IPv6 응용 전환 기술

Application Technologies for IPv6 Transition

신명기(M.K. Shin) 차세대인터넷표준연구팀 선임연구원
홍용근(Y.G. Hong) 차세대인터넷표준연구팀 연구원
이숙영(S.Y. Lee) 차세대인터넷표준연구팀 연구원
이주철(J.C. Lee) 차세대인터넷표준연구팀 연구원
김용진(Y.J. Kim) 차세대인터넷표준연구팀 책임연구원, 팀장

본 고에서는 IPv6의 응용 전환 기술에 관해 소개한다. 차세대인터넷 IPv6의 도입을 고려할 때는 망의 전환 뿐만 아니라 각 호스트에 존재하는 응용의 전환도 함께 고려되어야 한다. 본 고에서는 IPv4 응용에서 IPv6 응용으로 전이되는 과정에서 발생하는 문제점들을 분류하고, IPv6 전환단계 동안 응용 개발자, 관리자들이 혼동 없이 자연스럽게 IPv6 응용을 개발, 선택할 수 있도록 가이드라인을 제시한다.

I. 서 론

이미 널리 확산되어 많은 사용자를 확보하고 있는 기존의 IPv4(Internet Protocol version 4) 인터넷 망을 IPv6(Internet Protocol version 6) 기반의 차세대인터넷 망으로 전환하기 위해서는 프로토콜, 라우팅, 어드레싱 그리고 DNS(Domain Name Service) 등을 체계적으로 전환하기 위한 고려사항들이 필요하다. IPv6의 성공적인 전이를 위한 기본적인 방법들로는 프로토콜 관점에서 IPv4/IPv6 듀얼스택 등의 기술들이 포함된 RFC 2893[1]과 RFC2185[2] 등을 들 수가 있다. 망의 전환 관점에서 보면, NAT-PT(Network Address Translation-Protocol Translation)-RFC2766[3], SIIT (Stateless IP/ICMP Translation)-RFC2765[4], 6 to 4-RFC 3056[5], 그리고 DSTM(Dual Stack Transition Mechanism)[6] 등과 같은 다양한 IPv4/IPv6 전환 메커니즘들이 개발되어 있다. 그러나 이러한 IPv4/IPv6 전환 메커니즘들이 도입된다고 해도, 호스트 내에 있는 응용의 전환이 완료되어야 비로소 IPv6로의 완전한 전환이 가능하게 되었다고 볼 수 있다.

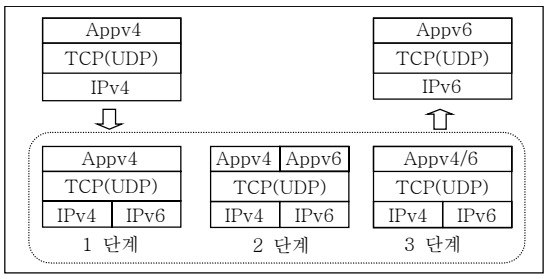
본 고에서는 이러한 응용관점에서의 IPv6 전환에 대해 기술한다. 응용전환의 초기단계에는 BIS (Bump-In-the-Stack)-RFC2767[7]나 BIA(Bump-In-the-API)[8]가 사용될 수 있으나, 이것은 응용의 소스 코드가 존재하지 않는 경우에만 가능하므로 결국 응용 개발자들은 IPv6로의 전이를 위해 기존의 혹은 새롭게 개발되는 응용들을 수정해야만 한다. 이러한 과정에서 IPv4 스택과 IPv6 스택을 모두 포함하는 듀얼스택 상에 다양한 전환 기술과 다양한 응용 버전(IPv4 전용응용, IPv6 전용응용, IPv4와 IPv6를 모두 지원하는 응용)이 혼재하는 시기가 등장할 것이며, 이를 위해 응용 개발자 및 사용자는 자신의 환경에 맞는 적절한 해결책이 필요하다. 따라서 본 고에서는 IPv4/IPv6 전환기간 중에 응용 개발자나 관리자가 겪을 수 있는 문제점들을 먼저 분류하고 각 단계에 대한 해결책과 가이드라인을 함께 제시한다. 이

를 통해 다양한 전환 기술을 이용하여 응용 개발자와 관리자가 혼돈 없이 IPv6 망으로 진화할 수 있도록 한다.

II. IPv6 응용 전환 단계

기존의 IPv4를 사용하던 호스트가 IPv6로 진화하는 과정중에 발생할 수 있는 다양한 호스트 스택 및 응용 버전의 단계는 (그림 1)과 같다[9].

- 1단계: 듀얼스택에 IPv4 전용응용만이 존재하는 경우–IPv6 전개의 초기 단계에 해당하며, IPv6 프로토콜 스택은 존재하나 아직 IPv6 응용은 없는 경우에 해당한다.
- 2단계: 듀얼스택에 IPv4 전용응용과 IPv6 전용응용이 분리되어 함께 존재하는 경우–똑같은 응용이 IPv4와 IPv6 두 개의 버전으로 분리되어 있는 경우에 해당한다.
- 3단계: 듀얼스택에 IPv4와 IPv6를 동시에 지원하는 하나의 응용이 존재하는 경우–하나의 응용이 IPv4와 IPv6 연결을 모두 지원할 수 있다. 따라서 기존의 IPv4 전용응용이 별도로 필요치 않다.



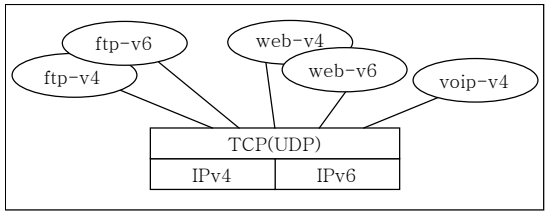
(그림 1) IPv6 응용 전환 단계

III. IPv6 응용 전환시 문제점

1. 듀얼스택 vs. 응용 버전

(그림 1)의 각 전환 단계를 고려했을 때, 상당기간 하나의 호스트에 기존의 IPv4 스택과 IPv6 스택이 공존하는 시기가 존재한다. 응용의 전환 관점에서 봤을 때, 이러한 기간 동안에는 IPv6 스택을 지원하는 IPv6 응용은 기존에 이미 서비스를 제공하고 있던 IPv4 스택을 지원하는 응용과는 독립적으로 도입되게 된다. 바꾸어 얘기하면, 한 호스트에서의 듀얼스택 프로토콜이 존재한다는 의미는 IPv4와 IPv6 응용 모두가 존재한다는 점을 포함하는 것이 아니라는 점이다. 예를 들어 (그림 2)에서 볼 수 있듯이 한 듀얼스택 호스트 내에 기존의 ftp나 www 등은 IPv4와 IPv6 응용이 모두 존재하지만, 새로운 voip와 같은 응용은 IPv4 응용은 존재하지만, IPv6 응용은 아직 존재하지 않을 수 있다.

따라서, 초기에는 듀얼스택을 갖추고 있긴 하나 IPv6 응용을 제공하고 있지 않는 호스트도 존재하게 되며, 이런 경우 외부로부터 임의의 응용과 IPv6 연결은 설정되었으나 IPv6 응용이 없어 연결이 다시 해제되는 비효율적인 일이 발생할 수도 있다. IPv4 스택과 IPv6 스택이 공존하는 시기에는 IPv4 전용응용, IPv6 전용응용 그리고 IPv4와 IPv6를 동시에 지원하는 응용이 함께 혼재할 수 있으며, 따라서 응용 개발자 혹은 관리자들은 자신의 망 상황과 호스트에서 지원되는 다양한 상황을 고려하여 적절한 응용을 선택 혹은 구현해야 한다.



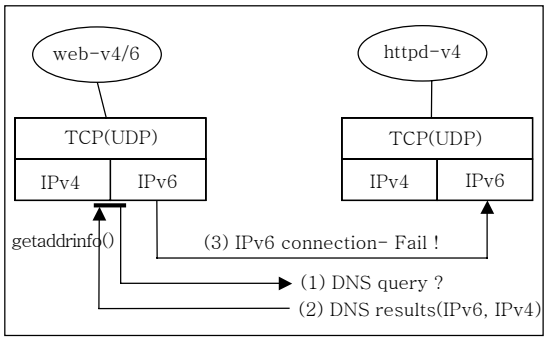
(그림 2) 듀얼스택상의 응용버전의 다양성

2. DNS 이름 해석

호스트 내의 DNS 이름 해석(name resolution) 역할은 목적지 노드의 주소 정보를 얻어오는 것이다. 사실 IPv6 주소에 관한 질의의 경우도, DNS 처리과정에서 사용되는 연결은 최근 IPv6 연결 보다는 IPv4 연결을 통해 수행되고 있다. 이러한 문제 및 해결책은 DNS 전환에 대한 요구사항 문서[10]에 언

급되어 정리되어 있다. 응용 관점에서 DNS 이름 해석과 관련된 문제는 DNS 이름 해석만으로는 상대 응용의 프로토콜 버전을 알 수 없다는 점이다. 다른 말로 표현하면, DNS 질의 결과로 인해 반환되는 주소들은 해당 목적지 노드들의 프로토콜 버전을 나타내는 것이지, 그 위에 동작되고 있는 응용의 버전을 가리키는 것은 아니기 때문이다.

예를 들어 (그림 3)에서와 같이 클라이언트 응용이 연결하고자 하는 서버가 IPv6 주소를 가지고 있어 DNS에 등록이 되어 있는 IPv6 호스트이긴 하나, 그 서버응용이 아직 IPv6로 구현되어 있지 않은 상태인 경우, 클라이언트에 해당하는 IPv4/IPv6 지원 응용이 getaddrinfo() API를 통해 DNS로부터 얻은 서버 호스트 주소들 중 IPv6 주소를 이용하여 서버와 통신하고자 한다면 당연히 연결은 실패하게 될 것이다(디폴트 주소 선택 방법[11]에 따르면 DNS 질의 결과로 인해 얻어지는 주소들 중에 IPv6 주소는 IPv4 주소에 비해 우선적으로 선택되게 된다).



(그림 3) DNS 결과와 응용 버전의 불일치

3. 응용 선택 문제

응용 전환기간 동안 응용 관리자는 IPv4 전용응용, IPv6 전용응용, 그리고 IPv4와 IPv6 모두를 지원하는 응용 등 다양한 응용을 갖을 수 있다. 이때 응용 사용자는 DNS 질의 결과만을 가지고는 통신하고자 하는 상태 응용의 버전을 알 수 없기 때문에 (III. 1, 2 참조) 이러한 다양한 응용 버전에 대해 혼란스러울 수 있다. 만약 시스템 상에 IPv4와 IPv6 모두를 지원하는 응용이 존재한다면, 응용의 버전을 선택하는 데는 별 문제가 없을 것이다. 그러나 모든 응용이 IPv4/IPv6 모두를 지원하도록 포팅될 때까지는, 혹은 여러 가지 이유(코드 수정의 용이성, 초기 IPv6 전용 코드 형태로 구현된 응용의 존재, 혹은 소스 코드의 분실 등)로 인해 같은 응용의 다른 프로토콜 버전을 갖는 여러 응용이 존재하게 되므로, 이러한 환경에서 응용 관리자, 사용자들이 판단할 수 있는 응용 선택의 규칙이 필요하게 된다.

IV. 응용 전환을 위한 가이드라인

1. 1단계: 듀얼 스택에 IPv4 전용응용만이 존재하는 경우

IPv6 프로토콜은 이미 호스트에 적용되어 있으나, 그를 기반으로 한 응용이 아직 존재하지 않을 수 있다(II. 1단계 참조). 이것은 기존의 IPv4를 위한 응용들이 아직 IPv6 스택을 위해 변경되지 않은 경우이거나 혹은, 기존의 IPv4를 위한 응용의 소스 코드가 존재하지 않아 그것이 불가능한 경우를 모두 포함한다. 이러한 상황에서 듀얼스택상에 존재하는 IPv4 응용이 IPv6 호스트와 연결하고자 한다면, 그 호스트에는 응용 변환 기술에 해당하는 BIA[8]가 구현되어 있어야만 한다.

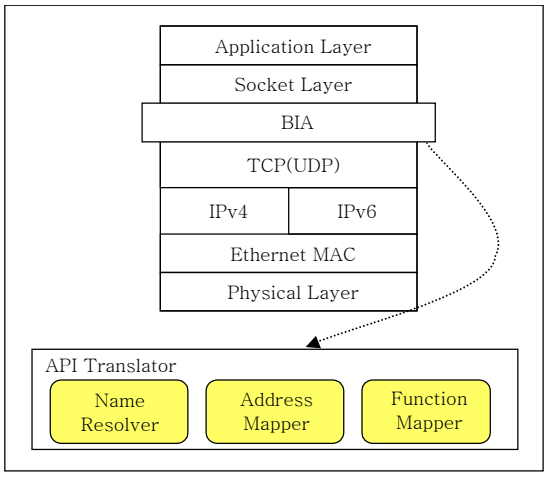
BIA[8]와 BIS[7]의 가장 중요한 차이점은 BIS[7]는 IPv6 프로토콜 스택이 존재하지 않는 호스트들을 대상으로 하고 있는 반면, BIA[8]는 IPv6 스택은 있으나 그 응용이 없는 경우를 위한 기술이라는 것이다. 그러므로, 반드시 응용의 소스코드가 존재할 경우에는 종국에는 IPv6용으로 수정할 것을 권고하며, 소스코드가 없어 그것이 불가능한 경우에 한하여, BIA[8] 기술을 이용하여 IPv4 응용이 IPv6 응용과 통신할 수 있도록 한다. 이렇듯, BIA[8]는 응용의 소스코드를 가지고 있지 않는 초기 단계에 적절한 변환 기술이다((그림 4) 참조).

BIA[8]가 구동되는 클라이언트 응용은 III. 1, 2에서 언급한 문제들을 포함하고 있다. 즉, DNS 결과

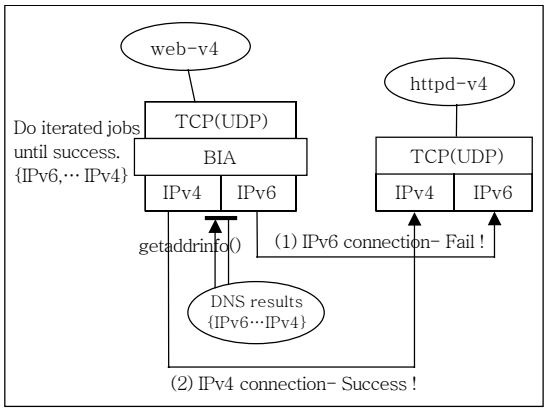
만으로는 통신하고자 하는 상대 응용의 프로토콜 버전을 알 수 없으므로 통신에 실패할 수 있다. 이를 해결하기 위한 BIA[8]는 아래와 같은 2가지 방법을 제안하고 있다.

- 클라이언트 응용 자체적으로 모든 DNS 질의 결과에 따른 주소들에 대해 연결이 성공할 때까지 반복적인 연결 시도를 한다(getaddrinfo() API는 질의를 요청한 클라이언트 응용에 DNS 서버에 저장된 모든 목적지 주소들(IPv6, IPv4 순서로)을 반환할 수 있다).
- [BIA]가 위와 같은 동일한 반복작업을 수행한다((그림 5) 참조).

이러한 2가지 방법은 응용의 구현 방식에 따라 선택적으로 운영할 수 있다.



(그림 4) BIA 구조



(그림 5) BIA에서의 DNS 결과에 따른 반복작업

2. 2단계: 듀얼스택에 IPv4 전용응용과 IPv6 전용응용이 분리되어 함께 존재하는 경우

응용이 IPv6용으로 수정되는 과정에서, 같은 기능을 하지만, IPv4와 IPv6의 다른 프로토콜 스택을 지원하는 두 버전의 응용이 한 호스트에 함께 존재할 수 있다(II. 2단계 참조). 예를 들어 IPv4 스택을 지원하는 IPv4 전용응용인 ping과 IPv6 스택만을 지원하는 IPv6 전용응용인 ping6가 그런 경우에 해당한다. 사실상 이러한 진화 단계는 (그림 1)의 3단계(IPv4와 IPv6를 동시에 지원하는 응용)로 가기 위한 과도기에 해당하나, 초기 IPv6 응용을 도입하기 위해서 IPv4용 응용에서 IPv6용 응용으로 수정하는 단계 그리고 새로운 IPv6 응용을 처음 개발하는 단계가 이에 해당하게 된다.

이 기간동안에는 같은 기능을 하지만 이름이 다른 (IPv6를 위한 응용에는 대부분 이름의 마지막에 6가 따라온다) 두 개의 응용이 함께 존재한다. 이러한 문제를 해결하기 위해서는 아래와 같은 단순한 응용 선택 규칙이 필요하다.

- IPv6 라우팅이 가능한 경우에는 IPv6용 응용을 먼저 선택한다.
- 그렇지 않는 경우에는 IPv4용 응용으로 연결을 시도한다.

이 단계에서는 다양한 변환 기술들이 적용될 것이며, 예를 들어 NAT-PT[3]는 상위의 응용의 버전과 그것과 연결을 맺을 다른 쪽 응용의 버전이 다른 경우에도 응용들이 이를 눈치채지 못하게 자연스럽게 두 응용을 연결시켜 주므로, 이 단계에 가장 적절한 변환 기술이라고 할 수 있다.

IPv6용 전용응용은 IPv4용 응용에서 사용하던 IPv4용 gethostbyname() 같은 소켓 API들 대신 gethostbyname2(), getipnodebyname() 또는 getaddrinfo()와 같은 IPv6를 위한 소켓 API들로 일대일 변경하는 과정을 통해 비교적 쉽게 구현될 수 있

다. 이렇게 구현된 IPv6용 응용은 비교적 수정해야 하는 코드의 양이 적어 간편하고 빨리 IPv6 응용을 도입할 수 있는 장점이 있는 반면, NAT-PT[3]를 제외한 다른 변환 기술을 더 이상 적용할 수 없다는 단점이 있다.

3. 3단계: 듀얼스택에 IPv4와 IPv6를 동시에 지원하는 하나의 응용이 존재하는 경우

이 단계에서는 IPv4와 IPv6를 모두 지원하는 응용이 등장하면서 IPv4 전용응용이 더 이상 존재하지 않게 된다(II. 3단계 참조). 이 단계가 IPv4에서 IPv6로 진화하는 중간 단계 중 가장 바람직한 단계로서 변환 기술들 중에는 DSTM[6]과 같은 기술이 가장 적절하다. 또한 이 단계에서는 IPv4와 IPv6를 모두 지원하는 응용이 그들 중 적절한 주소를 선택하기 위해 디폴트의 주소 선택 규칙[1]이 중요하게 사용된다.

이 단계의 응용은 연결하고자 하는 다른 쪽 응용의 버전이나 지원되고 있는 프로토콜 스택의 상태에 전혀 무관하게 연속적인 시도를 통해 자연스럽게 연결을 성공시킬 수 있다. 이를 위해 DNS 질의의 응답으로 얻은 모든 주소에 대해서 상대 호스트에 반복적으로 연결을 시도하는 구문이 추가적으로 구현되어야 한다(IPv4/IPv6 모두를 지원하는 응용을 구현하는 방법은 VI장에 기술한다).

V. IPv6 전환 메커니즘 고려사항

다양한 IPv6 전환 메커니즘이 적용될 때 응용 개발자와 관리자는 어떤 메커니즘이 자신의 호스트 내에서 최적인지를 알아야 하며, 그러한 메커니즘의 사용으로 인해 운영상의 문제점들은 없는지 확인하여야 한다. 또한 호스트상에서 IPv6 전환 메커니즘을 위한 별도의 설정이 필요한지를 알아야 한다. 기본적으로 모든 IPv6 전환 메커니즘의 존재는 호스트 내에서 투명하여야 한다.

현재 IPv6 전환 메커니즘 중에 응용 고려사항들이 있는 몇 가지를 살펴보면, 먼저 NAT-PT[3]의 경우는 IPv6 응용에서 NAT-PT 변환기 상의 DNS-ALG로부터 변환되어 온 IPv6 변환주소(IPv6 translated address)를 정상적인 IPv6 주소들과 구분할 수 있어야 한다. DSTM[6]의 경우에는 DSTM 메커니즘이 듀얼노드를 기본으로 가정하고 있으므로 그 상위에서 동작되는 응용은 IPv6 전용응용보다는 IPv4/IPv6 모두를 지원하는 응용이 바람직하다.

VI. IPv4/IPv6 모두를 지원하는 응용 구현 방법

일반적으로 IV장 3단계에서 사용되는 IPv4/IPv6 모두를 지원하는 응용을 구현하는 방법은 다음과 같이 정리된다[12]-[15].

1. 규칙 1 – IPv4/IPv6 모두를 지원하는 코드로 수정

가. IPv4 전용, IPv6 전용 구조체 혹은 API는 회피

IPv4 전용, 혹은 IPv6 전용으로만 사용되는 구조체와 API는 사용하지 않는다. <표 1>과 <표 2>는 각각 IPv4 전용, IPv6 전용 구조체 및 IPv4 전용, IPv6 전용 API를 나타낸 것이다.

<표 1> IPv4 전용, IPv6 전용 구조체

IPv4	IPv6
AF_INET/PF_INET	AF_INET6/PF_INET6
AF_INET	AF_INET6
struct sockaddr_in	struct sockaddr_in6
struct in_addr	struct in6_addr

<표 2> IPv4 전용, IPv6 전용 API

IPv4	IPv6
gethostbyname()	getipnodebyname()
gethostbyaddr()	getipnodebyaddr()
inet_addr()	inet_pton()
inet_ntoa()	inet_ntop()

나. IPv4와 IPv6 모두를 지원하는 구조체와 API를 사용

IPv6 전용, IPv6 전용 구조체와 API를 대신하여 IPv4와 IPv6 모두를 지원하는 구조체와 API를 사용한다. 대표적인 구조체 및 API는 각각 아래와 같다.

```
struct sockaddr_storage
          struct sockaddr_storage {
            u_char ss_len;
            u_char ss_family;
            u_char padding[128-2];
          }
getaddrinfo()
getnameinfo()
```

2. 규칙 2 – 연결 가능한 주소를 찾기 위한 반복적인 연결 시도

가. 프로그램 코드 순서를 비교

프로그램 코드 순서에 따라 변경할 API나 구조체들을 확인한다.

나. DNS 결과에 대한 반복적인 연결을 시도

getaddrinfo()로부터 반환된 주소들로부터 연결 가능한 주소를 찾기 위해 for 문, 혹은 while 문을 사용하여 반복적인 연결을 시도한다. 예를 들면, 전형적인 IPv6 전용 클라이언트 프로그램이 (그림 6)과 같다면, 이 코드는 (그림 7)과 IPv4/IPv6 지원 코드로 수정될 수 있다.

```
int connect(char *host, char *service)  {
      struct sockaddr_in dest;
      int sock, ret;
      struct hostent *hp;

      hp = gethostbyname(host);
      if(hp == NULL || hp->h_addrtype != AF_INET
            || hp->h_length != 4)
         /* Handle error */
      dest.sin_family = AF_INET;
      dest.sin_port = htons(port);
      bcopy(hp->h_addr, &dest.sin_addr, 4)
      sock = socket(AF_INET, SOCK_STREAM, 0);
      if(sock == -1)
          /* Handle error */
      ret = connect(sock, (struct sockaddr *)
            &dest, sizeof(dest));
      if(ret == 1)
          /* Handle error */
      return(sock);
   }
```

(그림 6) IPv4 전용 클라이언트 코드 예

```
int connect(char *host, char *service)  {
      struct addrinfo *res, *aip, hints;
      int error, s = -1;
      bzero(&hints, sizeof(hints));
      hints.ai_flags = AI_ADDRCONFIG;
      hints.ai_socktype = SOCK_STREAM;
      error = getaddrinfo(host, service, &hints, &res);
      if(error != 0)
      {/* Handle errors */}
      /* The iterated job through DNS results */
      for(api = res; aip != NULL; aip = aip->ai_next)
      {s = socket(api->ai_family,
            aip->ai_socketype,aip->ai_protocol);
      if(s == -1) continue;
      if(connect(s, aip->ai_addr, aip->ai_addrlen)
                  == -1)
      {(void)close(s);s = -1;
       continue;
      }    }
      freeaddrinfo(res);
      return(s);
   }
```

(그림 7) IPv4/IPv6 지원 클라이언트 코드 예

VII. 결 론

이미 널리 확산되어 많은 사용자를 확보하고 있는 기존의 IPv4 인터넷 망을 IPv6 기반의 차세대인터넷 망으로 전환하기 위해서는 망 전환 관점에서의 다양한 IPv6 전환 메커니즘 외에도 응용 관점에서의 고려사항들에 대한 논의가 필요하다. 본 고에서는 IPv4 응용에서 IPv6 응용으로 전이되는 과정에서 발생하는 문제점들을 분류하고, 이를 위한 해결책으로 가이드라인을 제시하였다. 이를 통해 응용 개발자와 관리자는 혼동 없이 IPv6 전환 단계동안 자연스럽게 IPv6 응용을 개발, 선택할 수 있을 것이다.

참 고 문 헌

[1] R. Gilligan and E. Nordmark, “Transition Mechanisms for IPv6 Hosts and Routers,” RFC 2893, Aug. 2000.

[2] R. Callon and D. Haskin, “Routing Aspects of IPv6 Transition,” RFC 2185, Sep. 1997.

[3] G. Tsirtsis and P. Srisuresh, “Network Address Translation - Protocol Translation(NAT-PT),” RFC 2766, Feb. 2000.

[4] E. Nordmark, “Stateless IP/ICMP Translator(SIIT),” RFC 2765, Feb. 2000.

[5] B. Carpenter and K. Moore, “Connection of IPv6 Domains via IPv4 Clouds,” RFC 3056, Feb. 2001.

[6] Jim Bound *et al.*, Dual Stack Transition Mechanism(DSTM), <draft-ietf-ngtrans-dstm-07.txt>, work in progress, Mar. 2002.

[7] K. Tsuchiya, H. Higuchi, and Y. Atarashi, “Dual Stack Hosts Using the “Bump-In-the-Stack” Technique(BIS),” RFC 2767, Feb. 2000.

[8] Seungyun Lee *et al.*, “Dual Stack Hosts Using “Bump-in-the-API(BIA),” <draft-ietf-ngtrans-bia-05.txt>, work in progress, Mar. 2002.

[9] M. Shin *et al.*, “Application Aspects of IPv6 Transition,”<draft-shin-ngtrans-appli-cation-transition-01.txt>, Feb. 2002.

[10] A. Durand, “NGtrans IPv6 DNS Operational Requirements and Roadmap,” <draft-ietf-ngtrans-dns-ops-req-02.txt>, work in progress, Sep. 2001.

[11] Richard Draves, “Default Address Selection for IPv6,” <draft-ietf-ipngwg-default-addr-select-06.txt>, work in progress, Sep. 2001.

[12] R. Gilligan, S. Thomson, J. Bound and W. Stevens, “Basic Socket Interface Extensions for IPv6,” RFC 2553, Mar. 1999.

[13] Erik Nordmark, “IPv6: Another Y2K for Software,” IPv4 to IPv6 Migration, IPv6 Summit, Sep. 2001.

[14] Sun Microsystem, Inc. “Porting Networking Application to the IPv6 APIs, Solaris Version 8,” 2001.

[15] Jun-ichiro itojun Itoh, “Implementing AF-independent Application,” http://www.kame.net/newsletter/19980604/, 2001.