# The Network Administrator's Guide V0.4

Olaf Kirch 1999년 8월 15일

번역자 이 승lvl@chollian.net, 신동원kaien@aapd.metal.pusan.ac.kr

현재 8장까지 번역이 되어 있습니다. 뒷부분을 좀 마무리 지어주세요 !

## 목 록

# 1 Introduction to Networking

## 1.1 역사

네트워킹을 하고자 한 생각은 아마 전자통신만큼 오래되었을 것이다. 석기 시대 때, 살았던 사람들을 잘 관찰해보면, 각 개인의 의사를 전달하기 위해서, 북을 사용해 왔다. 'A'라는 동 굴인이 돌 던지기 게임을 위해, 'B'라는 동굴인을 자기집에 초대하고 싶어한다고 가정하자. 그러나, 그들은 너무 멀리 떨어져서 살고 있기 때문에, 'A'가 북을 쳐서 'B'가 들리게끔 했 다. 그럼 'A'가 할 수 있었던 다른 방법은 없었을까? 1) 그는 'B'가 있는 장소로 직접 걸어 갈 수도 있었고, 더욱 커다란 북을 사용할 수도 있었으며, 그들 사이에서 중간쯤에 살고 있 는 'C'를 메시지를 전달하게 할 수도 있었다. 이 마지막 방법이 바로 네트워킹이다.

물론, 네트워킹은 우리 선조들의 방법과 도구에서 무수히 발전해 온 것이다. 요즈음에는, 토요일 축구시합{{. 유럽에서는 아직도 이와 같은 근본정신을 특별한 날에 보여주고 있다. 약속을 하기위해서, 광학섬유나 마이크로웨이브와 같은 거대한 선로를 통 해서 서로서로 얘기를 주고받을 수 있는 컴퓨터를 가지고 있다. 다음에 전선같은 골치아픈 얘기는 잊어버리고, 축구에 관한 이야기 뿐만 아니라, 이러한 전통이 이루어 지게된 유래에 대해서 다룰 것이다.

우리는 이 안내서에서 두 가지 형태의 네트워킹 즉, UUCP를 기초로 한 것과 TCP/IP를 토대로 한 것을 다루게 될 것이다. 이것들은 두 컴퓨터 사이에서 데이터를 전송하기에 적합 한 프로토콜이며, 소프트웨어 패키지이다. 이 장에서는, 이 두가지 네트워킹에 대한 기본적 인 개념에 대해서 논의할 것이다.

우리는 네트워크를 참여자간에 데이터를 공급해 주는 몇몇 호스트들의 서비스에 의존함 으로써 각 컴퓨터와 통신할 수 있는 *host* 모음으로 정의를 내린다. 호스트들이 컴퓨터 일때 도 있고, 그렇지 않을 때도 있다; 그 호스트는 X-terminal이나 인텔리전트 프린터가 될 수 있다. 소규모 host 집합을 site라 부르기도 한다.

통신은 몇몇 언어나 코드의 정렬없이는 불가능하다. 이러한 언어들을 총괄하여 *protocols* 이라 부른다. 하지만 여러분이 여기에 프로토콜을 쓸 생각은 하지말아야 한다. 이를테면 헤 드가 만날 때 오히려 코드의 동작이 일정한 형태를 갖추었는지 살펴보아라. 매우 유사한 형 태로써, 컴퓨터에서 사용하는 프로토콜들은 네트워크가 두 개 이상의 호스트 사이에서 메시 지를 교환하기 위한 엄격한 규칙에 지나지 않는다.

## 1.2 UUCP Networks

UUCP는 *Unix-to-Unix Copy*를 줄인 말이다. 프로그램 패키지로 일을 시작하는 이 것은 시 리얼 라인을 통해서 파일을 전송하고, 전송된 것을 스케줄하며, 리모트 사이트상에 있는 실 행 프로그램들을 시작시킨다. 70년대 후반에 첫 실행을 거친 이후로 많은 변화를 겪어 왔었 지만, 그것이 제공하는 서비스들은 여전히 스파르타식으로 행해지고 있다. 이것의 주요 어플 리케이션은 여전히 다이얼업 전화 연결을 토대로 하고 있는 광 지역 정보 통신망에서 동작 한다.

처음에 UUCP는 1977년 벨 연구소에서 유닉스 개발 사이트 간에 통신을 하기 위해 개발 되었다. 1978중반에, 이 네트워크는 무려 80개 사이트에 연결되었다. 이것은 리모트 프린팅 뿐만아니라 어플리케이션으로써 전자우편이 통용되고 있었다. 그러나 이 시스템에서 중점적 으로 하는 일은 새로운 소프트웨어를 배포하고, 버그를 고치는 일이었다. 그 시대에는 많은 것을 변화시키진 못했다. 오늘날 UUCP는 더 이상 UN★X에 제한되어 있지 않다. 그리고 AmigaOS, DOS, Atari's TOS 등의 다양한 플랫폼에서 사용할 수 있는 공개용과 상업용 포트가 있다.

UUCP 네트워크의 주요 단점중에 하나는 대역폭이 적다는 것이다. 한편, 최대 전송률을 가지는 전화 설비 지역이 많은 제한을 가진다는 것이다. 다른 한편으로는 UUCP 링크는 그 연결이 거의 영구적이다; 대신에 호스트들은 오히려 규칙적인 간격을 두고 서로 다이얼업 으로 접속한다. 그러므로, 대부분의 시간이 메일 메시지를 UUCP 네트워크로 전송시키는 데 에 소비된다. 그래서 몇몇 호스트의 디스크는 하는일 없이 빈둥거리게 되며, 연결이 성립되 는 다음 시간까지 기다리게 된다.

이러한 제한에도 불구하고, 여전히 많은 UUCP 네트워크가 전세계에서 운영되고 있으며, 개개의 사용자들이 적당한 가격으로 이 네트워크에 접근하고 있다. UUCP가 인기 있는 가 장 주요한 요인으로는 The Big Internet Cable로 연결되어 있는 여러분의 컴퓨터와 비교해 서 그 가격이 매우 싸다는 것이다. UUCP 노드를 가지는 컴퓨터로 만들기 위해 UUCP 실행 이 가능한 하나의 모뎀이 필요하며, 다른 UUCP 노드가 메일과 뉴스를 받는 데에 기꺼이 여러분을 만족시켜 줄 것이다.

### 1.2.1 UUCP를 사용하는 방법

UUCP는 그것의 이름 지시기가 비교적 단순하다: 그것은 기본적으로 하나의 호스트에서 다 른 호스트로 파일을 복사한다. 그것은 또한 리모트 호스트상에서 확실하게 작업을 수행시킨 다. 여러분의 컴퓨터가 swim이라는 이름을 가진 가상의 호스트로 접근해서, 인쇄 명령인 *lpr*을 실행한다고 가정하자. 그러면 여러분은 **swim** - *bash 쉘(GNU Bourne Again Shell)을 사용할 경우, 여러분은 느낌표(!)로 빠져나가야 할 지도 모른다. 왜냐하면 bash는 그것을 history 문자로 사용하기 때문이다.* 상에서 이 책을 인쇄하기 위해 명령행에서 다음을 입력할 수 있다.

```
$ uux -r swim!lpr !netguide.dvi
```

UUCP 형태의 명령어로써 *uux*는 swim을 위한 하나의 *job*을 스케줄한다. 이 작업은 입 력 파일인 *netguide.dvi*로 이루어져 있으며, 이 파일을 *lpr*로 보내주기를 요청할 것이다. -r 옵션은 *uux*에게 직접 리모트 시스템을 부르지 않게끔 말해준다. 하지만 연결이 확립 될 때까 지 작업을 저장시켜준다. 이러한 작업을 *spooling*이라 부른다.

UUCP의 또 다른 특성으로는, 여러 호스트를 거쳐서 작업과 파일들을 표면화 시켜준다 는 점이다. 위 예제에서 본 swim이라는 가상의 호스트가 groucho와 연결한 하나의 UUCP 를 가진다고 가정하자. groucho는 UNIX 어플리케이션의 거대한 아카이브를 유지시켜준다. 여러분의 사이트로 tripwire-1.0.tar.gz 파일을 다운로드 하기 위해 다음과 같은 명령을 쓸 수 있다.

```
$ uucp -mr swim!groucho!~/security/tripwire-1.0.tar.gz trip.tgz
```

생성된 작업은 **groucho**로부터 파일을 가져오기 위해 **swim**을 요청할 것이며, 그 것을 여 러분의 사이트로 보내준다. UUCP는 그 파일을 *trip.tgz*로 저장할 것이며, 파일의 도착 을 알 리는 편지를 여러분에게 통보해 줄 것이다. 이것은 세 단계로 되어 있다. 첫 번째 단계는 여 러분의 사이트가 swim을 위한 작업을 보내준다. 다음 단계로 **swim**이 **groucho** 와 접촉했 을 때, 그 파일을 다운로드한다. 마지막으로, **swim**에서 실제 여러분의 호스트로 파일 을 전 송한다.

오늘날 UUCP 네트워크에서 제공하는 가장 중요한 서비스로는 전자우편과 뉴스가 있다. 우리는 나중에 이러한 것들을 다루게 될 것이다. 이것들에 대한 간단한 소계를 다음 단락에 서 보여주고 있다.

전자우편 - 간략한 소계 - 여러분은 이러한 호스트에 접근하는 방법을 몰라도 리모트 호스트 상에 있는 사용자들과 서로 메시지를 교환하기 위해 전자우편을 사용할 것이다. 여 러분의 사이트에서 목적 사이트로 메시지를 보내는 작업은 메일 처리 시스템에 의해 완전히 수행된다. UUCP 환경에서, 메일은 수신 주소와 메일 메시지를 전하려고 하는 인접호스트 상에서 대개 *rmail* 명령을 주면 메일을 전송하게 된다. *rmail* 명령은 메시지를 다른 호스트 로 전송하게 되며, 그 메시지는 목적 호스트에 도착하게 된다. 우리는 이 부분을 13장에서 자세하게 다룰 것이다.

News는 게시판 시스템에 여러 종류로 분류되어 기술되어 있다. 대개 이러한 형태를 유 즈넷 뉴스라고 부르며, 이것은 참여하고 있는 사이트 수가 무려 120,000개에 달하는 뉴스 교 환망 으로 가장 널리 알려져 있다. 최초의 유즈넷 데이터는 1979년으로 거슬러 올라가 보면, 새로운 Unix V7과 함께 UUCP가 배포된 이후에, 세명의 졸업생들이 Unix 환경하에서 일반 적인 정보를 교환하자는 생각에서 출발하였다. 그들은 몇몇 쉘 스크립트로 뉴스 프로그램을 작성하였고, 이것이 최초의 넷뉴스 시스템이 되었다. 1980년에 이 네트워크는 North Carolina 대학에서 만들어낸 유즈넷 최초의 사이트인 **duke, unc** 그리고 **phs**를 연 결하였다. 그리고 나서, 유즈넷은 마침내 성장하기 시작하였다. 비록 유즈넷이 UUCP를 기반으로 하는 최초의 네트워크임에도 불구하고, 더 이상 하나의 독립된 형태의 네트워크로 제한하지 않는 다.

정보에 있어서 가장 기본적인 단위는 기사이다. 하나의 계층화된 뉴스그룹들이 특정 주 제별로 제공되는 기사들을 게시할 것이다. 모든 뉴스그룹들이 대부분의 사이트를 선택하게 되며, 가치있는 기사들이 하루에 60MB 정도 등록된다.

UUCP 세계에서, 뉴스는 요청한 그룹들로부터 모든 기사들을 모아놓고, 몇 개의 *batches* 라고 하는 곳에 그것들을 묶어놓고, UUCP 링크를 거쳐서 보내게 된다. 이것들은 수신 사이 트에 보내지게 되며, 나중에 이것들을 풀기 위해서는 *rnews* 명령을 사용하면 된다.

마지막으로, UUCP는 공용 엑세스를 제공하기 위한 많은 다이얼 업 아카이브 사이트를 위한 선택매체가 될 수도 있다. 여러분은 UUCP를 가지고 그것들을 다이얼 업하거나, guest 사용자로 로긴해서, 그 사이트로 접근할 수 있으며, 공개적으로 접근가능한 아카이브 지역에 서 파일들을 전송받을 수도 있다. 이들 guest계정은 대개 로긴명과 패스워드로 **uucp/nuucp** 또는 그와 유사한 것들을 가진다.

## 1.3 TCP/IP Networks

비록 UUCP를 선택하는 이유가 적은 비용으로도 네트워크로 연결할 수 있다는 점이라할지 라도, store-and-forward 기술에서는 다루기 힘든 상태가 많이 있음이 증명되었다. 예를 들 어 Local Area Networks (LANs). 이러한 것들은 대개 같은 빌딩이나 심지어는 같은 층에 위치하고 있는 소수의 컴퓨터로 구성되어 있다. 그것은 동질의 작업환경을 제공해주기 위해 연결되어 있다. 전형적으로, 여러분은 이러한 호스트들 사이에서 파일을 공유하거나, 다른 기계에 깔려있는 어플리케이션들을 실행하고 싶어할 것이다.

이러한 작업은 네트워크로 연결을 할 때, 완전히 다른 접근법이 필요하다. 작업지시기에 따라 파일 전체를 발송하지 않고, 모든 자료를 더 작은 덩어리 (패킷)로 나누어 놓고, 목적 호스트 즉, 그것들이 모여있는 곳으로 즉시 발송한다. 이 네트워크 형태를 *packet-switched* 네트워크라고 부른다. 다른 네트워크 중에서도 이것은 네트워크를 통한 대화식 어플리케이 션을 실행하기 위한 것이다. 물론 소프트웨어적인 면에서 보면 대단히 복잡해진다.

UNIX 시스템과 비 UNIX 사이트에서 가지고 있는 해결방안으로써 TCP/IP라고 알려진 프로토콜을 채택하고 있다. 이 절에서는 TCP/IP에 관한 기초적인 개념들을 다루게 된다.

### 1.3.1 Introduction to TCP/IP-Networks

TCP/IP는 1969년 미 국방성에서 DARPA (Defense Advanced Research Projects Agency) 라는 연구 프로젝트로 시작하였다. 이것은 ARPANET라는 실험용 네트워크로써 1975년에 일반인들도 사용가능한 형태로 변환되었다.

1983년에, 새로운 형태의 프로토콜인 TCP/IP가 표준 프로토콜로 채택되었으며, 그 네트 워크에 있는 모든 호스트들은 이 네트워크를 사용하기 위해 TCP/IP 프로토콜을 필요로 하 게 되었다. ARPANET이 마침내 인터넷 (1990년, ARPANET 그 자체가 또 다른 형태로 변 한것)으로 성장하였을 때 TCP/IP의 사용은 인터넷의 한계를 넘어서 네트워크로 퍼져나갔 다. 가장 주목할 만한 것으로는 UNIX 지역 통신망을 들 수 있지만, ISDN과 같은 빠른 디 지털 전화 설비의 출현이 있은 후, 그것은 또한 미래에는 다이얼업 네트워크를 위한 전송도 가능하리라고 본다.

다음 절에서는 TCP/IP에 관한 자세한 정보를 논의해 볼까 한다. 하나의 예를 들자면, Fredland의 어딘가에 위치하고 있을 Groucho Marx University (GMU)를 고려해 볼 것이다. 몇몇 부서들은 하나의 망을 공유하는 경우도 있고, 그 밖의 대부분의 부서들은 자체적으로 하나의 망을 가지는 경우, 또 다른 경우로 하나의 부서가 여러개의 망을 가지는 경우가 있 다. 이것들은 서로 연결되어 있으며, 단독 고속 링크를 통하여 인터넷으로 떠나기도 한다.

여러분의 컴퓨터가 Mathematics Department에 있는 UNIX 호스트 (그 이름은 **erdos**) 에 LAN으로 연결되어 있다고 가정하자. **quark**라고 부르는 Physics Department에 있는 호스 트로 접근하기 위해, 다음과 같은 명령을 입력하라.

```
$ rlogin quark.physics
Welcome to the Physics Department at GMU
(ttyq2) login:
```

프롬프트에서, **andres**라고 하는 여러분의 로긴명과 패스워드를 입력하라. 그러면 **quark** 에서 마치 시스템의 콘솔 환경에 여러분이 안주해 있는 것처럼, 입력할 수 있게 하기 위하 여, 하나의 쉘을 준다. 여러분이 그 쉘을 빠져나가면, 다시 여러분 자신의 기계로 되돌아 가 게 된다. 여러분은 동시에 작용할 수 있는 것, 즉 TCP/IP에서 제공하고 있는 대화식 어플리 케이션인 remote login을 사용하여 왔었다.

여러분은 quark로 접속해 들어가는 동안 플로팅 프로그램이나, PostScript previewer 기 능과 같이, X11을 토대로 하는 어플리케이션을 쓰고 싶어 할지도 모른다. 여러분의 호스트 화면상에 이러한 어플리케이션을 출력하기 위해서는 DISPLAY 환경 변수를 설정해야 한다:

```
$ export DISPLAY=erdos.maths:0.0
```

만약 어플리케이션을 지금 당장 실행시킨다면, quark 대신에 X 서버와 접촉하게 될 것 이고, 화면에는 풀 윈도우로 표현될 것이다. 물론, 이것은 여러분이 erdos상에서 X11을 실 행시킬 필요가 있다. 여기에 있는 이 점은 TCP/IP가 **quark**와 **erdos**를 허가하는 것이며, 이 것은 앞으로는 X11 패킷을 보내고, 뒤로는 여러분이 단독 시스템에 있는 것같이 만들어 준 다. 이 네트워크는 거의 여기에선 앞뒤가 투명하게 되어 있다.

TCP/IP 네트워크에서 또 하나 중요한 어플리케이션으로는 NFS (표준 Network Fiie Sy stem)을 들 수 있다. 이것은 네트워크 투명성을 표시하는 또다른 형태이다. 왜냐하면, 그것 은 기본적으로 다른 호스트로부터 디렉토리 계층을 마운트하기 때문이다. 그래서, 그것들이 로컬 파일 시스템으로 나타난다. 예를 들어, 사용자의 홈 디렉터리들는 LAN상에 있는 다른 모든 호스트들을 디렉터리와 마운트함으로써, 중앙 서버에 존재할 수 있다. 이 작업은 사용 자들이 어떤 호스트든지 접근해 들어갈 수 있으며, 같은 홈 디렉터리에 있는 사용자 파일을 발견하게 된다. 유사하게, 오직 한 대의 기계에 TEX와 같은 거대한 양의 디스크 영역이 필 요한 어플리케이션을 설치하는 것이나, 다른 기계에 이러한 디렉터리들을 전송하는 것도 가 능하게 되었다. 우리는 11장에서 NFS에 대해 자세하게 다룰 것이다.

물론, 이것들은 TCP/IP 네트워크에서 여러분이 할 수 있는 것중 한가지 예에 불과하다. TCP/IP 네트워크에서는 무한한 가능성을 내다볼 수 있다.

우리는 지금 TCP/IP를 구현할 수 있는 방법에 아주 가까워져 가고 있다. 다음절에서는 하드웨어를 예로 들면서, 작업에 몰두해 보자.

### 1.3.2 Ethernets

LAN을 통해서 사용하는 하드웨어 형태중에서 일반적으로 가장 널리 사용하는 것이 Ether- net이다. 이더넷은 커넥터, 탭이나 트랜스 시버를 통하여 그것에 접속하게 되는 하나의 단독 케이블로 이루어져있다. 초당 10M bit를 전송할 수 있는 이더넷이 그다지 비싸지 않기 때문 에 상당한 인기를 구가하고 있다.

이더넷에는 세 가지 기본적인 요소 즉, thick, thin 그리고 twisted pair로 이루어져 있다. Thin과 Thick 이더넷는 각각 하나의 동축케이블을 사용하고 있으며. 대역과 여러분이 이 케이블에 호스트를 접속시키는 방법이 다르다. Thin Ethernet은 꼬인선에 접속되어 있는 T 형 "BNC" 커넥터를 컴퓨터 뒷부분에 있는 플러그에 꽂아 넣는다. Thick Ethernet은 핀을 이용해서 선에 작은 구멍을 뚫고, 거기에 트랜스 시버를 꽂아 넣는다. 여러개의 호스트를 트 랜스 시버에 연결할 수 있다. Thin 과 thick Ethernet 선은 각각 최대 200 또는 500미터까 지 사용할 수 있고, 이것을 10base-2 그리고 10base-5라고 부른다. Twisted pair는 원래 전 화 설치시 찾을 수 있었던, 두 개의 동선으로 이루어진 케이블이다. 그러나 대개 10base-T 라고 알려진 하드웨어가 추가적으로 필요하다.

비록 thick Ethernet에 호스트를 추가시키는 작업이 약간은 힘들다 할지라도, 그것은 네 트워크를 망가뜨리지 않는다. thinnet 설치시 호스트를 추가하기 위해서는, 적어도 몇분만이 라도 네트워크 서비스를 중단시켜 두어야 한다. 왜냐하면, 커넥터에 꽂을 선을 잘라야 하기 때문이다.

대부분의 사람들을 가격이 싸다는 이유로 thin Ethernet을 더 좋아하는 경향이 있다: PC 카드는 적어도 US 달러로 $50정도 되고, 전선은 미터당 2내지 3센트정도이다. 그러나 대용 량 필요로 하는 곳에는 thick Ethernet가 더 적당하다. 예를 들면, GMU의 수학부는 thick Ethernet를 사용한다. 그래서, 네트워크에 호스트를 추가할 때마다 서비스를 중단시키는 일 은 없을 것이다.

이더넷 기술의 약점이라고 한다면, 케이블 길이에 제한이 있다는 것이다. 그래서, LAN을 사용할 경우, 방해가 되는 부분이다. 그러나, 여러 이더넷 부분들은 리피터, 브릿지, 또는 라 우터를 사용해서, 서로를 연결할 지도 모른다. 리피터는 단순히 두 개 이상의 요소들 시이에 있는 신호들을 복사한다. 그래서, 모든 부분들이 하나의 이더넷인 것처럼 행동한다. 필요 조 건이라면, 네트워크에다가 두 개의 호스트에 네 개이상의 호스트를 달순없다. 브리지와 라우터는 더욱더 복잡하게 되어 있다. 이것들은 들어오는 데이터를 분석해서, 로컬 호스트상에 수신 호스트가 없다면, 그것을 앞쪽으로 끄집어 낸다.

이더넷은 하나의 호스트가 같은 이더넷상에 있는 다른 호스트로 최고 1500바이트 패킷 (또는 프레임)을 보내주는 버스 시스템처럼 작동한다. 그 호스트는 이더넷 보드의 펌웨어로 여섯 바이트씩 주소화되어 있다. 이러한 주소들은 대개 두 개의 숫자가 콜론으로 구별되어 여섯 개씩 순차적으로 쓰여져있다. 예를 들어, **aa:bb:cc:dd:ee:ff**.

프레임은 하나의 스테이션이 마치 접속되어 있는 모든 스테이션처럼 보이게끔 해서 보낸 다. 하지만 목적 호스트는 실제로 스테이션을 찾아내어서 처리한다. 만약 두 개의 스테이션 을 동시에 보내려고 시도했을 때, 발생하는 충돌은 두 개의 스테이션의 보내기를 중지시킴 으로써 그러한 문제가 해결되며, 몇분후에 재시도한다.

### 1.3.3 Other Types of Hardware

Groucho Marx University와 같은 거대한 장소에서, 이더넷는 오직 하나의 형태로 사용되는 것은 아니다. Groucho Marx University에서, LAN의 각 부는 campus backbone으로 연결되 어 있고, 그것은 FDDI (*Fiber Distributed Data Interface*)를 사용하는 광학섬유전선 이다. FDDI는 전송중인 자료를 완전히 다르게 접근하여 사용한다. 이것은 기본적으로, 여기저기에 보내는 즉 다시말해서, 만약 그것이 토큰을 포착한다면 하나의 스테이션이 단지 프레임을 보내기 위해 허가하게 될 *tokens*의 수를 포함한다. FDDI의 주요 이점으로는 100Mbps 의 속도를 낼 수 있고, 최대 선길이가 최고 200km까지 가능하다는 것이다.

먼거리의 네트워크을 연결하기 위해, 다른 종류의 기계가 자주 사용되며, 그 기계는 X.25 에 기초를 두고 있다. U.S.에 있는 Tymnet나 독일에 있는 Datex-P와 같은 Public Data N- etwork는 이 서비스를 제공하고 있다. X.25는 즉, Packet Assembler/Disassembler 또는 PAD와 같은 특별한 하드웨어를 필요로 한다. X.25는 네트워킹 프로토콜을 정의함에도 불구 하고, TCP/IP 그리고 다른 프로토콜을 사용하고 있는 네트워크를 접속하기 위해 자주 사용 된다. IP 패킷이 X.25에 정밀하게 표시할 수 없게된 이후에, 그것들은 단순히 X.25에 싸여서 네트워크에 보내지게 된다.

자주, 무선 아마추어들은 네트워크에 접속하기 위해 대개 그들의 컴퓨터를 장비로 사용 한다: 이것은 *packet radio* 또는 *ham radio*라 부른다. ham radio에 의해 사용되 는 프로토콜 을 우리는 AX.25라 부른다. 이것은 X.25에서 유래한 것이다.

다른 기술로는 사용자체가 좀 느리지만 값은 싼 다이얼업 엑세스를 위한 시리얼 라인을 포함하고 있다. 이것들은 이직도 패킷을 보내기 위해, SLIP나 PPP와 같은 또 다른 프로토 콜을 필요로 한다. 이것은 아래에 기술되어 있다.

### 1.3.4 The Internet Protocol

물론 여러분의 네트워크를 하나의 이더넷으로 제한하길 원치 않을 것이다. 이상적으로 말하 면, 어떤 하드웨어를 사용하고 있는지 또는 얼마나 많은 서브유니트를 가지고 있는지에 상 관없이 네트워크를 사용하고 싶어할 것이다. 예를 들어, Groucho Marx University와 같은 거대한 장소에서, 여러분은 대개 여러 가지 방법으로 접속해야 하고, 여러개로 분리되어 있 는 이더넷를 가지고 있을 것이다. GMU에서, 수학부는 두 개의 이더넷s을 사용한다: 하나는 교수들이나 졸업생들을 위해 빠른 기계를 사용하는 네트워크와 또 다른 하나는 학생들을 위해 조금 더 느린 기계를 사용하는 네트워크가 있다. 둘다 FDDI campus backbone에 연결되 어 있다.

이 연결은 이른바 *gateway*라고 하는 제공된 호스트에 의해 처리된다. 게이트웨이는 두 개의 이더넷과 광학섬유전선 사이에서 그것들을 복사함으로써, 들어오는 패킷과 나가는 패 킷을 처리한다. 예를 들어, 만약 여러분이 Maths Department에 있고, 리눅스 컴퓨터에서 Physics Department의 LAN 상에 있는 **quark** 호스트로 접근하고 싶다면, 네트워킹 소 프트 웨어는 패킷을 **quark**로 직접 보낼 수 없다. 왜냐하면, 같은 이더넷상에 있는 것이 아 니기 때문이다. 그래서 게이트웨이가 운송업자 역할을 한다. 백본을 사용해서, **sophus**라 이 름지 어진 게이트웨이는 Physics Department에 있는 동급의 게이트웨이인 niels에게 이들 패킷 을 보낸다. **niels**는 목적 호스트로 패킷을 전달하는 역할을 한다. **erdos**와 **quark**의 데이터 흐름도는 그림 1.1에 나와 있다.

그림 1.1: erdos에서 quark으로 자료를 세 단계로 보내는 과정

리모트 호스트로 보내는 자료의 방향을 계획하는 작업을 *routing*라고 하며, 이러한 관계로 볼 때, 패킷은 대개 *datagrams*에 적용된다. 이러한 작업을 용이하게 하기 위해, 하드 웨어와 독립적으로 사용되는 단독 프로토콜 즉, IP 또는 *Internet Protocol*이 자료 교환작업을 제어 한다. 2장에서, IP와 라우팅에 관해 좀 더 상세하게 다룰 것이다.

IP의 주요 잇점으로는 물리적으로 다른 네트워크를 외관상으로 동질의 네트워크로 변화 시켜준다. 이것을 인터네트워킹이라고 하고, 그 결과 발생하는 ”meta-network”를 *internet*이 라 부른다. 여기에서 *an internet*과 *the Internet*은 미묘한 차이점이 있다는 것을 주의하라.

물론, IP는 또한 하드웨어를 독립적으로 어드레싱하는 작업이 필요하다. 이러한 작업은 IP 어드레스라고 부른는 하나의 유일무일한 32비트 수를 각 호스트에 할당함으로써 완성된 다. 하나의 IP 어드레스는 대개 네 개의 십진수를 도트문자로 구별해놓고, 각자리에 8비트씩 분배해 놓는다. 예를 들어, **quark**는 **0x954C0C04**라는 IP 어드레스를 가지고 있고, 그것은 다시 **149.76.12.4**로 표현한다. 이러한 형태를 *dotted quad notation*이라고 부른다.

자 그럼, 여러분은 우리가 세가지 다른 형태의 주소를 가지고 있다고 말할 것이다. 즉, 첫 번째는 **quark**와 같은 호스트명, 그리고 IP 어드레스, 마지막으로, 6바이트 이더넷 주소와 같은 하드웨어 주소가 있다. 어쨌든간에, 이러한 모든 주소들이 하나같이 일치해야된다. 그 래서, 여러분이 *rlogin quark*라고 입력하면, 네트워킹 소프트웨어는 **quark**의 IP 어드레스를 줄 수 있게 된다. 즉, IP가 어떤 자료를 Physics Department’s 이더넷로 넘겨줄 때, 그것은 어떻게해서 든지 이더넷 어드레스를 IP 어드레스와 일치시켜야 한다.

지금 이점에 대해서 자세하게 논의할 순 없지만, 2장에서 이것을 다루기로 하겠다. 지금 은 *hostname resolution*이라고 부르는 주소들을 찾는 단계와 호스트 명을 IP 어드레 스와 일치시키는 것, 문자들을 하드웨어 주소로 일치시키는 과정을 기억하는 것만으로도 충분하 다.

### 1.3.5 IP over Serial Lines

사실 시리얼 라인에서, SLIP 또는 Serial Line IP라고 알려진 표준 프로토콜이 자주 쓰인다. CSLIP 또는 compress SLIP는 SLIP을 변형시킨 것이며, 이것은 시리얼 링크에 의해 제공되 는 대역폭을 상대적으로 낮게 사용하기 위해서 IP 헤더를 압축하는 작업을 한다. - *SLIP은 RFC 1055에 기술되어 있다. 헤더를 압축하는 작업을 하는 CSLIP는 RFC 1144를 토대로 해서, 기술되어 있다.* PPP 또는 *Point-to-Point Protocol*이라고 하는 또 다른 시리얼 프로토콜이 있다. PPP는 SLIP보다 더 많은 특징을 가지고 있다. SLIP에서는 제공하지 못하는 PPP만의 주요한 이점 으로는 IP 데이터그램을 전송하는 데에 제한이 없다는 것이다. 그것은 전달되는 어떠한 형 태의 데이터그램도 허용할 수 있게끔 제작되어 있다.

### 1.3.6 The Transmission Control Protocol

물론 요즈음에는 하나의 호스트에서 다른 호스트로 자료를 보내는 기능만 있는 것은 아니 다. 만약 여러분이 **quark**로 접속하고자 한다면, **erdos**상에 있는 **rlogin** 프로세스와 **quark** 상에 있는 쉘 프로세스 사이에 믿을 수 있는 연결을 가지고 싶어할 것이다. 그리하여, 이 정 보가 보내지고 이것은 송신기에 의해 패킷으로 나누어지게 되며, 수신기에 의해 문자 스트림으로 다시 합쳐지게 되는 것이다. 이것이 사소한 것처럼 보이지만 매우 어려운 작업을 수 반하고 있다.

IP에 관한 지식이 매우 중요하긴 하지만 그렇게 믿을 수 있는 것은 아니다. 여러분의 E- thernet상에 있는 열 명의 사람이 GMU의 FTP서버로부터 XFree86 최신 배포본을 전송받는 다고 가정하자. 여기서 발생하는 부하량은 실로 엄청날 것이며, 이것을 게이트웨이가 처리할 것이다. 왜냐하면, 전송속도가 매우 느릴 것이고, 메모리의 양이 부족할 지도 모르기 때문이 다. 지금 만약 여러분이 quark로 패킷을 보내고자 한다면, sophus가 잠시동안 버퍼 영역을 벗어날지도 모르기 때문에 그러한 것을 기대하기란 어렵다. IP는 단순하게 그것을 삭제함으 로써 그러한 문제를 해결한다. 그러면 패킷은 사라지며, 그것은 다시 되부를 수도 없다. 데 이터를 보존하고 완성하며, 에러를 찾아내어서 재전송하는 것이 통신 호스트의 주요 임무이 다.

이러한 작업은 아직도 TCP 또는 *Transmission Control Protocol*이라고 하는 또 다 는 프로토콜에 의해 수행되며, IP의 최상위에서 작업한다. TCP 본질적인 특성이라고 한다면, 여러분의 호스트와 리모트 머신상에 있는 두 개의 프로세스들을 단순히 연결시켜주는 착각 을 일으키게 하기 위해 IP를 사용하는 것이다. 그래서, 여러분은 자료가 어떤 경로를 거치는 지는 알 필요가 없다. TCP 연결은 본질적으로 읽기도 하고 쓰기도 하는 프로세스 둘 다를 가지고 있는 송수신 파이프와 같이 동작한다. 즉 전화통화를 생각해 보면 된다.

TCP는 두 개의 호스트를 수반하고 있는 IP를 거친 연결의 종점과 각 호스트상에 있는 이른바 *port* 수를 동일하게 간주한다. 포트들은 네트워크 연결을 위한 연결장치 관 점에서 본 것이다. 한가지 예를 들어 만약 여러분이 전화선을 변형시킬 수 있다면, IP 어드레스는 지역 코드 ( 즉, 도시와 연관시킬 수 있는 숫자)와 비교할 수 있고, 포트 번호는 로컬 코드 (즉, 각 개인의 전화와 연관시킬 수 있는 숫자)와 비교할 수 있다.

*rlogin*을 예로 들어 보면, 클라이언트 어플리케이션 (*rlogin*)은 **erdos**상에 있는 하나의 포트를 열어주고, quark상에 있는 포트 번호 513에 연결시키며 rlogind 서버가 그 뒤를 따르는 것으로 알려져 있다. 이것으로 TCP 연결을 확립시킨다. 이러한 연결을 사용해서, rlogind가 인증 절차를 수행시키면 쉘이 나타나게 된다. 그 쉘의 표준 입력과 출력을 TCP가 연결되어 있는 곳에 전송시킨다. 그래서 여러분의 기계에서 rlogin라고 입력하게 되면, 이 입력된 신호가 TCP 스트림을 통과하게 될 것이고, 쉘의 표준 입력으로 받아들여지게 되는 것이다.

### 1.3.7 The User Datagram Protocol

물론 TCP가 TCP 네트워킹에서 사용자 프로토콜로써만 존재하는 것은 아니다. 비록 rlogin 과 같은 어플리케이션에 적합한 프로토콜이라 하더라도, 그것에 수반되어 있는 오버헤드는 NFS와 같은 어플리케이션에는 대단히 부적합하다. 대신에, TCP와 유사한 프로토콜인 UDP 또는 *User Datagram Protocol*을 사용한다. TCP와 같이 UDP 또한 리모트 머신상에 있는 어떤 포트에 서비스를 접속하기 위해 하나의 어플리케이션을 허용하고 있지만, 이것을 위한 연결을 확립해 놓진 않는다. 대신에, 여러분이 단독 패킷을 목적 서비스에 보내기 위해 사용할 수도 있다.

여러분이 각 부의 중앙 NFS 서버 - *galois*로부터 계층적으로 TEX 디렉토리에 마운 트 되어 있고, LATEX 사

용방법에 대해 기술해 놓은 문서를 보고 싶어한다고 가정하자. 우선 파일 전체를 에디터로 읽어 들여라. 하지만, galois로 TCP 연결을 확립하고, 파일을 보내고, 그것을 다시 배포하는 데에는 너무나도 많은 시간이 걸릴 것이다. 대신에, galois로 만들어 진 하나의 요청 즉, 이것은 한쌍의 UDP 패킷에 있는 파일을 보내는 것이며, 속도면에서 훨 씬 더 빠르다. 하지만 UDP는 손실된 패킷이나 충돌이 일어난 패킷을 보존하지 않는다. 이 러한 경우에 가장 적절한 어플리케이션으로는 NFS가 있으며, 이것은 그러한 패킷들을 보호 해준다.

### 1.3.8 More on Ports

포트는 네트워크 연결을 위한 연결 포인트로 볼 수 있다. 만약 하나의 어플리케이션이 어떤 서비스를 제공하고자 한다면, 그것은 하나의 포트에 그 자체를 연결시키고, 클라이언트를 기 다린다. (이것을 포트에 *listening* 한다고 부른다.) 이 서비스를 사용하길 원하는 클라 이언트 는 로컬 호스트에 하나의 포트를 할당하고, 리모트 호스트 상에 있는 서버의 포트에 연결시 킨다.

포트의 중요한 특성중에 하나로는 연결이 클라이언트와 서버사이에서 이루어지고, 서버 의 다른 복사본들이 서버 포트에 연결되며, 더욱더 많은 클라이언트를 위해 listen한다. 이를 테면, 이것은 모두다가 같은 포트 513을 사용해서, 같은 호스트에 여러 다른 원격 접속을 동 시에 허가한다. TCP는 이러한 서로를 간에 연결을 확립할 수 있다. 왜냐하면, 그것들이 모 두 다른 호스트나 포트에서 발달한 것이기 때문이다. 예를 들어, 만약 여러분이 **erdos** 에서 **quark**로 접속한다면, 첫 번째 rlogin 클라이언트가 로컬 포트 1023을 사용할 것이고, 두 번 째 클라이언트는 포트 1022를 사용할 것이다. 하지만 둘 다는 **quark**의 포트 513에 연 결될 것이다.

이 예제에서 포트의 사용은 하나의 클라이언트가 특별한 서비스를 얻기 위해서 특별한 포 트를 연결하는 랑데부 포인트로 볼 수 있다. 클라이언트의 순서를 위해 적절한 포트 번호를 식별하기 위해서는, 이러한 번호를 할당할 수 있는 양쪽의 시스템 관리자사이에 그러한 합 의가 이루어져야 한다. rlogin과 같이 널리 사용되는 서비스를 위해, 이러한 번호들은 중점 적으로 관리되어야 한다. 이것은 IETF - *Internet Engineering Task Force*에 의해 이 루어 지며, 그것은 할당 번호가 붙은 RFC를 정기적으로 배포한다. 이것은 다른 것들 중에 *well-known services*로 할당된 포트 번호들을 기술한다. 리눅스는 그러한 번호 를 위해 /etc/services라고 부르는 파일 매핑 서비스 명을 사용한다. 그것은 The services and proto cols Files (9.3절)에서 자세하게 기술할 것이다.

비록 TCP 와 UDP 연결이 포트들에 의존하고 있다 하더라도 이들 번호들은 절대 충돌 이 일어나지 않는다. 이 의미는 TCP 포트 513은 UDP 포트 513과 다르다는 것이다. 사실상, 이들 포트들은 두 개의 다른 서비스 즉, rlogin (TCP) 와 rwho (UDP)와 같은 두 개의 다른 서비스를 엑세스 포인트로 제공한다.

### 1.3.9 The Socket Library

UNIX 운영 체제에서, 모든 작업과 위에서 기술한 프로토콜을 수행하는 소프트웨어는 대개 리눅스에서와 같이 커널의 일부분이다. UNIX 세계에서 가장 일반적으로 사용하는 프로그 래밍 인터페이스는 Berkeley Socket Library이다. 그것의 이름은 소켓을 포트로 보고 플러 그를 꽂아 접속하는 것과 같이 포트를 연결한다는 유추에서 유래한 것이다. 그것은 리모트 호스트와 전송 프로토콜 그리고 서비스를 명시하기 위해 (*bind(2)*) 호출을 사용한다. 이것으 로 인해 프로그램은 (*using connect(2), listen(2),* 그리고 *accept(2)*)를 연결하거나 들을 수 있다. 소켓 라이브러리가 다소 보편적이기는 하지만, 그것은 소켓 (AF_INET 소켓)을 기본 으로 하는 TCP/IP 클래스 뿐만아니라 연결 지역을 기계 (AF_UNIX 클래스)로 조종하는 클래스를 제공한다. 몇몇 실행으로 XNS (*Xerox Networking System*) 프로토콜 또는 X.25 와 같은 또 다른 클래스 또 한 처리할 수 있다.

리눅스에서, 소켓 라이브러리는 표준 *libc C* 라이브러리의 일부분이다. 현재, 그것은 *AF_INET*와 *AF_UNIX* 소켓만을 지원하지만, Novell의 네트워킹 프로토콜 지 원을 통합시 키는 노력으로 인해, 마침내 하나 이상의 소켓 클래스를 통합시킬 수 있게 되었다.

## 1.4 Linux Networking

리눅스는 전세계의 프로그래머들이 이루어낸 노력의 결과이며, 전세계 네트워크 없이는 가 능하지 못했다. 이미 초기 단계에서 여러 사람들이 네트워크 호환 작업을 이루어낸 것도 과 히 놀랄만한 것도 아니다. 이미 초기

단계에서 UUCP를 리눅스 상에서 실행에 옮겼으며, 1992년 가을에 Ross Biro와 다른 사람들이 TCP/IP를 기초로한 네트워킹을 시작하였고, 그 것은 Net-1으로 알려지게 되었다.

1993년 Ross가 개발 활동을 중단한 이후, Fred van Kempen이 새롭게 작업에 착수하기 시작하였고, 그러한 노력으로, Net-2를 만들어 내게 되었다. 1992년 여름에 첫 공식 배포본 인 Net-2d를 만들어 냈다. (이것은 0.99.10 커널의 일부분이다.) 그리고 여러 사람들 중에서 Alan Cox가 Net-2Debugged를 유지하고 실험하고 있었다. 심각한 버그를 수정하고, 코드에 여러 가지 수정작업이 이루어진 이후로, 그 이름이 Net-3으로 바뀜으로써 드디어 Linux 1.0 을 배포하기에 이르렀다. 현재에는 여러 가지 네트워킹 코드가 공식 커널 배포본에 포함되 어 있다.

Net-3는 가장 광범위하게 변화하는 이더넷 보드 뿐만아니라, SLIP (시리얼 라인을 통해 네트워크 전송), 그리고 PLIP (패러랠 라인을 통해 네트워크 전송)을 위한 장치 드라이버를 제공한다. 랜 환경에서 가장 잘 동작하는 TCP/IP 구현을 가지고 있는 리눅스는 Net-3와 함 께 상업용 PC 유닉스를 능가하는 동작 가능 시간을 보여주고 있다. 현재 개발하고 있는 취 지는 인터넷 호스트 상에서 안정성있게 리눅스를 실행하는 것을 목표로 두고 활동하고 있 다.

이러한 작업을 더욱 용이하게 해주는 것으로써, 여러 가지 프로젝트가 추진중에 있으며, 리눅스의 융통성을 강화하는 데에 큰 몫을 해줄 것이다. PPP (Point-to-Point Protocol, 시 리얼 라인을 통해서 네트워크 전송을 하는 또 다른 방법)를 위한 드라이버가 현재 베타 단 계에 있으며, ham radio를 위한 AX.25 드라이버는 알파 단계에 와 있다. Alan Cox는 또한 Novell의 IPX 프로토콜을 위한 드라이버를 구현하고 있지만, 완전한 네트워킹을 위해 적합 한 호환성을 가지기 위한 노력으로 인해, Novell의 IPX 프로토콜 개발은 잠시 동안 주춤하 고 있다. 왜냐하면, 필요한 문서를 Novell 측에서 마지못해 제공해 주었기 때문이다. 장래가 유망한 또 다른 사업으로는, 유닉스를 위한 NetBIOS 서버인 samba가 있었으며, Andrew Tridgell에 의해 만들어 지고 있다.- *NetBIOS는 lanmanager와 작업그룹들을 토대로 동작하는 Windows와 같이 어플리케이션상에 있는 프로토콜이다.*

### 1.4.1 Different Streaks of Development

그 동안에, Fred는 Net-2e의 개발 작업을 계속 진행하였으며, 더욱 더 개선된 네트워킹 계 층을 제작했다. 이 글을 쓰고 있는 현 시점에서, Net-2e는 여전히 베타 소프트웨어였다. Net-2e의 가장 주목할 만한 점이라면, DDI,*Device Driver Interface*를 합병한 것이 었다. DDI는 항상 동일한 엑세스와 모든 네트워킹 장치와 프로토콜을 위한 구성법을 제공하였다.

Linux와 FreeBSD를 위한 ISDN을 만들어낸 Matthias Urlichs는 또 다른 TCP/IP 네트워킹 을 구현하였다. 이 작업을 위해 그는 몇몇 BSD 네트워킹 코드를 Linux 커널에 집적시켰다.

그러나 미래를 예견할 수 있었다 하지만 Net-3는 그대로 머물러 있었다. 현재 Alan은 ham radio amateurs를 사용하는 AX.25 프로토콜의 구현 작업을 하고 있다. 의심할 여지 없 이 커널을 위해 "module"이라는 코드를 개발하여 네트워킹 코드에 새로운 활력을 불어 넣 어 주었다. Modules은 여러분이 커널 실행시간에 드라이버를 추가할 수 있게끔 해준다.

네트워크를 구현하는 방법이 다르다 할지라도 모든 사람들은 같은 서비스를 제공하기 위 해 노력했다. 그래도 커널과 장치 레벨 사이에 주요한 차이점은 있었다. 그래서, 여러분들은 Net-2d 또는 Net-3, 그리고 vice versa로부터 유틸리티를 가지는 Net-2e 커널을 동작시키 는 시스템을 구성할 수는 없을 것이다. 이것은 단지 커널 내부를 다루는 명령을 제공해 줄 뿐이며, 오히려 어플리케이션이나 rlogin 또는 telnet과 같은 일반적인 네트워킹 명령에 더욱 더 가깝다.

그렇지만, 이러한 모든 네트워크 버전의 차이점이 여러분을 걱정시킬만큼의 문제거리는 아니다. 여러분이 개발 활동에 참여하지 않더라도, 여러분이 사용하는 TCP/IP 코드에 대해 걱정할 필요는 없는 것이다. 공식 커널 배포는 항상 커널에서 표현하는 네트워킹 코드와 호 환하는 네트워킹 도구집을 수반할 것이다.

### 1.4.2 Where to Get the Code

리눅스 네트워크 코드의 최신 버전은 anonymous FTP를 사용하는 여러 사이트에서 구할 수 있다. Net3를 위한 공식 FTP 사이트는 **sun.site.unc.edu** 사이트의 *system/Network/sunacm*에 미러되어 있는

**sunacm.swan.ac.uk**이다. Net-2e의 최신 패치 키트와 바이너리들은 **ftp.aris.com**에서 찾아볼 수 있다. Matthias Urlichs' BSD-derived 네트워킹 코드는 **ftp.ira.uka.de**의 */pub/system/linux/netbsd*방에서 구할 수 있다.

최신 커널은 **uic.funet.fi**의*/pub/OS/Linux/PEOPLE/Linux*에서 찾아 볼 수 있다.; sunsite와 tsx-11.mit.edu사이트가 이 디렉토리를 미러시켜 놓았다.

## 1.5 Maintaining Your System

이 책을 통해서, 우리는 주로 설치와 구성에 관한 개관을 다룰 것이며, 특히 관리면을 집중 적으로 다룰 것이다. - 서비스를 셋팅한 후에, 여러분은 실행작업 역시 유지시켜 줘야 한다. 그러면 여러분에겐 mail과 news와 같은 서비스도 필요하게 될것이며, 여러분의 시스템을 최신식으로 유지하기 위해 루틴 작업도 해줄 필요가 있게 된다. 다음 장에서 이러한 작업에 관해 자세하게 다루어 보자.

에러 상태나 예상치 못한 일들을 대비하여 어플리케이션 로그 파일과 시스템을 검사하는 일은 시스템을 유지시키기 위한 최소한의 작업이다. 일반적으로, 여러분은 대개, 이러한 작 업을 하기 위해, 한 쌍의 관리 쉘 스크립트를 작성해서, 이것들을 *cron* 항목에 넣어 두고 정 기적으로 실행하고 싶어할 것이다. *smail* 과 C *News*와 같은 몇몇 주요한 어 플리케이션의 소스 배포에 있어서는 그런 스크립트를 포함시키고 있다. 여러분이 필요한 것이 무어인지, 더 좋아하는 것이 무엇인지 파악해서, 스크립트를 작성해야 한다.

cron 작업에서 얻어지는 출력은 관리 계정으로 우송된다. 많은 어플리케이션들은 에러 보고서, 사용량 또는 root 계정으로 요약하는 로그파일을 보낼 것이다. 만약 여러분이 root 계정으로 자주 로그인 한다면, 이것은 대단히 민감해질 것이다. ; 여러분의 개인 계정으로 root의 메일을 전송하기 위해서는 14장에서도 언급하게될 mail alias를 설정하는 것도 좋은 방법이 될 것이다.

하지만 여러분은 여러분의 사이트를 주의깊게 설정해야 한다. Murphy's law는 표면화되 는 몇몇 문제들을 보증해준다. 그러므로, 시스템을 유지시킨다는 것은 그러한 불평거리를 쓸 모 있게 만든다는 의미이다. 대개 사람들은 시스템 관리자가 적어도 root 계정을 사용해서, email을 통해 접근한다고 예상하고 있지만, 관리 측면에서 확실하게 책임을 져야할 사람들 이 접근하기 위해 일반적으로 사용하는 또 다른 주소가 있다. 이를테면, 작동불능 상태의 메 일 구성에 대해 불평하는 것은 대개 **postmaster**로 주소화 되어 있다. ; news 시스템 에 관 한 문제거리들은 **newsmaster** 이나 **usenet**으로 보고가 될지도 모른다. **hostmaster** 로 발송되는 메일은 호스트의 기본 네트워크 서비스와 만약 여러분이 네임 서버를 실행하 고 있다면, DNS 네임 서비스를 담당하고 있는 사람에게 재전송되어야 한다.

### 1.5.1 System Security

네트워크 환경에 있어서 시스템 관리 측면의 또 다른 중요한 작업으로는 침입자로부터 여러 분의 시스템과 사용자를 보호하는 것이다. 부주의하게 시스템을 관리하는 것은 고의적으로 사람들에게 표적을 제공하는 것과 마찬가지이다. ; 패스워드를 추측하는 것에서부터 Ethern et을 기웃거리는 일은 공격 범위를 줄여주는 결과를 초래할 것이며, 날조된 메일 메시지에 서 데이터 손실까지 또는 사용자의 사생활 침해와 같은 문제를 일으키게 된다. 우리는 그것 들이 발생할 수도 있는 배경을 논의하면서, 그러한 특별한 문제에 관해 해결방안을 모색할 것이다.

이 절에서는 시스템 보안을 다루는 기본적인 기술과 그에 따른 예를 들어 보일 것이다. 물론, 이 화제들로 여러분이 직면하게 될 모든 보안 문제들을 다룰수는 없다. ; 단지 일어날 수 있는 문제들을 다룰뿐이다. 그래서, 보안에 관련되어 있는 좋은 책을 읽는 것 또한 중요 하며, 그것이 시스템을 네트워크에 올려놓기 위해선 필수적이다. Simon Garfinkel의 "Practical UNIX Security" ([Spaf93]을 참조하라.) 는 상당히 추천할 만 한 책이다.

시스템 보안은 좋은 시스템을 관리하기위해 시작되었다. 이것은 중요한 모든 파일과 디 렉토리의 소유권과 허가권을 검사하고, 특별하게 사용하는 계정의 상태를 확인하는 작업도 포함하고 있다. 이를테면, COPS 프로그램은 보기드문 허가 또는 다른 예외적인 상황들을 위해, 파일시스템과 일반적인 구성 파일들을 검사할 것이다. 그리고 사용자의 패스워드를 어 떤 특별한 규칙에 따라 추측하기 힘들게 만드는 것도 현명한 방법이다. 이를테면, 쉐도우 패 스워드는 적어도 다섯 개의 문자를 가지는 패스워드를 필요로 한다. 그 패스워드에는 대소 문자와 번호를 포함하고 있다.

# 2 Issues of TCP/IP Networking

이 장에서는 여러분의 리눅스 머신을 TCP/IP 네트워크로 연결할 때, 부딪치게 될 세부사 항들과 IP 어드레스, 호스트 네임, 라우팅의 유래에 관해 알아본다. 그리고, 필요한 설정작업을 이 해하기 위해서 알아야 되는 기본적인 개념들과, 이러한 설정작업에 필요한 도구들을 다루어 보기로 하자.

## 2.1 Networking Interfaces

네트워킹 환경에서 사용되는 설비의 다양성을 감추기 위해서, TCP/IP는 하드웨어를 제어하 기 위 한 하나의 추상적인 *interface*를 정의해 두고 있다. 이 인터페이스는 한 쌍의 연산자를 제공한다. 그리고, 그것은 모든 종류의 하드웨어를 같은 형태로 두고, 패킷을 보내고 받는 작업을 한 다.

네트워크에 사용되는 각 주변장치들은 그에 해당하는 인터페이스가 커널에 표시되어 있어 야 한다. 예를 들어, 리눅스에서 사용하는 Ethernet 인터페이스는 *eth0* 그리고, *eth1* 로 표시되어 있고, SLIP 인터페이스는 *sl0, sl1* 등등으로 표시되어 있다. 이들 인터페이스의 이름은 여러분이 커널에 특별한 물리적인 장치의 이름을 매기고 싶을 때, 구성 목적으로 사용한다. 그것들이 꼭 특별한 의미를 가지고 있는 것은 아니다.

TCP/IP 네트워킹을 사용 가능하게 하기 위해서, 하나의 IP 어드레스에 하나의 인터페이스 를 할당해야 한다. IP 어드레스는 전세계에서 통신을 할 경우, 자신의 신분을 밝혀주는 유일한 수단 이 된다. 이 어드레스는 위에서 언급한 인터페이스의 이름과는 다르다. ; 만약 여러분이 인 터페이 스를 문에 비유한다면, 어드레스는 그 문에 붙어 있는 문패와 같다.

여러분이 설정해야 하는 또 다른 장치 인수들이 있다. 이것들중 하나로써 데이터 그램의 최 대 크기를 설정하는 부분이 있다. 이것으로 하드웨어의 특별한 부분들을 처리할 수 있다. 이것을 MTU 또는 *Maximum Transfer Unit*라고 부른다. 다른 속성들은 다음에 소개하기로 하자.

## 2.2 IP Addresses

1 장에서 언급한대로, IP 네트워킹 프로토콜이 이해할 수 있는 어드레스수는 32비트이다. 네트워 킹 환경에 있는 모든 기계들은 이 수의 범위내에서 할당할 수 있다. 만약 여러분이 다른 네트워 크와의 TCP/IP 교환이 이루어지지 않는 일반적인 지역 네트워크를 운영하고 있다면, 여 러분의 개인 취향에 따라 이들 번호들을 할당할 수 있을 것이다. 그러나, 인터넷에 있는 모든 사이 트들은 중앙기관 즉, NIC - *Network Information Center - 대개, 프로바이더들이 여러분에게 IP address를 할당하며, 여러분은 그것을 사야한다. 또는 여러분들이 원하는 IP address를 직접 NIC에 연락해서 구할 수도 있다. 연락 주소는 다음과 같다.* `hostmaster@internic.net` *에 의해 그 번호들을 할당 받을 것이다.*

IP address를 쉽게 읽기 위해서, octet라고 부르는 네 개의 8비트 수로 나누어 놓았다. 예를 들 어, **0x954C0C04**의 IP address를 가지는 *quark.physics.groucho.edu*는 실제로 *149.76.12.4*로 쓰여져 있다. 이러한 형태를 *dotted quad notation*이라 부른다.

이 표기법을 쓰는 또 다른 이유로써, IP address는 맨 앞쪽 옥텟을 network 숫자로, 나머지 부 분을 host 숫자로 구분해 놓고 있다. 여러분이 NIC에게 IP address를 요청할 때, 여러분이 계획한 대로 할당해 주진 않는다. 대신에, 여러분이 하나의 네트워크 숫자를 받았다면, 그 네트워크 범위 내에서 여러분의 선호도에 따라, 모든 유효한 IP address를 할당할 수는 있다.

호스트 부분은 네트워크 규모에 의존하기 때문에 더욱더 작아지거나, 크게될 필요가 있다. 그 러한 여러 가지 필요성을 충족시켜주기 위해 네트워크에도 여러 클래스가 있으며, 이것은 또 다 른 관점에서 IP address를 분할해 놓고 있다.

**Class A**

Class A는 **1.0.0.0**에서 **127.0.0.0**까지의 네트워크를 포함하고 있다. 이 네트워크 숫자는 첫 번째 옥텟에 포함되어 있다. 그리고 이것은 24 비트 호스트 부분 즉, 대략 160만 개의 호스트를 허용할 수 있다.

**Class B**

Class B는 **128.0.0.0**에서 **191.255.0.0**까지의 네트워크를 포함하고 있 다. ; 네트워크 숫자는 첫 두 옥텟에 포함되어 있다. 그리고 이것은 16320개의 네트워크를 허용하고 있으며, 각 65024개의 호스트를 가지고 있다.

**Class C**

Class B는 **192.0.0.0**에서 **223.255.255.0**까지의 네트워크를 포함하고 있다. 네트워크 숫자는 첫 세 옥텟에 포함되어 있다. 그리고 이것은 거의 2백만개의 네트워크를 허용하고 있으며, 최고 254개의 호스트를 가질 수 있다.

**Class D, E, and F**

**224.0.0.0**에서 **254.0.0.0**의 범위내에 있는 주소들은 실험용이거나 미래를 위해 예약되어 있기 때문에, 어떤 네트워크도 명시하지 않는다.

1장에서 보인 것을 예로 든다면,**quark**의 주소인 **149.76.12.4**는 Class B에 해당하 는 네트워크 **149.76.0.0**는 호스트 **12.4**를 가진다고 말할 수 있다.

위애서 보인 글에서 여러분은 호스트 부분에 있는 각 옥텟이 가능한 모든 값들을 허용하지 않 는다는 사실을 어쩌면 알아차렸을 지도 모른다. 왜냐하면, 모든 0과, 모든 255를 가지는 호스트 숫자들은 특별한 목적을 위해 이미 예약되어 있기 때문이다. 모든 호스트 부분에 있는 주소 비트 들이 0인것은 네트워크를 나타내고, 그 부분이 1인 것은 브로드캐스트 주소라고 부르고, 이것은 네트워크에 명시되어 있는 모든 호스트를 나타낸다. 그래서, **149.76.255.255**는 사용할 수 있는 호스트 주소가 아니라, 네트워크 **149.76.0.0**에 있는 모든 호스트를 나타낸다.

특별히 예약되어 있는 두 개의 네트워크 주소 즉, **0.0.0.0**과 **127.0.0.0**가 있다. 첫 번째 주소는 다른 말로 *default route*라고 부르고, 그 다음 것은 *loopback address*라고 부른다. 디폴트 라우트는 IP의 경로 배정 방법에 관한 정보를 포함하고 있으며, 그 내용은 다음에 설명할 것이다.

Network 127.0.0.0 is reserved for IP traffic local to your host.번역을 못한 부분 대개, 어드레스 127.0.0.1은 여러분의 호스트에 loopback interface라고 부르는 특별한 인터페 이스로 할당될 것이며, 그것은 마치 폐쇄회로와 같이 작동한다. TCP 또는 UDP에서 건너온 IP 패킷들은 마치 어떤 네트워크로 도착되고 있는 것과 같이 루프백 인터페이스로 되돌려질 것이다. 이러한 방법으로 여러분이 실제 네트워크를 사용하지 않고도 네트워킹 소프트웨어 를 개발하고 시험할 수 있다. 여러분이 독립형 호스트상에서 네트워킹 소프트웨어를 사용하고 자 할 때, 유용하게 쓰일 수 있는 또 다른 어플리케이션이 있다. 이것이 꼭 특별한 것만은 아니다. 이를테면, 많은 UUCP 사이트들이 IP와의 연결을 가지는 것은 아니지만, 그럼에도 불구하고, 여전히 INN 뉴스 시스템을 실행하고 싶어한다. 리눅스에서 적절한 운영을 할려면, INN은 루프백 인터페이스를 필요로 한다.

## 2.3 Address Resolution

이제까지 여러분은 IP address가 어떻게 만들어지는지 보아왔다. 여러분은 그것들이 각각 다른 호스트에 있는 Ethernet상에서 어떻게 사용되는지 궁금할지도 모른다. 결국, Ethernet 프로 토콜은 여섯 개의 옥텟숫자로 호스트를 증명하는데, 그것은 일반적인 하나의 IP address를 가지는 것은 아니다. 그렇지 않은가?

그렇다. 그것은 Ethernet address위에 IP address를 대응시키기 위한 메카니즘이 필요한 이 유 이다. 이것을 다른말로, *Address Resolution Protocol* 또는 ARP라고 부른다. ARP는 Ethernet를 전혀 제한하지는 않지만, ham radio와 같은 또 다른 형태의 네트워크에서도 사용된다. ARP 에 기 초를 두고 있는 생각으로서, 150여명의 군중속에서 Mr. X. Ample를 찾아야 할 때, 대부분 의 사람 들은 어떻게 할까? ; 주위를 둘러 보면서 그의 이름을 부르면, 그가 대답할 것이다.

ARP가 주어진 IP address와 일치하는 Ethernet address를 찾고자 할 때, Ethernet의 특징중 의 하나인 "브로드캐스팅"을 사용한다. 그것은 네트워크에 있는 모든 지역에 자료를 동시에 보내는 형태이다. ARP가 보내는 브로드캐스트 자료는 IP address를 위한 하나의 질의를 포함하고 있다. 그 자료를 받는 각 호스트는 그 자체의

IP address와 그것을 비교해서, 만약 그것이 일치 한다면, 조회중인 호스트는 그 대답을 ARP로 보낸다. 그 조회중인 호스트는 대답을 보낼 송신자의 Ether net address를 알아낼 수 있다.

물론 여러분은 전세계에 퍼져 있는 무수히 많는 Ethernet을 그 호스트가 어떻게 찾을지, 또 왜 꼭 Ethernet이어야 하는지 궁금할 것이다. 이러한 질문속에는 라우팅이라는 것이 무엇인지 도 포함 하게 된다. 즉, 라우팅은 네트워크에 있는 호스트의 물리적인 위치를 알아내는 것이다. 이것 에 대 해서는 다음 절에서 자세하게 다룰 것이다.

잠깐동안, ARP에 관한 이야기는 접어두기로 하자. 한때, 호스트가 Ethernet address를 발견 해 서, 그것을 ARP 캐쉬에 저장했다. 그래서, 다음번에 자료를 호스트로 보내고자 할 경우, 그것을 위한 질의는 가지고 있지 않았다. 아무리 그러하더라도, 이 정보를 영원히 보존하고자 하는 생각 은 현명하지 못한 것이다. 이를테면, 기술적인 문제로 인해 리모트 호스트의 Ethernet 카드 를 대 신할 수도 있다. 그래서, ARP는 그다지 쓸모가 없게 되었다. IP address를 위한 또 다른 질의를 추출해내기 위해서, ARP 캐쉬에 있는 개체들을 언젠가는 버리게 된다.

때때로, 주어진 Ethernet address와 관련되어 있는 IP address를 발견하는 것도 필요하다. 이 것은 디스트없는 기계가 네트워크에 있는 서버로부터 부트하고자 할 경우에 발생한다. 랜 에서는 이러한 현상이 결코 드물지만은 않다. 그러나 디스트없는 클라이언트는 가상적으로 그 자체 에 관 한 어떠한 정보도 가지고 있지 않다. - Ethernet address를 제외하고! `So what it basically does is broadcast a message containing a plea for boot servers to tell it its IP address.` 이것을 위한 또 다른 프로토콜 즉, *Reverse Address Resolution Protocol* 또는 RARP가 있다. BOOTP 프로토콜과 함께, 이것은 네트워크를 통해 디스크없는 클라이 언트를 부트스트랩핑하기 위해 정의해 놓은 절차를 제공한다.

## 2.4 IP Routing

### 2.4.1 IP Networks

여러분이 누군가에게 편지를 보낼 때, 대개 여러분은 우편봉투에 국가, 시(군), 우편번호 등 등, 완 벽한 주소를 기입할 것이다. 여러분이 그것을 우편함에 넣으면, 우편업무를 하는 우체부 아 저씨가 그것을 목적 주소로 가져갈 것이다; 그것은 핀지봉투에 명시되어 있는 국가 또는 시(군)으 로 보내 질 것이다. 그러면, 그곳에 있는 우체국에서는 그 편지를 목적지로 보낼 것이다. 계층적 구성은 오히려 분명하다; 여러분이 편지나 소포를 어디에서 부치던간에, 그 지역 우체국장은 그 편지(소 포)가 가야할 곳을 대략 알 것이다. 그러나, 그 편지가 목적주소로 어떻게 가는지는 알 필요 가 없 을 것이다.

IP 네트워크도 이와 유사한 형태로 되어있다. 전체 인터넷은 *automonous systems*라 고 불리우 는 몇 개의 네트워크로 이루어져 있다. 각 시스템은 내부적으로 각 구성 호스트사이에서 라우팅 을 수행한다. 그래서, 목적 호스트의 네트워크으로 가는 경로를 발견함으로써, 데이터그램을 운반 하는 작업의 양을 줄일 수 있다. 이것은 데이터 그램이 특별한 네트워크에 있는 어떤 호 스트로 옮겨지자 마자, 오로지 네트워크 그 자체에 의해서, 그것을 처리한다는 의미를 담고 있다.

### 2.4.2 Subnetworks

위에서 설명한 것과 같이, IP address를 호스트 부분과 네트워크 부분으로 나눔으로써, 이 구조를 나타낼 수 있다. 목적 네트워크는 IP address의 네트워크 부분에서 유래한 것이다. 그래서, 동일 한 IP 네트워크 번호를 가진 호스트들은 같은 네트워크에서 발견된다. - *Autonomous 시스템들 이 조금더 일반적이다. 그것들은 여러개의 IP 네트워크를 포함할 지도 모른다.*

그것이 수백개의 더욱더 작은 네트워크 집합과 Ethernet와 같은 물리적인 네트워크로 이루 어 진 가장 작은단위들로 이루어진 후로는, 네트워크에서 *inside*라고 하는 유사한 스키마 를 제공하는 것도 이치에 맞는 말이다. 그러므로, IP는 하나의 IP 네트워크로 세분화되고, 그것이 여러개의 *subnet*으로 나누어진다.

IP 네트워크 부분에서 특정 IP address 범위로 데이터 그램을 배달하는 일을 하나의 IP 서 브 넷이 맡고 있다. 클래스 A, B, 또는 C와 같이 그것도 IP address의 네트워크 부분으로 확 인되었 다. 그러나 요즘에는 호스트 부분에 몇 비트를 포함시킴으로써, 네트워크 부분을 확장시킨 다. 서 브넷 번호로 해석되는 비트들의 번호는

*subnet mask* 또는 *netmask*에 의해 주 어진다. 이것은 32 비트로 이루어진 숫자들이며, IP address의 네트워크 부분을 위한 비트 마스크를 표시한다.

Figure 2.1: Subnetting a class B network

그러한 네트워크의 한 예로써, Groucho Marx University의 네트워크를 들 수 있다. 그것은 클 래스 B에 해당하는 네트워크 번호 **149.76.0.0**을 가지며, 그것의 넷 마스트는 **255.255.0.0**이 된다.

내부적으로 GMU 대학의 네트워크는 여러개의 작은 네트워크로 이루어져 있다. 그래서, IP 주 소의 범위가 254개의 서브넷 즉, **149.76.1.0**에서 **149.76.254.0**으로 분해되었다. 예를 들어, Theoretical Physics 부는 **149.76.12.0**으로 할당되었다. 그리고 campus backbone은 그자체의 네트워크를 가지며, **149.76.1.0**을 할당 받았다. 이러한 서브넷들은 같은 IP 네트워크 번호를 공유하고 있다. 반면에 세 번째 옥텟은 그것들 사이에서 구분되어 사용된다. 그리하여 그것들은 **255.255.255.0**이라는 하나의 서브넷 마스크를 사용할 것이다.

그림 2.1은 quark의 주소인 **149.76.12.4**가 어떤 식으로 해석되는지를 보여준다. 그 주소가 어떻게 클래스 B 네트워크에 속하게 되는지 또, 어떻게 서브네팅을 사용하는지도 보여준다.

서브네팅 (기술적인 용어로 서브넷을 이렇게도 부른다.)이 오직 네트워크에서 internal division 으로서만 가치 있는 것은 아니다. 보통 네트워크 관리자가 이 서브넷을 관리하게 되는데, 대 개 현 존하는 경계를 나타내기 위해 서브넷을 만든다. 그것들은 물리적 (두개의 Ethernet 사이에 서)이고, 관리적 (두 department사이에서) 이며, 또한 지리적이며, 이러한 서브넷들을 능가하는 권한 이 몇 몇 사람들에게 주어진다. 하지만 이 구조는 오직 네트워크의 내부적인 활동에 영향을 미칠 수 있 으며, 바깥 세계에서는 그 모습이 나타나지 않는다.

### 2.4.3 Gateways

서브넷팅을 함으로써, 관리상의 이점만을 얻는 것은 아니다. 그것은 자주 하드웨어 한계의 중요성 을 우리에게 인식시켜 주기도 한다. Ethernet와 같이 물리적인 네트워크에 있는 호스트 관 점에서 본다면, 매우 제한되어 있다. 그 제한사항이라는 것은 직접적으로 통신할 수 있는 호스트는 오직 해당 네트워크상에 있어야 한다는 점이다. 다른 모든 호스트들은 gateways라는 것을 통해 서 연결 될 수 있다. 게이트웨이는 두 개이상의 물리적인 네트워크에 동시에 연결되어 있는 하나의 호스 트이다. 그리고 그것은 그것들 사이에서 패킷을 교환하는 작업을 구성해 준다.

만약 호스트가 논리적인 물리 네트워크에 있다면, IP를 쉽게 인식시키기 위해서, 다른 물리 적 네트워크는 또 다른 IP 네트워크에 속해 있어야 한다. 예를 들어, 네트워크 번호 **149.76.4.0**이 mathematics LAN에 있는 호스트로 예약되어 있는 경우, 그 데이터 그램 을 **quark**로 보내고자 할 때, **erdos**상에 있는 네트워크 소프트웨어는 즉시 IP address, **149.76.12.4**를 나타내어 준다. 그리고, 그 자료는 게이트웨이 (초기값으로는 **sophus**로 되어 있다.)를 거쳐서, 목적 호스트에 도착할 것이다.

**sophus** 그 자체는 두 개의 전혀 다른 서브넷에 연결되어 있다. : Mathematics Department, 그리고 campus backbone. 그것은 *eth0*와 *fddi0*라고 하는 각각 다른 인터페이스를 거쳐서 접근한다. 지금 현재, 우리가 할당할 IP address는 무엇일 까? 그리고 서브넷 **149.76.1.0** 또는 **149.76.1.4** 중에 어디에 그것을 할당해 주어야 할까?

답은 둘다이다. Maths LAN에 있는 호스트와 통신을 하고자 할 때, sophus는 IP address **149.76.4.1**를 사용해야 하고, 백본에 있는 호스트와 통신을 하고자 할 경우에는 **149.76.1.4**를 사용해야 한다.

그리하여, 게이트웨이는 네트워크당 하나의 IP address를 할당받는다. 이러한 address들은 각 해당하는 인터페이스와 일치되어 있으며, 게이트웨이를 거쳐서, 서브넷에 연결된다. 다음 표에서 는 sophus에서 일치하는 인터페이스와 어드레스를 보여주고 있다.

마지막에 보이는 개체는 루프백 인터페이스인 *lo*이다. 이것은 2.2절에서 소개가 되었 다.

그림 2.2는 Groucho Marx University (GMU)에 있는 네트워크 토폴로지의 단면을 보여주고 있다. 두 개의 서브넷에 있는 호스트들은 양쪽으로 물려있는 address를 보여주고 있다.

Figure 2.2: A part of the net topology at Groucho Marx Univ.

일반적으로, 여러분은 호스트나 인터페이스에 어드레스를 추가시키는 방법의 차이점에 대해 서 는 무시해 버려도 상관없다. **erdos**와 같이 하나의 네트워크에 있는 호스트들을 위해 서, i 엄밀히 말해 여러분은 이곳저곳의 IP address를 가지고 있는 호스트를 조회해 볼 것이다. 하지만, 여러분이 게이트웨이를 참조할 때, 이 차이점이 매우 중요한 작용을 할 수도 있다.

### 2.4.4 The Routing Table

여기서는 데이터 그램을 리모트 네트워크로 넘겨줄 때, 어떻게 IP가 사용할 게이트웨이를 선택하 는지에 초점을 맞출 것이다.

**quark**로 데이터 그램을 보내줄 때, **erdos**는 목적 주소를 검사하고, 지역 네트워크 에 그것이 실제로 존재하는지를 확인하였다. 이 작업과 **erdos**가 디폴트 게이트웨이인 **sophus**로 자료를 보내는 작업은 같은 맥락이라고 볼 수 있다. **sophus**는 **quark**가 어떤 네트워크와도 직접적으로 연결되어 있지 않다는 것을 인식한다. 그래서, **sophus** 는 다음에 거치게 될 다른 게이트웨이를 찾아내게 될 것이다. 정확하게 선택했다면, 그것은 Physics Department로 가는 게이트웨이인 **niels**일 것이다. **sophus**는 적합한 게이트웨이를 가진 목적 네트워크와 교신하기 위한 몇몇 정보를 필요로 하게 된다.

이것을 사용하는 라우팅 정보 IP는 기본적으로 게이트웨이에 연결되어 있는 네트워크 테이 블 을 의미한다. 일반적으로 다목적용 개체를 제공해야 하며, 이것은 네트워크 **0.0.0.0**과 관련되어 있는 게이트웨이이다. 알려지지 않은 네트워크로 모든 패킷들은 디폴트 라우트를 거쳐서 보내지게 된다. **sophus**상에서, 이 테이블은 다음과 같이 보일 것이다.

sophus가 직접적으로 연결되어 있는 네트워크에서의 라우트는 게이트웨이를 필요로 하지 않 는다. 이러한 경우의 게이트웨이 개체는 "-"로 표시되어 있다.

라우팅 테이블은 여러 가지 의미로 해석할 수 있다. 규모가 작은 LAN을 위해서는 부트시간 때 에 수동으로 route 명령어를 입력해서 그것들을 IP로 피드백하고, 구성하는 것이 가장 효과 적이 다. (5장을 참조하라). 이것보다 조금 더 큰 네트워크를 위해서는 실행시간에 routing daemons를 조절해 주어야 한다. 이것들은 네트워크의 중앙 호스트에서 실행되며, 네트워크 사이에서 최적의 라우트를 설정해 주기 위해서 라우팅 정보를 교환할 것이다.

네트워크의 규모에 의존하는 또 다른 라우팅 프로토콜을 사용할 것이다. Groucho Marx campus와 같은 자발적인 시스템에서 라우팅을 하기 위해서는 internal routing protocols을 사용 한다. 가장 두드러지게 사용하는 것중 하나가 바로 RIP, *Routing Information Protocol* 이며, 그것은 BSD routed 데몬에 의해 실행된다. 자발적인 시스템에서 라우팅을 하기 위해서는 EGP (*Ext ernal Gateway Protocol*) 또는 BGP (*Border Gateway Protocol*) 과 같은 *external routing protocols*를 사용해야 한다. RIP 뿐만 아니라 이러한 것들도 Cornell's 대학의 gated 데몬에 의해 실행되고 있다. - *많은 사람들이 routed가 불안정하다고 생 각한다. gated가 RIP를 지원하는 이후로는 routed대신에 gated를 사용하는 편이 더 낫다.*

### 2.4.5 Metric Values

RIP를 기본으로 하고 있는 동적 라우팅은 어떤 목적 호스트나 "*hops*" 번호에 기초를 두고 있는 네트워크를 위해 최고의 라우트를 선택한다. 그리고 데이터 그램은 도착하기 전에 게이트 웨이를 거쳐야 한다. 단거리 라우트는 RIP보다 전송률이 더 좋다. 16이상의 홉(라우팅 경로에서 차 지하는 하나의 포지션)을 거치는 장거리 라우트는 쓸모 없는 것으로 간주되며, 폐기 처리된다. 다시 말해 서 접속이 안된다는 의미이다.

여러분의 지역 네트워크에 있는 라우팅 정보를 관리하고, RIP를 사용하기 위해서는 모든 호 스 트에 gated를 실행시켜야 한다. 부트시간에 gated는 네트워크 인터페이스에서 일어나는 모 든 활 들을 검사한다. 활동하고 있는 인터페이스가 하나 이상이라면 (여기서 루프백 인터페이스는 계산 하지 않는다.) 호스트가 여러 네트워크 사이에서 패킷들과 라우팅 정보를 활발히 교환하고 제공한 다고 말할 수 있다. 그렇지 않다면, 즉 다시말해 활동하고 있는 인터페이스가 없다면, RIP에 관한 최신 정보를 받거나 지역 라우팅 테이블을 갱신하는 작업이 소극적으로 이루어지고 있다고 말할 수 있다.

지역 라우팅 테이블로부터 정보를 제공할 때, gated는 라우팅 테이블 엔트리와 관련되어 있 는 metric value 로 라우트 길이를 계산한다. 라우트를 구성할 때, 시스템 관리자가 이 미터값 을 계 산하며, 이 라우트를 사용하는 실제 비용을 곰곰히 생각해 보아야 한다. 그러므로 호스트와 직접 적으로 연결되어 있는 서브넷의 미터값은 항상 0이 되어야 한다. 반면에, 두 개의 게이트웨 이를 거치는 하나의 라우트는 미터값이 두자리가 되어야 한다. 하지만, 여러분이 RIP나 gated를 사용 하지 않을 때는 미터값에 대해서 걱정하지 않아도 된다.

## 2.5 The Internet Control Message Protocol

IP는 우리가 아직 언급하지 못한 companion protocol을 가지고 있다. 이것은 다름아닌 *Internet Control Message Protocol* (ICMP) 이며, 다른 호스트와의 메시지 교류시 발생하는 에러를 교환하기 위해 커널 네트워킹 코드를 사용한다. 이를테면, 여러분이 현재 **erdos**상에 있고, **quark**에 있는 12345 포트로 원격 접속하고자 하며, 그 포트에서는 어떤 프로세스 *listening*도 하지 않고 있다고 가정한다. 이 포트를 위한 첫 번째 TCP 패킷이 **quark**에 도착할 때, TCP의 네트워킹층은 도착한 패킷을 인식할 것이고, 즉시 "Port Unreachable" 상태의 ICMP 메시지를 **erdos**로 되돌려 줄 것이다.

이해할 수 있는 ICMP 메시지는 수 없이 많으며, 그 중에는 에러 상태를 취급하는 메시지도 있다. 그 중에 *Redirect message*라 불리우는 매우 흥미로운 메시지가 하나 있다. 비록 더욱더 짧은 경로가 있다하더라도, 그것은 라우팅 모듈에 의해 운영되며, 다른 호스트가 게이트웨이를 통해서 그것을 사용할 때 감지된다. 예를 들어, 부팅한 후에 **sophus**의 라우팅 테이블이 불완전한 상태가 될 수도 있고, Mathematics 네트워크와 FDDI 백본에 경로가 포함되어 있을 수도 있으며, *Groucho Computing Center's gateway* (gccl)에 있는 라우트 포인팅이 초기값으로 설정되어 있을 수도 있다. 그래서 **quark**에 있는 패킷들이 Physics Department에 물려있는 게이트웨이인 **niels**보다 오히려 gccl로 보내질 것이다. 형편없는 경로 배정으로 어떤 데이터 그램을 전송받을 때, gccl은 그 패킷들을 **niels**로 다시 전송할것이고, 동시에 최상의 경로 배정을 지시하는 ICMP Redirect 메시지를 **sophus**로 전송할 것이다.

지금하게될 내용이 가장 기본적인 설정작업을 수동으로 해야하는 번거로움을 피하기 위한 좋 은 방법처럼 보일수도 있지만 RIP나 ICMP Redirect messages가 동적 라우팅 구성에 의 존하고 있다하더라도 이것이 항상 좋은 생각만은 아니다. ICMP Redirect 와 RIP는 몇몇 라우팅 정보가 실제로 믿을 만한 것인지를 검증하기 위한 어떤 선택사항도 제공해 주지 않는다. 이것이 혹 여러 분의 전체 네트워크 트래픽을 분열시키기 위해 고의로 쓸데 없는 작업을 허용하고 있는지도 모른 다. 이러한 이유 때문에, 그것들이 마치 호스트의 경로를 재 발송하는 것처럼, 네트워크 라 우트에 영향을 미치는 Redirect messages들을 치료하기 위한 몇몇 Linux 네트워킹 코드가 있다.

## 2.6 The Domain Name System

### 2.6.1 Hostname Resolution

위에서 기술한 대로, TCP/IP 네트워킹에서 어드레싱은 32비트 숫자들로 운영된다. 하지만, 여러 분들은 이 숫자들을 기억하는데 많은 어려움을 느낄 것이다. 그래서, 호스트는 일반적으로 gauss 또는 strange와 같은 정규 이름을 가지고 있다. 이 이름과 일치하는 IP 어드레스를 찾는 것 이 어 플리케이션의 의무이다. 이러한 과정을 *host name resolution*이라고 부른다.

주어진 호스트명의 IP 어드레스를 찾아야 하는 어플리케이션은 호스트와 IP 어드레스를 찾 기 위해 자체적으로 어떤 체계를 가지고 있진 않다. *Instead, it relies on number of library functions that do this transparently, called gethostbyname(3) and gethostbyaddr(3).* 전통적으로, 이러한 것들과 그 절차에 연관되어 있는 숫자는 *resolver library*라고 하는 여러개의 라이브러리로 그룹화되어 있다; 리눅스 상에서 이러한 것들은 표준 libc에 한 부분이다. 일상적으로, 기능들의 모음들을 "*the resolver*"라고 부른다.

현재 Ethernet과 같은 조그마한 네트워크에서나 심지어 그것들의 클러스터에서도 호스트명 을 어드레스에 대입시키는 테이블을 유지하기란 정말 힘든 작업이다. 이러한 정보들은 대개 파 일명 이 */etc/hosts*라고 하는 곳에서 유지되고 있다. 호스트를 추가하거나 삭제할 때, 또는 어드레스들을 반환할 때, 여러분은 모든 호스트에

있는 hosts파일을 갱신해 주어야 한 다. 분명히 이것은 몇대의 컴퓨터로 네트워크를 구성하는 것보다 더 귀찮은 작업일지도 모른다.

Sun Microsystems가 개발한 NIS, *Network Information System*에서 이러한 문제를 해결하기 위한 하나의 방편으로 YP 즉, 옐로우 페이지라는 것을 내 놓았다. NIS는 마스터 호스트에 있는 데이터 베이스에 hosts 파일과 또 다른 정보들을 저장해 놓는다. 그러면 클라이언트는 필요 한 정 보를 데이터 베이스에서 검색할 수 있다. 이러한 방법은 아직 LAN과 같은 중급 네트워크에 적합 한 방법이다. 왜냐하면, 전체 hosts 데이터 베이스를 유지하고, 그것을 모든 서버에 분배해 주어야 하기 때문이다.

인터넷 상에서, 어드레스 정보는 기본적으로 *HOSTS.TXT*라고 하는 데이터베이스에 저장되어 있다. 이 파일은 *Network Information Center* 또는 NIC에 의해 유지되고 있으며, 이 것은 모든 참 여 사이트에 전송되고 설치되어야 한다. 네트워크가 계속해서 성장할 때, 이러한 구성에는 몇가지 문제점들이 발생한다. 게다가 관리상의 문제점으로써, 정규적으로 HOSTS.TXT파일을 설치 해야 하고, 그 파일을 서버에 정기적으로 분배해야 하는 문제점도 포함하고 있다. 심지어 NIC에 등록 되어야 하는 모든 이름에 심각한 문제점들이 발생할 수도 있으며, 이름을 가지고 있지 않은 것이 밖으로 유출되는지를 확인해 보기도 해야 한다.

1984년, 이러한 이유로써, 새로운 이름 해결 방법 즉, *Domain Name System*이라는 것이 채택 되었다. DNS는 Paul Mockapetris가 개발하였고, 그와 동시에 주소와 관련된 모든 문제들을 해결 했다.

### 2.6.2 Enter DNS

DNS는 도메인과 호스트명을 계층적으로 구성하고 있다. 도메인은 어떤 의미와 연관되어 있는 사이트들의 집합이다. – 도메인이 적절한 네트워크 형태 (예를 들어 대학에 있는 모든 기 계들, 또는 BITNET에 있는 모든 호스트들)로 되어 있기도 하고, 특정 기구 (미국 정부) 또는 지 리적으 로 묶여 있기도 하다. 이를 테면, 대학들은 **edu** 도메인으로 그룹화되어 있고, 각 종합대학과 단과대학은 다시 그것들의 호스트를 포함하고 있는 여러개의 subdomain을 사용한다. Groucho Marx University는 groucho.edu 도메인을 부여받았을 것이고, Mathematics Department의 LAN은 maths.groucho.edu를 할당받았을 것이다. 부문 네트워크에 있는 호스트들은 그 자체의 호스트명을 도메인명으로 사용할 것이다; 그래서 **erdos**가 erdos.maths.groucho.edu로 알려져 있는것인 지도 모른다. 이것을 *fully qualified domain name* 또는 FQDN이라 부르며, 이것으로 인해 특정 호스트가 전세계에서 유일 무이하게 입증될 수 있다.

`Figure 2.3: A part of the domain name space`

그림 2.3은 도메인 네임 영역을 보여주고 있다. 이 트리에서 루트에 있는 개체는 하나의 점-도트- (이것을 root domain이라 부른다.) 으로 표시한다. 그리고 다른 모든 도메인을 포 함 하고 있다. 호스트명을 어떤 함축적인 의미를 가진 지역 도메인명을 사용하기 보다 오히려 fully qualified domain name으로 표시하기 위해, 때때로 그것은 trailing dot로 쓰여진다. 이것은 그 이름의 마지막 요소가 루트 도메인이라는 것을 의미한다.

이름 개체에 의존하고 있는 하나의 도메인은 top-level, second-level, 또는 third-level이 라고 부르기도 한다. 그리고 많은 레벨들이 세분화되고 있지만, 그렇게 많은 것은 아니다. 다음에 여러 분이 자주 볼수 있는 top-level에 관해 설명해 놓았다.

**edu**

(대개 미국에서 사용함) 교육기관, 예 : 대학

**com**

영리단체 예 : 회사(company)

**org**

비 영리단체. 개인 UUCP 네트워크도 종종 이 도메인을 사용한다.

**net**

게이트웨이와 네트워크에서 관리를 목적으로 하는 호스트

**mil**

미국 국방성 기구

**gov**

미국 정부 기관

**uucp**

이전에 도메인없이 UUCP 이름만을 사용하던 모든 사이트 명이 공식적으로 이 도 메인을 사용하게 되었다.

인터넷에서는 법적으로 네 개의 도메인 (**edu, net, mil, gov**)을 미국에서만 사용할 수 있게 하고 있으나 미국에 속해있지 않은 나라에서도 이들 도메인을 사용하는 경우가 있다. 그 중 특수하게, **net** 도메인을 들수가 있지만, **mill**과 **gov**는 오로지 미국에서만 사용할 수 있다.

미국 이외의 나라에서는 일반적으로 ISO-3166에 정의되어 있는 두 개의 문자로 각 나라의 top-level 도메인을 나타낸다. 이를테면, 필란드는 **fi** 도메인을 사용하고, 프랑스는 **fr**을, 독일은 **de**, 그리고 호주는 **au**를 top-level 도메인으로 사용한다. top-level 도메인 다음에오는 호스트 명은 각나라의 NIC에서 자유롭게 구성할 수 있다. 예를 들어, 호주에서 second-level 도메인을 국제적으로 사용하는 top-level 도메인과 유사하게 즉, **com.au** 또는 **edu.au**처럼 사용할 수 있다. 독일과 같은 나라에서는 특별한 도메인을 써서 특정 기구를 직접적으로 언급하기 위해 약간은 긴 이 름을 사용하기도 한다. 예를 들어, *ftp.information.unierlangen.de* `<ftp://ftp.information.unierlangen.de>` 와 같은 호스트명을 사용하는 것이 보기 드문 것만은 아니다. 독일과 같은 능력있는 나라에서는 보통 사용하는 호스트명과 완전히 다른 것을 사용하기도 한다.

물론, 이러한 국제적인 도메인이 아래에서 설명하게될 호스트를 의미하는 것은 아니다. 그 도 메인은 실제로 그 나라에 위치하고 있다; 오직 그 나라의 호스트는 그 나라의 NIC에서 등 록시키 고 있다. 스웨덴의 회사가 호주에 지사를 둘 경우, 그 지사에 있는 모든 호스트들은 그들의 top-level 도메인을 se 로 등록시킨다.

현재, 네임 영역에 있는 도메임 네임을 계층적으로 구성하게 되면, 그 이름들이 중복되는 문 제를 말끔히 해결할 수 있다. ; DNS와 호스트의 이름은 전세계에서 오직 하나이어야 한다. 게다 가, fully qualified name들은 기억하기 쉬워야 한다. 그리고 이미 거대한 하나의 도메인을 여 러 서브 도메인으로 나누기 위한 좋은 방법들이 나와있다.

그리고 DNS는 심지어 관리자를 거쳐서 해야하는 작업 즉, 서브도메인을 만들 수 있는 권한 을 여러분에게 위임해주는 것보다 더 한 것을 허가해 주기도 한다. 예를 들어, Groucho Computing Center에 있는 유지자(maintainer)가 각 부(department)를 위한 서브 도메인을 만들 수도 있다. 이미 위에서 maths와 physics라는 서브도메인을 보았다. 만약 Physics Department에 있는 네트 워크가 엉망진창인 상태로 발견이 된다면, 이 네트워크 관리자에게 physics.groucho.edu 도 메인 을 관리하게끔 할지도 모른다. 어쩌면 이 사람들은 그들이 좋아하는 호스트명을 사용할 수 도 있 고, 유행에 따라 네트워크를 관리할 수도 있으며, 외부 간섭을 전혀받지 않은 상태에서 IP 주소를 할당할 수도 있다.

이런식으로 일어날 수 있는 현상들 때문에, 네임 영역은 zone으로 나누어지게 되며, 각 네임 영 역은 하나의 도메인으로 뿌리를 내리게 된 것이다. 여기서 zone과 domain사이에는 아주 민감한 차이가 있다는 것을 주의하라; domain groucho.edu는 Groucho Marx University에 있는 모든 호스트를 둘러싸고 있는 반면에 zone groucho.edu는 Computing Center가 직접적으로 관리 하는 호스트 예를 들어 수학부 (Mathematics Department)만을 포함하고 있다. Physics Department에 있는 호스트들은 다른 zone 즉, physics.groucho.edu에 속해 있다. 그림 2.3에서, 하나의 zone의 시작이 작은 원으로 표시되어 있고, 그 원의 왼쪽에는 도메인이 있다.

### 2.6.3 Name Lookups with DNS

잠깐 보아서 이러한 모든 도메인과 존(zone)은 대단히 복잡한 작업에 대한 하나의 해결방 안처럼 보인다. 결국, 호스트명을 할당할 수 있는 중심 권한이 없다면, 보잘 것 없는 어플리케이션 이라도 어떻게 안다고 가정할 수 있겠는가?

여기 DNS에 관해 정말 소박하게 답변해 놓은 것이 있다. 만약 여러분이 erdos의 IP 주소 를 찾고 싶다면, DNS는 그것을 관리하고 있는 사람에게 물어보라고 말할 것이다. 그러면 그 관리자 가 여러분이 알고 싶어 하는 정보를 알려줄 것이다.

사실, DNS는 거대하게 분포되어 있는 데이터베이스이다. 이것은 네임 서버의 의미로써 수 행 되는데 주어진 도메인과 도메인 집합에 관한 정보를 제공한다. 각 존(zone)을 위해서, 적어 도 두 개의 네임 서버가 있으며, 그 네임 서버는 그 존(zone)에 있는 호스트에 관한 모든 정보를 가지고 있다. erdos의 IP 주소를 구하기 위해서는, groucho.edu zone을 위한 네임 서버에 접속해 서, 바 라는 정보를 얻어야 한다.

여러분이 생각하는 것보다 어쩌면 더 쉬울 지도 모른다. 내가 Groucho Marx University에 있 는 네임 서버에 어떻게 도달할 수 있는가? 또, 여러분의 컴퓨터가 address-resolving oracle 도 갖 추어 놓지 않은 경우에도 DNS는 또한 그러한 것을 제공해 준다. 여러분의 어플리케이션이 erdos 에 관한 정보를 찾아내고자 할 경우, 로컬 네임서버에 접속해서, 이른바 interative query를 수행 한다. 여러분의 로컬 네임서버는 루트 도메인을 위한 네임 서버에게 질의를 보냄으로써 작업 을 시 작하게 된다. 그리고 그것은 네임서버에게 erdos.maths.groucho.edu의 주소를 요청한다. 루트 네임서버는 이 이름이 루트권한에 속하지 않는다는 것을 인식하게 되며, 오히려 edu 도메인 에 그 러한 권한이 있다고 판단한다. 그래서, 루트 네임서버는 더 자세한 정보를 알고 싶다면, edu zone 네임서버로 접속하라고 말해줄 것이며, 그들의 주소와 함께 모든 edu 네임서버 목록을 페쇄 한다. 그러면, 여러분의 로컬 네임 서버는 edu 네임 서버중의 하나, 이를테면 a.isi.edu에게 질의 를 보 내게된다. 루트 네임 서버와 유사한 방법으로써, a.isi.edu는 groucho.edu가 있는 지역을 알 아차 리고, 여러분에게 그 서버가 있는 위치를 가르쳐 준다. 그러면 로컬 네임 서버는 erdos에게 질의 를 보내게 되며, 마지막으로 그것은 그 주소가 있는 지역을 알아차리게 되고, 일치하는 IP 주소를 그 어플리케이션으로 보내게 된다.

지금까지 설명한 것에서 보면, 단순하게 IP 주소를 찾는데에 엄청나게 많은 트래픽이 걸리 는 것처럼 보인다. 하지만 HOSTS.TXT에서 보게될 엄청나게 많은 양의 문서를 읽는 것보다는 간단 한 작업이다. 그러나 이러한 과정속에서도 개선되어야 할 많은 문제점들이 있다.

미래에는 질의를 하는동안 그 응답시간을 줄이기 위해, 네임서버는 로컬 cache에다가 구한 정 보를 저장할 것이다. 그래서 다음에 여러분의 로컬 네트워크에서 누구나가 groucho.edu에 있는 호스트의 주소를 찾고자 할 경우, 여러분의 네임서버는 전체 과정을 또 다시 거치지 않고 직접적 으로 groucho.edu에 접속하게 될 것이다. - *만약 그렇지 않다면, DNS가 다른 것과 같이 안좋은 방법일지도 모른다. 왜냐하면, 각 질의가 루트 네임 서버를 필요로 하기 때문이다.*

물론, 네임서버가 영원히 이 정보를 간직하고 있지는 않을 것이다. 오히려 약간의 기간이 지 나 면, 그것을 폐기처분할 것이다. 이러한 만료시간을 *time to live* 또는 TTL이라고 부 른다. 한 지 역을 책임지는 관리자가 DNS 데이터 베이스에 있는 각 자료에 이 TTL을 할당한다.

### 2.6.4 Domain Name Servers

네임서버들은 authoritative로 불리는 지역안에 있는 호스트에 관한 정보를 가지고 있다. 그 래서, 때때로 그것은 master name servers라고 하기도 한다. 이 지역에 있는 호스트에게 보내는 어떠 한 질의조차도 마지막에는 이러한 마스터 네임 서버에서 끝나게 된다.

한 지역의 간섭화면을 제공하기 위해서는 그것의 마스터 서버가 더 잘 표시해 줄 것이다. 데 이터 파일로부터 얻은 정보를 그 지역에 적재시키는 마스터 서버중에 하나인 primary 서버 를 만 들고, 규칙적인 간격으로 primary 서버에서 자료를 그 지엑에 전송해 주는 또 다른 secondary 서 버들을 만들어 줌으로써 이러한 작업을 이루어 낼 수 있다.

여러 네임서버를 가지는 이유중에 하나로는 적재 작업을 분산시키기 위해서이고, 또 다른 이 유로는 과다한 작업양을 여러 네임서버에 분배하기 위해서이다. 하나의 네임 서버 머신이 충돌이 나 손실과 같은 현상으로 인해 네트워크 연결에 실패했다면, 다른 서버로 모든 질의를 요청 할 것 이다. 물론, 이러한 구성이 여러분을 서버고장 (모든 DNS 요청에 대해 잘못된 결과를 산출 해내는 경우, 예를 들어 서버 프로그램으로 인해 소프트웨어 버그가 발생되는 경우)으로부터 보호 해 주지 는 못한다.

물론 여러분은 또한 실행하고 있는 네임 서버에서 제공되는 어떤 도메인도 믿지 못할 것 이 다. - *어쩌면 거의 그럴지도 모른다. 적어도 네임서버는 localhost를 위한 네임 서비스 와* ***127.0.0.1****에 해당하는 룩업을 예약*

*해 주어야 한다.* 그럼에도 불구하고 이러한 서버 형태가 유용한 경우도 있다. 이것은 여전히 로컬 네트워크 에서 실행하고 있는 어플리케이션을 위한 DNS 질의들을 처리하고 그 정보를 저장할 수 있 다. 이 러한 형태를 *caching-only* 서버라고 부른다.

### 2.6.5 The DNS Database

우리는 위에서 DNS가 호스트의 IP 주소를 처리하는 것 뿐만 아니라 네임서버에서 정보를 교환하는 일도 한다는 것을 알았다. 사실 DNS 데이터베이스는 많은 다른 형태의 엔트리를 가지고 있 다.

DNS 데이터 베이스에 있는 하나의 정보 조각들을 *resource record* 줄여서 RR이라고 부른다. 각 레코드는 그것과 관련되어 있는 형태를 가지고 있고, 그것을 표현하는 데이터층을 기술 해 주 고 있으며, 하나의 클래스는 그것을 사용하는 네트워크 형태를 명시해 주고 있다. 후자는 IP 주소 들 (the IN class) 또는 MIT에서 사용되는 Hesiod 네트워크의 주소와 같은 또 다른 어드레 싱 방 법의 필요를 수용시키고 있다. 기본적인 resource record 형태는 하나의 IP 주소와 함께 하나의 fully qualified domain name과 관련되어 있는 하나의 레코드를 말한다.

물론, 호스트가 여러개의 이름을 가질 수도 있다. 하지만 이러한 이름들중 하나는 꼭 공식적 으 로 확인될 수 있는 canonical host name 이어야 한다. 반면에 다른 이름들은 단순히 전자에 서 언 급하고 있는 가명들이다. 이 두가지 형태에서 차이점을 말한다면, canonical 호스트명은 관 련되어 있는 레코드가 오직 하나밖에 없지만, 다른 호스트명은 canonical 호스트명을 가리키고 있는 CN- AME형태의 레코드를 가지고 있다.

우리가 여기서 모든 형태의 레코드를 다룰 수는 없지만, 다음장에서 몇 개를 설명해 놓았으 며, 여기 믿을 만한 예제를 들어 놓았다. 그림 2.4는 physics.groucho.edu 지역(zone)을 위한 네 임서 버로 적재되는 도메인 데이터베이스의 한 부분을 보여주고 있다.

`Figure 2.4: An excerpt from the named.hosts file for the Physics Department`

A와 CNAME은 일단 제쳐놓고, 여러분은 파일의 제일 윗 부분에서 특별한 레코드를 볼수 있 다. 이것은 SOA (Start of Authority) 리소스 레코드이다. 이것은 그 지역에 있는 일반적인 정보 를 가지고 있다. 이를 테면, 이것은 모든 레코드를 위한 time-to-live의 초기값을 포함하고 있다.

예제 파일에서 도트(.)로 끝나지 않는 모든 이름은 groucho.edu 도메인과 연관되어 해석된 다 는 것을 명심하라. SOA 리소스에서 사용되는 특별한 이름인 "@"은 그 자체의 도메인 네임 을 나 타낸다.

우리는 위에서 groucho.edu 도메인을 위한 네임서버들이 어쨋든 간에 physics 지역(zone) 에 관한 정보를 알고 있고, 그래서 그들의 네임서버로 질의를 요청할 수 있는 것을 보아왔다. 이것은 대개 한쌍의 레코드에 의해 수행된다 ; NS 레코드는 서버의 FQDN을 가지고 있고, 하나의 레코 드는 그 이름과 관련되어 있는 주소를 가지고 있다. 이러한 레코드가 네임 영역에 함께 저 장되는 이래로, 그것들을 자주 glue records라고 부르기도 한다. 이것들은 부 지역(parent zone)이 실제로 종속 지역에 있는 호스트에 관한 정보를 가지고 있는 레코드들의 대표적인 예이다. glue 레코드 는 그림 2.5에서 보는 것과 같이 physics.groucho.edu를 위한 네임서버를 가리키고 있다.

`Figure 2.5: An excerpt fro the named.hosts file for GMU.`

### 2.6.6 Reverse Lookups

호스트에 속해 있는 IP-주소를 찾는 것 이외에도 주소에 해당하는 찾는 것이 때때로 더 바람하다. 이것을 *reverse mapping*라 부르고 신분을 검증하기 위

해서 여러 네트워크 서비스에 의해 사용된다. 단독 hosts 파일을 사용할 때, reverse lookups는 단순히 그 질 의에 해당하는 IP 주소를 가지는 호스트를 위한 파일을 찾아준다. DNS를 가지고 네임 영역 을 철 저하게 찾는 작업은 물론 질의와는 상관없는 작업이다. 대신에, 지금까지 만들어 지고 있는 특별 한 도메인인 in-addr.arpa은 dotted-quad 표기법으로 모든 호스트의 IP 주소를 포함하고 있다. 이를 테면, **149.76.12.4**라는 IP 주소는 **4.12.76.149.in-addr.arpa**라는 이름과 일치한다. 이러한 이름들을 그것들의 canonical 호스트명과 연결시킨 리소스 레코드를 *PTR*이라고 부른다.

어떤 권한을 가지는 지역을 만들어 내는 것은 대개 그 지역을 관리하는 사람이 IP 주소를 호 스트명에 할당하는 모든 작업을 완전하게 통제할 수 있다는 것을 의미한다. 그들은 대개 수동으 로 설정할 수 있는 하나이상의 IP 네트워크와 서브넷을 가진 이래로, DNS 지역(zone)과 IP 네트 워크를 1 대 ? (one-to-many)로 매핑하는 경향이 있다. 이를테면, 물리부(Physics Department)는 서브넷 **149.76.8.0, 149.76.12.0** 그리고 **149.76.14.0**를 포함하고 있다.

그 결과, in-addr.arpa 도메인에 있는 새로운 지역은 physics 지역에 따라 만들어 져야 하고, 그 부(department)에 있는 네트워크 관리자에게 권한을 위임받아야 한다; **8.76.149.in-addr.arpa, 12.76.149.in-addr.arpa** 그리고, **14.76.149.in-addr.arpa**. 그렇지 않고, Collider Lab에 다가 새로운 호스트를 설치하는 경우, 그들의 in-addr.arpa 지 역 (zone) 파일에 입력되어 있는 새로운 주소를 가지기 위해서 그들의 부(parent) 도메인에 접속할 필요가 있을 것이다.

서브넷 12를 위한 지역(zone) 데이터베이스가 그림 2.6에 나타나 있다. 그들의 부 지역 (parent zone)의 데이터베이스와 일치하는 glue 레코드들은 그림 2.7에 나타나 있다.

Figure 2.6: An excerpt from the named.rev file for subnet 12

Figure 2.7: An excerpt from the named.rev file for network 149.76.

이것들중 가장 중요한 결과를 들자면, 지역(zone)은 단지 IP 네트워크의 superset으로 만들 어 질 수 있고, 이러한 네트워크의 넷마스크는 바이트를 경계로 해야 한다. Groucho Marx 대학에 있는 모든 서브넷들은 **255.255.255.0**인 넷마스크를 가지며, 어쨌든 **in-addr.arpa** 지역은 각 서브넷을 위해 만들어 질 수 있었다. 그러나, 대신에 넷마스크를 **255.255.255.128** 로 준다면, 서브넷 **149.76.12.128**을 위한 지역을 절대 만들어 질 수 없다. 왜냐하면, **12.76.149.in-addr.arpa** 도메인이 권한을 가지는 두 개의 지역 (각각 호스트명이 하나는 1에서 127까지, 또 하나는 128에서 255까지의 지역)으로 나누어져 있다고 DNS에게 말할 수 있는 방법이 없기 때문이다.

# 3 Configuring the Networking Hardware

## 3.1 Devices, Drivers, and all that

현재까지 우리는 네트워크 인터페이스와 일반적인 TCP/IP 개관에 대해 이야기 해 보았 다. 하지만, 하드웨어의 한 부분을 제어하는 커널에서 "Networking code"가 도대체 무슨일 을 하는지 정확히는 알지 못한다. 이러한 경우를 위해서, 이 장에서는 인터페이스와 드라이 버의 개념에 대해 다루어 볼까 한다.

우선 하드웨어 그 자체를 설명할 것이다. 예를 들어, 이더넷 보드; 이것은 얇은 에폭시 수지로 이루어져 있고, 그 속에는 각 번호를 가진 많은 양의 작은 칩들로 채워져 있으며, 그 보드를 PC의 슬롯에 꽂아 넣으면 된다. 여기서는 이런식으로 장치에 대해 설명할 것이다.

이더넷 보드를 사용할 수 있게 하기 위해서, 리눅스 커널에 특별한 기능(옵션) 을 표시해 두어야 한다. 그러한 특별한 방법으로 장치를 제어해 주어야 한다. 이러한 것들을 이른바 장 치 드라이버라고 한다. 예를 들어, 리눅스는 기능면에서 이더넷 보드와 유사한 종류의 장치 드라이버를 가지고 있다. 그러한 장치 드라이버는 그것의 제작자인 Donald Becker의 이름을 따서 "*Becker Series Drivers*"라고 부른다. 다른 예를 들어, D-Link 드라이버라는 것 이 있는 데, 이것은 병렬 포트에 연결되어 있는 D-Link 패킷 어댑터를 처리해 준다.

그런데, 장치 드라이버를 "처리한다"라는 말은 어떤 의미일까? 위에서 이더넷 보드에 대 해 설명해 놓은 부분으로 가보자. 드라이버는 어쨌든 간에, 주변장치의 내장 로직과 통신할 수 있어야 한다 : 즉, 드라이버는 보드로 명령어와 데이터를 보내야 하는 반면에, 보드는 드 라이버로 부터 받은 어떠한 데이터라도 전송받을 수 있어야 한다.

PC에서의 이러한 통신은 입출력 메모리 영역에서 이루어지며, 그것은 내장 레지스터와 대응할 수 있다. 입출력 메모리는 일반적으로 레지스터의 시작 부분이나 base address에 기 술되어 있다. 이더넷 보드의 전형적인 베이스 주소는 0x300, 또는 0x360이다.

그림 3.1: 장치 드라이버, 인터페이스 그리고 하드웨어와의 관계

대개, 여러분은 베이스 주소와 같은 하드웨어 개관에 대해서는 걱정하지 않아도 된다. 왜 냐하면, 커널이 부트 시간에 보드의 위치를 감지해 내기 때문이다. 이러한 것을 *autoprobing*이라고 부른다. 즉, 이것은 커널이 여러 메모리 위치를 읽어 들이고, 어떤 이더넷 보드가 설치되어 있는지를 그 데이터와 비교한다. 하지만, 자동으로 감지해낼 수 없는 이더넷 보드 또한 있을지도 모른다; 표준 보드와 전혀 호환성이 없는 값싼 이더넷 카드를 만들어 내는 경우이다. 그리고 나서 커널을 부팅할 때, 이더넷 장치를 감지해 내려고 시도할 것이다. 만약, 여러분이 하나 이상의 보드를 사용하고 있다면, 이러한 정보를 커널에 기술해 놓아야 한다.

여러분이 커널에 기술해 놓아야 하는 또 다른 변수로는 인터럽트 요청 채널이 있다. 커 널에서는 각 하드웨어 부품에 대해 인터럽트를 매기는데, 그들 부품들은 이 인터럽트에 의 해 처리될 필요가 있는 경우가 있다. 예를 들어, 어떤 데이터가 도착할 때, 특별한 상태가 발생하기도 한다. PC에서, 인터럽트들은 0과 1 그리고 3에서 15까지 번호를 부여한 15개의 인터럽트 채널들 중 하나에서 발생한다. 하드웨어 부품들이 각각 하나씩 인터럽트번호를 가 지고 있으며, 이러한 인터럽트번호를 *interrupt request number*, 또는 IRQ. - *IRQ 2와 9는 실제로 같다. 왜냐하면, PC는 각 8개의 IRQ를 가진 인터럽트 프로세서를 두줄로 직렬배열하고 있기 때문이다. 즉, 두 번째 프로세서는 첫 번째 프로세서의 IRQ 2에 연결되 어 있다.*라고 부른다.

2장에서 기술한대로, 커널은 이른바 인터페이스를 통해 장치(device)를 엑세스한다. 인터 페이스는 모든 종류의 하드웨어에서 데이터를 받거나 보내거나 하는 추상적인 기능을 제공 한다.

인터페이스는 그 이름과 동일한 것으로 간주한다. 이러한 것들은 커널에서 정의된다. 즉, /dev 디렉토리에 꼭 장치 파일이 있는 것은 아니다. 전형적으로 이더넷 인터페이스를 위한 이름으로는 eth0, eth1이 있다. 각 장치에 해당하는 인터페이스의 할당은 그 장치가 구성되 어 있는 순서에 따라 결정된다; 이를테면, 첫 번째로 설치되어 있는 이더넷 보드는 eth0이 될것이고, 다음것은 eth1으로 이름지어 질 것이다. 이러한 규칙들 중 물론 예외도 있다. SLIP 인터페이스는 동적으로 할당된다. 다시 말해서, SLIP 연결이 확립될 때, 인터페이스가 시리얼 포트에 할당된다.

그림 3.1에서 우리는 하드웨어, 장치 드라이버 그리고 인터페이스간의 관계를 볼 수 있다.

부팅할 때, 커널이 감지하는 장치와 설치되어 있는 인터페이스가 화면에 나타난다. 다음 예는 우리가 흔히 볼 수 있는 부트 화면이다.

```
.
.
This processor honours the WP bit even when in supervisor mode. Good.
Floppy drive(s): fd0 is 1.44M
Swansea University Computer Society NET3.010
IP Protocols: ICMP, UDP, TCP
PPP: version 0.2.1 (4 channels) OPTIMIZE_FLAGS
TCP compression code copyright 1989 Regents of the University of California
dl0: D-Link DE-600 pocket adapter, Ethernet Address: 00:80:C8:71:76:95
Checking 386/387 coupling... Ok, fpu using exception 16 error reporting.
Linux version 1.1.11 (okir@monad) #3 Sat May 7 14:57:18 MET DST 1994
```

지금 이것은 커널에서 TCP/IP 와 SLIP, CSLIP 그리고 PPP를 사용가능하게 컴파일하는 과 정의 일부분이다. 밑에서 세 번째 행은 D-Link 포켓 어댑터가 감지되었고, 설치되어 있는 인터페이스는 dl0라는 것을 말해준다. 만약 여러분이 다른 종류의 이더넷 카드를 가지고 있 다면, 커널은 대개 그 종류에 해당하는 카드를 감지해서, eth0 라는 인터페이스를 시작시키 는 행을 출력해 줄 것이다. 만약 여러분이 현재 설치되어 있는 이더넷 카드를 가지고 있다 면, 어떤 메시지도 볼 수 없다. 즉 이것은 커널이 여러분의 보드를 감지해낼 수 없다는 것을 뜻한다. 이것에 대해서는 다음절에서 상세히 다루겠다.

## 3.2 Kernel Configuration

대부분의 리눅스 배포본에서는 모든 일반적인 종류의 PC 하드웨어를 구동시켜 주는 부트 디스크를 가지고 있다. 이것은 그 부트 디스크에서 커널이 모든 일반적인 종류의 드라이버 를 가지고 있다는 것을 의미한다. 그러나 커널이 그 부분을 스왑 아웃 할 수 없기 때문에 이전의 시스템 메모리를 소비하게 된다. 그러므로, 여러분은 실제로 필요로 하고, 원하는 드 라이버만 포함시켜서 커널을 구성해야 한다.

리눅스 시스템을 구동시킬 때, 여러분은 꼭 자기가 만들고 있는 커널을 잘 알고 있어야 한다. 이것에 대해 기본적으로 설명하고 있는 문서로는 Matt Welsh가 쓴 " *Installation and Getting Started*"가 있다. 이것도 또한 *Linux Documentation Project* (LDP) 시리즈중 하나 이다. 이 절에서, 우리는 네트워킹과 관련되어 있는 구성 옵션만을 다룰 것이다.

여러분이 make config를 실행하기에 앞서, 일반적인 구성에대해 답해야 할 것이다. 이 를테면, 여러분이 커널의 수치연산 프로세서를 원하고 있는지 아닌지... 이러한 것들중 하나 로써, TCP/IP 네트워킹을 원하는지도 답해야 한다. 실제로 네트워킹을 하고 싶다면 'y'를 입력해야 한다.

### 3.2.1 Kernel Options in Linux 1.0 and Higher

일반적인 옵션에 대한 답변을 완성한 후에, SCSI 드라이버와 같은 여러 가지 전반적인 형 태에 대한 질문에 답변해야 한다. 다음에 보이는 것은 네트워킹 지원에 관한 질문이다. 이 구성옵션에 관한 세부 항목들은 이 질문이 끝난후 계속해서 나타날 것이며, 이러한 질문도 커널이 발전해 감에 따라 더 늘어날 것이다. 지금 보이는 것은 대부분의 커널 버전 1.0과 1.1에서 제공되는 옵션이다. (그에 관한 주석문은 이탤릭체로 나타낸다.);

```
*
* Network device support
*
Network device support? (CONFIG_ETHERCARDS) [y]
```

꺽쇠 묶음([])에서 나타난 매크로 이름은 무시해 버려라. 여러분이 어떤 형태의 네트워킹 장 치 즉, 이더넷, SLIP 또는 PPP를 사용하고자 한다면, 이 질문에 'y'라고 답해야 한다. 이 질 문에 'y'라고 답했다면, 자동으로 이더넷류의 장치를 지원하게 된다. 다른 형태의 네트워크 드라이버를 지원하고자 한다면, 개별적으로 선택해야 한다.

```
SLIP (serial line) support? (CONFIG_SLIP) [y]
 SLIP compressed headers (SL_COMPRESSED) [y]
PPP (point-to-point) support? (CONFIG_PPP) [y]
PLIP (parallel port) support? (CONFIG_PLIP) [n]
```

이러한 질문에 답변하려면 적어도 리눅스에서 제공하는 여러 가지 프로토콜에 대해서 약 간의 지식은 알고 있어야 한다. SLIP은 시리얼 라인을 통해서 IP 데이터 그램을 전송하는 것이다. compressed headers 옵션은 CSLIP을 위한 지원사항을 물어보는 것인데, 이 CSLIP 는 TCP/IP 헤더를 적어도 세바이트로 압축하는 기술을 말한다. 이 커널옵션이 자동으로 CSLIP을 지원해 주는 것이 아님을 기억하라. 대개 이것을 위해 특 별한 커널 기능을 필요로 한다.

PPP는 시리얼 라인을 통해서 네트워크 트래픽을 보내주는 또 다른 프로토콜이다. SLIP 보다 약간더 다루기 쉽고, IP에 제한되어 있지 않으며, 그것이 수행될 때, IPX를 지원해 준 다. 최근에 들어와서 이 PPP 옵션을 제공해 주고 있지만, 이 커널에서는 아직 이 옵션이 없 다.

PLIP는 패러랠 포트와의 연결을 통해서 IP 데이터 그램을 보내주는 방법을 제공한다. 이 것은 대개 DOS를 실행하고 있는 PC와 통신하기 위해서 사용한다.

다음 질문은 여러 컴퓨터 회사에서 만들어낸 이더넷 보드에 관한 질문들이다. 더욱더 많 은 드라이버가 개발되고 있다. 만약 여러분이 여러 다른 기계에서 사용할 수 있는 커널을 만들고자 한다면, 하나이상의 드라이버를 선택할 수 있다.

```
NE2000/NE1000 support (CONFIG_NE2000) [y]
WD80*3 support (CONFIG_WD80x3) [n]
SMC Ultra support (CONFIG_ULTRA) [n]
3c501 support (CONFIG_EL1) [n]
3c503 support (CONFIG_EL2) [n]
3c509/3c579 support (CONFIG_EL3) [n]
HP PCLAN support (CONFIG_HPLAN) [n]
AT1500 and NE2100 (LANCE and PCnet-ISA) support (CONFIG_LANCE) [n]
AT1700 support (CONFIG_AT1700) [n]
DEPCA support (CONFIG_DEPCA) [n]
D-Link DE600 pocket adaptor support (CONFIG_DE600) [y]
AT-LAN-TEC/RealTek pocket adaptor support (CONFIG_ATP) [n]
*
* CD-ROM drivers
*
...
```

파일 시스템 절(section)에서, 마지막으로, 환경 구성 스크립트는 여러분에게 NFS, 네트 워킹 파일시스템을 지원할 것인지를 물어볼 것이다. NFS는 파일시스템을 여러 호스트로 보 내주는 역할을 한다. 꼭 그것이 호스트에 붙어 있는 임시 하드 디스크 인것처럼 파일을 보 여 준다.

```
NFS filesystem support (CONFIG_NFS_Fs) [y]
```

### 3.2.2 Kernel Options in Linux 1.1.14 and Higher

리눅스 1.1.14에서는 약간의 구성환경을 바꾸었으며, IPX 지원을 추가시켰다. 다음절에서는 여러분이 원하는 일반적인 네트워킹 옵션을 물어볼 것이다. 이것은 여러 가지 네트워킹 옵 션에 관한 질문을 말한다.

```
*
* Networking options
*
TCP/IP networking (CONFIG_TNET) [y]
```

여러분이 TCP/IP 네트워킹을 사용한다면, 이 질문에 'y'라고 답해야 한다. 그렇지 않고 'n'이라고 답했다 하더라도, IPX를 지원하는 커널을 컴파일할 수 있다.

```
IP forwarding/gatewaying (CONFIG_FORWARD) [n]
```

두 개의 이더넷이나 이더넷과 SLIP 링크사이에서 여러분의 시스템을 게이트웨이로써 사 용하고 있다면, 이 옵션을 사용할 수 있다. 이 옵션을 초기값대로 사용하지 않는다 하더라 도, 이른바 방화벽으로 호스트를 구성하고 싶어할 지도 모른다. 방화벽은 두 대 이상의 네트 워크에 연결되어 있는 호스트이지만, 그 네트워크 사이에서 라우트 트래픽을 하진 않는다. 방화벽은 대개 내부망에서 위험부담을 느끼고 있는 회사망으로부터 사용자들을 보호하는데 에 사용된다. 사용자들은 방화벽에 접속해서, 인터넷 서비스를 사용하지만, 그 회사망으로 들어오는 어떤 연결도 방화벽에 접근할 수 없기 때문에, 외부 공격으로부터 그 회사의 기계 를 보호할 수 있다.

```
*
* (it is saft to leave these untouched)
*
PC/TCP compatibility mode (CONFIG_INET_PCTCP) [n]
```

이 옵션은 몇몇 PC/TCP버전과, DOS를 기초로하는 PC에서, 구동하는 상업용 TCP/IP와는 비호환적으로 작동한다. 만약 여러분이 이 옵션을 사용한다면, 일반적으로 사용하는 UNIX 기계와 통신할 수 있지만, 그 기계에 연결하는 속도는 느려지게 될지도 모른다.

```
Reverse ARP (CONFIG_INET_RARP) [n]
```

이 기능은 RARP, *Reverse Address Resolution Protocol*을 사용할 수 있게 해준다. RARP는 디스크없는 클라이언트와 부팅할 때, IP 어드레스를 필요로하는 X 터미널에 사용 된다. 여러분이 몇몇 클라이언트를 제공할 계획이라면, RARP를 사용해야 한다. 최근에 나 온 네트워크 패키지들 (net-0.32d)은 rarp라고 하는 작은 유틸리티를 포함하고 있다. 이 유 틸리티로 시스템을 RARP 캐쉬에 추가시킬 수 있다.

```
Assume subnets are local (CONFIG_INET_SNARL) [y]
```

TCP를 통해서 데이터를 보낼 때, 데이터가 IP로 들어가기 전에, 커널은 여러패킷의 흐름을 중단시켜야 한다. 호스트를 위해서는 이더넷과 같은 로컬 네트워크를 통해서 데이터를 보낼 수있으며, 그 호스트는 먼거리에서 들어오는 데이터나 거대한 패킷또한 사용할 수 있을 것 이다.{{. 이것은 매우 작은 최대 패킷크기의 분열을 피하기 위한 방법이다. }} 만얀 여러분이 SNARL을 사용하지 않는다면, 커널은 그들의 네트워크들이 실제로 하나의 인터페이스를 가지고 있는 로컬네트워크라고 가정할 것이다. 그럼에도 불구하고 여 러분이 Groucho Marx University에 있는 클래스 B 네트워크를 찾고자 한다면, 클래스 B의 전체네트워크가 로컬이 되지만, 대부분의 호스트들의 인터페이스는 단지 하나이상의 서브넷 만을 가질 것이다. 만약 여러분이 SNARL을 사용한다면, 커널은 모든 서브넷이 로컬이라고 가정할 것이며, 대학에 있는 모든호스트와 통신할 때, 거대한 패킷을 사용하게 될 것이다.

만약 여러분이 특별한 호스트에 보내는 데이터를 위해서 조그마한 패킷을 사용하고자 한 다면, (이를테면, SLIP 연결을 통해 데이터를 보내고자 하는 경우) 여러분은 route에 mtu옵 션을 사용해서, 그 문제를 해결할 수 있다. 이것에 대해서는 이장의 맨끝부분에서 거론할 것 이다.

```
Disable NAGLE algorithm (normally enabled) (CONFIG_TCP_NAGLE_OFF) [n]
```

Nagle는 이른바 tinygrams라고 부르는 특별하게 보내는 작은 IP 패킷을 피하기위한 규 칙이다. 대화식 네트워킹 툴이 이러한 tinygram을 만들어 내는데, telnet 또는 rsh와 같은 네트워킹 툴로 이러한 tinygram을 보낸다. SLIP과 같은 저 대역폭 연결에서는 tinygram을 파과할 수 있다. Nagel 알고리즘은 어떤 상황하에서 발생하는 데이터를 TCP 전송층으로 건 어들이는 작업을 할 것이다. 만약 여러분이 전송도중 패킷을 잃어버릴 염려가 있다면, Nagle 알고리즘을 사용하지 않을 수도 있다.

```
The IPX protocol (CONFIG_IPX) [n]
```

이 옵션은 노벨 네트워킹에서 사용하는 전송프로토콜인 IPX를 사용할 수 있게 해준다. 이것은 여전히 개발중에 있고, 아직 실제로는 사용할 수 없다. 이것을 사용하는 한가지 이점 이라면, 언젠가는 여러분이 IPX를 기반으로하고 있는 DOS 유틸리티를 사용할 수 있고, PPP 연결을 통해서, 노벨에 기초를 두고 있는 네트워크에서 라우트 트래픽이 가능하다는 것이다. 노벨 네트워킹에서 고급 프로토콜을 지원할 날이 그다지 가깝지만은 않지만, 현재 소비되는 끔직한 많은 양의 비용을 생각해 보면, 반가운 소식중의 하나일 것이다.

1.1.16 커널에서, 리눅스는 또 다른 종류의 드라이버와 더미 드라이버를 지원해 주고 있 다. 다음 질문은 장치 드라이버를 사용할 것인지를 물어보는 질문이다.

```
Dummy net driver support (CONFIG_DUMMY) [y]
```

더미 드라이버를 사용하는 사람이 그다지 많지는 않지만, 스탠드얼론이나 SLIP 호스트에 서는 매우 유용하게 사용할 수 있다. 이것은 기본적으로 루프백 인터페이스를 매스커레이드 한 것이다. 루프백 인터페이스를 사용하는 이유는 이더넷에서가 아닌 SLIP을 사용하는 호스 트에서 구동하기 때문이며, 이것은 항상 여러분의 IP 어드레스를 유지시키는데에 도움을 준 다. 더미 인터페이스에 관한 더 자세한 것은 5장에서 다루것이다.

## 3.3 A Tour of Linux Network Devices

리눅스 커널은 여러형태의 장비를 위해서 많은 하드웨어 드라이버를 지원해 준다. 이 절에 서는 흔히 볼 수 있는 드라이버와 그것에 해당하는 인터페이스에 대해 간략히 설명하겠다.

리눅스에서는 표준으로 사용하는 인터페이스가 몇몇있다. 하나이상의 인터페이스를 지원 하는 대부분의 드라이버는 그 인터페이스 이름이 eth0, eth1과 같이 각각에 번호를 부여하 고 있다.

**lo**

로컬 루프백 인터페이스. 네트워크 어플리케이션 뿐만아니라 시험용 목적으로 사용된다. 어떤 경우 즉, 만들어진 데이터그램이 즉시 호스트의 네트워킹층으로 되돌아 오는 경우에는 마치 폐쇠회로와 같이 작동한다. 커널에는 항상 적어도 하나 이상의 루프백 장치가 나타나 있다.

**ethn**

n번째 이더넷 카드. 대부분의 이더넷 보드에서 사용하는 일반적인 인터페이스의 이름.

**dln**

이 인터페이스는 D-Link DE-600 포켓 어댑터와, 또 다른 이더넷 장치를 엑세스한 다. 이것은 패러랠 포트를 통해 구동하는 DE-600에서만은 특별하게 사용된다.

**sln**

n번째 SLIP 인터페이스. SLIP 인터페이스는 SLIP에 할당하는 시리얼 라인의 순서와 연관시켜서 생각해 볼 수 있다. 즉, SLIP을 구성하고 있는 첫 번째 시리얼 라인은 **sl0**가 된다. 커널은 최고 네 개의 SLIP 인터페이스를 지원해 준다.

**plipn**

n번째 PLIP 인터페이스. PLIP는 패러랠 라인을 통해서 IP 데이터 그램을 전송한 다. 커널에서는 최고 세 개까지 PLIP 인터페이스를 제공해 주고 있다. 이 인터페이스는 시스템이 부팅할 때, PLIP 드라이버에 할당되며, 패러 포트에 대응된다.

ISDN 또는 AX.25와 같은 인터페이스 드라이버들은 미래에 추가될지도 모른다. IPX (노 벨 네트워킹 프로토콜)과 AX.25 (ham radio amateurs에서 사용됨)를 위한 드라이버들은 현 재 개발중에 있으나 아직 초기 단계에 머물러 있다.

다음 절에서 우리는 위에서 기술한 드라이버 사용에 관한 자세한 정보를 다룰 것이다.

## 3.4 Ethernet Installation

현재 리눅스 네트워크 코드는 여러 가지 이더넷 카드 상표를 지원해 주고 있다. 대부분의 드라이버는 Donald Becker (becker@cesdis.gsfc.nasa.gov)에 의해 만들어 지고 있다. 그 는 National Semiconductor 8390 chip을 사용하는 카드를 위한 드라이버를 만들어낸 사람이 다. 이 드라이버는 Becker Series Drivers로 우리에게 잘 알려져 있다. 이 드라이버 중에는 패러랠 포트를 통해서 이더넷에 접근할 수 있게 해주는 D-Link 포켓 어댑터를 위한 드라이 버도 있다. 이러한 드라이버는 Bj rn Ekwall (bj0rn@blox.se)에 의해 만들어 졌다. DEPCA 드라이버는 David C. Davies (davies@wanton.lkg.dec.com)에 의해 만들어 졌다.

### 3.4.1 Ethernet Cabling

만약 여러분이 일생에 딱 한번 이더넷을 설치하고자 한다면, 여기 케이블링이란 용어가 여 러분에게 적합할 것이다. 이더넷은 케이블링에 대해서는 매우 까다롭다. 이 케이블 양쪽 끝 레지스터는 50 옴(ohm)으로 맞추어져 있어야 하며, 여러분은 어떻게 해서든지 그것들을 분 기시켜 놓으면 안된다. (이를테면, 세 개의 케이블은 스타형(star-shape)으로 연결되어야 한 다. 만약 여러분이 T자 형태로 접합되어 있는 BNC 커넥터와 함께 얇은 동축 케이블을 사 용하고 있다면, 반드시 보드의 커넥터에 접합할 부분을 꼬아서 연결시켜야 한다.

만약 여러분이 thicknet에 연결하려고 한다면, 반드시 트랜스시버를 거쳐서 여러분의 호 스트를 접촉시켜야 한다. (때때로 이것을 Ethernet Attachment Unit라고 부른다.) 여러분은 그 트랜스시버를 보드에 있는 15핀 AUI 포트에 꽂아넣고 실드 케이블을 사용할 수도 있다.

### 3.4.2 Supported Boards

지원하고 있는 보드의 완전한 리스트를 볼려면 Ethernet HOWTO 문서를 참고하라. 이것은 매달 Paul Gortmaker. - *Paul에게 문의할 사항이 있다면, gpg109@rsphysse.anu.edu.au로 연락하기 바란다.* 에 의해 comp.os.linux.announce에 포스트되고 있다.

여기에서 보는 목록들은 리눅스에서 지원하는 가장 널리 알려진 보드를 말해주고 있다. 실제로 HOWTO 목록에는 여기서 보는 것의 세배정도의 목록을 볼 수 있다. 이 목록에서 여러분이 가지고 있는 이더넷 보드를 찾으려고 한다면, HOWTO 문서를 보는편이 더 낫다. 이 문서에는 때때로 이러한 카드를 운영하는 중요한 세부항목들을 포함하는 경우도 있다. DMA에 기초를 두고 있는 이더넷 보드는 Adaptec 1542 SCSI controller과 같은 DMA 채널 을 사용한다. 여러분이 이더넷 보드의 DMA 채널을 다른 것으로 바꾸어 놓지 않는한, 이더 넷 보드가 만들어내는 패킷 데이터가 위치하는 지역이 마음대로 변할수도 있다.

**3Com EtherLink**

3c503, 3c503/16, 3c507 그리고 3c509를 지원한다. 3c501도 지원하지 만 이것은 속도가 매우 느리다.

**Novell Eagle NE1000 과 NE2000 그리고 여러 가지 호환기종들.**

NE1500과 NE2100도 지원한다.

**Western Digital**

SMC/WD8003과 WD8013 (SMC Elite와 SMC Elite Plus와 같다.) 을 지원하며, SMC Elite 16 Ultra도 새롭게 지원하고 있다.

**Hewlett Packard**

HP 27252, HP 27247B, 그리고 HP J2405A를 지원한다.

**D-Link**

DE-600 포켓 어댑터, DE-100, DE-200 그리고 DE-220-T를 지원한다. 그리고, PCMCIA 카드. - *다른 랩탑과 연관되어 tsx-11.mit.edu에 있는 packages/laptops에 올라오고 있다.*인 DE-650-T를 위한 패치 킷도 있다.

**DEC**

DE200 (32K/64K), DE202, DE100 그리고 DEPCA rev E를 지원한다.

**Allied Teliesis**

AT1500과 AT1700을 지원한다.

리눅스에서 이러한 카드 중 하나를 사용하고자 한다면, 리눅스 배포본에 포함되어 있는 커널을 컴파일하여 사용할 수도 있다. 이러한 카드는 일반적으로 그에 해당하는 드라이버를 가지고 있다. 장기간 동안 사용하고자 한다면, 여러분이 실제로 필요한 드라이버를 커널에 포함시켜서 컴파일 하는 편이 더 낫다.

### 3.4.3 Ethernet Autoprobing

부팅할 때, 이더넷 코드는 여러분의 보드를 지정된 지역에 놓으려고 할 것이다. 이 코드는 다음에 보이는 어드레스와 순서대로 카드를 검사할 것이다.

오토프로빙 코드에는 두가지 한계가 있다. 그중 하나는 모든 보드를 적절히 인식할 수 없다는 것이다. 이것은 일반적인 보드의 호환기종과 WD80x3 보드에서 종종 발생하는 현상 이다. 두 번째 문제는 커널이 순간에 하나 이상의 보드를 오토프로브할 수 없다. 이러한 현 상은 여러분이 어떤 보드가 어떤 인터페이스를 제어하는지를 모르는 것과 같이 생각해 볼 수있다.

만약 여러분이 하나이상의 보드를 사용하고 있거나, 오토프로브가 여러분의 보드를 감지 하는데에 실패했다면, 여러분은 반드시 카드의 베이스 어드레스와 이름을 커널에 명시해야 한다.

Net-3에서, 이것을 수행하기 위해서는 두가지 다른 형태의 방법을 취할 수 있다. 그 중 한가지 방법으로는 커널 소스 코드에 있는 drivers/net/Space.c 파일 (드라이버에 대한 모 든 정보를 담고 있다.)에 특정 정보를 변경시키거나 추가시켜 주는 것이다. 여러분이 네트워 킹 코드에 익숙해 있다면 이 방법을 추천해 주고 싶다. 더 나은 방법으로는 부팅할 때, 이 정보를 커널에 제공하는 것이다. 만약 부트 시스템으로 lilo를 사용한다면, lilo.conf 파일에 append 옵션을 명시해 둠으로써 커널에 있는 변수들을 그냥 지나칠 수 있다. 이더넷 장치 를 위한 정보를 커널에 명시하기 위해서는, 다음에 보이는 변수들을 없애줄 수 있다.

```
ether=irq, base_addr, param1, param2, name
```

처음 네 개의 변수들은 숫자로 되어 있는 반면에 마지막 변수는 장치명을 뜻하는 것이 다. 모든 숫자값들은 임의로 추가시킬 수 있다; 만약 그것들을 생략하거나 0으로 설정해 두 었다면, 커널은 그 장치를 검사함으로써, 그 값을 감지해내려 하거나 초기값을 사용할 것이 다.

첫 번째 변수는 장치에 할당되어 있는 IRQ를 설정하는 부분이다. 초기값으로, 커널은 장 치의 IRQ 채널을 자동으로 감지할 것이다. 3c503 드라이버는 특별한 형태를 가지고 있다. 이것은 IRQ를 5, 9, 3, 4를 선택하고, 이 라인에서 사용하기 위한 보드를 구성한다.

base_addr 변수는 보드에 I/O 베이스 어드레스값을 주는 역할을 한다; 위에서 봤던 어 드레스를 검사하기 위해서는 커널에 0이라는 값을 주어야 한다.

나머지 두 개의 변수들은 다른 형태의 드라이버에서 다르게 사용될 수도 있다. WD80x3 과 같은 공유 메모리 보드를 사용하기 위해서는, 공유 메모리 지역의 시작과 끝 어드레스를 명시해 주어야 한다. 다른 카드는 대개 디버깅 정보를 설정하기 위해서 param1 변수를 사 용한다. 1부터 7까지의 숫자는 그 디버깅 정보의 수준이 증가하는 것을 나타낸다. 반면에 8 은 완전히 다른 역할을 한다; 0은 초기값을 의미한다. 3c503 드라이버는 내부 트랜스시버 (초기값) 또는 외부 트랜스시버 (숫자값은 1)를 선택하기 위해서 param2를 사용한다. 전자 는 보드에 붙어있는 BNC 커넥터를 사용하고, 후자는 AUI 포트를 사용한다.

만약 여러분이 두 개의 이더넷 보드를 가지고 있다면, 리눅스를 자동감지해 주는 하나의 보드를 가질 수 있으며, lilo에서 두 번째 보드의 변수를 지나칠 수 있다. 그러나 여러분은 먼저 드라이버가 실제로 두 번째 보드를 찾는지 확인해야 한다. 그렇지 않으면, 또 다른 하 나가 전혀 등록되지 않는 현상이 발생할 수도 있다. 여러분은 lilo에 있는 reserve 옵션을 그냥 지나치게 함으로써 이러한 문제를 해결할 수 있다. 두 번째 보드에 주어진 I/O 영역의 감지를 피하기 위해서는 커널에 분명히 명시해 두어야 한다.

이를테면, 여러분이 인터페이스가 eth1이고 어드레스 0x300에 있는 이더넷 보드를 리눅 스에 설치하고자 한다면, 여러분은 커널에 있는 다음과 같은 변수를 없애주어야 한다.

```
reserve=0x300,32 ether=0,0x300,eth1
```

reserve 옵션은 어떤 장치를 검사할 때, 보드의 I/O 영역에 접근하는 장치가 없는지를 확인한다. 여러분은 또한 eth0를 오토프로빙하는 작업을 무시해 버리기 위해서도 커널 변수 를 사용할 수 있다.

```
reserve=0x340,32 ether=0,0x340,eth0
```

완전히 오토프로빙을 해제하기 위해서는, 다음과 같은 변수를 base_addr에 명시해 줄 수 있 다.

```
ether=0, -1, eth0
```

## 3.5 The PLIP Driver

PLIP, Parallel Line IP는 여러분이 두 대의 컴퓨터 만을 연결해서 네트워크를 구성하고자 할 때 사용하는 아주 값싼 방법이다. 이것은 패러랠 포트와 10kBps에서 20kBps까지의 속도 를 낼수 있는 특별한 케이블을 사용한다.

PLIP는 원래 주식회사 Crynwr에서 제작한 것이다. 이것은 패러랠 포트를 사용하여 PC 에서 장시간동안 네트워크를 하기위해 만들어 졌으며, 단 방향 프린터 포트를 사용한다; 이 것은 PC에서 주변장치로 데이터를 보낼

때 단지 여덟 개의 데이터 라인만을 사용할 수 있 다. PLIP는 입력을 위해서 포트의 다섯가지 상태 라인만을 사용함으로써 이러한 작업을 수 행한다. 그리고 PLIP는 모든 데이터를 4비트씩 전송해야하는 제한 사항을 가지고 있다. 이 러한 운영 모드를 mode zero PLIP라고 부른다. 오늘날, 이러한 단 방향 포트는 더 이상 사 용되지 않고 있다. 그래서, mode 1이라고 부르는 PLIP 확장시킨 것이 나왔는데, 이것은 전 체 8비트 인터페이스를 사용하게끔 제작되었다.

현재, 리눅스는 오직 mode 0만을 지원해 주고 있다. 이것은 PLIP의 초기 코드와는 그 성격이 판이하게 다르다. 지금은 Crynwr에서 수행하는 PLIP와 NCSA telnet. - *NCSA telnet는 이더넷 또는 PLIP를 통해 DOS에서 TCP*IP를 구현해주는 특별한 프로그램이며, telnet와 FTP를 지원해 주고 있다./에서 사용하는 PLIP 드라이버와 호환성을 갖도록 만들어내고 있는 추세이다. PLIP를 사용해서 두 대의 컴퓨터를 연결하기 위해서는, 몇 몇 가게에서 판매하고 있는 "Null Printer" 또는 "Turbo Laplink" 케이블과 같은 특별한 케이블을 사용해야 한다. 하지만 여러분 자신도 쉽게 이것을 만들 수 있다. 이것에 대한 자세한 사항은 부록 A에 소개하고 있다.

리눅스에서 사용하는 PLIP 드라이버는 무수히 많은 사람들이 이루어낸 성과이다. 이것은 현재 Niibe Yutaka가 관리하고 있다. 만약 이 드라이버가 추가되어 있는 커널이 컴파일되어 있다면, 각 프린터 포트를 위한 네트워크 인터페이스가 설정되어 있을 것이다. plip0는 패러 랠 포트 lp0와 일치하며, plip1은 lp1과 일치한다. 포트에 인터페이스를 매핑하는 작업은 다 음과 같다;

만약 여러분이 다른 방법으로 프린터 포트를 구성하고 있다면, 리눅스 커널 소스 또는 새로운 커널에 있는 drivers/net/Space.c에 있는 값을 변경시켜 주어야 한다.

하지만 이러한 매핑작업으로 인해 평상시 사용하는 패러랠 포트를 사용할 수 있다는 의 미는 아니다. 일치하는 인터페이스가 구성되었을때만 PLIP 드라이버를 엑세스할 수 있다.

## 3.6 The SLIP and PPP Drivers

SLIP (*Serial Line IP*)와 PPP (*Point-to-Point Protocol*)은 시리얼 라인을 통해 서 IP 패킷을 보내는 프로토콜로 널리 알려져 있다. 리눅스에서는 인터넷에 여러분의 컴퓨터를 접근시키기위해 동적 SLIP 과 PPP연결을 제공해 주고 있다. 그래서 각 사용자에게 IP 연결을 제공해 주고 있다.

SLIP와 PPP를 실행하기 위해서, 하드웨어 정보를 수정할 필요는 없다; 여러분은 어떤 시리얼 포트를 사용할 수 있다. 시리얼 포트 환경이 TCP/IP 네트워킹에 명시되어 있지 않 은 관계로 여러 장에서 이것에 대해 기술해 놓고 있다. 더 자세한 정보를 얻고 싶다면, 4장 을 참고하기바란다.

# 4 Setting up the Serial Hardware

netland에 사는 몇몇 사람들은 T1 인터넷 링크에 돈을 소비하지 않고, 자신의 PC에 정성을 쏟는다는 유머가 있다. 그럼에도 불구하고, 매일 뉴스와 메일을 받기 위해서, SLIP 링크, UUCP 네트워크, 공용 전화망을 사용하는 전자게시판 시스템에 의존한다고 말하고 있다.

이 장에서는 그러한 연결을 유지하기 위해 모뎀에 의존하는 모든 사람들에게 필요한 정 보를 가져다 줄 것이다. 하지만 이장에서 그러한 모든 정보를 가져다 줄 수 있는 것은 아니 다. 이를테면, 여러분의 모뎀을 다이얼인 방식으로 구현하는 방법같은 것들... 이러한 모든 화제들은 Greg Hankings. - *그의 주소는 gregh@cc.gatech.edu이다.* 가 쓴 Serial HOWTO에 기록되어 있을 것이며, 정기적으로 comp.os.linux.announce에 포스팅된다.

## 4.1 Communication Software for Modem Links

리눅스에서 유용하게 사용할 수 있는 몇가지 통신 패키지가 있다. 이것들 대부분이 *terminal program*이라고 하는 것인데, 이것은 사용자가 다른 컴퓨터에 접속하는 것을 도와 준다. 일반적으로 사용하는 터미널 프로그램으로는 kermit가 있다. 전화번호부와 원격 컴퓨 터 시스템에 접속하거나 전화를 걸어주는 스크립트 언어를 제공해주는 더욱더 안정되고, 유 용한 프로그램들이 얼마든지 있다. 그것들 중에 하나가 minicom이라는 것이 있는

데, 이것은 도스에 길들어져 있는 사용자들이 터미널 프로그램을 사용할 수 있게 도와준다. 이러한 프 로그램들 중 *seyon*이라고 하는 X윈도우용 통신 패키지도 있다.

리눅스용 BBS 패키지들 또한 전자게시판을 구현하려는 많은 사람들에게 도움을 주고 있다. 이러한 패키지 중 일부는 sunsite.unc.edu사이트의 /pub/Linux/system/Network 디 렉토리에서 찾을 수 있다.

터미널 프로그램과는 다르게 여러분의 컴퓨터에서 다른 곳으로 데이터를 전송하기 위해 대화식 시리얼 링크를 사용하는 소프트웨어도 있다. 이 기술을 사용해서 얻는 이점이라면, 몇 수십 킬로바이트 크기의 데이터를 전송 받을 때, 이를테면, 메일박스에 있는 온라인 메일 을 받을때나, 게시판에서 재미있는 글을 읽을 때의 시간을 좀 더 줄일 수 있다는 것이다. 다 른 한편으로, 여러분이 받는 정보를 적재하지 않기 때문에 더욱더 많은 디스크 공간을 필요 로 한다.

이러한 종류의 전형적인 통신 소프트웨어라고 한다면, 그것은 바로 UUCP일 것이다. 이 것은 이쪽 호스트에서 저쪽 호스트로 파일을 복사할 때나, 원격 호스트에서 프로그램을 실 행시키고자 할 때, 적합한 프로그램이다. 이것은 대개 개개인의 네트워크에서 뉴스나 메일을 받을 때 자주 사용한다. 리눅스에서 실행할 수 있는 Ian Taylor의 UUCP 패키지는 다음 장 에서 설명하겠다. 비대화식 통신 소프트웨어는 Fidonet를 거쳐 사용된다. ifmail과 같은 Fidonet 어플리케이션 포트또한 유용하게 사용할 수 있다.

SLIP, *serial line Internet protocol*은 대화식 (interactive)과 비대화식 프로그램사이에 서 중간 매개 역할을 한다. 많은 사람들이 그들의 대학망을 다이얼업하거나 FTP 세션을 구현 하기 위해 일반적으로 사용하는 공용 SLIP 서버를 위해 SLIP를 사용한다. SLIP은 또한 영 구적으로나 반영구적인 연결방법으로 LAN-to-LAN 커플링을 위해 사용하기도 하며, ISDN 에서도 사용한다.

## 4.2 Introduction to Serial Devices

시리얼 장치를 엑세스하기 위해 제공되는 유닉스 커널 장치를 tty, *TeletypeTM*이라 고 부른 다. 이것은 초기 유닉스 시절에 터미널 제조 업체중 한군데에서 사용했다. 현재는 문자로 데 이터를 처리하는 터미널 형태로 사용하고 있다. 이 장을 통해서, 우리는 커널 장치에 대한 용어를 정립해 나갈 것이다.

리눅스 배포본에는 세가지 형태의 tty: (가상) 콘솔, pseudo(의사)-터미널 (X11과 같은 어플리케이션으로 사용하는 two-way 파이프와 유사하다.) 그리고 시리얼 장치를 사용한다. 맨 마지막것도 시리얼 연결을 통해서 대화식 세션을 수행하기 때문에 이것도 tty에 포함시 킨다: 이것은 터미널 라인을 통한 하드 와이어드 터미널이나 리모트 컴퓨터에서 유래한 것 이다.

Tty는 구성 변수값을 가지고 있으며, 이것은 ioctl(2) 시스템 콜을 사용하도록 설정되어 있다. 이것들 중 다수의 tty는 여러 가지 형태의 연결을 처리하기 위해 더욱더 유연하게 다 룰 필요가 있은 후로, 오직 시리얼 장치에 맞추어져 있다.

가장 특이할 만한 라인 변수에는 라인 속도와 패리티를 들 수 있다. 그리고, 대문자와 소 문자를 변환시켜주는 옵션과 개행문자 (line feed)로 바꾸어주는 옵션도 있다. 또한 tty 드라 이버는 line discipline를 지원해 주는데, 이것은 완전히 다르게 동작하는 장치 드라이버를 만들 때 사용한다. 예를 들어, 리눅스에서 사용하는 SLIP 드라이버를 line discipline로 사용 하기도 한다.

라인의 속도를 측정할 때 사용하는 비트가 있다. 올바른 용어는 Bit rate라고 하는데, 이 것은 라인의 전송 속도를 의미하며, 초당 전송되는 비트수 (bps)를 말하는 것이다. 때때로, 여러분은 사람들에게 Baud rate라고 하는 말을 들었을 것이다. 이것은 그다지 올바른 용어 는 아니다. 이러한 두가지 형태의 용어는 절대 바꾸어서 말할 수 없다. Baud rate라는 말은 몇몇 시리얼 장치의 물리적인 특성을 말하는 것이며, 주로 전송되는 펄스의 클럭수를 말하 는 것이다. Bit rate는 두지점간에 존재하는 시리얼 연결의 현재 상태를 의미하는 것이며, 주로 초당 전송되는 평균 비트 수를 가리킨다. 전기적인 펄스가 발생할 때 생성되는 하나 이상의 비트를 인코드 시키는 대부분의 장치에서는 이 두가지 용어가 다르게 사용된다는 것 을 아는 것은 매우 중요하다.

## 4.3 Accessing Serial Devices

유닉스에서 사용하는 모든 장치와 유사하게, 시리얼 포드또한 특별한 장치 파일을 통해서 엑세스할 수 있다. 그 장치 파일은 /dev 디렉토리에 위치해 있다. 이 장치파일들은 시리얼 드라이버와 각 포트에 연관되어 있는 장치

파일의 두가지 양상을 띄고 있다. 이러한 파일에 의존하고 있는 장치들은 완전히 다르게 동작할 것이다.

첫 번째 장치파일은 포트를 통해서 다이얼링인을 할 때마다 사용한다; 네 개의 주번호를 사용하며, 그 이름은 ttyS0, ttyS1등이 있다. 두 번째 장치 파일은 포트를 통해서 다이얼링 아 웃을 할 때 사용되며, 파일 이름은 cua0, cua1을 사용한다. 다섯 개의 주번호를 사용한다.

만약 여러분이 COM1에서 COM4 포트중 하나를 사용한다면, 부 번호는 COM 포트번호 에 63을 더한 값이 될 것이다. 만약 이와 다르게 설정해 놓았거나, 다중 시리얼 라인을 지원 하는 보드를 사용하고 있다면, Serial HOWTO를 읽어보기 바란다.

여러분이 모뎀을 COM2에 맞추어 놓았다고 가정하자. 부번호는 65가 될것이며, 주번호 는 다이얼링 아웃을 위해 5를 사용할 것이며, 장치로는 cua1을 사용해야 한다. /dev 디렉토 리에 시리얼 tty목록이 있다. 다섯 번째와 여섯 번째 칸은 각각의 주 번호와 부 번호를 보 여주는 것이다.

```
$ ls -l /dev/cua*
crw-rw-rw-   1 root        5,   64 Nov 30 19:31 /dev/cua0
crw-rw-rw-   1 root        5,   65 Nov 30 22:08 /dev/cua1
crw-rw-rw-   1 root        5,   66 Oct 28 11:56 /dev/cua2
crw-rw-rw-   1 root        5,   67 Mar 19  1992 /dev/cua3
```

이러한 것들이 아무것도 없다면, 루트로 접속해서, 이러한 것을 만들어 주어야 한다

```
# mknod -m 666 /dev/cua1 c 5 65
# chown root.root /dev/cua1
```

어떤 사람들은 사용자들이 모뎀이 위치한 포트가 cua1이라는 것을 기억하지 않아도 되 도록 /dev/modem에 심볼릭 링크를 만들기를 권하기도 한다. 그러나 어떤 프로그램에서는 이 modem이라는 장치를 사용할 수 없고, 실제 장치명을 사용해야 하는 경우도 있다. 이러 한 프로그램들은 그 장치가 사용하는 신호에 이른바 lock files라는 것을 사용하기 때문이다. 관례에 따르면, cua1을 사용하는 잠금 파일은 LCK...cua1이 된다. 같은 포트를 대해 다른 장 치 파일을 사용한다는 것은 프로그램들이 각각의 다른 lock file들을 인식하지 못하고 있으 며, 동시에 장치 파일들을 사용한다는 의미이다. 결과적으로 보면, 어플리케이션들은 전혀 작동하지 않을 것이다.

## 4.4 Serial Hardware

현재 리눅스에서는 RS-232를 표준으로 사용하는 거의 모든 시리얼 보드를 지원해 주고 있 다. RS-232는 현재 PC 시장에서 사용되는 모든 시리얼 통신의 표준 규격이다. 이것은 단독 비트 전송 뿐만아니라 비트 동기를 위한 몇몇 회로들을 사용한다. 추가로 사용되는 라인들 은 모뎀에서 사용하는 반송파와 handshake의 존재 유무를 나타내주기 위한 것이다.

비록 하드웨어 handshake를 임의로 사용하는 것이지만, 매우 유용하게 쓰인다. 이것은 데이터를 받을 준비가 되어 있는지를 나타내 주는 상태와 수신자가 들어오는 데이터를 처리 할때까지 잠시 멈추어 있어야 하는 상태가 있다. 이러한 상태를 각각 "Clear to Send" (CTS) 와 "Ready to Send" (RTS)라고 부르며, 일반적으로 하드웨어 handshake 로써, 주로 "RTS/CTS"라고 부른다.

PC에서, RS-232 인터페이스는 대개 National Semiconductor 16450 칩 또는 이것의 새로 운 버전인 NSC 16550A. - *NSC 16550이라는 것도 있지만, 이것의 FIFO는 절대 작동 하지 않는다.*에서 유래한 UART 칩을 사용한다. 몇몇 제품들 (Rockwell 칩셋을 사용하는 대부분의 내장형 모뎀)은 완전히 다른 칩을 사용하고 있으며, 그 칩은 마치 16550 인 것처럼 작동하도록 프로그램되어 있다.

16450칩과 16550칩의 주요 차이점이라고 한다면, 후자는 16 바이트 FIFO 버퍼를 가지고 있는 반면, 전자는 단지 1 바이트 버퍼를 가지고 있다는 것이다. 즉 16450칩은 최고 속도 9600 보드에 적합하게 만들어져 있는 반면, 16550 호환 칩은 그 이상의 속도를 필요로 한다. 리눅스는 원래 UART 칩이였던 8250 칩도 지원한다.

커널이 기본 환경설정을 할 때, COM1에서 COM4까지 네 개의 표준 시리얼 포트를 확 인한다. 이전에 설명했듯이 이 포트들은 부번호 64에서 67까지의 장치를 할당받을 것이다.

여러분이 시리얼 포트를 적절하게 구성하려 한다면, rc.serial 스크립트에 Ted Tso의 setserial 명령을 설치해야 한다. 시스템 부팅시에 이 스크립트는 /etc/rc를 호출할 것이다. 전형적인 rc.serial 스크립트는 다음과 같다:

```
# /etc/rc.serial - serial line configuration script.
#
# Do wild interrupt detection
/sbin/setserial -W /dev/cua*

# Configure serial devices
/sbin/setserial /dev/cua0 auto_irq skip_test autoconfig
/sbin/setserial /dev/cua1 auto_irq skip_test autoconfig
/sbin/setserial /dev/cua2 auto_irq skip_test autoconfig
/sbin/setserial /dev/cua3 auto_irq skip_test autoconfig

# Display serial device configuration
/sbin/netserial -bg /dev/cua*
```

각 변수에 대한 설명을 알고 싶다면, setserial에 함께 따라오는 문서를 읽어보기 바란다.

만약 여러분의 시리얼 카드가 감지되지 않았거나, *setserial -bg* 명령어가 잘못된 설 정을 화면에 출력한다면, 여러분이 직접 그 구성환경을 설정해 주어야 한다. Rockwell(라겔) 칩셋 을 가지고 있는 내장형 모뎀을 사용하는 사용자들이 이 문제에 대해 보고해 주었다. 예를들 어 UART 칩이 NSC 16450으로 보고되었다면, 사실 그것은 NSC 16550 호환칩이다. 여기서 여러분은 다음과 같이 구성 명령을 바꾸어 주어야 한다.

```
/sbin/setserial /dev/cua1 auto_irq skip_test autoconfig uart 16550
```

COM 포트, 베이스 어드레스 그리고 IRQ 설정을 변경하는 옵션도 이와 유사하다. 이것 에 대해서는 setserial(8) 매뉴얼 페이지를 참고하기 바란다.

여러분의 모뎀이 하드웨어 핸드셰이크를 지원하고 있다면, 그것이 사용가능한지를 확인 해야 한다. 대부분의 통신 프로그램들은 이것을 사용가능하게 만들어 주지 않는다. 꼭 여러 분이 수동으로 설정해 주어야 한다. stty 명령을 사용하면, rc.serial 스크립트에서 가장 잘 수행된다:

```
$ stty srtscts < /dev/cua1
```

하드웨어 핸드셰이크를 효과적으로 검사하기 위해서는 다음과 같이 설정해 주어라.

```
$ stty -a < /dev/dua1
```

이것은 여러분에게 장치를 위한 모든 옵션을 보여줄 것이다; 옵션앞에는 꼭 '-'를 붙여 준다. 예를 들어 -crtscts옵션은 그것이 꺼져있다는 것을 의미한다.

# 5 Configuring TCP/IP Networking

이 장에서는 여러분의 컴퓨터에서 TCP/IP 네트워킹 설정에 필요한 모든 사항들을 다루어 불 생각이다. IP 주소 할당을 시작으로 해서, 천천히 TCP/IP 네트워킹 인터페이스의 환경구 성을 해나갈 것이다. 그리고 여러분이 네트워크 설치를 할 때 발생하는 여러 가지 문제를 해결하기 위해 유용하게 사용할 수 있는 몇가지 도구도 소개할 생각이다.

이 장에서 하는 대부분의 작업은 일반적으로 한번은 해야 할 작업이다. 여러분의 네트워 크에 새로운 시스템을 추가시키거나 시스템 전체를 재구성할 때, 대부분의 구성파일들을 손 봐주어야 한다. TCP/IP를 구성하기 위

해 사용하는 어떤 명령들은 시스템이 부팅되는 시간 에 실행된다. 시스템 부팅시 실행되는 파일들은 /etc/rc 스크립트에서 불러온다.

이 스크립트에서 네트워크와 관계되어 있는 내용을 기술해 놓은 파일을 **rc.net** 또는 **rc.inet**라고 한다. 때때로, 여러분은 **rc.inet1** 과 **rc.inet2**라고 하는 두 개의 스크립트를 볼 수 도 있을 것이다. 전자가 커널의 네트워킹 부분을 초기화 시키는 반면, 후자는 기본적인 네트워킹 서비스와 어플리케이션을 실행시키는 역할을 한다. 지금 부터는 후자와 관계된 내용만을 다룰 생각이다.

이 장에서는 **rc.inet1** 스크립트가 수행하는 작업에 대해 다룰 것이고, 다음 장(6장)에서 는 그것과 관계되어 있는 어플리케이션에 대해 다룰 것이다. 여러분이 이 장을 다 읽어 본 다 면, 여러분의 컴퓨터에 TCP/IP 네트워킹을 적절하게 구성할 수 있을 것이다. 그럼 먼저, **rc.inet1**에 있는 예제 명령을 사용해서 스크립트를 구성하라. 그리고 나서, 시동 시간에 **rc.inet1**이 실행되는지 확인하고 컴퓨터를 재부팅하라. 여러분이 좋아하는 리눅스 배 포본에 rc 스크립트와 관련되어 있는 좋은 예제 파일이 있을 것이다.

## 5.1 Setting up the proc Filesystem

Net-2 배포본의 몇몇 구성 도구는 proc 파일시스템에 의존하고 있다. 이것은 파일시스템과 같은 메카니즘을 통해서 커널로 run-time 정보를 엑세스하게 하는 인터페이스이다. 마운트 되면, 여러분은 다른 파일시스템에서와 같이 파일을 나열하거나 그 내용을 볼 수 있다. 시스 템 평균 적재량을 나타내는 loadavg 파일과 meminfo를 포함하고 있는 항목들은 현재 core 메모리와 스왑 사용법을 나타내 준다.

여기에 사용되는 네트워킹 코드는 net 디렉토리를 추가한다. 이 디렉토리에는 커널 ARP 테이블, TCP/IP 연결 상태, 그리고 라우팅 테이블과 같은 몇 개의 파일을 포함한다. 대부분 의 네트워크 관리 도구들은 이들 파일로부터 그와 관련되어 있는 정보를 얻는다.

proc 파일 시스템 (또는 procfs 로도 알려져 있다.)은 대개 부팅시간에 /proc와 마운트된 다. 가장 좋은 방법은 /etc/fstab에 다음과 같은 라인을 추가시켜 주는 것이다.

```
# procfs mont point:
none /proc proc defaults
```

그리고, /etc/rc 스크립트에서 "*mount /proc*"를 실행시킨다.

요즈음에 와서 procfs는 대부분의 커널에서 기본값으로 설정되어 있다. 만약 procfs가 여 러분의 커널에 있지 않다면, 여러분은 "mount: fs type procfs not supported by kernel" 과 같은 메시지를 얻을 것이다. 이럴 때는 커널을 재 컴파일하고 그 과정에서 procfs 지원 여 부를 묻는 질문에, 'y'라고 답해야 한다.

## 5.2 Installing the Binaries

만약 여러분이 이전에 패키지화된 리눅스 배포본을 사용하고 있다면, 그것은 아마도 네트워 킹 어플리케이션과 유틸리티에 따라오는 예제파일을 포함할 것이다. 그러한 경우에만, 여러 분이 새로운 커널 배포본을 설치하고자 할 때, 새로운 유틸리티를 구하던지 다시 설치를 해 주어야 한다. 새로운 커널은 때때로 변경된 커널 네트워킹 층에 관한 정보를 포함하는 경우 도 있다. 이러한 경우에 여러분은 기본 구성 도구를 갱신해주어야 한다. 어쩌면, 커널을 재 컴파일 하는 경우에만 최신 바이너리 패키지가 필요한 경우도 있다. 이것들은 대개 커널과 함께 *net-XXX.tar.gz*라는 이름으로 압축되어 배포된다. XXX는 버전 번호이다. 리눅 스 1.0 에 맞는 배포본은 0.32b이며, 1.1.12버전 이후의 커널은 0.32d를 필요로 한다.

여러분 힘으로 표준 TCP/IP 네트워크 어플리케이션을 설치하고 컴파일하고자 한다면, 여러분은 대부분의 리눅스 FTP 사이트에서 커널 소스를 구할 수 있다. Net-BSD 또는 다 른 소스에서는 다소 심하게 패치한 것도 있다. Xmosaic, xarchie 또는 Gopher과 IRC 클라 이언트와 같은 어플리케이션들은 개별적으로 구해야 한다.

Net-3의 공식 FTP 사이트는 *sunsite.unc.edu* `<http://sunsite.unc.edu>` 이며, 그 아래 *system/Network/sunacm*에 미러되어 있는 *sunacm.swan.ac.uk* `<http://sunacm.swan.ac.uk>`도 있다. 최신 Net-2e 패치 킷과 바이너리들은 *ftp.aris.com* `<ftp://ftp.aris.com>` 에서 찾아 볼 수 있다. BSD

에서 파생된 Matthias Urlichs의 네트워킹 코드는 *ftp.ira.uka.de* `<ftp://ftp.ira.uka.de>`에 있는 */pub/system/linux/netbsd*에서 구할 수 있을 것이다.

## 5.3 Another Example

이 책의 나머지 부분에서는 Groucho Marx University보다 조금 더 단순한 예를 들기로 하 겠다. 그리고 여러분이 실제로 부딪치게될 작업에 조금더 가까이 가보기로 하겠다. virtual beer를 양조하는 Virtual Brewery라고 하는 조그마한 회사가 있다고 가정하자. 그들의 사업 을 더욱더 효과적으로 관리하기 위해서, virtual 양조자가 그들의 컴퓨터를 네트워크에 연결 하려고 한다. 그리고 네트워크에 연결하고자 하는 컴퓨터는 리눅스 1.0을 구동시키려 한다.

양조장 건물 건너편에는 그와 비슷한 일을 하는 Virtual Winery가 있다. 여기서는 그들 자체내에 이더넷을 가지고 있다. 두 회사는 경영상의 목적으로 그들만의 네트워크를 구성하 려고 한다. 첫 단계로써, 두 서브넷 사이에서 데이터를 전송하기 위해 게이트웨이 호스트 컴 퓨터를 설정할 것이고, 메일과 뉴스를 교환하기 위해, UUCP를 바깥 세상에 링크시키려 할 것이다. 그리고 가끔 인터넷과의 연결을 위해서 SLIP 연결을 설정하려 할 것이다.

## 5.4 Setting the Hostname

비록 전부다 그렇다고 할 순 없지만, 대부분의 네트워크 어플리케이션은 로컬 네트워크명에 의존하고 있으며, 이치에 맞는 값으로 설정되어 있다. 이것은 대개 부팅할 동안 hostname 명령을 실행시킴으로써 설정된다. hostname에 이름을 설정하기 위해서는 다음과 같이 해야 한다.

```
# hostname name
```

이것을 위해서는 도메인네임없는 호스트명 (unqualified hostname)을 사용하는 것이 실 용적이다. 이를테면, Virtual Brewery에 있는 호스트는 vale.vbrew.com 또는 vlager.vbrew.com을 사용할 수도 있다. 이것들은 공식적으로 사용 하는 이름이며, fully qualified domain name (FQDN)이다. 그들의 로컬 호스트네임은 vale 와 같은 첫 번째 이름이 될 것이다. 하지만 로컬 호스트네임은 호스트의 IP 주소를 찾아내 는데에 자주 사용되기 때문에, 여러분은 resolver library가 호스트의 IP 주소를 찾아낼 수 있는지를 확인해야 한다. 즉, 이것은 여러분이 /etc/hosts에 그 이름을 입력해 주어야 된다 는 의미이다.

몇몇 사람들은 FQDN의 나머지 부분에 도메인 네임을 설정하기 위해서, domainname이 라는 명령어를 사용하라고 제안하기도 한다. 이 방법으로 여러분은 hostname과 domainname에서 나오는 결과물을 조합해서, 다시 FQDN을 얻을 수 있다. 하지만 이것이 최고의 방법은 아니다. 호스트의 NIS 도메인을 설정하기 위해서 일반적으로 domainname 명령을 사용한다. 이 도메인은 여러분이 속해 있는 도메인과는 다르다. NIS는 10장에서 다루기로 하겠다.

## 5.5 Assigning IP Addresses

여러분의 호스트에서 standalone operation (이를테면, INN 넷뉴스 소프트웨어를 실행할 수 있어야 한다.)을 위한 네트워킹 소프트웨어를 구성한다면, 이절을 읽지 않아도 된다. 왜냐하 면, 여러분에게 필요한 것은 루프백 인터페이스 (항상 127.0.0.1이다.)를 위한 IP 주소만을 필요로 하기 때문이다.

이더넷과 같은 실제 네트워크에서는 좀더 복잡한 작업을 필요로 한다. 여러분의 호스트 를 실제 존재하고 있는 네트워크에 연결하기 하고자 한다면, 접속하고자 하는 네트워크에서 IP 주소를 받을 수 있는지 관리자에게 물어 보아야 한다. 여러분이 직접 모든 네트워크를 설정한다면, 이전에 설명한 대로 여러분 자신에게 IP 주소를 할당해 주어야 한다.

로컬 네트워크에 있는 호스트들은 대개 같은 논리적인 IP 네트워크와 주소를 공유해야 한다. 즉 여러분이 IP 네트워크 주소를 할당해 주어야 한다. 만약 여러분이 여러 가지 물리 적인 네트워크를 가지고 있다면, 다른 네트워크 번호를 그것들에게 할당해 주거나, 하나의 IP 주소를 여러 서브네트워크로 쪼개기 위해 서브네트워킹

을 사용해야 한다. 여러분의 네트 워크가 인터넷에 연결되어 있지 않다면, 네트워크 주소를 마음대로 선택할 수 있다. 여러분 이 클래스 A, B 또는 C 중 하나를 선택하지 않았다면, 그 네트워크는 정확하게 작동하지 않을 것이다. 하지만, 여러분이 가까운 미래에, 인터넷을 사용할 생각이라면, 공식 IP 주소를 구해야 한다. 가장 최선의 방법은 여러분의 네트워크 서비스 프로바이더에게 물어보는 것이 다. 여러분이 인터넷에 접속할 경우에만 네트워크 번호를 구하고자 할 경우, hostmaster@internic.net으로부터 Network Address Application Form을 구해야 한다.

여러 가지 이더넷을 운영하기 위해서는 여러분의 네트워크를 서브넷으로 갈라놓아야 한 다. 서브넷팅은 단지 여러분이 하나 이상의 broadcast network를 가지고 있을 때만 필요하 다는 것을 알아 두어라; 여기서 *point-to-point* 링크는 생각하지 않는다. 예를 들어, 여러분 이 이더넷을 가지고 있고, 하나 이상의 SLIP를 바깥세상과 연결시키고자 한다면, 여러분의 네트워크를 서브넷으로 갈라 놓지 않아도 된다. 그 이유는 7장에서 설명하기로 하겠다.

한가지 예로, 양조장의 네트워크 관리자가 클래스 B에 해당하는 네트워크 번호를 NIC에 게 요청하고 나서 **192.72.0.0**을 부여받았다. 두 개의 이더넷을 수용하기 위해서, 관리 자는 추가적으로 서브넷 비트에 있는 호스트 부분에 해당하는 8 비트를 사용하기로 결정한다. 이 렇게 되면, 각 서브넷에 254개의 호스트를 부여할 수 있는 8 비트를 또 다시 가지게 된다. 그리고 나서, 관리자는 서브넷 번호로 brewery에게 1을, winery에게 2라는 번호를 할당한 다. 그러면, 각 네트워크 주소는 **191.72.1.0**과 **191.72.2.0**이 되며, 서브넷 마스크는 **255.255.255.0**이 될 것이다.

두 개의 네트워크에서 게이트웨이 역할을 하고 있는 vlager은 그것들 중 1이라는 호스 트 번호를 할당받았으며, IP 주소로는 각각 191.72.1.1과 191.72.2.1을 주었다. 그림 5.1은 두 개의 서브넷과 게이트웨이를 보여준다.

```
Figure 5.1: Virtual Brewery and Virtual Winery - the two subnets.
```

이 예제에서 나는 쉽게 이것을 유지하기 위해 클래스 B 네트워크를 사용하고 있다; 클래 스 C 네트워크가 조금 더 현실적이다. 새로운 네트워킹 코드를 가지고 있는 서브넷팅은 바이 트 바운더리에 제한되어 있지 않다. 그래서, 심지어 클래스 C 네트워크를 여러개의 서브넷 으로 나누기도 한다. 이를테면, 여러분은 넷마스크에서 호스트 부분에 해당하는 2비트를 사 용할 수 있다. 그렇게 되면, 각 네 개의 서브넷에 64개의 호스트를 부여할 수 있게 된다. - *각 서브넷의 마지막 숫자는 브로드캐스트 주소로 예약되어 있다. 그래서 사실상 각 서브넷마다 63개의 호스트를 부여할 수 있다.*

## 5.6 Writing hosts and networks Files

여러분의 네트워크를 서브넷으로 나눈후, /etc/hosts 파일을 사용하기 위해서 몇가지 호스트 네임 리솔루션(hostname resolution)을 준비해야 한다. 만약 DNS나 address resolution을 위 한 NIS를 사용할 생각이 아니라면, hosts 파일에 모든 호스트를 넣어 두어야 한다.

비록 여러분이 정상작동하에서 DNS나 NIS를 실행하고자 하는 경우에라로, /etc/hosts에 있는 모든 호스트네임의 서브넷을 가지고 싶어할 지도 모른다. 한가지 예를 들어, 부팅시에 아무런 네트워크 인터페이스가 실행되고 있지 않다 하더라도, 여러분은 name resolution을 가지고 싶어 할 것이다. 이것이 매우 편한 것일 뿐만아니라, rc.inet 스크립트에서 상징화된 호스트네임을 사용하도록 허락해 준다. 그래서, IP 주소들을 변경하고자 할 때, 거대한 rc 파일을 개별적으로 편집하는 대신, 갱신된 hosts파일을 모든 컴퓨터에 복사하고 나서, 재부 팅해야 한다. 대개, 여러분은 hosts에 모든 로컬 호스트네임과 주소를 넣어 둘 것이다. 그리 고 만약 사용한다면, 게이트웨이와 NIC 서버도 추가시켜야 한다. - *만약 여러분이 Peter Eriksson의 NYS를 사용한다면, 어떤 NIS 서버의 주소가 필요 할 것 이다. ypbind를 사용한 다른 NIS 수행작업은 실행시간에 그들의 서버에 위치한다.*

초기화 테스트동안에, 여러분의 resolver가 오직 hosts 파일에서 정보를 사용하는지 확인 해야 한다. 여러분의 DNS 또는 NIS 소프트웨어는 그것들이 사용되었을 때, 이상한 결과를 초래하는 예제파일과 같을지도 모른다. 호스트의 IP 주소를 찾을 때, 오직 /etc/hosts를 사 용하는 모든 어플리케이션을 만들기 위해서는, 여러분이 직접 /etc/host.conf 파일을 편집해 주어야 한다. 프롬프트 다음에 다음과 같은 라인을 추가하라.

```
order hosts
```

resolver 라이브러리의 설정은 6장에서 상세하게 다룰 것이다.

hosts 파일은 각 라인에 IP 주소, 호스트명, 그리고 추가적으로 오는 호스트명의 가명 목 록을 가지고 있다. 각 필드는 공백이나 탭으로 구분지으며, 주소 필드는 첫 번째 칸에서 시 작해야 한다. 첫 번째 칸에 해쉬표시 (#)를 가지고 있는 라인은 명령행에서 주석 처리된다.

호스트명은 FQDN이나 로컬 도메인으로 사용할 수 있다. vale를 예로 들어 보자. 여러분 은 대개 vale.vbrew.com과 같이 완전하게 자격을 갖춘 이름을 입력했을 것이다. vale 자체 는 hosts 파일을 의미한다. 그래서 vale라는 이름을 공식적인 이름과 단축형 로컬 네임으로 사용할 수 있다.

다음은 Virtual Brewery에서 hosts 파일이 어떻게 구성되어 있는지를 보여주는 예제 파 일이다. 이 파일에는 두 가지 특별한 이름 즉, vlager-if1과 vlager-if2가 포함되어 있는데, 이것들은 vlager에서 사용되는 인터페이스로써, 각각의 주소를 가지고 있다.

```
#
# Hosts file for Virtual Brewery/Virtual Winery
#
# IP            local         fully qualified domain name
#
127.0.0.1       localhost
#
191.72.1.1      vlager        vlager.vbrew.com
191.72.1.1      vlager-if1
191.72.1.2      vatout        vstout.vbrew.com
191.72.1.3      vale          vale.vbrew.com
#
191.72.2.1      vlager-if2
191.72.2.2      vbeaujolais   vbeaujolais.vbrew.com
191.72.2.3      vbardolino    vbardolino.vbrew.com
191.72.2.4      vchianti      vchianti.vbrew.com
```

여러분은 때때로 호스트의 IP 주소에 있는 네트워크 번호를 심볼릭네임으로 사용하고 싶 어할 것이다. 그렇게 되면, hosts 파일은 /etc/networks라고 하는 파일을 가지게 될 것이다. 그 파일은 네트워크 이름을 네트워크 번호에 대응시켜주는 역할을 한다. Virtual Brewery에 다음과 같은 networks 파일을 설치할 수도 있다:

```
# /etc/networks for the Virtual Brewery
brew-net     191.72.1.0
wine-net     191.72.2.0
```

## 5.7 Interface Configuration for IP

4장에서 설명한 대로 하드웨어를 설정하고 나면, 커널 네트워킹 소프트웨어라고 알려진 장 치를 만들어야 한다. 여기에서는 네트워크 인터페이스를 구성하고, 라우팅 테이블을 초기화 시키는 명령을 사용한다. 이러한 작업은 대개 시스템이 부팅될 때, rc.inet1 스크립트 파일에 서 수행된다. 여기에서는 ifconfig와 route라는 명령을 사용한다.

ifconfig라는 명령어는 커널 네트워킹 층에 접근할 수 있는 인터페이스를 만들 때 사용된 다. 그리고 IP 주소와 또 다른 변수의 할당작업과 인터페이스를 활성화 시키는데에도 사용 하며, 이러한 작업을 "taking up"이라고 부른다. 여기에서 활성화 한다는 것은 커널이 인터 페이스를 통해서 IP 데이터그램을 송수신 한다는 의미이다. 다음 명령은 이러한 작업을 수 행할 때 사용하는 가장 간단한 방법이다.

```
ifconfig interface ip-address
```

즉 이것은 ip-address를 interface에 할당하고 이것을 활성화 시킨다는 의미이다. 다른 모 든 변수들은 초기값으로 설정된다. 이를테면, 클래스 B 주소에 해당하는 255.255.0.0과 같은 IP 주소의 네트워크 클래스를 초기 서브넷 마스크로 간주하기도 한다. ifconfig는 이장의 마 지막 부분에서 상세하게 다룰 것이다.

route는 여러분이 커널 라우팅 테이블에서 라우트를 추가하거나 삭제할 때 사용하는 명 령어이다. 이것은 다음과 같이 사용한다.

```
route [add|del] target
```

여기서 add와 del은 target에 라우트를 추가할지 삭제할지를 결정하는 변수이다.

### 5.7.1 The Loopback Interface

첫 번째로 반응하는 인터페이스는 루프백 인터페이스이다.

```
# ifconfig lo 127.0.0.1
```

간혹 여러분은 IP 주소 대신에 사용하는 호스트명으로써 localhost라는 것을 볼수 있을 것이다. ifconfig는 hosts 파일에서 그 이름을 찾을 것이며, 그 파일에서 그 호스트명에 해당 하는 IP 주소 로써, **127.0.0.1**을 선언할 것이다.

```
# Sample /etc/hosts entry for localhost
localhost      127.0.0.1
```

인터페이스의 구성정보를 보기 위해서는, ifconfig 다음에 다음과 같이 인터페이스명을 적 어 주면 된다:

```
# ifconfig lo
lo        Link encap Local Loopback
          inet addr 127.0.0.1  Bcast [NONE SET]  Mask 255.0.0.0
          UP BROADCAST LOOPBACK RUNNING MTU 2000 Metric 1
          RX packets 0 errors 0 dropped 0 overrun 0
          TX packets 0 errors 0 dropped 0 overrun 0
```

보시다시피, 루프백 인터페이스의 주소 **127.0.0.1**이 클래스 A에 속한다음 부터는 그것 의 넷마스크는 **255.0.0.0**으로 할당되었다. 여러분도 알다시피, 인터페이스는 브로드캐스트 주소 를 가질 수 없게 되어 있다. 어쨌든 간에 이것은 루프백을 위해서도 그리 유용한 것은 아니 다. 하지만, 여러분의 호스트에 rwhod라고 하는 데몬프로그램을 실행시킨다면, rwho를 적절 하게 사용하기 위해서는 루프백 장치의 브로드캐스트 주소를 설정해 주어야 한다. 브로드 캐스트를 설정하는 방법은 "5.8 All about ifconfig" 절에 설명되어 있다.

현재 여러분은 작은 규모의 네트워크 정도는 설정할 수 있을 것이다. 그래도 빼먹은 것 이 있다면, IP를 말해주는 개체를 라우팅 테이블에 아직 추가하지는 않았다. 127.0.0.1이라는 목적지 주소를 라우트 해줌으로써, 여러분은 이 인터페이스를 사용할 수 있을 것이다. 방금 설명한 내용은 다음과 같이 해주면 된다.

```
# route add 127.0.0.1
```

또 다시, 여러분은 IP 주소 대신에 localhost라는 호스트명을 사용할 수 있게 되었다.

그런 다음에, 여러분은 모든 작업이 올바르게 작동중인지를 확인 해 보아야 한다. 이러한 작업에는 ping라는 도구를 사용하면 된다. ping은 sonar device와 맞먹는 네트워킹을 해주 며, 주어진 주소가 실제로 도착되었는지, 데이터그램을 보낼때나 그것을 다시 되돌려 보낼 때 발생하는 지연시간을 측정하는 등의 여러 가지 작업을 할 때 사용한다. 그 지연시간을 대개 "round-trip time"이라고 부른다.

```
# ping localhost
PING localhost (12.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=32 time=1 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=32 time=0 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=32 time=0 ms
^C

--- localhost ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0/0/1 ms
```

위에서 보여진 것처럼, ping을 실행시켰을 때, 사용자가 인터럽트를 걸지 않는한 그것은 영원히 패킷을 내보낼 것이다. 여러분이 직접 Ctrl-C를 타이프 하게 되면, 위와 같이 ˆC가 표시된다.

윗 예제는 **127.0.0.1**에 해당하는 패킷이 ping을 사용함과 동시에 적절하게 전송되고 다 시 되돌아 왔다는 것을 보여준다. 이것은 여러분이 첫 네트워크 인터페이스를 성공적으로 설정했다는 것을 의미한다.

만약 ping을 해서 얻은 출력이 위 예제와 전혀 다르게 보인다면, 문제가 조금 있다는 것 을 의미하는 것이다. 이런 경우에는 그 출력물이 제대로 설치되고 있지 않은 몇몇 파일을 가리키고 있는지 확인해 보아라. 즉 ifconfig와 route가 여러분이 실행시키고 있는 커널 배포 본과 호환되고 있는지 확인해 보아라. 결국 커널 컴파일시 네트워킹을 할 수 있게 만들어 놓아야 한다. (/proc/net 디렉토리에서 여러분은 이러한 정보를 볼 수 있다.) route 명령을 잘못 입력한 경우, 여러분의 모니터에는 ”Network unreachable”이라고 하는 에러 메시지가 뜰 것이다. 이런 경우, 혹시라도 ifconfig에서 부여한 것과 똑같은 주소를 입력했는지 확인해 보아라. 위에서 설명한 것만으로도 스탠드 얼론 호스트에서 충분히 네트워킹 어플리케이션 을 구동시킬 수 있다. 위에서 사용한 명령을 rc.inet1에 추가 시킨후 rc.inet1 스크립트들이 /etc/rc로부터 실행되고 있는지 확인해 보아라. 실행되고 있다면, 여러분의 컴퓨터를 재부팅 시켜라. 그리고 나서 여러 가지 어플리케이션을 한번 사용해 보아라. 이를테면, ”telnet localhost”라는 명령은 telnet이 여러분의 호스트에 접속을 시도하고 있음을 뜻한다.

그리고, 루프백 인터페이스는 이 책에서 보인 예제 뿐만아니라 실제로 몇몇 어플리케이 션에서 사용되고 있다. 그러므로, 여러분의 네트워크가 접속되었는지 그렇지 않은지를 개의 치 말고, 항상 루프백 인터페이스를 설정해 두어야 한다.

### 5.7.2 Ethernet Interfaces

이더넷 인터페이스 설정 또한 루프백 인터페이스와 매우 유사하다. 즉 여러분이 서브넷을 사용할 때, 몇가지 변수를 더 사용할 뿐이다.

Virtual Brewery에서 우리는 IP 네트워크를 여러개의 서브넷으로 나누어 보았다. 그것은 근본적으로 클래스 B에 해당하는 네트워크를 클래스 C에 해당하는 서브넷으로 이러한 환 경을 인식시키기 위한 인터페이스를 만들기 위해서는, 다음과 같은 명령을 주면 된다.

```
# ifconfig eth0 vstout netmask 255.255.255.0
```

즉, 이것은 vstout (**191.72.1.2**)라는 인터넷 주소를 eth0 인터페이스에 할당하는 작업이 다. 여기서 여러분이 넷마스크를 설정해 두지 않았다면, ifconfig는 IP 네트워크 클래스로부 터 넷마스크를 분류할 수 없을 것이다. 즉, 넷마스크를 255.255.0.0으로 인식하는 결과를 초 래하게 된다.

```
# ifconfig eth0
eth0    Link encap 10Mps Ethernet HWaddr 00:00:C0:90:B3:42
        inet addr 191.72.1.2 Bcast 191.72.1.255 Mask 255.255.255.0
        UP BROADCAST RUNNING MTU 1500 Metric 1
        RX packets 0 errors 0 dropped 0 overrun 0
        TX packets 0 errors 0 dropped 0 overrun 0
```

지금 여러분은 ifconfig가 브로드캐스트 주소 (위에서 보는 Bcast)를 일반적인 값으로 설 정해 준다는 것을 볼 수 있다. 이 값은 호스트 비트의 모든 설정값을 가진 호스트 네트워크 번호이다. 또한, message transfer unit (커널이 이 인터페이스로 전송할 수 있는 이더넷 프 레임의 최대 크기)는 1500 바이트 최대값을 가진다. 이러한 모든 값들은 추후에 설명하게 될 특별한 옵션으로 override되어 있다.

루프백 설정작업 때와 유사하게, 지금부터 여러분은 라우팅 엔트리를 설치해야 한다. 이 작업은 eth0를 통해서 커널에 도달할 수 있는 네트워크를 통보해 주는 역할을 한다. Virtual Brewer에서 여러분은 다음과 같은 명령을 줄 수 있다.

```
# route add -net 191.72.1.0
```

route가 어떤 경로를 거쳐서 인터페이스를 감지해 내지는 못하지만 이러한 작업이 오히 려 간단할지도 모른다: 커널은 구성되어 있는 모든 인터페이스를 검사하고 목적 주소 (이 경우에는 **191.72.1.0**)를 인터페이스 주소의 네트워크 부분 (인터페이스와 넷마스크의 비트 부분)과 비교를 한다. 여기에서 인터페이스는 단지 eth0와 일치된다.

그런데, 여기서 -net 옵션은 무엇일까? 이것은 route가 네트워크로 가는 경로와 단독 호 스트 (위에서도 보았듯이 이것은 localhost가 된다.)로 가는 경로, 두가지 다를 처리하기 때 문에 이 옵션을 사용한다. 주소가 dotted quad notation으로 주어질 때, route는 호스트 부분 의 비트가 네트워크 부분인지 호스트명 부분인지를 추적할 것이다. 만약 주소의 호스트 부 분이 0으로 되어 있다면, route는 그 주소가 네트워크를 나타내고 있다고 가정한다. 그래서, route는 **191.72.1.0**이 네트워크 번호 보다 오히려 호스트 주소라고 가정할 것이다. 왜 냐하 면, route가 지금 서브넷을 사용하고 있는지 알 수 없기 때문이다. 그러므로, -net 옵션을 줌으로써, 그것이 네트워크를 나타내고 있다고 명백하게 말해 주어야 한다.

물론, 위에서 준 route 명령은 어쩌면 조금 지루한 작업일 수도 있지만, 철자를 잘못 치 는 경우를 막을 수 있다. 이것보다 조금 더 편리한 방법이라면, /etc/networks에 네트워크 이름을 정의해 둘 수도 있다. 이것은 명령을 조금더 읽기 쉽게 하기 위한 명령이다; 심지어 -net옵션을 나타내 줄 수도 있다. 왜냐하면, route가 191.72.1.0이 네트워크를 가리키고 있 다는 것을 알고 있기 때문이다.

```
# route add brew-net
```

지금까지 여러분은 기본적인 설정작업을 끝마쳤으며, 여러분의 이더넷 인터페이스가 실 제로 작동하고 있는지 알고 싶다. 여러분의 이더넷에서 vlager과 같은 호스트를 선택하라.

```
# ping vlager
PING vlager: 64 byte packets
64 bytes from 191.72.1.1: icmp_seq=0, time=11. ms
64 bytes from 191.72.1.1: icmp_seq=1, time=7. ms
64 bytes from 191.72.1.1: icmp_seq=2, time=12. ms
64 bytes from 191.72.1.1: icmp_seq=3, time=3. ms
^C

----vstout, vbrew.com PING Statistics----
4 packets transmitted, 4 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 3/8/12
```

만약 여러분이 이와 다른 출력을 보았다면, 그것은 시스템이 깨졌다는 것을 의미한다. 만 약 평상시 보다 패킷 손실율이 지나치게 많다면, 그것은 하드웨어 문제일 가능성이 높다. 예 를들어, 터미네이터가 불량이라던지... 여러분이 만약 어떤 패킷도 받을 수 없다면, netstat로 인터페이스 구성환경을 검사해 보아야 한다. ifconfig에서 나타나는 패킷의 상태는 인터페이 스로 어떻게 패킷이 전달되는지를 알려준다. 만약 여러분이 원격 호스트로 접속하고 있다면, 그 기계 또한 인터페이스 상태를 검사해 보아야 한다. 이러한 방법으로 손실된 패킷이 어디 에 있는지를 알 수 있다. 게다가 여러분은 그 두 개의 호스트가 올바른 라우팅 엔트리를 가 지고 있는지를 알아 보기 위해서는 route라는 명령을 주어서 라우팅 정보를 살펴보아야 한 다. 아무런 옵션없이 route만 주어도

완전한 커널 라우팅 테이블을 출력한다. (-n 옵션은 호 스트 명을 사용하는 대신에 도트로 구분되어 있는 주소를 출력하는데에 사용한다.)

```
# route -n
Kernel routing table
Destination  Gateway  Genmake          Flags  Metric  Ref  Use    Iface
127.0.0.1    *        255.255.255.255  UH     1       0    112    lo
191.72.1.0   *        255.255.255.0    U      1       0     10    eth0
```

이러한 필드가 가지고 있는 의미는 'Checking with netstat' 절에서 설명한다. Flag는 각 인터페이스에 대한 일련의 플래그이다. U는 언제나 활동중인 인터페이스를 보여주는 것이 고, H는 그 목적 주소가 호스트를 가리키고 있다는 것을 뜻한다. 만약 H 플래그가 네트워크 라 우트로 설정되어 있다면, 반드시 route 명령 다음에 -net 옵션을 붙여주어야 한다. 라우트가 제대로 작동하고 있는지 알아보려면, Use 필드가 두 개의 ping 호출사이에서 증가하고 있는 지를 확인해 보아라.

### 5.7.3 Routing through a Gateway

앞절에서는 하나의 이더넷 상에서 호스트를 설정하는 경우를 살펴보았다. 게이트 웨이를 통 해 또 다른 곳으로 연결되어 있는 네트워크를 많이 볼 수 있을 것이다. 이러한 게이트웨이 들은 단순하게 두 개 이상의 이더넷과 연결되어 있는 경우도 있지만, 인터넷과 같은 외부세 계와 연결되는 경우도 있다. 게이트웨이를 사용하기 위해서는 네트워킹 층에 추가적으로 라 우팅 정보를 제공해 주어야 한다.

이를테면, Virtual Brewery와 Virtual Winery의 이더넷들은 vlager이라고 하는 게이트웨 이에 연결되어 있다. vlager이 이미 구성되어 있다고 가정하자. 우리는 단지 vstout의 라우 팅 테이블에 또 다른 엔트리를 추가 시켜 주기만 하면 된다. 이렇게 하게 되면, 이 라우팅 테이블이 커널에 이야기 해서, vlager을 통해 Winery 네트워크에 있는 모든 호스트와 연락 할 수 있게 해준다. 이러한 작업에서 route에 적합한 incantation은 아래와 같다: gw 키워 드 는 다음 변수가 게이트웨이를 가리키도록 해주는 역할을 한다.

```
# route add wine-net gw vlager
```

물론, 여러분이 이야기 하고 싶은 Winery 네트워크에 있는 어떤 호스트라도 Brewery 네 트워크에 일치하는 라우팅 엔트리가 있어야 한다. 그렇지 않으면, 여러분이 직접 vstout에 서 vbardolino로 데이터를 보낼 수도 있을 것이다. 하지만 vbardolino에서 돌아온 응답은 더 큰 버킷으로 보내질 것이다.

다음 예제는 두 개의 고립된 이더넷 사이에서 패킷을 교환하는 게이트웨이를 나타내준 다. 현재 vlager이 SLIP 링크를 통해서 인터넷과 연결되어 있다고 가정하자. 우리는 vlager에서 처리되는 데이터그램이 Brewery 이외의 목적 네트워크로 가길 원할 것이다. 이 러한 작업은 vstout를 디폴트 게이트웨이로 만들어 줌으로써 해결할 수 있다.

```
# route add default aw vlager
```

**0.0.0.0**이라는 주소를 가지고 있으며, 네트워크 이름으로 default라고 하는 것은 디폴트 라 우트를 나타내는 것이다. 이 이름은 route에 내장되어 있기 때문에 /etc/networks에 추가할 필요는 없다.

만약 호스트를 ping했을 때, 하나 이상의 게이트웨이를 거치면서 패킷의 거대한 손실이 발생된다면, 현재 혼잡한 네트워크에 있다는 것을 의미한다. 패킷 손실은 기술 부족면 보다 는 일시적인 과부하 때문에 발생하는 것이다. 그런 경우 들어오는 데이터가 지연되거나 감 소되기도 한다.

### 5.7.4 Configuring a Gateway

두 개의 이더넷 사이에서 패킷을 교환하기 위해 컴퓨터를 구성하는 작업은 매우 간단하다. 다시, vlager로 돌아와서 이것이 두 개의 이더넷 보드를 갖추고 있으며, 두 개 중의 하나의 네트워크로 연결하고 있다고 가정하자.

여러분은 각각의 인터페이스를 구성해 주어야 하며, 그 인터페이스에 그것들만의 IP 주소를 할당해 주어야 한다.

두 개의 인터페이스에 관한 정보를 아래와 같은 방법으로 hosts 파일에 추가시켜 주는 것이 유용하다. 그렇게 되면, 그 인터페이스에게 이름을 부여해 주는 작업이 용이해 지기 때 문이다:

```
191.72.1.1      vlager       vlager.vbrew.com
191.72.1.1      vlager-if1
191.72.2.1      vlager-if2
```

다음과 같은 순차적인 명령으로 두 개의 인터페이스를 설정해 줄 수 있다:

```
# ifconfig eth0 vlager-if1
# ifconfig eth1 vlager-if2
# route add brew-net
# route add wine-net
```

### 5.7.5 The PLIP Interface

두 대의 컴퓨터를 PLIP 링크를 시킬때는 이더넷을 사용할 때 해야 하는 작업과는 약간 다 르다. 전에는 브로드캐스트 네트워크와는 정 반대로, 단지 두 대의 호스트를 연결시켰기 때 문에 *point-to-point*라고 불렀다.

예를 들어, Virtual Brewery에 있는 몇몇 근로자들이 그들의 랩톱 컴퓨터를 PLIP을 사 용해서 vlager에 연결한다고 가정하자. 랩톱 그 자체를 vlite라고 부르며, PLIP에서는 단지 하나의 패러랠 포트만이 필요하다. 부팅시에, 이 포트는 plip1으로 등록될 것이다. 이 링크를 활성화 시키기 위해서는, 다음과 같은 명령을 사용해서, plip1 인터페이스를 구성해 주어야 한다.

```
# ifconfig plip1 vlite pointopoint vlager
# route add default gw vlager
```

첫 번째 명령어는 인터페이스를 구성하는 것이다. 즉, vlager의 주소를 가지고 있는 원 격지 주소로 point-to-point 연결을 한다고 커널에게 말해주는 역할을 한다. 그리고 두 번째 명령어는 게이트웨이 역할을 하는 vlager을 사용해서 디폴트 라우트를 설치하는 것이다. vlager상에서, ifconfig가 하는 역할은 링크를 활성화시키는 데에 꼭 필요하다. (route는 그 다지 필요하지 만은 않다.):

```
# ifconfig plip1 vlager pointopoint vlite
```

흥미로운 점은 vlager에 있는 plip1 인터페이스가 꼭 IP 주소를 가지고 있어야 될 필요 는 없지만 실제로 **191.72.1.1**이라는 주소를 가지고 있을 수도 있다.

현재 우리는 랩톱 컴퓨터에서 Brewery의 네트워크로 경로를 지정해 주어야 한다; Brewery의 호스트에서 vlite로 경로를 배정하는 과정에서 빼먹은 부분이 있다. 약간은 귀찮 은 방법이지만, 모든 호스트의 라우팅 테이블에 vlager이름의 게이트웨이를 vlite로 다시 경로를 배정해 주는 것이다:

```
# route add vlite gw vlager
```

임시적인 라우트에 직면했을 때, 그에 대한 좋은 해결책으로는 동적 라우팅을 사용하는 방법이 있다. 즉 라우팅 정보를 동적으로 분배하기 위해서는 모든 네트워크에 있는 호스트 에 라우팅 데몬인 gated를 설치해야 한다. 그러나 초기 시절에는 proxy ARP를 사용했었다. 그당시, proxy ARP를 가지고 있는 vlager은 그 자체의 이더넷 주소를 보냄으로써, vlite로 오는 어떤 ARP 질의에도 응답할 수 있었다. 이러한 효과로 vlite에 있는 모든 패킷들이 vlager로 완벽하게 전송되고, 그런다음 그 패킷들은 랩톱 컴퓨터로 다시 전송될 수 있었다. proxy ARP에 관한 자세한 사항들은 'Checking tht ARP Tables'에서 다루기로 하자.

미래의 Net-3 배포본에서는 plipconfig라고 하는 도구를 포함할 것이다. 이 도구는 여러 분이 프린터 포트의 IRQ를 설정할 수 있도록 만들어 준다. 어쩌면 이것이 일반적으로 사용 하는 ifconfig 명령 대신에 사용될 수도 있을 것이다.

### 5.7.6 The SLIP and PPP Interface

비록 SLIP와 PPP 링크가 PLIP 연결 때 처럼 단순하게 point-to-point 링크를 사용하고는 있지만, 이 두가지에 대해 이야기 할 것이 더 많다. 대개, SLIP 연결을 성립하기 위해서는 먼저 여러분의 모뎀을 통해서 원격지로 다이얼링 업을 해야하고, SLIP 모드에 맞게 시리얼 라인을 설정해 주어야 한다. PPP는 단순히 유행에 따라 사용된다. SLIP와 PPP 링크를 설정 할 때 필요한 도구는 7장과 8장에서 자세히 설명하겠다.

### 5.7.7 The Dummy Interface

더미 인터페이스는 정말 색다른 것이지만 매우 유용하게 쓰인다. 이것은 스탠드얼론 호스트 와 IP 네트워크 연결해서 다이얼 업 링크를 지원해 준다. 사실 후자도 스탠드얼로 호스트라 고 할 수 있다.

스탠드 얼론 호스트에서는 단독 네트워크 장치와 대개 주소가 **127.0.0.1**로 할당된 루프 백 장치를 활성화 시키는 일을 한다. 어떤 경우에는, 여러분이 로컬 호스트의 공식 IP 주소 로 데이터를 보낼 필요도 있다. 이를테면, *vlite*라고 하는 랩톱 컴퓨터가 있다고 가정 하자. 그것은 오랜동안 연결되어 있는 어떤 네트워크의 연결을 끊는 경우도 있다. vlite에 있는 어 플리케이션이 같은 호스트상에 있는 또 다른 어플리케이션으로 어떤 데이터를 보내고 싶어 할 지도 모른다. /etc/hosts에 있는 vlite가 **191.72.1.65**라는 IP 주소를 찾은 다음, 그 어플 리케이션은 이 주소로 데이터를 보내려고 시도할 것이다. 그 컴퓨터에서 활성화된 인터페이 스라고는, 루프백 인터페이스밖에 없으며, 실제로 커널은 이 주소가 그 인터페이스를 참조하 고 있는지는 알지 못한다. 결과적으로 볼 때, 커널은 그 데이터그램을 폐기처분하고 어플리 케이션으로 어떤 에러를 보내줄 것이다.

이러한 곳에 더미 디바이스가 필요하다. 이것은 단지 루프백 인터페이스를 변경시켜 줌 으로써 이러한 딜레마를 해결해 준다. vlite의 경우에, 여러분은 단순히 **191.72.1.65**라는 주소를 할당해 주고, 호스트의 라우트가 그 주소를 가리키도록 해 주기만 하면 된다.

**191.72.1.65**를 위한 모든 데이터그램은 지역적으로 전송될 것이다.

```
# ifconfig dummy vlite
# route add vlite
```

## 5.8 All About ifconfig

ifconfig에는 우리가 위에서 설명한 것보다 훨씬 더 많은 변수가 있다. 일반적으로 사용하는 옵션으로는 다음과 같은 것이 있다.

```
ifconfig interface [[-net | -host] address [parameters]]
```

interface는 인터페이스명 이고, address는 인터페이스로 할당된 IP 주소이다. dotted quad notation로 표기되어 있는 IP 주소나 그 이름은 ifconfig가 /etc/hosts와 /etc/networks 에서 찾을 것이다. -net와 -host 옵션은 ifconfig가 네트워크 번호나 호스트 주소를 개별적 인 주소로 다룰 때 사용한다.

만약 ifconfig가 단지 인터페이스 이름만을 가지고 있다면, 그것은 인터페이스의 구성환경 을 나타낼 것이다. 아무 변수 없이 ifconfig만을 입력하였을 때는, 여러분이 설정한 모든 인 터페이스를 나타낼 것이다; -a 옵션은 활동하고 있지 않은 인터페이스의 목록을 보여줄 것 이다. 이더넷 인터페이스인 eth0는 다음과 같이 보여질 것이다:

```
# ifconfig eth0
eth0     Link encap 10Mbps Ethernet HWaddr 00:00:C0:90:B3:42
         inet addr 191.72.1.2 Bcast 191.72.1.255 Mask 255.255.255.0
         UP BROADCAST RUNNING  MTU 1500  Metric 0
         RX packets 3136 errors 217 dropped 7 overrun 26
         TX packets 1752 errors 25 dropped 0 overrun 0
```

MTU와 Metric 필드는 현재 MTU와 인터페이스의 미터값 (metric value)을 보여준다. 미 터값 (metric value)은 전형적으로 라우트의 량을 계산하기 위해 몇몇 운영 체제에 의해서 사용되었다. 리눅스는 이러한 값을 사용하진 않지만, 호환성을 가지고 있기는 하다.

RX와 TX 라인은 얼마나 많은 패킷을 받고 있는지, 전송되었는지, 얼마나 많은 에러가 발 생했는지, 또는 메모리 부족으로 얼마나 많은 양의 패킷이 손실되었는지, 오버런으로 인해 얼마나 많은 피해가 있는지를 보여준다. 리시버 오버런 (receiver overrun)은 대개 커널이 인터럽트를 거는 속도보다 패킷이 더 빠르게 전송될 때 발생한다. 아래 설명은 ifconfig에 속해 있는 옵션을 보여주고 있으며, 각 옵션이 하는일이 무엇인가를 나타내 주고 있다.이러 한 옵션은 항상 ifconfig 다음에 (-) 대쉬를 붙여서 사용한다.

**UP**

이것은 인터페이스를 "up"하라는 표시이다. 즉, IP 층 (layer)로 접근가능하게 만들 때 사용한다. 이 옵션은 address가 명령어로 주어질 때 수행된다. 이것은 또한 인터페이스를 재사용할 때 쓰이며, 이것은 down 옵션을 일시적으로 사용가능하게 만들어 준다. (이 옵션은 UP RUNNING 플래그와 일치한다.)

**down**

이것은 인터페이스를 "down"하라는 표시이다. 즉, IP 층(layer)으로 접근하지 못하게 만들 때 사용한다. 이것은 실제로 그 인터페이스를 통해서 어떤 IP 트래픽을 사용 하지 못하게 만든다. 이것이 이 인터페이스를 자동으로 사용할 수 있게 해주는 모든 라우팅 엔트리들을 지워버리는 것이 아님을 기억해 두라. 만약 여러분이 그 인터페이스를 영원히 사용하지 못하게 만들어 버릴것이라면, 이러한 라우팅 엔트리들을 지워버림과 동시에, 경로를 지정해 주어야 한다.

**netmask mask**

이것은 인터페이스로 사용되고 있는 서브넷 마스를 할당해 준다. 이것은 0x와 같이 32비트 16진수로 표시하거나, 도트로 구분하는 네 개의 십진수로 표시한 다.

**pointopoint address**

이 옵션은 두 개의 호스트를 point-to-point IP 링크를 위해 사용된다. 예를 들어 SLIP 또는 PLIP 인터페이스를 구성할 때 이 옵션이 필요 하다. (만약 point-to-point 주소가 설정되어 있다면, ifconfig는 POINTOPOINT 플래그를 표시해 줄 것이다.)

**broadcast address**

브로드캐스트 주소는 대개 호스트 부분의 모든 비트를 설정함으로 써, 네트워크 번호를 구성한다. 몇몇 IP implementation들은 다른 스키마를 사용한다; 이 옵션 은 이러한 이상한 환경을 사용할 수 있게 해준다. (만약 브로드캐스트 주소가 설정되어 있다면, ifconfig는 BROADCAST 플래그를 표시 해 줄 것이다.)

**metric number**

이 옵션은 인터페이스가 만들어진 라우팅 테이블 엔트리의 미터값을 할당하는데에 사용될지도 모른다. 이 metric는 네트워크를 위한 라우팅 테이블을 만들기 위해 *Routing Information Protocol* (RIP)에 의해 사용된다. ifconfig에 사용되는 디폴트 미터값은 0이다. 만약 여러분이 RIP 데몬을 실행하지 않고 있다면, 이 옵션을 사용할 필요는 없다; 만약 RIP 데몬을 실행시켰다면, 이 미터값을 변경 시킬 필요는 거의 없다.

**mtu bytes**

이것은 *Maximum Transmission Unit*, 즉 인터페이스가 하나 의 트랜잭션에서 처리할 수 있는 최대 옥텟수를 설정할 때 사용한다. 이더넷에서 MTU 디폴트값은 1500이며, SLIP 인터페이스에서는 296이 된다.

**arp**

이것은 이더넷이나 패킷 라디오와 같은 브로드캐스트 네트워크를 명시하는데에 사용 하는 옵션이다. 이것은 또한 호스트의 물리 주소가 네트워크로 접근하는 것을 감지해내기 위해 사용되는 ARP, *Address Resolution Protocol*을 사용할 수 있게 해준다. 브로드 캐스트상에서는 디폴트로 설정되어 있다. (ARP를 사용하고 있지 않다면, ifconfig는 NOARP라고 표시해 줄 것이다.)

**-arp**

인터페이스에서 ARP사용을 할 수 없게 해 주는 옵션이다.

**promisc**

promiscuous 모드로 인터페이스를 설정해준다. 브로드캐스트 네트워크상에서, 이것은 패킷이 다른 호스트에 묶여 있음에도 불구하고, 모든 패킷을 받아 주는 인터페이스 를 만들어 준다. 이것은 또한 네트워크 트래픽이 Ethernet snooping와 같은 패킷 필터를 사용할 수 있게끔 만들어 준다. 대개 이 옵션은 네트워크의 문제를 해결하는 좋은 기술이다 다른 한편으로, 이것은 칩입자들이 여러분의 패스워드를 알아내기 위해 네트워크 트래픽을 넘기거나 다른 성가신 일을 하게 만들 수도 있다. 이러한 칩입에 대항하는 한 방편으로는 여러분의 컴퓨터로 칩입자들이 직접 들어올 수 없게끔 하는 것이다. Kerberos와 SRA와 같은 인증 프로토콜을 사용하는 것이다. (이 옵션은 PROMISC와 일치한다.)

**-promisc**

promiscuous 모드를 꺼 놓는다.

**allmulti**

멀티캐스트 주소는 같은 서브넷에 있을 필요가 없는 호스트 그룹을 브로드캐스트한다. 멀티캐스트 주소는 아직 커널에서 지원하지는 않는다. ( 이 옵션은 ALLMULTI 플래그와 일치한다.)

**-allmulti**

멀티캐스트 주소를 사용하지 않게 한다.

## 5.9 Checking with netstat

다음으로, 나는 여러분의 네트워크 환경을 검사하고 활성화 시킬 때 유용하게 사용하는 도 구를 설명할 것이다. 이것은 netstat라고 부르며, 사실 여러 가지 도구와 함께 사용한다. 그 도구의 각 기능들은 다음절에서 설명하겠다.

### 5.9.1 Displaying the Routing Table

-r 플래그와 netstat를 같이 사용하게 되면, 위에서 route를 설명할 때와 마찬가지로 커널의 라우팅 테이블을 표시해 준다. vstout에서는 다음과 같이 나타난다:

```
# netstat -nr
Kernel routing table
Destination     Gateway     Genmask          Flags  Metric Ref Use  Iface
127.0.0.1       *           255.255.255.255  UH     1      0    50  lo
191.72.1.0      *           255.255.255.0    U      1      0   478  eth0
191.72.2.0      *           255.255.255.0    UGN    1      0   250  eth0
```

*-n* 옵션은 netstat가 심볼릭 호스트와 네트워크 이름대신에 도트로 구분된 네 개의 IP 숫자로 주소를 표시하게끔 해준다. 이것은 여러분이 네트워크를 통해서 주소를 찾는 작업을 피하고 싶을 때 유용하게 사용된다. (예를 들어, DNS 또는 NIS 서버)

netstat의 출력에서 두 번째 칼럼은 게이트웨이가 라우팅 엔트리를 가리키고 있는지를 보여준다. 만약 게이트웨이를 사용하고 있지 않다면, 위와 같이 아스트릭 문자 (*)가 표시된 다. 그 다음 세 개의 칼럼은 라우트의 "일반성(generality)"를 보여준다. 주어진 IP 주소가 그와 적합한 라우트를 발견했을 때, 커널은 모든 라우팅 테이블 엔트리를 거쳐서, genmask 와 목적 라우트를 AND 연산자로 비교한다.

네 번째 칼럼은 아래와 같이 여러 가지 플래그 표시해 준다:

**G**

라우트가 게이트웨이를 사용한다.

**U**

인터페이스가 사용되고 있다.

**H**

오직 단독 호스트만이 라우트를 거쳐서 접근할 수 있다. 예를들어, 이러한 경우의 루프백 엔트리는 **127.0.0.1**이다.

**D**

테이블 엔트리가 설정된 경우, ICMP 리다이렉트 메시지에 의해 운영되고 있다.

**M**

테이블 에트리가 설정된 경우, ICMP 리다이렉트 메시지에 의해 수정되고 있다.

netstat 출력에서 Ref 칼럼은 이 라우트를 참조하는 번호를 나타낸다. 즉, 얼마나 많은 라우트가 이 라우트에 의존하고 있는지를 나타낸다. 마지막 두 칼럼은 라우팅 엔트리가 사 용되었는지, 얼마나 많은 데이터 그램이 인터페이스로 전송되었는지를 나타내준다.

### 5.9.2 Displaying Interface Statistics

*-i* 플래그와 netstat를 함께 사용하면, 현재 구성되어 있는 네트워크 인터페이스의 상태를 보여준다. 거기에 다가 *-a* 플래그를 주게 되면, 커널에 존재하는 것 뿐만 아니라, 현재 구성 되어 있는 모든 인터페이스를 보여 줄 것이다. vstout에서, netstat의 출력은 다음과 같다:

```
    $ netstat -i
    Kernel Interface table
    Iface  Mtu  Met    RX-OK RX-ERR   RX-DRP RX-OVR  TX-OK   TX-ERR
TX-DRP TX-OVR Flags
    lo       0   0     3185      0       0       0    3185      0       0       0 BLRU
    eth0  1500   0   972633     17      20     120  628711    217       0       0 BRU
```

MTU와 Met 필드는 인터페이스의 현재 MTU와 미터값 (metric value)을 보여준다. RX 와 TX 칼럼은 얼마나 많은 패킷과 에러가 전송되고 보내졌는지 (RX-OK/TX-OK), 그리고 손상을 입었는지 (RX-ERR/TX-ERR), 얼마나 많은 양의 패킷이 감소되었는지 (RX-DRP/TX-DRP), 오버런 으로 인해 손실된 양은 얼마나 되는지 (RX-OVR/TX-OVR)를 나타내 준다.

마지막 칼럼은 이러한 인터페이스가 어떻게 설정되었는지를 나타내주는 플래그이다. 이 러한 형태의 긴형태의 플래그 이름은 여러분이 ifconfig로 인터페이스 구성환경을 잡아줄 때 출력된다.

**B**

브로드캐스트 주소가 설정되어 있다.

**L**

이 인터페이스는 루트백 인터페이스이다.

**M**

모든 패킷이 전송되고 있다. (promiscuous 모드)

**N**

Trailer은 피한다.

**O**

이 인터페이스를 위한 ARP가 꺼져 있다.

**P**

이것은 point-to-point 연결이다.

**R**

인터페이스가 실행되고 있다.

**U**

인터페이스가 up상태임

### 5.9.3 Displaying Connections

netstat는 활동하고 있는 소켓을 표시해 주기 위한 옵션을 가지고 있다. *-t, -u, -w* 그리고, *-x* 옵션은 활동중인 TCP, UDP, RAW 또는 UNIX 소켓 연결을 보여준다. 여기에 *-a* 옵션 을 추가한다면, 현재 연결을 기다리는 소켓을 표시해 준다. 현재 여러분의 시스템에서 실행 되고 있는 모든 서버의 목록을 보여 줄 것이다.

vlager에서 netstat -ta는 다음과 같은 화면을 출력한다.

```
$ netstat -ta
Active Internet connections
Proto  Recv-Q  Send-Q  Local Address    Foreign Address     (State)
tcp         0       0  *:domain         *:*                 LISTEN
tcp         0       0  *:time           *:*                 LISTEN
tcp         0       0  *:smtp           *:*                 LISTEN
tcp         0       0  vlager:smtp      vbardolino:1040     ESTABLISHED
tcp         0       0  *:telnet         *:*                 LISTEN
tcp         0       0  localhost:1046   vbardolino:telnet   ESTABLISHED
tcp         0       0  *:chargen        *:*                 LISTEN
tcp         0       0  *:daytime        *:*                 LISTEN
tcp         0       0  *:discard        *:*                 LISTEN
tcp         0       0  *:echo           *:*                 LISTEN
tcp         0       0  *:shell          *:*                 LISTEN
tcp         0       0  *:login          *:*                 LISTEN
```

이것은 대개 연결을 기다리는 모든 서버를 보여준다. 하지만 네 번째 라인은 vstout에서 들어오는 SMTP연결을 보여준다. 그리고 여섯 번째 라인은 vbardolino로 telnet을 이용한 외부연결이 있음을 나타낸다.

*-a* 플래그를 사용하면, 모든 집단의 모든 소켓을 보여준다.

## 5.10 Checking the ARP Tables

어떤 경우에는 커널의 ARP 테이블의 내용을 보거나 변경시키는 것이 유용할 때도 있다. 예 를 들어, 여러분이 똑 같은 인터넷 주소가 하나 더 있다고 의심하는 경우, 복잡한 네트워크 문제를 발생시킬 수도 있다. 이러한 문제를 해결하기 위해 만들어진 것이 바로 arp이다. 명 령행에서 옵션은 다음과 같이 쓰인다.

```
arp [-v] [-t hwtype] -a [hostname]
arp [-v] [-t hwtype] -a hostname hwaddr
arp [-v] -d hostname [hostname...]
```

모든 hostname 변수는 심볼릭 호스트 네임이나 dotted quad notation으로 표기된 IP 주 소를 말하는 것이다.

첫 번째 명령행은 만약 그것이 no hostname으로 주어졌다면, 알려진 모든 호스트와 IP 주소 그리고 특별한 호스트의 ARP 엔트리를 보여준다. 예를 들어, vlager에서 arp를 사용 하게 되면 다음과 같은 출력이 나타난다.

```
# arp -a
IP address       HW type               HW address
191.72.1.3       10Mbps Ethernet       00:00:C0:5A:42:C1
191.72.1.2       10Mbps Ethernet       00:00:C0:90:B3:42
191.72.2.4       10Mbps Ethernet       00:00:C0:04:69:AA
```

vlager, vstout 그리고 vale의 이더넷 주소를 보여주고 있다.

*-t* 옵션을 사용하면, 특별한 형태의 하드웨어 출력을 제한할 수 있다. 이것은 각각 ether, ax25, 또는 pronet, 10Mbps 이더넷을 기본으로 하고있는 하드웨어, AMPR AX.25, 그 리고 IEEE 802.5 token ring 방식의 하드웨어가 될 수도 있다.

*-s* 옵션은 ARP 테이블에 hostname의 이더넷 주소를 영구히 추가시키고자 할 때 사 용한 다. hwaddr 변수는 하드웨어 주소를 명시한다. 기본적으로는 이더넷 주소를 나타낸다. 그리 고 이것은 각각 콜론 (:)으로 구별되어 있는 여섯 개의 16진수로 구성되어 있다. 여러분은 어쩌면 *-t* 옵션을 사용해서, 다른 형태의 하드웨어 주소를 설정할 수도 있다.

원격 호스트가 ARP 질의를 거부하는 경우에는, ARP 테이블에 IP 주소를 수동으로 잡아 주라는 메시지가 뜬다. 이러한 현상이 발생하는 원인이라면, ARP 드라이버에 버그가 발생 했다던지, 호스트의 IP 주소를 잘못 인식한 네트워크에 또 다른 호스트가 있을 경우 이러한 문제가 발생한다. ARP 테이블에 있는 hard-wiring IP 주소는 여러분의 이더넷 상에서 여러 분의 호스트를 보호할 수 있는 도구이다.

*-d* 스위치와 함께 arp를 사용하게 되면, 주어진 호스트와 연관되어 있는 모든 ARP 엔트 리들을 삭제해 버린다. 이것은 인터페이스로 하여금 문제시 되고 있는 IP 주소에 대한 이더 넷 주소를 가지게끔 하기위해 강제로 재시도 하는데에 사용되기도 한다. 이것은 또한 잘못 구성되어 있는 시스템이 잘못된 ARP 정보를 브로드캐스트하는데에도 유용하게 쓰인다. (물 론 이러한 작업을 하기 전에, 여러분이 깨진 호스트를 재구성해야 한다.)

*-s* 옵션은 proxy ARP를 구현하는데에도 사용된다. 이것은 gate라고 하는 호스트 를 fnord라고 하는 또 다른 호스트 게이트웨이로 작동하도록 만들어 주는 기술로써, 두 개의 주소가 이름하여 gate라고 하는 같은 호스트를 참조하도록 만들어 준다. 즉, 그것은 그 자 체의 이더넷 인터페이스를 가리키는 fnord를 위한 ARP 엔트리를 사용함으로써 그렇게 할 수 있다. 호스트가 fnord를 위한 ARP 질의를 보내고자 할 때, gate는 이더넷 주소를 포함 하고 있는 응답을 되돌려 줄 것이다. 질의를 하고 있는 호스트가 gate로 모든 데이터그램을 보내 고자 할 때에는 의무적으로 fnord에 그 자료들을 전송할 것이다.

이를테면, 여러분이 TCP도 구현하지 못하고, 라우팅도 그다지 이해하지 못하는 DOS 머 신에서 fnord로 엑세스하고자 할 때에는 이러한 곡예도 필요하다. 여러분이 proxy ARP를 사용한다면, 마치 fnord가 로컬 서브넷에 있는 것처럼, 여러분이 DOS 머신에 접속한 것처 럼 보일 것이다. 그래서, 게이트웨이를 통해 라우트를 하는 방법은 알필요가 없다.

proxy ARP에서는 매우 유용하게 사용할 수 있는 또 다른 어플리케이션이 있다. 즉, 다 이얼 업 링크를 사용해서, 여러분의 호스트를 일시적으로 게이트웨이처럼 동작하게 만들어 주는 것이다. 이전에, 우리는 이따금 PLIP 링크를 거쳐서, vlager에 연결되어 있는 랩톱 vlite를 보았다. 물론 여러분이 proxy ARP를 제공하고자 하는 호스트의 주소는 게이트웨이 에 있는 같은 서브넷 상에서 동작할 것이다. 이를테면, proxy ARP를 사용하고 있는 vstout 는 Brewery 서브넷 (191.72.1.0)에서는 호스트가 될 수 있지만, Winery 서브넷 (191.72.2.0) 에서는 절대로 호스트가 될 수 없다.

fnord에게 proxy ARP를 제공하는 적절한 방법은 다음과 같다; 물론 gate는 이더넷 주 소를 가지고 있어야 한다.

```
# arp -s fnord 00:00:c0:a1:42:e0 pub
```

다음과 같은 방법으로 proxy ARP 엔트리를 제거할 수도 있다.

```
# arp -d fnord
```

### 5.11 The Future

리눅스 네트워킹은 여전히 진화하고 있다. 커널에서 주요 변화라고 한다면, 구성환경을 전보 다 매우 유연하게 변경시킬 수 있다는 것이다. 즉, 커널은 여러분이 실행시간에 네트워크 장 치를 구성하게 해준다. 이를 테면, ifconfig 명령은 IRQ와 DMA 채널과 같은 변수를 설정 해준다.

또 다른 변화라고 한다면, route 명령에 mtu 플래그를 추가 시킨 점이다. 이 명령으로 특 별한 라우트를 위해 최대 전송 단위 (Maximum Transmission Unit)를 설정해 줄 수 있다. MTU가 설정된 라우트는 인터페이스에 명시되어 있는 MTU를 무효화 시킬 수 있다. 여러 분은 전형적으로 게이트웨이와 매우 낮은 MTU를 필요로 하는 목적 호스트를 연결하고 있 는, 게이트웨이를 통해 라우트를 사용할때에는, 이 옵션을 사용할 것이다. 예를 들어, 호스트 wanderer이 SLIP 링크를 통해서 vlager에 연결되어 있다고 가정하자. vstout에서 wanderer 로 데이터를 보내고자 할 때, wanderer에 있는 네트워킹 층 (layer)은 패킷들이 이더넷을 거 쳐서 보내지기 때문에, 최고 1500 바이트 패킷을 사용할 것이다. 한편, SLIP 링크는 296 바 이트 MTU로 운영되어야 하고, vlager의 네트워크 층은 IP 패킷들을 296 바이트씩 쪼개어 서 보내야 한다. 대신에 여러분이 vstout에서 라우트를 설정할 때, 시작시 296 바이트 MTU 를 사용하게끔 설정해 놓았다면, 상대적으로 조각을 나눌 때 드는 비용을 줄일 수 있다.

```
# route add wanderer gw vlager mtu 296
```

여러분이 직접 설정할 수 있는 mtu 옵션또한 'Subnet Are Local' 정책 (SNARL)의 결 과로 취소되었다는 것을 명심하라. 이 정책은 커널 환경 구성 옵션에도 영향을 주었으며, 3 장에서 설명했었다.

## 6 Name Service and Resolver Configuration

2장에서 설명한 바대로, TCP/IP 네트워킹은 호스트네임을 IP 주소를 변환하기 위한 여러 가지 스키마에 의존하고 있다. 가장 간단한 방법으로 /etc/hosts에 저장되어 있는 호스트 테 이블의 이름구역을 여러 지역으로 쪼개는 방법은 아무런 이득을 가져다 주지는 못한다. 이 러한 방법은 관리자 한사람에 의해 운영되며, 외부세계와 아무런 IP 트래픽이 발생하지 않 는 규모가 작은 LAN에서는 유용하다. hosts 파일의 형식은 이미 5장에서 설명하였다.

다른 방법으로, 여러분은 호스트네임을 IP 주소로 변환시킬 때 사용되는 BIND - *Berkeley Internet Name Domain Service*를 사용할 수도 있다. BIND를 구성하는 작 업은 정 말 따분한 일이지만, 네트워크 토폴로지를 쉽게 만들려면, 한번은 해야할 작업이다. 리눅스 나 또 다른 유닉스 계열 시스템에서, 네임 서비스는 named라는 프로그램을 통해 제공된다. 시동시, 이것은 마스터 파일들을 그 자체의 저장소(cache)에 적재하고, 리모트 또는 로컬 사 용자 프로세스에서 질의를 기다린다. BIND를 설정하는 방법은 여러 가지가 있지만, 모든 호스트에 네임 서버를 설정할 필요는 없다.

이 장에서는 네임 서버 운영에 관한 기본지식만 다룰 생각이다. 만약 여러분이 작은 LAN이상의 환경이나, 인터넷상에서 BIND를 사용할 계획이라면, 예를 들어, Cricket Liu의 "DNS and BIND" ([AlbitzLiu92]를 참조하라.)와 같이, 더 좋은 책을 읽어 보아야 할 것이 다. 이러한 정보를 위해서, 여러분은 BIND 소스에 포함되어 있는 release notes를 확인해 볼수도 있다. 또한 *comp.protocols.tcp-ip.domains*이라고 하는 DNS 뉴스 그룹도 있다.

### 6.1 The Resolver Library

"the resolver"은 특별한 어플리케이션이 아니라 "resolver library"를 지칭하는 것이다. 이것 은 C 라이브러리에 본 바탕을 두고 있는 기능의 모음집이다. 중심이 되는 루틴으로는 호스 트에 속해 있는 모든 IP 주소를 찾거나 IP 주소에 있는 모든 호스트를 찾아주는 gethostbyname(2)와 gethostbyaddr(2)를 들 수 있다. 이것들은 단순히 hosts에 있는 정보를 찾거나, 네임 서버의 네임을 질의하거나, NIS (Network Information Service)의

hosts 데이 터베이스를 사용하도록 구성되어 있다. smail과 같은 어플리케이션은 이러한 것들을 위한 드라이버를 포함할 수도 있으며, 이것은 특별한 경우에 필요하다.

### 6.1.1 The host.conf File

여러분의 resolver 셋업을 제어하는 것이 바로 host.conf 파일이다. 이것은 /etc 디렉토리 에 있고, resolver가 사용할 서비스를 말해 주며, 그러한 서비스들은 순서대로 나열되어 있다.

host.conf에 있는 옵션들은 각각 독립된 한 라인에 존재한다. 필드는 스페이스나 탭으로 구분되어 있다. 해쉬 표시 (#)가 되어 있는 라인은 그 다음에 나올 각 옵션에 대해 짧막한 설명을 해주는 부분이다.

다음과 같은 옵션이 있다:

**order**

이것은 resolving service가 처리되는 순서를 결정한다. 이와 함께 사용되는 옵션으로는 bind, hosts, nis가 있는데, 각각이 하는 일은 네임 서버에게 질의를 한다든지, /etc/hosts에서 정보를 찾는다든지, NIS에서 필요한 정보를 찾는 역할을 한다. 이러한 것들 중 몇 개 혹은 전부를 명시할 수도 있다. 라인에 나타나는 순서는 각 서비스가 처리되는 순서 를 의미한다.

**multi**

옵션을 사용할 수 있게 또는 사용할 수 없게 만든다. /etc/hosts에 있는 하나의 호스트가 여러개의 IP 주소를 가지게끔 할려면, 대개 "multihomed"를 사용한다. 이 플래그는 DNS나 NIS 질의에 아무런 영향을 끼치지 않는다.

**nospoof**

5장에서 설명한 바대로, 여러분이 DNS는 in-addr.arpa 도메인을 사용해서, IP 주소에 해당하는 호스트 네임을 찾게 해준다. 네임 서버에 의해 잘못된 호스트 네임을 제공하는 것을 "spoofing"라고 한다. 이러한 점을 막기 위해서, resolver는 오리지널 IP 주소가 호스트네임과 연관되어 있는지를 검사하도록 구성되어 있다. 그렇지 않다면, 호스트네임은 어떤 에러를 발생시킬 것이다. 이러한 작업을 위해서는 nospoof로 설정해 놓아야 한다.

**alert**

이 옵션은 변수를 사용할 수 있게 하거나 사용할 수 없게 만든다. 이 옵션을 on 시켜 놓으면, spoof 시도 (attempt)는 resolver가 syslog에 메시지를 저장하도록 만들 것이 다.

**trim**

이 옵션은 도메인 네임을 변수로 설정한다. 즉, 도메인 네임은 룩업과정이 일어 나기 전에 호스트네임에서 삭제될 것이다. 이것은 hosts 엔트리를 사용하고자 할때 유용하게 쓰 인다. hosts 엔트리는 여러분이 로컬 도메인 없이 호스트네임을 명시하고자 할 때, 사용되는 것 이다. 호스트에 추가적으로 붙어 있는 로컬 도메인 네임의 룩업과정은 삭제되고, /etc/hosts에서 이러한 작업이 진행될 것이다.

**trim**

옵션은 여러분의 호스트를 여러 로컬 도메인으로 간주하게끔 만들어 준다.

다음은 vlager에 대한 예제파일이다;

```
# /etc/host.conf
# We have named running, but no NIS (yet)
order   bind hosts
# Allow multiple addrs
multi on
# Guard against spoof attempts
```

```
nospoof on
# Tirm local domain (not really necessary).
trim   vbrew.com.
```

### 6.1.2 Resolver Environment Variables

host.conf에서 설정하는 부분을 무시해 버리는 여러 가지 환경변수가 있다.

**RESOLV_HOST_CONF**

이것은 /etc/host.conf 대신에 읽어 들일 파일을 명시한다.

**RESOLV_SERV_ORDER**

host.conf에 주어진 순서를 무시해 버린다. hosts, bind 그리고 nis에서 주어지는 서비스들은 스페이스, 콤마, 콜론 또는 세미 콜론으로 구분되어 있 다.

**RESOLV_SPOOF_CHECK**

주어진 spoofing를 측정할건지를 결정한다. 완전히 사용할 수 없게 할려면, 그 뒤에 off를 붙여라. spoof 검사를 가능하도록 만들어 주는 warn과 warn off는 각각 로깅 온 (logging on)과 로깅 오프 (logging off)를 한다. * 변수는 spoof를 체크하겠다는 의미이지만, host.conf에 규정된 대로, 로깅을 하지는 않는다.

**RESOLV_MULTI**

on 또는 off라는 변수는 host.conf에서 multi 옵션을 무시해 버릴 때 사용할 수도 있다.

**RESOLV_OVERRIDE_TRIM_DOMAINS**

이 환경은 host.conf를 무시해 버리는 트림 도메인 목록을 명시한다.

**RESOLV_ADD_TRIM_DOMAINS**

이 환경은 host.conf에 추가된 트림 도메인을 명시 한다.

### 6.1.3 Configuring Name Server Lookups – resolv.conf

여러분이 호스트 룩업을 위한 BIND 네임 서비스를 사용하기 위해서, resolver library를 구 성하려고 한다면, 사용할려는 네임서버를 말해 주어야 한다. 이러한 작업을 하기 위해서는 resolv.conf를 편집해야 한다. 이 파일이 없거나 파일안이 텅비어 있다면, resolver는 네임 서 버가 여러분의 로컬 호스트에 있다고 가정해 버린다.

만약 여러분의 로컬 호스트에서 네임서버를 실행하려고 한다면, 독립적으로 설정해 주어 야 한다. 그러나, 로컬 네트워크에 네임서버가 존재하고 있다면, 이것을 사용하는 것이 훨씬 더 경제적이다.

resolv.conf에서 가장 중요한 옵션은 nameserver이다. 이것은 사용할 네임서버에게 IP 주 소를 할당해 주는 일을 한다. 만약 여러분이 nameserver 옵션을 사용해서 여러 가지 네임 서버를 명시하고자 한다면, 그것들은 주어진 순서대로 처리된다. 먼저, 가장 믿을 만한 서버 를 택하는 것이 좋다. 현재, 가질 수 있는 네임서버의 수는 세 개다.

no nameserver가 주어진다면, resolver는 로컬 호스트에 있는 네임서버로 연결하려고 할 것이다.

domain과 search 옵션은 둘다 디폴트 도메인 설정시에 사용된다. 즉, BIND에서 첫 번째 질의가 실패했다면, 이러한 옵션들은 호스트네임에 덧붙혀진다. search 옵션은 접속을 시도 할려는 도메인의 목록을 명시한다. 각 항목들은 스페이스나 탭으로 구분되어 있다.

no search 옵션이 주어진다면, 그 자체의 도메인 네임을 사용해서, 로컬 도메인 네임으로 부터 디폴트 서치 리스트가 만들어 지며, 최고 루트까지 부모 도메인이 추가된다. 로컬 도메 인 네임은 domain 문장을 사용해서 만들 수도 있다; 만약 아무것도 주지 않는다면, resolver는 getdomainname(2) 시스템 콜을 사용해서 도메인 네임을 구할 것이다.

지금 이러한 설명이 조금 복잡하게 들린다면, Virtual Brewery에서 resolv.conf파일을 사용 하는 예를 생각해 보자:

```
# /etc/resolv.conf
# Our domain
domain          vbrew.com
#
# We use vlager as central nameserver:
nameserver      191.72.1.1
```

vale라는 이름을 resolv하려고 할 때, resolver는 vale.vbrew.com과 vale.com과 같이 vale 를 사용하는 이름을 모두 찾을 것이다.

### 6.1.4 Resolver Robustness

만약 여러분이 거대한 네트워크에서 LAN을 구현하려고 한다면, 중앙 네임 서버를 사용해야 한다. 이것을 사용하는 이점이라면, 모든 질의가 그 저장소 (cache)로 들어가기 때문에 많은 저장소를 만들어 낼 수 있다는 것이다. 하지만 이러한 스키마에도 약점은 있다: 대학의 백본 망이 파괴되었을 때, 각각의 LAN에서는 아무런 작업도 할 수 없다. 왜냐하면, resolver가 더 이상 네임서버에 도달 할 수 없기 때문이다. 그리고 X 터미널에도 접속할 수 없고, 프린 터도 사용할 수 없게 된다.

캠퍼스 백본이 파과되는 일이 매우 드문 경우이지만, 이러한 경우를 대비해서 예방조치 를 취해 두는 것도 좋은 방법이다.

로컬 네임서버를 설정하기 위한 한가지 옵션으로는 여러분의 로컬 네임서버에서 호스트 네임을 resolv하라. 그리고 다른 호스트네임을 위한 모든 질의를 메인 서버로 향하게 하라. 만약 여러분 자체의 도메인을 실행하고 있다면, 이것이 적절한 방법이 될 것이다.

다른 방법으로, /etc/hosts에 있는 여러분의 도메인이나 LAN을 위한 백업 호스트 테이 블을 유지할 수 있다. 만약 중앙 네임 서버가 다운되는 경우를 대비해서, resolver가 호스트 파일을 가리키지 않도록 할려면, /etc/host.conf에 "order bind hosts"를 추가시켜라.

## 6.2 Running named

대부분의 유닉스 계열 시스템에서 도메인 네임 서비스를 제공해 주는 프로그램은 named (대개 name-dee라고 발음한다.)이다. 이것은 원래 BSD에서 개발되었으며, 클라이언트에게 네임서비스를 제공해 주고, 또 다른 네임서버를 사용할 수 있게끔 해준다. 현재 대부분의 리 눅스 버전에서 사용되고 있는 버전은 BIND-4.8.3이다. 최근 버전인 BIND-4.9.3은 아직 베타 테스트 중이고, 가까운 시일내에 리눅스에서도 사용할 수 있을 것이다.

이 절은 Domain Name System이 어떻게 작동되는지를 이해시키는 부분이다. 만약 이해 할 수 없는 부분이 나온다면, 2장을 다시 한번 읽어보기 바란다. 그 장은 DNS에 관한 기본 적인 정보를 설명해 놓고 있다.

named는 대개 시스템이 부팅될 때, 시작되며, 시스템이 셧다운 되기 전까지 작동한다. /etc/named.boot 라는 구성파일에서 이러한 정보를 알 수 있으며, 이 파일에는 도메인 네임 을 주소에 대응시킬 때 사용하는 zone 이란 파일도 포함되어 있다. 이러한 파일의 형식과 의미는 다음절에 설명되어 있다.

named를 실행시키기 위해서는, 프롬프트에서 단순히 다음과 같이 하라.

```
# /usr/sbin/named
```

named는 named.boot와 그 가운데 명시되어 있는 zone 파일을 읽고나서, 실행될 것이다. 그것의 프로세스 id는 ASCII형태로 /var/run/named.pid에 쓰여져 있으며, 필요하다면, 프라 이머리 서버로부터 전송받을 수도 있으며, DNS 질의를 위한 포트 53에서 리스닝할 수도 있 다.

### 6.2.1 The named.boot File

named.boot파일의 크기는 대개 매구 작고, 포함되어 있는 정보또한 그리 많진 않지만, zone 에 관한 정보를 포함하고 있는 마스터 파일과 또 다른 네임 서버를 가리키고 있다. 부트 파 일에서 세미 콜론으로 시작하는 문법은 다음 라인으로 연장한다는 의미이다. 자세한 정보를 위해서 named.boot의 형식을 논하기 전에, 그림 6.1에서 주어진 vlager을 위한 예제 파일을 먼저 살펴보자.

```
;
; /etc/named.boot file for vlager.vbrew.com
;
directory       /var/named
;
;            domain                 file
;-----------------------------------------------
cache        .                      named.ca
primary      vbrew.com              named.hosts
primary      0.0.127.in-addr.arpa   named.local
primary      72.191.in-addr.arpa    named.rev
```

그림 6.1: vlager를 위한 named.boot파일

이 예제에서 cache와 primary는 named에 정보를 적재시키는 명령이다. 이 정보는 두 번째 문장에 명시되어 있는 마스터 파일로부터 읽어 들인다. 그것들은 텍스트 형식으로 되 어 있는 DNS 자원 레코를 포함하고 있으며, 다음에 볼 것이다.

이 예제에서, 우리는 세 개의 도메인을 갖도록 named를 구성하였다. 이를테면, 이들 중 첫 번째 라인은 프라이머리 네임을 vbrew.com으로 활동하도록 named에게 통보했으며, 이 것은 named.hosts 파일에서 zone 데이터를 읽어 들인다. directory 라는 키워드는 모든 zone 파일이 /var/named에 위치하고 있다는 것을 말해준다.

cache는 매우 특별한 것으로써, 네임서버를 실행하고 있는 모든 기계를 가상적으로 표현 한다. 이것은 named가 그 자체의 저장소와 named.ca와 같은 저장(cache)파일로부터 root name server hints를 사용할 수 있게끔 해준다. name server hint에 대해서는 다음에 설명 하겠다.

다음은 여러분이 named.boot에서 사용할 수 있는 가장 중요한 옵션의 목록들이다.

**directory**

이것은 zone 파일이 존재하고 있는 디렉토리를 명시한다. 파일들의 이름이 이 디렉토리와 연관되어서 주어진다. 여러 가지 디렉토리는 directory를 여러차례 사용함으로써 명시할 수 있다. 표준 리눅스 파일시스템에서는 /var/named가 되어야 한다.

**primary**

이것은 변수로써 domain name과 file name을 사용한다. named 도메인을 위 해 서는 믿을 만한 로컬 서버를 사용하라. 프라이머리 서버에서, named는 주어진 마스터 파일로 부터 zone 정보를 적재시킨다. 일반적으로, 모든 부트 파일에는 적어도 하나의 primary 엔트리가 존재할 것이다. 즉, **127.0.0.0** 네트워크를 변환시키면, 로컬 루프백 네트워크가 될 것이다.

**secondary**

이것은 변수로써, domain name와 address list 그리고 file name을 사용한 다. 로컬 서버를 명시된 도메인을 위한 두 번째 마스터 서버로 변환시켜 놓는다. 두 번째 서버는 도메인에 있는 믿을 만한 데이터를 가지고 있지만, 파일에서 자료를 가지고 오 진 못한다. 하지만 프라이머리 서버로부터 자료를 전송받을려고 할 것이다. 프라이머리 서버에 있는 IP 주소중 적어도 하나는 named로 주어져야 한다. 로컬 서버는 데이터를 zone 데이터베이스에 성공적으로 전송할 때까지, 각주소에 접속할 것이며, 세 번째 변수로 주어진 백업 파일에 저장되어 있다. 만약 어떤 프라이머리 서버도 응답하지 않는다면, zone 데이터는 대신

에 백업파일에서 그 정보를 검색할 것이다. named는 규칙적인 간격으로 zone 데이터를 리프레시할 것이다. 이것은 이전에 SOA 자원 레코드 형태로 연결되었을 때 설명한 적이 있다.

**cache**

이것은 domain name과 file name을 변수로써 사용한다. 이 파일은 root server hint를 포함하고 있으며, 모든 레코드 목록이 로트 네임서버를 가리키도록 되어 있다. 오직 NS와 A레코드가 인식될 것이다. domain 변수는 일반적으로 루트 도메인 네임을 지칭하는 것이다. 이 정보는 named에서 절대적인 것이다: 만약 cache 문이 부트 파일에서 발생하지 않았다면, named는 더 이상 로컬 저장소를 개발하지 않는다. 만약 질의를 받은 다음 서버가 로컬 네트워크에 존재하고 있지 않다면, 이것은 그러한 수행작업을 중단시켜 버릴 것이고, 네트워크 적재 작업을 심하게 증가 시킬 것이다. 게다가 named는 어떤 루트 네임 서버에도 도달할 수 없을 것이고, 그리하여, 믿을 만한 것을 제외하고는 어떤 주소도 해결 (resolve)하지 못할 것이다. 이러한 규칙에서 예외가 있다면, 그것은 전송중인 서버를 사용할 때 뿐일 것이다. (아래에 있는 forwarders 옵션)

**forwarders**

이것은 변수로써 address list를 사용한다. 이 목록에 있는 IP 주소들은 만약 로컬 저장소에서 질의하는 과정이 실패로 끝이 났다면, named가 질의할 수 있 는 네임 서버의 목록을 명시한다. 이것들은 질의에 응답하는 것이 하나라도 있을 때 까지 이러한 작업을 계속한다.

**slave**

이것은 네임 서버를 slave 서버로 만들어 준다. 그 자체내에서는 질의를 수행할 수 없지만, forwarders 문을 써서, 명시된 서버로 질의를 향하게 끔 만 든다.

여기에는 기술되어 있진 않지만, sortlist와 domain과 같은 옵션이 더 있다. 추가적으로 zone 데이터 파일에서 사용되는 두 개의 지시기가 있다. 그것들은 $INCLUDE와 $ORIGIN. 이다. 이것들이 거의 필요로 하지 않은 관계로 여기서는 설명하지 않았다.

### 6.2.2 The DNS Database Files

named.hosts와 같이 named에 의해 포함되어 있는 마스터 파일은 항상 origin이라고 부르는 것과 연관되어 있는 도메인을 가지고 있다. origin은 cache와 primary 명령을 명시해 놓은 도메인 네임이다. 여러분은 마스터 파일안에, 이 도메인과 관련되어 있는 도메인과 호스트네 임을 명시해야 한다. 만약 absolute라는 파일앞에 도트가 붙어 있다면, 이 파일은 환경 구성 파일 이름이라고 간주하고, 그렇지 않고 다른 문자가 붙어 있다면, 대개 이 파일은 origin파 일이라고 간주한다. 모든 origin은 그 앞에 @을 붙인다.

마스터 파일에 있는 모든 데이터는 resource records 또는 줄여서 RR로 나누어져 있다. 이것들은 DNS파일을 통해서 사용할 수 있는 정보의 가장 작은 단위로 만들어져 있다. 각 자원 레코드는 어떤 형태를 가지고 있다. 이를테면, 하나의 레코드는 IP address를 호스트네 임과 대응시킬때 사용되고, CNAME 레코드는 공식적인 호스트네임을 가지고 있는 호스트 의 익명과 연관되어 있다. 예를 들어, 115페이지에 있는 그림 6.3을 보면, virtual brewery에 해당하는 마스터 파일인 named.hosts를 볼 수 있다.

마스터 파일에 있는 자원 레코드를 일반적인 포맷으로 할당하기 위해서는 다음과 같이 하 라.

```
[domain] [tt1] [class] type rdata
```

각 필드는 공백과 탭으로 구분되어 있다. 만약 첫 번째 라인을 쓰기 전에 여는 괄호가 나오고, 닫는 괄호가 마지막 필드에 해당한다면, 하나의 개체는 여러 가지 라인으로 이어져 있다. 세미콜론과 새로운 라인사이에 있는 것은 무시된다.

**domain**

이것은 각 개체를 도메인 네임에 적용시키는 명령이다. 만약 아무런 도메인도 주어지지 않았다면, RR은 도메인이 이전에 적용시킨 RR이라고 가정한다.

**ttl**

특정한 시간이 지난후에 resolver가 어떤 정보를 강제로 폐기시키게 하기 위해서 는 RR을 "*time to live*" 줄여서 ttl과 연결시켜야 한다. ttl필드는 정보가 서버로부터 검색된 후에 유효하게 될 때 까지의 시간을 명시한다. 그 시간은 10 진수로 표시하며 대개 여덟 개의 아라비아 숫자로 되어 있다. 만약 아무런 ttl값도 주어지지 않았다면, 이전의 SOA 레코드의 minimun 필드를 초기값으로 설정한다.

**class**

이것은 IP 주소를 위한 IN 또는 Hesiod 클래스에 있는 개체들을 위한 HS와 같 은 주소 클래스이다. TCP/IP 네트워킹에서, 여러분은 이 IN을 만들어야 한다. 아무런 class 필드도 주어지지 않았다면, 이것을 이전의 RR 클래스로 가정한다.

**type**

이것은 RR의 형태를 기술한다. 일반적인 형태는 A, SOA, PTR 그리고 NS이다. 다음절에서 여러 가지 RR의 형태를 보여준다.

**rdata**

이것은 RR과 관련되어 있는 데이터를 가두어 놓는 역할을 한다. 이 필드의 형식 은 RR 형태에 의존한다. 다음절에서 각각의 RR에 대해 설명해 놓고 있다. DNS 마스터 파일에서 사용되는 RR 목록들을 전부다 기술해 놓지는 않았다. 여기서 설명하 지 않은 RR 목록들이 아주 많이 있다. 여기에서는 일반적으로 사용하는 몇가지만 기술해 놓았다.

**SOA**

이것은 권한 구역을 표시해 주고 있다. (SOA는 "Start of Authority"를 의미한 다.) 이 신호는 SOA RR에 해당하는 레코드가 도메인을 위한 믿을 만한 정보를 가지고 있다는 것을 표시해 준다. primary 문장에 포함되어 있는 모든 마스터 파일은 이러한 구역을 위한 SOA 레코드를 포함시켜야 한다. 여기에 있는 리소 스 데이터는 다음과 같은 필드를 포함하고 있다:

**origin**

이것은 이 도메인을 위한 프라이머리 네임 서버의 호스트네임을 가리킨다. 이것은 대개 절대적인 이름으로 주어진다.

**contact**

이것은 도메인을 유지 관리하고 있는 사람의 전자우편 주소를 가리킨다. 이것 은 도트 대신에 '@'이라는 문자를 사용한다. 이를테면, Virtual Brewery를 관리하고 있는 사람이 janet이라고 가정하자. 그러면 이 사람의 도메인 주소는 janet.vbrew.com이 된 다.

**serial**

이것은 구역(zone) 파일의 버전 번호를 가리킨다. 이러한 번호는 십진수 하나로 표시되어 있다. 구역 파일에 데이터가 변경될 때 마다, 이 번호는 하나씩 증가한다. 두 번째 네임서버에 의해 사용되는 이 시리얼 번호는 구역(zone) 정보가 변경되었다는 것을 인식시켜 준다. 그러한 데이터가 최고가 될 때까지 두 번째 네임서버는 일정한 간격을 두고 프라이머리 서버에게 SOA 레코드를 요청하고, 저장된 SOA 레코드를 시리얼 번호와 비교한다. 만약 그 번호가 변경되었다면, 두 번째 서버는 프라이머리서버로부터 모든 구역(zone) 데이터베이스를 전송받는다.

**refresh**

이것은 두 번째 서버가 프라이머리 서버의 SOA 레코드를 검사하는 동안에 기다리는 시간을 나타내 준다. 이것도 대부분 10진수로 되어 있으며 8개의 아라비아 숫자로 나타낸다. 일반적으로, 네트워크 위상(topology)은 그다지 자주 변경되지 않는다. 그래서, 거대한 네트워크나 이보다 작은 네트워크에서도 하루정도의 간격을 두고 명시해 주어야 한 다.

**retry**

이 번호는 두 번째 서버가 프라이머리 서버와 접속하는 시간 간격을 명시해 준 다. 만약 이 번호를 크게 잡는다면, 일시적인 접속 실패나 네트워크 문제로 인해 두 번째 서버 가 네트워크 자원을 소비하는 결과를 초래할 수도 있다. 한시간 이나 반시간 정도가 알맞다.

**expire**

이것은 두 번째 서버가 더 이상 프라이머리 서버에 접속할 수 없는 상태가 되었을 때, 이 서버가 마지막으로 모든 구역(zone)데이터를 폐기 처분할 때 걸리는 시간을 나타내준다. 일반적으로 매우 크게 잡힐 것이다. Craig Hunt ([Hunt92])는 42일을 의미한다.

**minimum**

이것은 자원(resource) 레코드를 위한 ttl의 초기값을 나타내주며, 이것은 명백하게 규정지울 수 없다. 이것은 특정한 시간이 경과한 후 에 RR(자원 레코드)을 폐기 처분하기 위해 또 다른 네임 서버가 필요하다. 그러나 어느 정도의 시간이 흐르면, 두 번째 서버는 구역정보를 갱신하지 않는다. 대개 LAN에서 네트워크 위상이 잘 변경되지 않기 때문에 minimum은 조금 크게 잡아야 한다. 한 주 또는 한 달을 잡는 것이 올바른 방법이다. 하나의 RR이 자주 변경되는 경우에, 여러분은 그것들을 다른 ttl로 할당할 수 있다.

**A**

이것은 호스트네임을 가지고 있는 IP 주소와 관련되어 있다. 자원(resource) 데이터 필드는 dotted quad notation로 표기되어 있는 주소를 가지고 있다. 각 호스트에는 오직 하나의 A 레코드가 할당되어야 한다. A 레코드에서 사용되는 호스트네임은 공식적인 호스트네임으로 간주한다. 다른 모든 호스트네임들은 CNAME 레코드를 사용한 공식적인 호스트네임과 대응되어야 한다.

**NS**

이것은 종속(subordinate) 구역의 마스터 네임 서버를 가리킨다. NS 레코드를 가져야 하는 이유는 2.5절에 나타나 있다. 자원(resource)데이터 필드는 네임서버의 호스트네임을 가지고 있다. 호스트네임을 변경시키기 위해서는 추가적으로 A 레코드가 필요하다. 소위 glue 레코드라고도 하며, 이것은 네임서버의 IP 주소에 관한 정보를 가지고 있다.

**CNAME**

이것은 canonical hostname이라고 하는 호스트의 별명과 관련되어 있다. 마스터 파일이 제공하는 A 레코드들 중에는 호스트네임도 포함되어 있다; 별명(alias)은 단순히 CNAME 레코드에 연결되어 있지만, 그 자체로서는 아무런 레코드도 가지고 있지 않다.

**PTR**

이 레코드 형태는 호스트네임을 가지고 있는 in-addr.arpa 라는 도메인과 연관 지어 사용한다. 이것은 IP 주소과 대응하는 호스트네임으로 변경시킬 때 사용한다. 이때 호스트네임은 공식적으로 사용하고 있는 호스트네임이어야 한다.

**MX**

이것은 RR이 mail exchanger를 가리키도록 한다. 메일 교환기(mail exchanger)를 가지는 이유는 13장 Mail Routing on the Internet에서 설명할 것이다. MX 레코드는 메일 교환기를 사용해서 domain을 host 네임으로 바꾸어 주는 역할을 한다.

```
[domain] [ttl] [class] MX preference host
```

모든 메일 교환기는 그것과 관련되어 있는 정수형태로 되어 있는 preference를 가지고 있다. domain으로 메일을 전달하길 바라는 우편물 대행업자 (mail transport agent)는 이러한 전달과정이 성공할 때 까지, MX 레코드를 가지고 있는 모든 호스트에게 질의를 할 것이다. 우선순위가 제일 낮은 것부터 질의를 할 것이다.

**HINFO**

이 레코드는 시스템의 하드웨어와 소프트웨어에 관한 정보를 제공한다. 이것의 문법은 다음과 같다.

```
[domain] [ttl] [class] HINFO hardware software
```

hardware는 이 호스트에 의해 사용되는 하드웨어를 가리켜 주는 필드이다. 이것 을 명시해 주기 위해서 사용하는 특별한 변환작업이 있다. 여기서 사용하는 이름 목록은 "Assigned Numbers" (RFC 1340)에 주어져 있다. 만약 하나의 필드에 공 백을 주려고 한다면, 그 필드를 "로 묶어야 한다. software 필드는 시스템에 의해 사용되는 운영체제 소프트웨어를 가리키는 이름이다. 이 이름도 "Assigned Numbers" RFC에 포함되어 있다.

### 6.2.3 Writing the Master Files

그림 6.2, 6.3, 6.4 그리고 6.5는 brewery에서 vlager로 지정되어 있는 네임서버의 예제파일 들이다. 여기서 보인 예제파일들은 대체로 간단한 것들이다. 만약 더욱더 상세한 것을 원한 다면, named에서 얻지 말고, Cricket Liu 와 Paul Albitz([AlbitzLiu92])가 쓴 "DNS and BIND"를 참조하라.

그림 6.2에 보이는 named.ca 저장(cache) 파일은 루트 네임서버를 위해 레코드를 어떻게 주느냐 하는 것을 보여주는 예이다. 전형적인 저장 파일은 대개 12개의 네임서버에 대해 설 명해 놓는다. 이 장의 맨 끝에 설명되어 있는 nslookup라는 도구를 사용해서, 여러분은 루 트 도메인을 위한 현재 네임서버 목록을 구할 수 있다.

```
; /var/named/named.ca         Cache file for the brewery
;                  We're not on the Internet, so we don't need
;                  any root servers. To activate these
;                  records, remove the semicolons.
;
; .                   99999999   IN   NS   NS.NIC.DDN.MIL
; NS.NIC.DDN.MIL      99999999   IN   A    26.3.0.103
; .                   99999999   IN   NS   NS.NASA.GOV
; NS.NASA.GOV         99999999   IN   A    128.102.16.10
```

그림 6.2: named.ca 파일

### 6.2.4 Verifying the Name Server Setup

여러분의 네임 서버 설정을 검사하기 위해 사용하는 좋은 도구가 있다. nslookup라고 하는 이것은 대화식으로나 명령행에서 사용할 수 있다. 간단하게 다음과 같이 사용할 수 있고,

```
nslookup hostname
```

이것은 resolv.conf에 명시되어 있으며 hostname에 해당하는 네임서버에게 질의할 것이다. (만약 서버안에 하나 이상의 파일이 있다면, nslookup은 임의로 하나를 선택할 것이다.)

개인 호스트에서 사용하는 대화식 모드에서는 DNS 레코드형태를 질의하고, 해당 도메인 에게 전체 구역 정보를 전송한다.

아무런 인수없이 nslookup을 사용하면, 사용할 네임서버를 화면에 출력하고, 대화식 모 드로 들어갈 것이다. > 프롬프트에서, 여러분은 질의해야할 도메인 네임을 입력할 수도 있 다.

기본값으로 클래스 A 레코드를 요청한다. 이 레코드는 도메인 네임과 연관되어 있는 IP 주소를 포함하고 있다.

여러분은 "set type=type"라고 입력함으로써 이러한 형태를 변경시킬 수 있다. type는 6.2절에 기술되어 있는 자원 레코드 이름중 하나가 된다.

예를 들어, 여러분은 다음과 같은 대화상자(dialogue)를 가질 수도 있다:

```
$ nslookup
Default Name Server:  rs10.hrz.th-darmstadt.de
Address:  130.83.56.60
```

```
> sunsite.unc.edu
Name Server:  rs10.hrz.th-darmstadt.de
Address:  130.83.56.60

Non-authoritative answer:
Name:  sunsite.unc.edu
Address:  152.2.22.81
```

만약 여러분이 어떤 IP 주소도 가지고 있지 않은 호스트네임을 찾거나, DNS 데이터베이 스에서 또다른 레코드를 찾고자 하는 경우, nslookup는 "No type A records found"라는 에 러를 화면에 출력할 것이다. 하지만 여러분은 "set type" 명령에 A라는 것을 입력해서 레코 드를 위한 질의를 만들 수 있다. 예를 들어, unc.edu의 SOA 레코드를 얻기 위해서는, 다음 과 같이 하라:

```
> unc.edu
*** No address (A) records available for unc.edu
Name Server:  rs10.hrz.th-darmstadt.de
Address:  130.83.56.60

> set type=SOA
> unc.edu
Name Server:  rs10.hrz.th-darmstadt.de
Address:  130.83.56.60

Non-authoritative answer:
unc.edu
        origin = ns.unc.edu
        mail addr = shava.ns.unc.edu
        serial = 930408
        refresh = 28800 (8 hours)
        retry   = 3600 (1 hour)
        expire  = 1209600 (14 days)
        minimum ttl = 86400 (1 day)
Authoritative answers can be found from:
UNC.EDU nameserver = SAMBA.ACS.UNC.EDU
SAMBA.ACS.UNC.EDU      internet address = 128.109.157.30
```

유사한 방법으로 MX 레코드를 질의하게 되면, 주어진 이름과 연관되어 있는 모든 리소 스 레코드를 되돌려 주게 된다.

```
> set type=MX
> unc.edu
Non-authoritative answer:
unc.edu preference = 10, mail exchanger = lambada.oit.unc.edu
lambada.oit.unc.edu      internet address = 152.2.22.80

Authoritative answers can be found from:
UNC.EDU nameserver = SAMBA.ACS.UNC.EDU
SAMBA.ACS.UNC.EDU       internet address = 128.109.157.30
```

디버깅까지 해주는 nslookup 어플리케이션은 named.ca 파일에서 현재 사용하고 있는 루 트네임서버 목록을 보여준다.

```
> set type=NS
> .
Name Server:  fb0430.mathematik.th-darmstadt.de
Address:  130.83.2.30

Non-authoritative answer:
(root) nameserver = NS.INTERNIC.NET
(root) nameserver = AOS.ARL.ARMY.MIL
(root) nameserver = C.NYSER.NET
(root) nameserver = TERP.UMD.EDU
(root) nameserver = NS.NASA.GOV
(root) nameserver = NIC.NORDU.NET
(root) nameserver = NS.NIC.DDN.MIL

Authoritative answers can be found from:
(root) nameserver = NS.INTERNIC.NET
(root) nameserver = AOS.ARL.ARMY.MIL
(root) nameserver = C.NYSER.NET
(root) nameserver = TERP.UMD.EDU
(root) nameserver = NS.NASA.GOV
(root) nameserver = NIC.NORDU.NET
(root) nameserver = NS.NIC.DDN.MIL
NS.INTERNIC.NET internet address = 198.41.0.4
AOS.ARL.ARMY.MIL         internet address = 128.63.4.82
AOS.ARL.ARMY.MIL         internet address = 192.5.25.82
AOS.ARL.ARMY.MIL         internet address = 26.3.0.29
C.NYSER.NET      internet address = 192.33.4.12
TERP.UMD.EDU     internet address = 128.8.10.90
NS.NASA.GOV      internet address = 128.102.16.10
NS.NASA.GOV      internet address = 192.52.195.10
NS.NASA.GOV      internet address = 45.13.10.121
NIC.NORDU.NET    internet address = 192.36.148.17
NS.NIC.DDN.MIL    internet address = 192.112.36.4
```

nslookup에서 사용할 수 있는 모든 명령어는 nslookup 명령에 help를 입력함으로써 얻 을 수 있다.

### 6.2.5 Other Useful Tools

여러분이 BIND 관리자로써 어떤 일을 할 때, 도움을 줄 수 있는 몇가지 도구가 있다. 이 문서에서는 그것들중 두가지만 간단히 설명하겠다. 그것들을 사용하는 방법에 대해서는 그 도구에 따라오는 설명서를 참조하기 바란다.

hostcvt는 여러분의 /etc/hosts 파일을 named에 해당하는 마스터파일로 변환시킴으로써, BIND 환경을 초기화시킬 때 도움을 줄수 있는 도구이다. 이것은 이전에 했던 A 레코드와 PTR 레코드를 대응시키고, 별명(alias)을 유지하는 일을 한다. 물론, 전체적인 작업을 하는 것은 아니다. 이를테면, SOA 레코드에 있는 타임 아웃 값을 일치시켜야 한다든지, MX레코 드를 추가시켜야 하는 작업은 여전히 여러분들의 몫이다. BIND 소스의 일부분인 hostcvt는 몇몇 리눅스 FTP 서버에 있는 스탠드얼론 패키지를 찾는데 사용할 수도 있다.

여러분의 네임서버를 설정하고 난후, 구성환경을 시험해 보고 싶어할런지도 모른다. 이러 한 작업을 하기 위해서는 perl에 기초를 두고 있는 dnswalk라는 도구를 사용하라. 이것은 DNS를 순회하면서, 일반적인 오류를 검사하고, 정보가 일치하는지를 검증하는 역할을 한다. dnswalk는 comp.source.misc에서 주기적으로 배포되고

있으며, 이 그룹(여러분이 어떤 사 이트도 들어보지 않았다면, ftp.uu.net를 저장해 두는 것도 좋은 생각이 될 것이다.)에 있는 정보를 보관하고 있는 모든 FTP 사이트에서도 쉽게 구할 수 있다.

# 7 Serial Line IP

시리얼 라인 프로토콜인 SLIP과 PPP는 인터넷에 접속할 때 사용하는 프로토콜이다. 모뎀과 는 별개로 FIFO 버퍼 장치를 갖추고 있는 시리얼 보드에는 어떤 하드웨어도 필요하지 않 다. 이 시리얼 보드를 사용하는 것은 우편함을 사용하는 것보다 더 이해하기 쉬우며, 적당한 가격대로 다이얼업 IP를 제공하는 사설 단체들이 증가하고 있다.

리눅스에서도 유용하게 사용할 수 있는 SLIP과 PPP 보드가 있다. SLIP는 어느 정도까지 개발이 되었고, PPP는 최근에 Michael Callahan과 Al Longyear에 의해서 개발되고 있다. 이 부분에 대해서는 다음장에서 자세하게 설명할 것이다.

## 7.1 General Requirements

SLIP와 PPP를 사용하기 위해서는 먼저 이전 장에서 설명했던 기본적인 네트워킹 환경을 설정해 놓아야 한다. 적어도, 네임 리솔루션을 제공해 주는 루프백 인터페이스 정도는 설정 해 놓아야 한다. 인터넷에 접속할 때, 어쩌면 DNS를 사용하고 싶을 것이다. 가장 간단한 옵 션으로는 네임 서버의 주소를 resolv.conf 파일에 넣어 두는 것이다. SLIP 연결이 활성화 되 자마자 이 서버를 활성화 시킬 것이다. 그리고 접속할 곳을 이 네임 서버에 적어 두는 것도 좋은 방법이다.

그러나, 이 방법이 최선의 것만은 아니다. 왜냐하면, 모든 네임 룩업은 SLIP/PPP 링크를 통 해서 이루어지기 때문이다. 만약 대역을 소비하는 것이 걱정된다면, caching-only 네임 서버 를 설정해 둘 수도 있다. 만약 실제로 도메인을 제공하지 않는다면, 이러한 링크는 단지 호 스트에서 제공되는 DNS 질의에 의존한 채 활동할 것이다. 이 과정에서 이점이 있다면, 그 것은 캐시를 만들 수 있다는 것이고, 그럼으로써 대부분의 질의는 오직 한번 시리얼 라인을 거쳐서 보내지게 된다. caching-only 서버에 있는 named.boot 파일은 다음과 같은 형태를 띄고 있다.

```
; named.boot file for caching-only server
directory                           /var/named
primary     0.0.127.in-addr.arpa   db.127.0.0 ; loopback net
cache       .                      db.cache  ; root servers
```

게다가 named.boot 파일에 루트 네임 서버의 유효 목록을 가지고 있는 db.cache 파일을 설 정해 두어야 한다.

## 7.2 SLIP Operation

전화접속에 의한 IP-server들은 종종 특별한 사용자계정을 통하여 SLIP 서비스를 제공한다. 그러한 계정으로 접속한 후에 당신은 일반적인 shell로 떨어지지 않는다 ; 대신 프로그램이나 혹은 shell script가 그 서버의 serial line을 위한 SLIP driver를 실행키기게 되고 적절한 네트워크 인터페이스를 설정하게 된다.

어떤 운영체제에서, SLIP driver는 사용자 영역의 프로그램이다; 커널의 일부분일 경우에 속도는 더 빠르다. 그러나 이것은 serial line이 SLIP 모드로 전환될 것을 요구한다. 이것은 SLIPDISC라는 특별한 tty line discipline에 의해 이루어진다. tty가 normal line discipline(DISCO)인데 반하여 이것은 단지 user processes의 data만을 교환하며 보통 읽기(2)와 쓰기(2) 호출을 사용하고, SLIP driver는 tty를 통하여 읽거나 쓸 수는 없다. 반면에 모든 serial port를 통하여 오는 모든 data는 SLIP driver를 직접 통과한다. SLIPDISC에서 그 역할은 뒤바뀌게 된다: 이제 어떤 사용자 영역의 process들이 tty를 통해 쓰거나 읽는 과정이 차단되고, 그동안 serial port를 통해 오는 모든 data들이 SLIP driver에 직접적으로 전달된다.

SLIP driver 그 자체는 SLIP 프로토콜을 통한 많은 변화들을 이해한다. 전형적인 SLIP과는 별개로, 그것은 IP-packet을 통해 나가는 소위 Van Jacobson header compression 에서 동작하는 CSLIP 또한 이해한다. 이것

은 interactive session들을 위한 throughput을 현저하게 발달시킨다. 추가적으로 이들 프로토콜들을 위한 6-bit 버전들이 있다.

Serial line을 SLIP 모드로 변환하기 위한 간단한 방법은 slattach tool을 이용한 방법이다. 당신의 모뎀이 /dev/cua3에 있다고 하고, SLIP server에 성공적으로 접속하였다고 하자. 그러면 당신은 이렇게 실행한다.

```
# slattach /dev/cua3 &
```

이것은 cua3의 line discipline을 SLIPDISC로 전환하고, 그것을 SLIP network interface들 중의 하나로 붙이게 된다. 만약 이것이 당신의 첫번째 active SLIP link라면 line은 sl0에 붙게 된다; 두번째는 sl1에 붙게될 것이고 계속 이런 식으로 나간다. 현재의 커널은 동시에 8개의 SLIP link들을 지원한다.

Slattach에 의해 기본값으로 선택되는 encapsulation은 CSLIP이다. 당신은 -p 스위치에 의해 다른 모드를 선택할 수 있다. 일반적인 SLIP(압축하지 않는)을 사용하기 위해서 당신은 다음 명령을 사용할 수 있다.

```
# slattach -p slip /dev/cua3 &
```

또다른 모드들은 cslip, slip6, cslip6(SLIP의 6-bit version을 위한), 그리고 adaptive SLIP을 위한 adaptive가 있다. 후자는 커널이 원격 사용자들의 SLIP encapsulation 종류를 찾아내도록 내버려둔다.

당신은 당신의 peer가 하는 것과 동일한 encapsulaiton을 사용해야 한다는 것을 명심하여야 한다. 한 예로, 만약 cowslip이 CSLIP을 사용한다면 당신도 역시 그것을 사용해야 한다. Mismatch의 증상은 원격 호스트에 ping을 하였어도 어떠한 packet도 다시 돌려보내주지 않는다. 만약 다른 원격호스트가 당신에게 ping을 하여도 당신은 콘솔에서 "Can't build ICMP header"라는 메세지를 보게된다. 이러한 어려움을 해결하는 방법 중 하나는 adaptive SLIP을 사용하는 것이다.

사실, slattach는 단지 SLIP만을 가능하게 해주는 것이 아니라, PPP나 KISS(ham radio사용자들을 위한 또다른 프로토콜)와 같은 serial line을 통한 다른 프로토콜의 사용도 가능하게 해준다. 더 자세한 사항을 위해서는 slattach(8) 매뉴얼 페이지를 참고하라.

SLIP driver로 전환한 후에, 당신은 network interface를 조절해 주어야 한다. 다시 우리는 표준이 되는 ifconfig과 route 명령을 사용한다. vlager라는 계정으로 접속한 경우를 생각해서 우리가 cowslip이라는 서버에 dial up으로 접속했다고 하자. 당신은 이렇게 실행해야 한다.

```
# ifconfig sl0 vlager pointopoint cowslip
# route add cowslip
# route add default gw cowslip
```

첫번째 명령줄은 interface를 point-to-point link로 cowslip으로 설정한다. 두번째와 세번째 명령줄은 cowslip으로의 route를 더하는 것이고, default route로 cowslip을 gateway로 쓰겠다는 뜻이다.

SLIP link를 끝낼 때에는, 당신은 먼저 cowslip을 통한 모든 route들을 del option으로 삭제해야 한다. interface를 종료하고 slattach에 접속을 끊겠다는 신호를 보내야 한다. 그 다음에 다시 당신의 터미널 프로그램을 이용하여 모뎀을 끊어야 한다.

```
# route del default
# route del cowslip
# ifconfig slo down
# kill -HUP 516
```

## 7.3 Using dip

이제 전장의 것은 다소 간단하였다. 그럼에도 불구하고 당신은 위에서 보여진 전절의 모든 과정들을 단순한 명령만으로 자동적으로 실행하기를 원할지도 모른다. dip이 바로 이것을 위한 것이다. 이 글이 쓰여지는 현재의

버전은 3.3.7이다. 이것은 많은 사람들에 의해 heavily하게 패치되었으므로 당신은 '그' dip 프로그램에 대해서 더 이상 말할 필요가 없다. 이러한 발전의 다른 변형들이 미래의 버전에 hopefully하게 merged in 될것이다.

dip은 당신의 모뎀을 제어하고,line을 SLIP 모드로 변환하고, interface를 조정하기 위한 간단한 scripting 언어의 번역을 할 수 있다. 이것은 다소 원시적이고 제한적이기는 하지만 대부분의 경우에 충분하다. 언젠가 새로운 release의 dip에서는 더 많은 versatile 언어가 지원될 것이다.

SLIP interface를 조절하기 위해서, dip은 root의 권한을 요구한다. 그것은 dip의 setuid를 root로 할 것을 유혹하는 데 그러면 모든 사용자들이 어떤 SLIP 서버에 root 권한이 없이도 접속할 수 있게 된다. 이것은 매우 위험한데, 왜냐하면 bogus interfaces를 설정하는 것과 dip을 이용한 default route는 당신의 network를 심각하게 손상시킬수도 있기 때문이다. 더 나쁜 경우, 당신의 사용자들에게 '어떤' SLIP 서버에도 접속할 수 있는 권한을 주게 되는 것이며 당신의 네트워크에서 위험한 공격을 가할 수 있게 된다. 그래서 당신이 당신의 사용자들에게 SLIP 연결을 하도록 하고 싶다면, 각각의 개별적인 SLIP 서버를 위한 작은 wrapper 프로그램을 사용하고, 이 wrapper들이 접속을 만들기 위한 특별한 script들을 invoke하게 된다. 이 프로그램들은 안전하게 root 권한을 만들 수 있다.

### 7.3.1 A Sample Script

**Figure: A sample dip script**

```
# cow slip의 전화접속을 위한 dip 예제 스크립트
# 로컬과 원격 이름과 주소를 설정
get $local vlager
get $remote cowslip

port cua3                # 시리얼 포트를 선택
speed 38400              # 최대 속도를 설정
modem HAYES              # 모뎀 종류를 설정
reset                    # 모뎀과 tty를 재설정
flush                    # flush out modem response

# 전화걸기를 준비
send ATQ0V1E1X1\r
wait OK 2
if $errlvl != 0 goto error
dial 0123456789
if $errlvl != 0 goto error
wait CONNECT 60
if $errlvl != 0 goto error

# 오케이, 이제 연결!
sleep 3
send \r\n\r\n
wait ogin: 10
if $errlvl != 0 goto error
send Cvlager\r
wait ssword: 5
if $errlvl != 0 goto error
    #better not leave your password in ascii (thanx noud)
password
wait running 30
if $errlvl != 0 goto error
```

```
#당신의 원격 IP를 설정하기 위해
get $remote remote
print remote = $remote
if $errlvl != 0 goto error
wait to 3
get $local remote
print local = $local
if $errlvl != 0 goto error


# 이제 우리는 연결되었고, 원격측은 CSLIP을 시작
print Connected to $remote with address $rmtip
default                 # Make this link our default route
mode CSLIP               # We go to CSLIP mode, too
# 에러가 떨어졌을 경우
error:
print CSLIP to $remote failed.
```

예제 스크립트가 실제로 만들어 졌다. Argument로 지정된 스크립트를 이용해 dip이 cowslip을 연결하기 위해 사용될 것이다.

```
# dip cowslip.dip
DIP: Dialup IP Protocol Driver version 3.3.7 (12/13/93)
Written by Fred N. van Kempen, MicroWalt Corporation.

connected to cowslip.moo.com with addr 193.174.7.129
#
```

Cowslip에 접속하고, CSLIP을 가능하게 한 후에 dip은 터미널을 떠나서 백그라운드 작업으로 전환될 것이다. 그런 후에 당신은 CSLIP 링크를 이용하여 일반적인 네트워킹 서비스들을 이용할 수 있다. 접속을 끊기 위하여, 단순히 -k옵션을 사용하면 된다. 이것은 전화를 끊는 신호를 /etc/dip.pid에 있는 dip의 pid를 알아내어 dip에 보낸다.

```
# kill -k
```

dip의 스크립트 언어에서, $ 표시가 앞에 붙는 키워드들은 변수들의 이름을 표시한다.dip은 아래에서 보여질 미리 정의되어진 변수들의 집합을 가지고 있다. SLIP 링크와 연관된 local과 remote의 호스트네임을 포함하고 있는 $remote와 $local 이 그 예이다.

예제 스크립트에 있는 처음 두 선언들은 dip이 변수들을 설정하는 명령들을 갖고 오는 것이다. 여기에서 local과 remote의 호스트 네임은 상대적으로 vlager와 cowslip으로 설정되었다.

다음 다섯개의 선언들은 터미널 라인과 모뎀을 설정한다. *reset*은 reset 문자열을 모뎀으로 보낸다. ; Hayes 호환 모뎀의 경우, 이것은 ATZ 명령이다. 다음 선언은 모뎀의 반응을 분출시킨다. 그리하여 로긴 chat이 다음의 몇 줄이 제대로 작동할 수 있도록 해주는 것이다. 이 chat은 곧이 곧대로 앞으로만 나아간다: 이것은 단순히 cowslip의 전화번호인 49188로 전화를 걸고, hey-jude 암호를 이용하여 Svlager 계정에 로그인한다. wait명령은 dip이 첫번째 argument를 기다리도록 한다.; 두번째 argument로 주어진 숫자는 만약 어떠한 문자열도 수신되지 않았을 경우 주어진 그 시간만큼 대기하도록 하는 것이다. if 명령은 명령이 수행되는 동안 어떠한 에러도 발생하지 않을 경우 로그인 과정에서 흩뿌린다(?).

마지막 명령 default는 로그인 한 후에 수행된다. 이 명령은 라인에 SLIP 모드를 가능하게 해주고, 당신에게 맞는 인터페이스와 라우팅 테이블을 설정해주는 default route를 모든 호스트와 모드의 SLIP 링크를 만들어준다.

### 7.3.2 A dip Reference

매우 널리 쓰임에도 불구하고, dip은 아직 그 명령이 잘 정리되지 않았다. 이 부분에서는 거의 대부분의 dip 명령어들을 정리해 볼 것이다. 당신은 dip의 테스트 모드에서 모든 명령어들을 훑어볼 수 있고 help 명령을 이용해서 그 안으로 들어가 볼 수 있다. 명령들의 문법적인 부분들을 알아보기 위해 어떤 변수도 없이 그것이 입력해야 한다. ; 물론 변수들이 없이는 어떠한 동작도 하지 않는다.

```
DIP> help
DIP knows about the following commands:

        databits default  dial     echo     flush
        get      goto     help     if       init
        mode     modem    parity   print    port
        reset    send     sleep    speed    stopbits
        term     wait

DIP> echo
Usage: echo on|off
DIP>
```

아래에서, 예들은 DIP> 프롬프트를 어떻게 테스트 모드에서 입력하는지 보여주고, 어떻게 그 결과가 나타나는지를 보여준다.

**The Modem Commands** 당신의 시리얼 라인과 모뎀을 조절하기 위해 dip이 제공하는 명령은 매우 많다. 이들 중 시리얼 포트, 속도와 데이타비트, 정지비트, 패러티를 설정하는 port와 같이 일반적인 라인 변수를 설정하는 명령들은 아주 명백하다.

modem 명령은 모뎀의 종류를 선택한다. 현재 지원되는 타입은 HAYES(대문자가 요구됨)밖에 없다. 당신은 반드시 dip에 모뎀의 타입을 제공해주어야 하며, 그렇지 않을 경우 다이얼과 reset명령은 거부될 것이다. reset 명령은 reset 문자열을 모뎀에 보낸다.; 거기에 쓰이는 문자열은 선택된 모뎀의 종류에 따른다. 일반적으로 Hayes호환 모뎀의 경우에는 ATZ가 된다.

flush코드는 모뎀이 멀리 보낸 명령들을 분출시키는데 사용된다(?). 그렇지 않으면 reset 명령뒤에 따라오는 chat 스크립트는 혼란스러워지게 되는데 이것은 먼저번의 OK 응답을 읽에 되기 때문이다.

init 명령은 모뎀이 전화를 걸기전에 초기화를 위해 선택한다. Hayes호환 모뎀을 위한 기본값은 “ATE0 Q0 V1 X1”이다.

dial 명령은 마지막으로 초기화 문자열을 보낸 후에 원격 시스템에 전화를 건다. Hayes 모뎀을 위한 기본명령은 ATD이다.

**echo and term** echo 명령은 디버깅 목적을 위해 사용된다. echo on 하게 되면 시리얼을 통해 가는 모든 내용을 콘솔에 dip이 echo하게 된다. 이것은 echo off에 의해 꺼진다.

dip은 또한 당신이 일시적으로 스크립트 모드를 떠나 터미널 모드로 들어가도록 허락해 준다. 이 모드에서 당신은 dip을 일반 터미널의 프로그램처럼 사용할 수 있으며, 시리얼 라인에 쓰고, 그것으로부터 읽을 수 있다. 이 모드를 종료하기 위해서는 Ctrl-]을 입력하라.

**The get Command** get 명령은 dip의 변수를 설정하는 방법이다. 가장 간단한 형식은 위에서 계속 보여진대로 변수에 상수를 설정하는 방법이다. 그러나 아마도 당신은 사용자에게 어떤 값 대신에 어떤 특정한 키워드를 입력하도록 하고 싶을 것이다.

```
DIP> get $local ask
Enter the value for $local:
```

세번째 방법은 원격호스트로부터 값을 얻어오는 것이다. 처음에 보여진 Bizarre와 같이 어떤 경우에는 이것이 매우 유용하다: 어떤 SLIP 서버들은 SLIP 링크에 당신만의 IP-주소를 허락하지 않고, 당신에게 어떤 주소가 할당되어 있는 지를 메세지를 통해 보여주면서 형편없는 주소를 당신이 어디에서 접속하든 당신에게 할당할 것이다. 만약 그 메세지가 "Your address:193.174.7.202" 이런 식으로 보인다면 아래의 dip code가 당신이 원하는 주소를 택할 수 있도록 해 줄 것이다.

```
wait address: 10
get $locip remote
```

**The print command** 이것은 dip이 시작된 콘솔에 텍스트를 echo하기 위한 명령이다. 다음과 같이 어떠한 dip의 변수라도 print 명령에 사용될 수 있다.

```
DIP> print Using port $port at speed $speed
Using port cua3 at speed 38400
```

**Variable Name** dip은 단지 먼저 정의되어 있는 변수들만을 이해한다. 변수명은 반드시 $로 시작해야 하며 소문자들로 쓰여져야 한다.

$local과 $locip 변수는 로컬 호스트의 이름과 IP-주소를 포함한다. 호스트네임을 설정하는 것은 dip이 인정된 호스트네임을 $local에 저장하며, 동시에 그에 해당하는 IP-주소를 $locip에 저장한다. 유사한 경우가 $locip를 설정할 경우에 일어난다.

$remote와 $remotip 변수들도 똑같이 원격호스트의 이름과 주소를 포함한다. $mtu는 접속의 MTU 값을 포함한다.

이들 다섯개의 변수들이 get 명령을 이용해 직접적으로 설정되는 값들이다. 다른 변수들의 host는 단지 그에 해당하는 명령들만을 이용해 설정되나 print 선언에 의해 사용된다. ; 이런 것들은 $modem, $port, 그리고 $speed이다.

$errlvl은 마지막 실행된 명령의 결과에 결정되는 변수이다. 만일 이 값이 0이면 성공을 의미하고 다른 0이 아닌 값일 경우에는 에러를 의미한다.

**The if and goto command** if 명령은 일반적으로 생각하는 if라기보다는 상황에 따른 경우이다. 사용 문법은

```
if var op number goto label
```

이 표현은 반드시 $errlvl, $locip, and $rmtip와 같은 변수들 사이에서 간단한 비교여야 한다. 두번째 피연산자는 반드시 정수여야 한다; 연산자는 ==, !=, <, >, <=, 그리고 >= 중의 하나이어야 한다.

goto 명령은 라벨을 포함하고 있는 라인의 스크립트의 연속적인 실행을 가능하게 한다. 라벨은 반드시 그 라인의 첫부분이어야 하며, 콜론이 뒤에 따라붙어야 한다.

**send, wait and sleep** 이 명령들은 dip의 간단한 chat 스크립트들을 돕는 도구들이다. send는 그것의 argument들의 시리얼 라인으로 출력한다. 변수들을 지원하지는 않지만 n과 b와 같은 모든 C-스타일의 역슬래쉬 문자시퀀스를 인식한다. 틸드문자 ()는 리턴/개행문자의 대용으로 사용한다.

wait는 한 단어를 하나의 argument로 간주하고, 그것이 이 단어를 인식할 때까지 시리얼 라인을 통한 모든 입력을 검사한다. 그 단어 자신은 어떠한 빈칸도 가지고 있어서는 안된다. 선택적으로, 당신은 두번째 argument로

써 timeout을 주어야한다.; 만약 기대되었던 문자가 오랜 시간이 지나도 수신이 되지 않는다면 그 명령은 1이라는 $errlvl를 반환한다.

sleep 선언은 어느 정도의 특정의 시간을 대기하기 위해 사용한다. 한 예로 로그인 시퀀스를 완벽하게 인내심있게 기다리기 위해 사용된다. 다시, 그 시간간격은 초로 표현된다.

**mode and default** 이 명령들은 시리얼 라인을 SLIP모드로 전환하고 인터페이스를 조절하기 위해 사용된다.

mode 명령은 데몬모드로 들어가기 전에 dip에 의해 실행되는 마지막 명령이다. 에러가 발생하지 않는다면 그 명령은 반환되지 않는다.

mode는 프로토콜의 이름을 argument로 취급한다. 현재 dip은 SLIP과 CSLIP을 유효한 이름들로 인식한다. 그러나 현재의 버전의 dip은 adaptive SLIP은 인식하지 못한다.

시리얼 라인에서 SLIP 모드를 가능하게 한 후에, dip은 인터페이스를 point-to-point link로 설정하기 위해 *ifconfig*을 실행하게 되고 경로를 원격 호스트에 맞추기 위해 route명령을 시도한다.

추가로, 만약에 스크립트가 모드 전에 기본 명령을 실행한다면, dip는 기본 route point를 SLIP 링크로 정할 것이다.

## 7.4 Running in Server Mode

당신의 SLIP 클라이언트를 설정하는 것은 무척 힘든 일이다. 그러나 이에 반해, 즉 당신의 호스트를 SLIP서버로 만드는 것은 훨씬 쉬운 일이다.

이것을 가능하게 하는 방법 중 하나는 diplogin을 사용해 dip을 서버모드로 사용하는 것이다. 이것의 주요 설정화일은 이 호스트가 정해주는 로그인 이름과 주소가 담겨있는 /etc/diphosts 화일이다. 상대적으로 당신은 BSD에서 유래한 도구인 sliplogin을 사용할 수 있다. sliplogin은 당신이 실행해야 하는 호스트 접속과 접속을 끊는 쉘 스크립트를 좀 더 간편하게 설정할 수 있다. 현재 이것은 베타버전이다.

양쪽 프로그램 모두 하나의 SLIP 클라이언트 당 하나의 로그인 계정을 요구한다. 한 예로 당신이 dent.beta.com의 Arthur Dent의 SLIP 서비스 제공자가 된다고 상상해보라.당신은 당신 호스트의 passwd 화일에 다음과 같은 내용을 추가함으로써 dent라는 계정을 만들어야 한다.

```
dent:*:501:60:Arthur Dent's SLIP account:/tmp:/usr/sbin/diplogin
```

그런 후에 당신은 dent의 비밀번호를 passwd utility를 사용해서 설정해야 한다.

이제 dent가 로그인 하고, dip은 서버를 시작할 것이다. 그가 정말로 SLIP을 사용하도록 허가 되었는지 알아보기 위해 dip은 /etc/diphosts에서 사용자 이름을 찾을 것이다. 이 화일은 각각의 SLIP 사용자들의 허가 권리와 연결 파라미터룰 세밀히 검토할 것이다. dent를 위한 예제가 아마도 다음과 같을 것이다.

```
dent::dent.beta.com:Arthur Dent:SLIP,296
```

첫번째 콜론으로 나누어진 부분은 로그인한 사용자의 이름일 것이다. 두번째 부분은 비밀번호를 포함하고 있을 것이다(아래를 보라). 세번째 부분이 거는 쪽의 호스트네임이거나 IP-주소이다. 다음으로 오는 내용들은 그다지 별 의미가 없는 것들이다. 마지막 부분은 연결 파라미터들이다. 이것은 콤마로 분리되어 MTU가 뒤어 붙는 프로토콜(현재로서는 SLIP이나 CSLIP)을 열거한다.

dent가 로그인 했을때, diplogin은 diphosts 화일로부터 그에 관한 정보를 추출하고 만약, 두번째 부분이 비어있지 않다면 "외부 보안 비밀번호"를 요구한다. 사용자에 의해 입력된 이 문자열은 diphosts화일의 password(암호화되지 않은)와 비교된다. 만약 둘이 서로 맞지 않다면 로그인 시도는 거부된다.

그렇지 않고 diplogin이 시리얼라인을 CSLIP이나 SLIP 모드로 진행해 나가면, 인터페이스와 route를 설정한다. 이 연결은 사용자가 접속을 끊거나 모뎀이 전화선에서 끊길때까지 남아있다. 그러면 diplogin은 라인을 일반 라인으로 돌리고 빠져나온다.

diplogin은 super-user 권한을 요구한다. 만약 당신이 dip를 root로 실행할 권한이 없을 경우 단순한 링크 대신에 당신은 diplogin을 분리된 dip의 복사본으로 만들어야 한다. 그러면 diplogin은 dip 그 자신의 상태에 영향을 미치지 않고 안전한 setuid를 가질수 있게 된다.

# 8 The Point-to-Point Protocol

## 8.1 Untangling the P's

SLIP과 마찬가지로, PPP는 시리얼 접속을 통해 데이타를 보내는 프로토콜이지만 전자의 부족함을 보충한다. 그것은 시작할때의 최대 datagram의 크기와 IP 주소와 같을 것들을 조절할 수 있도록 해주며 클라이언트의 확인을 해준다. 이런 각각의 능력때문에, PPP는 각각의 프로토콜을 가진다. 아래에서 우리는 간략하게 이와 같은 기초적인 PPP의 블록들을 알아볼 것이다. 이것은 거의 완벽하지는 않다. ; 만약 당신이 PPP에 관해 더 알고자 한다면 당신은 RFC-1548에 있는 내용들을 읽어야 하며 많은 RFC에 관한 내용들 또한 읽어보아야 할 것이다.

PPP의 아주 낮은 부분이 16-bit checksum에 의해 PP 프레임들의 경계를 결정하는 High-Level Data Link Control 프로토콜이다. 좀 더 원시적인 SLIP encapsylation과는 반대로, PPP 프레임은 IP가 아닌 Novell의 IPX나 Appletalk와 같은 프로토콜들 패킷들을 잡아둘 수 있다. 이는 PPP가 프레임에 의해 전달되는 패킷의 종류들을 확인하는 기본 HDLC 프레임에 프로토콜 영역을 추가함으로써 이루어진다. LCP, Link Control Protocol,은 한쪽에서 받아들일수 있는 최대 데이타의 크기인 datagram의 크기를 말해주는 Maximum Receive Unit(MRU)와 같은 데이타링크에 속해있는 교신옵션을 위해 HDLC의 맨 윗부분에 사용된다.

PPP 링크의 설정과정에서 중요한 단계는 클라이언트에의 허가이다. 비록 그것이 위임된 것이기는 아니기는 하지만 다이얼업 라인들에서는 반드시 필요하다. 일반적으로 호스트로 불리는 쪽에서는 어떤 특별한 암호를 아는가에 의해서 클라이언트 자신이 인증하도록 묻는다. 만일 전화를 건 쪽이 정확한 암호를 대는데 실패하면 접속은 끊어진다. PPP에서 인증의 방법은 두 가지이다 ; 즉, 전화를 건 쪽 또한 서버에 그 자신을 인증하도록 한다. 이 인증과정들은 서로서로에게 전혀 무관하다. 여기에는 두가지 서로 다른 인증 프로토콜이 있는데, 다음에 더 자세하게 다룰 것이다. 이것들은 Password Authentication Protocol, 혹은 PAP, 그리고 Challenge Handshake Authentication Protocol, 혹은 CHAP라 이름 붙여져 있다.

IP나 AppleTalk 등과 같이 데이타 링크를 거치는 각각의 네트워크 프로토콜들은 Network Control Protocol(NCP)를 사용하여 유동적으로 설정된다. 한 예로 링크를 거쳐 IP datagram을 보내기 위해, PPP는 IP-datagram들의 Van-jacobson 헤더 압축을 지원한다. 이것은 TCP패킷들의 헤더들을 3바이트 정도로 작게 하기 위한 기술이다. 이것은 CSLIP에서도 역시 사용되며 보다 더 쉽게 VJ-헤더압축에서 언급된 바 있다. 압축의 사용은 IPCP를 통해 start up 될 때 교신이 이루어진다.

## 8.2 PPP On

PPP는 기능은 크게 두 부분으로 나뉜다. 첫번째는 커널 내에 위치하는 낮은 수준의 HDLC driver이고, 두번째는 여러가지 control protocol을 조정하는 사용자영역의 pppd deamon이다. 현재 release된 PPP는 linux-ppp-1.0.0이며 커널 PPP 모듈, pppd, 원격 시스템에 전화를 거는데 쓰이는 chat이란 프로그램을 포함하고 있다.

PPP 커널 드라이버는 Michael Callanhan에 의해 쓰여졌다. pppd는 Drew Perkins와 다른 사람들에 의해 쓰여진 SUN과 386BSD를 위한 자유로운 PPP implementation에서 derived 고 Paul Mackerras에 의해 유지되고 있다. 이것은 Al Longyear에 의해 포팅되었다. SLIP과 같이, PPP는 특별한 line discipline을 위해 implemented되었다. 어떤 serial line을 PPP link로 쓰기 위해서는, 당신은 먼저 일반적으로 당신의 모뎀을 통해 접속을 형성해야 하며, 그 다음 line을 PPP mode로 바꾸어야 한다. 이 모드에서, 모든 들어오는 data는 들어오는 validity(각각의 HDLC frame은 16-bit의 checksum을 운반한다)을 위한 HDLC frames들을 검사하고, 그것을 다시 풀고 diapatche하는 PPP 드라이버를 거쳐야 한다. 현재, 그것은 IP datagram들을 조정할 수 있고, 선택적으로 Van-Jacobson header compression을 사용할 수 있다. IPX가 지원하자 마자, PPP 드라이버는 IPX 패킷도 조정할

수 있을 것이다.

커널 드라이버는 링크를 통해 실제 네트워크 트래픽 이전에 필요한 전체적인 초기화와 authentication phase를 수행하는 PPP daemon, pppd에 의해 aid된다. pppd의 행동은 잘 조절된 많은 옵션들을 사용하는 것과 같을 것이다. PPP는 다소 복잡하기 때문에, 그것은 하나의 장에서 모든 것들을 설명하는 것은 불가능하다. 이 책은 pppd의 모든 것들을 수용해 낼수 없기 때문이다. 그러나 단지 당신에게 소개하는 정도는 할 수 있을 것이다. 더 많은 정보를 위해서는 매뉴얼 페이지와 pppd 소스판에 있는 README들을 참고하라. 그러면 이 장에서 다 논의되지 못했기 때문에 당신이 궁금해 할 수 있는 사항에 대해서 도움을 받을 수 있을 것이다. 만약 당신의 문제가 모든 문서를 읽었음에도 제기된다면, pppd의 발전에 관련된 많은 사람들이 속해있는 뉴스그룹 comp.protocols.ppp를 참고하라.

## 8.3 Running pppd

당신이 PPP link를 통해 인터넷에 접속하고 싶다면, loopback device나 resolve와 같은 것들을 이용해서 기본적인 네트워킹 capabillity들을 셋업해야 한다. 양쪽다 전장에서 설명되었었다. Serial link를 이용한 DNS의 사용에 관해서 이야기하여야 할 것들이 있다. ; 이것에 관해서는 SLIP에 관해 논의된 장을 참고하라. pppd를 이용한 PPP 접속에 관해 간략한 예를 들기 위해 다시 당신이 vlager에 있다고 하자. 당신은 이미 PPP 서버인 c3op에 접속했고 ppp 계정에 로그하였다. c3po는 벌써 그것의 PPP 드라이버를 구동하였다. 다이얼을 위한 통신프로그램을 빠져나온 후 당신은 다음 명령을 실행해야 한다.

```
# pppd /dev/cua3 38400 crtscts defaultroute
```

이것은 serial line cua3을 PPP모드로 flip하고 c3po로의 IP-link를 만든다. Serial port를 통한 전송속도는 38400bps가 될 것이다. Crtscts 옵션은 9600bps 이상의 속도에서 확실한 port의 하드웨어 handshake를 켠다. Pppd가 시작한 후 첫번째로 하는 일은 LCP를 사용하는 원격의 여러 개의 link 특성들과 교신하는 것이다. 일반적으로, 기본적인 옵션의 설정으로 잘 동작하므로 여기서는 더 논의하지 않는다. 나중의 섹션에 좀 더 자세한 LCP로 돌아갈 것이다. 이제 pppd는 IP control protocol인 IPCP를 사용하는 IP parameter와 교신할 것이다. 위에서 pppd에 특정한 IP-주소를 설정하지 않았기 때문에, 그것은 resolver를 사용하여 local hostname에서 얻어진 주소를 사용하려고 할 것이다. 양쪽다 그들의 주소를 서로에게 알려줄 것이다.

일반적으로 이러한 기본값에 대해서 잘못된 것은 없다. 심지어 당신의 컴퓨터에 이더넷에 있다하더라도 이더넷과 PPP interface 모두에 같은 IP-주소를 사용할 수 있다. 그럼에도 불구하고 pppd는 당신에게 다른 주소를 사용하도록 허락하거나, 다른 주소를 사용할 것인지를 물어온다. 이러한 옵션들은 다음 장에서 논의될 것이다.

IPCP 셋업으로 통과한 후에, pppd는 당신의 네트워킹층을 PPP 링크를 사용하기 위해 준비할 것이다. 첫번째로 PPP 네트워크 인터페이스를 point-to-point 링크로 조절하고, ppp0를 첫번째 PPP 링크로, ppp1을 두 번째로, 이런 식으로 계속해 나간다. 다음으로 링크의 다른 한 쪽 끝인 호스트를 가르키는 routing table entry를 셋업할 것이다. 위에서 보여진 예에서 pppd는 c3po로 기본 네트워크 라우트 포인트를 만들 것이다. 왜냐하면 그것이 defaultroute 옵션으로 주어졌기 때문이다. 이것은 당신의 local network에 있지 않는 호스트로 향한 모든 datagram들이 c3po로 가도록 한다. 또다른 많은 routing scheme들을 pppd는 제공하며, 그것은 다음 장에서 자세하게 설명될 것이다.

## 8.4 Using Options Files

pppd의 command line argument들을 설명하기전에, pppd는 기본 옵션으로 되어 있는 몇몇 화일들을 찾아본다. 이 화일들은 모든 확실한 command line argument들을 포함하고 있고, 그 양이 몇 줄이 될런지 알 수 없다. 소개되는 comment들은 특정한 신호를 가지고 있다. 첫번째 옵션 파일은 pppd가 시작할 때 언제나 찾는 /etc/ppp/options이다. 이 화일이 당신에게 당신의 사용자들이 보안에 대한 타협을 하도록 해주기 때문에 이 화일에 전반적인 기본 설정을 맞추어 놓는 것이 좋다. 한 예로, pppd가 peer로 부터 어떤 종류의 인증(PAP나 CHAP)을 하도록 하기 위해 이 화일에 auth에 관한 옵션을 추가할 수 있다. 이 옵션은 사용자들에 의해 덮어쓰여지지 않기 때문에 database들에 인증되어 있지 않은 어떤 system에도 PPP접속을 하는 것은 불가능하다.

/etc/ppp/options이 읽혀지고 난 후에 찾는 다른 옵션 화일은 사용자의 홈디렉토리에 있는 .ppprc화일이다. 그것은 각 사용자들에게 그들만의 기본옵션들을 설정할 수 있도록 해준다. /etc/ppp/options의 예제화일은 다음과 같이 보일것이다:

```
# Global options for pppd running on vlager.vbrew.com
auth   # 인증을 요구함
usehostname  # CHAP을 위한 local hostname을 사용함
lock   # UUCP-style 디바이스를 잠그기 위해 사용함
domain.vbrew.com # 우리의 도메인 네임
```

이들 옵션의 첫번째에 있는 두가지 옵션들이 인증을 위해 사용되며 아래에서 설명되었다. lock 키워드는 pppd가 device를 잠그는 표준 UUCP의 방법을 사용하도록 한다. 이것에 의해 각각의 serial device를 이용하는 프로세스들은, /dev/cua3과 같은, LCK..cua3과 같은 lock 파일을 device가 사용중인 UUCP spool directory로 생성한다. 이것은 minicom이나 uucico와 같은 다른 프로그램들이 PPP가 사용되는 동안 serial device를 사용하는 것을 방지하는데 필요하다. 이러한 옵션들이 전체 설정 화일에 쓰여진 이유는 위에서 보여진 것과 같은 옵션들은 다시 덮어쓰여질 수 없기 때문이고, 그래서 적당한 수준의 보안을 유지할 수 있다. 그러나 이제 보여질 접속에 관한 문자열과 같은 어떤 옵션들은 나중에 다시 덮어씌어 질수 있다는 것에 주목하라.

## 8.5 Dialing out with chat

위에서 보여진 예에서 당신을 불편하게 만들지도 모르는 것 중의 하나는 당신이 pppd를 시작하기 전에 일일이 연결을 해야한 다는 것이다. dip와는 다르게, pppd는 원격의 시스템에 전화를 걸고 접속하는 pppd 자신의 스크립트 언어를 갖고 있지않다. 그러나 이 일을 하기 위해 pppd는 외부 프로그램이나 shell 스크립트에 의존한다. 접속을 위해 실행되어야 할 명령은 command line 옵션으로 pppd에 주어질 수 있다. pppd는 명령의 표준 입력과 출력을 시리얼 라인으로 돌린다. 이를 위한 유용한 프로그램으로는 Don Libes에 의해 쓰여진 expect가 있다. expect는 바로 이런 종류의 프로그램을 위해 고안된 매우 강력한 언어인 Tcl에 기반을 두고 있다. Pppd 패키지는 당신에게 UUCP 스타일의 chat 스크립트를 열거할 수 있게 해주는 chat라는 유사한 프로그램과 함께 딸려온다. 기본적으로, chat 스크립트는 원격 시스템에서 날아오는 문자열과 우리가 그에 대답해야 하는 문자열이 교차되는 순서에 의해 구성되어있다. 우리는 상대적으로 이것들을 expect와 send 문자열이라고 한다. 다음은 chat스크립트의 전형적인 발췌이다.

```
ogin: b1ff ssword: s3kr3t
```

이것은 chat가 원격 시스템에서 로그인 프롬프트를 보내오기를 기다렸다가 로그인 네임인 b1ff를 답하는 것을 말해준다. 우리는 단지 ogin: 만을 기다리는데 이로 인해 로그인 프롬프트가 대문자 L인지 소문자 l인지 신경쓰지 않아도 되며, 혹 잘못 날아왔을 경우도 상관없다. 그 다음 문자열은 chat가 패스워드 프롬프트를 기다렸다가 우리의 응답을 보내는 문자열이다. 이것이 기본적으로 chat 스크립트가 하는 것이다. 물론 PPP 서버에 다이얼업에 의한 완전한 스크립트는 적당한 모뎀명령을 포함해야 한다. 당신의 모뎀이 Hayes command set을 이해한다고 하고, 서버의 전화번호가 318714라고 하자. 완전한 c3po에의 접속을 만들기 위한 완전한 chat 스크립트는 다음과 같을 것이다.

```
$ chat -v '' ATZ OK ATDT318714 CONNECT '' ogin: ppp word: GaGariN
```

정의에 의해, 처음 문자열은 expect 문자열이 되어야 할 것이지만 모뎀은 우리가 그것을 kick(?)하기전에 모뎀은 아무런 응답도 없을 것이므로 우리는 처음에 빈 문자열을 줌으로써 chat이 처음 expect 문자열을 건너뛰게 해야한다. 그 다음 우리는 ATZ를 보내, Hayes-compatible 모뎀을 위한 리셋 명령을 주고, (OK) 응답을 기다린다. 다음 chat을 통해 전화번호를 다이얼명령으로 보내고, CONNECT 메세지를 기다린다. 여기서 다시 빈 문자열을 받게 되는데, 우리가 아직 아무것도 보내지 않았기 때문이다. 그러나 다시 로그인 프롬프트를 기다리게 된다. Chat 스크립트는 위에서 기술한 그대로 동작한다는 것을 명심하라. -v 옵션은 syslog 대몬의 local2 facility에 의해 모든 활동에 대한 chat log를 만든다. (facility : 만일 당신이 이들 log 메세지를 리다이렉트하

도록 syslog.conf를 수정했다면, 이 화일은 읽을 수 없기때문에 chat가 로그들의 암호와 모든 것을 포함한 전체 chat 스크립트를 디폴트로 만든다.)

사용자들이 ps 명령을 이용해 process command를 볼 수 있기 때문에 chat 스크립트를 command line에서 열거하는 것은 다소 위험하다. 그러므로 당신은 chat script를 dial-c3po라 불리는 하나의 화일에 넣음으로서 이런 문제를 피할 수 있다. 당신은 -f '화일명' 옵션을 줌으로 인해 command line에서 명령을 나열하는 대신 화일로부터 스크립트를 읽어들일 수 있다. 그래서 완전한 pppd '주문'은 다음과 같을 것이다.

```
# pppd connect "chat -f dial-c3po" /dev/cua3 38400 -detach \
        crtscts modem defaultroute
```

다이얼 업 스크립트의 나열에 의한 연결 옵션 이외에도, 우리는 command line에 두 개의 옵션을 추가하였다: pppd에게 콘솔에 달라붙지 말고 백그라운드 프로세스가 되도록 말해주는 -detach이다. 또, modem 키워드는 모뎀에 어떤 동작-전화를 걸기전이나 건 후 선을 끊는 시리얼 장치의 특별한 동작-을 수행한다. 만약 당신이 이 키워드를 사용하지 않는다면, pppd는 포트의 DCD 라인을 모니터하지 않을 것이며 그러면 만일 원격 시스템이 갑자기 끊어졌다든가 하는 경우를 전혀 알아차리지 못한다. 위에서 보여진 예들은 다소 간단한 것이다; chat는 훨씬 더 복잡한 스크립트를 수행할 수 있다. 아주 유용한 한 가지 경우는 chat에 에러가 났을 경우 이를 취소하는 문자열을 보낼 수 있다. 전형적인 취소 문자열들은 당신이 건 전화가 통화중이거나, 전화기를 들지 않았을 경우에 보여지는 BUSY, NO CARIIER와 같은 메세지들이다. chat가 이를 바로 알아차리게 하기 위해서 time out을 기다리기보다는 당신은 chat 스크립트의 시작에 ABORT 키워드를 쓰는 것이 더 낫다.

```
$ chat -v ABORT BUSY ABORT 'NO CARRIER' '' ATZ OK ...
```

이와 유사한 경우인데, 당신은 TIMEOUT 옵션을 chat 스크립트에 추가함으로써 timeout 값을 수정할 수 있다. 더 자세한 내용은 chat(8) 매뉴얼 페이지를 읽어보아라. 때때로, 당신은 또한 어떤 종류의 조건적인 chat 스크립트의 실행을 필요로 할 것이다. 한 예로, 원격 시스템의 로긴 프롬프트를 받지 않았을 경우, BREAK를 보내거나 캐리지 리턴을 보내야 할 것이다. 당신은 expect 문자열에 sub-스크립트를 추가함으로써 이를 해결할 수 있다. 그것은 하이픈으로 구분되어 있는 스크립트 전체 그 자체와 같이 send-와 expect-문자열들의 순서로 구성되어있다. sub-스크립트는 기대했던 문자열이 제때에 받아지지 않았을 때에 실행된다. 위의 예에서 우리는 chat 스크립트를 다음과 같이 수정할 수 있다.

```
ogin:-BREAK-ogin: ppp ssword: GaGariN
```

이제, chat는 원격 시스템의 로그인 프롬프트가 보이지 않을때, sub-스크립트는 첫 BREAK를 보내고, 다시 로그인 프롬프트를 기다리게 된다. 프롬프트가 나타났을때, 스크립트는 평소와 같이 계속되고, 만일 그렇지 못하면 에러와 함께 끝이 난다.

## 8.6 Debugging Your PPP Setup

기본적으로, pppd는 모든 경고들과 에러 메세지들을 syslog의 daemon facility에 로그한다. 콘솔에서도 syslog는 이러한 내용들을 그냥 버려버리기 때문에 당신은 syslog.conf의 첫머리에 이것을 파일로 리다이렉트하도록 하는 내용을 더해야한다.

```
  daemon.*                 /var/log/ppp-log
```

만약 당신의 PPP 셋업이 한 번에 성공하지 않는다면 이 로그화일을 들야다봄으로써 무엇이 잘못되고 있는가를 알 수 있다. 이것이 별로 도움이 되지 않는다면 디버그 옵션을 이용해 외부 디버깅 output을 동작하게 할 수 있다. 이것은 syslog에 보내지거나 받은 모든 control packet의 내용을 pppd log를 만든다. 모든 메시지는 deamon facility로 간다.

마지막으로, 가장 과감한 수단은 kdebug 옵션에 의지하는 pppd에 의해 커널-레벨에 의한 디버깅을 가능하게 하는 것이다. 이것은 다음 값들의 bitwise OR 수적인 독립변수를 따른다.: 1은 일반적인 디버그 메세지, 2는

HDLC 프레임으로 들어오는 메세지의 프린팅, 그리고 4는 HDLC 프레임을 통해 나가는 드라이버의 프린트이다. 이러한 커널 디버깅 메세지들은 갈무리하기 위하여, 당신은 /proc/kmsg 화일을 읽도록 syslog 데몬을 실행시키거나 klogd데몬을 실행시켜야 한다. 모두 커널 디버깅을 syslog의 커널 facility로 가도록 지시한다.

## 8.7 IP Configuration Options

IPCP가 링크 설정시간에 있는 두어개의 IP parameter와 교신하기 위하여 사용된다. 일반적으로 각각의 peer는 어떤 값들을 기본값으로부터 바꾸려고 하거나, 어떤 값을 가르키는 IPCP Configuration Request packet을 보낼지도 모른다. 이들의 수신에 의해, 원격 호스트 검사들은 어떤 값들이 설정되어 있는냐에 따라 그것은 인지하거나 거부하게 된다. pppd는 교신을 위한 많은 IPCP 옵션들을 당신에게 준다. 이들의 command line 옵션에 의한 조정에 의해 관해서는 다음에 논의한다.

### 8.7.1 ChoosingIPAddresses

위의 예에서 우리는 c3po에 전화를 걸고 IP 링크를 만들었었다. 링크의 어느 한쪽에서 특정한 IP-주소를 선택하도록 하는 준비가 없었다. 그 대신 local IP-주소로 우리는 vlager의 주소를 선택했고, c3po가 그 자신의 것을 주도록 했다. 그러나 때때로 링크의 한쪽이나 또다른 한쪽에 어떤 주소를 사용할 것인지 조절하는 것은 매우 유용하다. pppd는 이에 해당하는 여러가지 변화를 줄 수 있다. 특정한 주소들을 묻기위하여, 당신은 일반적으로 pppd의 다음 옵션을 줄 수 있다.

```
local addr:remote addr
```

여기서 local_addr과 remote_addr은 4부분으로 이루어진 주소 표기법이거나 호스트네임들로 주어져야한다. 이것은 pppd가 첫번째 주소를 자신의 IP-주소로, 두번째를 peer의 주소로 하도록 만든다. 만약 peer가 둘중 어느 것도 IPCP 교신 중 거부한다면 IP-링크는 성립되지 않을 것이다.

만일 당신이 peer 사용자들의 어떠한 주소도 받아들이지 않고 단지 local address만을 원한다면 remote_addr 부분은 비워두면 된다. 예로, 130.83.4.27이라는 IP주소를 써J서 vlager를 사용하고 싶다면 단지 command line에서 130.83.4.27:라고 하면 된다. 유사하게, remote_addr만을 사용하기 위해서는 local_addr 부분을 비워두면 된다. 기본값으로, pppd는 당신의 호스트네임과 연관된 주소를 사용한다.

어떤 PPP서버들은 많은 수의 클라이언트 사이틀들에의 주소를 유동적으로 관리한다: 주소들은 단지 전화가 걸려왔을때만 결정되고, 다시 로그오프할 때 해제한다. 그런 다이얼업 서버에서는, 당신은 서버가 당신에게 사용할 주소를 물어오기 보다는 pppd가 서버로부터 특정한 IP-주소를 요구하는 지 않는다는 것을 주의해야 한다. 이것은 당신이 local_addr 변수를 나열하지 말아야 한다는 것을 의미한다. 덧붙여, 당신은 local host의 주소를 사용하는 대신 peer가 제공하는 IP-주소를 사용하도록 하는 noipdefault 옵션을 사용해야 한다.

### 8.7.2 Routing Through a PPP

네트워크 인터페이스를 설정한 후에, pppd는 일반적으로 호스트 경로를 peer에게만 셋업할 것이다. 만약 그 원격호스트가 랜에 있다면, 당신은 '뒤'에 있는 peer 역시 호스트에 연결되기를 원할 것이다. ; 다시말해 네트워크 경로가 설정되어야 한다.

우리는 이미 기본옵션으로 사용할 때 기본 경로로 설정한다는 것을 살펴보았다. 이 옵션은 당신이 전화를 건 PPP 서버를 당신의 인터넷 게이트웨이로 사용할 때 매우 유용하다.

그 반대의 경우, 당신의 시스템이 하나의 호스트를 위한 게이트웨이로 사용될때, 역시 쉬운 방법으로 가능하다. 한 예로, loner라는 가정용 컴퓨터의 사용자인 가상 양조장의 일꾼의 경우를 생각해 볼 수 있다. PPP를 통해 vlager로 접속하는 경우, 그는 양조장 subnet의 주소를 사용할 수 있다. vlager에서는, pppd에 이제는 loner에 proxy ARP를 사용할 수 있는 옵션을 인스톨하는 proxyarp 옵션을 줄 수 있다. 이것은 자동적으로 loner가 양조장과 와인양조장에 있는 모든 호스트에 접근할 수 있도록 만든다.

그러나 모든 경우가 저것처럼 쉽지는 않다. 예로, 두 개의 로컬영역 네트워크를 링크하는 것과 같은 경우일 것이다. 이는 명확한 네트워크 경로를 추가해주어야만 한다. 왜냐하면 이들 네트워크들은 그들 자신만의 기본 경로들을 가지고 있기 때문이다. 그 외에도, PPP 링크를 기본 경로로 사용하여 loop를 형성하게 되는 경우에 양쪽의 peer를 가지게 되는 경우, peer들은 연결되어 있는 시간이 끝날 때까지 마치 탁구를 하는 것처럼 어디로 가야할 지를 모르게 된다.

그 예로, 가상 양조장이 그 연결을 어떤 도시에 열어 놓았다고 하자. 양조장의 B 클래스 네트워크 subnet3인 보조는 그들 자신의 IP 네트워크 넘버 191.72.3.0을 이용하여 이더넷을 운영한다. 그들은 고객의 데이타베이스 등등을 업데이트하기 위해 PPP를 통해 양조장의 주 이더넷에 접속하기를 원할 것이다. 다시, vlager는 게이트웨이처럼 행동하고, 그들의 peer는 sub-etha라 불리고 191.72.3.1..의 IP주소를 가지게 된다.

Sub-etha가 vlager에 접속할 때, 그것은 일반적으로 vlager로 향하는 기본 경로 포인트를 만들 것이다. vlager에서 우리는 sub-etha를 거치는 subnet-3를 위한 네트워크 경로를 설치해야 한다. 이를 위해, 우리는 그렇게까지 어렵지 않은 pppd의 형식-ip-up명령-을 사용할 수 있다. 이것은 간단한 쉘 스크립트이거나 PPP 인터페이스가 제대로 설정되고 난 후에 실행되는 /etc/ppp에 위치하는 프로그램이다. 그것이 있을때, 그것은 다음과 같은 파라미터에 의해 실행된다.

```
ip-up iface device speed local addr remote addr
```

여기서 ifcae는 사용되고 있는 네트워크 인터페이스를 명명하고, device는 사용되고 있는 시리얼 장치의 경로명(stdin/stdout이 사용된다면 /dev/tty)이며, speed는 장치의 속도이다. local_addr과 remote_addr는 링크의 양쪽 끝에 :로 나뉘어진 4개 부분으로 된 IP-주소들을 준다. 우리의 경우, ip-up 스크립트는 아마도 다음과 같은 내용을 포함하고 있을 것이다.

```
#!/bin/sh
case $5 in
191.72.3.1)            # this is sub-etha
        route add -net 191.72.3.0 gw 191.72.3.1;;
esac
exit 0
```

이와 비슷한 투로, /etc/ppp/ip-down은 PPP 링크가 죽고 난 후 ip-up가 한 모든 행동을 취소하기 위해 사용된다.

그러나 라우팅 계획은 아직 완벽한 것이 아니다. 우리는 양쪽 PPP 호스트들에 라우팅 테이블의 내용을 설정했지만, 아직 양쪽의 호스트들의 다른 모든 네트워크들은 PPP 링크에 관해서는 아무것도 알지 못한다. 그러나 만약 보조의 모든 호스트들이 sub-etha에 있는 그들의 기본 라우트 지점을 가지고 있고, 모든 양조장의 호스트들이 vlager로 향하는 기본 경로를 가진다하더라도 이것은 그다지 큰 문제는 아니다. 만약 이것이 바로 그런 경우가 아니라면 당신은 gated와 같은 라우팅 데몬만을 사용하면 된다. vlager를 향한 네트워크 경로를 생성하고 난 후에, 라우팅 데몬은 서브넷에 붙어있는 모든 호스트들에 새로운 경로를 알려줄 것이기 때문이다.

## 8.8 Link Control Options

앞에서 우리는 이미 링크의 특성들을 교섭하기 위한 Link Control Protocol,LCP를 다루었었다.

LCP에 의해 조정되는 두개의 가장 중요한 옵션들은 maximun receive unit와 Asynchoronous Control Character Map이다. 많은 LCP옵션들이 있기는 하지만, 여기서 논의하기에는 너무 전문화가 되어 있는 것이 사실이다. 그것들을 더 자세히 알아보려면 RFC-1548을 참조하기 바란다.

흔히, async map으로 불리는 Asynchoronous Control Character Map는 반드시 빠져 있어야 하는 control character들을 구분해야 하는 전화선과 같은 비동시적인 링크들을 위해 사용된다. 예를 들어 어떤 잘못된 설정을 가진 모뎀은 XOFF를 받았을 때 죽을지도 모르기 때문에 소프트웨어 handshake를 위해 XON과 XOFF character들을 아마 당신이 원할 지도 모른다. 다른 후보자들은 Ctrl-](텔넷 escape character)를 포함한다. PPP는 ASCII 코드 0에서 31중 어느 값이라도 async map에 나열하면 빠져나오도록 해준다.

async map은 를 ASCII 널문자(ASCII 31)에 해당하는 최소한의 한 문자를 가진 32-비트 폭의 비트맵이다. 만약 비트가 설정되면, 링크를 통해 그것을 보내기 전에 반응하는 문자는 반드시 escape되었는지 신호를 보낸다. 시작하면서 async map은 모든 control character들이 escape되는 0xffffffff을 맞추어진다.

당신의 peer에게 어떤 것들 중 특별한 것을 제외하고는 모든 control character들을 escape할 필요가 없다고 말하기 위해, asyncmap 옵션을 사용하여 새로운 asyncmap을 pppd에 열거할 수 있다. 예로, 단지 ^s와 ^Q(ASCII 17과 19, 일반적으로 XON과 XOFF)만 escape 되어야 한다고 할 때, 다음과 같은 옵션을 사용할 수 있다.

```
asyncmap 0x000A0000
```

최대 수신 유닛(Maximum Receive Unit), 혹은 MRU는 peer에게 우리가 받고자 하는 최대HDLC 프레임들의 크기를 보낸다. 이 부분에서 MTU(Maximum Transfre Unit:최대 전송 유닛)을 떠올릴 수도 있겠으나, 이 둘은 거의 공통점이 없다. MTU는 커널 네트워킹 디바이스의 파라미터이고, 그 인터페이스가 조절할 수 있는 최대 프레임의 크기를 나타낸다. MRU는 MRU보다 더 큰 어떠한 프레임도 동작하지 않도록 원격의 말미에 주는 단순한 충고 정도이다.; 그럼에도 불구하고 인터페이스는 최소한 1500프레임까지는 수신할 수 있어야 한다.

그러므로 MRU는 선택하는 것은 어떤 링크가 전송을 할 수 있는지가 문제가 아니라 당신이 어떤 최대의 처리량을 주는 것인가 하는 것이다. 만일 링크를 통해 인터액티브한 프로그램을 사용하기를 원한다면, MRU를 제일 낮은 296으로 주는 것이 좋은데 이로 인해 이따금씩 큰 패킷(뭐, FTP 같은)이 당신의 커서를 "점프"하게 하지 않을 것이다. pppd에 MRU를 296으로 맞추라고 하기 위해, 당신은 mru 296이라는 옵션을 주면 된다. 그러나 작은 MRU들은 당신이 VJ 헤더 압축을 사용하지 않도록 하지 않았을 때 (일반적으로는 가능) 제대로 알아먹는다.

pppd는 링크가 종료되기 전에 교환되는 설정 요청들의 최대 숫자들과 같은 교섭 프로세스의 전체적인 동작을 조정하는 두어개의 LCP옵션을 이해한다. 만약 당신이 무엇을 하는지 정확하게 알고 있지 않다면, 그냥 놔두는 것이 더 낫다.

마지막으로, LCP 에코 메세지들에 적용되는 두 가지 옵션이 있다. PPP는 두가지 메세지들, Echo 요청과 Echo 응답, 을 정의한다. pppd는 이 모양을 어떤 링크가 계속 동작중인지 점검하기 위해 사용한다. 당신은 이것들을 초로 표시하는 시간과 함께 lcp-echo-interval 옵션을 통해 가능하게 할 수 있다. 이 시간 간격 동안 원격 호스트로부터 어떠란 수신도 없다면 pppd는 Echo 요청을 발생시키고, peer가 Echo 응답을 보내오기를 기다린다. 만약 peer가 응답을 해오지 않는다면, 링크는 몇 번의 요청을 보낸 후에 끊어진다. 이 숫자는 lcp-echo-failure 옵션을 사용하여 지정할 수 있다. 기본적으로 이 내용들은 두 가지 모두 동작하지 않도록 되어 있다.

## 8.9 General Security Considerations

잘못 설정된 PPP 데몬은 보안구멍으로 인해 큰 재앙을 초래한다. 이것은 아무 사용자나 당신의 이더넷으로 들어오도록 하는 최악의 경우이다. 이 장에서는 당신의 PPP 설정을 안전하게 하는 몇가지 측정을 해보도록 하겠다.

pppd의 문제점 중 하나는 네트워크 디바이스와 라우팅 테이블을 설정하는데 root권한을 요구하는 것이다. 일반적으로 당신은 이 문제를 root로 실행시켜서 풀 것이다. 그러나 pppd는 사용자들에게 여러가지 보안과 관련된 옵션들을 사용하도록 허락한다. 이러한 옵션들을 교묘하게 사용한 사용자들의 공격으로부터 보호하기 위해, 지난 장에서 설명한 전체적인 기본 옵션들을 /etc/ppp/options 화일에 적어주는 것을 제안한다. 그들중 authentification 옵션과 같은 것들은 사용자들에 의해 덮어 쓰여질 수 없기 때문에 교묘한 술책에 믿음직한 보호를 해 준다.

물론, 당신 자신도 역시 시스템으로부터 PPP를 말하는 것을 보호해야 한다. 다른 사람들처럼 모든 호스트들을 밀어내기 위하여, 항상 peer로 부터 어떤 종류의 인증을 갖고 있어야 한다. 추가적으로 당신은 외부의 호스트들이 그들이 선택하는 어떠한 IP-주소도 사용 못하도록 해야 하지만 최소한 몇 개는 제한해야 한다. 다음 장에서 이 내용을 다룬다.

## 8.10 Authentication with PPP

### 8.10.1 CHAP versus PAP

PPP를 사용할 때 각각의 시스템은 아마 시스템의 peer에게 그 자신을 두가지 인증 프로토콜 중 하나에 의한 인증을 요구할 지도 모른다. 이 둘이 바로 Password Authentication Protocol(PAP), 그리고 Challenge Handshake Authentication Protocol(CHAP)이다. 접속이 되었을때, 링크의 각각 끝은 다른 한쪽에 그 자신의 인증을 전화를 건쪽이든 전화를 받는 쪽이든 요구하게 된다. 그러므로 이제 아래에서 인증하는 쪽과 인증받는 쪽을 구분하게 될 때 '클라이언트'와 '서버'의 개념을 다소 느슨하게 설명해도 될 것이다. PPP 데몬은 그 peer에게 바람직한 인증 프로토콜을 확인하는 또다른 LCP 설정 요청을 보냄으로써 인증을 물을 수 있다.

PAP는 일반적인 로그인 과정과 같은 간단한 방법으로 동작한다. 클라이언트는 사용자명과 (일반적으로는 암호화된)암호를 보내어 그 자신을 증명하고 서버는 그 자신의 비밀 데이타베이스와 그것을 비교해 본다. 이 기술은 시리얼 라인을 통해 암호를 알아내려 하는 엿듣는 사람과 반복되는 '시행착오' 공격에 취약하다.

CHAP는 이러한 약점을 가지고 있지 않다. CHAP는 인증자(즉, 서버)는 무작위로 만들어진 "challenge" 문자열을 그의 호스트명과 함께 클라이언트에 보낸다. 클라이언트는 호스트명을 적절한 암호, challenge와 조합해서,찾기위해 사용하고, 그 문자열을 단 하나의 hashing 함수를 이용해서 암호화한다. 그 결과는 클라이언트의 호스트명과 함께 서버로 되돌아간다. 그 서버는 이제 같은 계산을 수행하고, 같은 결과가 되돌아왔는 지를 확인하고 클라이언트를 인지한다.

CHAP의 다른 특징은, 단지 클라이언트가 접속을 시작할때의 인증만을 원하지 않고, 클라이언트가 다른 침입자에 의해 바뀌지는 않았는지, 예를 들면 전화선을 바꿔친다든지하는, 확인하기 위해 특정한 시간간격을 두고 challenge들을 클라이언트에 보낸다.

pppd는 상대적으로 /etc/ppp/chap-secrets와 ppp-secrets라는 다른 화일로 CHAP와 PAP를 위한 두개의 비밀키를 유지한다. 이중 하나나 다른 화일을 통해 다른 원격 서버로 들어감에 의해, 당신은 CHAP나 PAP에 의해 아주 훌륭한 콘트롤을 받게 되고 인증을 받는다.

기본적으로, pppd는 원격서버로부터 인증을 요구하지 않지만 원격서버에서 인증요구가 왔을때는 거기에 동의하여 인증을 하게 된다. CHAP가 PAP보다는 훨씬 더 강력하기 때문에, pppd는 가능하면 CHAP를 사용하려한다. 만약 peer가 그것을 지원하지 않는다거나 만약 pppd가 원격 시스템의 chap-secret 화일로부터 CHAP secret를 찾지 못한다면, PAP로 바뀐다. 만약 PAP 또한 가지고 있지 않다면 인증을 거부하게 되고, 결론적으로 접속은 끊어지게 된다.

이러한 양식은 여러가지로 수정된다. 예로, 주어진 인증 키워드에 대해, pppd는 peer에peer 그 자신이 인증을 하도록 요구하게 된다. pppd는 CHAP나 PAP 데이타베이스에 peer를 위한 secret가 존재하는 하는 한 CHAP나 PAP를 사용하도록 동의할 것이다. 특정한 인증 프로토콜을 켜고 끄는 다른 옵션도 있다. 여기서는 그 내용은 다루지 않겠다. pppd(8) 메뉴얼을 참고하도록 하라.

만약 당신이 PPP를 통한 모든 시스템에 당신과 그들 자신이 인증하도록 하려면, 당신은반드시 auth 옵션을 전체적인 /etc/ppp/options 화일에 넣도록 하고, 각각의 시스템에 해당하는 비밀번호들을 chap-secret화일이 넣어야 한다. 만약 시스템이 CHAP를 지원하지 않는다면, 그를 위한 pap-secret 화일을 첨부하라. 이 방법으로 당신은 당신의 호스트에 　증받지 않은 어떤 시스템도 접근하지 못하게 할 수 있다.

다음 두 섹션은 PPP의 두가지 secret 화일인 pap-secrets와 Chap-secrets화일에 관해서 다룰 것이다. 이들은 /etc/ppp에 위치하며 클라이언트들, 서버들, 비밀번호들, 세부분으로 이루어진 내용을 담고 있으며 선택적으로 딸린 IP-주소들의 리스트를 포함한다. 클라이언트와 서버 필드의 해석은 CHAP와 PAP는 서로 다르며, peer에게 어떤 방식으로 인증을 하느냐에 의존하거나 혹은 서버가 우리와 함께 인증을 요구하느냐에 달려있다.

### 8.10.2 The CHAP Secrets File

CHAP를 사용한 서버와 그 자신이 인증을 하기 위해서, pppd는 pap-secrets화일에서 로컬 호스트명과 같은 클라이언트 필드를 찾고, CHAP Challenge에서 보내진 원격 호스트명과 같은 서버 필드를 찾는다. peer에게 그 자신을 인증하도록 다시 질의하였을때, 역할은 단순히 반대로 바뀐다:pppd는 원격 호스트명과 같은 클라이언

트 필드를 찾고,(클라이언트의 CHAP 응답에서 보내진) 로컬 호스트명과 같은 서버필드를 찾는다.

아래는 vlager를 위한 간단한 chap-secrets 화일이다.

```
# CHAP secrets for vlager.vbrew.com
#
# client          server            secret                addrs
#---------------------------------------------------------------------
vlager.vbrew.com  c3po.lucas.com    "Use The Source Luke" vlager.vbr
c3po.lucas.com    vlager.vbrew.com  "riverrun, pasteve"   c3po.lucas
*                 vlager.vbrew.com  "VeryStupidPassword"  pub.vbrew.
```

c3po와의 PPP연결이 이루어 졌을때 c3po는 vlager에게 CHAP Challenge를 보내어 CHAP를 사용해 그 자신을 인증하도록 한다. pppd는 이제 vlager.vbrew.com과 같은 클라이언트 필드를 찾아 chap-secrets 화일을 검색하고 c3po.lucas.com과 같은 서버필드를 찾고, 위에서 보여진 첫번째 줄에서 찾아낸다. 이제 challenge 문자열과 secret(Use The Source Luke)로부터 CHAP 응답이 생겨나고, 그것을 c3po로 보낸다.

동시에 pppd는 특정한 challenge 문자열을 포함하고 있고 확실히 검증된 호스트명인 vlager.vbrew.com를 포함하고 있는 c3po를 위한 CHAP challenge를 작성한다. c3po는 방금 우리가 이야기했던 방식으로 CHAP 응답을 만들고, 그것을 vlager에 되돌려 준다. 이제 pppd는 CHAP 응답으로부터 클라이언트 호스트명(c3po.vbrew.com)을 뽑아내서, 클라이언트로서 c3po가 맞는 부분이 있는지를 chap-secret 화일로부터 찾고, 서버로서 vlager가 있는지를 찾는다. 두분째 줄이 이 역할을 하고, 그래서 pppd는 CHAP challenge와 secert riverrun, pasteve를 암호화 하고 그 결과를 c3po의 CHAP 응답과 비교한다.

네번째 선택적인 필드는 첫번째 필드에서 명명된 클라이언트들에 해당하는 IP-주소들을 나열한다. 그 주소들은 4개 부분을 이루어졌거나, resolver에 의해 인식되는 호스트명들일것이다. 예로 만약 c3po가 IPCP 교섭중에 리스트에 있지 않는 IP 주소를 사용하기를 원한다면, 그 요구는 거절되고, IPCP는 닫혀버릴 것이다. 위에서 보여진 예제화일에서, 그러므로 c3po는 그 자신의 IP-주소를 사용하는 것이 제한된다. 만약 주소 필드가 비어있다면 어떠한 주소들도 허용된다.; -값은 클라이언트 또한 사용하지 못하도록 방지해준다.

chap-secrets 화일의 세번째 줄은 어떠한 호스트명도 클라이언트, 서버필드의 *에 적합하기 때문에 어떤 호스트라도 vlager에 PPP 링크를 만들수 있도록 해준다. 유일한 요구사항은 그것이 secret를 알고, pub.vbrew.com을 사용해야 한다는 것이다. pppd는 항상 서버/클라이언트가 짝을 이루도록 값을 넣기 때문에 와일드카드 호스트명으로 시작하는 entry들은 아마 secrets 화일 어디에서도 보일지도 모른다.

이제 pppd가 어떻게 secrets 화일로부터 호스트명에 이르게 되는지 알아보자. 전에 설명했던 대로, 원격 호스트명은 항상 CHAP Challenge나 응답패킷속에 있는 peer에 의해서 제공된다. 로컬 호스트명은 기본적으로 gethostname(2) 함수에 의해서 전해진다. 만약 당신이 시스템명을 당신의 제한된 호스트명으로 준다면 다음과 같이 domain 옵션을 사용해서 pppd에 domain name을 주어야 한다.

```
# pppd ...domain vbrew.com
```

이것은 Brewery의 domain name을 모든 인증과 관련된 활동들과 함께 vlager에 덧붙인다. 로컬 호스트명을 위한 progpppd의 수정된 다른 옵션들은 usehostname과 name이다. 당신이 "local:varremote"라는 command line 명령을 통해 로컬 IP 주소를 주었을 때, local은 4개의 :로 구분된 숫자를 대신하는 name이고, pppd는 이를 로컬 호스트명으로 사용할 것이다. 더 자세한 예는 pppd(8) 매뉴얼 페이지를 참고하기 바란다.

### 8.10.3 The PAP Secrets File

PAP secerts 화일은 CHAP와 매우 비슷하다. 처음의 두 필드는 항상 사용자명과 서버명을 담고 있다; 세번째는 PAP secret를 담고 있다. 원격이 인증 요청을 보내올때, pppd는 로컬 호스트명과 서버필드가 같은지를 사용하며, 사용자 필드는 요청에서 보내진 사용자명과 같은 지를 검사한다. peer와 자신을 위한 인증이 이루어질

때, pppd는 라인을 통해 로컬 사용자명과 같은 사용자 필드와 함께 온 온 secret를 선택하고 서버필드와 같은 원격 호스트명을 사용한다.

예제 PAP secrets 화일은 아마 다음과 같을 것이다.

```
# /etc/ppp/pap-secrets
#
# user          server          secret          addrs
vlager-pap      c3po            cresspahl       vlager.vbrew.com
c3po            vlager          DonaldGNUth     c3po.lucas.com
```

첫번째 줄은 우리 자신이 c3po에 인증하기 위해 사용된다. 두번째 줄은 어떻게 c3po로 이름지어진 사용자가 그 자신이 우리와 함께 인증되는 가를 나타내 준다.

첫번째 컬럼에 있는 vlager-pap라는 이름은 우리가 c3po로 보내는 사용자명이다. 기본적으로 pppd는 로컬 호스트명을 사용자명으로 선택한다. 그러나 당신은 name 뒤에 따라오는 user 옵션을 사용하여 다른 이름을 열거할 수 있다.

peer와의 인증을 위해 pap-secrets 화일에서 entry를 가져 올 때, pppd는 원격 호스트의 이름을 알아야 한다. 그것을 알아낼 방법이 없기 때문에 당신은 반드시 peer의 호스트명뒤에 따라오는 remotename이라는 키워드를 사용해서 command line에서 열거해야한다. 예로 위에서 보여진 예에서 c3po로의 인증을 위해, 우리는 반드시 다음 옵션을 pppd의 command line에 적어주어야 한다.

```
# pppd ...domain vbrew.com
```

네번째 부분에(그리고 그 뒤에 오는 모든 부분은), CHAP에서 그랬던 것처럼 당신이 어떤 IP-주소들을 특정한 호스트에 허용할 것인지를 나열할 수 있다. peer는 이제 리스트에 있는 주소들만을 요청할 수 있다. 예제화일에서, 우리는 c3po에 그것의 실제 IP 주소를 사용하도록 하고 있다.

PAP는 인증방법으로는 다소 약하다는 것을 명심하라. 그리고 가능하다면 CHAP를 사용하기를 권장한다. 그래서 여기에서는 PAP에 관해 폭넓게 다루지는 않을 것이다. PAP를 사용하는데 관심이 있다면 pppd(8) 매뉴얼 페이지에서 더 많은 PAP의 특징이 관해서 찾아보도록 하라.

## 8.11 Configuration a PPP Server

pppd를 서버로 쓰기 위해서는 단지 command line에서 적당한 옵션을 추가해 주기만 하면 된다. 가장 이상적으로는, 당신은 ppp라는 특별한 계정을 하나 만들어서 그 계정의 로그인 쉘을 pppd 옵션을 주는 스크립트로 주면 된다. 한 예로 당신은 /etc/passwd에 다음과 같이 추가하면 된다.

```
ppp:*:500:200:Public PPP Account:/tmp:/etc/ppp/ppplogin
```

물론, 당신은 위의 예에서 보여진 것과는 다른 uid와 gid를 원할지도 모른다. 또 passwd 명령을 이용해서 위의 계정의 비밀번호를 부여해야 한다.

ppp 로그인 스크립트는 아마 다음과 같을 것이다.

```
#!/bin/sh
# ppplogin - script to fire up pppd on login
mesg n
stty -echo
exec pppd -detach silent modem crtscts
```

mseg 명령은 다른 사용자들이 사용중인 tty에 쓰기를 불가능하게 한다. 예로, write 명령을 들 수 있다. stty 명령은 문자의 echo를 끈다. 이것은 peer가 보낸 내용이 ehco되어 되돌아 올수도 있기 때문에 필요하다. 위에서 주어진 가장 중요한 pppd 옵션은 -detach 명령이다. 이것은 control중인 tty에서 pppd가 detach하는 것을 막기 때문이다. 만약 우리가 이 옵션을 사용하지 않는다면 pppd는 백그라운드로 들어가게 되고, 어떤 쉘 스크립트는 끝나게 될지도 모른다. 이로 인해 시리얼 라인이 끊기고 접속이 해제된다. silent 옵션은 pppd가 시작하기 전에 전화거는 쪽 시스템이 패킷을 받을 때까지 기다리도록 하게 한다. 이것은 전화거는 쪽의 시스템이 PPP 클라이언트의 시작이 너무 늦을 때 timeout이 되어 끊기는 것을 방지해 준다. 모뎀은 만약 peer가 접속에서 끊기지 않았는지 감시하는 DTR 라인의 pppd watch를 만들고, crtscts는 하드웨어 handshake를 작동시킨다.

이러한 옵션들 이외에도, 당신은 이 외의 더 강력한 인증을 원할 수도 있다, 한 예로 pppd의 command line에서 사용자의 인증을 열거하거나, 전체적인 옵션에서 설정할 수 도 있다. man 페이지는 더 많은 개개의 인증 프로토콜에 관한 옵션을 켜고 끄는 것에 대한 논의를 하고 있다.