IPv6 Programming Introduction

Linjiang Li

6/19 2014

IPv6 and why is IPv6?

- **Internet Protocol version 6** (**IPv6**) , which is intended to replace IPv4.
- With more and more new devices being connected to the Internet, the IPv4 address will be exhaustion.
- IPv6 uses a 128-bit address, allowing 2^{128}, or approximately 3.4×10^{38}addresses, or more than 7.9×10^{28} times as many as IPv4, which uses 32-bit addresses. IPv4 provides approximately 4.3 billion addresses.
- The number of IPv6 addresses would be about 40,000 addresses for every atom on the surface of the earth.

IPv6 Addressing

IPv6 Addressing

IPv4 32-bits

IPv6 128-bits

2^{32} = 4,294,967,296

2^{128} = 340,282,366,920,938,463,463,374,607,431,768,211,456

$2^{128} = 2^{32} * 2^{96}$

2^{96} = 79,228,162,514,264,337,593,543,950,336 times the number of possible IPv4 Addresses (79 trillion trillion)

IPv6 Addressing

World's population is approximately 6.5 billion

$$\frac{2^{128}}{6.5 \text{ Billion}} = 52 \text{ Trillion Trillion IPv6 addresses per person}$$

Typical brain has ~100 billion brain cells (your count may vary)

$$\frac{52 \text{ Trillion Trillion}}{100 \text{ Billion}} = 523 \text{ Quadrillion (523 thousand trillion) IPv6 addresses for every human brain cell on the planet!}$$

IPv6 Address Format

Representation

- 16-bit hexadecimal numbers
- Numbers are separated by (:)
- Hex numbers are not case sensitive
- Abbreviations are possible

  Leading zeros in contiguous block could be represented by (::)

  Example:

  2001:0db8:0000:130F:0000:0000:087C:140B

  2001:0db8:0:130F::87C:140B

  Double colon only appears once in the address

IPv6 Address Prefix

Prefix Representation

- Representation of prefix is just like CIDR
- In this representation you attach the prefix length
- Like v4 address:

  198.10.0.0/16

- V6 address is represented the same way:

  2001:db8:12::/48

- Only leading zeros are omitted. Trailing zeros are not omitted

  2001:0db8:0012::/48 = 2001:db8:12::/48

  2001:db8:**1200**::/48 ≠ 2001:db8:12::/48

IPv6 Address Model

- Addresses are assigned to interfaces
  - Change from IPv4 mode:
- Interface “expected” to have multiple addresses
- Addresses have scope
  - Link Local
  - Unique Local
  - Global
- Addresses have lifetime
  - Valid and preferred lifetime

IPv6 Address Types

- Unicast

  Address of a single interface. One-to-one delivery to single interface

- Multicast

  Address of a set of interfaces. One-to-many delivery to all interfaces in the set

- Anycast

  Address of a set of interfaces. One-to-one-of-many delivery to a single interface in the set that is closest

- No more broadcast addresses

Some Special Addresses

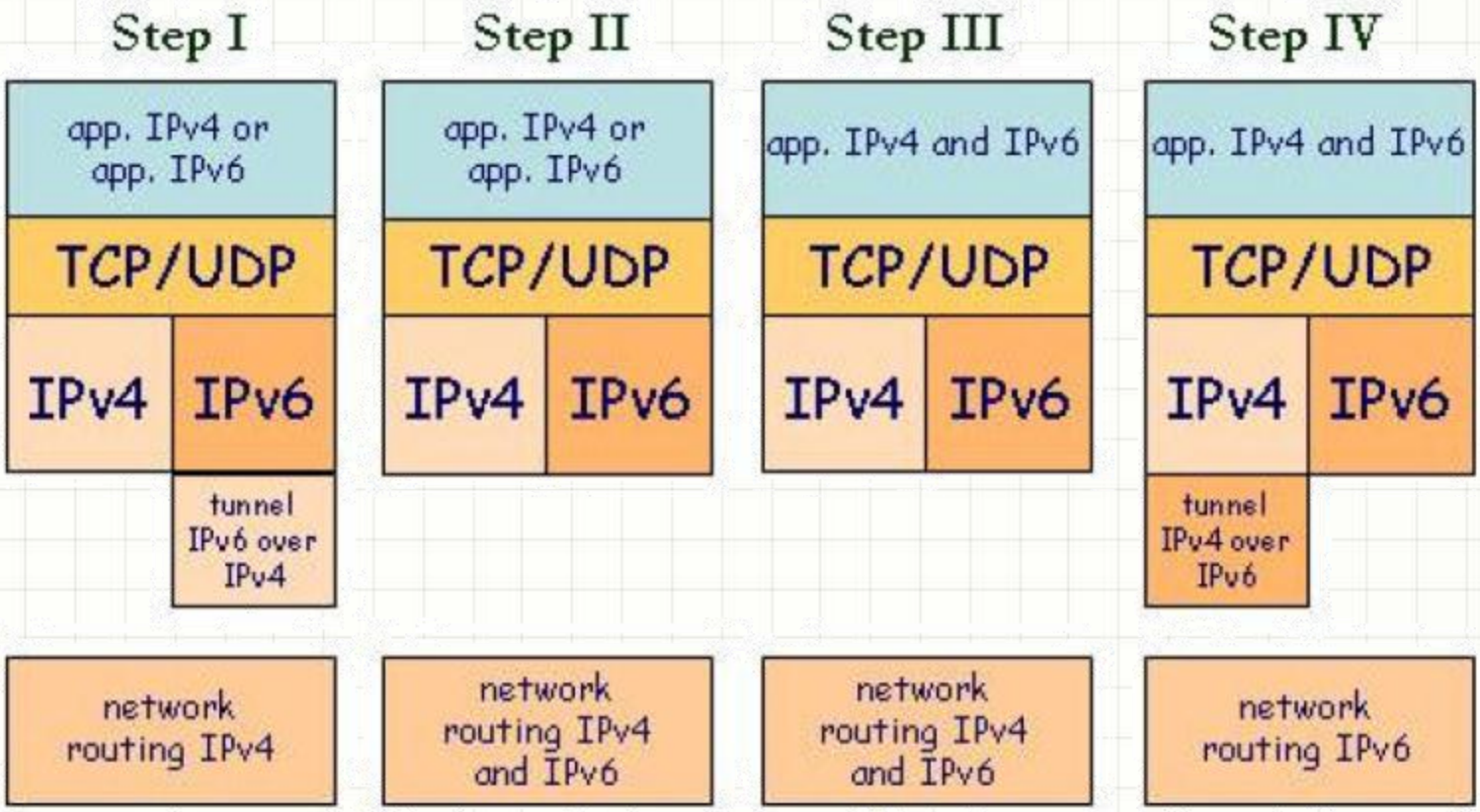
Symbolic Name (IPv4 / IPv6)	IPv4 Address	IPv6 Address
INADDR_ANY / IN6ADDR_ANY_INIT	0.0.0.0	::
INADDR_LOOPBACK / IN6ADDR_LOOPBACK_INIT	127.0.0.1	::1
INADDR_BROADCAST	255.255.255.255	Not exist

- **More IPv4/IPv6 information, refer attached.**

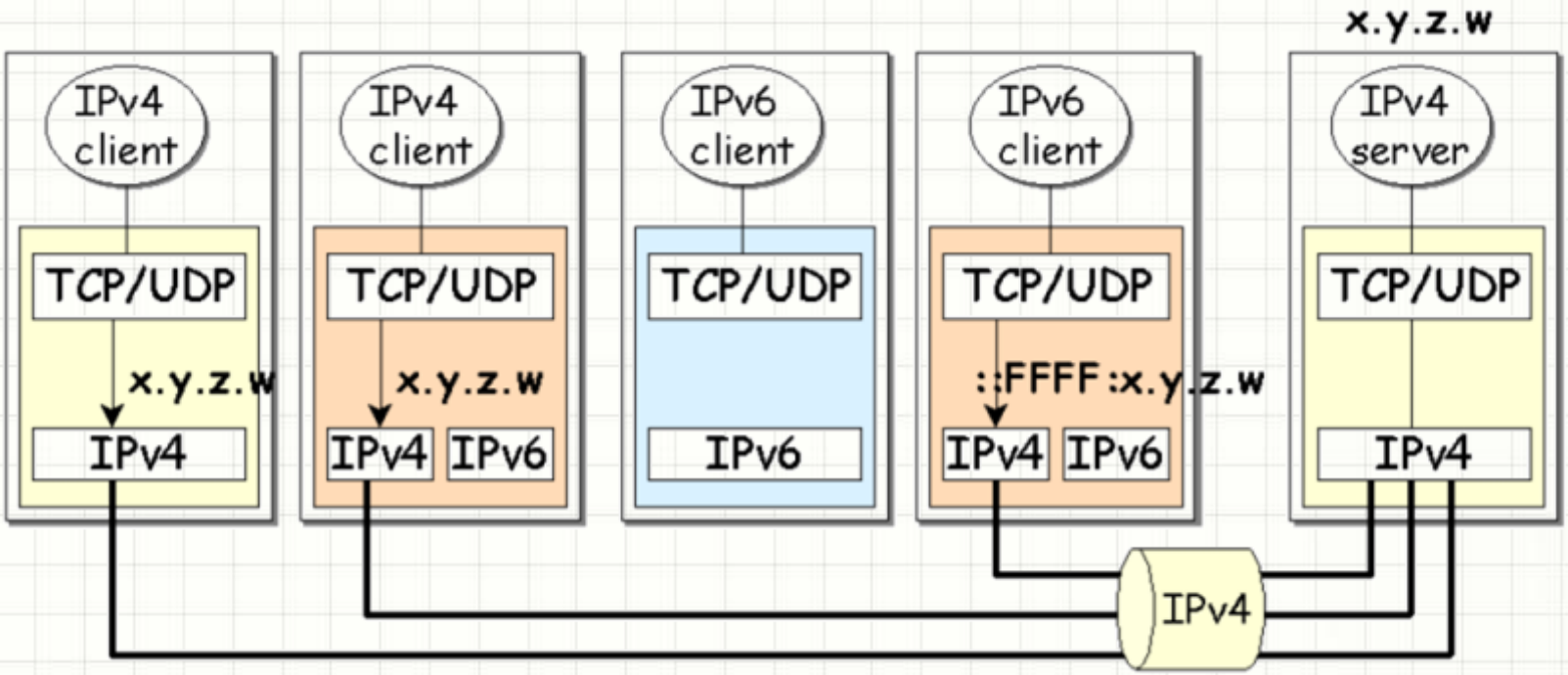
ipv6_reference_card.pdf

Transition & Interoperability

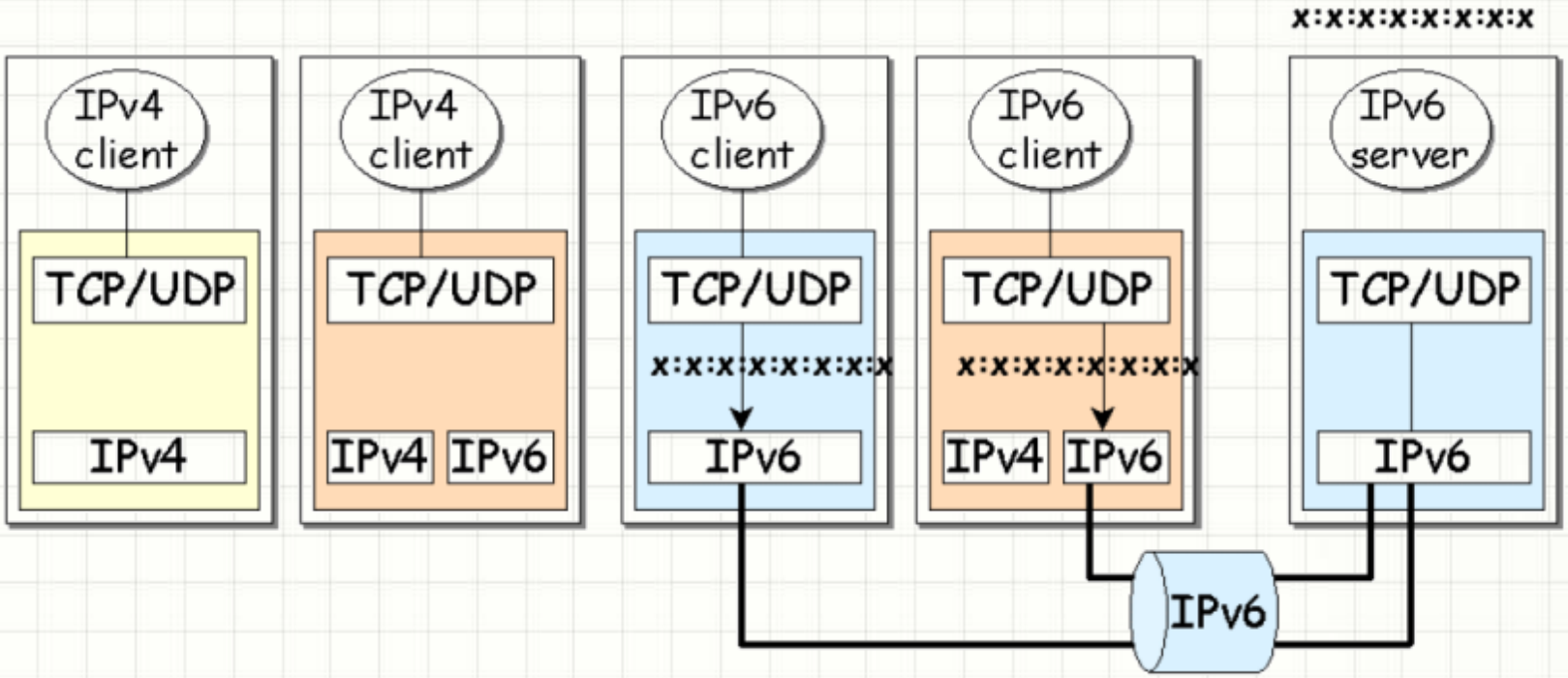
Transition scenario



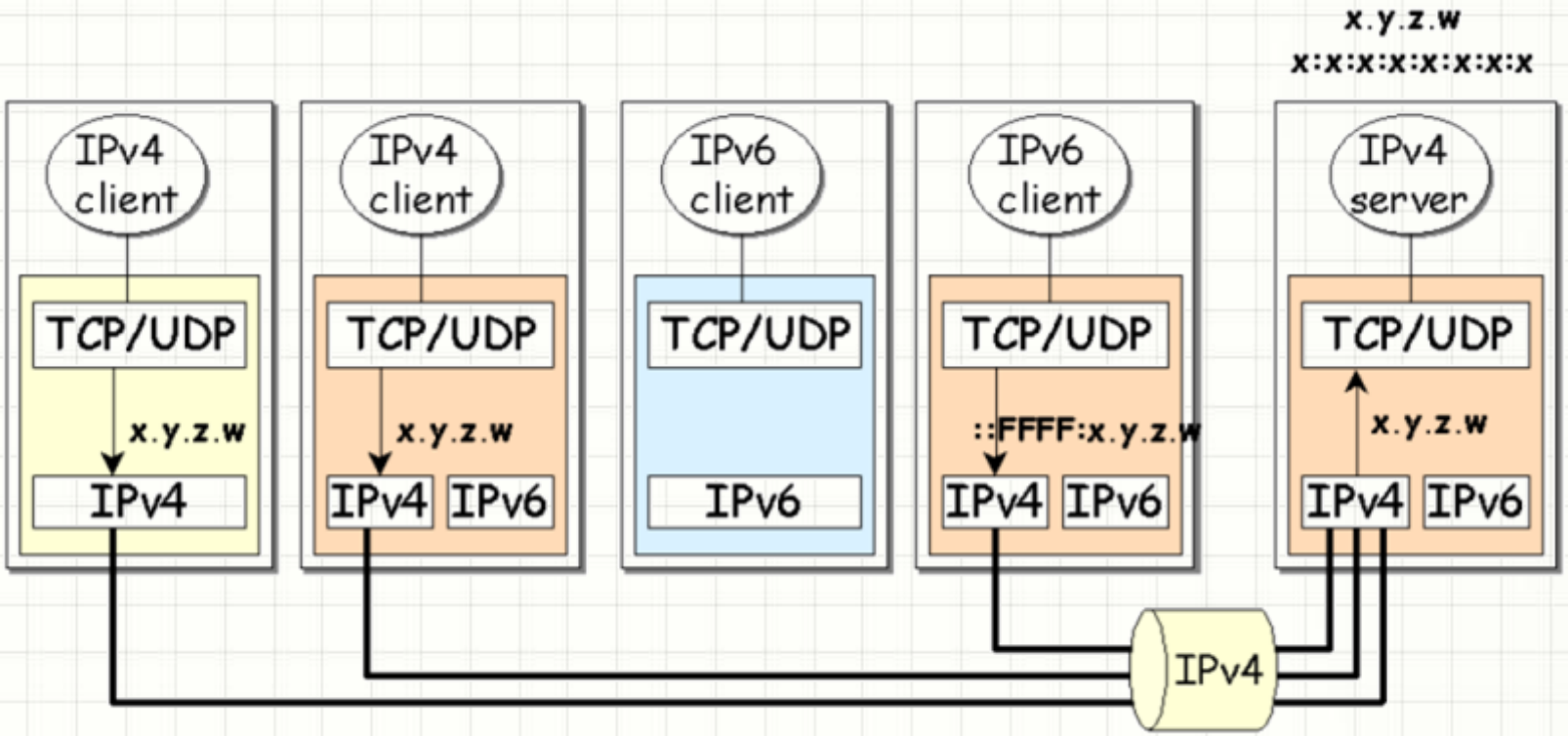
IPv4/IPv6 Interoperability (IPv4 server at IPv4 only node)



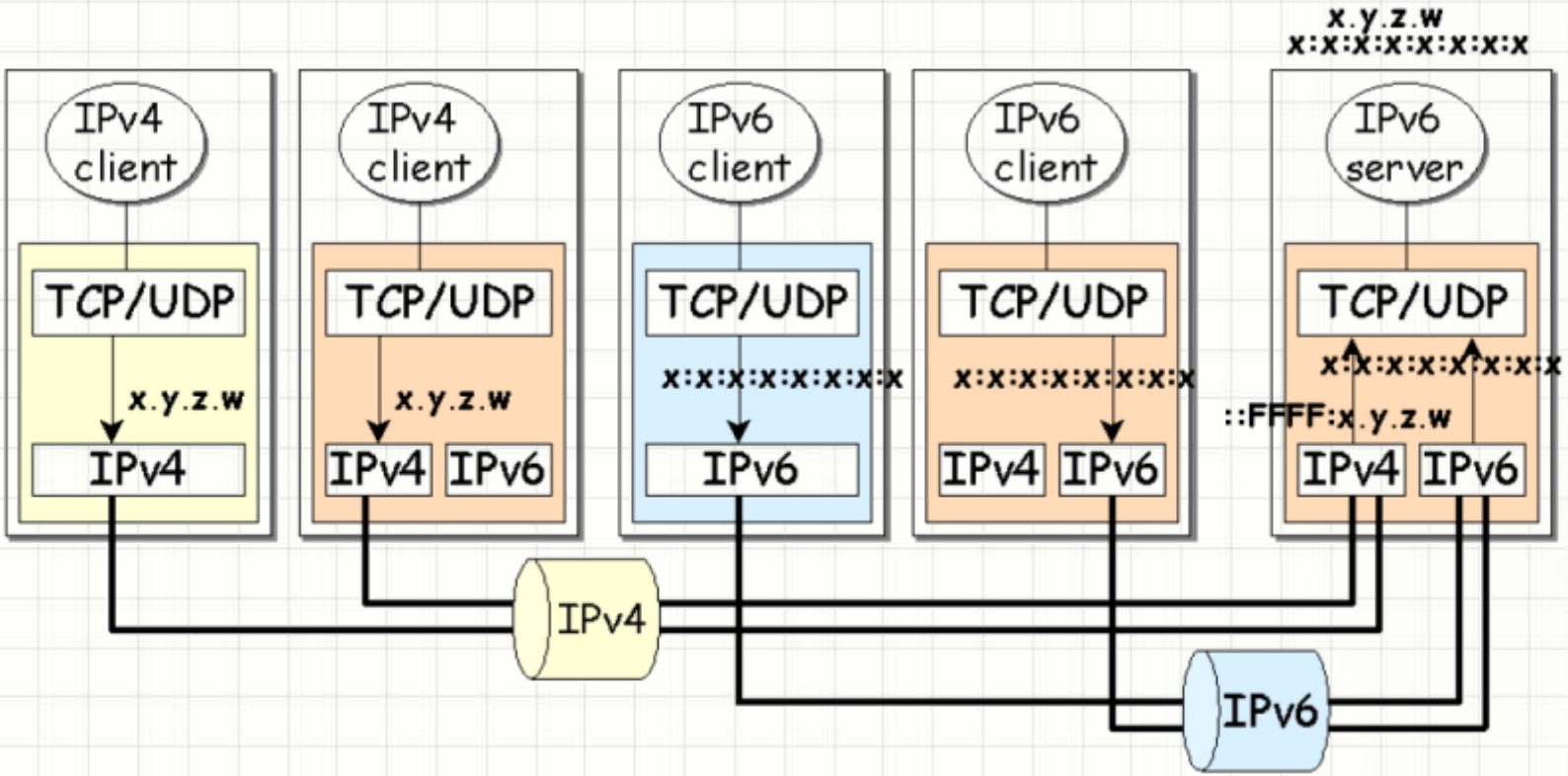
IPv4/IPv6 Interoperability (IPv6 server at IPv6 only node)



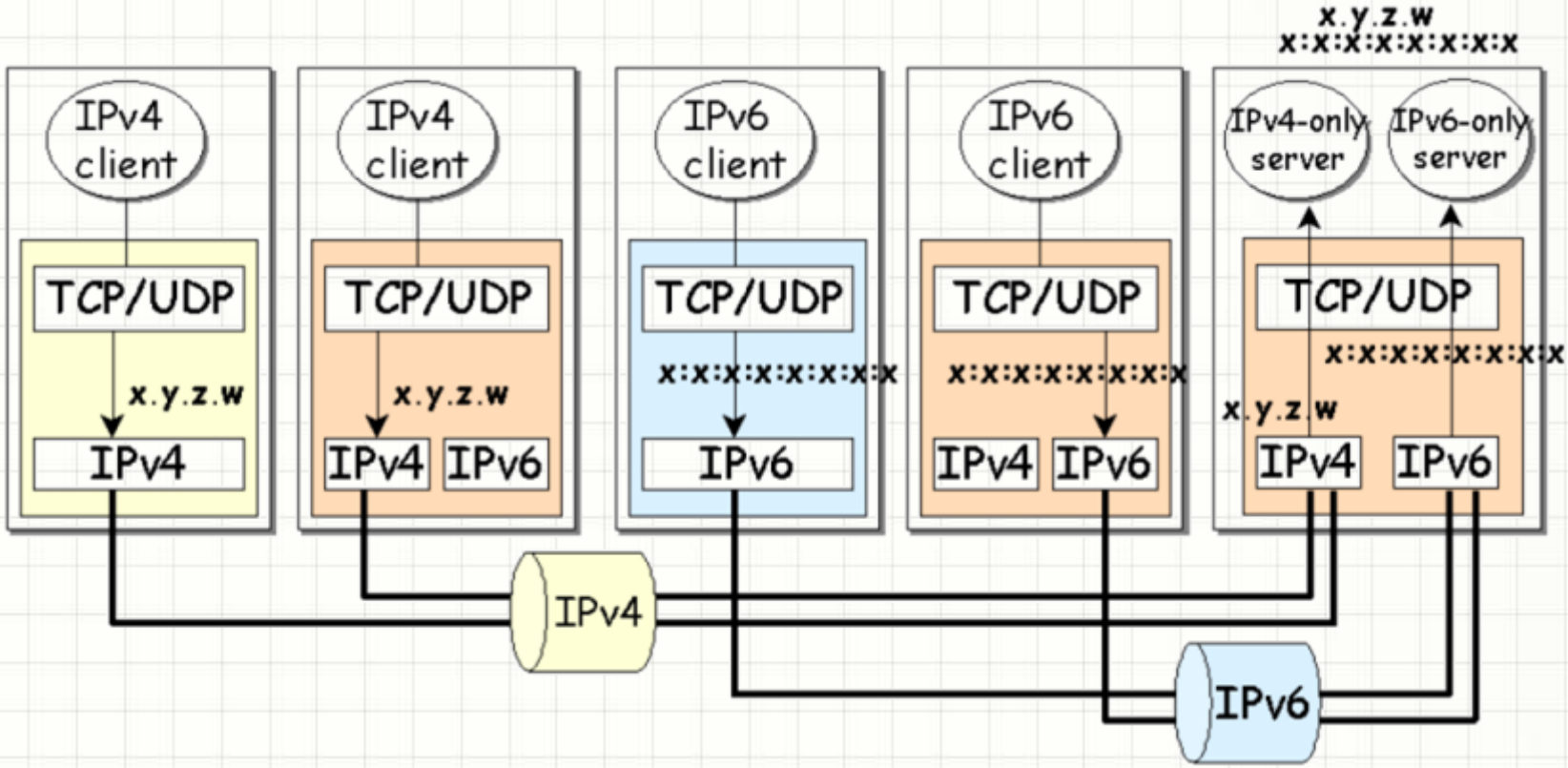
IPv4/IPv6 Interoperability (IPv4 server at dual stack)



IPv4/IPv6 Interoperability (IPv6 server at dual stack)



IPv4/IPv6 Interoperability (IPv4-only server and IPv6-only server at dual stack)



IPv4/IPv6 Interoperability Summary

		IPv4 server application		IPv6 server application	
		IPv4 node	Dual-stack	IPv6 node	Dual-stack
IPv4 client	IPv4 node	IPv4	IPv4	X	IPv4
	Dual-stack	IPv4	IPv4	X	IPv4
IPv6 client	IPv6 node	X	X	IPv6	IPv6
	Dual-stack	IPv4	IPv4 / X	IPv6	IPv6

Source Code Porting

Socket Address Structures (IPv4)

```
struct sockaddr {
    sa_family_t sa_family;      /* Address family */
    char sa_data[14];           /* protocol-specific address */
};
```

```
typedef uint32_t in_addr_t;
struct in_addr  {
    in_addr_t s_addr;                       /* IPv4 address */
};

struct sockaddr_in {
    sa_family_t sin_family;                 /* AF_INET */
    in_port_t sin_port;                     /* Port number.  */
    struct in_addr sin_addr;                /* Internet address.  */

    /* Pad to size of `struct sockaddr'.  */
    unsigned char sin_zero[sizeof (struct sockaddr) -
                           sizeof (sa_family_t) -
                           sizeof (in_port_t) -
                           sizeof (struct in_addr)];
};
```

Socket Address Structures (IPv6)

```
struct in6_addr {
    union {
        uint8_t u6_addr8[16];
        uint16_t u6_addr16[8];
        uint32_t 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 {
    sa_family_t sin6_family;    /* AF_INET6 */
    in_port_t sin6_port;        /* Transport layer port # */
    uint32_t sin6_flowinfo;     /* IPv6 flow information */
    struct in6_addr sin6_addr;  /* IPv6 address */
    uint32_t sin6_scope_id;     /* IPv6 scope-id */
  };
```

Socket Address Structures (Portable address)

```
/* Structure large enough to hold any socket address (with the
historical exception of AF_UNIX). 128 bytes reserved.  */

#if ULONG_MAX > 0xffffffff
# define __ss_aligntype __uint64_t
#else
# define __ss_aligntype __uint32_t
#endif
#define _SS_SIZE        128
#define _SS_PADSIZE     (_SS_SIZE - (2 * sizeof (__ss_aligntype)))

struct sockaddr_storage
  {
    sa_family_t ss_family;      /* Address family */
    __ss_aligntype __ss_align;  /* Force desired alignment.  */
    char __ss_padding[_SS_PADSIZE];
  };
```

Socket API – no change

```
int     socket  (int domain, int type, int protocol);
int     listen  (int s, int backlog);
ssize_t write   (int fd, const void *buf, size_t count);
int     send    (int s, const void *msg, size_t len, int flags);
int     sendmsg (int s, const struct msghdr *msg, int flags);
ssize_t read    (int fd, void *buf, size_t count);
int     recv    (int s, void *buf, size_t len, int flags);
int     recvmsg (int s, struct msghdr *msg, int flags);
int     close   (int fd);
int     shutdown(int s, int how);
```

Socket API – change

- The socket address provided by application to kernel

```
int bind    (int sockfd, struct sockaddr *my_addr, socklen_t addrlen);

int connect(int sockfd, const struct sockaddr *serv_addr,
            socklen_t addrlen);

int sendto (int s, const void *msg, size_t len, int flags,
            const struct sockaddr *to, socklen_t tolen);
```

- The socket address provided by kernel to application

```
int  accept      (int s, struct sockaddr *addr, socklen_t *addrlen);

int  recvfrom    (int s,  void  *buf, size_t len, int flags,
                  struct sockaddr *from, socklen_t *fromlen);

int  getpeername(int s, struct sockaddr *name, socklen_t *namelen);

int  getsockname(int s, struct sockaddr *name, socklen_t *namelen);
```

Typical update when porting to IPv6

- int **socket** (int domain, int type, int protocol);
  - IPv4 source code:

```
socket(PF_INET, SOCK_STREAM, 0);   /* TCP socket */
socket(PF_INET, SOCK_DGRAM,  0);   /* UDP socket */
```

  - IPv6 source code:

```
socket(PF_INET6, SOCK_STREAM, 0); /* TCP socket */
socket(PF_INET6, SOCK_DGRAM,  0); /* UDP socket */
```

Typical update when porting to IPv6

- int **bind** (int sockfd, struct sockaddr *my_addr, socklen_t addrlen);
  - IPv4 source code

```
struct sockaddr_in addr;
socklen_t          addrlen = sizeof(addr);

/*
   fill addr structure using an IPv4 address before calling socket
   funtion
*/

bind(sockfd,(struct sockaddr *)&addr, addrlen);
```

  - IPv6 source code:

```
struct sockaddr_in6 addr;
socklen_t           addrlen = sizeof(addr);

/*
   fill addr structure using an IPv6 address before calling socket
   function
*/

bind(sockfd,(struct sockaddr *)&addr, addrlen);
```

Typical update when porting to IPv6

- int **bind** (int sockfd, struct sockaddr *my_addr, socklen_t addrlen);
  - Portable source code

```
struct sockaddr_storage addr;
socklen_t               addrlen= sizeof(addr);

/*
   fill addr structure using an IPv4/IPv6 address and
   fill addrlen before calling socket function
*/

bind(sockfd,(struct sockaddr *)&addr, addrlen);
```

Typical update when porting to IPv6

- int **accept** (int s, struct sockaddr *addr, socklen_t *addrlen);
  - IPv4 source code

```
struct sockaddr_in addr;
socklen_t          addrlen = sizeof(addr);

accept(sockfd,(struct sockaddr *)&addr, &addrlen);

/*
   addr structure contains an IPv4 address
*/
```

  - IPv6 source code:

```
struct sockaddr_in6 addr;
socklen_t           addrlen = sizeof(addr);

accept(sockfd,(struct sockaddr *)&addr, &addrlen);

/*
   addr structure contains an IPv4 address
*/
```

Typical update when porting to IPv6

- int **accept** (int s, struct sockaddr *addr, socklen_t *addrlen);

  - Portable source code

```
struct sockaddr_storage addr;
socklen_t               addrlen = sizeof(addr);

accept(sockfd,(struct sockaddr *)&addr, &addrlen);

/*
   addr structure contains an IPv4/IPv6 address
   addrlen contains the size of the addr structure returned
 */
```

Address conversion functions

- IPv4 address conversion functions

```
/*
   From text to IPv4 binary representation
*/
int       inet_aton (const char *cp, struct in_addr *inp);
in_addr_t inet_addr( const char *cp);

/*
   From IPv4 binary to text representation
*/
char     *inet_ntoa(struct in_addr in);
```

- IPv4 /IPv6 address conversion functions

```
/*
   From presentation to IPv4/IPv6 binary representation
*/
int inet_pton(int family, const char *src, void *dst);

/*
   From IPv4/IPv6 binary to presentation
*/
const char *inet_ntop(int family, const void *src,
                      char *dst, size_t cnt);
```

Address conversion functions

- IPv4 source code

```
struct sockaddr_in addr;
char *straddr;

memset(&addr1, 0, sizeof(addr));
addr.sin_family = AF_INET;          /* family */
addr.sin_port = htons(MYPORT);      /* port, networt byte order */

/*
   from text to binary representation
*/
inet_aton("138.4.2.10", &(addr.sin_addr));


/*
   from binary to text representation
*/
straddr = inet_ntoa(addr.sin_addr);
```

Address conversion functions

- IPv6 source code:

```
struct sockaddr_in6 addr;
char straddr[INET6_ADDRSTRLEN];

memset(&addr, 0, sizeof(addr));
addr.sin6_family = AF_INET6;        /* family */
addr.sin6_port = htons(MYPORT);     /* port, networt byte order */

/*
   from presentation to binary representation
*/
inet_pton(AF_INET6, "2001:720:1500:1::a100",
          &(addr.sin6_addr));

/*
   from binary representation to presentation
*/
inet_ntop(AF_INET6, &addr.sin6_addr, straddr,
          sizeof(straddr));
```

Name resolving

struct hostent ***gethostbyaddr**(const void *addr, socklen_t len, int type);

struct hostent ***gethostbyname**(const char *name);

```
struct addrinfo {
    int     ai_flags;           /* AI_PASSIVE, AI_CANONNAME */
    int     ai_family;          /* AF_UNSPEC, AF_INET, AF_INET6 */
    int     ai_socktype;        /* SOCK_STREAM, SOCK_DGRAM ... */
    int     ai_protocol;        /* IPPROTO_IP, IPPROTO_IPV6 */
    size_t  ai_addrlen;         /* length of ai_addr */
    struct sockaddr ai_addr;    /* socket address structure */
    char   ai_canonname;        /* cannonical name */
    struct addrinfo ai_next;    /* next addrinfo structure */
};

/* function to get socket address structures */

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

/* function to free the resources allocated by getaddrinfo */

void freeaddrinfo(struct addrinfo *res);
```

Name resolving

```
n = getaddrinfo(hostname, service, &hints, &res);

/*
    Try open socket with each address getaddrinfo returned,
    until getting a valid socket.
*/

resave = res;

while (res) {
    sockfd = socket(res->ai_family,
                    res->ai_socktype,
                    res->ai_protocol);

    if (!(sockfd < 0))
        break;

    res = res->ai_next;
}

freeaddrinfo(ressave);
```

Application Deployment

Application		Deploy Node		Connections	
Network API used	Application type	IPv6 kernel support	Dual stack activated	IPv4 connection	IPv6 connection
IPv6 extensions (portable code)	IPv6-enabled	Yes	Yes	IPv4 stack	IPv6 stack
IPv6 extensions (portable code)	IPv6-enabled		No (only IPv4 is activated)	IPv4 stack	Address resolution successful Connection error
Without IPv6 extensions (old API)	IPv4-only	No		IPv4 stack	Error

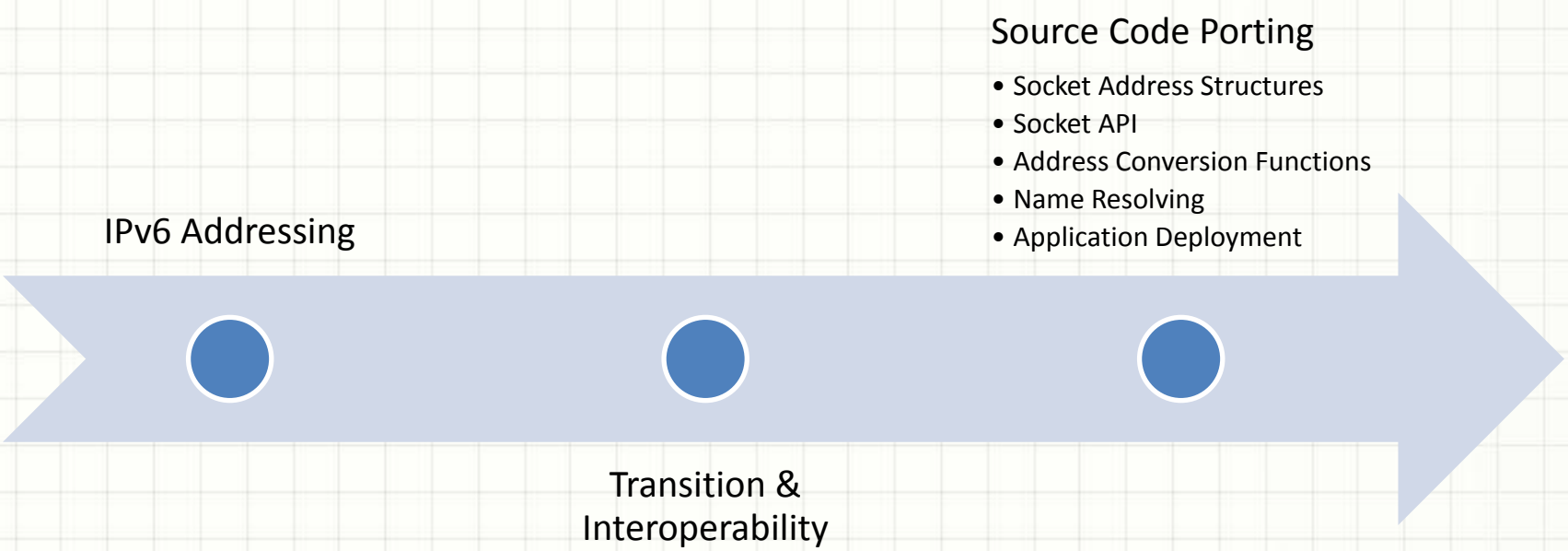
Q: How to know whether the linux kernel support ipv6?

A: *ls -lrt /proc/net/if_inet6*

Q: How to know whether the ipv6 stack be activated?

A: *ping6 –c5 ::1*

Summary


Source Code Porting
• Socket Address Structures
• Socket API
• Address Conversion Functions
• Name Resolving
• Application Deployment
IPv6 Addressing
Transition &
Interoperability

QUESTIONS?