 사용자에게만 *sliplogin*을 실행할 수 있도록 하기 위해선 `/etc/group` 파일에 아래와 같은 엔트리를 추가해야 한다.

```
 ..
slip::13:radio,fred
 ..
```

sliplogin 패키지를 인스톨하면 Makefile이 *sliplogin* 프로그램의 소유자 그룹을 slip으로 변경하여 이 그룹에 속한 사용자만이 이 프로그램을 쓸 수 있도록 한다. 위의 예제는 radio와 fred라는 사용자에게만 *sliplogin*을 실행할 수 있도록 하는 것이다.

바이너리 파일을 /sbin에, *man* 페이지를 8번 섹션에 설치하려면 아래와 같이 한다.

```
# cd /usr/src
# gzip -dc .../sliplogin-2.1.1.tar.gz | tar xvf -
# cd sliplogin-2.1.1
# <..edit the Makefile if you don't use shadow passwords..>
# make install
```

설치 전에 바이너리 파일을 다시 컴파일하길 원하면 make install 이전에 make clean을 추가한다. 바이너리 파일들을 다른 곳에 설치하고 싶다면 Makefile 의 *install* 규칙을 수정하면 된다.

추가 정보를 위해선 패키기에 함께 들어 있는 README 파일들을 참고하라.

SLIP 호스트에서의 /etc/passwd 설정. 보통 /etc/passwd 파일 안에 SLIP 접속자에 대한 특별한 로긴 엔트리를 넣어야 한다. 흔히 쓰이는 방법은 접속지 호스트의 호스트명 앞에 대문자 'S'를 붙이는 것이다. 예를 들어 radio라는 호스트에서 접속한다면 /etc/passwd에 아래와 같은 엔트리를 추가한다.

```
Sradio:FvKurok73:1427:1:radio SLIP login:/tmp:/sbin/sliplogin
```

잘 동작하기만 한다면 이 계정의 이름 자체는 중요한 것이 아니다.

주의할 것은 접속자가 이 머신으로부터 쉘을 받을 것이 아니므로 특정한 홈 디렉토리를 가질 필요가 없다는 것이다. 따라서 /tmp를 쓰는 것은 좋은 선택이다. 또한 일반 로긴 쉘 대신 *sliplogin*이 사용되었다는 것을 주의한다.

/etc/slip.hosts의 설정 *sliplogin*은 접속자를 위한 자세한 설정 사항을 얻기 위해 로긴 이름에 해당하는 엔트리를 /etc/slip.hosts 파일에서 찾는다. 이 파일은 접속자에게 부여되어 알맞게 설정되는 IP 주소와 netmask를 지정하는 곳이다. radio라는 접속자 호스트에 대한 정적인 설정과 albert라는 접속자 호스트에 대한 동적 설정에 대한 예제 엔트리가 아래에 있다.

```
#
Sradio   44.136.8.99   44.136.8.100  255.255.255.0  normal      -1
Salbert  44.136.8.99   DYNAMIC       255.255.255.0  compressed  60
#
```

/etc/slip.hosts 파일의 엔트리들:

1. 접속자의 로긴 명.
2. 서버 머신, 다시 말해 이 머신의 IP 주소.
3. 접속자가 할당받을 IP 주소. 이 필드가 DYNAMIC이라면 IP 주소는 다음에 설명될 /etc/slip.tty 파일에 들어있는 정보에 근거해서 할당된다. **주의사항:** 이것이 제대로 작동하려면 적어도 1.3 버전 이상의 것을 사용해야 한다.
4. 접속자 머신에 부여될 netmask. dotted decimail notation으로 표기되며 예를 들어 255.255.255.0은 C 클래스 네트웍의 mask다.
5. 압축이나 다른 slip의 기능들을 키고 끄도록 하는 slip 모드 설정. 가능한 값은 "normal"이나 "compressed"이다.

6. 연결이 끊기지 않고 아이들(데이타그램 이동이 없는) 상태로 있을 수 있는 시간을 지정하는 타임아웃 값. 음수인 경운 이 기능을 끈다.

7. 추가적인 인자들.

주의사항: 두 번째와 세 번째 필드엔 점으로 분리되는 십진수로 표기되는 IP 주소나 호스트명 어느 것이나 쓸 수 있다. 호스트명을 쓰는 경우엔 그 호스트명하는 IP 주소를 찾을 수 있어야 한다. 그렇지 않다면 스크립트 실행이 실패할 것이다. 이는 그 호스트로 텔넷을 해 봄으로써 시험해 볼 수 있으며 '*Trying nnn.nnn.nnn...*'과 같은 메시지가 나오면 여러분의 머신이 그 호스트명에 해당하는 IP 주소를 찾을 수 있는 것이다. '*Unknown host*' 메시지가 나오면 찾을 수 없는 것이다. IP 주소를 찾을 수 없을 경우 IP 주소를 사용하거나 name resolver의 설정을 수정해야 한다(`Name Resolution` 부분을 참고하라).

가장 일반적인 slip 모드는 다음과 같다.

normal

to enable normal uncompressed SLIP.

compressed

to enable van Jacobsen header compression (cSLIP)

보통 이 두 모드는 상호 배타적이며 어느 쪽이든 사용할 수 있다. 다른 옵션들에 대한 자세한 정보는 *man* 페이지를 참고하라.

`/etc/slip.login` 파일의 설정. *sliplogin*은 `/etc/slip.hosts`를 검색하여 해당 엔트리를 찾은 후에 실제로 SLIP 인터페이스를 IP 주소와 netmask로 설정하기 위해 `/etc/slip.login` 파일을 실행한다. *sliplogin*페키지와 함께 제공되는 `/etc/slip.login` 파일 예제는 아래와 같다.

```
#!/bin/sh -
#
#       @(#)slip.login  5.1 (Berkeley) 7/1/90
#
# generic login file for a SLIP line.  sliplogin invokes this with
# the parameters:
#     $1       $2       $3    $4, $5, $6 ...
#   SLIPunit ttyspeed   pid   the arguments from the slip.host entry
#
/sbin/ifconfig $1 $5 pointopoint $6 mtu 1500 -trailers up
/sbin/route add $6
arp -s $6 <hw_addr> pub
exit 0
#
```

여러분은 이 스크립트가 단순히 *ifconfig*와 *route* 명령만을 사용하여 SLIP 장치를 해당 IP 주소와 원격지 IP 주소, netmask를 가지고 설정하고 SLIP 장치를 통한 라우팅을 만든 다는 것을 알 수 있다. *slattach* 명령을 사용하는 경우와 똑같다.

서버 머신과 같은 이더넷 상의 다른 호스트들이 접속해 온 호스트에 닿을 수 있도록 하기 위해선 *Proxy ARP*을 써야 한다는 것을 주의해야 한다.

`/etc/slip.logout` 파일의 설정. 접속이 끊겼을 경우 여러분은 직렬 장치가 보통 상태로 되돌아 가서 이후의 접속자들이 정확히 로긴할 수 있기를 원할 것이다. 이는 `/etc/slip.logout` 파일을 이용하면 가능하다. 이 파일은 형식이 매우 간단하며 `/etc/slip.login` 파일과 같은 인자들을 가지고 호출된다.

```
#!/bin/sh -
#
#               slip.logout
#
/sbin/ifconfig $1 down
arp -d $6
exit 0
#
```

이 파일이 하는 일은 앞에서 만들어진 라우팅 정보를 지우기 위해 인터페이스의 작동을 '정지시키는' 일이 전부이다. 또한 proxy arp들을 지우기 위해 *arp* 명령도 사용하는데 만약 여러분의 서버 머신이 이더넷 포트를 가지고 있지 않다면 이 스크립트엔 *arp* 명령은 필요 없다.

`/etc/slip.tty` **파일의 설정.** 동적 IP 주소 할당을 사용한다면 (`/etc/slip.hosts` 파일 안에 어떤 호스트라도 `DYNAMIC` 으로 설정했다면) 어떤 포트에 어떤 주소가 할당될 것인지에 대한 목록을 갖도록 `/etc/slip.tty` 파일을 설정해야 한다. 이 파일은 사용자에게 주소를 동적으로 할당할 경우에만 필요하다.

이 파일은 SLIP 연결을 지원하는 *tty* 장치들과 그 포트로 접속해 들어온 사용자에게 배정될 IP 주소들을 열거해 놓은 테이블이다.

형식은 다음과 같다.

```
# slip.tty    tty -> IP address mappings for dynamic SLIP
# format: /dev/tty?? xxx.xxx.xxx.xxx
#
/dev/ttyS0      192.168.0.100
/dev/ttyS1      192.168.0.101
#
```

이 테이블은 `/dev/ttyS0` 포트로 접속해 들어왔으며 `/etc/slip.hosts` 파일의 원격지 주소 필드가 `DYNAMIC`로 설정된 사용자에게 `192.168.0.100` 주소를 할당한다는 것을 의미한다.

이런 방식으로 특정 주소가 지정되어 있지 않은 모든 사용자에게 포트 하나당 한 주소만 할당해야 한다. 이는 낭비를 줄이기 위해 필요한 주소의 수를 최소로 할 수 있도록 해준다.

7.6.2 *dip*을 쓰는 SLIP 서버

아래의 정보들은 *dip*의 man 페이지에서 구한 것이라는 것을 먼저 말하고 싶다. man 페이지에는 리눅스를 SLIP 서버로 사용하는 방법이 간단히 설명되어 있다. 아래 내용은 *dip337o-uri.tgz* 페키지에 대한 것이며 다른 버전에선 제대로 작동하지 않을 수도 있다는 것도 주의해야 한다.

*dip*은 들어오는 연결에 대한 작동 모드(input mode)를 가지고 있다. 이 모드에선 *dip*을 실행한 접속자에 대항하는 엔트리를 찾아서 직렬 연결을 `/etc/diphosts` 파일 안에 있는 해당 정보에 따라 SLIP 연결로 설정한다. 이 모드는 *dip*을 *diplogin*으로 실행시켜서 사용할 수 있다. 따라서 이런 방법을 쓰면 로긴 쉘로 *diplogin*을 사용하는 특별한 계정을 만드는 것만으로 *dip*을 SLIP 서버로 사용할 수 있다.

가장 먼저 해야 할 일은 아래처럼 심볼릭 링크를 만드는 것이다.

```
# ln -sf /usr/sbin/dip /usr/sbin/diplogin
```

그리곤 `/etc/passwd` 파일과 `/etc/diphosts` 파일 모두에 엔트리들을 추가해야 하는데 이 엔트리들의 형식은 아래와 같다.

*dip*을 가지고 리눅스를 SLIP 서버로 설정하기 위해서 접속자들을 위해서 (input mode로 작동하는)*dip*을 로긴 쉘로 사용하는 특별한 SLIP 계정을 만들어야 한다. 흔히 사용되는 방법은 모든 SLIP 계정을 대문자 'S'로 시작하도록 하는 것이다. 예를 들면 'Sfredm' 처럼.

SLIP 사용자에 대한 `/etc/passwd` 엔트리의 예제가 아래에 있다.

```
Sfredm:ij/SMxiTlGVCo:1004:10:Fred:/tmp:/usr/sbin/diplogin
^^         ^^         ^^  ^^   ^^   ^^   ^^
|          |          |   |    |    |    \__ 로긴 쉘로 쓰이는 diplogin
|          |          |   |    |    \_______ 홈 디렉토리
|          |          |   |    \____________ 사용자 이름
|          |          |   \_________________ 사용자 그룹 ID
|          |          \_____________________ 사용자 ID
|          \________________________________ Encrypted User Password
\___________________________________________ Slip 사용자 로긴 명
```

사용자가 접속한 후 사용자 확인이 끝나면 *login* 프로그램은 *diplogin*을 실행시킨다. *diplogin*이라는 이름으로 수행되면 *dip*은 자신이 로긴 쉘로 사용되고 있다는 것을 자동으로 알게 된다. *diplogin*으로 실행될 때 가장 먼저 하는 것은 자신을 실행시킨 사용자의 userid를 얻기 위해 *getuid()* 펑션 콜을 호출하는 것이다. 그리곤 userid나 접속이 들어온 *tty* 이름 중 어느 하나에 해당되는 엔트리를 `/etc/diphosts` 파일에서 찾아 가장 먼저 발견되는 것을 가지고 연결을 설정한다. `diphosts` 파일 내에 사용자에 대한 엔트리를 추가할지 혹은 기본 설정을 쓰도록 할 지에 대한 판단에 따라 여러분은 정적 주소를 받는 사용자와 동적으로 주소를 받는 사용자를 모두 수용하는 서버를 만드는 것처럼 만들 수 있다.

*dip*은 접속을 받아들이는 모드에서 실행될 때 자동으로 'Proxy-ARP' 엔트리를 추가하므로 이를 수동으로 추가하는 것에 대해 신경쓸 필욘 없다.

`/etc/diphosts`의 설정. `/etc/diphosts`는 *dip*이 원격지 호스트에 대한 설정 사항을 찾아보기 위해 사용된다. 이 원격지 호스트들은 리눅스 머신으로 접속을 해 오는 사용자들일 수도 있고 여러분이 리눅스 머신으로 접속을 해 들어가는 머신일 수도 있다.

`/etc/diphosts`의 일반적인 형식은 아래와 같다:

```
 ..
Suwalt::145.71.34.1:145.71.34.2:255.255.255.0:SLIP uwalt:CSLIP,1006
ttyS1::145.71.34.3:145.71.34.2:255.255.255.0:Dynamic ttyS1:CSLIP,296
 ..
```

필드 설명:

1. `login name`: getpwuid(getuid())의 리턴 값이나 tty 명
2. `unused`: compat. with passwd
3. `Remote Address`: 접속해 오는 호스트의 IP 주소, 숫자나 이름
4. `Local Address`: I이 머신의 IP 주소, 역시 숫자나 이름
5. `Netmask`: dotted decimal notation 으로
6. `Comment field`: 원하는 사항을 여기에 삽입.
7. `protocol`: Slip, CSlip 등등
8. `MTU`: 십진수

원격 SLIP 사용자에 대한 /etc/net/diphosts의 한 예가 아래에 있다:

```
Sfredm::145.71.34.1:145.71.34.2:255.255.255.0:SLIP uwalt:SLIP,296
```

이는 원격지 주소는 145.71.34.1로, MTU는 296으로 하여 SLIP 연결을 설정한다.

혹은

```
Sfredm::145.71.34.1:145.71.34.2:255.255.255.0:SLIP uwalt:CSLIP,1006
```

원격지 주소는 145.71.34.1로 MTU는 1006으로 하여 cSLIP 가능한 연결을 설정한다.

따라서 정적으로 할당된 IP 주소를 받을 사용자들은 모두 /etc/diphosts 안에 엔트리를 가져야 한다. 특정 포트로 들어오는 사용자들의 자세한 설정이 동적으로 할당되도록 하려면 tty 장치에 대한 엔트리를 추가해야 하며 사용자에 대한 엔트리를 설정해선 안된다. 또한 명심해야 할 것은 사용자들이 어떤 모뎀으로 접속해 들어오든 알맞은 설정을 쓸 수 있도록 하기 위해서 접속에 사용되는 모든 tty 장치에 대해 적어도 하나의 엔트리를 만들어야 한다는 것이다.

접속해 들어올 때 사용자는 SLIP 로긴 id와 암호를 입력할 수 있는 일반적인 로긴 과 패스워드 프롬프트를 받게 된다. 사용자가 인증되면 특별한 메시지를 받지 않고 사용자 쪽은 SLIP 모드로 전환된다. 이제 사용자는 SLIP 연결이 가능하며 diphosts 파일로 부터 받아지는 해당 인자들로 설정이 된다.

7.6.3 *dSLIP* 패키지를 쓰는 SLIP 서버.

Matt Dillon <dillon@apollo.west.oic.com>은 다이알-인 뿐 아니라 다이알-아웃 SLIP도 지원하는 패키지를 만들었다. Matt의 패키지는 연결을 관리하는 작은 프로그램들과 스크립트들의 묶음이다. 스크립트들 중 적어도 하나가 *tcsh* 을 필요로 하므로 이를 설치해야 한다. 또한 한 스크립트에서 *expect*을 필요로 하기 때문에 Matt은 이 유틸리티의 실행 파일도 제공한다. 이 패키지를 여러분의 기호에 맞게 작동하도록 하려면 *expect*에 대한 약간의 경험이 필요할 것이나 없다고 해서 패키지를 사용치 못하는 것은 아니다.

Matt이 README 파일에 설치 순서에 대해 잘 설명해 놓았으므로 여기서 굳이 그 과정을 반복하진 않겠다.

아래의 홈 사이트에서 *dSLIP* 패키지를 구할 수 있다:

apollo.west.oic.com

/pub/linux/dillon_src/dSLIP203.tgz

혹은 아래에서도 구할 수 있다:

metalab.unc.edu

/pub/Linux/system/Network/serial/dSLIP203.tgz

make install을 하기 **전에** README 파일을 읽고 /etc/passwd와 /etc/group 엔트리들을 만들도록 한다.

8 다른 네트웍 기술들

아래 부분들은 특정 네트웍 기술에 관한 내용이다. 각 부분들에 담긴 내용들이 다른 네트웍 기술에 적용될 필요는 없다. 기술의 알파벳 순서대로 기술되어 있다.

8.1 ARCNet

ARCNet 장치 이름은 'arc0e', 'arc1e', 'arc2e' 등이거나 'arc0s', 'arc1s', 'arc2s' 등이다. 커널이 처음으로 발견하는 카드가 'arc0e'나 'arc0s'를 배정받고 나머지는 발견되는 순서대로 배정된다. 이름 끝의 문자는 이더넷 캡슐화 패킷 형식과 RFC1051 패킷 형식 중 어느 것을 선택했는 지를 나타낸다.

Kernel Compile Options:

```
Network device support  --->
    [*] Network device support
    <*> ARCnet support
    [ ]   Enable arc0e (ARCnet "Ether-Encap" packet format)
    [ ]   Enable arc0s (ARCnet RFC1051 packet format)
```

이더넷 카드를 지원하도록 커널을 알맞게 만들었다면 카드의 설정은 쉽다.

보통 해야할 일은 아래와 같다:

```
root# ifconfig arc0e 192.168.0.1 netmask 255.255.255.0 up
root# route add -net 192.168.0.0 netmask 255.255.255.0 arc0e
```

더 많은 정보를 원하면 /usr/src/linux/Documentation/networking/arcnet.txt와 /usr/src/linux/Documentation/networking/arcnet-hardware.txt를 참조하면 된다.

ARCNet 지원은 Avery Pennarun, apenwarr@foxnet.net에 의해 개발되었다.

8.2 Appletalk (AF_APPLETALK)

Appletalk 지원은 이미 있는 네트웍 장치를 사용하므로 특별한 장치 이름을 갖지 않는다.

Kernel Compile Options:

```
Networking options  --->
    <*> Appletalk DDP
```

Appletalk 지원은 리눅스 머신이 애플 네트웍과 통신할 수 있도록 해준다. 이것은 애플 컴퓨터와 리눅스 간에 프린터와 디스크 같은 자원을 공유하려할 때 매우 중요하다. *nettalk*라는 추가적인 소프트웨어도 필요하다. Wesley Craig netatalk@umich.edu는 미시간 대학에 'Research Systems Unix Group'이라는 팀을 만들었고 그 팀이 Appletalk 프로토콜 스택을 구현하는 소프트웨어와 몇 가지 유용한 유틸리티들을 제공하는 *nettalk* 패키지를 만들었다. *nettalk* 패키지는 여러분의 리눅스 배포본에 들어 있거나 홈 사이트인 UniversityofMichigan에서 ftp로 구할 수 있다.

이 패키지를 빌드해서 설치하는 과정은 아래와 같다:

```
user% tar xvfz .../netatalk-1.4b2.tar.Z
user% make
root# make install
```

소프트웨어를 실제로 컴파일 하려고 *make*를 부르기 전에 'Makefile'을 고칠 필요가 있을 수 있다. 특히 파일들이 설치되는 장소를 나타내는 DESTDIR 변수를 변경할 지도 모른다. 기본값인 /usr/local/atalk 는 꽤 좋은 위치이다.

8.2.1 Appletalk 소프트웨어의 설정.

Appletalk 소프트웨어가 잘 작동하도록 하기 위해 해야 할 첫 번째 일은 /etc/services 파일에 알맞은 엔트리가 들어가 있도록 하는 것이다. 필요한 엔트리는 아래와 같다:

```
rtmp  1/ddp   # Routing Table Maintenance Protocol
nbp   2/ddp   # Name Binding Protocol
echo  4/ddp   # AppleTalk Echo Protocol
zip   6/ddp   # Zone Information Protocol
```

다음 단계는 /usr/local/atalk/etc 디렉토리 (혹은 여러분이 패키지를 설치한 곳) 에 Appletalk 설정 파일을 만드는 것이다.

만들 첫번째 파일은 /usr/local/atalk/etc/atalkd.conf이다. 기본적으로 이 파일은 애플 머신들이 있는 네트웍을 지원할 네트웍 장치의 이름을 가지는 한 줄만 있으면 된다.

```
eth0
```

Appletalk 데몬 프로그램은 실행된 후에 나머지 자세한 사항을 추가할 것이다.

8.2.2 Appletalk을 통해 리눅스 파일 시스템 노출시키기.

네트웍상의 애플 머신들이 공유할 수 있도록 리눅스 머신의 파일 시스템을 네트웍 상에 노출시킬 수도 있다.

이를 위해선 /usr/local/atalk/etc/AppleVolumes.system 파일을 설정해야 한다. /usr/local/atalk/etc/AppleVolumes.default라는 다른 설정 파일도 있는데 이는 똑같은 형식을 가지면서 guest 권한으로 들어오는 사용자들에게 어떤 파일 시스템을 보여줄 것인가를 지정한다.

이 파일들의 설정 방법과 다양한 옵션들에 대한 자세한 사항은 *afpd* man 페이지에서 찾을 수 있다.

아래는 간단한 예제이다:

```
/tmp Scratch
/home/ftp/pub "Public Area"
```

이 예제는 /tmp 파일 시스템을 'Scratch'라는 AppleShare Volume으로 노출시키고 ftp의 public 디렉토리를 'Public Area'라는 AppleShare Volume으로 노출시킨다. 이 volume 이름은 필수 사항이 아니며 데몬이 임의로 선택해 줄 것이나 이렇게 지정하는 것이 문제되진 않는다.

8.2.3 Appletalk을 통해 리눅스의 프린터 공유하기.

여러분은 리눅스의 프린터를 애플 머신들과 매우 쉽게 공유할 수 있다. Appletalk Printer Access Protocol 데몬인 *papd* 프로그램을 실행시켜 주기만 하면 된다. 이 프로그램을 실행하면 이 프로그램은 애플 머신들로부터 요청을 받아서 프린트 작업을 리눅스의 프린터 데몬에게 보낸다.

이 데몬을 설정하려면 /usr/local/atalk/etc/papd.conf 파일을 고쳐야 한다. 이 파일의 문법은 일반적인 /etc/printcap 파일과 동일하다. 정의에 주는 이름은 Appletalk 명명 프로토콜인 NBP를 통해 등록된다.

간단한 설정 예제가 아래에 있다:

```
TricWriter:\
   :pr=lp:op=cg:
```

위의 예제는 'TricWriter'라는 이름의 프린터를 Appletalk 네트웍 상에 공유시키고 들어오는 모든 작업을 *lpd*을 사용해서 리눅스 프린터 'lp' (/etc/printcap 파일에 정의된 것 처럼) 로 인쇄된다. 'op=cg'라는 엔트리는 'cg'라는 리눅스 사용자가 프린터의 관리자라는 것을 나타낸다.

8.2.4 Appletalk 소프트웨어 시작하기.

이제 여러분은 기본적인 설정을 시험해 볼 수 있다. 잘 작동할 *rc.atalk* 파일이 *nettalk* 패키지 안에서 제공되며 여러분은 다음과 같이 해볼 수 있다:

```
root# /usr/local/atalk/etc/rc.atalk
```

제대로 작동한다면 아무런 에러 메시지 없이 시작되는 각 스테이지를 나타내는 메시지를 콘솔에 뿌려줄 것이다.

8.2.5 Appletalk 소프트웨어 테스트하기.

소프트웨어가 제대로 작동하는지 테스트하기 위해선 애플 머신으로 가서 Apple 메뉴의 Chooser를 선택한 후 AppleShare를 클릭한다. 여기에서 리눅스 박스가 나와야 한다.

8.2.6 Appletalk 소프트웨어 사용시 주의사항.

- IP netmask를 설정하기 전에 Appletalk 지원을 시작해야 할 수도 있다. 만약 Appletalk 프로그램을 시작하는 데 문제가 있거나 시작된 후에 IP 네트웍과 문제가 발생한다면 Appletalk 소프트웨어를 /etc/rc.d/rc.inet1 파일을 실행하기 이전에 시작하도록 해본다.
- *afpd* (Apple Filing Protocol Daemon)은 여러분의 하드디스크를 매우 지저분하게 만든다. 이 데몬은 마운트 지점 아래에 ".AppleDesktop"과 Network Trash Folder라는 두 디렉토리는 만든다. 그리곤 각 여러분이 접근하는 모든 디렉토리 아래에 .AppleDouble 파일을 만들어 resource fork 등을 저장한다. 따라서 후에 디렉토리를 청소하는 데 상당한 시간을 소비해야 하므로 /를 노출시키는 것은 다시 한 번 생각해 봐야 한다.
- *afpd* 프로그램은 Mac들에게 인코딩 되지 않은 상태의 패스워드를 요구한다. 보안이 문제가 될 수 있으므로 이 데몬을 인터넷에 연결된 머신에서 돌리는 것은 매우 신중히 생각해 봐야 한다.
- *netstat*이나 *ifconfig* 같은 현존하는 진단 툴은 Appletalk을 지원하지 않는다. 필요하다면 /proc/net/ 디렉토리에서 가공되지 않은 정보들을 얻을 수 있다.

8.2.7 추가 정보

리눅스에서 Appletalk를 설정하는 방법에 대한 자세한 설명은 thehamptons.com에 있는 Anders Brownworth의 *Linux Nettalk-HOWTO* 페이지를 참고하라.

8.3 ATM

Werner Almesberger <werner.almesberger@lrc.di.epfl.ch>는 리눅스에서 비동기 전송 모드 (Asynchronous Transfer Mode) 지원을 하기 위한 프로젝트를 관리하고 있다. 이 프로젝트의 현 상황에 대한 정보는 lrcwww.epfl.ch에서 얻을 수 있다.

8.4 AX25 (AF_AX25)

AX.25 장치의 이름은 2.0.* 커널에서는 'sl0', 'sl1' 등이고 2.1.* 커널에서는 'ax0', 'ax1' 등이다.

Kernel Compile Options:

```
Networking options  --->
    [*] Amateur Radio AX.25 Level 2
```

AX25와 Netrom, Rose 프로토콜들은 AX25-HOWTO 에서 다루어진다. 이 프로토콜들은 아마추어 라디오 사용자들에 의해서 패킷 라디오 실험에서 널리 사용된다.

이 프로토콜 구현의 대부분의 일은 Jonathon Naylor, jsn@cs.nott.ac.uk가 했다.

8.5 DECNet

DECNet에 대한 지원은 현재 진행중이며 2.1.* 커널의 후반부에 가능하리라 예상된다.

8.6 FDDI

FDDI 장치 이름은 'fddi0', 'fddi1', 'fddi2' 등이다. 커널이 처음 발견하는 카드에 'fddi0'을 부여하고 나머지는 발견되는 순서대로 이름을 할당 받는다.

Larry Stefani, lstefani@ultranet.com, 가 Digital Equipment의 FDDI EISA와 PCI 카드용 드라이버를 만들었다.

Kernel Compile Options:

```
Network device support  --->
    [*] FDDI driver support
    [*] Digital DEFEA and DEFPA adapter support
```

커널을 FDDI 드라이버를 지원하도록 만들어 설치했다면 FDDI 인터페이스의 설정은 이더넷 인터페이스와 동일하다. *ifconfig*와 *route* 명령에서 알맞은 FDDI 인터페이스 이름을 지정해 주기만 하면 된다.

8.7 프래임 릴레이

프래임 릴레이 장치 이름은 DLCI 캡슐화 장치인 경우는 'dlci00', 'dlci01' 등으로, FRAD(s)인 경우는 'sdla0', 'sdla1' 등으로 붙여진다.

프래임 릴레이는 자료의 이동이 폭발적이면서 간헐적인 경우를 위해 설계된 새로운 네트워킹 기술이다. 프래임 릴레이 억세스 장치 (FRAD: Frame Relay Access Device) 를 이용하여 프레임 릴래이 네트웍에 접속할 수 있다. 리눅스의 프레임 릴래이는 RFC-1490에 나온 대로 프레임 릴래이 위에서 IP를 지원한다.

Kernel Compile Options:

```
Network device support  --->
    <*> Frame relay DLCI support (EXPERIMENTAL)
    (24)   Max open DLCI
    (8)   Max DLCI per device
    <*>   SDLA (Sangoma S502/S508) support
```

Mike McLagan, mike.mclagan@linux.org, 이 프레임 릴래이 지원 부분과 설정 툴들을 만들었다.

현재 지원되는 유일한 FRAD는 SangomaTechnologiesS502A, S502E and S508 뿐이다.

커널을 다시 만든 후 FRAD와 DLCI 장치를 설정하기 위해선 프레임 릴레이 설정 툴이 필요하다. 이는 ftp.invlogic.com에서 구할 수 있다. 이 툴들을 컴파일 해서 설치하는 것은 매우 직관적이지만 맨 상위 레벨의 Makefile이 없기 때문에 수작업을 해야만 한다.

```
user% tar xvfz .../frad-0.15.tgz
user% cd frad-0.15
user% for i in common dlci frad; make -C $i clean; make -C $i; done
root# mkdir /etc/frad
root# install -m 644 -o root -g root bin/*.sfm /etc/frad
root# install -m 700 -o root -g root frad/fradcfg /sbin
rppt# install -m 700 -o root -g root dlci/dlcicfg /sbin
```

위의 명령들은 *sh*의 문법을 사용한다는 것을 주의하라. 만약 *tcsh* 같은 *csh* 계열을 사용한다면 *for* 루프가 약간 다를 것이다.

이 툴들을 설치한 후엔 /etc/frad/router.conf 파일을 만들어야 한다. 여러분은 예제 파일 중 하나를 약간 고친 아래의 예제를 사용할 수 있을 것이다.

```
# /etc/frad/router.conf
# This is a template configuration for frame relay.
# All tags are included. The default values are based on the code
# supplied with the DOS drivers for the Sangoma S502A card.
#
# A '#' anywhere in a line constitutes a comment
# Blanks are ignored (you can indent with tabs too)
# Unknown [] entries and unknown keys are ignored
#

[Devices]
Count=1                 # number of devices to configure
Dev_1=sdla0             # the name of a device
#Dev_2=sdla1            # the name of a device

# Specified here, these are applied to all devices and can be overridden for
# each individual board.
#
Access=CPE
Clock=Internal
KBaud=64
Flags=TX
#
# MTU=1500              # Maximum transmit IFrame length, default is 4096
# T391=10               # T391 value    5 - 30, default is 10
# T392=15               # T392 value    5 - 30, default is 15
# N391=6                # N391 value    1 - 255, default is 6
# N392=3                # N392 value    1 - 10, default is 3
# N393=4                # N393 value    1 - 10, default is 4

# Specified here, these set the defaults for all boards
# CIRfwd=16             # CIR forward   1 - 64
# Bc_fwd=16             # Bc forward    1 - 512
```

```
# Be_fwd=0              # Be forward    0 - 511
# CIRbak=16             # CIR backward  1 - 64
# Bc_bak=16             # Bc backward   1 - 512
# Be_bak=0              # Be backward   0 - 511


#
#
# Device specific configuration
#
#

#
# The first device is a Sangoma S502E
#
[sdla0]
Type=Sangoma            # Type of the device to configure, currently only
                        # SANGOMA is recognized
#
# These keys are specific to the 'Sangoma' type
#
# The type of Sangoma board - S502A, S502E, S508
Board=S502E
#
# The name of the test firmware for the Sangoma board
# Testware=/usr/src/frad-0.10/bin/sdla_tst.502
#
# The name of the FR firmware
# Firmware=/usr/src/frad-0.10/bin/frm_rel.502
#
Port=360                # Port for this particular card
Mem=C8                  # Address of memory window, A0-EE, depending on card
IRQ=5                   # IRQ number, do not supply for S502A
DLCIs=1                 # Number of DLCI's attached to this device
DLCI_1=16               # DLCI #1's number, 16 - 991
# DLCI_2=17
# DLCI_3=18
# DLCI_4=19
# DLCI_5=20
#
# Specified here, these apply to this device only,
# and override defaults from above
#
# Access=CPE            # CPE or NODE, default is CPE
# Flags=TXIgnore,RXIgnore,BufferFrames,DropAborted,Stats,MCI,AutoDLCI
# Clock=Internal        # External or Internal, default is Internal
# Baud=128              # Specified baud rate of attached CSU/DSU
# MTU=2048              # Maximum transmit IFrame length, default is 4096
# T391=10               # T391 value    5 - 30, default is 10
# T392=15               # T392 value    5 - 30, default is 15
# N391=6                # N391 value    1 - 255, default is 6
# N392=3                # N392 value    1 - 10, default is 3
# N393=4                # N393 value    1 - 10, default is 4

#
```

```
# The second device is some other card
#
# [sdla1]
# Type=FancyCard        # Type of the device to configure.
# Board=                # Type of Sangoma board
# Key=Value             # values specific to this type of device


#
# DLCI Default configuration parameters
# These may be overridden in the DLCI specific configurations
#
CIRfwd=64               # CIR forward   1 - 64
# Bc_fwd=16             # Bc forward    1 - 512
# Be_fwd=0              # Be forward    0 - 511
# CIRbak=16             # CIR backward  1 - 64
# Bc_bak=16             # Bc backward   1 - 512
# Be_bak=0              # Be backward   0 - 511

#
# DLCI Configuration
# These are all optional. The naming convention is
# [DLCI_D<devicenum>_<DLCI_Num>]
#

[DLCI_D1_16]
# IP=
# Net=
# Mask=
# Flags defined by Sangoma: TXIgnore,RXIgnore,BufferFrames
# DLCIFlags=TXIgnore,RXIgnore,BufferFrames
# CIRfwd=64
# Bc_fwd=512
# Be_fwd=0
# CIRbak=64
# Bc_bak=512
# Be_bak=0

[DLCI_D2_16]
# IP=
# Net=
# Mask=
# Flags defined by Sangoma: TXIgnore,RXIgnore,BufferFrames
# DLCIFlags=TXIgnore,RXIgnore,BufferFrames
# CIRfwd=16
# Bc_fwd=16
# Be_fwd=0
# CIRbak=16
# Bc_bak=16
# Be_bak=0
```

/etc/frad/router.conf 파일을 만들었다면 남은 마지막 과정은 실제 장치들을 설정하는 것이다. 이는 일반적인 네트웍 장치 설정보다 약간 더 까다로우며 FRAD 장치를 DLCI 캡슐화 장치보다 먼저 가동해야 한다는 것을 명심해야 한다. 명령들의 수가 많으므로 쉘 스크립트 안에 넣어두는 것이 좋다.

```
#!/bin/sh
# Configure the frad hardware and the DLCI parameters
/sbin/fradcfg /etc/frad/router.conf || exit 1
/sbin/dlcicfg file /etc/frad/router.conf
#
# Bring up the FRAD device
ifconfig sdla0 up
#
# Configure the DLCI encapsulation interfaces and routing
ifconfig dlci00 192.168.10.1 pointopoint 192.168.10.2 up
route add -net 192.168.10.0 netmask 255.255.255.0 dlci00
#
ifconfig dlci01 192.168.11.1 pointopoint 192.168.11.2 up
route add -net 192.168.11.0 netmask 255.255.255.0 dlci00
#
route add default dev dlci00
#
```

8.8 IPX (AF_IPX)

IPC 프로토콜은 Novell NetWare(tm)의 직역 네트웍 호나경에서 가장 많이 쓰인다. 리눅스는 이 프로토콜을 지원하고 있으며 IPX용 라우터나 네트웍의 한 단말로 작동하도록 설정할 수 있다.

Kernel Compile Options:

```
Networking options  --->
    [*] The IPX protocol
    [ ] Full internal IPX network
```

IPX 프로토콜과 NCPFS는 IPX-HOWTO 에서 더욱 자세히 다루어진다.

8.9 NetRom (AF_NETROM)

NetRom 장치의 이름은 'nr0', 'nr1' 등이다.

Kernel Compile Options:

```
Networking options  --->
    [*] Amateur Radio AX.25 Level 2
    [*] Amateur Radio NET/ROM
```

AX25와 Netrom, Rose 프로토콜들은 AX25-HOWTO에서 다루어진다. 이 프로토콜들은 아마추어 라디오 사용자들에 의해서 패킷 라디오 실험에서 널리 사용된다.

이 프로토콜들의 구현의 대부분은 Jonathon Naylor, jsn@cs.nott.ac.uk에 의해 이루어졌다.

8.10 Rose protocol (AF_ROSE)

Rose 장치 이름은 커널 2.1.*에서 'rs0', 'rs1' 등이다. Rose는 커널 2.1.*에서 지원된다.

Kernel Compile Options:

```
Networking options  --->
    [*] Amateur Radio AX.25 Level 2
    <*> Amateur Radio X.25 PLP (Rose)
```

AX25와 Netrom, Rose 프로토콜들은 AX25-HOWTO에서 다루어진다. 이 프로토콜들은 아마추어 라디오 사용자들에 의해서 패킷 라디오 실험에서 널리 사용된다.

이 프로토콜들의 구현의 대부분은 Jonathon Naylor, jsn@cs.nott.ac.uk에 의해 이루어졌다.

8.11 SAMBA - 'NetBEUI', 'NetBios', 'CIFS' 의 지원.

SAMBA는 Session Management Block 프로토콜을 구현한 것이다. Samba는 Microsoft와 다른 시스템들이 여러분의 디스크와 프린터를 마운트할 수 있게 해준다.

SAMBA와 그 설정은 SMB-HOWTO에서 자세히 다루어진다.

8.12 STRIP 지원 (Starmode Radio IP)

STRIP 장치 이름은 'st0', 'st1' 등이다.

Kernel Compile Options:

```
Network device support  --->
        [*] Network device support
        ....
        [*] Radio network interfaces
        < > STRIP (Metricom starmode radio IP)
```

STRIP은 MosquitoNetProject라 불리는 Stanford의 연구 프로젝트 용으로 설계된 일련의 Metricom 라디오 모뎀용 프로토콜이다. 이 프로젝트에 직접적인 관심이 없을 지라도 이와 관련해선 흥미있는 읽을 거리가 많다.

Metricom 라디오들은 직렬 장치에 연결되어 spread spectrum 기술을 사용하며 보통 약 100kbps의 속도를 낼 수 있다. Metricom 라디오에 대한 정보는 MetricomWebServer에서 볼 수 있다.

현재 Stanford의 네트웍 툴과 유틸리티들은 STRIP 드라이버를 지원하지 않으며 여러분은 MosquitoNet 웹 서버에서 특화된 툴들을 받아야 한다. 여러분에게 어떤 소프트웨어가 필요한지에 대한 자세한 사항은 MosquitoNetSTRIPPage에 나와있다.

대략적인 설정은 *slattach* 프로그램을 수정하여 직렬 tty 장치를 STRIP과 작동하도록 만든 후 그 'st[0-9]' 장치를 이더넷 장치에서와 같이 한다. 여기에 한 가지 예외사항이 있는데, 기술정인 문제로 STRIP은 ARP 프로토콜을 지원하지 않으므로 서브넷 상의 각 호스트들에 대한 ARP 엔트리는 직접 수작업으로 설정해 줘야 한다. 이것이 그렇게 귀찮은 일은 아닐 것이다.

8.13 Token Ring

Token ring 장치 이름은 'tr0', 'tr1' 등이다. Toekn Ring은 한 번에 LAN 상의 한 스테이션(호스트)에게만 자료를 전송할 수 있는 권리를 줌으로써 패킷 충돌을 방지하는, IBM의 표준 LAN 프로토콜이다. 'token'은 한 순간에 한 스테이션에게만 배정된다. 그 스테이션이 자신의 자료를 전송한 후 token을 다음 스테이션에게 넘긴다. token은 작동하는 모든 스테이션들을 돌게 되며 따라서 이름이 'Token Ring'이다.

Kernel Compile Options:

```
Network device support  --->
        [*] Network device support
        ....
        [*] Token Ring driver support
        < > IBM Tropic chipset based adaptor support
```

token ring의 설정은 네트웍 장치의 이름이 다른 것을 제외하곤 이더넷 설정과 동일하다.

8.14 X.25

X.25는 C.C.I.T.T. (전 세계의 전화 통신 회사들로 구성된 표준 제정 위원회)에 의해 정의된 써킷 기반의 패킷 스위칭 프로토콜이다. X.25와 LAPB에 대한 구현이 진행중이며 최신의 2.1.* 커널은 이 일을 포함하고 있다.

Jonathon Naylor jsn@cs.nott.ac.uk 가 이 개발을 주도하고 있으며 리눅스의 X.25 관련 문제들을 토의하기 위한 메일링 리스트가 만들어졌다. 이 리스트에 가입하려면 majordomo@vger.rutgers.edu로 "subscribe linux-x25"란 내용의 메일을 보내면 된다.

설정 툴들의 초기 버젼은 Jonathon의 ftp 사이트인 ftp.cs.nott.ac.uk에서 구할 수 있다.

8.15 WaveLan Card

Wavelan 장치 이름은 'eth0', 'eth1' 등이다.

커널 컴파일 옵션:

```
Network device support  --->
        [*] Network device support
        ....
        [*] Radio network interfaces
        ....
        <*> WaveLAN support
```

WaveLAN 카드는 spread spectrum 무선 랜카드이다. 실제로 이 카드는 이더넷 카드와 매우 닮았으며 거의 같은 방식으로 설정된다.

Wavelan.com에서 Wavelan 카드에 대한 정보를 구할 수 있다.

9 Cables and Cabling 케이블과 케이블 연결

여러분 중에 납땜에 익숙한 사람들은 두 리눅스 머신을 연결하는 케이블을 직접 만들고 싶어할 것이다. 아래의 연결 다이어그램은 이러한 분들에게 도움이 될 것이다.

9.1 Serial NULL Modem cable

모든 NULL modem cable들이 같지는 않다. 많은 널 모뎀 케이블은 단지 자료의 송수신을 맞바꾸고 컴퓨터가 알맞은 신호가 보내진다고 생각하도록 만드는 것 밖에 하지 않는다. 이것도 잘 동작하지만 하드웨어 흐름 제어보다 덜 효율적인 소프트웨어 프름 제어 (XON/XOFF) 를 사용하여야만 한다. 아래의 케이블은 머신간의 가장 최선의 신호 송수신을 제공하며 하드웨어 흐름 제어 (RTS/CTS)를 사용할 수 있도록 해준다.

```
Pin Name  Pin                              Pin
Tx Data    2  -----------------------------  3
Rx Data    3  -----------------------------  2
RTS        4  -----------------------------  5
CTS        5  -----------------------------  4
Ground     7  -----------------------------  7
DTR       20 -\---------------------------  8
DSR        6  -/
RLSD/DCD   8  ---------------------------/-  20
                                         \-  6
```

9.2 Parallel port cable (PLIP cable)

두 머신간에 PLIP 프로토콜을 쓰길 원한다면 이 케이블은 여러분이 가진 패러럴 포트의 종류에 관계 없이 잘 작동할 것이다.

```
Pin Name    pin            pin
STROBE      1*
D0->ERROR   2  ----------- 15
D1->SLCT    3  ----------- 13
D2->PAPOUT  4  ----------- 12
D3->ACK     5  ----------- 10
D4->BUSY    6  ----------- 11
D5          7*
D6          8*
D7          9*
ACK->D3     10 ----------- 5
BUSY->D4    11 ----------- 6
PAPOUT->D2  12 ----------- 4
SLCT->D1    13 ----------- 3
FEED        14*
ERROR->D0   15 ----------- 2
INIT        16*
SLCTIN      17*
GROUND      25 ----------- 25
```

Notes: 알아야 할 것들:

- '*' 표시가 된 핀을 연결하지 말아라.
- 임시 접지는 18, 19, 20, 21, 22, 23, 24 번이다.
- 여러분이 사용하는 케이블이 metallic shield를 가지고 있다면 **한쪽 끝에서만** metallic DB-25 shell로 연결되어야 한다.

주의사항: 잘못 연결된 PLIP 케이블은 여러분의 콘트롤 카드를 망가뜨릴 수 있다. 매우 주의를 하고 잘못된 것이 없는지 모든 연결을 두 번씩 검사하라.

PLIP 케이블을 긴 거리에서 쓸 수는 있지만 가능하면 이를 피해라. 케이블의 명세는 약 1미터으 ㅣ길이도 가능하다고 한다. 전등이나 전원선, 라디오 발신기 같은 강한 전자기장의 원인들이 신호를 간섭하거나 심지어 콘트롤러에 상처를 줄 수도 있으므로 장거리 PILP 케이블을 쓸 때는 매우 주의하여라. 꼭 먼 거리에 있는 두 컴퓨터를 연결해야 한다면 thin-net 이더넷 카드 한 쌍과 동축 케이블을 구하는 것이 좋다.

9.3 10base2 (thin coax) Ethernet Cabling

10base2는 이더넷 케이블 연결의 표준으로 지름이 약 5mm 정도인 52옴짜리 동축 케이블을 사용한다. 머신들을 10base2로 연결할 때 기억해야 할 중요한 규칙이 두 개 있다. 첫째는 캐이블의 **양 끝에** 터미네이터를 써야 한가는 것이다. 터미네이터는 신호가 양 끝에 도달했을 때 반사되지 않고 흡수되도록 하기 위한 52옴짜리 저항이다. 케이블 양 끝에 터미네이터가 없으면 이더넷이 불안정하거나 심지어 작동을 안하는 것을 발견할 것이다. 보통 머신을 연결하기 위해선 'T pieces' 를 사용하며 이렇게 연결한 것은 아래처럼 생겼다.

```
|==========T=============T=============T==========T==========|
           |             |             |          |
           |             |             |          |
         -----         -----         -----      -----
         |   |         |   |         |   |      |   |
         -----         -----         -----      -----
```

양 끝의 '|'는 터미네이터를, '====='는 양(혹은 한쪽) 끝에 BNC 플러그가 달린 동축 케이블은, 'T'는 'T piece' 콘넥터를 나타낸다. 'T piece'와 PC의 이더넷 카드 사이의 길이는 가능한 한 짧게 유지해야 하며 이상적으로는 'T piece'가 이더넷 카드에 직접 연결된다.

9.4 Twisted Pair Ethernet Cable

여러분이 두 개의 twisted pair 이더넷 카드만들 가지고 있고 이들을 연결하길 원한다면 허브는 필요없다. 두 카드를 직접 연결할 수 있다. 이를 하는 방법을 그린 다이어그램은 `Ethernet-HOWTO` 에 포함되어 있다.

10 이 문서에 사용된 용어 해설

아래는 이 문서에서 사용된 매우 중요한 용어들 중 일부의 목록이다.

ARP

*Address Resolution Protocol*의 줄임말이며 네트웍 머신이 IP 주소로 해당 하드웨어 주소를 찾아내는 방법이다.

ATM

*Asynchronous Transfer Mode*의 줄임말이다. ATM 네트웍은 자료를 한 지점에서 한 지점으로 효율적으로 전달할 수 있는 표준 크기의 블럭으로 묶는다. ATM은 circuit switched packet network technology이다.

클라이이언트(client)

일반적으로 사용자가 있는 시스템의 끝부분에 존재하는 소프트웨어의 부분을 의미한다. 이에 대한 예외도 있는데 예를 들어 X11 윈도우 시스템에서는 사용자쪽에 있는 것이 서버(server)이고 멀리 떨어진 머신 쪽에 있는 것이 클라이언트(client)이다. 클라이언트는 서버에 의해 제공되는 서비스를 받는 프로그램이거나 시스템의 말단이다. *slip*이나 *ppp*와 같은 *peer to peer* 시스템에서는 클라이언트가 연결을 초기화하는 말단이 되며 원격지의 말단이 서버가 된다.

데이타그램(datagram)

데이타그램은 주소를 담은 해더와 자료의 이산적인 묶음이며 IP 네트웍 상의 기본적인 통신 단위이다. 이를 '패킷(packet)'이라 부르기도 한다.

DLCI

DLCI는 Data Link Connection Identifier이며 Frame Relay 네트웍을 통한 유일한 virtual point to point 연결을 구분하기 위해 사용된다. DLCI는 일반적으로 Freme Relay 네트웍 공급자에 의해 지정된다.

Frame Relay

Frame Relay는 산발적으로 일어나는 트래픽을 나르기에 이상적으로 적합한 네트웍 기술이다. 많은 Frame Relay 고객이 같은 네트웍 용량을 공유하고 서로 약간씩 다른 시간에 네트웍을 이용하므로써 네트웍 비용을 줄일 수 있다.

Hardware address

media access 층(layer)에서 물리적인 네트웍 상의 호스트를 유일하게 구분하기 위한 숫자이다. 이 예로는 *이더넷 주소(Ethernet Address)*와 *AX.25 Address*가 있다.

ISDN

*Integrated Services Digital Network*의 약자이다. 전화통신 회사가 자료 정보나 음성을 고객에게 전달할 수 있는 표준화된 방법을 제공한다. 기술적으로 ISDN은 circuit switched data network이다.

ISP

Internet Service Provider의 약자이다. 사람들에게 인터넷으로의 네트웍 연결을 제공하는 회사나 기관이다.

IP address

네트웍 상에서 TCP/IP 호스트를 유일하게 구분하는 숫자이다. 4 바이트 길이이며 보통 "dotted decimal notation"이라는 방식으로 표현되는데 이 방식에선 각 바이트가 십진수로 표기되며 바이트 사이엔 '.'를 둔다.

MSS

Maximum Segment Size (*MSS*)은 한 번에 전달될 수 있는 자료의 최대 크기이다. 지역적인 단편화(local fragmentation)을 막고자 한다면 MSS는 MTU-IP 해더와 동일해야 한다.

MTU

Maximum Transmission Unit (*MTU*)은 더 작은 단위로 쪼개질 필요 없이 IP 인터페이스에 의해 전송될 수 있는 데이타그램의 최대 크기를 결정한다. MTU값은 단편화되지 않은 채로 전송시키고자 하는 데이타그램의 최대 크기보다 커야한다. 이는 지역적으로만(locally) 단편화를 막을 뿐 전송 경로 상의 다른 링크가 더 작은 MTU를 가져서 그 곳에서 데이타그램의 단편화가 발생할 수도 있다. 대표적 값은 이더넷 인터페이스에는 1500이고 SLIP 인터페이스에는 576 이다.

라우트(route)

*route*은 데이타그램이 네트웍을 통해 목적지에 가기 위해 거치는 경로이다.

서버(server)

사용자에서 멀리 떨어진 시스템 단말이나 소프트웨어의 일부이다. 서버는 하나 이상의 클라이언트에게 어떤 서비스를 제공한다. 서버의 예로는 *ftp*, *Networked File System*, *Domain Name Server*등이 있다. *slip*이나 *ppp*같은 *peer to peer* 시스템에 있어서 서버는 연결에서 불려지는 말단쪽이며 부르는 말단쪽은 클라이언트이다.

윈도우(window)

The *window*은 수신측에서 한 번에 받을 수 있는 자료의 최대 크기이다.

11 ISP를 위한 리눅스 ?

여러분이 ISP의 용도로 리눅스를 쓰고자 한다면 `LinuxISPhomepage`를 볼 것을 권한다.

12 Acknowledgements

나는 이 문서에 대한 도움에 대해 다음 사람들에게 감사하고 싶다 (무작위순으로): Terry Dawson, Axel Boldt, Arnt Gulbrandsen, Gary Allpike, Cees de Groot, Alan Cox, Jonathon Naylor, Claes Ensson, Ron Nessim, John Minack, Jean-Pierre Cocatrix, Erez Strauss.

13 Copyright.

Copyright Information

역자주: 저작권 관련 부분은 의미 보존을 위해 번역하지 않습니다.