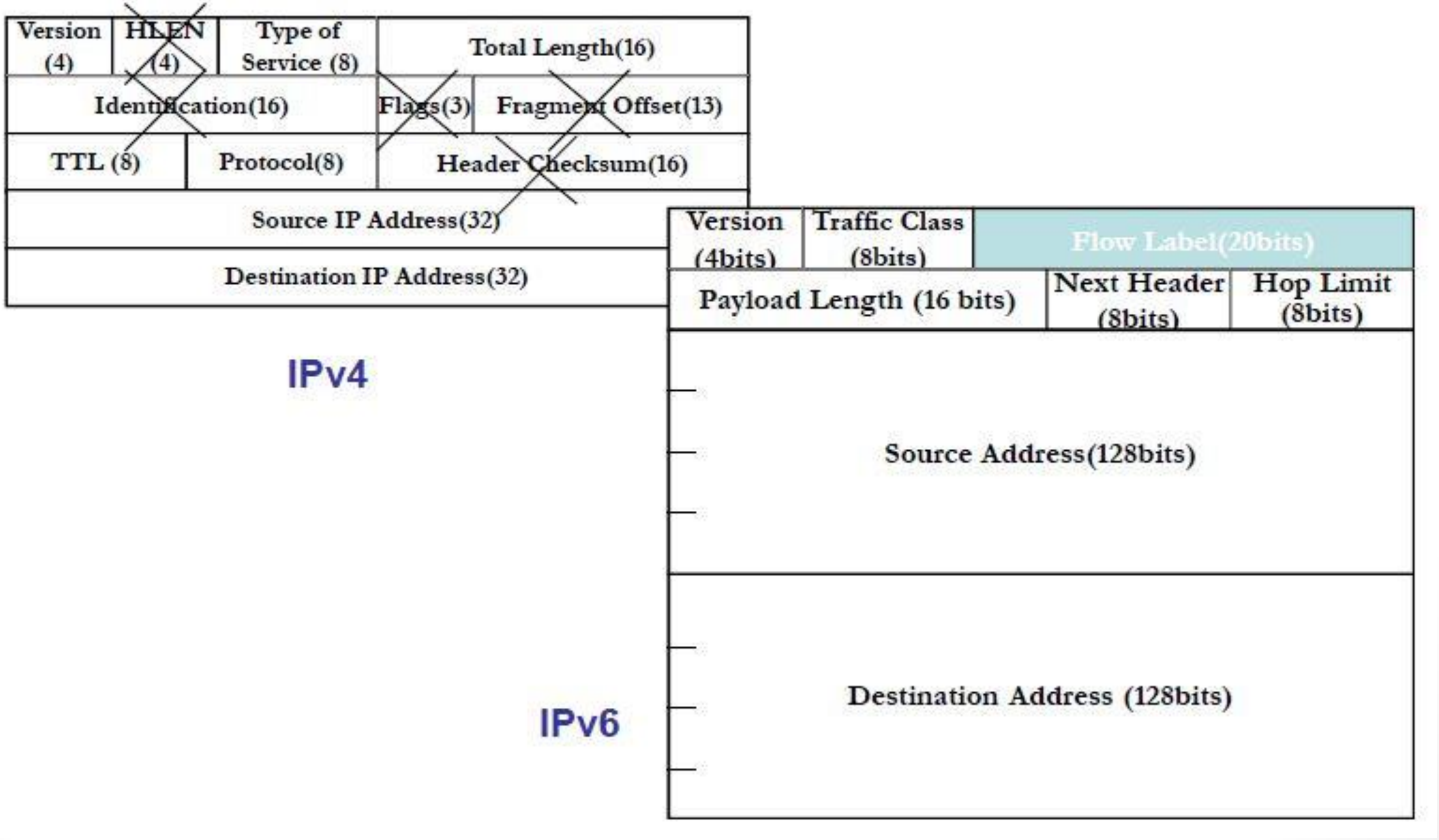
IPv6 적용

IPV6 기본 규격

IPv6 Basic header



IPv4

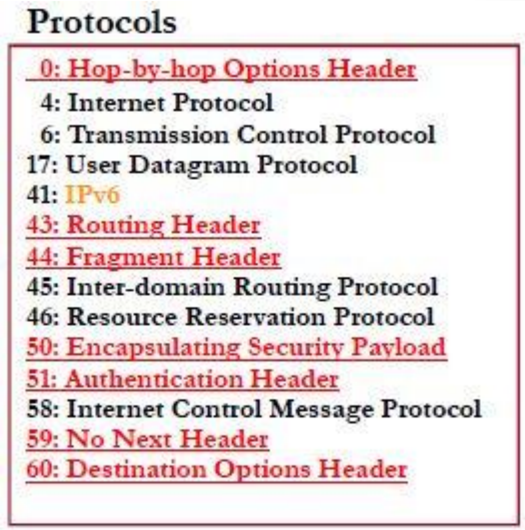
Version (4)	~~HLEN (4)~~	Type of Service (8)	Total Length(16)	
~~Identification(16)~~			~~Flags(3)~~	~~Fragment Offset(13)~~
TTL (8)		Protocol(8)	~~Header Checksum(16)~~	
Source IP Address(32)				
Destination IP Address(32)				

IPv6

Version (4bits)	Traffic Class (8bits)	Flow Label(20bits)	
Payload Length (16 bits)		Next Header (8bits)	Hop Limit (8bits)
Source Address(128bits)			
Destination Address (128bits)			

IPv6 - Extension Headers (1)

- Hop-by-Hop Options (0)
  - RSVP, PIM/MLD, etc.
- Routing (43)
  - Source Routing, MIPv6
- Fragment (44)
- Encapsulating Security Payload (50)
  - IPsec
- Authentication Header (51)
  - IPsec
- No Next Header (59)
- Destination Options (60)
  - MIPv6

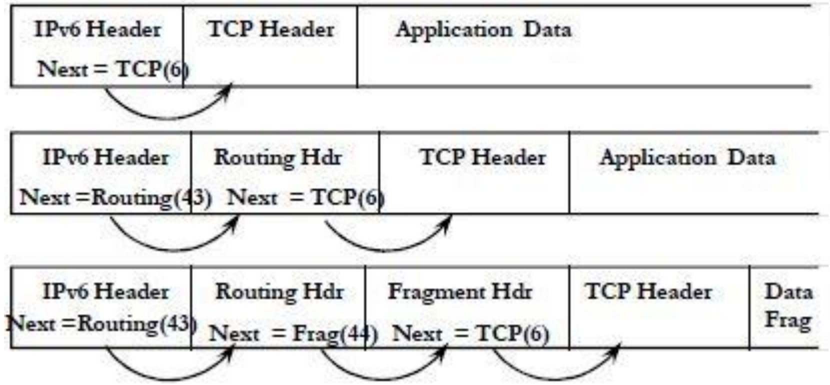


Protocols

0: Hop-by-hop Options Header
4: Internet Protocol
6: Transmission Control Protocol
17: User Datagram Protocol
41: IPv6
43: Routing Header
44: Fragment Header
45: Inter-domain Routing Protocol
46: Resource Reservation Protocol
50: Encapsulating Security Payload
51: Authentication Header
58: Internet Control Message Protocol
59: No Next Header
60: Destination Options Header

IPv6 - Extension Headers (2)

- Extension Header Order
  - IPv6 header
  - Hop-by-Hop Options header
  - Destination Options header*
  - Routing header
  - Fragment header
  - Authentication header
  - Encapsulating Security Payload header
  - Destination Options header*
  - upper-layer header

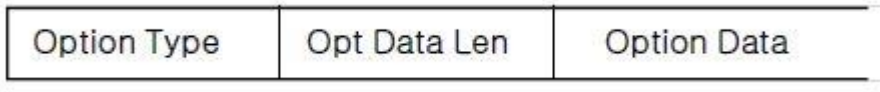


IPv6 Header Next = TCP(6)	TCP Header	Application Data		
IPv6 Header Next =Routing(43)	Routing Hdr Next = TCP(6)	TCP Header	Application Data	
IPv6 Header Next =Routing(43)	Routing Hdr Next = Frag(44)	Fragment Hdr Next = TCP(6)	TCP Header	Data Frag

IPv6 - Extension Headers (3)

- Two Options
  - Hop-by-Hop Options header
  - Destination Options header

- type-length-value (TLV) encoded Format



Option Type	Opt Data Len	Option Data

  - if the processing IPv6 node does not recognize the Option Type:
    - 00 - skip over this option and continue processing.
    - 01/10/11 - discard the packet.

IPV6 주소체계

IPv6 - Three Types of Addresses

- Unicast
  - An identifier for a single interface.
  - A packet sent to a unicast address is delivered to the interface identified by that address
- Anycast
  - An identifier for a set of interfaces.
  - A packet sent to an anycast address is delivered to one of the interfaces identified by that address (the "nearest" one)
- Multicast
  - An identifier for a set of interfaces
  - A packet sent to a multicast address is delivered to all interfaces identified by that address.

IPv6 – Addressing Model

- 128-bit addressing scheme
  - x:x:x:x:x:x:x:x
    - 'x's are the hexadecimal values of the eight 16-bit pieces of the address.
  - "::" indicates multiple groups of 16 bits of zeros.
    - 3FFE:2E01:0:0:0:31:0:21 -> 3FFE:2E01::31:0:21
  - ipv6-address/prefix-length
    - 3FFE:0000:0000:CD30:0000:0000:0000:0000/64
    - 3FFE::CD30:0:0:0:0/64
    - 3FFE:0:0:CD30::/64
    - 3FFE:0:0:CD3/64 (x)
    - 3FFE::CD30/64 (x)
    - 3FFE::CD3/64 (x)

IPv6 – Address Type Representation

Address type	Binary prefix	IPv6 notation
Unspecified	00...0 (128 bits)	::/128
Loopback	00...1 (128 bits)	::1/128
Multicast	11111111	FF00::/8
Link-local unicast	1111111010	FE80::/10
Site-local unicast	1111111011	FEC0::/10
Global unicast	(everything else)	

IPv6 – Unicast

- General Format

n bits	128 - n bits
subnet prefix	Interface ID

- Unspecified address
  - 0:0:0:0:0:0:0:0 = ::0
- Loopback address
  - 0:0:0:0:0:0:0:1 = ::1
- IPv6 Addresses with Embedded IPv4 Addresses
  - IPv4-compatible IPv6 address
  - IPv4-mapped IPv6 address
- Global Unicast Addresses
- Local-Use IPv6 Unicast Addresses
  - Link local address

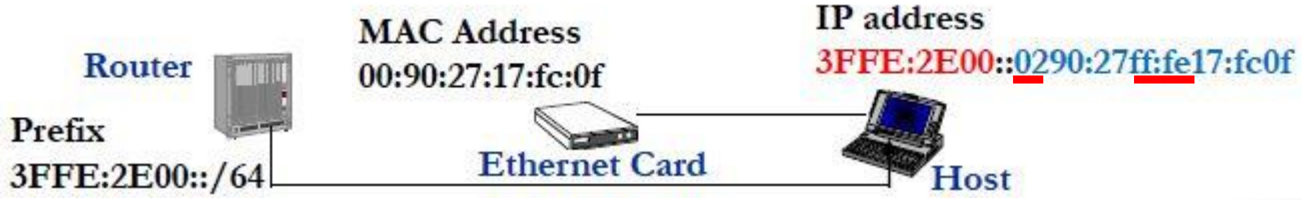
IPv6 – Address Auto-configuration

- 64-bit Interface Identifiers (eg., from 48-bit MAC)

00 15	16 31	32 47
cccccc0gcccccccc	ccccccccmmmmmmmm	mmmmmmmmmmmmmmmm

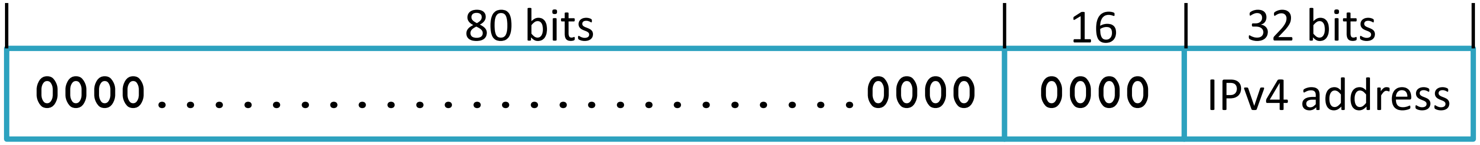
00 15	16 31	32 47	48 63
cccccc**1**gcccccccc	cccccccc**11111111**	**11111110**mmmmmmmm	mmmmmmmmmmmmmmmm
	ff	fe	

- 128-bit Address Auto-configuration
  - subnet prefix + Interface ID



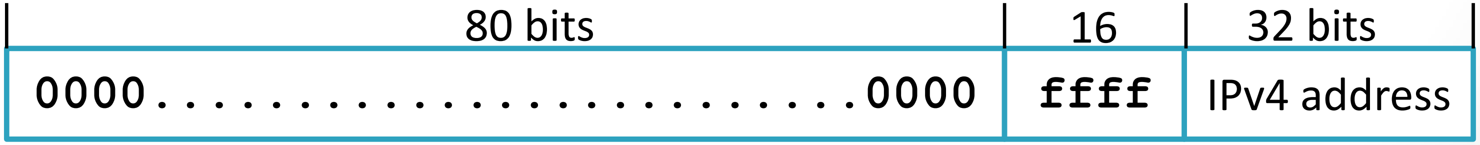
IPv6 Addresses with Embedded IPv4 Addresses

- IPv4-compatible IPv6 address
  - For hosts and routers to dynamically tunnel IPv6 packets over IPv4 routing infrastructure
  - ::203.232.252.110



80 bits	16	32 bits
0000.........................0000	0000	IPv4 address

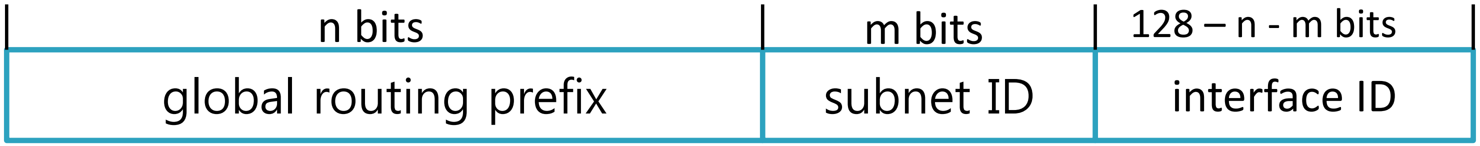
- IPv4-mapped IPv6 address
  - To represent the addresses of IPv4-only nodes as IPv6 addresses
  - ::FFFF:203.232.252.110



80 bits	16	32 bits
0000.........................0000	ffff	IPv4 address

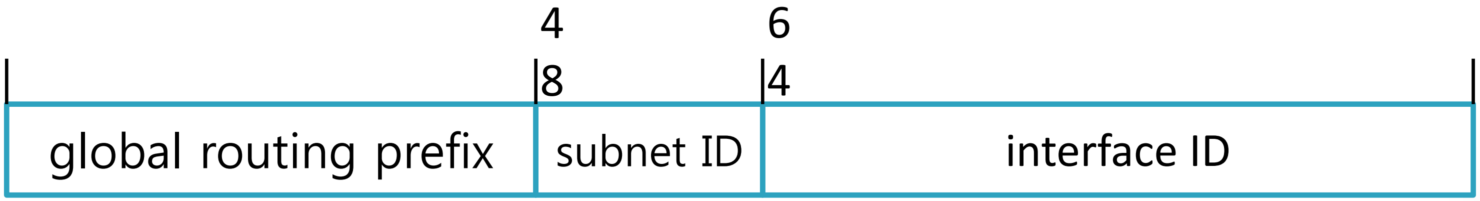
IPv6 – Global Unicast Address

- General format



n bits	m bits	128 – n - m bits
global routing prefix	subnet ID	interface ID

- Current policy



	48	64
global routing prefix	subnet ID	interface ID

Local-Use IPv6 Unicast Addresses

- Link-Local addresses, fe80::/10

10 bits	54 bits	64 bits
1111111010	0	interface ID

- Site-Local addresses, fec0::/10

10 bits	38 bits	16 bits	64 bits
1111111011	0	subnet ID	interface ID

IPv6 - A lot of Address

- Multiple unicast addresses to be assigned to interfaces
  - Different Reachability Scope
    - Link-local / site-local / global
  - Privacy Considerations
    - Public / temporary
  - Mobility
    - Home address / CoA
  - Multi-homing situation
  - Dual stack situation
    - IPv4 addresses

IPv6 Address - Default Policy Table

- Implementations SHOULD be configurable, via mechanisms at least as powerful as these policy tables.
- If not configured, then they SHOULD operate according to the default policy table:

Prefix	Precedence	Label
::1/128	50	0
::/0	40	1
2002::/16	30	2
::/96	20	3
::ffff:0:0/96	10	4

Source Address Selection

- Selecting IPv6 source for IPv6 destination:
  - Prefer same address (for loopback).
  - Prefer appropriate scope.
  - Avoid deprecated addresses.
  - Prefer home addresses over care-of addresses.
  - Prefer source assigned to originating interface.
  - Prefer matching label from policy table.
  - Prefer public addresses.
  - Use longest-matching-prefix.

Destination Address Ordering

- Select best source for each destination, IPv6 and IPv4:
  - Avoid unusable destinations.
  - Prefer matching scope.
  - Avoid deprecated source addresses.
  - Prefer home source addresses.
  - Prefer matching label from policy table.
  - Prefer destinations with higher precedence.
  - Prefer smaller scope destinations.
  - Use longest-matching-prefix.
  - Otherwise, leave order from DNS unchanged

IPV6 소켓 프로그래밍

주요 구조체 검토 (1)

- IPv4 주소

```
/* Internet address. */
struct in_addr {
        __be32  s_addr;
};

struct sockaddr_in {
  __kernel_sa_family_t  sin_family;     /* Address family                */
  __be16                sin_port;       /* Port number                   */
  struct in_addr        sin_addr;       /* Internet address              */

  /* Pad to size of `struct sockaddr'. */
  unsigned char         __pad[__SOCK_SIZE__ - sizeof(short int) -
                        sizeof(unsigned short int) - sizeof(struct in_addr)];
};
```

주요 구조체 검토 (2)

- IPv6 주소

```
/*
 *      IPv6 address structure
 */

struct in6_addr {
        union {
                __u8            u6_addr8[16];
                __be16          u6_addr16[8];
                __be32          u6_addr32[4];
        } in6_u;
#define s6_addr                 in6_u.u6_addr8
#define s6_addr16               in6_u.u6_addr16
#define s6_addr32               in6_u.u6_addr32
};

struct sockaddr_in6 {
        unsigned short int      sin6_family;    /* AF_INET6 */
        __be16                  sin6_port;      /* Transport layer port # */
        __be32                  sin6_flowinfo;  /* IPv6 flow information */
        struct in6_addr         sin6_addr;      /* IPv6 address */
        __u32                   sin6_scope_id;  /* scope id (new in RFC2553) */
};
```

주요 구조체 검토 (3)

- 범용 주소 형식

```
struct sockaddr
  {
    __SOCKADDR_COMMON (sa_);    /* Common data: address family and length.  */
    char sa_data[14];           /* Address data.  */
  };
```

```
/* Structure large enough to hold any socket address (with the historical
   exception of AF_UNIX).  We reserve 128 bytes.  */
#define __ss_aligntype  unsigned long int
#define _SS_SIZE        128
#define _SS_PADSIZE     (_SS_SIZE - (2 * sizeof (__ss_aligntype)))

struct sockaddr_storage
  {
    __SOCKADDR_COMMON (ss_);    /* Address family, etc.  */
    __ss_aligntype __ss_align;  /* Force desired alignment.  */
    char __ss_padding[_SS_PADSIZE];
  };
```

이진/문자열 주소변환 함수 (1)

- IPv4

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int inet_aton(const char *cp, struct in_addr *inp);

unsigned long int inet_addr(const char *cp);

unsigned long int inet_network(const char *cp);

char *inet_ntoa(struct in_addr in);

struct in_addr inet_makeaddr(int net, int host);

unsigned long int inet_lnaof(struct in_addr in);

unsigned long int inet_netof(struct in_addr in);
```

이진/문자열 주소변환 함수 (2)

- IPv4/IPv6

```
#include <arpa/inet.h>

int inet_pton(int af, const char *src, void *dst);

const char *inet_ntop(int af, const void *src, char *dst, socklen_t size);
```

```
char clntName[INET6_ADDRSTRLEN]; // Array to contain client address string

if (inet_ntop(AF_INET6, &clntAddr.sin6_addr.s6_addr, clntName,
                    sizeof(clntName)) != NULL)
          printf("Handling client %s\n", clntName);
```

도메인네임 서비스 이용 (1)

- 관련 구조체

```
/* Structure to contain information about address of a service provider.  */
struct addrinfo
{
  int ai_flags;                 /* Input flags.  */
  int ai_family;                /* Protocol family for socket.  */
  int ai_socktype;              /* Socket type.  */
  int ai_protocol;              /* Protocol for socket.  */
  socklen_t ai_addrlen;         /* Length of socket address.  */
  struct sockaddr *ai_addr;     /* Socket address for socket.  */
  char *ai_canonname;           /* Canonical name for service location.  */
  struct addrinfo *ai_next;     /* Pointer to next in list.  */
};
```

도메인네임 서비스 이용 (2)

- 관련 함수

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char *node, const char *service,
                    const struct addrinfo *hints,
                    struct addrinfo **res);

void freeaddrinfo(struct addrinfo *res);

const char *gai_strerror(int errcode);
```

도메인네임 서비스 이용 (3)

- getaddrinfo () 이용 예제

```
// Tell the system what kind(s) of address info we want
 struct addrinfo addrCriteria;                   // Criteria for address match
 memset(&addrCriteria, 0, sizeof(addrCriteria)); // Zero out structure
 addrCriteria.ai_family = AF_UNSPEC;             // Any address family
 addrCriteria.ai_socktype = SOCK_STREAM;         // Only stream sockets
 addrCriteria.ai_protocol = IPPROTO_TCP;         // Only TCP protocol

 // Get address(es) associated with the specified name/service
 struct addrinfo *addrList; // Holder for list of addresses returned
 // Modify servAddr contents to reference linked list of addresses
 int rtnVal = getaddrinfo(addrString, portString, &addrCriteria, &addrList);
 if (rtnVal != 0)
   fprintf(stderr, "getaddrinfo() failed : %s", gai_strerror(rtnVal));

 // Display returned addresses
 for (struct addrinfo *addr = addrList; addr != NULL; addr = addr->ai_next) {
   PrintSocketAddress(addr->ai_addr, stdout);
 }

 freeaddrinfo(addrList); // Free addrinfo allocated in getaddrinfo()
```

V6용 TCP 클라이언트 예제 (1)

- SetupTCPClient6Socket()
  - 호스트 이름으로 DNS 질의 후 결과로 반환된 IPv6 주소로 연결 (connect)

```
int SetupTCPClient6Socket(const char *host, const char *service)
{

  // Tell the system what kind(s) of address info we want
  struct addrinfo addrCriteria;                   // Criteria for address match
  memset(&addrCriteria, 0, sizeof(addrCriteria)); // Zero out structure
  addrCriteria.ai_family = AF_INET6 ;             // IPv6 address family
  addrCriteria.ai_socktype = SOCK_STREAM;         // Only streaming sockets
  addrCriteria.ai_protocol = IPPROTO_TCP;         // Only TCP protocol

  // Get address(es)
  struct addrinfo *servAddr; // Holder for returned list of server addrs
  int rtnVal = getaddrinfo(host, service, &addrCriteria, &servAddr);
  if (rtnVal != 0)
    fprintf(stderr, "getaddrinfo() failed : %s", gai_strerror(rtnVal));
```

V6용 TCP 클라이언트 예제 (2)

- SetupTCPClient6Socket() (계속)

```
  int sock = -1;
  for (struct addrinfo *addr = servAddr; addr != NULL; addr = addr->ai_next) {
    // Create a reliable, stream socket using TCP
    sock = socket(addr->ai_family, addr->ai_socktype, addr->ai_protocol);
    if (sock < 0)
      continue;  // Socket creation failed; try next address

    // Establish the connection to the echo server
    if (connect(sock, addr->ai_addr, addr->ai_addrlen) == 0)
      break;     // Socket connection succeeded; break and return socket

    close(sock); // Socket connection failed; try next address
    sock = -1;
  }

  freeaddrinfo(servAddr); // Free addrinfo allocated in getaddrinfo()

  return sock;
}
```

범용 TCP 클라이언트 예제 (1)

- SetupTCPClientSocket()
  - 호스트 이름으로 DNS 질의 후 결과로 반환된 IPv4 또는 IPv6 주소로 연결(connect)

```
int SetupTCPClientSocket(const char *host, const char *service)
{

  // Tell the system what kind(s) of address info we want
  struct addrinfo addrCriteria;                   // Criteria for address match
  memset(&addrCriteria, 0, sizeof(addrCriteria)); // Zero out structure
  addrCriteria.ai_family = AF_UNSPEC;             // v4 or v6 is OK
  addrCriteria.ai_socktype = SOCK_STREAM;         // Only streaming sockets
  addrCriteria.ai_protocol = IPPROTO_TCP;         // Only TCP protocol

  // Get address(es)
  struct addrinfo *servAddr; // Holder for returned list of server addrs
  int rtnVal = getaddrinfo(host, service, &addrCriteria, &servAddr);
  if (rtnVal != 0)
    fprintf(stderr, "getaddrinfo() failed : %s", gai_strerror(rtnVal));
```

범용 TCP 클라이언트 예제 (2)

- SetupTCPClientSocket() (계속)

```
  int sock = -1;
  for (struct addrinfo *addr = servAddr; addr != NULL; addr = addr->ai_next) {
    // Create a reliable, stream socket using TCP
    sock = socket(addr->ai_family, addr->ai_socktype, addr->ai_protocol);
    if (sock < 0)
      continue;  // Socket creation failed; try next address

    // Establish the connection to the echo server
    if (connect(sock, addr->ai_addr, addr->ai_addrlen) == 0)
      break;     // Socket connection succeeded; break and return socket

    close(sock); // Socket connection failed; try next address
    sock = -1;
  }

  freeaddrinfo(servAddr); // Free addrinfo allocated in getaddrinfo()

  return sock;
}
```

V6용 TCP 서버 예제 (1)

- SetupTCPServer6Socket()
  - DNS 질의를 통해 얻어진 IPv6 또는 IPv4 주소로 바인딩

```
int SetupTCPServer6Socket(const char *service)
{
  // Construct the server address structure
  struct addrinfo addrCriteria;                   // Criteria for address match
  memset(&addrCriteria, 0, sizeof(addrCriteria)); // Zero out structure
  addrCriteria.ai_family = AF_INET6;              // IPv6 address family
  addrCriteria.ai_flags = AI_PASSIVE;             // Accept on any address/port
  addrCriteria.ai_socktype = SOCK_STREAM;         // Only stream sockets
  addrCriteria.ai_protocol = IPPROTO_TCP;         // Only TCP protocol

  struct addrinfo *servAddr; // List of server addresses
  int rtnVal = getaddrinfo(NULL, service, &addrCriteria, &servAddr);
  if (rtnVal != 0)
    fprintf(stderr, "getaddrinfo() failed : %s", gai_strerror(rtnVal));

  int servSock = -1;
  for (struct addrinfo *addr = servAddr; addr != NULL; addr = addr->ai_next) {
    // Create a TCP socket
    servSock = socket(addr->ai_family, addr->ai_socktype,
        addr->ai_protocol);
    if (servSock < 0)
      continue;       // Socket creation failed; try next address
```

V6용 TCP 서버 예제 (2)

- SetupTCPServer6Socket() (계속)

```
 // Bind to the local address and set socket to listen

    if ((bind(servSock, addr->ai_addr, addr->ai_addrlen) == 0) &&
        (listen(servSock, MAXPENDING) == 0)) {

      // Print local address of socket
      struct sockaddr_in6 localAddr;
      socklen_t addrSize = sizeof(localAddr);
      if (getsockname(servSock, (struct sockaddr *) &localAddr, &addrSize) < 0)
         fprintf(stderr, "getsockname() failed");

      break;       // Bind and listen successful
    }

    close(servSock);  // Close and try again
    servSock = -1;
  }

  // Free address list allocated by getaddrinfo()
  freeaddrinfo(servAddr);

  return servSock;
}
```

V6용 TCP 서버 예제 (3)

- AcceptTCPConnection6()
  - 클라이언트와 연결 설정

```
int AcceptTCPConnection6(int servSock)
{
  struct sockaddr_in6 clntAddr; // Client address
  // Set length of client address structure (in-out parameter)
  socklen_t clntAddrLen = sizeof(clntAddr);

  // Wait for a client to connect
  int clntSock = accept(servSock, (struct sockaddr *) &clntAddr, &clntAddrLen);
  if (clntSock < 0)
    fprintf(stderr, "accept() failed");

  // clntSock is connected to a client!

  return clntSock;
}
```

범용 TCP 서버 예제 (1)

- SetupTCPServerSocket()
  - DNS 질의를 통해 얻어진 IPv6 또는 IPv4 주소로 바인딩

```
int SetupTCPServerSocket(const char *service)
{
  // Construct the server address structure
  struct addrinfo addrCriteria;                   // Criteria for address match
  memset(&addrCriteria, 0, sizeof(addrCriteria)); // Zero out structure
  addrCriteria.ai_family = AF_UNSPEC;             // Any address family
  addrCriteria.ai_flags = AI_PASSIVE;             // Accept on any address/port
  addrCriteria.ai_socktype = SOCK_STREAM;         // Only stream sockets
  addrCriteria.ai_protocol = IPPROTO_TCP;         // Only TCP protocol

  struct addrinfo *servAddr; // List of server addresses
  int rtnVal = getaddrinfo(NULL, service, &addrCriteria, &servAddr);
  if (rtnVal != 0)
    fprintf(stderr, "getaddrinfo() failed : %s", gai_strerror(rtnVal));

  int servSock = -1;
  for (struct addrinfo *addr = servAddr; addr != NULL; addr = addr->ai_next) {
    // Create a TCP socket
    servSock = socket(addr->ai_family, addr->ai_socktype,
        addr->ai_protocol);
    if (servSock < 0)
      continue;       // Socket creation failed; try next address
```

범용 TCP 서버 예제 (2)

- SetupTCPServerSocket() (계속)

```
 // Bind to the local address and set socket to listen

    if ((bind(servSock, addr->ai_addr, addr->ai_addrlen) == 0) &&
        (listen(servSock, MAXPENDING) == 0)) {

      // Print local address of socket
      struct sockaddr_storage localAddr;
      socklen_t addrSize = sizeof(localAddr);
      if (getsockname(servSock, (struct sockaddr *) &localAddr, &addrSize) < 0)
         fprintf(stderr, "getsockname() failed");

      break;       // Bind and listen successful
    }

    close(servSock);  // Close and try again
    servSock = -1;
  }

  // Free address list allocated by getaddrinfo()
  freeaddrinfo(servAddr);

  return servSock;
}
```

범용 TCP 서버 예제 (3)

- AcceptTCPConnection()
  - 클라이언트와 연결 설정

```
int AcceptTCPConnection(int servSock)
{
  struct sockaddr_storage clntAddr; // Client address
  // Set length of client address structure (in-out parameter)
  socklen_t clntAddrLen = sizeof(clntAddr);

  // Wait for a client to connect
  int clntSock = accept(servSock, (struct sockaddr *) &clntAddr, &clntAddrLen);
  if (clntSock < 0)
    fprintf(stderr, "accept() failed");

  // clntSock is connected to a client!

  return clntSock;
}
```

Scope_id 설정 (1)

- Link-local 주소를 이용하는 경우 sockaddr_in6 구조체의 scope_id 멤버 설정 필요
  - 하나의 호스트에 여러 개의 인터페이스가 있을 수 있고, 따라서 어떤 인터페이스를 이용할지 scope_id로 명시
    - 상대방 주소(인터페이스)와 함께 있는 인터페이스 명시
  - if_nameindex 활용
    - 모든 네트워크 인터페이스와 인덱스 반환

```
#include <net/if.h>

struct if_nameindex *if_nameindex(void);
```

    - 사용 후에는 반드시 if_freenameindex()를 이용하여 메모리를 반환시켜야 함
    - 관련 함수들
      - if_indextoname()
      - if_nametoindex()

Scope_id 설정 (2)

- if_nameindex 활용 예제
    - if_name_to_scope_id() – 자체 제작
        - 인터페이스 이름을 받아들여, 시스템 내에 있는 인터페이스 이름과 비교하여 이름에 해당하는 인덱스 반환

```
#include <net/if.h>

int if_name_to_scope_id(const char *if_name)
{
        int scope_id = -1;

        struct if_nameindex *if_idx, *ifp;
        if_idx = if_nameindex();

        for (ifp = if_idx; ifp->if_name != NULL; ifp++) {
                if (!strcmp(if_name, ifp->if_name)) {
                        scope_id = ifp->if_index;
                }
        }

        if_freenameindex(if_idx);

        return (scope_id);
}
```

Scope_id 설정 (3)

- if_name_to_scope_id( ) 사용 예제
  - 인터페이스 이름이 존재하는 경우 if_name_to_scope_id( ) 호출

```
...

sock = socket(addr->ai_family, addr->ai_socktype, addr->ai_protocol);

if (sock < 0)
     continue;  // Socket creation failed; try next address

struct sockaddr_in6 *sin = (struct sockaddr_in6 *)addr->ai_addr;

if (if_name) {  // not NULL
     int scope_id;

     if ((scope_id = if_name_to_scope_id(if_name)) >= 0) {
          sin->sin6_scope_id = scope_id;
     }
     else {
          fprintf(stderr, "Cannot find the scope_id of %s\n", if_name);
          close (sock);
          sock = -1;
     }
}
...
```

시스템 인터페이스 확인

- ifconfig -a

```
eno1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 203.232.252.110  netmask 255.255.255.0  broadcast 203.232.252.255
        inet6 fe80::ae16:2dff:fe89:32a4  prefixlen 64  scopeid 0x20<link>
        ether ac:16:2d:89:32:a4  txqueuelen 1000  (Ethernet)
        RX packets 31363186  bytes 22288924843 (20.7 GiB)
        RX errors 0  dropped 61760  overruns 0  frame 0
        TX packets 18906148  bytes 11237679682 (10.4 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 32
...
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 0  (Local Loopback)
        RX packets 1452812  bytes 22080139786 (20.5 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1452812  bytes 22080139786 (20.5 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

virbr0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 192.168.122.1  netmask 255.255.255.0  broadcast 192.168.122.255
        ether 52:54:00:3b:95:15  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```