Windows Socket Programming & IPv6 Translation Middleware

Dr. Whai-En Chen

VoIP and IPv6 Laboratory
Research Assistant Professor
Dept. of Computer Science and Information Engineering
National Chiao Tung University
Email: wechen@mail.nctu.edu.tw
TEL: +886-3-5731924
Outline

- Introduction to Socket/WinSock Programming
- IPv4 WinSock Programming
- IPv6 WinSock Programming
- IPv6 Translation Middleware- Socket-layer Translator
- Conclusions
Introduction

- What is Windows Sockets?
  - An Open Interface for Network Programming under Microsoft Windows

- What are its Benefits?
  - an open standard
  - source code portability
  - support dynamic linking

- What is its Future?
  - WinSock 2
Windows Sockets

Standard applications using standard interfaces to access standard services.
BSD Socket APIs

accept()  bind()  closesocket()  connect()
getpeername()  getsockname()  getsockopt()  htonl()
htons()  inet_addr()  inet_ntoa()  ioctlsocket()
listen()  ntohl()  ntohs()  recv()
recvfrom()  select()  send()  sendto()
setsockopt()  shutdown()  socket()
gethostname()
gethostbyname()  gethostbyaddr()
getprotobynumber()  getprotobynumber()
getservbyname()  getservbyport()
Winsock APIs

WSAAsyncGetHostByAddr()  WSAAsyncGetHostByName()
WSAAsyncGetProtoByName()  WSAAsyncGetProtoByNumber()
WSAAsyncGetServByName()  WSAAsyncGetServByPort()
WSAAsyncSelect()  WSACancelAsyncRequest()
WSACancelBlockingCall()  WSACleanup()
WSAGetLastError()  WSAIsBlocking()
WSASetBlockingHook()  WSASetLastError()
WSAStartup()  WSAUnhookBlockingHook()
Windows Sockets 2.0 Architecture

WinSock 2 API

WinSock 2 Application

Transport Functions

Name Space Functions

The WinSock 2 DLL

WS2_32.DLL (32 bit)

WinSock 2 Transport SPI

Transport Service Provider

Transport Service Provider

Name Space Service Provider

Name Space Service Provider

WinSock 2 Name Space SPI
Compatibility of Winsock

WinSock 2 Application

WinSock 1.1 Application

WinSock 1.1 API

WINSOCK.DLL (16 bit)
WSOCK32.DLL (32 bit)

WinSock 2 API

WS2_32.DLL (32 bit)

WinSock 2 SPI

TCP/IP
Transport
Service Provider

TCP/IP-based
Namespace
Service Provider
e.g. DNS
Winsock and OSI Model

The TCP/IP protocol suite compared to the OSI model and Windows Sockets model.
Client/Server Model

- Client-Server Model

- Client and Server Association
  - protocol (same for both Client and server sockets)
  - client IP address
  - client port number
  - server IP address
  - server port number
Client/Server Programming(1)

TCP Client

- socket
- connect
- send
- recv
- close/closesocket

TCP Server

- socket
- bind
- listen
- accept
- recv
- send
- close/closesocket
Client/Server Programming(2)

UDP Client

socket

sendto

recvfrom

close/closesocket

UDP Server

socket

bind

recvfrom

sendto

close/closesocket
IPv4 Socket Programming
Network Program Sketch

- Open a socket
- Name the socket
- Associate with another socket
- Send and receive between sockets
- Close the socket
Open a Socket

socket()

To open a socket you call the socket() function

```pascal
SOCKET PASCAL\textsuperscript{1} FAR socket (int af, int type, int protocol); /* protocol suite */ /* protocol type */ /* protocol name */
```

- **af**: "address family," otherwise known as the socket domain
- **type**: socket type
- **protocol**: the protocol to use
Name the Socket

- What’s in a Socket Name?
  - protocol, port number and IP address

- bind()

  ```
  int PASCAL FAR bind ( SOCKET s,    /*an unbound socket */
  struct sockaddr FAR *addr,            /*local port and IP addr */
  int namelen);                                   /*addr structure length*/
  ```

  **S**: socket handle
  **addr**: pointer to a socket address structure
  (always a sockaddr_in data structure for TCP/IP)
  **namelen**: length of socket structure pointed to by addr
  (always 4 for TCP/IP)
Name the Socket

- **sockaddr Structure**

```c
struct sockaddr
{
    u_short sa_family; // address family
    char  sa_data[14]; // undefined
};
```

- `sa_family`: address family
- `sa_data`: address structure data area defined according to address family value
Name the Socket

- sockaddr_in Structure

```c
struct sockaddr_in {
    short sin_family; /* address family (PF_INET) */
    u_short sin_port; /* port (service) number */
    struct in_addr sin_addr; /* IP address (32-bit) */
    char sin_zero[8]; /* <unused filler> */
};
```

- `sin_family`: address family
- `sin_port`: 16-bit port number in network order
- `sin_addr`: 32-bit Internet address in network order
Associate with Another Socket

- Protocol (same for both client and server sockets)
- client IP address
- client port number
- server IP address
- server port number
Associate with Another Socket

After the association is completed, the client and server know the socket name of their peer. The combination of the two socket names defines the association.
Associate with Another Socket

How a Server Prepares for an Association

`listen()`

```pascal
int PASCAL FAR listen ( SOCKET s, /* a named, unconnected
                      socket */
                      int  backlog) ;  /* pending connect queue
                               length  */

s:      socket handle to a named socket ( bind() called),
but not yet connected

backlog: length of the pending connection queue ( not the
         same as the number of accepted connections)
```
Associate with Another Socket

- How a Client Initiate an Association

connect()

```c
int PASCAL FAR connect (SOCKET s, /*an unconnected socket */
struct sockaddr FAR *addr, /*remote port and IP addr */
int namelen ); /* addr structure length */
```

- `s`: socket handle
- `addr`: pointer to socket address structure (always a `sockaddr_in` structure for TCP/IP)
- `namelen`: length of structure pointed to by `addr` (always 4 for TCP/IP)
Associate with Another Socket

- How a Server Completes an Association

```pascal
accept()

SOCKET PASCAL FAR accept (SOCKET s, /*a listening socket*/
struct sockaddr FAR *addr, /*name of incoming
socket*/
int FAR *addrlen);
```

- **s:** socket handle
- **addr:** pointer to socket address structure (always a
  sockaddr_in structure for TCP/IP)
- **addrlen:** length of socket structure that addr points to
  (always 4 for TCP/IP)
Send and Receiver between Sockets

Sending Data on a “Connected” Socket

send()

```pascal
int PASCAL FAR send (SOCKET s,               /*associated socket*/
                     const char FAR *buf,         /*buffer with outgoing data*/
                     int len,                   /*bytes to send*/
                     int flags );               /*option flags*/
```

- **s**: socket handle
- **buf**: pointer to a buffer that contains application data to send
- **len**: length of data (in bytes) to send
- **flags**: flags to affect the send (MSG_OOB, MSG_DONTROUTE)
Send and Receiver between Sockets

Sending Data on an “Unconnected” Socket

`sendto()`

```c
int PASCAL FAR sendto (SOCKET s,  / * a valid socket */
    const char FAR *buf,  / * buffer with outgoing data */
    int len,  / * bytes to send */
    int flags,  / * option flags */
    struct sockaddr FAR *to,  / * remote socket name */
    int tolen );  / * length of sockaddr */
```

**to:** pointer to socket structure (always a sockaddr_in for TCP/IP) that contains destination address and port number (socket name)

**tolen:** length of socket structure pointed to by to (always 4 for TCP/IP)
Send and Receiver between Sockets

- Receiving Data

recv()

```pascal
int PASCAL FAR recv (SOCKET s,               /*associated socket*/
                     char FAR *buf,                        /*buffer with outgoing data*/
                     int len,                                    /*bytes to send */
                     int flags );                               /*option flags */
```

recvform()

```pascal
int PASCAL FAR recvform (SOCKET s,               /*a valid socket*/
                         char FAR *buf,                        /*buffer with outgoing data*/
                         int len,                                    /*bytes to send */
                         int flags );                               /*option flags */
struct sockaddr FAR *from,    /*remote socket name */
                         int fromlen );                          /*length of sockaddr */
```
Send and Receiver between Sockets

s: socket handle
buf: pointer to a buffer that contains application data to send
len: length of data (in bytes) to send
flags: flags to affect the send (MSG_OOB, MSG_DONTROUTE)
from: pointer to socket structure (always a sockaddr_in for TCP/IP) that contains source address and port number (socket name)
fromlen: length of socket structure pointed to by from (always 4 for TCP/IP)
Other Useful Socket Functions

- **Byte Ordering Functions**
  - ntohs(), ntohl()
  - htons(), htonl()

- **Address Translation Functions**
  - inet_addr()- 將字串轉成32位元的IP位址
  - inet_nota()- 將32位元的IP位址轉成字串

- **Name Resolution**
  - gethostbyaddr()-利用 host 的位址來獲取該 host 的資料
  - gethostbyname()-利用 host 的名稱來獲取該 host 的資料
  - 傳回hostent的資料結構

- **WSAStartup() and WSACleanup()**
struct hostent {
    char FAR * h_name;
    char FAR * FAR * h_aliases;
    short h_addrtype;
    short h_length;
    char FAR * FAR * h_addr_list;
}

- 是一個 linked-list
Byte Ordering Function

Increasing memory address

Little-endian byte order:

<table>
<thead>
<tr>
<th>Address A+1</th>
<th>Address A</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-order byte</td>
<td>low-order byte</td>
</tr>
<tr>
<td>MSB</td>
<td>16bit value</td>
</tr>
</tbody>
</table>

Big-endian byte order:

<table>
<thead>
<tr>
<th>Address A</th>
<th>Address A+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-order byte</td>
<td>low-order byte</td>
</tr>
</tbody>
</table>

Increasing memory address
IPv4 Example for Daytime Server (Connection-oriented)

```c
int main(int argc, char **argv) {
    int listenfd, connfd;
    struct sockaddr_in servaddr;
    char buff[MAXLINE];
    time_t ticks;
    listenfd =
    socket(AF_INET, SOCK_STREAM, 0);
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(13);
    /* daytime server */

    bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
}
```
IPv4 Example for Daytime Server (Connection-oriented)

```
listen(listenfd, LISTENQ);

for ( ; ; ) {
    connfd = accept(listenfd, (SA *) NULL, NULL);

    ticks = time(NULL);
    snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
    write(connfd, buff, strlen(buff));

    Close(connfd);
}
```
int main(int argc, char **argv)
{
    int sockfd, n;
    char recvline[MAXLINE + 1];
    struct sockaddr_in servaddr;
    if (argc != 2)
        err_quit("usage: a.out <IPaddress>");
    if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        err_sys("socket error");
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(13);
    /* daytime server */
    if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
        err_quit("inet_pton error for %s", argv[1]);
IPv4 Example for Daytime Client (Connection-oriented)

```c
if (connect(sockfd, (SA *) &servaddr, sizeof(servaddr)) < 0)
    err_sys("connect error");

while ( (n = read(sockfd, recvline, MAXLINE)) > 0) {
    recvline[n] = 0;
    /* null terminate */
    if (fputs(recvline, stdout) == EOF)
        err_sys("fputs error");
}
if (n < 0)
    err_sys("read error");
exit(0);
```
提供轉換IPv4程式到IPv6之方法

- 介紹IPv4與IPv6之長度不同
- 介紹為何需要改變應用程式
- 介紹不用轉換的Socket API
- 介紹需要轉換的Socket API
- 介紹需要轉換的資料結構
IPv4/IPv6位址長度不同

- 數字位址
  - IPv4, 32位元位址長度
  - IPv6, 128位元位址長度

IPv4, 32 bits

IPv6, 128 bits
New Solutions for Applications

IPv4 AP  IPv6 AP
TCP/UDP  TCP/UDPv6
IPv4      IPv6
Layer 1 and 2

V4/v6 Protocol-independent Application
TCP/UDP  TCP/UDPv6
IPv4      IPv6
Layer 1 and 2

IPv4 AP  IPv6 AP
TCP/UDP  TCP/UDPv6
IPv4      IPv6
Layer 1 and 2
不需要轉換的Socket API (依序)

- **Server端的程式碼**
  - socket  open a socket
  - bind  bind local address to the socket
  - listen  listen on a port
  - accept  wait for the connection
  - read/write  if TCP
  - recvfrom/sendto  if UDP

- **Client端的程式碼**
  - socket  open a socket
  - connect  connect to a server
  - read/write  if TCP
  - recvfrom/sendto  if UDP
轉換需要改變的部分

- 與IP位址相關的Socket API與參數需要修改
- 程式部分有運用到IP位址的部分
  - 位址轉換函式
  - 位址複製函式
  - 位址比較函式
  - 位址相關之記憶體指派與變數宣告

IPv4程式設計者的自訂的函式與變數也需要修改
API與資料結構的轉換

- Socket參數名稱轉換

<table>
<thead>
<tr>
<th>IPv4</th>
<th>IPv6</th>
</tr>
</thead>
<tbody>
<tr>
<td>AF_INET</td>
<td>AF_INET6</td>
</tr>
<tr>
<td>PF_INET</td>
<td>PF_INET6</td>
</tr>
<tr>
<td>IN_ADDR_ANY</td>
<td>inaddr6_any</td>
</tr>
</tbody>
</table>
## API與資料結構的轉換

### 資料結構轉換

<table>
<thead>
<tr>
<th>IPv4</th>
<th>IPv6</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_addr</td>
<td>in6_addr</td>
</tr>
<tr>
<td>sockaddr</td>
<td>sockaddr_in6</td>
</tr>
<tr>
<td>sockaddr_in</td>
<td>sockaddr_in6</td>
</tr>
</tbody>
</table>
IPv4 Socket Address Structure

```
struct in_addr {
    in_addr_t s_addr; /* 32bit IPv4 address */
}; /* network byte ordered */

struct sockaddr_in {
    uint8_t sin_len; /* length of structure(16) */
    sa_family_t sin_family; /* AF_INET */
    in_port_t sin_port; /* 16bit TCP or UDP port number */
    struct in_addr sin_addr; /* 32bit IPv4 address */
    char sin_zero[8]; /* unused */
}; /* included in <netinet/in.h> */
```
IPv6 Socket Address Structure

Struct in6_addr{
    uint8_t    s6_addr[16];    /*128bit IPv6 address*/
    /*network byte ordered*/
};

#define SIN6_LEN                     /* required for compile-time tests */

struct sockaddr_in6 {
    uint8_t    sin6_len;    /* length of structure(24) */
    /* AF_INET6*/
    sa_family_t sin6_family;    /* Transport layer port# */
    /* network byte ordered*/
    in_port_t   sin6_port;    /* priority & flow label */
    /* network byte ordered*/
    uint32_t   sin6_flowinfo;    /* IPv6 address */
    /* network byte ordered*/
    struct in6_addr   sin6_addr;    /* network byte ordered*/
}; /* included in <netinet/in.h> */
## API與資料結構的轉換

### 資料結構參數轉換

<table>
<thead>
<tr>
<th>IPv4</th>
<th>IPv6</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin_len</td>
<td>sin6_len</td>
</tr>
<tr>
<td>sin_family</td>
<td>sin6_family</td>
</tr>
<tr>
<td>sin_port</td>
<td>sin6_port</td>
</tr>
<tr>
<td>sin_addr</td>
<td>sin6_addr</td>
</tr>
<tr>
<td>s_addr</td>
<td>s6_addr</td>
</tr>
</tbody>
</table>
## API與資料結構的轉換

### 函式轉換

<table>
<thead>
<tr>
<th></th>
<th>IPv4</th>
<th>IPv6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name-to_address</strong></td>
<td><strong>inet_aton()</strong></td>
<td><strong>inet_pton()</strong></td>
</tr>
<tr>
<td><strong>Functions</strong></td>
<td><strong>inet_addr()</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>inet_ntoa()</strong></td>
<td><strong>inet_ntop()</strong></td>
</tr>
<tr>
<td><strong>Address conversion</strong></td>
<td><strong>gethostbyname()</strong></td>
<td><strong>getipnodebyname()</strong></td>
</tr>
<tr>
<td><strong>Functions</strong></td>
<td><strong>gethostbyaddr()</strong></td>
<td><strong>getipnodebyaddr()</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>getnameinfo()</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>getaddrinfo()</strong></td>
</tr>
</tbody>
</table>
Data Structure Comparison

- AF independent
  - struct sockaddr
- IPv4 dependent
  - struct in_addr
  - struct sockaddr_in
- Name resolving
  - struct hostent

- AF independent
  - struct sockaddr_storage
- IPv6 dependent
  - struct in6_addr
  - struct sockaddr_in6
- Name resolving
  - struct addrinfo

IPv4

IPv6
Definitions and Function Calls

- Address Family & Protocol Family
  - AF_INET6 & PF_INET6 for IPv6
- No changes to transport socket APIs
  - `socket()`, `connect()`, `bind()`……
- Name resolving
  - AF dependent functions are obsolete
  - New AF independent functions
  - `gethostbyname()` and `gethostbyaddr()` - IPv4-only
  - `getaddrinfo()` and `getnameinfo()` - IPv4 & IPv6
getaddrinfo() & getnameinfo()

- Convert strings storing address and service into sockaddr structure
  - `getaddrinfo("www.kame.net","www",&hint,&res);

- Options are specified in hint
  - `hint` is an addrinfo structure

- Results are returned as a linked-list, each list node contains a sockaddr structure

- `freeaddrinfo()` to free returned linked-list
  - `freeaddrinfo(res);

- `getnameinfo()` converts from sockaddr into strings storing address and service
  - `getnameinfo(sa,name,sizeof(name),srv,sizeof(srv),0);`
Introduction to Checkv4.exe

- Provided by Microsoft
- Identifies potential problems in codes and makes recommendations
- Identifies most trivial problems
  - Successfully checks presence of IPv4 specified code. e.g. gethostbyname(), struct sockaddr_in, and so on.
- Gives some false alert
  - Identifies parameters in comment
- Results from Checkv4.exe
  - About 200 lines for CCL/ITRI SkinUA
Checkv4.exe (Partial Results)

D:\SIP\src\low>checkv4 *.c
checkv4.c:40: sockaddr_in: use sockaddr_storage instead, or use sockaddr_in6 in addition for IPv6 support
checkv4.c:64: PF_INET: use PF_INET6 in addition for IPv6 support
checkv4.c:127: hostent: use addrinfo instead
checkv4.c:133: AF_INET: use AF_INET6 in addition for IPv6 support
checkv4.c:137: INADDR_ANY: use getaddrinfo with nodename=NULL and AI_PASSIVE instead, or use in6addr_any in addition for IPv6 support
checkv4.c:139: inet_addr: use WSAStringToAddress or getaddrinfo with AI_NUMERIC_IHOST instead
checkv4.c:140: inet_addr: use WSAStringToAddress or getaddrinfo with AI_NUMERIC_IHOST instead
checkv4.c:141: gethostbyname: use getaddrinfo instead
checkv4.c:149: inet_ntoa: use WSAAddressToString or getnameinfo with NI_NUMERIC_IHOST instead
checkv4.c:203: sockaddr_in: use sockaddr_storage instead, or use sockaddr_in6 in addition for IPv6 support
checkv4.c:242: PF_INET: use PF_INET6 in addition for IPv6 support
checkv4.c:244: PF_INET: use PF_INET6 in addition for IPv6 support
checkv4.c:377: sockaddr_in: use sockaddr_storage instead, or use sockaddr_in6 in addition for IPv6 support
checkv4.c:418: inet_ntoa: use WSAAddressToString or getnameinfo with NI_NUMERIC_IHOST instead
Comparison of socket address structure

**IPv4**
- `sockaddr_in()`
  - length
  - AF_INET
  - 16-bit port#
  - 32-bit IPv4 address
  - (unused)
- Fixed length (16 bytes)
- Figure 3.1

**IPv6**
- `sockaddr_in6()`
  - length
  - AF_INET6
  - 16-bit port#
  - 32-bit flow label
- 128-bit IPv6 address
- Fixed length (24 bytes)
- Figure 3.4

**Unix**
- `sockaddr_un()`
  - length
  - AF_LOCAL
  - path name
  - (up to 104 bytes)
- Fixed length (up to 104 bytes)
- Figure 3.5

**Datalink**
- `sockaddr_dl()`
  - length
  - AF_LINK
  - interface index
  - type
  - name len
  - addr len
  - sel len
  - interface name
  - link-layer address
- Variable length
- Figure 17.1

---

Figure 3.5 Comparison of various socket address structures.
Socket address structure pass.

bind, connect, sendto

accept, recvfrom, getsockname, getpeername
Figure 3.10 Summary of address conversion functions.
NTPO&CCL SIP User Agent (UA)

- SIP-based VoIP phone running on Windows
- Support H.263 Video codec
- Support G.711u/G.711a/G.723/G.729 Audio codec
- Support registration
- Support authentication
Structure of SIP UA

- UI
  - UAPhone
  - CallManager
  - MediaManager
    - UACore
    - Codec
    - SDP
    - sipTx
    - RTP
    - SIP
    - Transport

Component Relationship of CallManager

UI

Function Call

Windows Event

CallManager

Function Call

Callback Function

UACore

Function Call

Function Call

MediaManager
GUI Problem

- IP Address control
  - Is IPv4 specified
  - Do not accept domain name
- Use Edit control instead
Get Local Address (1/2)

- Old method: gethostbyname()
  - gethostbyname() on local hostname
- Does getaddrinfo() on local hostname works?
  - Not works on Windows XP
  - Works on Windows 2003
Get Local Address (2 of 2)

- Make use of **IPHelper** functions
  - Presented in Windows from Windows 98
  - A Windows-only solution
  - Works on both windows XP and 2003
- Function name: **GetAdaptersAddresses()**
Parsing URI with IPv6

- IPv6 address in URI
  - `sip:wechen@[3ffe:1345:5643::3]:5060`
- Some parser assume semicolon will be used only to separate IP and Port
- Modify parsing algorithm to deal with IPv6 address.
- URI in SIP header may contains IPv6 address
  - `INVITE sip:wechen@[2001:238:f82:66::33]:5060`
- IP6 addrtype & IPv6 address in SDP
  - `c=IN IP6 FE80:60::2`
Goal of Porting SIP UA to IPv6

- Provide IPv6 communication to Users (a long-term solution)
- SIP UA should accept SIP URI that contains IPv6 literal address (specified in RFC 3261)
- SIP UA should correctly handle IPv6 addresses in SIP/SDP header fields
- SIP UA should operate with other IPv6 SIP UAs (KPhone and LinPhone) and SIP servers (IPtel and Partysip).
Modifications for SIP User Agent

- Auto IPv4/IPv6 negotiation requires modification in listening thread part and rewrite working flow of calling
  - The IP version is the same as the IP address that user choose
  - SIP UA will use either IPv4 or IPv6 at the same time.
  - Lower part in protocol stack should check an extra parameter that specifies address family
Modifications for SIP User Agent (cont.)

- IPv6 address Literal format has scope-id
  - E.g. fe80::201:2ff:fe85:37ed%3
  - Used by linked-local address
  - Identify the same address on different interface

- Scope-id must be specified when connecting to sites using link-local address
  - An extra parameter in data structure to keep this
Modifications for SIP User Agent (cont.)

- SIP URI may contain IPv6 address
  - E.g. sip:wechen@[2001:238:f82:6::2]:5060
  - Rewrite parser to ensure correctly dealing with colon

- Since IPv6 address are longer than IPv4 address, GUI components related to address should be modified

- Avoid using IPAddressControl that supports IPv4 address only
Results

- Changes 500+ out of 100,000+ lines in 150 files
- About 300 lines are not identified by checkv4.exe
- SIP UA supports
  - IPv4 or IPv6 communication
  - IPv6 address in SIP URI
  - IPv6 address in GUI and form
- Modifications in SIP UA
  - Transport – handle different IP versions
  - GUI – handle IPv6 address
  - CallManager – URI parsing/generating
## Modification Summary

<table>
<thead>
<tr>
<th>Module name</th>
<th>Modified files</th>
</tr>
</thead>
<tbody>
<tr>
<td>UA Core</td>
<td>5</td>
</tr>
<tr>
<td>sipTX</td>
<td>4</td>
</tr>
<tr>
<td>sip</td>
<td>5</td>
</tr>
<tr>
<td>sdp</td>
<td>1</td>
</tr>
<tr>
<td>rtp</td>
<td>6</td>
</tr>
<tr>
<td>transport</td>
<td>6</td>
</tr>
<tr>
<td>cc1RTP</td>
<td>2</td>
</tr>
<tr>
<td>MediaManager</td>
<td>4</td>
</tr>
<tr>
<td>UI</td>
<td>4</td>
</tr>
<tr>
<td>UA Profile</td>
<td>2</td>
</tr>
</tbody>
</table>

Total: 39 files
啟動SIPv6 User Agent

1. Double-click
2. Click right button
3. Preferences
4. Next Page

SkinUAd.exe
SkinUA MFC Application
設定SIPv6 User Agent的IPv6位址

1. 選擇「User Settings」分頁
3. 如果要跨越IPv4網路，則需要選擇6to4位址（Prefix是2002::/16）
設定SIPv6 User Agent的伺服器

1. 選擇「Server Settings」分頁
2. 取消「Use Proxy」選項
3. 取消「Registration」；若是有IPv6 SIP伺服器，則可以選取選項，並填入伺服器的IPv6位址
設定SIPv6 User Agent的Codec參數

1. 選擇「Codec Settings」分頁
2. 將要用的Codec放入「Active Codecs」選項中
3. 選取「Use Video」，若不需要影像則可以取消此選項
4. 按下「確定」按鈕，完成設定

Next Page
開始撥號 (輸入SIP URI)

1. 按下圖中按鈕
2. 可以直接輸入SIP URI (如：SIP:7221@3ffe:3600:1::1)
3. 或是可以按下「List」按鈕，從選單中選取
4-6. 按下「Load」按鈕，選取SIP URI，按下「OK」完成

SIP URI (SIP:7221@3ffe:3600:1::1)
撥號與接聽

1. 按下「Dial」按鈕，開始撥號
2. 受話方案下圖中電話筒圖案即可接聽
展示項目- SIPv6 User Agent (UA) 移植成果

1. 設定
2. 撥號
3.1 INVITE
3.2 INVITE
3.3 INVITE
3.4 200 OK
3.5 200 OK
3.6 200 OK
3.7 ACK
3.8 ACK
3.9 ACK
4. RTP

4.通訊影像

圖例:
- SIP Signaling (IPv6)
- SIP Signaling (Tunnel)

IPv6 Network (Showroom)
IPv6 Network (NCTU VoIP Lab)
Internet (IPv4)
Dual-stack Router
SIPv6 UA
Dual-stack Router
SIPv6 UA

2004/12/24
Speaker: Whai-En Chen
IPv6 Address
IPv6 address
Interoperability Testing

- Testing with 2 Linux SIP-based phone
  - Kphone 3.2 with IPv6 (patched by iptel)
  - Linphone 0.11.3 (claimed as IPv6 enabled)
- Environment
  - Windows XP SP1
  - Redhat linux 9.0
  - Partysip IPv6 SIP proxy
  - Iptel IPv6-enabled SIP server
Interoperability Testing Results

- To IPv6 SIP proxy

<table>
<thead>
<tr>
<th>Item</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register on iptel</td>
<td>Succeed</td>
</tr>
<tr>
<td>Register on partysip</td>
<td>Succeed</td>
</tr>
<tr>
<td>Call UA through partysip proxy server</td>
<td>Succeed</td>
</tr>
</tbody>
</table>
# Interoperability Testing Results

## To IPv6 SIP UA

<table>
<thead>
<tr>
<th>From</th>
<th>Kphone</th>
<th>Linphone</th>
<th>SkinUA</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPhone</td>
<td>OK</td>
<td>SIP ok</td>
<td>SIP ok</td>
</tr>
<tr>
<td>Linphone</td>
<td>SIP ok</td>
<td>OK</td>
<td>SIP ok</td>
</tr>
<tr>
<td>SkinUA</td>
<td>OK</td>
<td>SIP ok</td>
<td>OK</td>
</tr>
</tbody>
</table>

*Linphone & KPhone can not accept URI containing IPv6 Literal address in URI.*
IPv6 Translation Mechanism-
Bump-In-the-API
設計主機端轉換之中介軟體

- 可是要將應用程式升級成IPv6會有以下問題
  - 需要改用新的 API
  - 需要改用新的 Data structure

- 以SIP-based VoIP User Agent為例
  - 約有200行Socket API、資料結構需要轉換
  - 共約有600行位址相關函式、變數、記憶體指派需要修改

- 短期內將程式升級IPv6不容易
  - 需要改的函式、變數需要追蹤修訂
  - 程式版本升級時，亦需隨之修訂

- 提出一個轉換v4/v6的中介軟體，以BIA為基礎，設計應用層轉換機制
軟硬體來源與執行平台

- **BIA轉換器元件**
  - Function Mapper
  - Name Resolver
  - Address Mapper
  - ALG Manager
  - FTP-ALG

- **BIA轉換器的開發平台如下**
  - 作業系統: Windows XP SP1
  - 中央處理器: Intel Celeron 2GHz
  - 記憶體: 128 MB
  - 硬碟: 20GB
  - 編譯程式: Microsoft Visual C++ 6.0
  - 開發函式庫: Microsoft Platform SDK February 2003

- **BIA可以執行於微軟Windows XP/2003之上**
Name Resolving:
Translate IPv6 address to IPv4 address

C:\WINDOWS\system32>nrtest www.kame.net
he's hostname:www.kame.net
it's alias names:
addrtype is 2
addr length is 4
10.128.128.127
10.128.128.126
Socket-layer Translator Result

KAME Project

dancing kame by atelier argonage

KAME Project is a joint effort of six companies in Japan
to provide a free IPv6 and IPv4 stack and the stack for BSD variants to the world.

- Newsflash! NEW
  - August 4, 2003: A SNAP kit was generated. For the complete list of changes, check here.
  - Past newsflash

- Latest releases
  - KAME SNAP (weekly): August 4, 2003 NEW
    - List of mirrors services
    - Supported platforms: BSD/OS 3.1 (M), BSD/OS 4.3, FreeBSD 4.8 RELEASE, NetBSD 1.6.1, and OpenBSD 3.3.
  - Platforms with KAME code merged in:
    - FreeBSD 4.0 and beyond
    - OpenBSD 2.7 and beyond
    - NetBSD 1.5 and beyond
    - BSD/OS 4.2 and beyond
  - What is the difference between KAME kit and KAME integrated *BSD release?

- Related software
  - IPv6 enabled software and patches
  - more about our IPv6 enabled software and patch collections
Using IPv4 to Browse Without Socket-layer Translator

KAME Project

If you migrate to IPv6, you will be able to view the default home.

KAME Project is a joint effort of six companies in Japan to provide a free IPv6 and IPv4 router software stack for BSD vendors to the world.

- Newsflash!
  - August 4, 2003: A SNAP kit was generated. For the complete list of changes, check here.
  - Past newsflash

- Latest releases
  - KAME SNAP (weekly): August 4, 2003
    - List of mirror servers
      - Supported platforms: BSD/OS 3.1, FreeBSD 4.3, FreeBSD 4.4 RELEASE, NetBSD 1.6.1, and OpenBSD 3.3.
      - We do not support target file for the platforms marked with "X".
      - Platforms with KAME code assigned to:
        - FreeBSD 4.0 and beyond
        - FreeBSD 2.7 and beyond
        - NetBSD 1.5 and beyond
        - OpenBSD 4.2 and beyond
      - What's the difference between KAME kit and KAME-integrated *BSD released?

- Related software
  - IPv6-enabled software and patches
  - more about our IPv6-enabled software and patch collections
Conclusions

In this course, you can learn the following techniques

- IPv4 Windows Socket Programming
- IPv6 Windows Socket Programming
- IPv4/IPv6 Domain Name Resolution

You can try to do following advanced topics.

- Writing IPv4/IPv6 compatible programs
- Porting IPv4 applications to IPv6 version
- Writing ALG on Socket-layer Translator
- Writing IPv6 Test tools on SIPv6 Analyzer
References

- Microsoft Platform SDK
- MSDN Library
- VC++ 6.0
References

Appendix
IPv4 Header

Figure A.1 Format of the IPv4 header.
IPv6 Header

Figure A.2 Format of the IPv6 header.
### IPv4 Address

**Figure A.3** IPv4 address formats.

<table>
<thead>
<tr>
<th>Class</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.0.0.0 to 127.255.255.255</td>
</tr>
<tr>
<td>B</td>
<td>128.0.0.0 to 191.255.255.255</td>
</tr>
<tr>
<td>C</td>
<td>192.0.0.0 to 223.255.255.255</td>
</tr>
<tr>
<td>D</td>
<td>224.0.0.0 to 239.255.255.255</td>
</tr>
<tr>
<td>E</td>
<td>240.0.0.0 to 247.255.255.255</td>
</tr>
</tbody>
</table>

**Figure A.4** Ranges for the five different classes of IPv4 addresses.
<table>
<thead>
<tr>
<th>Allocation</th>
<th>Format prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>reserved</td>
<td>0000 0000</td>
</tr>
<tr>
<td>unassigned</td>
<td>0000 0001</td>
</tr>
<tr>
<td>reserved for NSAP</td>
<td>0000 001</td>
</tr>
<tr>
<td>reserved for IPX</td>
<td>0000 010</td>
</tr>
<tr>
<td>unassigned</td>
<td>0000 011</td>
</tr>
<tr>
<td>unassigned</td>
<td>0000 1</td>
</tr>
<tr>
<td>unassigned</td>
<td>0001</td>
</tr>
<tr>
<td>aggregatable global unicast</td>
<td>001</td>
</tr>
<tr>
<td>addresses</td>
<td></td>
</tr>
<tr>
<td>unassigned</td>
<td>010</td>
</tr>
<tr>
<td>unassigned</td>
<td>011</td>
</tr>
<tr>
<td>unassigned</td>
<td>100</td>
</tr>
<tr>
<td>unassigned</td>
<td>101</td>
</tr>
<tr>
<td>unassigned</td>
<td>110</td>
</tr>
<tr>
<td>unassigned</td>
<td>1110</td>
</tr>
<tr>
<td>unassigned</td>
<td>1111 0</td>
</tr>
<tr>
<td>unassigned</td>
<td>1111 10</td>
</tr>
<tr>
<td>unassigned</td>
<td>1111 110</td>
</tr>
<tr>
<td>unassigned</td>
<td>1111 1110 0</td>
</tr>
<tr>
<td>link-local unicast address</td>
<td>1111 1110 10</td>
</tr>
<tr>
<td>site-local unicast address</td>
<td>1111 1110 11</td>
</tr>
<tr>
<td>multicast addresses</td>
<td>1111 1111</td>
</tr>
</tbody>
</table>

Figure A.7 Meaning of high-order bits of IPv6 addresses.