Interoperability between IPv4 and IPv6 Clients and Servers

Shilpa Verma and Kunal Meher

Abstract—**Most applications today support IPv4 and thus there is a need to run these applications on IPv6 access network. IPv4 can't meet the rising needs of the present internet. Moreover, IPv6 can solve the problems that IPv4 faced, and present more new features. The Internet must make the transition from IPv4 to IPv6. So the application which is based on IPv4 will be applied in the environment of IPv6. At present, windows system is widely applied. Many applications use Socket Application Program Interface.**

Index Terms—**Compability, dual stack, socket.**

I. Introduction

Although porting an application from IPv4 to IPv6 is not a difficult task there are many hidden problems that must be considered and carefully analyzed. Most of the third party applications are not aware of the change of IP versions and may not be well designed for easy porting. These old applications need to be carefully analyzed and ported to avoid any functionality and interoperability problems on the new OS. Besides that applications need to be designed to run on IPv4 and IPv6 during the transition period. This needs great flexibility on the application itself to be aware of IP versions and adapt accordingly for reliable communication.

II. Transition of Socket Applications From IPv4 to IPV6

Socket is the basic interface of the internet application that is based on TCP/IP protocol which is proposed by University of California BSD UNIX Branch. Through the interface, it can communicate with any computer which has socket interface. It also can achieve information transmission, sending and receiving. Windows socket 2 is developed from Berkeley socket and provides support for IPV6.

It is necessary to include the files "winsock2.h", "ws2tcpip.h", "tpipv6.h" and import "ws2_32" library in order to get the support of Windows Socket 2 when applications supporting IPv6 are developed under the VC++6.0 environment [1].

There are two ways for Socket Programming:

1) Socket Programming based on TCP (Connection-Oriented)
2) Socket Programming based on UDP (Connection-less)

Manuscript received February 21, 2012; revised May 2, 2012. This work was supported as part of the partial fulfillment of the requirements of the degree of M.E. in Computer Engineering in Thadomal Shahani Engineering College, Mumbai.

S. Verma is with the Thadomal Shahani Engineering College, Bandra, Mumbai-400050 (e-mail: shilpachandra14@yahoo.com).

K. M. Meher is with Xavier Institute of Engineering, Mahim, Mumbai-400016 (e-mail: kunalmeher@gmail.com).

The socket interface API consists of a few distinct components:

- Core socket functions:

The functions that deal with setting up and tearing down TCP connections and sending and receiving UDP packets. These functions need not change for IPv6, but a new IPv6-specific address data structure is needed.

- Address data structures:

The "sockaddr_in" structure is the protocol-specific data structure for IPv4. This data structure actually includes 8octets of unused space. Unfortunately, the "sockaddr_in" structure is not large enough to hold the 16-octet IPv6 address as well as the other information (address family and port number) that is needed. So a new address data structure "sockaddr_in6" is defined for IPv6.

- Name-to-address translation functions:

The name-to-address translation functions for IPv4 in the socket interface are gethostbyname() and gethostbyaddr(). New functions defined to support IPv4 and IPv6 are getaddrinfo() and getnameinfo().

- Address conversion functions:

The address conversion functions inet_ntoa() and inet_addr() convert IPv4 addresses between binary and printable form. These functions are quite specific to 32-bit IPv4 addresses. inet_ntop() and inet_pton() are two analogous functions that convert both IPv4 and IPv6 addresses.

The difference of IPv4 and IPv6 are the address size and format, protocol family, transport layer API function and domain name resolve function. The socket type, address structure, address resolution function and transfer function are different when they are mapped to the program. The following is the comparison between IPv4 application, IPv6 application and protocol-independent application, shown in Fig. 1 Most core socket functions are used under IPv6 condition, such as socket(), bind(), accept(), connect() and so on, except for some appropriative functions of IPv4 which need to be modified [2].

Function description	IPv4 application	IPv6 application	Protocol-independent application
Protocol family	AF_INET	AF_INET6	AF_UNSPEC
Address structure	sockaddr_in	sockaddr_in6	sockaddr_storage
Binary to presentation format	inet_ntoa()	inet_ntop()	inet_ntop()
Presentation to binary format	inet_addr()	inet_pton()	inet_pton()
Get IP address by name	gethostbyname()	getaddrinfo()	getaddrinfo()
Get name by IP address	gethostbyaddr()	getnameinfo()	getnameinfo()
Create Socket	socket()	socket()	socket()
Bind IP address	bind()	bind()	bind()
Listen socket	listen()	listen()	listen()
Receive TCP connection	accept()	accept()	accept()
Establish TCP connection	connect()	connect()	connect()
Send data (TCP)	send()	send()	send()
Send data (UDP)	sendto()	sendto()	sendto()
Receive data (TCP)	recv()	recv()	recv()
Receive data (UDP)	recvfrom()	recvfrom()	recvfrom()
Close connection	close(socket)	close(socket)	close(socket)

Fig.1. Comparison between ipv4 application, ipv6 application and protocol-independent application.

A. IPv6 Compatibility with IPv4

Dual-stack sockets always require IPv6 addresses. The ability to interact with an IPv4 address requires the use of the IPv4-mapped IPv6 address format. Any IPv4 addresses must be represented in the IPv4-mapped IPv6 address format which enables an IPv6 only application to communicate with an IPv4 node. The IPv4-mapped IPv6 address format allows the IPv4 address of an IPv4 node to be represented as an IPv6 address. The IPv4 address is encoded into the low-order 32 bits of the IPv6 address, and the high-order 96 bits hold the fixed prefix 0:0:0:0:0:FFFF.

If the underlying protocol is actually IPv4, then the IPv4 address is mapped into an IPv4-mapped IPv6 address format. The family field in the SOCKADDR structure indicates AF_INET6, but an IPv4-mapped IPv6 address is encoded in the IPv6 address structure. For a dual-stack socket in listening mode, this means that any accepted IPv4 connections will return an IPv4-mapped IPv6 address. For a dual-stack socket that is connecting to an IPv4 destination, the SOCKADDR structure passed to connect must be an IPv4-mapped IPv6 address. Applications must take care to handle these IPv4-mapped IPv6 addresses appropriately and only use them with dual stack sockets. If an IP address is to be passed to a regular IPv4 address, the address must be a regular IPv4 address not an IPv4-mapped IPv6 address.

III. Analysis

There are 16 communication combinations between the client and the server with the IPv4 and IPv6 protocol under the client-server model as shown in table I [3].

Pure IPv6 client can't be connected to IPv4 server which has double stack. As clients are running out of IPv4 addresses, it is required to use IPv6 in future. In order to make pure IPv6 clients to share with IPv4 resources, IPv4 application needs to be migrated. The direct way to migrate IPv4 application is to upgrade it to the application that supports IPv6. There are two ways:

1. The first way is to upgrade it to the pure IPv6 application.
2. The second way is to upgrade it to the application that has support for both IPv4 and IPv6.

Pure IPv6 application running on dual stack host can communicate with IPv4 and IPv6 network separately. But pure IPv6 application communicates with IPv4 network using IPv4-mapped IPv6 address. If both IPv4 and IPv6 addresses are configured on the host, the difficulty of administration of networks also increases; it is not the ideal migration Strategy. The second method is ideal migration strategy as it provides IPv6 service as well as IPv4 service.

By default, an IPv6 socket created on Windows Vista and later only operates over the IPv6 protocol. In order to make an IPv6 socket into a dual-stack socket, the setsockopt function must be called with the IPV6 V6ONLY socket option to set this value to zero before the socket is bound to an IP address. When the IPV6 V6ONLY socket option is set to zero, a socket created for the AF INET6 address family can be used to send and receive packets to and from an IPv6 address or an IPv4 mapped address.

Once you have created the socket and set the appropriate option, the socket can be used to accept incoming IPv4/IPv6 connections or connect to an IPv4/IPv6 destination.

The major difference across systems is whether IPV6_V6ONLY is a) available, and b) turned on or off by default. It is turned off by default on Linux (i.e. allowing dual-stack sockets without setsockopt), and is turned on most Other systems [4].

In addition, the IPv6 stack on Windows XP doesn't support that option. In order to support both IPv4 and IPv6 on Windows XP with Service Pack 1 (SP1) and on Windows Server 2003, an application has to create two sockets, one socket for use with IPv4 and one socket for use with IPv6.

These two sockets must be handled separately by the application. When the server is IPv4 and an IPv6 client tries to communicate with it, the connection is only possible if the client address is an IPv4-mapped into IPv6 address. In this case, if the client chooses a pure IPv6 address, the server will not able to manage the client address.

TABLE I: IPV4 AND IPV6 COMBINATION OF CLIENT SERVER

	IPv4 Server / IPv4 Host	**IPv6 Server / IPv6 Host**	**IPv4 Server / Dual Stack**	**IPv6 Server / Dual Stack**
IPv4 Client / IPv4 Host	IPv4	no	IPv4	IPv4
IPv6 Client / IPv6 Host	no	IPv6	no	IPv6
IPv4 Client / Dual Stack	IPv4	no	IPv4	IPv4
IPv6 Client / Dual Stack	IPv4	IPv6	no / *	IPv6

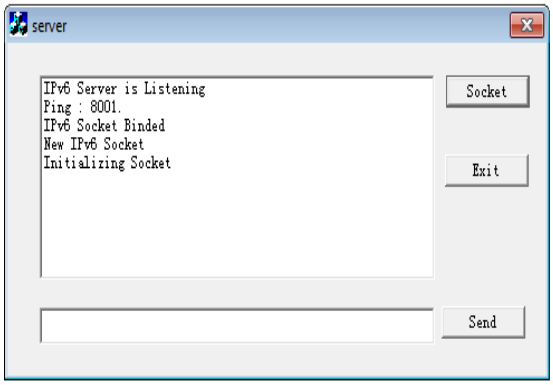

Fig. 2. IPv6 server.

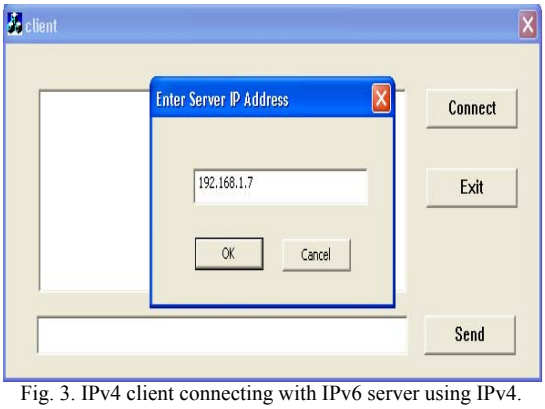

Fig. 3. IPv4 client connecting with IPv6 server using IPv4.

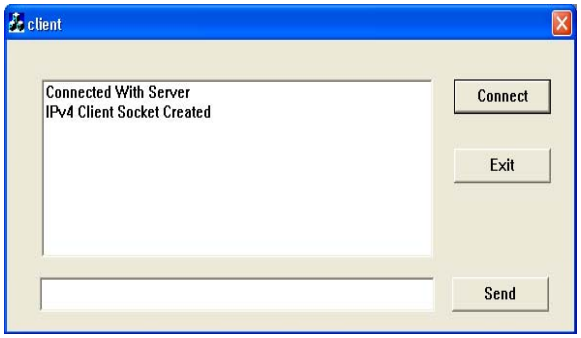

Fig. 4. IPv4 client and IPv6 server ready to communicate.

IV. Result of Implementation

To verify the Table I, a Client-Server Chat Application is realized using Windows Socket 2 which supports IPv4 and IPv6. The testing is done on Windows 7 Operating System. The IPv6 Server is started on double stack as shown in Fig. 2. The IPv4 Client either running on IPv4 host or double stack connects to IPv6 Server running on double stack by entering the server IP address as shown in Fig. 3. The IPv4 Client and IPv6 Server are ready to communicate as shown in Fig. 4.

V. Conclusion

IPv6 as the successor to the two decades old IPv4 has been designed to be a evolutionary protocol. IPv6 will solve IPv4 address shortage problem besides adding new functionality. The deployment of this new protocol brings set of new challenges to the Application Developers and Application Service Providers. Besides having new reliable and stable network to carry the IPv6 traffic all the IPv4 application need to ported to be able to be used in the new environment.

References

[1] C. Yuan-hang and H. Lijuan, "Transition from IPv4 to IPv6 Based on Socket applications," in *proc. International Conference on Networking and Digital Society*, pp. 40-43, 2009.
[2] K. Ettikan and T. W. Chong, Portability Issues for IPv4 to IPv6Applications.
[3] Y. H. Wang and Y. L. Xing, "Transition of Socket applications from IPv4 to IPv6," in *proc. 2nd International Conference on Computer Engineering and Technology*, vol. 6, pp. 75-78, 2010.
[4] Dual-Stack Sockets for IPv6 Winsock Applications. [online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/bb513665%28v=vs.85%29.aspx.