Application Development

Carl Williams
IPv6 Forum Fellow
Rocky Mountain IPv6 Task Force
IPv6 Summit Event (Denver)
April 21-22, 2009

Outline of Presentation

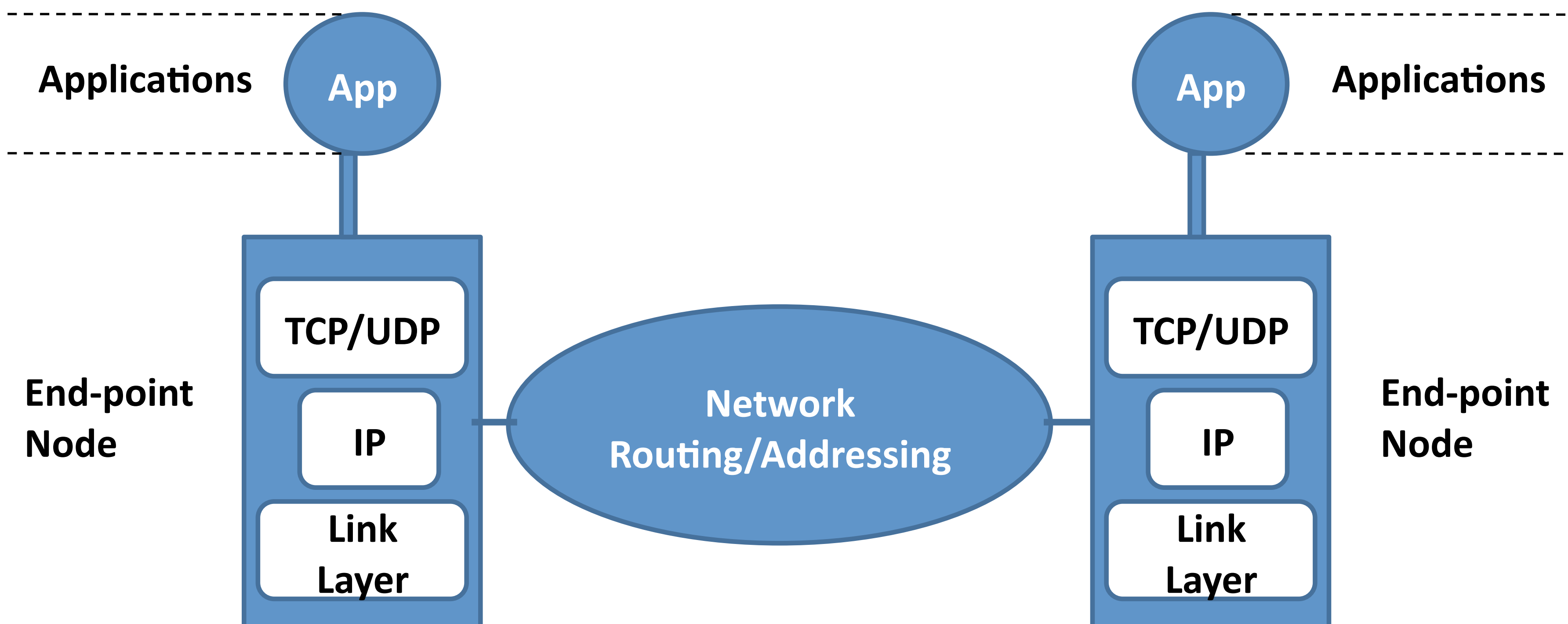
- As IPv6 is deployed, the application developers and the administrators will face several problems.
- This presentation clarifies the problems occurring in transition period between IPv4 applications and IPv6 applications.
- The presents guidelines that help application developers understand how to develop IP version-independent applications during the transition period.

What application developers can do

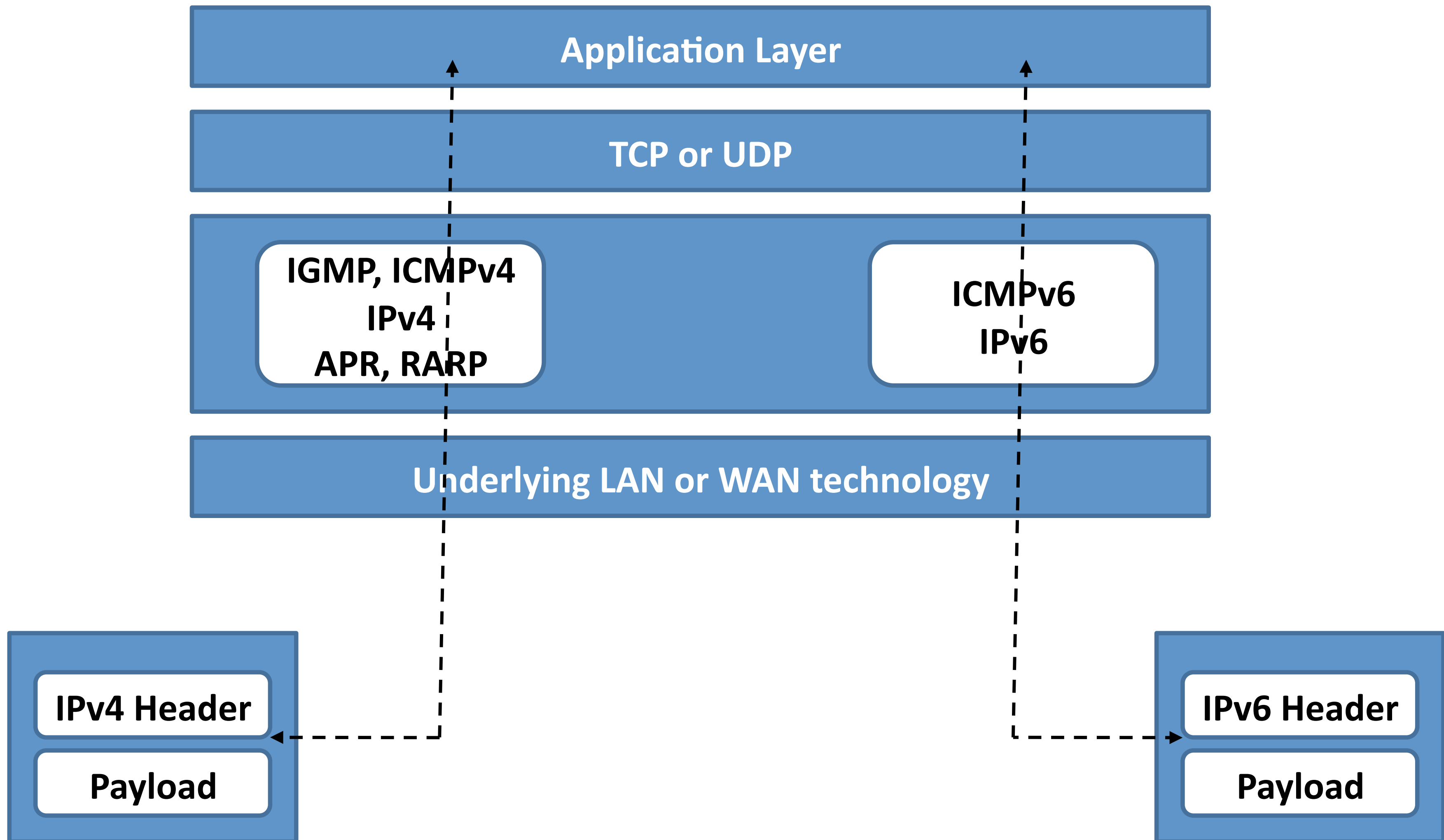
- It is important for programmers to “think IPv6”: To speed up IPv6 adoption
- Avoid risk of rolling out non compatible IPv6 programs once IPv6 will take place

Application Interoperability

Application Perspective within the Transition Architecture



Application Perspective within a Dual Stack



Application Transition Issues

Dual-stack vs. application versions

Operating System being dual stack does not mean having both IPv4 and IPv6 applications.

DNS name resolution

A client application can not know the version of peer application by only doing a DNS name lookup.

Application selection

Users may be confused by their various application versions (IPv4-only, IPv6-only, IPv4/IPv6) because they don't know the version of the peer application by DNS query results.

Impact of IPv6 stack on Applications

- Applications in a dual stack host prefer to use IPv6 address instead of IPv4
- In IPv6, it is normal to have multiple addresses associated to an interface. In IPv4, no address is associated to a network interface, while at least one (link local address) is in IPv6.
- The two protocols cannot communicate directly, even in dual stack hosts. There are some different methods to implement such communication, but they are out of scope of this document.

Impact of DNS on applications in a mixed IPv4/IPv6 world

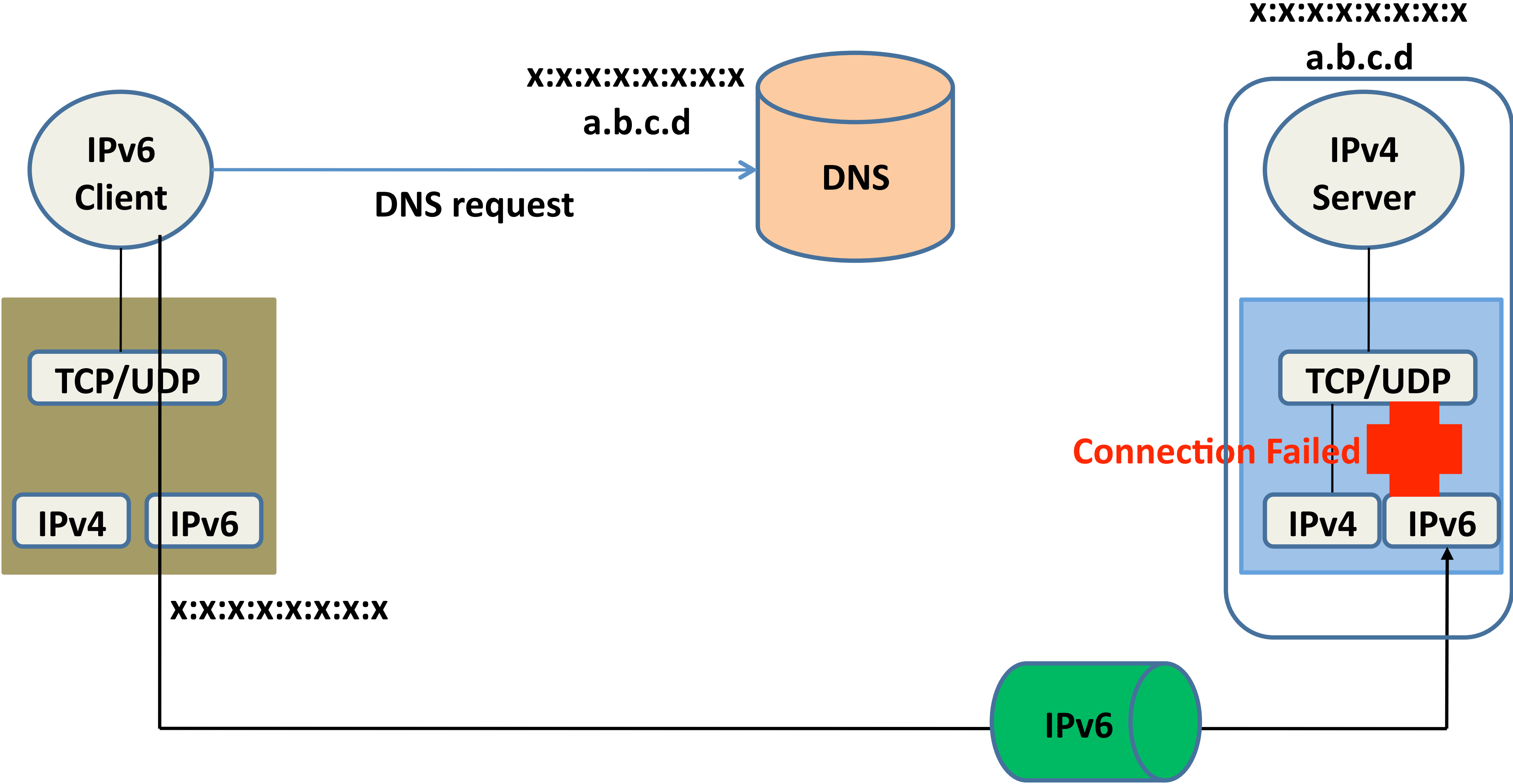
- Applications should try all addresses (both v4 and v6) they get from DNS if necessary. Applications should use the **getaddrinfo()** resolver function and try the addresses in the order it returns them; often **IPv6** first. Some applications fail to failover to IPv4 when **IPv6** fails
  - May result in long timeouts. Might wait up to 30s per address if no TCP/ICMP error
  - Also some firewalls just discard DNS packets with AAAA requests, resulting in long timeout, ad.doubleclick.net is one problem

Returning multiple addresses

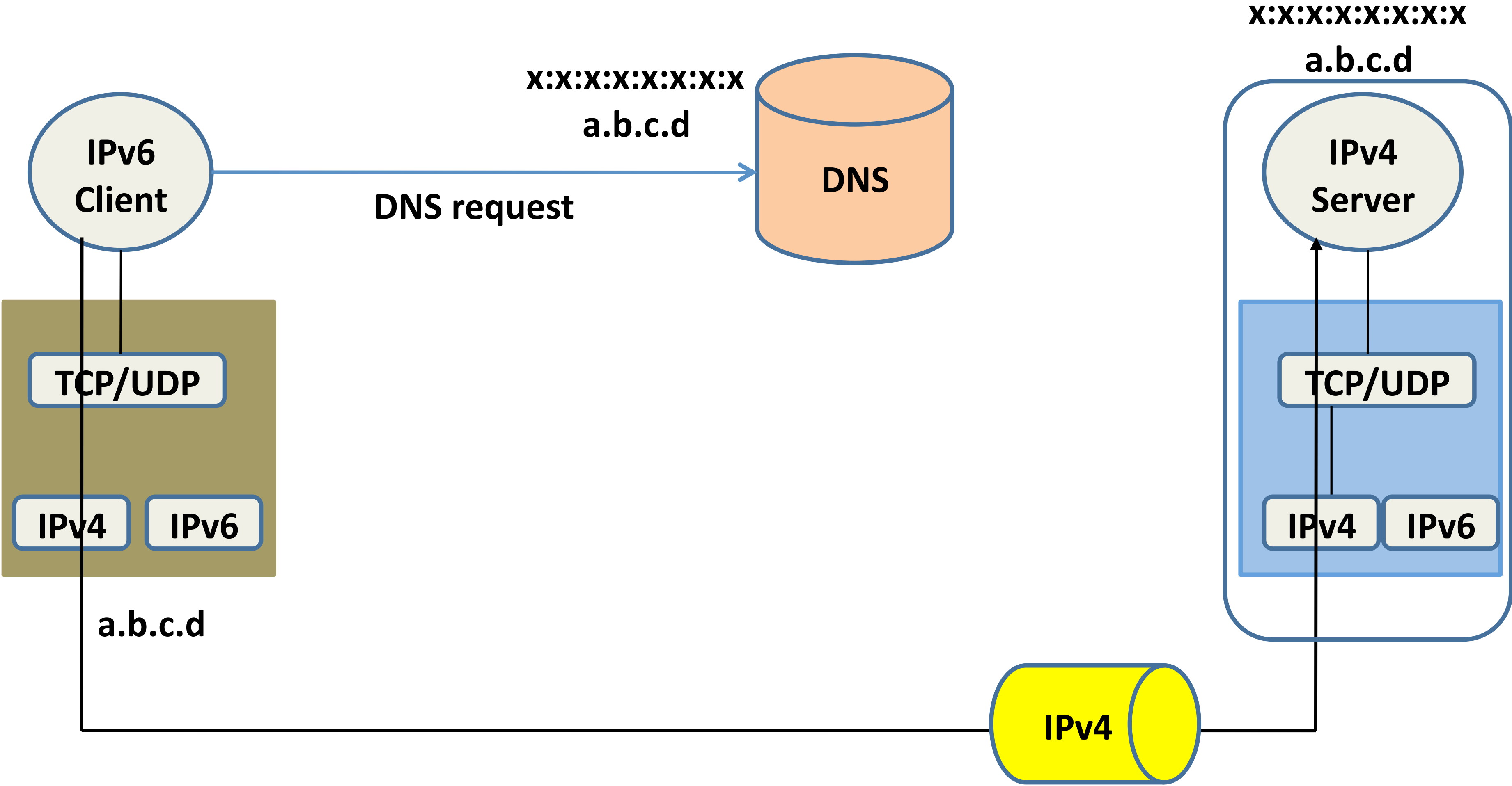
- **getaddrinfo()** can return multiple addresses,
- if a host have multiple address with multiple address families, as below:

```
testhost   IN  A a.b.c.d
           IN  AAAA   x:x:x:x:x:x:x:x
```

IPv6 enabled client connecting to an IPv4 server at dual stack node

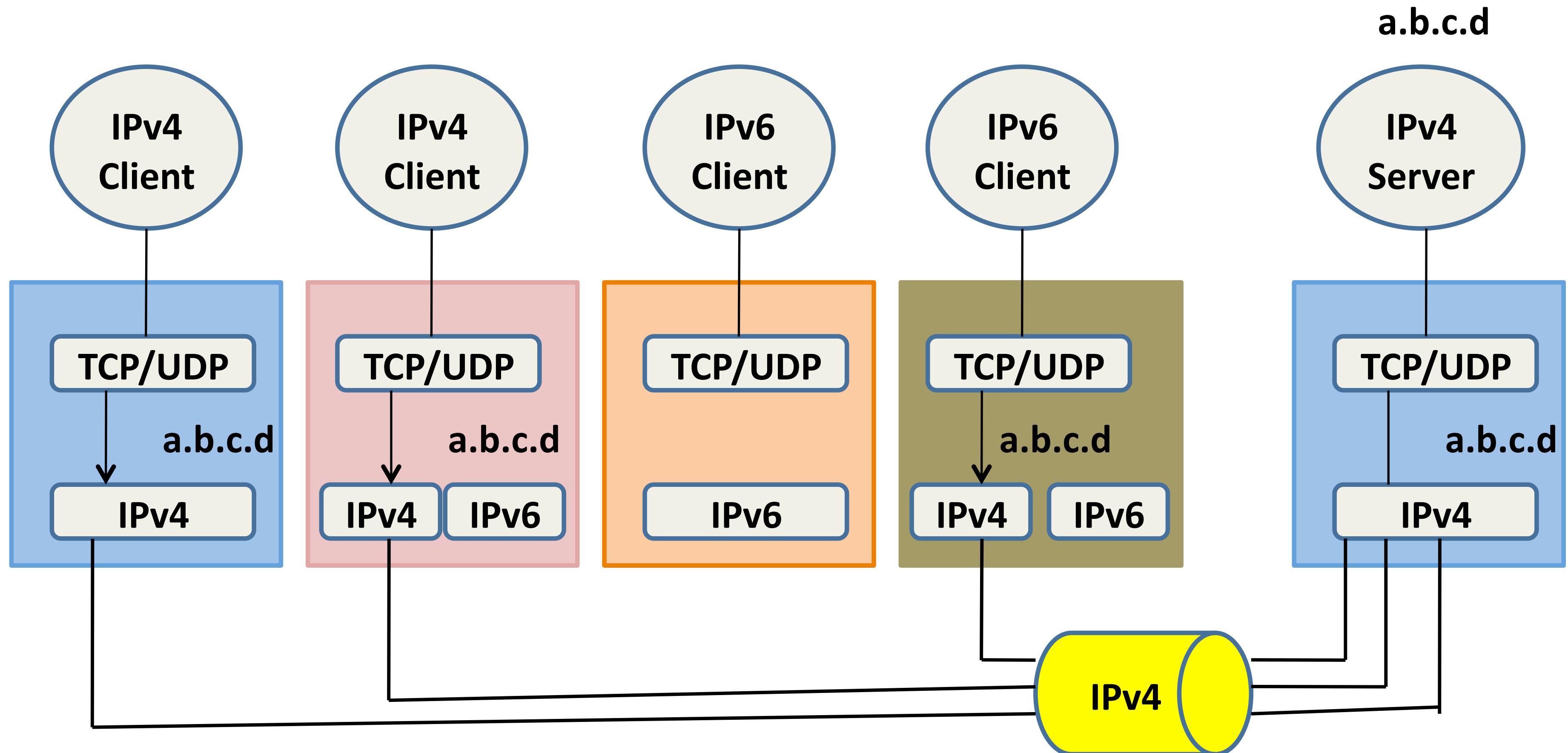


IPv6 enabled client connecting to an IPv4 server at dual stack node

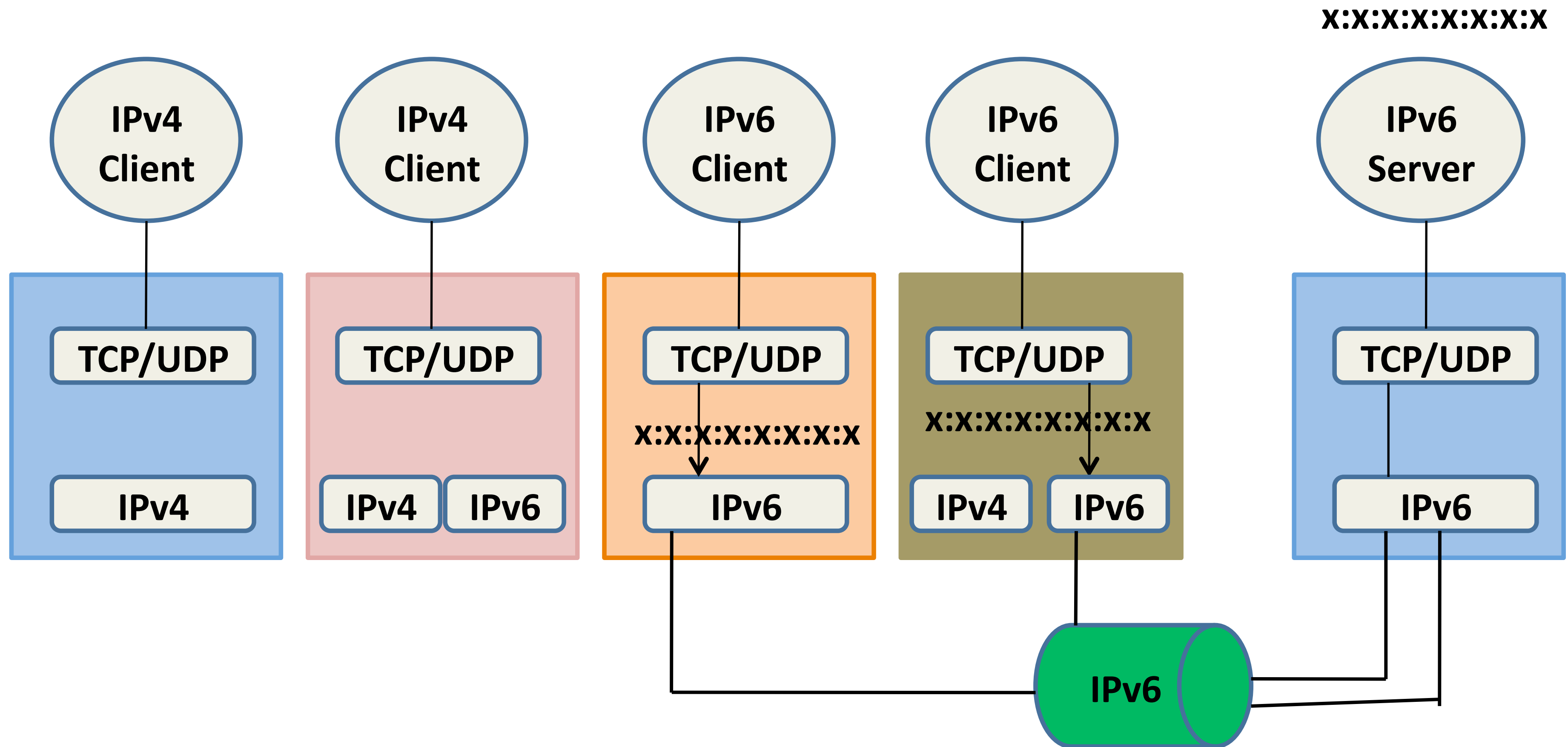


CASES
APPLICATION INTEROPERABILITY

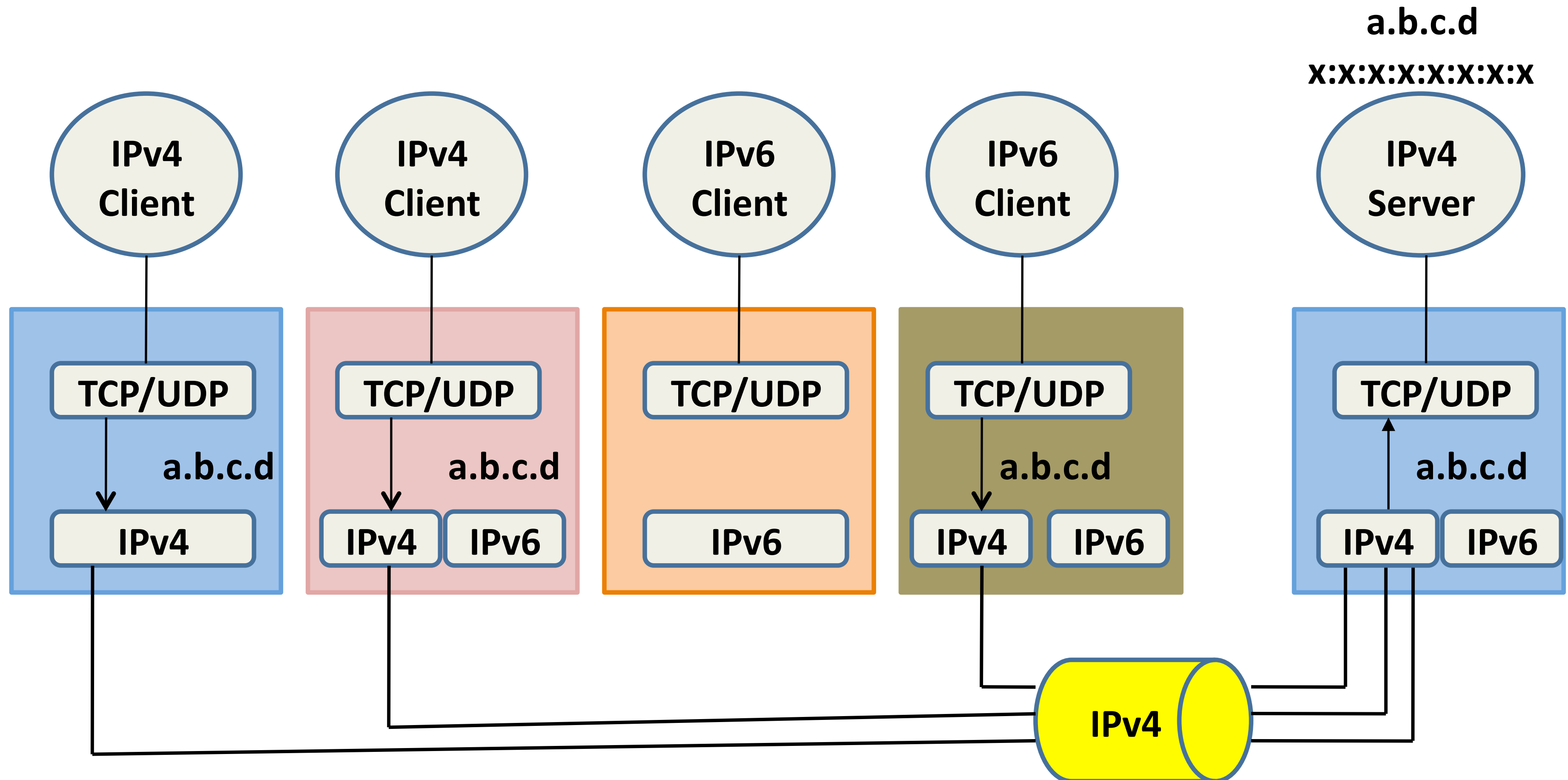
1 IPv6/IPv4 clients connecting to an IPv4 server at IPv4-only node



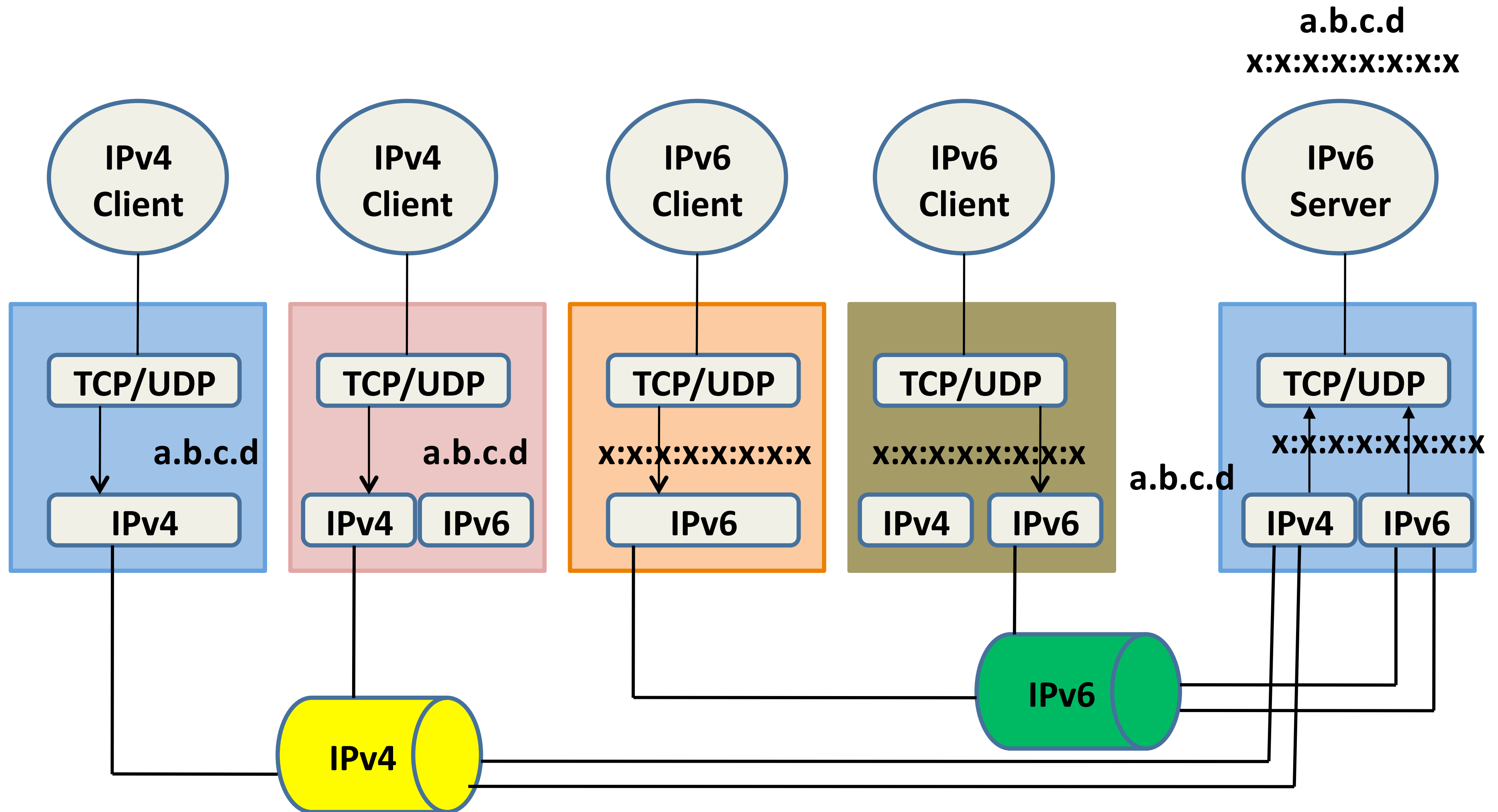
2 IPv6/IPv4 clients connecting to an IPv6 server at IPv6-only node



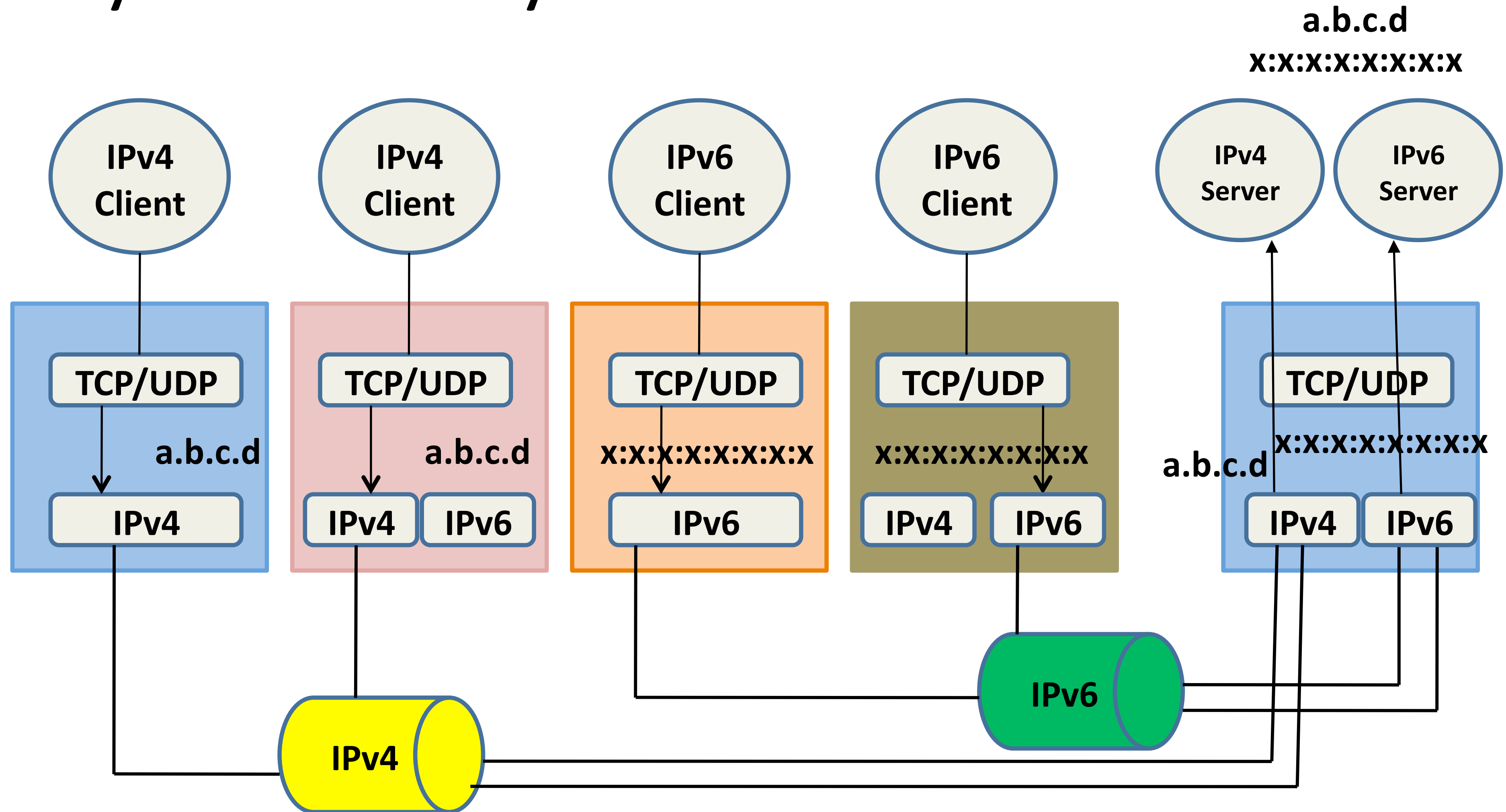
3 IPv6/IPv4 clients connecting to an IPv4 server at dual stack node



4 IPv6/IPv4 clients connecting to an IPv6 server at dual stack node



5 IPv6/IPv4 clients connecting to an IPv4-only & IPv6-only server at dual stack node



Client server & network type combinations

		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

Guideline Summary

- In order to allow applications to communicate with other IPv6 nodes, the first priority is to convert the applications supporting both IPv4 and IPv6.
- The applications should do iterated jobs for finding the working address out of addresses returned by getaddrinfo().
- The applications will have to work properly in IPv4-only nodes (whether IPv6 protocol is completely disabled).

Application development

- The same binary should work on hosts that support only one or both IP protocols
- Applications must be changed to use **IPv6** socket APIs (RFC 3493 and RFC 3542)

Application issues

- **IPv6** addresses in URLs (RFC 2732)
  - E.g. http://[2001:610:148:dead:210:18ff:fe02:e38]:80/
  - Not all applications support this
- IPv4 mapped **IPv6** addresses
  - Some operating systems allow applications to send/ receive IPv4 on **IPv6** sockets
  - An IPv4 address a.b.c.d is represented as ::ffff:a.b.c.d
  - Some poorly written applications may require you to write **IPv6** ACLs for mapped addresses to limit IPv4

Programming Languages

- Perl
  - Special modules like Socket6 and IO::Socket::INET6
- Python 2.3.4 and beyond works with **IPv6**
  - However, Windows binaries at python.org does not support it. 2.4 binaries will be built with **IPv6** support
- PHP
  - Partial **IPv6** support
  - Many PHP scripts work with **IPv6** with no change
- Java
  - SUN Java SDK 1.4 and beyond has **IPv6** support
  - Many Java applications work with **IPv6** with no change due to the higher level API

Application Interoperability

- For many years we will live in a dual IP protocol version world.
- We will see progressive spread of IPv6 deployment and a very relevant residual usage of IPv4 all over the world
- Ways for interoperating between two incompatible protocols need to be identified

Network Transparent Programming

- For Network Transparent Programming it is important to pay attention to:
  - Use of name instead of address in applications is advisable; in fact, usually the hostname remains the same, while the address may change more easily.
  - From application point of view the name resolution is a system independent process.
- Avoid the use of hardcoded
  - numerical address and binary
  - representation of addresses.
- Use *getaddrinfo and getnameinfo functions.*

Identify code to change

- To rewrite an application with IPv6 compliant code, the first step is to find all IPv4 dependent functions.
- A simple way is to check the source and header file with UNIX grep utility or using the IPv6 code scrubber. Example grep:

```
$ grep sockaddr_in *c *.h
$ grep in_addr *.c *.h
$ grep inet_aton *.c *.h
$ grep gethostbyname *.c *.h
```

Rewriting Applications

- Developers should pay attention to hardcoded numerical address, host names, and binary representation of addresses.
- It is recommended to put all network functions in a single file.
- It is also suggested to replace all *gethostbyname* with the *getaddrinfo function, a simple switch can be* used to implement protocol dependent part of the code.
- Server applications must be developed to handle multiple listen sockets, one per address family, using the *select call.*

Traditional IPv4 coding

```
#define PORT 2000              /* This definition is a number */

void server ()
{
int Sock;                      /* Descriptor for the network socket */
struct sockaddr_in SockAddr;   /* Address of the server socket descr */

   if ( ( Sock = socket(AF_INET, SOCK_STREAM, 0)) < 0 ) {
      error("Server: cannot open socket.");
      return;
   }

   memset(& SockAddr, 0, sizeof(SockAddr));
   SockAddr.sin_family = AF_INET;
   SockAddr.sin_addr.s_addr= htonl(INADDR_ANY); /* all local addresses */
   SockAddr.sin_port = htons(PORT);             /* Convert to network byte order */

   if (bind(Sock, (struct sockaddr *) &SockAddr, sizeof(SockAddr)) < 0) {
      error("Server: bind failure");
      return;
   }

   /* ... */
```

The code must be duplicated for each address family

With IPv6 – a new style

```
#define PORT "2000"                        /* This definition is a string */

void server ()
{
int Sock;                                  /* Descriptor for the network socket */
struct addrinfo Hints, *AddrInfo;          /* Helper structures */

   memset(&Hints, 0, sizeof(Hints));
   Hints.ai_family = AF_UNSPEC;             /* or AF_INET / AF_INET6 */
   Hints.ai_socktype = SOCK_STREAM;
   Hints.ai_flags = AI_PASSIVE;             /* ready to a bind() socket */

   if (getaddrinfo(NULL /* all local addr */, PORT, Hints, AddrInfo) != 0) {
      error("Server: cannot resolve Address / Port ");
      return;
   }

   // Open a socket with the correct addres
   if ((Sock=socket(AddrInfo->ai_family, AddrInfo->ai_socktype, AddrInfo->ai_protocol))<0){
      error("Server: cannot open socket.");
      return;
   }

   if (bind(Sock, AddrInfo->ai_addr, AddrInfo->ai_addrlen) < 0) {
      error("Server: bind failure");
      return;
   }
   /* ... */
```

Family-independent code

Fills some internal structures with family-independent data using literal / numeric host and port

Data returned by getaddrinfo() is used in a family-independent way

Adding IPv6 code to Old IPv4 Apps (1/2)

- We need to locate the code that needs to be changed
  - "string search" to locate the system calls related to the `socket` interface
    - This is simple
  - "visual inspection" for other parts of the code
    - This is not
- System calls related to the **`socket`** interface
  - Convert part of the code to become protocol independent
    - The most part of socket functions
  - Add special code for IPv6
    - Some functions (**`getsockopt(),setsockopt()`**) which behave differently in IPv4 and IPv6

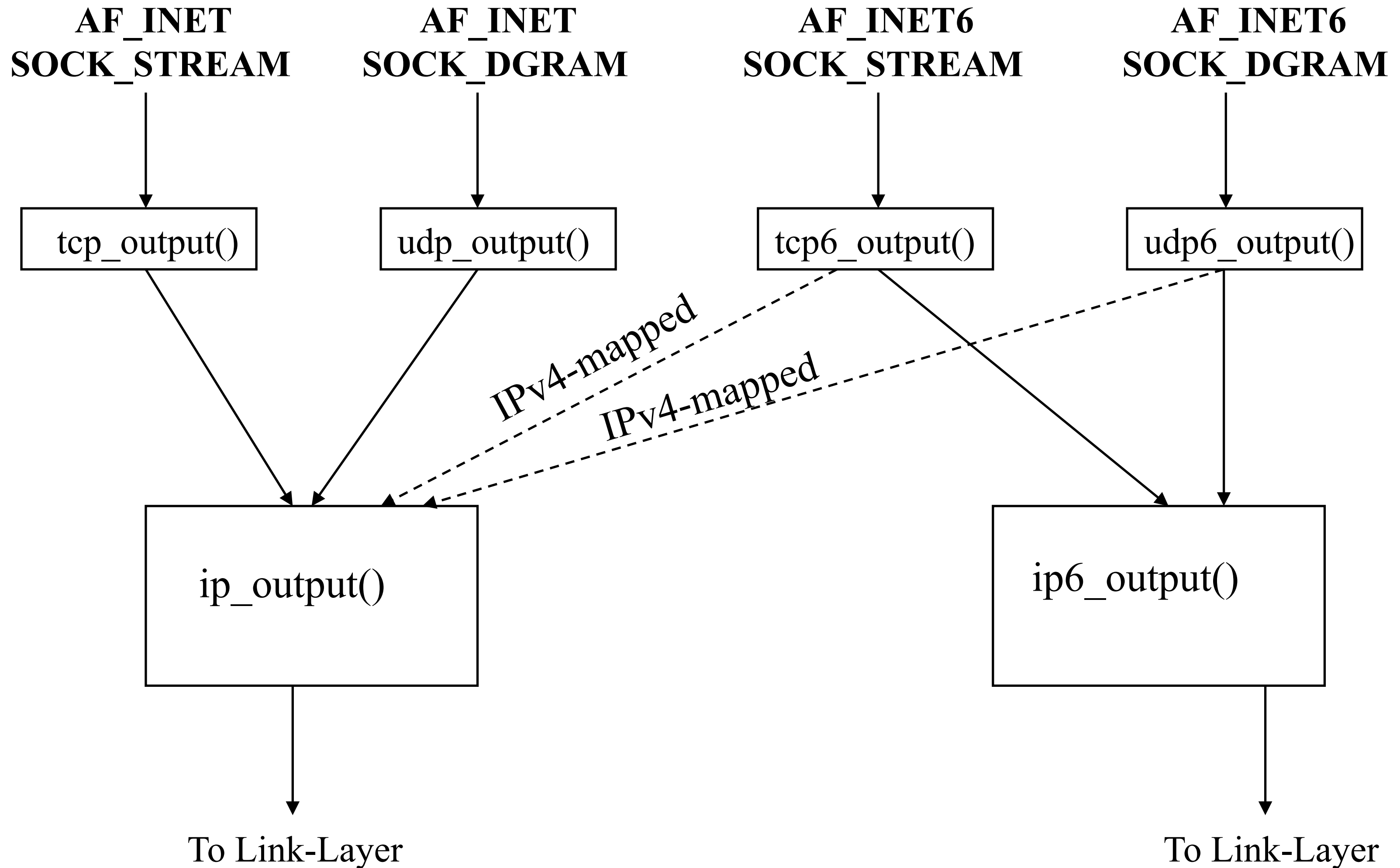
Adding IPv6 code to Old IPv4 Apps (2/2)

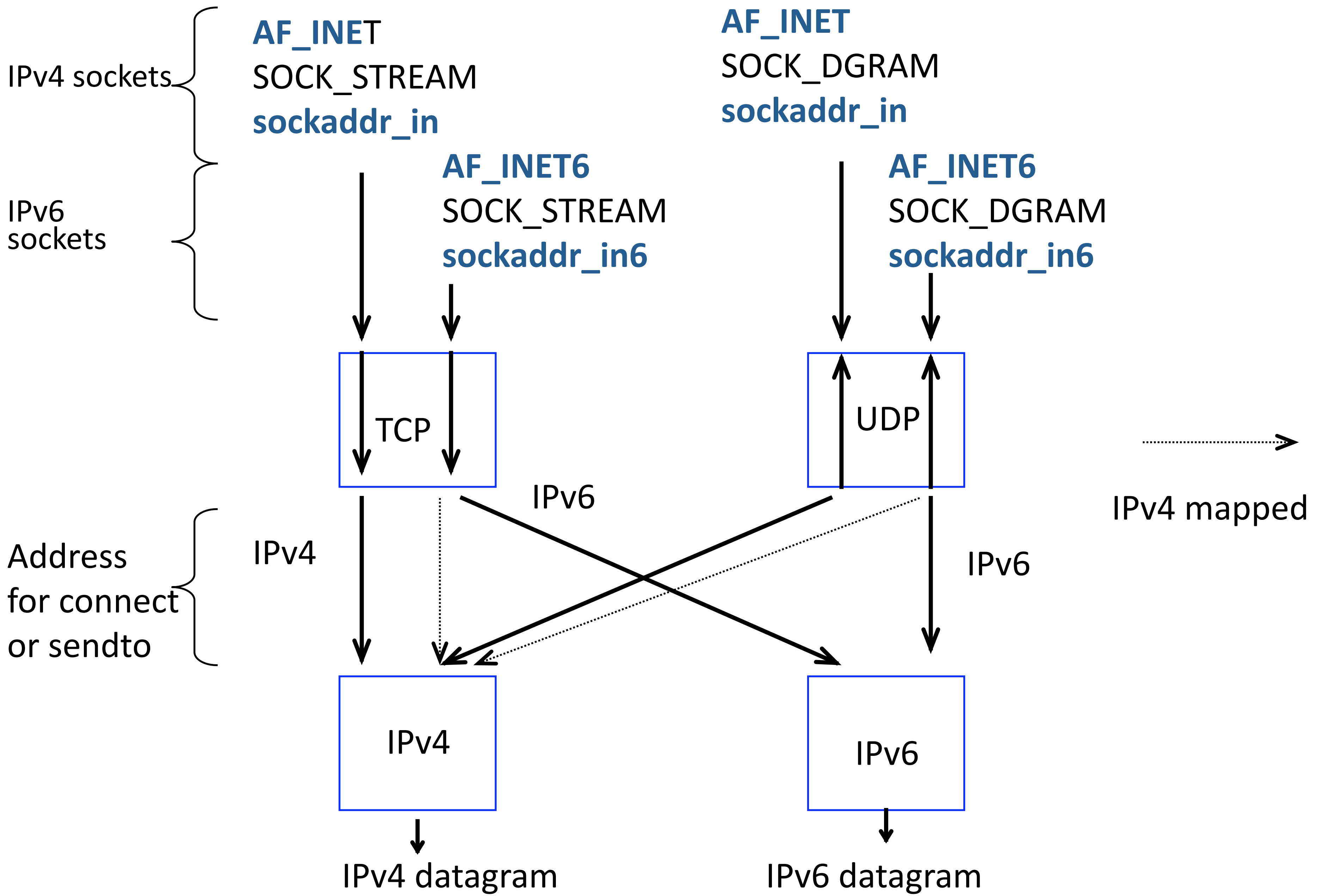
- Other code
  - Custom control used as input for an IPv4 address
  - Parsing or URLs
    - Several allowed strings
      - http://203.178.141.194
      - http://www.kame.net
      - http://2001:200:0:8002:203:47ff:fea5:3085
    - The ":" symbol is a "port delimiter" in IPv4, while it is the "address separator" in IPv6
      - http://www.kame.net:80
      - http://[2001:200:0:8002:203:47ff:fea5:3085]:80
  - Application-layer protocol
    - Is this protocol defining a field that carries IPv4 addresses (e.g. peer-to-peer applications)?
  - Difficult to locate

Writing new apps with both IPv4 and IPv6 Support

- For the most part, this is much easier than writing IPv4-only applications with the older BSD programming style
  - recommended to use: **`getaddrinfo()` and `getnameinfo()`**
  - Code is smaller and easier to understand than the one written according to the old socket interface
- Some code may be duplicated
  - **`getsockopt(),setsockopt()`**
  - URL parsing

Dual-Stacked Nodes:
Sending IPv4 and IPv6 Packets




IPv4 sockets
IPv6 sockets
Address for connect or sendto
AF_INET
SOCK_STREAM
sockaddr_in
AF_INET6
SOCK_STREAM
sockaddr_in6
AF_INET
SOCK_DGRAM
sockaddr_in
AF_INET6
SOCK_DGRAM
sockaddr_in6
TCP
UDP
IPv4 mapped
IPv6
IPv4
IPv6
IPv4
IPv6
IPv4 datagram
IPv6 datagram

Current Status of IPv6 Support for Networking Applications

- List of IPv6 Supported Networking Apps.
  - http://www.deepspace6.net/docs/ipv6_status_page_apps.html
- IPv6 application and patch database
  - http://ipv6.niif.hu/m/ipv6_apps_db/

Multicast capable applications

- Mbone tools, vic/rat etc
  - **IPv6** multicast conferencing applications
  - http://www-mice.cs.ucl.ac.uk/multimedia/software/
- VideoLAN
  - Video streaming, also **IPv6** multicast. Server and client
  - Many operating systems, both Windows and UNIX
  - http://www.videolan.org/
- DVTS http://www.sfc.wide.ad.jp/DVTS/
  - Streaming DV over RTP over IPv4/**IPv6**
  - DV devices using Firewire can be connected to two different machines and you can stream video between them over the Internet
- Mad flute
  - Streaming of files using multicast (IPv4/**IPv6** ASM/SSM)
  - Linux and Windows (not totally sure about *BSD status)
  - http://www.atm.tut.fi/mad/

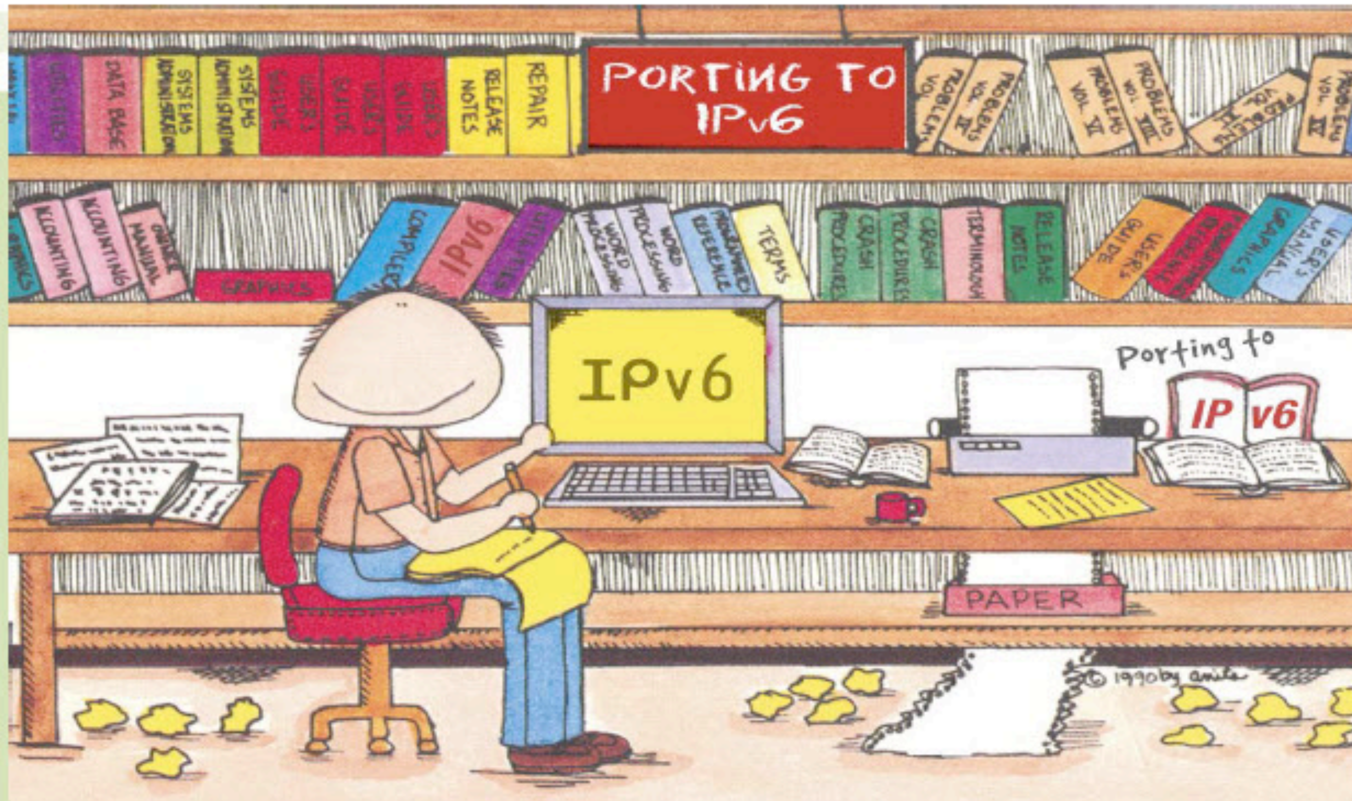
Conclusions for Application Development

- Effort required to add IPv6 support to and old IPv4-only application is not negligible
  - Far more than 50% of the lines of code need to be changed
  - Hidden costs (input forms, application-dependent protocols, etc.)
- Creation of new IPv4 and IPv6 applications from scratch
  - The socket interface is simpler than before
  - Some common issues:
    - Fallback: for clients
    - Dual-socket bind: for servers

References

- RFC 4038 on Application Aspects of IPv6 Transition
- RFC 3542 on Advanced Sockets API for IPv6
- RFC 3493 on Basic Socket Interface Extensions for IPv6

Acknowledgements



Jim Bound, Eva Castro, Pekka Savola

Thanks!