IPv6 Porting Applications

US IPv6 Summit
Dec 8-11, 2003
Eva M. Castro – eva@gsyc.escet.urjc.es

Systems and Communications Group (GSyC)
Experimental Sciences and Technology Department (ESCET)
Rey Juan Carlos University (URJC)
Agenda

- Transition Architecture
- Evolution of Applications
- Application Transition Scenarios
- Application Porting Considerations
- BSD Socket API
- IP Version Independent Applications
- Recommendations
Transition Architecture

- Network
- End-point nodes
- Applications

Applications

End-point Node

Network Routing/Addressing

TCP/UDP
IP
Link layer

App

TCP/UDP
IP
Link layer

App

Applications

End-point Node
Network (routing/addressing)

IPv4-only Network

IPv6-only Network

Transition Mechanisms:
Tunnels
Protocol translation

IPv4/IPv6

Dual Network

IPv4/IPv6

Heterogeneous Network
End-point Nodes (IP stack)

IPv4-only node

- TCP/UDP
- IPv4
- Link layer

Dual stack node

- TCP/UDP
- IPv4
- IPv6
- Link layer

IPv6-only node

- TCP/UDP
- IPv6
- Link layer
Applications (source code)

- IPv4-only Applications – Appv4
- IPv6-only Applications – Appv6
- Dual Applications – Dual app
Applications - Dual Nodes

IPv4 addresses: x.y.z.w
IPv6 addresses: x:x:x:x:x:x:x:x
IPv4 addresses: x.y.z.w
IPv6 addresses: x:x:x:x:x:x:x:x
IPv4-mapped IPv6 addresses: ::FFFF:x.y.z.w
IPv4 addresses: x.y.z.w
IPv4 addresses: x.y.z.w

TCP/UDP
IPv4 data
IPv4 header
IPv4
IPv6 header
IPv6
Data link header
Data link
Appv4
Appv6
Appv6
Appv4
TCP/UDP
IPv4 data
IPv4 header
IPv4
IPv6 header
IPv6
Data link header
Data link
Agenda

- Transition Architecture
- Evolution of Applications
- Application Transition Scenarios
- Application Porting Considerations
- BSD Socket API
- IP Version Independent Applications
- Recommendations
Evolution of Applications

Substituting IPv4 calls and structures by IPv6 ones

Appv4 → Appv6
Evolution of Applications

Substituting IPv4 calls and structures by IPv6 ones

ADVANTAGES:
• Easy task, short time

PROBLEMS:
• Application selection
• Source code maintainability

IPv4
TCP/UDP
IPv4
IPv6
Link layer
Evolution of Applications

- Appv4
- Adding IPv6 support
- Dual app
Evolution of Applications

ADVANTAGES:
- One application for all kind of nodes
- Source code maintainability

DISADVANTAGES:
- More changes, long time

Appv4

Adding IPv6 support

Dual app

TCP/UDP
IPv4
IPv6
Link layer
Evolution of Applications

- Substituting IPv4 calls and structures by IPv6 ones
- Adding IPv6 support
- Adding IPv4 support

Appv4 → Appv6

Dual app

December 2003   December 2003   -- Page Page 1313
Evolution of Applications

Gradual transition

Substituting IPv4 calls and structures by IPv6 ones

Adding IPv6 support

Adding IPv4 support

Dual app

Appv4

Appv6
Agenda

- Transition Architecture
- Evolution of Applications
- Application Transition Scenarios
- Application Porting Considerations
- BSD Socket API
- IP Version Independent Applications
- Recommendations
Application Transition Scenarios

From existing IPv4 applications in IPv4 node:

1. IPv4 applications in dual-stack nodes
2. IPv6 applications in dual-stack nodes
3. Dual applications in dual-stack nodes
4. Dual applications in IPv4 nodes
1. IPv4 applications in dual-stack nodes

- IPv4 source code dependencies
- Use IPv4
- If IPv6 communication is required:
  - port source code to IPv6, then applications use native IPv6, or
  - use transition mechanisms. Applications use IPv4, but IPv6 packets are exchanged:
    - **BIA:**
      - IPv4 applications + Dual stack nodes
    - **BIS:**
      - IPv4 applications + IPv4 nodes

Dual stack nodes
BIA – Bump In the API

Application Level

Appv4

BIA Socket Library

Address mapper

BSD sockets interface

Resolver

libc

O.S. Kernel

TCP/UDP

UNIX

IPv4

IPv6

TCP/UDP

UNIX

IPv4

IPv6
BIA - resolver calls

Application Level

Appv4

Resolver calls

BIA Socket Library

Address mapper

libc

BSD sockets interface

Resolver

O.S. Kernel

TCP/UDP

UNIX

IPv4

IPv6
BIA – socket calls

Application Level

Socket calls

Appv4

BIA addr

BIA Socket Library

REAL addr

Address mapper

libc

BSD sockets interface

Resolver

O.S. Kernel

TCP/UDP

UNIX

IPv4

IPv6

REAL addr

IPv6 TASK FORCE

December 2003 - Page 21
BIS - Bump In the Stack

Application Level

Appv4

BSD sockets interface

Resolver

O.S. Kernel

TCP/UDP

UNIX

IPv4

BIS module

IPv6

Address mapper
BIS – Bump In the Stack

Application Level
- Appv4

libc
- BSD sockets interface
- Resolver

O.S. Kernel
- TCP/UDP
- UNIX
- IPv4
- IPv6

BIS module
- Address mapper

REAL addr

December 2003 - Page 23
2. IPv6 applications in dual-stack nodes

- IPv6 source code dependencies
- Use IPv6
- If IPv4 communication is required:
  - change source to dual application, or
  - two different applications: appv4 & appv6, or
  - use IPv4-mapped IPv6 addresses
    - Built from IPv4 addresses

IPv4 addr

IPv4-mapped IPv6 addr
IPv6 Apps – IPv4-mapped IPv6 addr

IPv6 packets

IPv4-mapped IPv6 addr
::FFFF:a.b.c.d

IPv6 packets

IPv4-mapped IPv6 addr
::FFFF:a.b.c.d
## Client/Server interoperability

<table>
<thead>
<tr>
<th>IPv4 client</th>
<th>IPv4 node</th>
<th>Dual stack</th>
<th>IPv4 server</th>
<th>IPv4 node</th>
<th>Dual stack</th>
<th>IPv6 server</th>
<th>IPv6 node</th>
<th>Dual stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dual stack</td>
<td>IPv4</td>
<td>IPv4</td>
<td>Dual stack</td>
<td>IPv4</td>
<td>IPv4</td>
<td>IPv4</td>
<td>IPv6</td>
<td>IPv6</td>
</tr>
</tbody>
</table>
## Client/Server interoperability

<table>
<thead>
<tr>
<th>IPv4 client</th>
<th>IPv4 server</th>
<th>IPv6 server</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4 node</td>
<td>IPv4</td>
<td>IPv4</td>
</tr>
<tr>
<td>Dual stack</td>
<td>IPv4</td>
<td>IPv4</td>
</tr>
<tr>
<td>IPv6 node</td>
<td>Fail</td>
<td>IPv6</td>
</tr>
<tr>
<td>Dual stack</td>
<td>IPv4</td>
<td>IPv6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IPv6 client</th>
<th>IPv4 server</th>
<th>IPv6 server</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4 node</td>
<td>IPv4</td>
<td>IPv4</td>
</tr>
<tr>
<td>Dual stack</td>
<td>IPv4</td>
<td>IPv6</td>
</tr>
<tr>
<td>IPv6 node</td>
<td>Fail</td>
<td>IPv6</td>
</tr>
<tr>
<td>Dual stack</td>
<td>IPv4</td>
<td>IPv6</td>
</tr>
</tbody>
</table>
3. Dual applications in dual-stack nodes

- Applications use both protocols.

- Ways of implementation:
  - Applications request IPv6 addresses related to a host name if not valid, applications request IPv4 addresses.
    IP VERSION DEPENDENT SOURCE CODE
  - Applications request all addresses related to a host name, and try connection with the list of addresses until connection succeeds.
    IP VERSION INDEPENDENT SOURCE CODE
4. Dual applications in IPv4 nodes

- Write dual applications which can also run in IPv4-only nodes:
  - Try first AF_INET6 socket and the AF_INET.
  - If kernel does not have IPv6 support, the socket call will return an error.
  - Try AF_INET even socket call returned an error.
Agenda

- Transition Architecture
- Evolution of Applications
- Application Transition Scenarios
- Application Porting Considerations
- BSD Socket API
- IP Version Independent Applications
- Recommendations
Application Porting Considerations

IP version dependencies in applications:

- Presentation format for an IP address
- Transport layer API
- Name and address resolution
- Specific IP dependencies
1. Presentation format for an IP addr

IP addresses are usually provided in a presentation format, string: “10.0.0.1”

PROBLEMS

• Allocated memory to store the string could be not enough.
• IPv4 uses “.” as separator, IPv6 uses “:”. Parsers should be complaint with both formats.
• Ambiguity using “:” character in URLs: http://[IPv6Address]:portNumber

RECOMMENDATION

• Use FQDN
2. Transport layer API

- Network information storage.
- Address conversion functions.
- Communication API functions.
- Network configuration options.

**PROBLEMS**
- IPv4 Network API makes visible IP version dependencies, functions and structures must be changed

**RECOMMENDATION**
- Develop IP version independent applications
3. Name and address resolution

- Two basic resolution functions
- DNS queries/responses are sent using IPv4/IPv6, regardless data records.

PROBLEMS
- Existing IPv4 name/address resolution calls are not valid for IPv6.

RECOMMENDATION
- Use new IP version independent structures and functions
4. Specific IP dependencies

- IP address selection.
- Application framing.
- Storage of IP addresses.

PROBLEMS
- There are more IP dependencies apart from network API calls and structures.

RECOMMENDATION
- Review source code in detail
Agenda

- Transition Architecture
- Evolution of Applications
- Application Transition Scenarios
- Application Porting Considerations
- BSD Socket API
- IP Version Independent Applications
- Recommendations
BSD Socket API

IPv6 addresses are 128 bit long.

Changes in the application networking part

1. Data structures:
   Socket address struct

2. Conversion functions between:
   • String and binary representation
   • Name and address

3. Socket calls
Generic Socket Address

**sockaddr**

<table>
<thead>
<tr>
<th>Family</th>
<th>AF_XXXX</th>
</tr>
</thead>
</table>

- struct sockaddr {
  - sa_family_t sa_family;
  - char sa_data[14];
}

16 bits
### IPv4 socket address structure

**sockaddr_in**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Family</td>
<td>AF_INET</td>
</tr>
<tr>
<td>Port Number</td>
<td></td>
</tr>
<tr>
<td>IPv4 Address</td>
<td>(32 bits)</td>
</tr>
<tr>
<td>Unused</td>
<td></td>
</tr>
</tbody>
</table>

```c
struct sockaddr_in {
    sa_family_t     sin_family;
    in_port_t       sin_port;
    struct in_addr  sin_addr;
    char            sin_zero[8];
};

struct in_addr {
    uint32_t     s_addr;
};
```
### IPv6 socket address structure

#### sockaddr_in6

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Family</td>
<td>AF_INET6</td>
</tr>
<tr>
<td>Port Number</td>
<td>sin6_port;</td>
</tr>
<tr>
<td>Flow Info</td>
<td>sin6_flowinfo;</td>
</tr>
<tr>
<td>IPv6 Address</td>
<td>s6_addr[16];</td>
</tr>
<tr>
<td>Scope ID</td>
<td>sin6_scope_id;</td>
</tr>
</tbody>
</table>

```c
struct sockaddr_in6 {
    sa_family_t     sin6_family;
    in_port_t       sin6_port;
    uint32_t        sin6_flowinfo;
    struct in6_addr sin6_addr;
    uint32_t        sin6_scope_id;
};

struct in6_addr {
    uint8_t       s6_addr[16];
};
```
Protocol independent structure

struct sockaddr_storage {
    sa_family_t sin6_family;
    __ss_align; char __ss_padding[_SS_PADSIZE];
};
Structure Allocation

IPv4-only

```c
struct sockaddr_in serverAddr;
/* ... */
bind(serverfd, (struct sockaddr *)&serverAddr, &alen);
```

IPv6-only

```c
struct sockaddr_in6 serverAddr;
/* ... */
bind(serverfd, (struct sockaddr *)&serverAddr, &alen);
```

Protocol independent

```c
struct sockaddr_storage serverAddr;
/* ... */
bind(serverfd, (struct sockaddr *)&serverAddr, &alen);
```
IPv6 addresses are 128 bit long.

Changes in the application networking part

1. Data structures:
   Socket address struct

2. Conversion functions between:
   - String and binary representation
   - Name and address

3. Socket calls
Conversion functions

- RESOLVER: returns IP address structure.
Conversion: name & IP addr

<table>
<thead>
<tr>
<th>IPv4 only</th>
<th>IPv4 &amp; IPv6</th>
</tr>
</thead>
<tbody>
<tr>
<td>(v4 addr)</td>
<td>(v4, v6, v4-map v6 addr)</td>
</tr>
<tr>
<td>gethostbyname()</td>
<td>getaddrinfo()</td>
</tr>
<tr>
<td>gethostbyaddr()</td>
<td>getnameinfo()</td>
</tr>
</tbody>
</table>

Protocol Independent Functions
struct addrinfo hints, *res;

memset(0, &hints, sizeof(hints);
hints.ai_flags = AI_PASSIVE;
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM; /* SOCK_DGRAM */
getaddrinfo(NULL, DAYTIME_PORT, &hints, &res);
/* ... */
freeaddrinfo(res);
struct sockaddr STORAGE clientAddr;
char clientHost[ADDRLEN];
char clientPort[PORTLEN];
/* ... */
connectedfd = accept(serverfd,
    (struct sockaddr *)&clientAddr,
    &alen);

getnameinfo((struct sockaddr *)&clientAddr, addrLen,
    clientHost, sizeof(clientHost),
    clientPort, sizeof(clientPort),
    NI_NUMERICHOST);

printf("Request from host=\[%s\] port=\[%s\]\n",
    clienthost, clientservice);
}
Conversion: string & binary

- Conversions between the text representation and the binary value in network byte ordered (socket address structure).

<table>
<thead>
<tr>
<th>String -&gt; Binary</th>
<th>IPv4 only</th>
<th>IPv4 &amp; IPv6</th>
</tr>
</thead>
<tbody>
<tr>
<td>inet_aton()</td>
<td>inet_pton()</td>
<td></td>
</tr>
<tr>
<td>inet_addr()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary -&gt; String</th>
<th>IPv4 only</th>
<th>IPv4 &amp; IPv6</th>
</tr>
</thead>
<tbody>
<tr>
<td>inet_ntoa()</td>
<td>inet_ntop()</td>
<td></td>
</tr>
</tbody>
</table>
**inet_p ton**

### IPv4-only code

```c
c struct in_addr addr4;
c char *addrStr4="127.0.0.1";
c /* … */
inet_aton(addrStr4, &addr4);
```

### IPv6-only code

```c
c struct sockaddr_in6 addr6;
c char *addrStr6="::1";
c /* … */
inet_p ton(AF_INET6, addrStr6, &addr6);
```

### IPv4/IPv6 code

```c
c struct sockaddr_storage addr;
c int family = AF_INET6;
c char *addrStr="::1";
c /* … */
inet_p ton(family, addrStr, &addr);
```
### inet_ntop

#### Old IPv4-only code

```c
struct in_addr addr4;
char *addrStr4;
/* ... */
addrStr4 = inet_ntoa(addr4);
```

#### New IPv6-only code

```c
struct sockaddr_in6 addr6;
char addrStr6[INET6_ADDRSTRLEN];
/* ... */
inet_ntop(AF_INET6, addr6, addrStr6, INET6_ADDRSTRLEN);
```

#### IPv4/IPv6 code

```c
struct sockaddr_storage addr;
char addrStr[ADDRSTRLEN];
/* ... */
inet_ntop(addr.ss_family, addr, addrStr, sizeof(addrStr));
```
IPv6 addresses are 128 bit long.

Changes in the application networking part

1. Data structures: Socket address struct

2. Conversion functions between:
   • String and binary representation
   • Name and address

3. Socket calls
Socket calls (TCP)

Server
- socket
  - bind
  - listen
  - accept
  - read
  - write
  - close

Client
- socket
  - connect
  - write
  - read
  - close
Socket calls (UDP)

Server

1. socket
2. bind
3. recvfrom
4. sendto
5. close

Client

1. socket
2. sendto
3. recvfrom
4. close
• Address family, socket type and protocol in the *socket* call:

```
int socket (int family, int type, int protocol);
```

• Casting to generic socket structure *struct sock addr *.

- IPv4: from *(struct sockaddr_in *)
- IPv6: from *(struct sockaddr_in6 *)
- Protocol independent: from *(struct sockaddr_storage *)

• Size of socket address structure.
Socket Options (setsockopt)

- IPV6_UNICAST_HOPS

Multicast:
- IPV6_MULTICAST_IF
- IPV6_MULTICAST_HOPS
- IPV6_MULTICAST_LOOP
- IPV6_JOIN_GROUP
- IPV6_LEAVE_GROUP

- IPV6_V6ONLY for AF_INET6
Agenda

- Transition Architecture
- Evolution of Applications
- Application Transition Scenarios
- Application Porting Considerations
- BSD Socket API
- IP Version Independent Applications
- Recommendations
IP Version Independent Applications

- Use IP version independent structures:
  - sockaddr_storage

- Use IP version independent functions:
  - getaddrinfo() / getnameinfo()

- Not use inet_ntop() / inet_pton()

- Iterated jobs for finding the working address:
  - Server:
    - listening packets addressed to a specific port.
  - Clients:
    - connecting to one of the server addresses.
Server example

- get_list_of_IP_addresses
  - list_of_IP_addresses == null
    - no
    - create_socket (address_family)
      - Protocol not support
        - error?
          - no
          - yes
            - bind_socket
              - yes
              - error?
                - no
                - yes
                  - listen_socket
                    - NO SOCKET
Client example

1. `get_list_of_IP_addresses`
   - `list_of_IP_addresses == null`
     - no
     - `create_socket (address_family)`
       - Protocol not supported
         - error?
           - no
           - `connect_socket`
             - yes
             - error?
               - no
               - SOCKET CONNECTED
             - no
               - NO SOCKET
           - yes
           - SOCKET CONNECTED
Server applications supporting IPv4 & IPv6

- IPv6 servers:
  - IPv4 clients connect using an IPv4-mapped IPv6 address
  - IPv6 clients connect using an IPv6 address

- Dual servers: servers use different sockets for IPv4 and IPv6 connections (IPV6_ONLY):
  - IPv4 clients connect using an IPv4 address
  - IPv6 clients connect using an IPv6 address
Dual servers: different IPv4 & IPv6 sockets

- get_list_of_IP_addresses
  - list_of_IP_addresses == null || maxSock== 2
    - no
      - create_socket (address_family)
        - Protocol not support
          - error?
            - no
              - IPv6?
                - no
                - IPV6 ONLY
                  - bind_socket
                    - yes
                      - error?
                        - no
                        - listen_socket
                          - yes
                          - error?
                            - no
                            - NO SOCKET
                          - no
                            - NO SOCKET
                          - yes
                          - NO SOCKET
                        - no
                        - NO SOCKET
Agenda

- Transition Architecture
- Evolution of Applications
- Application Transition Scenarios
- Application Porting Considerations
- BSD Socket API
- IP Version Independent Applications
- Recommendations
Recommendations for applications

- Not use hard coded IP addresses
- IP version independent applications
  - Structures & functions
  - Iterated jobs to find a valid connection
- Develop dual applications:
  - Valid for both protocols IPv4 & IPv6