한글 FreeBSD 핸드북

Korea FreeBSD Users Group – http://www.kr.FreeBSD.org/

Advanced Networking: 고급 네트워킹

이 영 옥 <rick@rickinc.com> 역.

2004 년 8 월 7 일.
문서갱신번호 1.1

본 문서는 FreeBSD Documentation Project 에서 발행하는 FreeBSD Handbook 을 번역한 것으로 2004 년 8 월 7 일자 수정본을 기준으로 합니다. FreeBSD Handbook 은 지속적으로 갱신되기 때문에, 본 문서의 발간 이후에 갱신된 내용이 무엇인지 알고자 한다면, CVS Repository 를 조회하여 해당일 이후에 갱신된 내용만을 쉽게 찾아볼 수 있습니다. 또한 표지에 표시되는 날짜와 문서갱신번호는 이 문서의 갱신상태를 나타냅니다. 날짜는 번역의 대상이 되는 오리지날 핸드북의 최종 수정일자를 표시하며, 문서갱신번호는 문서에 수정이 있을 때 증가하므로 사용자가 이 문서의 갱신상태를 쉽게 파악할 수 있도록 합니다. 본 문서는 이영옥이 관리, 배포하므로 내용에 오탈자 및 오역이 발견될 경우엔 역자에게 알려주어 수정될 수 있도록 해 주십시오.

24 장 고급 네트워킹

24.1 개요

이번 장에서는 고급 네트워크에 대해 다룬다.

이번 장을 읽고 아래와 같은 사항을 알 수 있다:

- 게이트웨이와 라우트의 기본
- IEEE802.11과 블루투스 장치는 어떻게 설정하는가
- FreeBSD를 어떻게 브리지처럼 동작 시키는가
- diskless 머신이 네트워크로 부팅하도록 어떻게 설정하는가
- 네트워크 주소 변환(NAT)은 어떻게 설정하는가
- PLP로 두 대의 컴퓨터를 어떻게 연결하는가
- FreeBSD 머신에서 IPv6 설정
- FreeBSD 5.X로 ATM 설정

이번 장을 읽기 전에 알고 있어야 할 사항:

- /etc/rc 스크립트의 기본을 알고 있어야 된다.
- 기본 네트워크 용어에 익숙해야 된다.
- 새로운 FreeBSD 커널 설정과 설치 (8장)
- 소프트웨어 설치 (4장)

24.2 게이트웨이와 라우트(경로)

네트워크를 통해서 머신이 다른 머신을 찾을 수 있도록, 한 머신에서 다른 머신을 어떻게 찾는지 설명된 메커니즘이 있어야 된다. 이것을 라우팅(경로) 이라고 한다. "경로"는 목적지와 게이트웨이 주소 쌍으로 정의되어 있다. 이 주소 쌍은 여러분이 특정 *목적지*에 접속한다면 이 *게이트웨이*를 경유하여 통신하는 것을 보여준다. 목적지에는 각각의 호스트와 서브 넷 그리고 "기본 라우트" 이렇게 3 종류가 있다. "기본 라우트(경로)"는 적용할 다른 경로가 없을 때 사용된다. 나중에 기본 라우트에 대해 좀더 이야기할 것이다. 그리고 게이트웨이에도 역시 3 종류가 있다: 각각의 호스트와 인터페이스("링크"라고 부르기도 하는) 그리고 이더넷 하드웨어 주소(맥(MAC) 주소).

24.2.1 예제

다른 관점에서 라우터를 설명하기 위해 다음 예제의 netstat 를 활용한다:

```
% netstat -r
Routing tables

Destination        Gateway            Flags     Refs      Use      Netif Expire

default            outside-gw         UGSc        37      418      ppp0
localhost          localhost          UH           0      181      lo0
test0              0:e0:b5:36:cf:4f   UHLW         5    63288      ed0     77
10.20.30.255       link#1             UHLW         1     2421
example.com        link#1             UC           0        0
host1              0:e0:a8:37:8:1e    UHLW         3     4601      lo0
host2              0:e0:a8:37:8:1e    UHLW         0        5      lo0 =>
host2.example.com  link#1             UC           0        0
224                link#1             UC           0        0
```

처음 두 라인은 기본 라우트(다음 섹션에서 다루게 될)와 localhost 경로를 명시한다.

이 라우팅 테이블에서 *localhost* 의 인터페이스(*Netif* 컬럼)는 lo0 이고 루프백 장치로 알려져 있다. 이것은 시작과 끝이 같기 때문에 모든 트래픽을 LAN 을 통해 외부로 보내지 않고 내부로 보낸다.

그 다음은 0:e0:으로 시작되는 주소다. 이들은 맥 주소로 알려져 있는 이더넷 하드웨어 주소다. FreeBSD 는 자동으로 로컬 이더넷의 임의의 호스트(예제에서는 test0)를 찾아서 이 호스트로 가는 경로를 직접 이더넷 인터페이스 ed0 에 추가한다. 일정 시간동안 호스트로부터 응답이 없을 때 사용되는 이런 종류의 라우트와 관련된 타임 아웃도(*Expire* 칼럼)있다. 타임 아웃이 발생하면 이 호스트의 경로는 자동으로 삭제된다. 이들 호스트는 로컬 호스트에서 가장 짧은 경로를 계산하는 RIP 라는 메커니즘을 사용하여 식별된다.

FreeBSD 도 로컬 서브네트워크(10.20.30.255 는 서브네트워크 10.20.30 의 브로드케스트 주소이고 example.com 은 이 서브네트워크에 연결되어 있는 도메인 이름이다)의 경로 정보를 추가할 수 있다. 명시된 *link#1* 은 머신의 첫 번째 이더넷 카드를 의미한다. 이곳에 추가적으로 지정된 인터페이스가 없음을 의미한다.

이들 두 그룹(로컬 네트워크 호스트와 로컬 서브네트워크) 모두 **routed** 라는 데몬에 의해 자동적으로 경로가 설정된다. **routed** 가 동작하지 않는다면 정적으로 정의한(즉 입력한) 경로만 존재하게 된다.

host1 라인은 이더넷 주소로 알 수 있듯이 우리의 호스트를 의미한다. 우리가 호스트에 데이터를 보내면 FreeBSD 는 이더넷 인터페이스로 보내기 보다는 루프백 인터페이스(lo0)를 사용하는 것을 알고 있다.

두 번째 *host2* 라인은 우리가 ifconfig(8) 엘리어스(사용해야 되는 이유는 이더넷 섹션을 본다)를 사용할 때 발생하는 예제다. lo0 인터페이스 이후의 **=>** 표시는 루프백(이 주소도 로컬 호스트를 의미하기 때문이다)을 사용한다는 것이지만 엄밀히 말해서 엘리어스다. 이러한 경로는 엘리어스를 지원하는 호스트만 보여준다. 로컬 네트워크의 다른 호스트는 이런 경로로 *link#1* 라인만 가지고 있다.

다른 섹션에서 다루게 될 마지막 라인은(목적지 서브네트워크 224) 멀티캐스팅과 관련 있다.

마지막으로 각 경로의 다양한 속성은 *Flags* 칼럼에서 볼 수 있다. 아래 내용은 이들 플래그 몇 개와 의미를 설명한 것이다:

U	Up: 이 경로가 활성화됐다.
H	Host: 이 경로의 목적지는 싱글 호스트다.
G	Gateway: 어디서 보낸 것인지 확인하기 위해 이 목적지의 원격 시스템에 무엇이든 보낸다.

S	Static: 이 경로는 시스템이 자동으로 생성한 것이 아니고 수동으로 설정한 것이다
C	Clone: 머신에 연결했을 때 이 경로로 기반으로 새로운 경로가 만들어진다. 이런 종류의 경로는 보통 로컬 네트워크에 사용된다.
W	WasCloned: 로컬 네트워크(클론) 경로를 기반으로 자동으로 설정된 경로를 나타낸다.
L	Link: 이더넷 하드웨어에 대한 참조를 포함하는 경로.

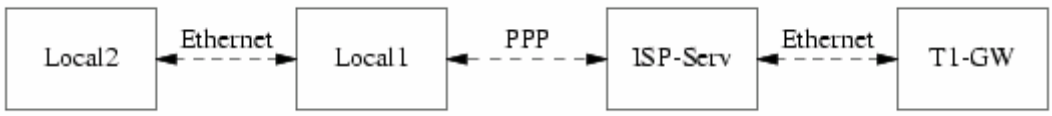
24.2.2 기본 라우트

로컬 시스템이 원격호스트에 연결할 때 경로를 결정하기 위해 알고 있는 경로가 있는지 라우팅 테이블을 체크한다. 도달하는 방법을 알고 있는 서브네트워크(복제된 경로)에 원격 호스트가 있다면 시스템은 이 인터페이스에 연결할 수 있는지 체크한다.

알고 있는 모든 경로로 실패했다면 시스템에는 마지막 기본 라우트(경로)라는 옵션이 하나 있다. 이 경로는 특별한 종류의 게이트웨이 경로(보통 시스템에 유일한)로서 플래그 필드에 항상 *c*가 표시되어 있다. 로컬 네트워크의 호스트를 위해 이 게이트웨이는 머신이 외부(PPP 링크, DSL, 케이블 모뎀, T1 또는 다른 네트워크 인터페이스를 통해서)로 직접 연결할 수 있도록 설정되어 있다.

외부로 나가는 게이트웨이 역할을 하는 머신에 기본 라우트를 설정한다면 기본 라우트는 인터넷 서비스 공급자(ISP)의 게이트웨이 머신이 된다.

기본 라우트의 예제를 보자. 이것은 일반적인 설정이다:



호스트 Local 1 과 Local 2 는 여러분의 사이트에 있다. Local 1 은 다이얼-업 PPP 를 통해 ISP 에 연결되어 있다. 이 PPP 서버 컴퓨터는 로컬 영역 네트워크를 지나서 다른 게이트웨이 컴퓨터에 연결되어 있고 외부 인터페이스를 거쳐 ISP 인터넷 망에 연결된다.

각 머신의 기본 게이트웨이는 아래와 같다:

호스트	기본 게이트웨이	인터페이스
로컬 2	로컬 1	이더넷

로컬 1	T1-GW	PPP

일반적으로 "왜 로컬 1 의 기본 게이트웨이를 ISP 의 서버로 설정하지 않고 T1-GW 로 설정하는가?"라고 자주 듣게 되었다.

PPP 인터페이스는 내부 연결에 ISP 의 로컬 네트워크(여러분 쪽의) 주소를 사용하기 때문에 ISP 로컬 네트워크에 있는 모든 머신의 경로는 자동으로 생성되고 있다. 그래서 T1-GW 머신에 어떻게 도달하는지 이미 알고 있기 때문에 ISP 서버에 데이터를 보내는 중간 단계가 필요 없어진다.

마지막으로 주소 X.X.X.1 을 로컬 네트워크의 기본 게이트웨이로 사용하는 것이 일반적이다. 그래서(같은 예제를 사용하여) 로컬 C 클래스 주소 공간이 10.20.30 이고 ISP 가 10.9.9 를 사용하였다면 기본 라우트는 다음과 같다:

호스트	기본 라우트
로컬 2 (10.20.30.2)	로컬 1 (10.20.30.1)
로컬 1 (10.20.30.1, 10.9,9,30)	T1-GW (10.9.9.1)

/etc/rc.conf 파일을 통해 기본 라우트를 쉽게 지정할 수 있다. 예제에서 Local2 머신의 /etc/rc.conf 에 다음 라인을 추가한다:

```
defaultrouter="10.20.30.1"
```

아니면 route(8) 명령으로 직접 지정할 수도 있다:

```
# route add default 10.20.30.1
```

24.2.3 이중 네트워크 호스트

두 개의 다른 네트워크에 있는 호스트에 관해 우리가 다루어야 할 한가지 설정이 있다. 기술적으로 게이트웨이 기능을 하는 어떤 머신(PPP 연결을 사용하는 위의 예제에서)이라도 이중 네트워크 호스트(dual-homed host)라고 할 수 있다. 그러나 이 용어는 두 개의 로컬 영역 네트워크에 연결되어 있는 머신을 언급할 때만 사용한다.

머신에 두 개의 이더넷 카드가 있으면 각 이더넷 카드는 서브네트워크로 나누어진 주소를

가지고 있다. 그렇지 않으면 머신은 오직 하나의 이더넷 카드를 가지고 있고 ifconfig(8) 엘리어싱을 사용할 것이다. 전자는 물리적으로 나누어진 두 개의 이더넷 네트워크에 사용되고 후자는 물리적인 하나의 네트워크지만 논리적으로 두 개로 나누어진 서브네트워크에서 사용된다.

양쪽에 라우팅 테이블을 설정하여 각 서브네트워크는 이 머신이 다른 서브네트워크의 게이트웨이(내부 라우트)로 설정되었음을 알고 있다. 두 서브네트워크의 라우터로 동작하는 머신을 설정하는 것은 패킷 필터링 또는 어느 한 방향이나 양쪽으로 방화벽이 필요할 때 가끔 사용된다.

이 머신이 실제로 두 인터페이스 사이의 패킷을 포워드 하기를 원한다면 FreeBSD 가 이 기능을 활성화하도록 해야 된다.

24.2.4 라우터 구축

네트워크 라우터는 한쪽 인터페이스에서 다른 쪽으로 패킷을 포워드하는 단순한 시스템이다. 인터넷 표준과 바람직한 엔니지어링 관습이 FreeBSD 프로젝트가 기본적으로 FreeBSD 에서 라우터 기능을 활성화하지 못하게 했다. 이 기능은 rc.conf(5)에서 다음 변수를 *YES* 로 변경하여 활성화할 수 있다:

```
gateway_enable=YES          # Set to YES if this host will be a gateway
```

이 옵션은 sysctl(8)변수 *net.inet.ip.forwarding* 를 *1* 로 설정한다. 일시적으로 라우팅을 멈추려면 이 설정을 잠시 *0* 으로 변경할 수 있다.

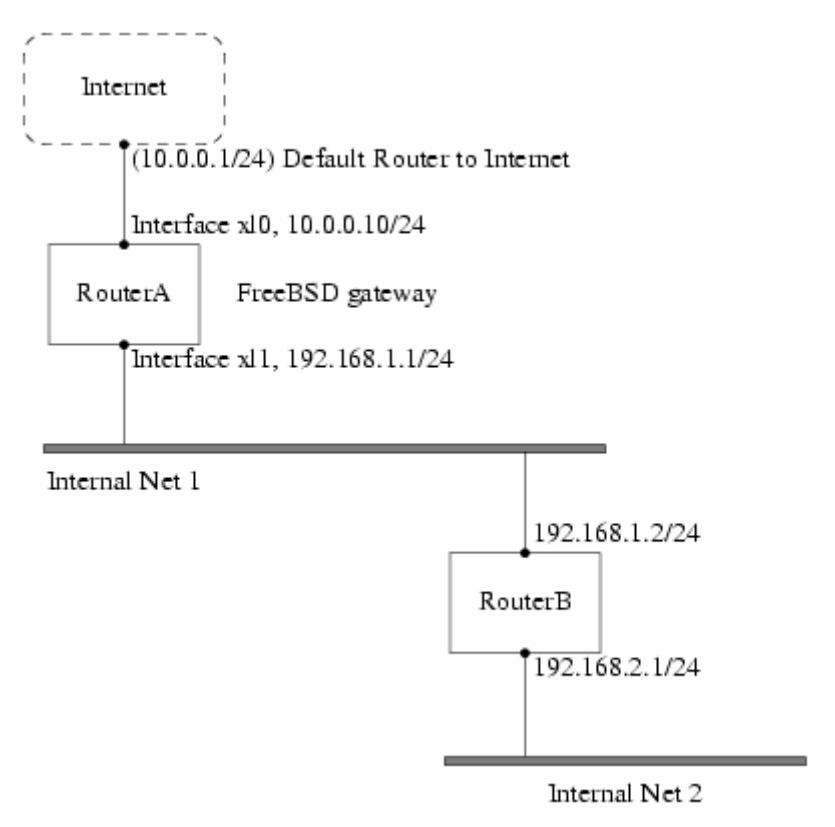
새로운 라우터는 데이터를 어디로 보낼지 알아야 한다. 네트워크가 아주 단순하다면 정적인 경로를 사용할 수 있다. FreeBSD도 RIP(버전 1 과 버전 2 두 가지로) 및 IRDP와 통신하는 표준 BSD 라우팅 데몬 routed(8)을 가지고 있다. BGP v4, OSPF v2 와 매우 정교한 다른 라우팅 프로토콜은 net/zebra 패키지로 사용할 수 있다. 더욱 복잡한 네트워크 라우팅 솔루션으로 GateD와 같은 상용 제품도 사용할 수 있다.

FreeBSD 를 라우터로 설정했더라도 라우터에 필요한 완벽한 인터넷 표준을 따르지 않는다. 그러나 일반적으로 사용하기에는 적당하다.

24.2.5 정적인 경로 설정

24.2.5.1 직접 설정

다음과 같은 네트워크를 가지고 있다고 가정한다:



이 시나리오에서 RouterA 는 나머지 인터넷에 라우터로 동작하는 FreeBSD 머신이다. 이 머신은 외부로 연결할 수 있도록 10.0.0.1 로 설정된 기본 라우트를 가지고 있다. 그리고 RouterB 는 이미 적절하게 설정되어 있어서 필요한 곳에 어떻게 도달하는지 알고 있다고 가정한다(이 그림에서 이렇게 설정하는 것은 간단하다. 게이트웨이로 192.168.1.1 을 사용하도록 RouterB 의 기본 라우트를 추가한다.)

RouterA 의 라우팅 테이블을 확인해 보면 다음과 비슷할 것이다:

```
% netstat -nr
Routing tables

Internet:
Destination        Gateway            Flags      Refs      Use  Netif Expire
default            10.0.0.1           UGS           0    49378    xl0
127.0.0.1          127.0.0.1          UH            0        6    lo0
10.0.0.0/24        link#1             UC            0        0    xl0
192.168.1.0/24     link#2             UC            0        0    xl1
```

현재 RouterA 의 라우팅 테이블로는 Internal Net 2 에 도달할 수 없다. 192.168.2.0/24 로의 경로가 없지만 직접 추가할 수 있다. 다음 명령은 192.168.1.2 를 다음 홉(hop)으로 사용하여 Internal Net 2 네트워크를 RouterA 의 라우팅 테이블에 추가한다:

```
# route add -net 192.168.2.0/24 192.168.1.2
```

이제 RouterA 는 192.168.2.0/24 네트워크의 호스트에 도달할 수 있다.

24.2.5.2 설정 유지

위 예제는 시스템에 정적인 라우트를 설정하기에 적당하다. 그러나 FreeBSD 머신을 재부팅 한면 라우팅 정보가 유지되지 않는다. 해결할 수 있는 방법은 /etc/rc.conf 파일에 고정 경로를 추가하는 것이다:

```
# Add Internal Net 2 as a static route
static_routes="internalnet2"
route_internalnet2="-net 192.168.2.0/24 192.168.1.2"
```

static_routes 설정 변수는 스페이스(space)로 나눠진 문자열 리스트다. 각 문자열은 경로 이름을 의미한다. 위의 예제는 *static_routes* 하나만 있고 문자열은 *internalnet2* 다. 그리고 route(8) 명령에 지정하는 모든 설정 매개변수를 지정하는 곳에 *route_internalnet2* 라는 설정 변수를 추가한다. 위의 예제에 다음 명령을 사용한다:

```
# route add -net 192.168.2.0/24 192.168.1.2
```

그래서 *"-net 192.168.2.0/24 192.168.1.2"*가 필요하다.

위에서 말했듯이 *static_routes*에 하나 이상의 문자열을 지정할 수 있다. 따라서 여러 개의 정적인 경로를 추가할 수 있다. 다음 라인은 192.168.0.0/24 와 192.168.1.0/24 네트워크에 정적인 경로를 추가하는 예를 보여 준다.

```
static_routes="net1 net2"
route_net1="-net 192.168.0.0/24 192.168.0.1"
route_net2="-net 192.168.1.0/24 192.168.1.1"
```

24.2.6 라우팅 전달

외부와의 경로를 어떻게 정의하면 되는지 이미 설명했지만 외부에서 우리를 어떻게 찾는지 설명하지 않았다.

라우팅 테이블 설정 방법을 알고 있기 때문에 특정 주소(예제에서 C 클래스 서브넷)로 향하는 모든 트래픽을 내부로 포워드 하는 네트워크의 특정 호스트로 보낼 수 있다.

여러분의 사이트에 주소를 할당할 때 서비스 공급자가 그들의 라우팅 테이블을 설정하기 때문에 여러분 서브네트워크로 향하는 모든 트래픽은 PPP 링크를 통해 여러분의 사이트로 보내진다. 그러나 다른 나라에 있는 사이트에서 여러분의 ISP 로 데이터를 어떻게 보내는가?

할당된 모든 주소에 관한 정보를 가지고 있는 시스템이 있고(DNS 정보 배포와 같은) 인터넷 백본에 연결 위치를 지정한다. "백본"은 인터넷 트래픽을 전 세계로 전달하는 주 라인이다. 각 백본 머신은 특정 네트워크에서 특정 백본으로 연결되는 트래픽 그리고 그 백본에서 여러분의 ISP 까지 연결되는 모든 서비스 제공자들을 거치는 마스터 경로 테이블의 복사본을 가지고 있다.

여러분 사이트와 연결되는 곳을 백본 사이트에 알리는 것은 인터넷 서비스 공급자의 몫이고 이것을 경로 전달이라고 한다.

24.2.7 문제 해결

가끔 경로 전달에 문제가 있어서 어떤 사이트는 연결할 수 없다. 라우팅이 다운된 곳을 찾기 위해 사용할 수 있는 가장 유용한 명령은 traceroute(8) 명령이다. 원격 머신에 연결되지 않은 것처럼 보인다면 역시 유용하다(다시 말해 ping(8)이 실패한 경우).

traceroute(8) 명령은 연결을 원하는 머신 이름으로 실행한다. 결국에 목적 호스트에 도달하던가 연결이 끊겨서 중단되더라도 traceroute(8) 명령은 게이트웨이 호스트를 따라가는 길을 보여준다.

더 많은 정보는 traceroute(8) 매뉴얼 페이지를 본다.

24.2.7 멀티캐스트 라우팅

FreeBSD 는 멀티캐스트 어플리케이션과 멀티캐스트 라우팅을 기본적으로 지원한다. 멀티캐스트 어플리케이션을 사용하기 위해 FreeBSD 를 특별히 설정할 필요는 없다; 어플리케이션은 보통 FreeBSD 와 상관없이 실행된다. 멀티캐스트 라우팅은 커널에 컴파일 해서 지원해야 한다:

```
options MROUTING
```

게다가 멀티캐스트 라우팅 데몬 mrouted(8)은 /etc/mrouted.conf 파일로 터널과 DVMRP 로 설정해야 된다. 멀티캐스트 설정에 대한 자세한 사항은 mrouted(8) 매뉴얼 페이지에서 찾을 수 있다.

24.3 무선 네트워킹

24.3.1 소개

네트워크 케이블이 없이 항상 컴퓨터를 사용할 수 있다는 것은 매우 유용하다. FreeBSD 는 무선 클라이언트로 사용할 수 있고 "액세스 포인트"로도 사용할 수 있다.

24.3.2 무선 모드 작동

무선 장치 802.11 을 설정하는 BSS 와 IBSS 두 가지 방법이 있다.

24.3.2.1 BSS 모드

BSS 모드는 일반적으로 사용되는 모드이며 인프라스트럭처(infrastructure) 모드라고도 한다. 이 모드에서 다양한 무선 액세스 포인트가 유선 네트워크에 연결된다. 각 무선 네트워크는 고유한 이름을 가지고 있다. 이 이름을 네트워크의 SSID 라고 한다.

무선 클라이언트는 이들 무선 액세스 포인트에 연결되며 IEEE 802.11 표준은 무선 네트워크 연결에 사용하는 프로토콜을 정의하고 있다. 무선 클라이언트는 SSID 를 설정하여 특정 네트워크에 연결할 수 있지만 정확히 설정하지 않고도 어떤 네트워크에든 연결할 수 있다.

24.3.2.2 IBSS 모드

ad-hoc 모드(무선 클라이언트 상호간의 통신을 지원하기 때문에 액세스 포인트가 필요 없다)로 부르기도 하는 IBSS 모드는 포인트와 포인트를 연결하기 위해 디자인 되었다. 실제로 두 종류의 ad-hoc 모드가 있다. 첫 번째는 ad-hoc 또는 IEEE ad-hoc 모드라고 부르기도 하는 IBSS 모드다. 이 모드는 IEEE 802.11 표준에 의해 정의된다. 두 번째는 demo ad-hoc 모드 또는 Lucent ad-hoc(그리고 좀 혼란스럽게 가끔 ad-hoc 모드로 부르기도 한다) 모드로 부른다. 이것은 예전 pre-802.11 ad-hoc 모드이고 래거시(legacy) 설치에만 사용되어 왔다. ad-hoc 모드는 더 이상 설명하지 않는다.

24.3.3 인프라스트럭처 모드(Infrastructure Mode)

24.3.3.1 액세스 포인트

액세스 포인트는 하나 이상의 무선 클라이언트가 중앙 허브로 사용할 수 있는 무선 네트워크 장치다. 액세스 포인트를 사용할 때 모든 클라이언트는 액세스 포인트를 통해 통신한다. 여러 개의 액세스 포인트로 집, 사무실, 공원 등을 무선 네트워크로 완벽하게 커버할 수 있다.

액세스 포인트는 일반적으로 여러 개의 네트워크에 연결된다: 무선카드 그리고 나머지 네트워크에 연결하기 위한 하나 또는 하나 이상의 유선 네트워크 카드다.

액세스 포인트는 미리 빌드된 것을 구입하거나 FreeBSD 에 지원되는 무선 카드로 직접 빌드 할 수 있다. 여러 벤더에서 다양한 무선 액세스 포인트와 카드를 생산한다.

24.3.3.2 FreeBSD 액세스 포인트 빌드

[액세스 포인트를 빌드해서 확인하는 방법]

1. 필요한 사항

FreeBSD 로 액세스 포인트를 만들려면 호환되는 무선 카드가 필요하다. 현재 유일하게 지원되는 카드는 Prism 칩셋이다. 그리고 FreeBSD 에서 지원되는 무선 네트워크 카드도 필요하다(FreeBSD 가 다양한 종류의 장치를 지원하기 때문에 어렵지 않다). 이 가이드에서는 네트워크에 연결되어 있는 네트워크 카드 사이의 모든 트래픽을 무선 장치와 유선으로 중개하는 bridge(4)를 설명한다.

FreeBSD 가 액세스 포인트를 사용하는 hostap 기능은 특정 버전의 펌웨어에서 가장 제대로 동작한다. Prism 2 카드는 펌웨어 버전 1.3.4 나 새로운 버전을 사용해야 된다. Prism 2.5 와 Prism 3 카드는 펌웨어 1.4.9 를 사용해야 된다. 예전 버전의 펌웨어는 정확하게 작동하지 않을 것이다. 여기서 카드를 업데이트하는 유일한 방법은 카드 생산자가 제공하는 Windows 펌웨어 업데이트 유틸리티를 사용해야 된다.

2. 설치

첫째로 시스템이 무선 카드를 인식해야 된다:

```
# ifconfig -a
wi0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        inet6 fe80::202:2dff:fe2d:c938%wi0 prefixlen 64 scopeid 0x7
        inet 0.0.0.0 netmask 0xff000000 broadcast 255.255.255.255
        ether 00:09:2d:2d:c9:50
        media: IEEE 802.11 Wireless Ethernet autoselect (DS/2Mbps)
        status: no carrier
        ssid ""
```

```
        stationname "FreeBSD Wireless node"
        channel 10 authmode OPEN powersavemode OFF powersavesleep 100
        wepmode OFF weptxkey 1
```

지금은 자세한 사항에 대해 신경 쓰지 않고 설치된 무선 카드를 보여준다는 것만 알고 있자. 무선 인터페이스를 인지하고 PC 카드를 사용하는데 문제가 있다면 pccardc(8)을 확인하고 pccardd(8)에 대한 더 자세한 사항은 매뉴얼 페이지를 참고한다.

그리고 액세스 포인트로 만들기 위해 FreeBSD 의 브리지 부분을 준비하도록 모듈을 로드 해야 된다. bridge(4) 모듈을 로드 하려면 간단히 다음 명령을 실행한다:

```
# kldload bridge
```

모듈을 로드 할 때 에러 메시지가 없어야 된다. 에러 메시지가 나타나면 커널에 bridge(4) 코드를 컴파일 해야 된다. 핸드북의 브릿지 섹션에서 이 태스크를 해결하도록 도와줄 것이다.

이제 브리지 기능이 갖추어졌다. FreeBSD 커널에게 어떤 인터페이스가 브리지인지 sysctl(8)로 명시해야 된다:

```
# sysctl net.link.ether.bridge=1
# sysctl net.link.ether.bridge_cfg="wi0 xl0"
# sysctl net.inet.ip.forwarding=1
```

FreeBSD 5.2-RELEASE 와 이후 버전에서는 다음 옵션을 사용한다:

```
# sysctl net.link.ether.bridge.enable=1
# sysctl net.link.ether.bridge.config="wi0,xl0"
# sysctl net.inet.ip.forwarding=1
```

무선 카드 설정은 다음 명령이 카드를 액세스 포인트로 설정한다:

```
# ifconfig wi0 ssid my_net channel 11 media DS/11Mbps mediaopt hostap up
stationname "FreeBSD AP"
```

ifconfig(8) 명령으로 wi0 인터페이스를 up, SSID 를 *my_net* 으로 설정 그리고 스테이션

이름을 *FreeBSD AP*로 설정한다. *media DS/11Mbps*는 카드를 11Mbps 모드로 설정하고 이 설정은 *mediaopt* 효과를 위해 필요하다. *mediaopt hostap* 옵션은 인터페이스를 액세스 포인트 모드로 설정한다. *channel 11* 옵션은 802.11b 채널을 사용하도록 설정한다. wicontrol(8) 매뉴얼 페이지에서 도메인을 규정하는데 유효한 채널 옵션을 보여준다.

이제 액세스 포인트가 동작하여 완벽하게 기능을 수행한다. wicontrol(8), ifconfig(8)과 wi(4)를 읽어서 더 많은 정보를 얻을 수 있다.

3. 상태 정보

액세스 포인트가 설정되어 동작하면 운영자는 액세스 포인트에 연결된 클라이언트를 확인하고 싶을 것이다. 언제라도 다음 명령을 입력하여 확인할 수 있다:

```
# wicontrol -l
1 station:
00:09:b7:7b:9d:16  asid=04c0, flags=3<ASSOC,AUTH>, caps=1<ESS>,
rates=f<1M,2M,5.5M,11M>, sig=38/15
```

이것은 매개변수로 하나의 스테이션이 연결되어 있는 것을 보여준다. 신호는 상대적인 세기(strength)만 보여주는데 사용된다. dBm 로 변환하거나 다른 펌웨어에서 사용하는 다른 단위로 변경된다.

24.3.3.3 클라이언트

무선 클라이언트는 액세스 포인트나 다른 클라이언트에 직접 접근하는 시스템이다. 일반적으로 무선 클라이언트는 하나의 무선 네트워크 카드 장치를 가지고 있다.

무선 클라이언트를 설정하는 몇 가지 다른 방법이 있다. 이들은 보통 BBS 와(액세스 포인트가 필요한 인프라스트럭처 모드) IBBS 의(ad-hoc 또는 peer - to - peer 모드) 다른 무선 모드에 기반한다. 예제에서는 액세스 포인트와 통신하기 위한 두 가지 중 가장 유명한 BSS 모드를 사용한다.

[무선 클라이언트 설정]

1. 필요한 사항

FreeBSD 를 무선 클라이언트로 설정하기 위해 FreeBSD 가 지원하는 무선 카드가 필요하다.

2. 무선 FreeBSD 클라이언트 설정

시작하기 전에 알고 있어야 되는 몇 가지가 있다. 이 예제에서는 암호화를 끄고 *my_net* 이라는 이름의 네트워크에 연결한다.

> Note: 이 예제는 데이터를 암호화하지 않아서 보안적으로 위험하다. 다음 섹션에서는 어떻게 암호화하고 왜 암호화가 중요한지 그리고 어떤 암호화 기술은 왜 아직도 완벽하게 통신을 보호할 수 없는지 설명한다.

카드가 FreeBSD 에서 감지되어야 한다:

```
# ifconfig -a
wi0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        inet6 fe80::202:2dff:fe2d:c938%wi0 prefixlen 64 scopeid 0x7
        inet 0.0.0.0 netmask 0xff000000 broadcast 255.255.255.255
        ether 00:09:2d:2d:c9:50
        media: IEEE 802.11 Wireless Ethernet autoselect (DS/2Mbps)
        status: no carrier
        ssid ""
        stationname "FreeBSD Wireless node"
        channel 10 authmode OPEN powersavemode OFF powersavesleep 100
        wepmode OFF weptxkey 1
```

그리고 네트워크에 맞게 카드를 정확히 설정한다:

ifconfig wi0 inet *192.168.0.20* netmask *255.255.255.0* ssid *my_net*

192.168.0.20 과 255.255.255.0 를 여러분의 무선 네트워크에 맞는 IP 와 넷 마스크로 바꾼다. 우리의 액세스 포인트는 무선 네트워크와 유선 네트워크 사이의 데이터 브리지 역할을 하기 때문에 유선 네트워크에 다른 장치처럼 나타난다.

이 설정이 끝나면 표준 유선 연결처럼 무선 네트워크에서도 호스트에 ping 을 보낼 수 있다.

무선 연결에 문제가 있다면 액세스 포인트와 관련된 것을(연결된) 체크한다:

```
# ifconfig wi0
```

다음과 같은 어떤 정보가 나타나므로 확인한다:

```
status: associated
```

정보가 나타나지 않는다면 액세스 포인트의 범위를 벗어났거나, 데이터를 암호화하도록 했거나 설정 문제일 것이다.

24.3.3.4 암호화

완벽히 보호되는 영역으로 네트워크를 유지할 수 없기 때문에 무선 네트워크를 암호화하는 것은 중요하다. 무선 데이터는 인근지역 전체로 브로드케스트 되므로 이 데이터를 누구나 읽을 수 있다. 이런 이유로 암호화가 필요하다. 데이터를 암호화해서 전파를 보내면 데이터를 잡기가 더 어려워진다.

클라이언트와 액세스 포인트 사이의 데이터를 암호화하는 아주 일반적인 두 가지 방법은 WEP 와 ipsec(4)다.

[무선 데이터 암호화하기]

1. WEP

WEP 는 Wired Equivalency Protocol 의 약어다. WEP 는 유선 네트워크처럼 무선 네트워크를 안전하게 만든다. 그러나 이것은 크랙되어서 깨지기 쉽다. 따라서 중요한 데이터를 암호화하는 곳에 사용하지 않는다.

암호화하지 않는 것 보다는 안전하기 때문에 새로운 FreeBSD 액세스 포인트에서 다음 명령으로 WEP 를 켠다:

```
# ifconfig wi0 inet up ssid my_net wepmode on wepkey 0x1234567890 media
```

DS/11Mbps mediaopt hostap

그리고 다음 명령으로 클라이언트에서 WEP 를 켤 수 있다:

ifconfig wi0 inet *192.168.0.20* netmask *255.255.255.0* ssid *my_net* wepmode on wepkey *0x1234567890*

0x1234567890 는 유일한 키로 바꾼다.

2. IPsec

ipsec(4)는 네트워크 사이의 데이터를 암호화하는 더욱 안전하고 강력한 툴이다. 이것이 무선 네트워크 데이터를 암호화하는 확실한 선택이다. ipsec(4) 보안과 사용법은 이 핸드북의 Ipsec 섹션에서 더 읽을 수 있다.

24.3.3.5 툴

무선 네트워크 설정과 디버깅에 사용할 수 있는 몇 가지 툴이 있고 여기서 이 중 몇 가지를 소개한다.

1. bsd-airtools 패키지

bsd-airtools 패키지는 WEP 키 크랙킹과 액세스 포인트 탐지 등을 포함하여 완벽한 무선 점검 툴 세트다.

bsd-airtools 유틸리티는 net/bsd-airtools 포트에서 설치할 수 있다. 포트에서 설치하는 정보는 핸드북 4 장에서 찾을 수 있다.

dstumbler 프로그램은 액세스 포인트를 감지해서 신호 대 잡음 비를 기록할 수 있는 패키지 툴이다. 액세스 포인트를 올리고 실행하는데 어려움을 겪는다면 dstumbler 이 문제해결을 도와줄 수 있다.

무선 네트워크 보안을 테스트하려면 여러분의 무선 네트워크 보안에 WEP 이 적당한지 결정하는데 도움을 줄 수 있는 "dweputils"를(dwepcrack, dwepdump 와 dwepkeygen) 선택할 수 있다.

2. wicontrol, ancontrol 과 raycontrol 유틸리티

이 유틸리티들은 무선 네트워크에서 무선 카드를 제어하는 툴이다. 위의 예제에서 무선 카드가 wi0 인터페이스이기 때문에 wicontrol(8)을 선택했다. Cisco 무선 장치를 가지고 있다면 an0 로 나타나기 때문에 ancontrol(8)을 사용한다.

3. ifconfig 명령

ifconfig(8) 명령은 wicontrol(8)과 같은 다양한 옵션을 사용할 수 있지만 몇 가지 옵션이 빠져있다. 명령어 라인 매개변수와 옵션은 ifconfig(8)을 체크한다.

24.3.3.6 지원되는 카드

액세스 포인트

현재 BSS 모드를 지원하는 유일한 카드는 Prism 2, 2.5 또는 3 칩셋 기반의 장치다. 정확한 리스트는 wi(4)를 본다.

클라이언트

거의 모든 802.11b 무선 카드가 현재 FreeBSD 에서 지원된다. Prism, Spectrum24, Hermes, Aironet 과 Raylink 기반의 대부분의 카드는 IBSS 모드에서(ad-hoc, peer-to-peer 과 BSS) 무선카드로 작동한다.

24.4 블루투스

24.4.1 소개

블루투스는 라이선스가 없는 10M 범위의 2.4Ghz 밴드에서 개인 네트워크를 생성할 때 사용하는 무선 기술이다. 네트워크는 보통 핸드폰, PDA 와 노트북 컴퓨터 같은 휴대용 장치에서 ad-hoc 형식으로 사용된다. 다른 유명한 무선 기술(Wi-Fi)과 다르게 블루투스는 FTP 같은 파일 서버, 파일 공유, 음성 전달, 시리얼 라인 에뮬레이션 등 고급 레벨의 서비스를 제공한다.

FreeBSD 에서 블루투스 스택은 Netgraph 프레임워크를(netgraph(4)를 본다) 사용한다.

블루투스의 광범위한 USB 동글은(dongles) ng_ubt(4) 드라이버로 지원된다. Broadcom BCM2033 칩 기반 블루투스 장치는 ubtbcmfw(4)와 ng_ubt(4) 드라이버로 지원된다. 3Com 블루투스 PC 카드 3CRWB60-A 는 ng_bt3c(4) 드라이버로 지원된다. 시리얼과 UART 기반 블루투스 장치는 sio(4), ng_h4(4) 그리고 hcseriald(8)이 지원한다. 이번 장은 USB 블루투스 동글을 사용하는 방법을 설명한다. 블루투스 지원은 FreeBSD 5.0 과 새로운 버전에서 사용할 수 있다.

24.4.2 장치 연결

기본적으로 블루투스 장치 드라이버는 커널 모듈로 사용할 수 있다. 장치를 연결하기 전에 드라이버를 커널에 로드 해야 된다.

```
# kldload ng_ubt
```

블루투스 장치가 시스템이 시작될 때 나타나면 /boot/loader.conf 에서 모듈을 로드 한다.

```
ng_ubt_load="YES"
```

USB 동글을 연결한다. 다음과 비슷한 출력 결과가 콘솔에(또는 syslog 에) 나타난다.

```
ubt0: vendor 0x0a12 product 0x0001, rev 1.10/5.25, addr 2
ubt0: Interface 0 endpoints: interrupt=0x81, bulk-in=0x82, bulk-out=0x2
ubt0: Interface 1 (alt.config 5) endpoints: isoc-in=0x83, isoc-out=0x3,
      wMaxPacketSize=49, nframes=6, buffer size=294
```

/usr/share/examples/netgraph/bluetooth/rc.bluetooth 를 /etc/rc.bluetooth 와 같은 적당한 위치로 복사한다. 이 스크립트는 블루투스 스택을 시작하고 정지하는데 사용된다. 장치를 빼기 전에 스택을 정지 시키는 것이 좋지만 일반적으로 치명적이지는 않다. 스택을 시작할 때 다음과 비슷한 내용을 보게 된다:

```
# /etc/rc.bluetooth start ubt0
BD_ADDR: 00:02:72:00:d4:1a
Features: 0xff 0xff 0xf 00 00 00 00 00
<3-Slot> <5-Slot> <Encryption> <Slot offset>
```

```
<Timing accuracy> <Switch> <Hold mode> <Sniff mode>
<Park mode> <RSSI> <Channel quality> <SCO link>
<HV2 packets> <HV3 packets> <u-law log> <A-law log> <CVSD>
<Paging scheme> <Power control> <Transparent SCO data>
Max. ACL packet size: 192 bytes
Number of ACL packets: 8
Max. SCO packet size: 64 bytes
Number of SCO packets: 8
```

24.4.3 호스트 컨트롤러 인터페이스 (HCI)

호스트 컨트롤러 인터페이스는(HCI) 베이스밴드(Baseband) 컨트롤러와 링크 관리자 그리고 하드웨어 상태 확인과 제어를 등록하는 일반적인 인터페이스다. 이 인터페이스는 블루투스 베이스밴드 기능에 접근하는 일관된 방법을 제공한다. 호스트의 HCI 레이어는 블루투스 하드웨어에 HCI 펌웨어와 데이터 및 명령을 전달한다. 호스트 컨트롤러 전송 레이어(즉 물리적인 버스) 드라이버는 각자 정보를 교환할 수 있도록 HCI 레이어를 양쪽에 제공한다.

hci 타입의 Netgraph 노드 하나는 한 대의 블루투스 장치에 생성된다. HCI 노드는 보통 블루투스 장치 드라이버 노드(down 스트림)와 L2CAO 노드(up 스트림)에 연결된다. 모든 HCI 운용은 장치 드라이버 노드가 아닌 HCI 노드에서 수행해야 된다. HCI 노드의 기본 이름은 “devicehci”다. 더 자세한 사항은 ng_hci(4) 매뉴얼 페이지를 확인한다.

가장 일반적인 태스크는 RF 접근 방식으로 블루투스 장치를 인지하는 것이다. 이 동작을 *inquiry* 라고 한다. inquiry 와 HCI 관련된 다른 동작은 hccontrol(8) 유틸리티로 실행된다. 아래 예제는 어떤 블루투스 장치가 범위에 있는지 발견하는 것을 보여준다. 몇 초 안 장치 리스트를 보여준다. 원격 장치가 *discoverable* 모드에 있다면 inquiry 에 반응한다.

```
% hccontrol -n ubt0hci inquiry
Inquiry result, num_responses=1
Inquiry result #0
BD_ADDR: 00:80:37:29:19:a4
        Page Scan Rep. Mode: 0x1
        Page Scan Period Mode: 00
        Page Scan Mode: 00
        Class: 52:02:04
```

```
        Clock offset: 0x78ef
Inquiry complete. Status: No error [00]
```

*BD_ADDR*은 네트워크 카드의 MAC 주소와 비슷한 블루투스 장치의 유일한 주소다. 이 주소는 장치와 통신할 때 필요하다. 사람이 읽을 수 있는 이름을 BD_ADDR에 할당할 수 있다. /etc/Bluetooth/hosts 파일은 감지한 블루투스 호스트에 대한 정보를 가지고 있다. 다음 예제는 사람이 읽을 수 있도록 원격 장치에 할당되어있는 이름을 어떻게 가져오는지 보여준다.

```
% hccontrol -n ubt0hci remote_name_request 00:80:37:29:19:a4
BD_ADDR: 00:80:37:29:19:a4
Name: Pav's T39
```

원격 블루투스 장치에 inquiry를 실행하면 "your.host.name(ubt0)"으로 찾는다. 로컬 장치에 할당된 이름은 언제라도 변경할 수 있다.

블루투스 시스템은 일대 일(point-to-point) 연결(오직 두 개의 블루투스 유닛만 연결된다)이나 일대 다중(point-to-multipoint) 연결을 제공한다. 일대 다중(Point-to-multipoint) 연결은 여러 대의 블루투스 장치와 동시에 연결된다. 다음 예제는 로컬 장치에 연결되어 있는 베이스밴드 연결 리스트를 어떻게 가져오는지 보여준다.

```
% hccontrol -n ubt0hci read_connection_list
Remote BD_ADDR      Handle Type Mode Role Encrypt Pending Queue State
00:80:37:29:19:a4       41  ACL    0 MAST    NONE       0     0 OPEN
```

*연결 제어*는 베이스밴드 연결을 종료해야 될 때 유용하다. 보통 직접 종료할 필요는 없다. 스택이 활동하지 않는 베이스밴드 연결을 자동으로 종료한다.

```
# hccontrol -n ubt0hci disconnect 41
Connection handle: 41
Reason: Connection terminated by local host [0x16]
```

hccontrol help 명령이 사용할 수 있는 HCI 명령의 전체 리스트를 보여준다. 대부분의 HCI 명령은 슈퍼 유저 권한이 필요 없다.

24.4.4 Logical Link Control 와 Adaptation Protocol (L2CAP)

논리적 링크 제어와 Adaptation Protocol(L2CAP)은 프로토콜 다중화(multiplexing), 프로토콜 분할과 재 조합으로 상위 레이어 프로토콜에 연결 지향과 비 연결 데이터(connectionless data) 서비스를 제공한다. L2CAP 는 상위 레벨 프로토콜과 어플리케이션이 64 킬로바이트 이상의 L2CAP 데이터 패킷을 보내고 받는 것을 허용한다.

L2CAP 는 *채널*(channels) 개념을 기반으로 한다. 채널은 베이스밴드 연결의 최 상단에 논리적으로 연결된 것이다. 각 채널은 다 대 일(many-to-one) 형식으로 하나의 프로토콜로 바운드 된다. 여러 채널이 같은 프로토콜로 바운드될 수 있지만 하나의 채널은 여러 프로토콜로 바운드될 수 없다. 채널에 도착하는 각각의 L2CAP 패킷은 적절한 상위 레벨 프로토콜로 인계된다. 여러 개의 채널을 같은 베이스밴드 연결에 공유할 수 있다.

하나의 *l2cap* 타입 Netgraph 노드가 블루투스 장치 하나에 생성된다. L2CAP 노드는 보통 블루투스 HCI 노드(dwon 스트림)와 블루투스 소켓 노드(up 스트림)에 연결된다. L2CAP 노드의 기본 이름은 "devicel2cap"다. 더 자세한 내용은 ng_l2cap(4) 매뉴얼 페이지를 참고 한다.

유용한 명령은 다른 장치에 ping 을 보낼 때 사용하는 l2ping(8)이다. 어떤 블루투스 도구는 전송한 모든 데이터에 대한 결과를 되돌려 주지 않을 것이기 때문에 다음 예제의 *0 bytes* 가 일반적인 결과다.

```
# l2ping -a 00:80:37:29:19:a4
0 bytes from 0:80:37:29:19:a4 seq_no=0 time=48.633 ms result=0
0 bytes from 0:80:37:29:19:a4 seq_no=1 time=37.551 ms result=0
0 bytes from 0:80:37:29:19:a4 seq_no=2 time=28.324 ms result=0
0 bytes from 0:80:37:29:19:a4 seq_no=3 time=46.150 ms result=0
```

l2control(8) 유틸리티는 L2CAP 노드를 다양하게 운용하는데 사용된다. 이 예제는 논리적 연결과(채널) 로컬 장치의 베이스밴드 연결 리스트를 어떻게 가져오는지 보여 준다.

```
% l2control -a 00:02:72:00:d4:1a read_channel_list
L2CAP channels:
Remote BD_ADDR       SCID/ DCID    PSM   IMTU/ OMTU State
00:07:e0:00:0b:ca      66/    64      3     132/   672 OPEN
% l2control -a 00:02:72:00:d4:1a read_connection_list
L2CAP connections:
Remote BD_ADDR      Handle Flags Pending State
00:07:e0:00:0b:ca       41 O               0 OPEN
```

다른 진단 툴은 btsockstat(1)이다. netstat(1)과 비슷하지만 블루투스 네트워크와 관련된 데이터 구조를 보여준다. 아래 예제는 위의 l2control(8)과 같은 논리적인 연결을 보여준다.

```
% btsockstat
Active L2CAP sockets
PCB         Recv-Q Send-Q Local address/PSM       Foreign address    CID   State
c2afe900         0      0 00:02:72:00:d4:1a/3      00:07:e0:00:0b:ca 66    OPEN
Active RFCOMM sessions
L2PCB     PCB        Flag MTU   Out-Q DLCs State
c2afe900 c2b53380 1      127   0       Yes  OPEN
Active RFCOMM sockets
PCB         Recv-Q Send-Q Local address       Foreign address    Chan DLCI State
c2e8bc80         0    250 00:02:72:00:d4:1a 00:07:e0:00:0b:ca 3      6    OPEN
```

24.4.5 RFCOMM 프로토콜

RFCOMM 프로토콜은 L2CAP 프로토콜을 통해 시리얼 포트 에뮬레이션을 제공한다. 이 프로토콜은 ETSI(European Telecommunications Standards Institute) 표준 TS 07.10 에 기반한다. RFCOMM 는 RS-232(EIATIA-232-E) 9 핀 시리얼 포트를 에뮬레이팅하는 간단한 추가적인 전송 프로토콜이다. RFCOMM 는 60 개의 블루투스 장치를 동시에 연결(RFCOMM 채널들)할 수 있다.

RFCOMM 의 목적에 맞는 완전한 통신 경로는 다른 장치에서(통신 종단 점) 실행되는 두 개의 어플리케이션과 그들 사이의 통신 세그먼트도 포함한다. RFCOMM 은 장치의 시리얼 포트를 사용하는 어플리케이션을 다루도록 디자인 되어있다. 통신 세그먼트는 장치에서 장치로 연결되는(직접 연결) 블루투스 링크다.

RFCOMM 은 장치 사이를 직접 연결하는 경우나 네트워크에서 장치와 모뎀의 연결만 관련한다. RFCOMM 은 한쪽은 블루투스 무선 기술을 사용하여 통신하고 다른 쪽에는 유선 인터페이스를 제공하는 통신 모듈처럼 다른 설정을 지원할 수 있다.

FreeBSD 에서 RFCOMM 프로토콜은 블루투스 소켓 레이어에서 실행된다.

24.4.6 장치 Pairing

기본적으로 블루투스 통신은 인증이 되지 않아서 어떤 장치라도 다른 장치와 통신할 수 있다. 블루투스 장치에(예를 들어 핸드폰) 특정 서비스를(예를 들면 전화 접속 서비스) 제공하기 위해 인증이 필요할 것이다. 블루투스는 *PIN 코드*로 인증할 수 있다. PIN 코드는 16 문자 길이의 ASCII 문자열이다. 유저는 양쪽 장치에 같은 PIN 코드를 입력해야 된다. 유저가 PIN 코드를 입력하면 양쪽 장치는 *link key* 를 생성한다. 그리고 링크 키는 각 장치나 특정 스토리지에 저장할 수 있다. 다음부터 양쪽 장치는 이전에 생성한 링크 키를 사용한다. 앞에서 설명한 절차를 *pairing* 이라고 하고 링크 키를 잃어 버렸다면 pairing 를 다시 해야 된다.

hcsecd(8) 데몬이 블루투스 인증 요청을 처리할 수 있다. 기본 설정 파일은 /etc/Bluetooth/hcsecd.conf 다. 임의적으로 “1234”라는 PIN 코드를 설정한 핸드폰 예제가 아래에 있다.

```
device {
        bdaddr   00:80:37:29:19:a4;
        name      "Pav's T39";
        key       nokey;
        pin       "1234";
      }
```

PIN 코드에는(길이를 제외한) 제한이 없다. 어떤 장치에는(예를 들면 블루투스 헤드 셋) 고정된 PIN 코드가 내장되어 있을 것이다. *-d* 옵션이 hcsecd(8) 데몬을 강제로 포그라운드로 유지시키기 때문에 어떤 일이 발생하는지 확인하기 쉽다. 원격 장치가 pairing 에 응답하도록 설정해서 원격 장치에 블루투스 연결을 시작한다. 원격 장치는 pairing 이 허용됐음을 알리고 PIN 코드를 요청한다. hcsecd.conf 에 있는 PIN 코드와 같은 PIN 코드를 입력하면 PC 와 원격 장치가 연결된다. 그렇지 않으면 원격 장치에서 pairing 을

시작할 수 있다. 다음 내용은 hcsecd 데몬의 샘플 출력이다.

```
hcsecd[16484]: Got Link_Key_Request event from 'ubt0hci', remote bdaddr
0:80:37:29:19:a4
hcsecd[16484]: Found matching entry, remote bdaddr 0:80:37:29:19:a4, name 'Pav's
T39', link key doesn't exist
hcsecd[16484]: Sending Link_Key_Negative_Reply to 'ubt0hci' for remote bdaddr
0:80:37:29:19:a4
hcsecd[16484]: Got PIN_Code_Request event from 'ubt0hci', remote bdaddr
0:80:37:29:19:a4
hcsecd[16484]: Found matching entry, remote bdaddr 0:80:37:29:19:a4, name 'Pav's
T39', PIN code exists
hcsecd[16484]: Sending PIN_Code_Reply to 'ubt0hci' for remote bdaddr
0:80:37:29:19:a4
```

24.4.7 서비스 감지 프로토콜 (SDP)

서비스 감지 프로토콜(SDP)은 클라이언트 어플리케이션에서 서버 어플리케이션이 제공하는 서비스와 이들 서비스의 특성을 감지한다는 의미다. 서비스의 특성은 종류나 제공되는 서비스 등급과 메커니즘 또는 서비스를 사용하기 위해 필요한 프로토콜 정보를 포함한다.

SDP 는 SDP 서버와 SDP 클라이언트 사이의 통신에 연관된다. 서버는 서버와 관련된 서비스의 특징을 설명하는 서비스 레코드 리스트를 관리한다. 각 서비스 레코드는 각 서비스에 대한 정보를 가지고 있다. 클라이언트는 SDP 서버가 관리하는 서비스 레코드에 SDP 를 요청하여 정보를 갱신한다. 클라이언트 또는 클라이언트와 관련된 어플리케이션이 서비스를 사용하려면 서비스 공급자에게 연결해야 된다. SDP 는 서비스와 서비스의 특징을 감지하는 메커니즘을 제공하지만 이들 서비스를 사용하는 메커니즘은 제공하지 않는다.

보통 SDP 클라이언트는 원하는 특징의 서비스를 검색한다. 그러나 SDP 서버의 서비스 레코드에 설명되어 있는 서비스 타입을, 서비스에 대한 사전정보 없이 감지하는 적절한 시간이 있다. 제공되는 서비스를 찾는 절차를 *browsing* 이라고 한다.

블루투스 SDP 서버 sdpd(8)과 명령어 라인 클라이언트 sdpcontrol(8)은 표준 FreeBSD 설치에 포함되어 있다. 다음 예제는 SDP 브라우저 쿼리를 어떻게 수행하는지 보여준다.

```
% sdpcontrol -a 00:01:03:fc:6e:ec browse
Record Handle: 00000000
Service Class ID List:
        Service Discovery Server (0x1000)
Protocol Descriptor List:
        L2CAP (0x0100)
                Protocol specific parameter #1: u/int/uuid16 1
                Protocol specific parameter #2: u/int/uuid16 1

Record Handle: 0x00000001
Service Class ID List:
        Browse Group Descriptor (0x1001)

Record Handle: 0x00000002
Service Class ID List:
        LAN Access Using PPP (0x1102)
Protocol Descriptor List:
        L2CAP (0x0100)
        RFCOMM (0x0003)
                Protocol specific parameter #1: u/int8/bool 1
Bluetooth Profile Descriptor List:
        LAN Access Using PPP (0x1102) ver. 1.0
```

각 서비스에는 특성 리스트가(예를 들어 RFCOMM 채널) 있다. 서비스에 따라 몇 가지 특성을 기억해야 될 것이다. 어떤 블루투스 도구는 서비스 브라우징을 지원하지 않기 때문에 리스트를 되돌려 주지 않을 것이다. 이런 경우 특정 서비스를 검색하는 것도 가능하다. 다음 예제는 OBEX Object Push(OPUSH) 서비스를 어떻게 검색하는지 보여준다.

```
% sdpcontrol -a 00:01:03:fc:6e:ec search OPUSH
```

FreeBSD 에서는 sdpd(8) 서버로 블루투스 클라이언트에게 서비스를 제공한다.

```
# sdpd
```

블루투스 서비스를 원격 클라이언트에 제공하려는 로컬 서버는 로컬 SDP 데몬으로 서비스를 등록한다. 이러한 어플리케이션에는 rfcomm_pppd(8)이 있다. 이 어플리케이션이

실행되면 로컬 SDP 데몬으로 블루투스 LAN 서비스를 등록된다.

로컬 SDP 서버로 등록된 서비스 리스트는 로컬 제어 채널을 통한 SDP 브라우저 쿼리로 볼 수 있다.

```
# sdpcontrol -l browse
```

24.4.8 다이얼-업 네트워크와(DUN) PPP(LAN) 프로필로 네트워크 접근

다이얼-업 네트워크(DUN) 프로필은 대부분 모뎀과 핸드폰에 사용된다. 이 프로필로 다루는 시나리오는 다음과 같다:

- 다이얼-업 인터넷 서버나 다른 다이얼-업 서비스에 연결하기 위한 컴퓨터의 무선 모뎀으로 핸드폰이나 모뎀을 사용한다.
- 데이터를 받기 위해 컴퓨터에 핸드폰이나 모뎀을 사용한다.

PPP(LAN) 프로필로 네트워크에 접근하는 것은 다음과 같은 상황에 사용할 수 있다:

- 하나의 블루투스 장치로 LAN에 접근할 때.
- 여러 개의 블루투스 장치로 LAN에 접근할 때.
- PC와 PC를 연결(시리얼 케이블 에뮬레이션을 통한 PPP 네트워크를 사용한다)할 때.

FreeBSD 에서 양쪽 프로필은 RFCOMM 블루투스 연결을 PPP 로 변환하는 ppp(8)와 rfcomm_pppd(8) -a 로 감싸서 사용한다. 그리고 프로필을 사용하기 전에 /etc/ppp/ppp.conf 에 새로운 PPP 라벨을 만들어야 된다. 예제는 rfcomm_pppd(8) 매뉴얼 페이지를 참고한다.

다음 예제에서 rfcomm_pppd(8)은 DUN RFCOMM 채널에서 BD_ADDR

00:80:37:29:19:a4 로 원격 장치에 RFCOMM 연결을 여는데 사용된다. 실제 RFCOMM 채널 번호는 원격 장치에서 SDP 를 통해 받아온다. RFCOMM 채널을 직접 지정하는 것도 가능하고 이 경우 rfcomm_pppd(8)은 SDP 쿼리를 수행하지 안는다. 그리고 원격 장치에서 RFCOMM 채널을 찾을 때 sdpcontrol(8)을 사용한다.

```
# rfcomm_pppd -a 00:80:37:29:19:a4 -c -C dun -l rfcomm-dialup
```

PPP(LAN)로 네트워크 접근 서비스를 제공하기 위해 sdpd(8) 서버가 실행되고 있어야 된다. LAN 클라이언트의를 위한 새로운 엔트리를 /etc/ppp/ppp.conf 파일에 생성하고 예제는 rfcomm_pppd(8) 매뉴얼 페이지를 참고한다. 마지막으로 RFCOMM PPP 서버는 유효한 RFCOMM 채널 번호에서 실행한다. RFCOMM PPP 서버는 로컬 SDP 데몬을 통해 자동으로 블루투스 LAN 서비스에 등록된다. 아래 예제는 RFCOMM PPP 서버를 어떻게 시작하는지 보여준다.

```
# rfcomm_pppd -s -C 7 -l rfcomm-server
```

24.4.9 OBEX Object Push (OPUSH) 프로필

OBEX 는 무선 장치 사이에서 파일을 전송하는데 광범위하게 사용되는 프로토콜이다. 주된 용도로 노트북이나 Palm 무선 장치 사이에서는 일반적인 파일 전송에, 핸드폰과 다른 장치 사이에서는 PIM 어플리케이션으로 명함이나 달력 엔트리를 전송하는 적외선 통신에 사용된다.

OBEX 서버와 클라이언트는 comms/obexapp 포트에서 사용할 수 있는 **obexapp** 패키지를 사용할 수 있다.

OBEX 클라이언트는 OBEX 서버에 주체를 보내(push)고 받는(pull)데 사용된다. 예를 들면 주체는 명함이나 시간 약속일 것이다. OBEX 클라이언트는 SDP 를 통해 원격 장치에서 RFCOMM 채널 번호를 가져올 수 있다. 여기서 RFCOMM 채널 번호 대신 서비스 이름을 지정할 수 있다. 지원되는 서비스 이름은 IrMC, FTRN 그리고 OPUSH 다. RFCOMM 채널을 번호로 지정할 수 있다. 아래는 장치 정보 주체를 핸드폰에서 가져오고 새로운 주체(명함)를 전화의 디렉터리에 보내는 OBEX 세션의 예제다.

```
% obexapp -a 00:80:37:29:19:a4 -C IrMC
obex> get
get: remote file> telecom/devinfo.txt
get: local file> devinfo-t39.txt
Success, response: OK, Success (0x20)
obex> put
put: local file> new.vcf
put: remote file> new.vcf
Success, response: OK, Success (0x20)
obex> di
Success, response: OK, Success (0x20)
```

OBEX 주체를 보내(push)는 서비스를 제공하려면 sdpd(8) 서버가 실행되고 있어야 된다. 입력되는 모든 주체가 저장되는 root 폴더를 생성해야 되고 기본 경로는 /var/spool/obex 다. 마지막으로 유효한 RFCOMM 채널번호에서 OBDEX 서버를 시작한다. 아래 예제에서 OBEX 서버를 어떻게 시작하는지 보여준다.

```
# obexapp -s -C 10
```

24.4.10 시리얼 포트 프로필(SPP)

시리얼 포트 프로필(SPP)은 블루투스 장치가 RS232(또는 비슷한) 시리얼 케이블 에뮬레이션을 수행하도록 한다. 이 프로필로 다루는 시나리오는 케이블을 대신하여 가상 시리얼 포트 개념으로 블루투스를 사용하는 레거시 어플리케이션이다.

rfcomm_sppd(1) 유틸리티는 시리얼 포트 프로필을 실행한다. pseudo tty 가 가상 시리얼 포트 개념에 사용된다. 아래 예제는 원격 장치의 시리얼 포트 서비스에 어떻게 연결하는지 보여준다. RFCOMM channel-rfcomm_sppd(1)는 SDP 를 통해 원격 장치에서 가져올 수 있기 때문에 지정할 필요가 없다. 이것을 무시하려면 명령어 라인에서 RFCOMM 채널을 지정한다.

```
# rfcomm_sppd -a 00:07:E0:00:0B:CA -t /dev/ttyp6
rfcomm_sppd[94692]: Starting on /dev/ttyp6...
```

연결된 pseudo tty 는 시리얼 포트로 사용할 수 있다.

```
# cu -l ttyp6
```

24.4.11 문제 해결

24.4.11.1 원격 장치에 연결하지 못한다.

예전의 어떤 블루투스 장치는 role 스위칭을 지원하지 못한다. 기본적으로 FreeBSD 가 새로운 연결을 허용했을 때 블루투스 장치는 role 스위칭을 수행해서 마스터가 된다. 이 기능을 지원하지 못하는 장치는 연결할 수 없다. 새로운 연결이 완료되면 role 스위칭이 수행되기 때문에 원격 장치가 role 스위칭을 지원하는지 확인하는 것은 불가능하다. 로컬에서 role 스위칭을 비활성하는 HCI 옵션이 있다.

```
# hccontrol -n ubt0hci write_node_role_switch 0
```

24.4.11.2 무엇인가 잘못되면 확인할 수 있는가?

확인할 수 있다. http://www.geocities.com/m_evmenkin/에서 다운로드 할 수 있는 hcidump-1.5 패키지를 사용한다. hcidump 유틸리티는 tcpdmp(1)과 비슷하다. 블루투스 패킷의 내용을 터미널에 표시하고 블루투스 패킷을 파일로 덤프하는데 사용할 수 있다.

24.5 브리지

24.5.1 소개

IP 서브넷을 생성하지 않고 하나의 물리적인 네트워크(이더넷 구간과 같은)를 두 개의 네트워크 구간으로 나누고, 라우터를 사용하여 두 개의 구간을 연결하는 것이 유용한 경우가 있다. 두 개의 네트워크를 연결하는 장치를 여기서는 "브리지"라고 부른다. 두 개의 네트워크 인터페이스를 가진 FreeBSD 는 브리지로 동작할 수 있다.

브리지는 각 네트워크 인터페이스의 맥 어드레스(이더넷 주소)로 동작한다. 브리지는 소스와 목적지가 다른 네트워크에 있을 때만 트래픽을 포워드 한다.

많은 것을 고려하면 브리지는 매우 적은 포트를 가진 이더넷 스위치와 같다.

24.5.2 브리지는 어떤 곳에 적당한가

요즘에 일반적으로 브리지가 사용되는 두 가지 상황이 있다.

24.5.2.1 구간의 트래픽이 높을 때

첫 번째 경우는 물리적인 네트워크 구간이 트래픽으로 과부하 상태지만 어떤 이유에서든 서브넷으로 네트워크를 나누고 라우터로 서브넷 사이를 연결하지 않을 때다.

신문사를 예로 들면 편집부와 생산부서가 같은 서브 네트워크에 있다고 가정하자. 편집부의 모든 유저는 서버 A 를 파일 서비스로 사용하고 생산부서의 유저들은 서버 B 를 사용한다. 이더넷을 사용하여 모든 유저가 연결되므로 네트워크의 부하가 심해서 느리고 종종 다운된다.

하나의 네트워크에서 편집부 유저는 한쪽 네트워크 구간을 사용하고 생산부서 유저를 다른 네트워크를 사용하도록 분리할 수 있다면 두 네트워크 구간을 브리지로 연결할 수 있다. 네트워크 패킷의 목적지가 브리지의 다른 쪽 인터페이스(다른 부서)에 있을 때만 다른 네트워크로 보내기 때문에 각 네트워크 구간의 과부하를 줄일 수 있다.

24.5.2.2 방화벽 필터링이 필요한 곳

일반적인 두 번째 경우는 네트워크 주소 변환(NAT) 기능이 없는 방화벽이 필요한 곳이다.

DSL 이나 ISDN 으로 ISP 에 연결된 작은 회사를 예로 들어보자. 이 회사는 ISP 로부터 13 개의 공인 IP 를 할당 받았고 네트워크에 10 대의 PC 가 있다. 이 상황에서 라우터 기반 방화벽을 사용하는 것은 서브넷 문제 때문에 어렵다.

IP 번호 문제 없이 DSL/ISDN 라우터의 다운(down) 스트림 경로에 브리지 기반 방화벽을 설치할 수 있다.

24.5.3. 브리지 설정

24.5.3.1 네트워크 인터페이스 카드 선택

브리지는 기능상 최소 두 개의 네트워크 카드가 필요하다. 불행히 FreeBSD 4.0 에서는 모든 네트워크 인터페이스 카드가 브리지를 지원하지 못한다. 지원되는 카드의 자세한 리스트는 bridge(4)를 읽는다.

계속 진행하기 전에 두 개의 네트워크 카드를 설치하고 테스트한다.

24.5.3.2 커널 설정 변경

커널이 브리지를 지원하도록 다음 라인을 추가한다:

```
options BRIDGE
```

커널 설정파일에 명시하고 커널을 다시 빌드한다.

24.5.3.3 방화벽 기능

브리지에 방화벽 기능을 추가하여 사용하려면 *IPFIREWALL* 옵션도 추가해야 된다. 브리지를 방화벽으로 설정하는 일반적인 정보는 14 장을 읽는다.

IP 패킷이 아닌 다른 페킷(ARP 와 같은)이 브리지를 통과하도록 하려면 방화벽 옵션을 설정해야 된다. 이 옵션은 *IPFIREWALL_DEFAULT_TO_ACCEPT* 이고 모든 패킷이 접근할 수 있도록(방화벽 기능을 활성화하면 모든 패킷을 거부한다) 방화벽의 기본 룰을 변경한다. 설정하기 전에 변경하는 방법과 변경하려는 룰의 의미를 알고 있어야 된다.

24.5.3.4 트래픽(대역폭) 제어 지원

브리지를 트래픽 제어기로 사용하려면 커널 설정에 *DUMMYNET* 옵션을 추가해야 된다. 더 많은 정보는 dummynet(4)를 읽는다.

24.5.4 브리지 활성화

시작할 때 브리지가 활성화되도록 다음 라인을 /etc/sysctl.conf 에 추가한다:

```
net.link.ether.bridge=1
```

그리고 지정된 인터페이스에서 브리지가 활성화되도록 다음 라인도 추가한다(*if1* 과 *if2* 는 여러분의 네트워크 인터페이스에 맞도록 변경한다):

```
net.link.ether.bridge_cfg=if1,if2
```

브리지가 ipfw(8)로 패킷 필터링를 하게 하려면 다음 라인을 추가한다:

```
net.link.ether.bridge_ipfw=1
```

FreeBSD 5.2-RELEASE 와 이후의 버전은 다음 라인을 대신 사용한다:

```
net.link.ether.bridge.enable=1
net.link.ether.bridge.config=if1,if2
net.link.ether.bridge.ipfw=1
```

24.5.5 다른 정보

네트워크에서 텔넷으로 브릿지에 접근하려면 네트워크 카드 하나에 IP 주소를 할당하면 된다. 두 카드의 주소를 동일하게 할당하지 않는다.

네트워크에 여러 개의 브리지가 있다면 두 대의 워크스테이션간의 경로는 하나뿐이어야 된다. 기술적으로 이 의미는 스패닝 트리(spanning tree: 브리지나 스위치 네트워크에서 발생한 논리적인 루프를 감지하고 계산하는데 사용되는 IEEE 802.1D 표준) 링크 관리가 지원되지 않는다.

브리지는 잠재적으로 ping 응답 시간을 증가시킬 수 있고 특히 한쪽 구획에서 다른 구획으로의 트래픽을 증가시킬 수 있다.

24.6 디스크 없이 운용하기(Diskless Operation)

FreeBSD 머신은 네트워크를 통해 부팅할 수 있기 때문에 NFS 서버로부터 마운트 한 파일 시스템을 사용하여 로컬 디스크 없이 운용할 수 있다. 표준 설정파일 이외에 시스템을 수정할 필요 없다. 이런 시스템은 필요한 모든 요소를 사용할 수 있도록 준비 되어있기 때문에 상당히 설정하기 쉽다:

- 네트워크를 통해 커널을 로드 할 수 있는 두 가지 방법이 있다:
  - ✓ PXE: Intel® Preboot Execution Environment system은 네트워크 카드나 마더보드에 다양한 기능이 내장된 부트 롬이다. 더 자세한 사항은 pxeboot(8)를 본다.
  - ✓ etherboot 포트(net/etherboot)는 네트워크를 통해 ROM-able 코드를 부트 커널에 생성한다. 코드를 네트워크 카드의 부트 PROM이나 로컬 플로피(또는 하드) 디스크 드라이브에 저장할 수 있으며 사용중인 MS-DOS 시스템에서 로드 할 수 있다. 다양한 네트워크 카드가 지원된다.
- 샘플 스크립트(/usr/share/examples/diskless/clone_root)는 생성하기 쉽고 서버에서 워크스테이션의 root 파일시스템을 관리한다. 이 스크립트를 약간 수정해야겠지만 시작하기 쉽다.
- /etc에있는 표준 시스템 시작파일은 디스크가 없는 시스템을 감지해서 시스템의 부팅을 지원한다.
- 스왑이 필요하다면 NFS 파일이나 로컬 디스크를 사용할 수 있다.

디스크가 없는 워크스테이션을 설정하는 여러 가지 방법이 있다. 다양한 요소가 관련이 있겠지만 대부분은 로컬 시스템(클라이언트)의 상황에 맞게 수정할 수 있다. 다음은 표준 FreeBSD 시작 스크립트와의 호환성을 강조하여 전체 시스템의 설정 변수를 설명한 것이다. 설명하는 시스템은 다음과 같은 특성을 가지고 있다:

- 디스크가 없는 워크스테이션은 읽기 전용으로 공유되어 있는 root 파일시스템과 /usr을 사용한다.

root 파일시스템은 디스크가 없이 운용하는데 알맞게 수정한 설정파일이거나 워크스테이션이 가지고 있던 설정파일을 수정한 표준 FreeBSD root 의 복사본이다.
root 파일시스템에서 쓰기 기능이 필요한 곳은 FreeBSD 4.X 에서는 mfs(8), FreeBSD 5.X 에서는 md(4) 파일시스템으로 덮어써야 된다. 이렇게 변경해도 시스템이 재 부팅하면 원상태로 복구 된다.

- 커널은 전송되어 Etherboot나 PXE 프로그램에 의해 로드 되고 상화에 따라 두 프로그램 중에서 한가지가 사용된다.

  주의: 앞에서 설명하였듯이 이 시스템은 보안상 안전하지 않다. 안전한 네트워크에 두고 다른 호스트와는 신뢰관계를 맺지 않는다.

이번 장의 모든 정보는 FreeBSD 4.9-RELEASE 와 5.2.1-RELEASE 에서 테스트됐다. 이 글은 주로 4.X 사용에 중점을 두고 작성되었다.

24.6.1 백그라운드 정보

디스크가 없는 워크스테이션을 설정하는 것은 상당히 직선적이고 에러가 발생하기 쉽다. 그리고 여러 가지 이유가 있기 때문에 원인을 찾아내기도 어렵다:

- 컴파일 시간 옵션은 실행할 때 다르게 동작할 수 있다.
- 에러 메시지는 종종 애매하거나 없을 것이다.

이러한 상황에서 메커니즘과 관련된 백그라운드 정보는 발생할 수 있는 문제 해결에 매우 유용하다.

성공적인 부팅을 위해 여러 가지 동작이 수행되어야 한다:

- 머신에 IP 주소, 실행할 수 있는 파일 이름, 서버 이름 그리고 root 경로 등과 같은 초기화 매개변수가 필요하다. 이 정보는 DHCP 또는 BOOTP 프로토콜을 사용하면 된다. DHCP는 BOOTP에서 확장된 것이기 때문에 포트 번호와 기본 패킷 포맷이 같다.

BOOTP 만 사용하도록 시스템을 설정할 수도 있다. bootpd(8) 서버 프로그램은 FreeBSD 기본 시스템에 포함되어있다.

그러나 DHCP 는 BOOTP(PXE 를 사용할 것이고 디스크가 없이 운용하는데 직접 연관은 없지만 다양한 기능을 추가하여 사용할 수 있는 쓸만한 설정파일) 이상의 다양한 장점을 가지고 있기 때문에, 여기서는 주로 DHCP 를 설정하는 방법 위주로 설명하고 가능하다면 bootpd(8)을 사용한 예제도 보여줄 것이다. 샘플 설정에서는 ISC DHCP 소프트웨어 패키지를(3.0.1.r12 가 테스트 서버에 설치되어 있다) 사용한다.

- 머신은 하나나 여러 개의 프로그램을 로컬 메모리에 전송해야 된다. 여기에 TFTP나 NFS가 사용된다. TFTP와 NFS를 선택하는 것은 컴파일 시간 옵션과 관련이 있다. 일반적으로 발생하는 에러의 원인은 잘못된 프로토콜에 파일이름을 지정하는 것이다: TFTP는 보통 서버 디렉터리의 한곳에 있는 모든 파일을 전송하기 때문에, 파일 이름은 이 디렉터리와 상대적이지만 NFS는 절대 파일 경로를 사용한다.

- 부트스트랩 중간의 프로그램과 커널을 초기화하고 실행해야 된다. 이 부분에서 몇 가지를 변경해야 된다:

  - ✓ PXE는 FreeBSD의 세 번째 로더의 수정된 버전인 pxeboot(8)을 로드 한다. 전송 제어를 하기 전에 loader(8)은 시스템 시작에 필요한 대부분의 매개변수를 가져와서 커널 환경에 둔다. 여기서 GENERIC 커널을 사용해도 된다.
  - ✓ Ehterboot는 특별한 준비 없이 커널을 직접 로드한다. 특정 옵션으로 커널을 빌드 해야 된다.

  PXE 와 Etherboot 는 4.X 시스템에서와 동일하게 동작한다. 왜냐하면 5.X 커널은 보통 loader(8)에게 더 많은 태스크를 시키기 때문에 PXE 는 5.X 시스템을 지원한다.

  BIOS 와 네트워크 카드가 PXE 를 지원한다면 이것을 사용하는 것이 좋다. 그렇지만 Etherboot 로 5.X 시스템을 시작하는 것도 가능하다.

- 마지막으로 머신이 파일시스템에 접근해야 된다. 이 모든 경우에 NFS가 사용된다.

그리고 diskless(8) 매뉴얼 페이지도 참고한다.

24.6.2 명령 설정

24.6.2.1 ISC DHCP 를 사용한 설정

ISC DHCP 서버는 BOOTP 와 DHCP 요청에 응답할 수 있다.

릴리즈 4.9 에서 ISC DHCP 3.0 은 기본 시스템에 포함되어 있지 않다. net/isc-dhcp3-server 포트나 적당한 패키지를 먼저 설치해야 된다. 포트와 패키지에 대한 일반적인 정보는 4 장을 참고한다.

ISC DHCP 가 설치된 후 실행하기 위해 설정파일이 필요하다(보통 /usr/local/etc/dhcpd.conf). 호스트 margaux 는 Etherboot 를 그러나 호스트 corbieres 는 PXE 를 사용하는 예제가 있다:

```
default-lease-time 600;
max-lease-time 7200;
authoritative;

option domain-name "example.com";
option domain-name-servers 192.168.4.1;
option routers 192.168.4.1;

subnet 192.168.4.0 netmask 255.255.255.0 {
  use-host-decl-names on; ❶
  option subnet-mask 255.255.255.0;
  option broadcast-address 192.168.4.255;

  host margaux {
    hardware ethernet 01:23:45:67:89:ab;
    fixed-address margaux.example.com;
    next-server 192.168.4.4; ❷
    filename "/data/misc/kernel.diskless"; ❸
```

```
    option root-path "192.168.4.4:/data/misc/diskless"; ❹
  }
  host corbieres {
    hardware ethernet 00:02:b3:27:62:df;
    fixed-address corbieres.example.com;
    next-server 192.168.4.4;
    filename "pxeboot";
    option root-path "192.168.4.4:/data/misc/diskless";
  }
}
```

❶ 이 옵션은 **dhcp** 에게 디스크가 없는 호스트의 호스트 이름으로 *host* 선언부의 값을 보내도록 한다. 그렇지 않으면 *host* 선언에 *option host-name margaux* 를 추가한다.

❷ *next-server* 는 로더난 커널 파일을 로딩하기 위해 사용하는 TFTP 또는 NFS 서버를 가리킨다(기본값은 DHCP 서버에서와 같은 호스트를 사용한다).

❸ *filename* 은 **Etherboot** 또는 PXE 가 다음 실행단계에서 로드 해야 될 파일을 정의한다. 사용하는 전송 방법에 따라 지정해야 된다. **Etherboot** 는 NFS 나 TFTP 를 사용하도록 컴파일 할 수 있고 FreeBSD 포트는 기본적으로 NFS 로 설정된다. 여기서 상대적인 파일 이름을 사용하는 것은 PXE 가 TFTP 를 사용하기 때문이다(이것은 TFTP 서버 설정과 관련이 있겠지만 대부분은 일반적인 설정을 가지고 있다). 그리고 PXE 는 커널이 아닌 pxeboot 도 로드 한다. pxeboot 를 로딩하는 것처럼 상당히 흥미 있는 것들이 FreeBSD CD-ROM 의 /boot 디렉터리에(pxeboot(8)은 GENERIC 커널을 로드 할 수 있기 때문에 PXE 를 사용하여 원격 CD-ROM 에서 부팅하는 것도 가능하다) 있다.

❹ *root-path* 옵션은 보통 NFS 표기에서 root 파일시스템 경로를 정의한다. PXE 를 사용할 때 BOOTP 커널 옵션을 활성화하지 않았다면 호스트의 IP 를 그대로 둘 수 있다. NFS 서버는 TFTP 와 같다.

24.6.2.2 BOOTP 사용 설정

여기 /etc/bootptab 에서 찾을 수 있는 **bootpd** 와 동일한 설정이 있다.

BOOTP 를 사용하기 위해 기본 옵션이 아닌 *NO_DHCP_SUPPORT* 옵션으로 Etherboot 를 컴파일 하는 것에 주의하고 PXE 는 DHCP 가 필요하다. bootpd 의 확실한 장점은 bootpd 가 기본 시스템에 있다는 것이다.

```
.def100:₩
                :hn:ht=1:sa=192.168.4.4:vm=rfc1048:₩
                :sm=255.255.255.0:₩
                :ds=192.168.4.1:₩
                :gw=192.168.4.1:₩
                :hd="/tftpboot":₩
                :bf="/kernel.diskless":₩
                :rp="192.168.4.4:/data/misc/diskless":

            margaux:ha=0123456789ab:tc=.def100
```

24.6.2.3 Etherboot 로 부트 프로그램 준비

Etherboot 의 웹 사이트(http://etherboot.sourceforge.net/)에는 주로 리눅스 시스템을 위한 문서가 많이 있지만 그래도 유용한 정보가 많이 있다. 다음은 FreeBSD 시스템에서 Etherboot 를 어떻게 사용할지 요점을 제시한다.

첫째로 net/etherboot 패키지나 포트를 설치한다. Etherboot 포트는 보통 /usr/ports/net/etherboot에서 찾을 수 있다. 시스템에 포트 트리가 설치되어 있다면 이 디렉터리에서 *make* 명령만 입력하고 모든 것을 주의 깊게 관찰한다. 그 밖의 포트와 패키지에 대한 정보는 4 장을 참고한다.

Etherboot 소스 디렉터리의 Config 파일을 수정하여 etherboot 설정을(다시 말해 NFS 대신 TFTP 를 사용하도록) 변경할 수 있다.

여기서는 부트 플로피를 사용할 것이다. 다른 방법은(PROM 이나 DOS 프로그램) Etherboot 문서를 참고한다.

부트 플로피를 생성하는 방법은 Etherboot 가 설치된 머신의 드라이브에 플로피를 넣고, 현재 디렉터리를 Etherboot 트리의 src 디렉터리로 변경한 후 다음 명령을 입력한다:

```
# gmake bin32/devicetype.fd0
```

devicetype 은 디스크가 없는 워크스테이션의 이더넷 카드 종류를 말한다. 정확한 *devicetype* 은 같은 디렉터리의 NIC 파일을 참고한다.

24.6.2.4 PXE 로 부팅

기본적으로 pxeboot(8) 로더는 NFS 로 커널을 로드 한다. /etc/make.conf 에 *LOADER_TFTP_SUPPORT* 옵션을 지정하여 TFTP 를 대신 NFS 를 사용하도록 컴파일 할 수 있다. 관련된 지시 사항은 /etc/defaults/make.conf 의(또는 5.X 시스템은 /usr/share/examples/etc/make.conf) 설명을 본다.

시리얼 콘솔로 디스크가 없는 머신을 설정하는데 유용하고 문서화되지 않은 *BOOT_PXELDR_PROBE_KEYBOARD* 와 *BOOT_PXELDR_ALWAYS_SERIAL*(후자는 FreeBSD 5.X 에만 있다) 두 가지 make.conf 옵션이 있다.

머신이 시작될 때 PXE 를 사용하려면 보통 BIOS 설정에서 *Boot from network* 옵션을 선택하거나 PC 를 초기화하는 동안 기능 키를 입력한다.

24.6.2.5 TFTP 와 NFS 서버 설정

TFTP 를 사용하도록 Etherboot 나 PXE 를 설정했다면 파일 서버에서 tftpd 를 활성화해야 된다:

① tftpd가 파일을 제공할 디렉터리를 생성한다. 예: /tftpboot

② /etc/inetd.conf에 다음 라인을 추가한다:

```
tftp    dgram   udp     wait    root    /usr/libexec/tftpd      tftpd -s /tftpboot
```

Note: 어떤 PXE 버전은 TCP 버전의 TFTP 를 사용해야 되는 경우가 있다. 이 경우 *gram udp* 를 *stream tcp* 로 교체한다.

③ inetd가 설정파일을 다시 읽도록 한다:

```
# kill -HUP `cat /var/run/inetd.pid`
```

tftpboot 디렉터리를 서버의 원하는 곳에 생성할 수 있다. 이 위치를 inetd.conf 와 dhcpd.conf 두 곳에 설정해야 된다.

NFS 를 활성화하고 NFS 서버로 적당한 파일시스템을 공유한다.

① 다음 라인을 /etc/rc.conf에 추가한다:

```
nfs_server_enable="YES"
```

② 디스크가 없는 워크스테이션에 파일시스템을 공유하는 것은 다음 라인을 /etc/exports에 추가하면 된다(볼륨의 마운트 위치를 조정하고 *margaux corbieres* 대신 디스크가 없는 워크스테이션의 이름을 사용한다):

```
/data/misc -alldirs -ro margaux corbieres
```

③ mountd가 설정파일을 다시 읽게 한다. 첫 단계에서 실제로 /etc/rc.conf를 사용하여 NFS를 활성화해야 되고 재 부팅하는 대신 다음 명령을 사용할 수 있다.

```
# kill -HUP `cat /var/run/mountd.pid`
```

24.6.2.6 디스크가 없는 커널 빌드

Etherboot 를 사용한다면 디스크가 없는 클라이언트를 위해 다음 옵션을 추가하여 커널 설정파일을 생성해야 된다(일반적으로):

```
options     BOOTP           # Use BOOTP to obtain IP address/hostname
options     BOOTP_NFSROOT   # NFS mount root filesystem using BOOTP info
```

또한 *BOOTP_NFSV3*, *BOOT_COMPAT* 그리고 *BOOTP_WIRED_TO* 를 사용해도 된다(4.X 는 LINT 를 5.X 는 NOTE 참고한다).

Note: PXE 를 사용할 때 위의 옵션으로 커널을 빌드할 필요는 없다(그러나 권장한다). 이들 옵션을 활성화하면 특별한 경우에 pxeboot(8)이 검색한 것과 새로운

값이 일치하지 않아서 발생할 수 있는 위험을 줄이고, 커널이 시작되는 동안 더 많은 DHCP 요청을 허용한다.

이들 옵션은 사실 커널에서 DHCP 와 BOOTP 를 일반적으로 활성화하기 때문에 역사적으로 약간의 혼란을 야기했다(강제로 BOOTP 또는 DHCP 를 사용하도록 할 수 있다).

커널을 빌드(8 장을 본다)해서 dhcpd.conf 에 지정한 곳에 복사한다.

Note: Etherboot 로 로드 할 수 있도록 5.X 커널에 장치 힌트(device hint)가 컴파일되어 있어야 된다. 설정파일에 다음 옵션을 추가한다(NOTES 파일을 본다):

```
hints        "GENERIC.hints"
```

24.7.2.7 root 파일시스템 준비

dhcpd.conf 의 *root-path* 에 디스크가 없는 워크스테이션을 위한 root 파일시스템을 생성해야 된다.

[root 파일시스템을 생성하는 두 가지 방법]

1. clone_root 스크립트를 사용하는 방법

이 방법이 root 파일시스템을 생성하는 가장 빠른 방법이지만 현재 FreeBSD 4.X 에서만 지원된다. 이 쉘 스크립트는 /usr/share/examples/diskless/clone_root 에 있으며 약간의 수정이 필요하고 최소한 파일시스템을 생성할 위치가 어느 곳인지(*DEST* 변수) 지정해야 된다.

필요한 사항은 스크립트 최 상단의 설명을 참고한다. 이곳에는 기본 시스템을 어떻게 빌드해야 되고 디스크가 없는 운용, 서브 네트워크 또는 각각의 워크스테이션 버전에 맞게 어떤 파일을 선택해야 되는지 설명한다. 또한 디스크가 없는 운용에 맞는 예제 설정파일 /etc/fstab 와 /etc/rc.conf 도 제공한다.

/usr/share/examples/diskless 의 README 파일에 다양한 백그라운드 정보가 있지만 diskless 디렉터리의 다른 예제는 clone_root 가 사용하는 것과 /etc 의 시스템 시작 스크립트와 구별되는 설정 방법을 제시한다. 여기서 제시하는 rc 스크립트를 수정하는 방법을 사용하지 않는다면 이들 문서는 참고만 한다.

2 표준 make world 프로시져 사용

이 방법은 FreeBSD 4.X 나 5.X 에 적용할 수 있고 최초 시스템을(root 파일시스템뿐만 아니고) DESTDIR 에 완벽하게 설치한다. 단순히 다음 스크립트를 실행한다:

```
#!/bin/sh
export DESTDIR=/data/misc/diskless
mkdir -p ${DESTDIR}
cd /usr/src; make world && make kernel
cd /usr/src/etc; make distribution
```

실행이 끝나면 필요에 따라 /etc/rc.conf 와 /etc/fstab 를 수정해서 DESTDIR 에 둔다.

24.6.2.8 스왑 설정

필요하다면 NFS 를 통해 서버에 있는 스왑 파일에 접근할 수 있다. 일반적으로 여기서 사용되는 방법 중 몇 가지는 5.X 에서는 더 이상 진행되지 않는다.

[NFS 를 통해 스왑 파일사용하기]

1. FreeBSD 4.X 에서 NFS 스왑 사용

스왑 파일 위치와 크기는 FreeBSD 에만 있는 BOOTP/DHCP 옵션 128 과 129 로 지정할 수 있다. ISC DHCP 3.0 이나 bootpd 의 설정파일 예제는 다음 절차를 따른다:

① dhcpd.conf에 다음 라인을 추가한다:

```
# Global section
option swap-path code 128 = string;
option swap-size code 129 = integer 32;

host margaux {
    ... # Standard lines, see above
    option swap-path "192.168.4.4:/netswapvolume/netswap";
    option swap-size 64000;
}
```

swap-path 는 스왑 파일이 있는 디렉터리 경로다. 각 파일은 swap.*client-ip* 라는 이름으로 되어있다.

dhcpd 의 예전 버전은 *option option-128 "...*처럼 더 이상 지원되지 않는 구문을 사용한다.

대신 /etc/bootptab 은 다음 구문을 사용한다:

```
T128="192.168.4.4:/netswapvolume/netswap":T129=64000
```

Note: /etc/bootptab 에서 스왑 크기는 16 진수 포맷으로 표현해야 된다.

② NFS 스왑 파일 서버에서 스왑 파일을 생성한다

```
# mkdir /netswapvolume/netswap
# cd /netswapvolume/netswap
# dd if=/dev/zero bs=1024 count=64000 of=swap.192.168.4.6
# chmod 0600 swap.192.168.4.6
```

192.168.4.8 은 디스크가 없는 클라이언트 IP 다.

③ NFS 스왑 파일 서버의 /etc/exports에 다음 라인을 추가한다:

```
/netswapvolume  -maproot=0:10 -alldirs margaux corbieres
```

그리고 위에서처럼 mountd 가 공유 파일을 다시 읽도록 한다.

2. FreeBSD 5.X 에서 NFS 스왑 사용

커널은 부팅할 때 NFS 스왑을 활성화하지 않는다. 스왑은 쓰기가 가능한 파일시스템으로 마운트하여, 스왑을 생성한 후 활성화하는 시작 스크립트로 활성화해야 된다. 적당한 크기로 스왑을 생성하려면 다음과 같이 실행한다:

```
# dd if=/dev/zero of=/path/to/swapfile bs=1k count=1 oseek=100000
```

활성화하려면 rc.conf 에 다음 라인을 추가해야 된다:

```
swapfile=/path/to/swapfile
```

26.6.2.9 잡다한 문제들

다음과 같이 특별한 경우가 있을 것이다.

① /usr를 읽기 전용으로 운용하기
디스크가 없는 워크스테이션이 X 를 실행할 수 있도록 설정되어있다면 기본적으로 /usr 에 에러 로그를 남기는 xdm 설정파일을 수정해야 된다.

② FreeBSD가 아닌 서버 운용하기
root 파일시스템 서버가 FreeBSD 에서 운용되지 않으면 FreeBSD 머신에서 root 파일시스템을 생성해서 tar 나 cpio 로 타겟 머신에 복사해야 된다.

이 경우 /dev 에 있는 특수 파일의 major/minor 정수 크기가 달라지는 문제가 있다. 이 문제를 해결하는 것은 FreeBSD 가 아닌 서버에서 디렉터리를 공유해서 FreeBSD 머신에 마운트하고, 정확한 장치 엔트리를 생성하도록 FreeBSD 머신에서 MAKEDEV 를 실행한다(FreeBSD 5.0 과 이 후 버전은 devfs(5)로 장치 노드를 생성하기 때문에 이들 버전에서 MAKEDEV 를 실행할 필요가 없다).

24.7 ISDN

ISDN 기술과 하드웨어에 대한 자세한 정보는 Dan Kegel 의 ISDN 페이지(http://www.alumni.caltech.edu/~dank/isdn/)에 있다.

ISDN 의 간단한 로드 맵은 다음을 사항을 따른다:

- 여러분이 유럽에 살고 있다면 ISDN 카드 섹션을 확인해 보기 바란다.
- 다이얼-업 기반의 ISDN으로 인터넷 공급자(전용선을 기반이 아닌)를 통해

인터넷에 연결할 계획이라면 터미널 아답터를 확인해본다. 이 방법을 사용한다면 큰 유연성을 가지고 있어서 인터넷 공급자를 바꾸더라도 별문제가 없을 것이다.

- 두 개의 랜을 같이 연결하거나 ISDN 전용선으로 인터넷에 연결한다면 stand alone 라우터/브리지 옵션을 고려해본다.

사용할 솔루션을 결정하기 가격이 중요한 요소가 되고 있다. 아래 설명은 최소 가격에서 최대 가격까지 순서대로 제시해 줄 것이다.

24.7.1 ISDN 카드

FreeBSD 의 ISDN 은 passive 카드만을 사용하는 DSS1/Q.931(또는 Euro-ISDN) 표준만 지원한다. FreeBSD 4.4 부터는 최초로 지원된 Primary Rate(PRI) ISDN 카드를 포함하여 펌웨어가 다른 신호 프로토콜을 지원하는 몇몇 active 카드도 지원된다.

isdn4bsd 소프트웨어는 raw HDLC(high level data link control 의 약어로 동기식 고속 데이터 전송을 능률적으로 실행하기 위한 제어방식)를 사용한 IP 또는 동기식 PPP 를 사용하여 다른 ISDN 라우터에 연결할 수 있다: *isppp* 로 커널 PPP 를 사용하거나 수정된 sppp(4) 드라이버를 사용하거나 유저 기반 ppp(8)을 사용한다. 유저 기반 ppp(8)를 사용하여 두 개나 그 이상의 ISDN B-채널에서 **채널 본딩**(channel bonding)도 가능하다. 300 Baud 모뎀 소프트웨어처럼 자동 응답 전화 어플리케이션도 사용할 수 있다.

> **채널 본딩(channel bonding):** beowulf 프로젝트에서 선보였던 이 기술은 이더넷 2 개를 같은 IP 로 할당하여 2 배의 전송속도를 낸다. 이더넷 1 개는 outbound 로 다른 하나는 Inbound 로 사용하는 방식이다.

많은 ISDN PC 카드가 FreeBSD 에서 지원하기 때문에 유럽을 포함하여 전 세계에서 성공적으로 사용되고 있다.

지원되는 passive ISDN 카드는 Infineon(이전 Siemens) ISAN/HSCX/IPAC ISDN 칩셋, Cologne 칩(ISA 버스만), W6692 칩의 Winbond 의 PCI 카드, Tiger300/320/ISAC 칩셋과 결합된 카드 그리고 AVM Fritz!Catd PCI v.1.0 과 AVM Fritz!Card PnP 와 같은 특정 벤더 칩셋 기반 카드도 가능하다.

현재 무난하게 지원되는 ISDN 카드는 AVM B1(ISA 와 PCI) BRI 카드와 AVM T1 PCI PRI

카드다.

isdn4bsd 에 대한 문서는 FreeBSD 시스템의 /usr/share/examples/isdn/ 디렉터리를 찾거나 erratas 와 isdn4bsd(http://people.freebsd.org/~hm/) 핸드북과 같은 문서 정보가 있는 isdn4bsd(http://www.freebsd-support.de/i4b/) 홈페이지에서 찾는다.

현재 지원되지 않는 ISDN PC 카드나 isdn4bsd 성능을 개선하는 것처럼 다른 ISDN 프로토콜을 지원하도록 하려면 Hellmuth Michaelis <hm@FreeBSD.org>에게 연락한다.

isdn4bsd 설치, 설정과 문제 해결에 대한 질문은 freebsd-isdn 메일링 리스트(http://lists.freebsd.org/mailman/listinfo/freebsd-isdn)를 이용할 수 있다.

24.7.2 ISDN 터미널 아답터

터미널 아답터(TA)는 일반적인 전화라인을 사용하는 ISDN 모뎀이다.

대부분의 TA 는 표준 Hayes 모뎀 AT 명령 세트를 사용하므로 모뎀에 옮겨서 사용할 수 있다.

TA 는 기본적으로 연결과 속도가 예전 모뎀보다 더 빠른 점을 제외하고 모뎀과 동일하게 동작한다. 따라서 모뎀의 경우와 똑같이 PPP 를 설정해야 된다. 그리고 가능한 최고 속도로 시리얼 속도를 설정한다.

TA 를 사용하여 인터넷 공급자에게 연결하는 가장 큰 장점은 유동적인 PPP 를 사용할 수 있다는 것이다. IP 주소가 점점 부족해지기 때문에 대부분의 인터넷 서비스 공급자들은 더 이상 고정 IP 를 제공하지 않는다. 대부분의 stand-alone 라우터는 유동 IP 할당을 적용하지 않는다.

TA 의 연결특성과 안정성은 운용중인 PPP 데몬에 따라 다르다. 그래서 PPP 를 설정했다면 FreeBSD 머신에서 사용중인 모뎀에서 ISDN 으로 쉽게 업그레이드할 수 있다. 그러나 사용중인 PPP 프로그램에서 문제가 있었다면 이 문제는 업그레이드 후에도 지속된다.

최고의 안정성을 원한다면 userland PPP 가 아닌 커널 PPP 옵션을 사용한다.

다음 TA 는 FreeBSD 와 작동한다.

- Motorola BitSurfer 와 Bitsurfer Pro.
- Adtran.

TA 벤더들이 자신들의 제품에서 거의 모든 표준 모뎀 AT 명령이 실행되도록 노력하기 때문에 대부분의 다른 TA 도 정확하게 동작할 것이다.

외장형 TA 의 실질적인 문제는 모뎀처럼 고급 시리얼 카드가 컴퓨터에 필요하다.

시리얼 장치의 자세한 이해, 동기식과 비동기식 시리얼 포트의 차이점을 알려면 FreeBSD 시리얼 하드웨어 튜토리얼을(http://www.freebsd.org/doc/en_US.ISO8859-1/articles/serial-uart/index.html) 읽어 본다.

128 kbs 회선이 있더라도 TA 가없는 표준 PC 시리얼 포트(비동기)는 115.2kbs 로 제한된다. ISDN 의 128kbs 를 완벽하게 사용하려면 TA 를 동기식 시리얼 카드에 연결해야 된다.

내장형 TA 를 구입하여 동기화/비동기화 문제를 피하려고 하지 않는다. 내장형 TA 에는 단순히 표준 PC 시리얼 포트 칩이 내장되어 있다. 내장형 TA 의 장점은 다른 시리얼 케이블을 구입하여 사용하지 않아도 되고 빈 콘센트를 찾지 않아도 된다는 것뿐이다.

TA 와 동기식 카드는 단순한 386 FreeBSD 머신에서도 최소한 stand-alone 라우터만큼 빠르고 더 유연한 설정이 가능할 것이다.

동기식카드/TA와 stand-alone 라우터를 선택하는 것은 양심적인 문제에 달려있다. 메일링 리스트에 이 문제에 대한 설명이 있고 완벽한 설명은 검색에서(http://www.freebsd.org/search/index.html) 찾아본다.

24.7.3 Stand-alone ISDN 브리지/라우터

ISDN 브리지나 라우터는 FreeBSD 또는 다른 운영체제에 특별한 기능이 아니다. 라우터와 브리지 기술에 대한 완벽한 설명은 네트워크 레퍼런스 책을 참고한다.

이 페이지의 문맥에서 라우터와 브리지라는 용어는 상호 교환하여 사용할 수 있다.

ISDN 라우터/브리지의 가격이 계속 떨어지기 때문에 쉽게 선택할 수 있다. ISDN 라우터는 로컬 이더넷 네트워크에 직접 연결하는 작은 장치이고 다른 브리지/라우터로 연결되는 것을 제어한다. PPP 와 다른 일반적인 프로토콜을 사용하여 통신할 수 있는 소프트웨어가 내장되어 있다.

라우터는 완벽한 동기식 ISDN 회선을 사용하기 때문에 표준 TA 보다 처리량이 훨씬 빠르다.

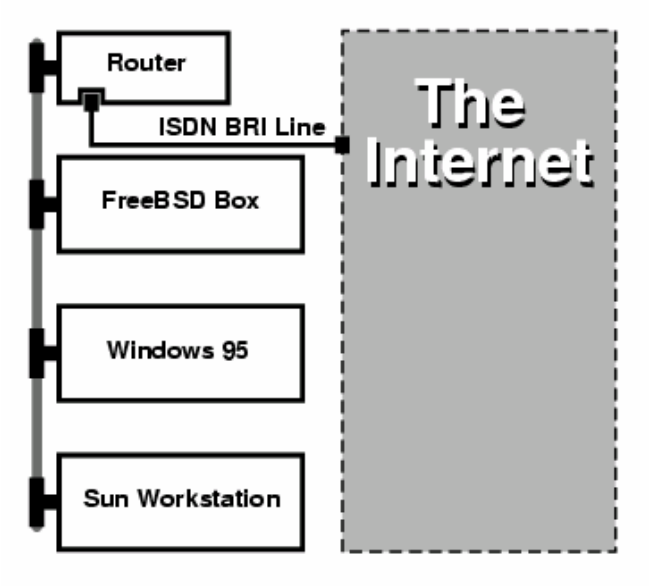
ISDN 라우터와 브리지의 주요 문제는 생산자간의 서비스 상호이용 문제가 아직 존재한다. 인터넷 공급자에게 연결할 계획이라면 이들과 상의해 보아야 될 것이다.

양쪽을 연결할 네트워크 카드만 구입하면 확실하게 동작하기 때문에 홈 네트워크를 사무실 네트워크에 연결하는 것처럼 두 개의 네트워크 구간을 연결할 계획이라면, 이 방법이 가장 간단한 관리 솔루션이다.

예를 들어 집에 있는 컴퓨터나 지점 사무실의 네트워크를 본점 사무실 네트워크에 연결하려면 다음 설정을 따르면 된다.

예제 24-1. 지점 사무실 또는 홈 네트워크

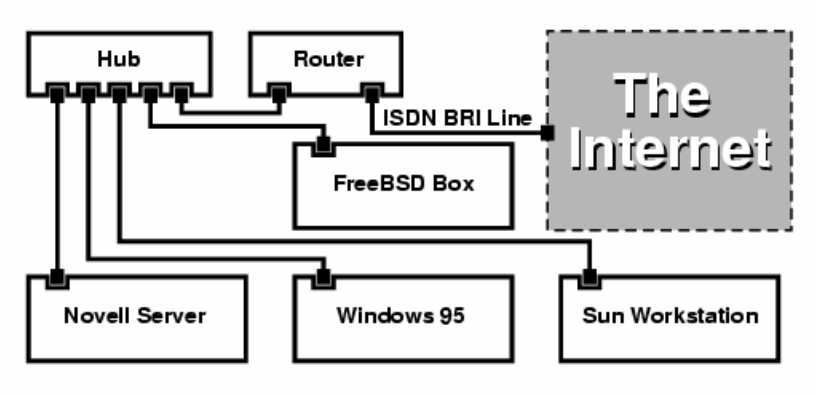
네트워크는 10 base 2 개의 이더넷으로 버스 기반 토플로지를 사용한다. 필요하다면 라우터를 AUI/10BT 트렌시버 네트워크 케이블에 연결한다.



홈/지점 사무실에 한대의 컴퓨터만 있다면 크로스 케이블을 사용하여 stand-alone 라우터에 연결할 수 있다.

예제 24-2. 본점 사무실 또는 다른 네트워크

네트워크는 10 base T 이더넷으로 스타 토플로지를 사용한다.



대부분의 라우터/브리지의 가장 큰 장점은 두 개의 독립된 PPP 회선으로 분리된 2 개의 사이트에 동시에 연결할 수 있다는 것이다. 이 설정은 두 개의 시리얼 포트를 가진 특정 모델을 제외하고 대부분의 TA 가 지원하지 않는다. 채널 본딩, MMP 등과 혼돈하지 않는다.

예를 들어 사무실에 ISDN 전용회선이 있고 다른 ISDN 회선이 필요하지 안다면 이 설정은 매우 유용한 기능이다. 사무실에 있는 라우터를 인터넷과 연결된 B 채널(64Kbps) 전용선에 연결하고 분리된 데이터 연결에 다른 B 채널을 사용할 수 있다. 두 번째 B 채널은 다이얼-인, 다이얼-아웃 또는 더 많은 대역폭을 위해 첫 번째 B 채널과 동적으로 본딩(MPP 등)할 수 있다.

그리고 이더넷 브리지는 IP 트래픽 뿐만 아니라 더 많은 것을 전달할 수 있다. IPX/SPX 와 사용하고 있는 다른 프로토콜도 보낼 수 있다.

24.8 네트워크 주소 변환

24.8.1 요약

일반적으로 natd(8)로 알려져 있는 FreeBSD 의 네트워크 주소 변환 데몬은 들어오는 로(raw) IP 패킷을 허용하고 그 소스를 로컬머신으로 변경하며 이렇한 패킷들을 외부 IP 패킷 스트림으로 다시 내보내는 데몬이다. 말하자면 natd(8)은 데이터를 돌려줄 때 원래 데이터의 위치를 결정할 수 있기 때문에 소스 IP 주소와 포트를 변경하여 최초의 요청자에게 재 전송해 준다.

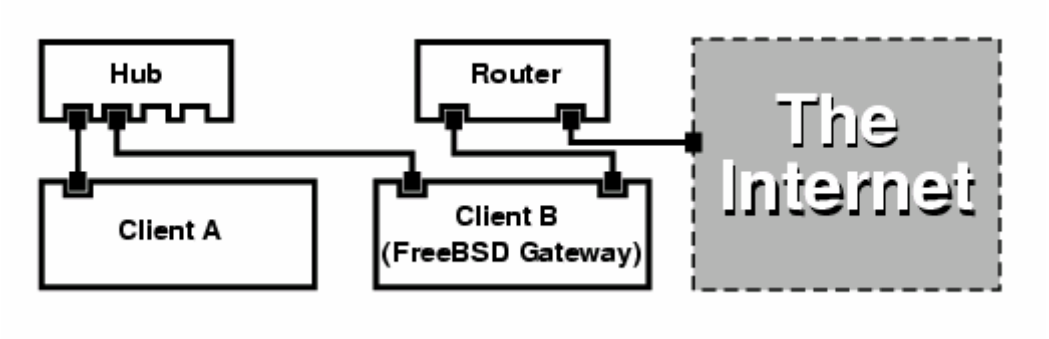
NAT 의 일반적인 용도는 인터넷을 공유하는데 사용한다.

24.8.2 설정

IPv4 에서 IP 주소 부족이나 케이블 또는 DSL 과 같은 고속 인터넷 유저의 증가로 사람들은 인터넷 공유 솔루션의 필요성을 많이 느끼게 됐다. 하나의 인터넷 회선과 IP 주소로 여러 대의 컴퓨터를 온라인에 연결하는 기능이 natd(8)을 선택하는 이유가 되었다.

유저는 보통 하나의 IP 주소와 케이블 또는 DSL 라인에 연결된 한대의 머신으로 LAN 에 있는 여러 대의 컴퓨터를 인터넷에 연결할 수 있기를 바란다.

이렇게 회선을 공유하려면 인터넷에 연결되어 있는 FreeBSD 머신이 게이트웨이처럼 동작해야 된다. 이 게이트웨이 머신은 두 개의 네트워크 카드가 필요하다 -- 하나는 인터넷 라우터에 다른 하나는 LAN 에 연결된다. LAN 의 모든 머신은 허브나 스위치를 통해 연결된다.



일반적으로 인터넷을 공유하는데 이런 식의 설정이 사용된다. LAN 에 있는 머신 중 한대가 인터넷에 연결되어 있고 나머지 머신은 "게이트웨이" 머신을 통해 인터넷에 접속된다.

24.8.3 설정

다음 옵션이 커널 설정파일에 필요하다:

```
options IPFIREWALL
options IPDIVERT
```

게다가 선택사항으로 다음 내용을 추가하는 것도 괜찮을 것 같다:

```
options IPFIREWALL_DEFAULT_TO_ACCEPT
options IPFIREWALL_VERBOSE
```

다음 내용을 /etc/rc.conf 에 추가한다:

```
gateway_enable="YES" ❶
firewall_enable="YES" ❷
firewall_type="OPEN" ❸
natd_enable="YES" ❹
natd_interface="fxp0" ❺
natd_flags=""
```

gateway_enable="YES"	머신이 게이트웨이로 동작하도록 설정한다. sysctl net.inet.ip.forwarding=1 을 실행해도 같은 효과를 얻을 수 있다.
firewall_enable="YES"	부팅할 때 /etc/rc.firewall 에서 방화벽 룰을 활성화한다.
firewall_type="OPEN"	모든 패킷을 허용하도록 방화벽에 미리 내장된 룰을 지정한다. 추가적인 종류는 /etc/rc.firewall 을 확인한다.
natd_interface="fxp0"	어떤 인터페이스로 패킷이 포워드 되는지 보여준다(이 인터페이스는 인터넷에 연결되어 있다).
natd_flags=""	부팅할 때 natd(8)에 설정하는 추가적인 옵션.

/etc/rc.conf 에 지정된 이전 옵션이 부팅할 때 natd -interface fxp0 를 실행한다. 이 명령을 수동으로도 실행할 수 있다.

> **Note:** natd(8)에 너무 많은 옵션이 있기 때문에 설정파일을 사용할 수 있다. 이 경우 설정파일은 /etc/rc.conf 에 정의되어 있어야 된다:
>
> ```
> natd_flags="-f /etc/natd.conf"
> ```
>
> /etc/natd.conf 파일에는 라인 별로 설정옵션이 하나씩 지정되어 있다. 예를 들면 다음 섹션의 경우 다음 파일을 사용한다:

```
redirect_port tcp 192.168.0.2:6667 6667
redirect_port tcp 192.168.0.3:80 80
```

설정파일에 대한 자세한 정보는 natd(8) 매뉴얼 페이지의 *-f* 옵션을 참고한다.

LAN 의 뒤 단에 있는 각 머신과 인터페이스는 RFC 1918 에 정의되어있는 사설 네트워크 IP 주소와 natd 머신의 내부 IP 주소를 기본 게이트웨이로 할당 받는다.

예를 들어 LAN 뒷단의 클라이언트 A 와 B 는 192.168.0.2 와 192.168.0.3 의 IP 주소를 할당 받지만 natd 머신의 LAN 인터페이스는 192.168.0.1 의 IP 주소로 설정한다. 클라이언트 A 와 B 의 기본 게이트웨이를 natd 머신의 192.168.0.1 로 설정해야 된다. natd 머신의 외부 또는 인터넷 인터페이스에는 natd(8)가 작동하기 위해 특별히 수정할 것은 없다.

24.8.4 포트 리다이렉션

natd 의 단점은 인터넷에서 LAN 에 있는 클라이언트에게 접근할 수 없다는 것이다. LAN 에 있는 클라이언트는 외부로 연결할 수 있지만 외부에서 연결할 수 없다. 이것은 LAN 에 있는 클라이언트 머신으로 인터넷 서비스를 할 때 문제가 된다. 이 문제를 해결하는 단순한 방법은 natd 머신의 선택된 인터넷 포트를 LAN 에 있는 클라이언트로 리다이랙트한다.

예를 들어 IRC 서버가 클라이언트 A 에서 운용되고 웹 서버는 클라이언트 B 에서 운용된다면 정확히 작동하도록 포트 6667(IRC)과 80(web)으로 들어오는 연결을 각 머신으로 리다이랙트 해야 된다.

적절한 옵션으로 *-redirect_port* 를 natd(8)에 적용한다. 구문은 다음과 같다:

```
-redirect_port proto targetIP:targetPORT[-targetPORT]
               [aliasIP:]aliasPORT[-aliasPORT]
               [remoteIP[:remotePORT[-remotePORT]]]
```

위의 예제에서 인자는 다음과 같다:

```
-redirect_port tcp 192.168.0.2:6667 6667
-redirect_port tcp 192.168.0.3:80 80
```

이 인자는 적절한 tcp 포트를 LAN 에있는 클라이언트 머신에게 리다이랙트한다.

-redirect_port 인자는 각 포트뿐만 아니라 포트의 범위를 지정하는데 사용할 수 있다. 예를 들어 *tcp 192.168.0.2:2000-3000 2000-3000* 은 포트 2000 에서 3000 으로 들어오는 모든 연결을 클라이언트 A 의 2000 에서 3000 번으로 모두 리다이랙트한다.

이들 옵션은 natd(8)을 실행할 때 직접 입력, /etc/rc.conf 의 *natd_flags=""*에 입력하여 또는 설정파일을 통해 적용할 수 있다.

더 많은 설정옵션은 natd(8)을 참고한다.

24.8.5 주소 리다이랙션

주소 리다이랙션은 여러 개의 IP 주소를 사용할 수 있을 때 유용하지만 머신 한대에 이 주소들을 할당해야 된다. 이렇게 해서 natd(8)은 각 LAN 클라이언트에 각자의 외부 IP 주소를 할당할 수 있다. natd(8)은 LAN 에있는 클라이언트로부터 외부로 나가는 패킷을 적절한 외부 IP 주소로 재 작성하고, 특정 IP 주소로 입력되는 모든 트래픽을 지정된 LAN 클라이언트로 리다이랙트한다. 이러한 방식을 고정적 NAT 라고도 한다. 예를 들면 IP 주소 128.1.1.1, 128.1.1.2, 그리고 128.1.1.3 이 natd 게이트웨이 머신에 있다. 128.1.1.1 을 natd 게이트웨이 머신의 외부 IP 주소로 사용할 수 있지만 128.1.1.2 와 128.1.1.3 은 LAN 클라이언트 A 와 B 에 포워드된다.

-redirect_address 구문은 다음과 같다:

```
-redirect_address localIP publicIP
```

localIP	LAN 클라이언트의 내부 IP 주소
publicIP	LAN 클라이언트에 대응되는 외부 IP 주소

예제에서 이 인자는 다음과 같다:

```
-redirect_address 192.168.0.2 128.1.1.2
-redirect_address 192.168.0.3 128.1.1.3
```

-redirect_port 처럼 이들 인자도 /etc/rc.conf 의 *natd_flags=""* 옵션이나 설정파일을 통해 적용할 수 있다. 주소 리다이랙션에서 특정 IP 주소로 들어오는 모든 데이터는 리다이랙트되기 때문에 포트 리다이랙션은 필요 없다.

natd 머신의 외부 IP 주소를 활성화하고 외부 인터페이스로 엘리어스 되어야 한다. rc.conf(5)을 확인한다.

24.9 Parallel Line IP (PLIP)

PLIP 는 양쪽 패러럴 포트에서 TCP/IP 를 사용할 수 있게 한다. 네트워크 카드가 없는 머신이나 노트북에 설치할 때 유용하다. 이 섹션에서는 다음과 같은 사항을 다룬다:

- 패러럴 케이블(laplink) 만들기
- PLIP로 두 대의 컴퓨터 연결하기

24.9.1 패러럴 케이블 만들기

대부분의 컴퓨터 소매점에서 패러럴 케이블을 구입할 수 있다. 구입할 수 없거나 직접 만드는 방법을 알고 싶다면, 다음 표에서 일반 패러럴 프린터 케이블이 아닌 네트워크용 패러럴 케이블 만드는 것을 보여준다

표 24-1. 네트워크용 패러럴 케이블 배선

A-name	A-End	B-End	Descr.	Post/Bit
DATA0	2	15	Data	0/0x01
-ERROR	15	2		1/0x08
DATA1	3	13	Data	0/0x02
+SLCT	13	3		1/0x10

DATA2 +PE	4 12	12 4	Data	0/0x04 1/0x20
DATA3 -ACK	5 10	10 5	Strobe	0/0x08 1/0x40
DATA4 BUSY	6 11	11 6	Data	0/0x10 1/0x80
GND	18-25	18-25	GND	-

24.9.2 PLIP 설정

랩 링크(laplink) 케이블을 준비하자. 그리고 양쪽 컴퓨터의 커널에서 lpt(4) 드라이버를 지원하는지 확인한다:

```
# grep lp /var/run/dmesg.boot
lpt0: <Printer> on ppbus0
lpt0: Interrupt-driven port
```

FreeBSD 4.X 에서 패러럴 포트는 interrupt driven 포트(포트의 인터럽트를 사용하여 데이터를 전송하는)여야 되고 커널 설정파일에 다음과 같은 라인이 있는지 확인한다:

```
device ppc0 at isa? irq 7
```

FreeBSD 5.X 에서는 /boot/device.hints 파일에 다음 라인이 필요하다:

```
hint.ppc.0.at="isa"
hint.ppc.0.irq="7"
```

커널 설정파일에 *device plip* 라인이 있는지 또는 plip.ko 커널 모듈이 로드 되었는지 확인한다. 이 두 가지 경우 패러럴 네트워크 인터페이스는 ifconfig(8) 명령을 직접 사용할 때 나타나야 한다. FreeBSD 4.X 에서는 다음과 같이 나타나고:

```
# ifconfig lp0
lp0: flags=8810<POINTOPOINT,SIMPLEX,MULTICAST> mtu 1500
```

그리고 FreeBSD 5.X 는 아래와 같이 나타난다:

```
# ifconfig plip0
plip0: flags=8810<POINTOPOINT,SIMPLEX,MULTICAST> mtu 1500
```

> **Note:** 패레럴 인터페이스에 사용하는 장치 이름은 FreeBSD 4.X (lpx)와 FreeBSD 5.X (plipx)에서 다르다.

랩 링크 케이블을 두 컴퓨터의 패러럴 인터페이스에 꽂는다.

root 유저에서 양쪽 네트워크 인터페이스의 매개변수를 설정한다. 예를 들면 FreeBSD 4.X 로 운용되는 호스트 host1 과 FreeBSD 5.X 로 운용되는 호스트 host2 를 연결하려면 다음과 같이 설정한다:

host1 <-----> host2		
IP Address	10.0.0.1	10.0.0.2

host1 에서는 다음과 같이 인터페이스를 설정한다:

```
# ifconfig lp0 10.0.0.1 10.0.0.2
```

host2 에서는 다음과 같이 인터페이스를 설정한다:

```
# ifconfig lp0 10.0.0.2 10.0.0.1
```

이제 연결이 정상적으로 동작한다. 더 자세한 사항은 lp(4)와 lpt(4) 매뉴얼 페이지를 참고한다.

그리고 양쪽 호스트를 /etc/hosts 에 추가해야 된다:

127.0.0.1	localhost.my.domain localhost
10.0.0.1	host1.my.domain host1
10.0.0.2	host2.my.domain

연결이 정상인지 확인하려면 각 호스트에서 상대 컴퓨터로 ping 테스트를 한다. 예를 들어

host1 에서 다음과 같이 실행한다:

```
# ifconfig lp0
lp0: flags=8851<UP,POINTOPOINT,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        inet 10.0.0.1 --> 10.0.0.2 netmask 0xff000000
# netstat -r
Routing tables

Internet:
Destination        Gateway            Flags      Refs      Use       Netif Expire
host2              host1              UH            4   127592         lp0
# ping -c 4 host2
PING host2 (10.0.0.2): 56 data bytes
64 bytes from 10.0.0.2: icmp_seq=0 ttl=255 time=2.774 ms
64 bytes from 10.0.0.2: icmp_seq=1 ttl=255 time=2.530 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=255 time=2.556 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=255 time=2.714 ms

--- host2 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/stddev = 2.530/2.643/2.774/0.103 ms
```

24.10 IPv6

IPv6(또한 IPng "IP next generation"으로 유명한)는 유명한 IP 프로토콜(IPv4 로 알고 있는)의 다음 버전이다. 현재 다른 *BSD 시스템처럼 FreeBSD 도 KAME IPv6 레퍼런스 실행을 포함하고 있다. 그래서 FreeBSD 시스템은 IPv6 에 필요한 모든 것을 가지고 있다. 이번 섹션은 IPv6 설정과 운용에 대해 초점을 맞췄다.

1990 년대 초기에 사람들은 IPv4 의 주소가 급속히 감소하는 것을 감지하였다. 인터넷의 팽창에 따라 두 가지 주된 문제가 나타나고 있다:

- 주소 부족. 요즘 이 문제는 사설 IP 주소와(10.0.0.0/8, 192.168.0.0/24등) 네트워크 주소 변환(NAT)으로 더 이상 염려되지 않는다.
- 라우팅 테이블 엔트리가 점점 커진다. 이것은 요즘에도 문제가 되고 있다.

IPv6 에서 이들 문제와 다른 쟁점:

- 128 bit 주소 공간. 다시 말해서 이론적으로 340,282,366,920,938,463,463,374,607,431,768,211,456의 주소를 사용할 수 있다. 이 의미는 우리 행성에서 평방 미터당 대략 6.67 * 10^27의 IPv6 주소가 있다는 것이다.
- 라우터는 라우팅 테이블에 네트워크 어그리게이션(aggregation) 주소만 저장하면 되기 때문에 라우팅 테이블의 평균 공간을 8192 엔트리로 줄일 수 있다.

다음과 같이 IPv6 의 다양하고 유용한 특성이 있다:

- 주소 자동 설정 (RFC2462).
- 애니캐스트(Anycast) 주소
- 필수적인 멀티스트(Multicast) 주소
- IPsec (IP 보안)
- 간략한 헤더 구조
- 모바일 IP
- IPv4를 IPv6로 변환하는 메커니즘

더 많은 정보는 다음 사이트를 참고한다:

- playground.sun.com에서(http://www.sun.com/) IPv6 요약
- KAME.net (http://www.kame.net/)
- 6bone.net (http://www.6bone.net/)

24.10.1 IPv6 주소에 대한 백그라운드

유니캐스트(Unicast), 애니캐스트(Anycast)와 멀티캐스트(Multicast)의 IPv6 주소가 있다.

유니캐스트 주소는 일반적인 주소다. 패킷을 유니캐스트 주소로 보내면 주소에 속한 인터페이스에 정확하게 도달한다.

애니캐스트 주소는 유니캐스트 주소에서 구문적으로는 구분할 수 없지만 이들 인터페이스의 그룹 주소로는 구분할 수 있다. 애니캐스트 주소로 향하는 패킷은 가장 가까운(라우터 측정) 인터페이스에 도달한다. 애니캐스트 주소는 라우터에서만 사용될 것이다.

멀티캐스트 주소는 인터페이스 그룹을 식별한다. 멀티캐스트 주소로 향하는 패킷은 멀티캐스트 그룹에 속하는 모든 인터페이스에 도달한다.

> Note: IPv4 브로드케스트 주소는(보통 xxx.xxx.xxx.255) IPv6 에서 멀티캐스트 주소로 표현된다.

표 24-2. 예약되어 있는 IPv6 주소

Ipv6-주소	미리 예약되어 있는 길이(Bits)	설 명	비 고
::	128 비트	지정되어 않음 (unspecified)	예: IPv4 에서 0.0.0.0
::1	128 비트	루프백 주소	예: IPv4 에서 127.0.0.1
::00:xx:xx:xx:xx	96 비트	IPv4 에 내장되어있음	낮은 32 비트 주소들은 IPv4 주소다. 그리고 IPv6 주소와 호환되는 IPv4 라고 부른다.
::ff:xx:xx:xx:xx	96 비트	IPv4 를 IPv6 주소로 매핑	낮은 32 비트 주소들은 IPv4 주소다. IPv6 를 지원하지 않는 호스트를 위한 것이다.
fe80:: - feb::	10 비트	link-local	cf. loopback address in IPv4
fec0:: - fef::	10 비트	site-local	
ff::	8 비트	멀티캐스트	
001 (base 2)	3 비트	글로벌 유니캐스트 (global unicast)	모든 글로벌 유니캐스트 주소는 이 풀(pool)에서 할당된다. 처음 3 비트는 “001”이다.

24.10.2 IPv6 주소 읽기

규정된 형식은 x:x:x:x:x:x:x:x 으로 표현되고 각 "x"은 16 비트 16 진수 값이다. 예를 들면 FEBC:A574:382B:23C1:AA49:4592:4EFE:9982 과 같이 읽는다.

가끔 주소는 모두 0 인 긴 서브스트링을 가지고 있으므로 각 서브스트링은 "::"으로 줄일 수 있다. 예를 들어 fe80::1 은 규정된 형식 fe80:0000:0000:0000:0000:0000:0000:0001 과 동일하다.

세 번째 형식은 IPv4 스타일로 유명한(10 진수) 점 "."으로 32Bit 씩 분리하여 작성한다. 예를 들어 2002::10.0.0.1 는 2002::a00:1 과 동일하게 규정된 표현식 2002:0000:0000:0000:0000:0000:0a00:0001 과 일치한다.

이제 다음 내용을 읽을 수 있다:

```
# ifconfig
rl0: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu
1500
        inet 10.0.0.10 netmask 0xffffff00 broadcast 10.0.0.255
        inet6 fe80::200:21ff:fe03:8e1%rl0 prefixlen 64 scopeid 0x1
        ether 00:00:21:03:08:e1
        media: Ethernet autoselect (100baseTX )
        status: active
```

fe80::200:21ff:fe03:8e1%rl0 은 로컬 주소로 링크되도록 자동으로 설정된다. 자동 설정의 일부분으로 MAC 주소에서 생성된다.

IPv6 주소의 구조에 대한 더 자세한 사항은 RFC2373 을 참고한다

24.10.3 연결

현재 다른 IPv6 호스트와 네트워크에 연결하는 4 가지 방법이 있다:

- 실험적으로 6bone에 등록한다.
- ISP로부터 IPv6 네트워크를 제공받고 설명은 인터넷 공급자에게 문의한다.
- IPv6을 IPv4로 변경한다.
- 다이얼-업 연결을 사용한다면 net/freenet6 포트를 사용한다.

현재 가장 일반적인 방법이므로 여기서는 6bone 에 연결하는 방법을 설명한다

첫째로 6bone 사이트(http://www.6bone.net/)에서 가장 가까운 6bone 회선을 찾는다. 약간의 행운이 있다면 책임자에게 글을 보내서 설정하는 방법을 들을 수 있다(http://www.ipv6.or.kr/ 사이트를 참고한다). 보통 GRE (gif)터널 설정이 필요하다.

일반적인 gif(4) 터널을 설정하는 예제가 있다:

```
# ifconfig gif0 create
# ifconfig gif0
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
# ifconfig gif0 tunnel MY_IPv4_ADDR  HIS_IPv4_ADDR
# ifconfig gif0 inet6 alias MY_ASSIGNED_IPv6_TUNNEL_ENDPOINT_ADDR
```

대문자 단어는 6bone 노드에서 받은 정보로 교체한다.

위 설정으로 구축한 터널이 작동하는지 ping6(8) 'ing ff02::1%gif0 로 체크하면 두 개의 핑이 되돌아오는 것을 볼 수 있다.

> **Note:** ff02:1%gif0 주소에 관심이 있다면 이것은 멀티캐스트 주소다. *%gif0*은 네트워크 인터페이스 gif0 에 멀티캐스트 주소가 사용됨을 나타낸다. 터널의 다른 끝단에 멀티캐스트 주소로 ping 을 보내기 때문에 응답이 오는 것이다.

라우터를 6bone 업 링크로 설정하는 것은 상당히 직관적이다:

```
# route add -inet6 default -interface gif0
# ping6 -n MY_UPLINK
# traceroute6 www.jp.FreeBSD.org
(3ffe:505:2008:1:2a0:24ff:fe57:e561) from 3ffe:8060:100::40:2, 30 hops max, 12 byte
packets
 1  atnet-meta6  14.147 ms  15.499 ms  24.319 ms
 2  6bone-gw2-ATNET-NT.ipv6.tilab.com  103.408 ms  95.072 ms *
 3  3ffe:1831:0:ffff::4  138.645 ms  134.437 ms  144.257 ms
 4  3ffe:1810:0:6:290:27ff:fe79:7677  282.975 ms  278.666 ms  292.811 ms
 5  3ffe:1800:0:ff00::4  400.131 ms  396.324 ms  394.769 ms
 6  3ffe:1800:0:3:290:27ff:fe14:cdee  394.712 ms  397.19 ms  394.102 ms
```

이 결과는 머신 마다 다르다. 모질라 같은 브라우저에서 IPv6 를 활성화했다면 IPv6 사이트 www.kame.net 에 접속해서 춤추는 거북이를 볼 수 있다.

24.10.4 IPv6 에서 DNS

IPv6 를 위한 두 가지 종류의 DNS 레코드가 사용되었다. IETF 에서 A6 레코드를 사용하지 않는다고 선언해서 이제 AAAA 레코드가 표준이다.

AAAA 레코드를 사용하는 것도 역시 상당히 직관적이다. 호스트 네임에 새로운 IPv6 주소를 할당하려면 다음 내용을 주 DNS 존 파일에 추가한다.

```
MYHOSTNAME              AAAA      MYIPv6ADDR
```

이 경우 여러분의 DNS 존을 DNS 공급자에게 제공할 필요가 없다. 현재 bind 버전과(버전 8.3 과 9) dns/djbdns(IPv6 패치)에서 AAAA 레코드를 지원한다.

24.10.5 /etc/rc.ocnf 변경

24.10.5.1 IPv6 클라이언트 설정

여기서는 LAN 에서 클라이언트로 동작하는 머신을 설정하는 방법을 지원한다. 부팅할 때 인터페이스에 rtsol(8)이 자동으로 설정되도록 하려면 다음 라인을 추가한다:

```
ipv6_enable="YES"
```

fxp0 인터페이스에 2001:471:1f11:251:290:27ff:fee0:2093 과 같은 IP 를 고정적으로 할당하려면 다음을 추가한다:

```
ipv6_ifconfig_fxp0="2001:471:1f11:251:290:27ff:fee0:2093"
```

2001:471:1f11:251::1 의 기본 라우터를 추가하려면 /etc/rc.conf 에 다음 내용을 추가한다:

```
ipv6_defaultrouter="2001:471:1f11:251::1"
```

24.10.5.2 IPv6 라우터 / 게이트웨이 설정

여기서는 6bone(http://www.6bond.net/)같은 터널 제공자가 보내준 자료를 설정하고 재부팅하더라도 설정이 지속되도록 도와준다. 시작할 때 터널을 복원하기 위해 다음과 같은 설정을 /etc/rc.conf에서 사용한다:

설정하려는 일반적인 터널링 인터페이스를 나열한다. 예를 들어 gif0 는 다음과 같이 추가한다:

```
gif_interfaces="gif0"
```

로컬 쪽 끝단 *MY_IPv4_ADDR*을 원격지의 끝단 *REMOTE_IPv4_ADDR*로 인터페이스를 설정하려면 다음 내용을 추가한다:

```
gif_config_gif0="MY_IPv4_ADDR REMOTE_IPv4_ADDR"
```

할당한 IPv6 주소를 여러분의 IPv6 터널 끝단에 적용하려면 다음 내용을 추가한다:

```
ipv6_ifconfig_gif0="MY_ASSIGNED_IPv6_TUNNEL_ENDPOINT_ADDR"
```

마지막으로 IPv6 기본 라우트를 설정한다. 다음은 반대 쪽 IPv6 터널이다:

```
ipv6_defaultrouter="MY_IPv6_REMOTE_TUNNEL_ENDPOINT_ADDR"
```

24.10.6 라우터 알림(Router Advertisemen)와 호스트 자동 설정

이 섹션은 IPv6 기본 라우트를 알리도록 rtadvd(8)를 설정한다.

라우터 알림(Router Advertisement): IPv6 라우터는 라우터 알림 메시지를 정기적으로 또는 라우터 요청(Router Solicitation) 메시지에 대한 응답으로 보내며, 여기에는 hop limit,

링크 접두사, 링크 MTU, 라우터 작동시간 등 호스트에서 필요로 하는 정보가 포함된다.

rtadvd(8)을 활성화하려면 /etc/rc.conf 에 다음 라인을 추가한다:

```
rtadvd_enable="YES"
```

IPv6 라우터 요청(Router solicitation)을 하는 인터페이스를 지정하는 것은 중요하다.

라우터 요청(Router Solicitation): 라우터 요청 메시지는 링크에 있는 IPv6 라우터를 검색하기 위해 IPv6 호스트에서 전송하며, 호스트는 정기적인 라우터 메시지를 기다리지 않고 멀티캐스트 라우터 요청 메시지로 IPv6 라우터에게 즉시 응답하라는 메시지(Router Advertisement)를 보낸다.

예를 들어 rtadvd(8)에게 fxp0 를 사용하도록 하려면 다음 라인을 추가한다:

```
rtadvd_interfaces="fxp0"
```

이제 /etc/rtadvd.conf 설정파일을 생성해야 된다. 여기 예제가 있다:

```
fxp0:₩
    :addrs#1:addr="2001:471:1f11:246::":prefixlen#64:tc=ether:
```

fxp0 대신 사용하려는 인터페이스를 지정한다.

2001:471:1f11:246::는 여러분에게 할당된 값으로 변경한다.

/64 서브넷 전체를 사용한다면 변경할 것이 없지만 그렇지 않으면 *prefixlen#*를 정확한 값으로 변경한다.

24.11 FreeBSD 5.X 에서 ATM

24.11.1 ATM(PVCs)을 통한 클래식컬 IP 설정

Classical IP over ATM(CLIP)은 IP 로 ATM 을 사용하는 가장 단순한 방법이다. switched connections(SVCs)과 permanent connections(PVCs)을 사용할 수 있다. 이 섹션은 PVCs 기반 네트워크를 어떻게 설정하는지 설명한다.

24.11.1.1 완전한 네트워크 설정

PVCs 로 CLIP 를 설정하는 첫 번째 방법은 전용 PVC 로 각 시스템을 다른 시스템과 연결한다. 이 방법은 설정하기 쉽지만 시스템이 많아지면 실용성이 떨어진다. 예를 들어 네트워크에 4 대의 시스템이 있고 각 시스템은 ATM 아답터 카드로 ATM 네트워크와 연결되어 있다고 가정한다. 첫 단계는 IP 주소 할당 계획과 시스템간의 ATM 연결이다. 다음 호스트와 IP 를 사용한다고 가정한다:

호스트	IP 주소
hostA	192.168.173.1
hostB	192.168.173.2
hostC	192.168.173.3
hostD	192.168.173.4

완벽하게 연결되어 있는 네트워크를 만들기 위해 각 시스템 쌍에 ATM 을 연결해야 된다:

Machines	VPI.VCI couple
hostA - hostB	0.100
hostA - hostC	0.101
hostA - hostD	0.102
hostB - hostC	0.103
hostB - hostD	0.104
hostC - hostD	0.105

각 연결의 끝에서 VPI 와 VCI 값은 다를 수 있지만 여기서는 간단히 같다고 여긴다. 그리고 각 호스트의 ATM 인터페이스를 설정해야 된다:

```
hostA# ifconfig hatm0 192.168.173.1 up
hostB# ifconfig hatm0 192.168.173.2 up
hostC# ifconfig hatm0 192.168.173.3 up
hostD# ifconfig hatm0 192.168.173.4 up
```

모든 호스트의 ATM 인터페이스가 hatm0 이라고 가정하고 PVCs 를 hostA 에 설정한다(이미 ATM 스위치가 설정되어 있다고 가정하고 스위치 설정은 매뉴얼 페이지를 참고한다).

```
hostA# atmconfig natm add 192.168.173.2 hatm0 0 100 llc/snap ubr
hostA# atmconfig natm add 192.168.173.3 hatm0 0 101 llc/snap ubr
hostA# atmconfig natm add 192.168.173.4 hatm0 0 102 llc/snap ubr

hostB# atmconfig natm add 192.168.173.1 hatm0 0 100 llc/snap ubr
hostB# atmconfig natm add 192.168.173.3 hatm0 0 103 llc/snap ubr
hostB# atmconfig natm add 192.168.173.4 hatm0 0 104 llc/snap ubr

hostC# atmconfig natm add 192.168.173.1 hatm0 0 101 llc/snap ubr
hostC# atmconfig natm add 192.168.173.2 hatm0 0 103 llc/snap ubr
hostC# atmconfig natm add 192.168.173.4 hatm0 0 105 llc/snap ubr

hostD# atmconfig natm add 192.168.173.1 hatm0 0 102 llc/snap ubr
hostD# atmconfig natm add 192.168.173.2 hatm0 0 104 llc/snap ubr
hostD# atmconfig natm add 192.168.173.3 hatm0 0 105 llc/snap ubr
```

주어진 ATM 아답터로 이들을 지원하는데 UBR 이 아닌 다른 트래픽 contracts 를 사용할 수 있다. 이 같은 경우 트래픽 contract 의 이름을 트래픽 매개변수 뒤에 지정한다. 다음 명령으로 atmconfig(8) 툴 도움말을 보거나 atmconfig(8) 매뉴얼 페이지를 참고한다.

```
# atmconfig help natm add
```

같은 설정을 /etc/rc.conf 로도 할 수 있다. hostA 는 다음과 비슷하다:

```
network_interfaces="lo0 hatm0"
ifconfig_hatm0="inet 192.168.173.1 up"
natm_static_routes="hostB hostC hostD"
route_hostB="192.168.173.2 hatm0 0 100 llc/snap ubr"
route_hostC="192.168.173.3 hatm0 0 101 llc/snap ubr"
route_hostD="192.168.173.4 hatm0 0 102 llc/snap ubr"
```

모든 CLIP 라우트의 현재 상태는 다음 명령으로 볼 수 있다:

```
hostA# atmconfig natm show
```