GFD-I.044
Category: Informational
IPv6_WG

T. Chown, University of Southampton
S. Jiang, UCL
J. Bound, HP
P. O'Hanlon, UCL

7 January 2005

Guidelines for IP version independence in GGF specifications

Status of This Memo

This memo provides information to the Grid community regarding IP version independence in GGF specifications.   It does not define any standards or technical recommendations. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document serves two functions. Its motivation is to aid in the creation of IP-version independent specifications and consequently, in the transition of IPv4 applications to support IPv6 operation.   First, it describes how to avoid IPv4 dependencies in GGF specifications. Secondly, it outlines new, IPv6-specific issues for application designers and implementers. It should be used by all GGF WGs and as a checklist for document approval.

T. Chown, University of Southampton
S. Jiang, UCL
J. Bound, HP
P. O'Hanlon, UCL

7 January 2005

Contents

1. Introduction

The goal of this document is to help the reader understand the issues in making applications Internet Protocol version 6 (IPv6) aware, such that new specifications can be written in an IP-independent fashion. It describes how to avoid Internet Protocol version 4 (IPv4) dependencies in Global Grid Forum (GGF) specifications. It is intended that it should be used by all GGF Working Groups (WGs) and as a checklist for document approval.

The document also outlines the design and implementation issues when considering IPv6-enabled applications. While certain issues are implementation-specific, the author of the specification should be aware if these issues, where there may be differences in operation between IPv4 and IPv6.

Some documentation already exists in the general area of application issues for IPv4 and IPv6 integration, e.g. the LONG project guide [LONG-PORTING] and Internet Engineering Task Force (IETF) IPv6 Operations WG studies [APP-ASPECTS] (the latter being considered for final publication as an Informational RFC at the time of writing).

In this guide we first discuss the requirement for dual or hybrid stack operation for IPv4 and IPv6. We then discuss standards or specifications aspects, before looking at implementation oriented issues and those that are specific to IPv6 (highlighting differences and similarities to IPv4).

Finally, we list some specific recommendations to the writers of GGF specifications, to ensure that specifications are independent of the version of IP and that they guide implementers to avoid accidental dependencies.

2. IPv4 and IPv6 Operational Relationships

Internet Protocol Version 6 (IPv6) is the successor to the current version of IP (IPv4). It has a number of benefits including the larger address space, autoconfiguration, better aggregation of routing tables, a complete solution for mobile IP, IPsec being available end-to-end globally, and a simplified header format.

The larger address space removes the need for Network Address Translation (NAT) [RFC3022], making end-to-end application operation simpler to consider for the designer and developer.

The base IPv6 specification is given in [RFC2460] and the addressing architecture in [RFC3513].

IPv6 will not replace IPv4 in the foreseeable future, except in areas of significant IPv4 address space drought. In most circumstances there will be a long period of coexistence. As a result many applications will need to be aware of both protocols, and able to run over either. Existing applications will need to be ported to support IPv6, while new applications that are aware of network details will need to be designed with IPv6 in mind from the outset.

While IPv4-IPv6 interworking can be achieved with translation (such as NAT-PT [RFC2766]) and proxy methods (such as dual-stack application layer gateways), it is generally architecturally cleaner if a client wishing to interact with an IPv6 service uses IPv6 to communicate directly, rather than relying on an intermediary translation. Applications will need to continue to operate between IPv4 endpoints, but also be able to communicate using IPv6 when available and selected.

Thus the general case for IPv6 operation would be an IPv4 and IPv6-capable application, running over a suitable transport (e.g. TCP, UDP) on top of a dual or hybrid IPv4 and IPv6 stack, with the underlying network configured for and running both protocols.

3. Standards and Specification Issues

3.1 IP Address Representation

The most obvious difference between IPv4 and IPv6 lies in the address size and format itself. In IPv4, addresses are 32 bits, represented as a dot-delimited decimal quad address, while in IPv6 they are 128 bits, represented as colon-delimited hexadecimal address.

3.2 Storage and Display of IP Addresses

Thus there are different storage requirements for addresses in each protocol. From the implementation perspective these issues are discussed in Section 4.2, where storage in an IP-independent format is presented.

These may affect specifications where text representations of addresses are being handled. An IPv4 address may be up to 15 characters long (12 digits plus three dots), while an IPv6 address may be up to 39 characters long (32 hexadecimal digits plus seven colons). The minimum length of a displayed IPv4 address is seven characters (four digits plus three dots), and an IPv6 address is three characters (two colons and a digit) for a regular address, or two characters ('::') for the unspecified address. The '::' notation indicates one or more groups of 16 bits of zeros.

Note that IPv4-mapped IPv6 addresses [TRANSMECH] can be longer than 39 characters. Such addresses may be written in the form ::ffff:<IPv4 address>, e.g. ::ffff:152.78.70.1, but the full expansion could be as long as 0000:0000:0000:0000:0000:ffff:nnn.nnn.nnn.nnn, i.e. 45 characters. See Section 4.5 below for more discussion of these mapped addresses.

3.3 Use of Fully Qualified Domain Names

Some applications may pass IP addresses in the payload of their data. In the case of IPv6 it will be commonplace for hosts to have multiple IPv6 addresses, and potentially for more renumbering events to occur. There are also additional IPv6 host addresses for hosts implementing IPv6 Privacy Extensions [RFC3041] (see Section 5).

As a result, there is a stronger argument for hosts to exchange fully qualified domain names (FQDNs) rather than IP addresses, especially given the FQDN is an IP-independent identifier for the host. It is currently not uncommon practice for applications, including peer-to-peer applications, to exchange IP addresses as data for communication endpoints. The storage of IP addresses (either IPv4 or IPv6) in files for non-trivial lengths of time is not recommended. Developers should recognize that peer IP addresses stored in files will generally become stale faster than domain names.

3.4 Handling Literal IPv6 Addresses

In IPv4, the common delimiter for address and port representation is a colon. Since IPv6 addresses contain colons, a new method for expressing address:port pairs is required where literal addresses are used.

The method adopted to handle this problem in application or context-dependent URIs [RFC2396] is the format specified in RFC2732, i.e. [address]:port, e.g. http://[2001:0DB8:a0:1::1]:8080.

The '[]' solution of RFC2732 can be used for other situations, e.g. in SIP-based applications.

3.5 Documentation Examples

There is an IETF proposal to use a common documentation prefix in specification documents [RFC3849], namely 2001:DB8::/32. To reduce the likelihood of conflict and confusion when relating documented examples to deployed systems, the use of this IPv6 unicast address prefix is recommended in examples contained within specifications.

4. Implementation Issues

Implementation issues span many areas. We outline these in this section. While specifications should not be written to be or become implementation-specific, they should be aware of implementation constraints.

4.1 APIs

The introduction of IPv6 requires changes to the APIs. There are currently two main programming platforms supporting IPv6, namely C and Java.

The new APIs and data structures for TCP/IP sockets (as used in the C programming language) are defined in the Basic Socket Extensions for IPv6 [RFC3493] (which obsoletes RFC2553) and the Advanced Socket API for IPv6 [RFC3542] (which obsoletes RFC2292).

It is easier to port applications when network components are modular and well–isolated, and do not make assumptions about the IP version (e.g. representing an IP address by four integer values); the same principle should apply to new implementations.

These specify the socket address structures, address conversion functions, socket options and name resolution functions. The definitions include IP-independent functions, as well as those for IPv6-only applications. In the current state of IPv6 deployment, IP-independent applications are preferred, such that they can operate in the presence of either or both protocols (without recompilation).

However, there are still currently some subtleties in behaviour between platforms, e.g. in binding to IPv4 and IPv6 simultaneously, due to different *bind()* call implementations.

The Java Development Kit (JDK) as of version 1.4.0 supports basic IPv6 functionality for Linux and Solaris platforms. MS Windows support is expected in JDK1.5 (at the time of writing support for IPv6 exists in the publicly available beta JDK1.5). The JDK includes network preferences for IPv6 (i.e. *java.net.preferIPv4Stack, java.net.preferIPv6Addresses*) [JDKv6].

There is as yet no definition within Java for advanced API functions, e.g. writing a Flow Label field from a Java application. There needs to be action within the Java community to investigate and specify advanced API functionalities where required, including handling of IPv4-mapped addresses. These issues are being highlighted within another draft on Java from the IPv6-WG.

IPv6 support is also being made available in other languages used by the Grid community, including Perl (for example, via the IO::Socket::INET6 made available by the Euro6IX project) and Python (as standard).

4.2 Storage of IP addresses

In the sockets API, there are data structures that may be used for IPv4 or IPv6 applications – *sockaddr_in()* and *sockaddr_in6()* – but also a generic IP independent structure *sockaddr_storage()* that hides the specific structure that the application is using. The latter should be preferred for IP-independent applications.

For IP address storage we have *in_addr* (IPv4-only), *in6_addr* (IPv6-only), and *addrinfo* (IP dependent). Again, the latter is preferred.

As described in Section 3.2, IPv4 and IPv6 have different textual representations.

There are differences in special addresses, e.g. the *loopback/localhost* address is 127.0.0.1 in IPv4 and ::1 in IPv6. Use of *localhost* by name abstracts that difference.

The LONG project guide [LONG-PORTING] contains IP-independent programming examples for the sockets API (C language); these principles are reinforced by IETF IPv6 Operations WG studies [APP-ASPECTS].

4.3 Resolution and conversion functions

The new IP-independent functions for name-to-address lookups in C are *getnameinfo()* and *getaddrinfo(),* which replace *gethostbyname()* and *gethostbyaddr().*

It is important to note that one should not assume IPv6 connectivity by the presence of an IPv6 DNS record (a AAAA record). The target host may have no or only some IPv6 services actually enabled.

The choice of preferred protocol, and address selection mechanisms, are defined in [RFC3484], by which a returned address list can be inspected to select addresses for source and destination addresses. An application may be configured to prefer IPv6 where available, but it is recommended that it be possible for that preference to be overridden.

Regarding reverse DNS lookups, there is an ongoing transition at the time of writing from the ip6.int to ip6.arpa namespace [RFC3152]. Some transitional address space (e.g. under the 6to4 prefix of 2002::/16) has no defined reverse lookup namespace.

There are also new functions for conversion of addresses from binary to text/string format.

4.4 Parsing and Displaying IP address

New code will be required to parse IPv6 address where entered as input or parameters to applications. Such code will need to be aware of IPv6 address formats, including conventions such as the ‘::’ shortcut for zero value 16bit sequences.

Support for automatic parsing of literals in various languages is growing, for example *Java.net.URI* will automatically parse and handle them.

IP addresses may be used in configuration files, or perhaps in access control files. In such situations FQDNs could be used.

The different formats and lengths of IPv4 and IPv6 addresses may have a significant impact on the layout of such addresses when presented in graphical user interfaces.

4.5 IPv4-mapped Address Handling

An IPv4 client application on an IPv4-only node can talk to an IPv6 application on a dual stack node using IPv4 packets between the nodes; however the IPv6 application will see the addresses as IPv4-mapped IPv6 addresses, of the form ::ffff:a.b.c.d where a.b.c.d is the IPv4 address.

This mapping may occur in the API, or the mapped addresses could be seen on the wire. The latter is undesirable for security (spoofing) reasons [V6MAP-HARM].

There is a view that IPv4-mapped IPv6 addresses add implementation complexity, cause degraded portability, and increase access control complexity, and should perhaps be deprecated. However, they serve a useful purpose, and have a wide installed base, and are thus likely to remain in place. An application may want to treat all addresses as IPv6 including IPv4 as documented in RFC3493.

Applications should handle IPv4-mapped IPv6 addresses correctly and securely.

5. Implications of new features of IPv6

5.1 Network Address Translation (NAT)

Network Address Translation (NAT) [RFC3022] is defined for IPv4. It was originally intended as a method of IPv4 address conservation until IPv6 was defined, although some sites use NAT in conjunction with private IP addresses (see below) for reasons of address stability or perceived security benefits.

With IPv6's address space, there is no technical need for IPv6 networks to use NAT. A fundamental belief of the IPv6 Forum is that end-to-end is re-enabled with IPv6 and thus application designers should assume end-to-end transparency when considering IPv6 applications.

5.2 Private, Local Scope IP Addresses

IPv4 has a set of address ranges reserved for private network usage [RFC1918]. In the initial IPv6 Address Architecture, IPv6 included unicast site-local scope addressing. However, site-locals as defined are being deprecated (for reasons including address leakage and ambiguity) within the IETF.

Application designers should not currently assume the presence of any unicast site-local scoped address range in IPv6. A replacement for the original site-local definition is currently being defined within the IETF IPv6 Operations WG [UNIQUE-LOCAL].

5.3 IPv6 Anycast Address

The IPv6 addressing architecture defines an "anycast" address which is an IPv6 address [RFC2526] that is assigned to one or more network interfaces (typically belonging to different nodes), with the property that a packet sent to an anycast address is routed to the "nearest" interface having that address, according to the routing protocols' measure of distance.

While implementation of anycast addressing requires local router configuration, the availability of anycast should be considered by specification authors.

5.4 IPv6 Flow Label

Usage of the IPv6 Flow Label field, which occupies 20 bits of the IPv6 Header, was first defined in RFC1809, and then further referenced in RFC2460. The Flow label was initially designed for use in an Integrated Services QoS environment, but it has seen little if any usage to date, and there has been some confusion over aspects of the original specification.

The IETF has thus updated the definition of the Flow Label semantics [RFC3697], which describes how the Flow Label field may be used, and how nodes may act upon the value of the Flow Label field.

Application designers may only exploit the IPv6 Flow Label where both communicating hosts are IPv6 capable.

5.5 IPv6 Privacy Extensions

When an IPv6 node uses IPv6 Stateless Address Autoconfiguration it will always generate the same 64-bit host part for its 128-bit address. When a host moves between networks with different prefixes, and it initiates connections from those networks, this raises a privacy (host tracking) issue.

IPv6 Privacy Extensions [RFC3041] addresses this issue by effectively using a random 64-bit host part for statelessly autoconfiguring hosts. This standard also allows a static host to regenerate a new privacy address regularly, e.g. every 24 hours. The host may still keep a regular global IPv6 address through which it can be contacted. This reduces privacy concerns, but means that existing IP-based authentication and usage assumptions may no longer hold.

Application designers need to consider that IPv6 hosts may connect to services while using Privacy Extensions.

5.6 IPv6 Multicast

IPv6 includes Multicast for basic features such as Neighbour Discovery. IPv6 link scope multicast replaces the function of broadcast on an IPv4 subnet.

The models for Any Source (ASM) or Source Specific (SSM) Multicast are generally similar between IPv4 and IPv6. It is likely that SSM will become more widely deployed in IPv6 due to its simpler architecture. However, this puts extra requirements on the application in comparison to PIM-SM (based on the ASM model). The MSDP method for handling inter-domain ASM is not being used in IPv6; instead a method based on embedding the Rendezvous Point address is under study.

Application developers should thus consider SSM operation where appropriate.

5.7 Path MTU Discovery

IPv6 requires Path Maximum Transmission Unit (PMTU) Discovery [RFC1981] to be implemented. Fragmentation is designed to occur at endpoints of communication, and not at routers on the path.

Section 5 of [RFC2460] requires that every network link support an MTU of at least 1280 octets.

Application developers may wish to consider performance issues of data unit sizing to align with the IPv6 PMTU.

5.8 Extensible IPv6 Header Format

The extensible nature of the IPv6 Next Header construct allows new IPv6 Headers to be defined and used by applications (subject to access through the API).

5.9 Differentiated Services Code Point (DSCP)

The semantics and use of the Differentiated Services Code Point (DSCP) for DiffServ-based Quality of Service is expected to be the same between IPv4 and IPv6, as described in [RFC2474].

5.10 IPsec

The support of Authentication (AH) and Encapsulating Security Payload (ESP) Headers is required in a "full implementation" of IPv6 as defined in [RFC2460], although their use is optional.

Section 4 of the IP Security Architecture [RFC2401] suggests that all IPv6 implementations will support IPsec, however in the early stages of IPv6 deployment such implementations are still in the minority.

Designers should assume that IPsec functionality will be the same between IPv4 and IPv6, but that IPv6 will benefit from wider implementation of IPsec in operating system products and that the removal of the need for NAT will enable end-to-end use of IPsec in tunnel or transport mode. Note that IPsec using only ESP can traverse a NAT, while the AH functionality is impaired by NATs. The behaviour of Internet Key Exchange (IKE) is not affected in the face of NATs.

5.11 IP Mobility

Mobile IPv6 improves on Mobile IPv4 through various features including Route Optimisation.

Application designers should not need to explicitly consider Mobility, which may be handled by the underlying IPv6 network (if the node supports Mobile IPv6 functionality).

6. IP-independent specifications: recommendations

There are general recommendations that those producing GGF specifications can follow. Consideration should also be given where appropriate to practical implementation issues.

6.1 Specification

Within specifications:

1. If an IP address must be included in a protocol element or in stored state required by a specification, an address type code must be included, as well as adequate space for either an IPv4 or an IPv6 address. The *addrinfo* structure may be appropriate.

2. Literal IPv6 addresses are recommended to use the format of RFC2732 where *address:port* pairs are expressed. Anywhere in a specification that the URI or URL format occurs, if the normative references do not include RFC2732 then there is in fact an IPv4 dependency, because RFC2396 (Uniform Resource Identifiers: Generic Syntax) only defines IPv4 literals.

3. Fully Qualified Domain Names (FQDNs) are recommended to be used in preference to IP addresses where practical to do so.

4. IPv6 addresses may potentially be shorter or longer than IPv4 addresses when represented as a text string. An IPv6 address may between two and 45 characters, as opposed to seven to 15 characters for an IPv4 address.

5. Special addresses, such as *loopback/localhost* (127.0.0.1 in IPv4, ::1 in IPv6), are represented differently in each protocol; use of *localhost* by name abstracts this difference.

6. The agreed IPv6 documentation prefix is recommended to be used in specification documents.

7. New implications of IPv6, as outlined in Section 5, should be considered.

6.2 Implementation

When implementing IP-independent applications:

1. It is recommended that code is developed to be IP-independent, not IPv4-only or IPv6-only.

2. IP-independent API's and data structures are recommended to be used, e.g. in C/C++ the *getnameinfo()* function and *addrinfo* for storage.

3. Code should be modular such that future changes to the networking mechanics are be minimal.

4. Care should be given to how IPv4 or IPv6 protocols are preferred and selected when both protocols are available.

5. Applications may need to iterate (or parallelise) connection attempts using multiple different source or address combination pairs due to multi-addressing (with multiple IPv6 addresses, or IPv4 and IPv6 addresses in dual stack nodes).

6. Graphical user interfaces must take into account the different textual lengths and separators of IPv4 and IPv6 addresses. When IP addresses need to be displayed or entered in user interfaces, both IPv4 and IPv6 formats must be supported. This may have significant impact on the layout of such graphical user interfaces.

7. New implications of IPv6, as outlined in Section 5, should be considered.

7. Security Considerations

This document is informational, providing guidance for IP-independence in GGF specifications. It does not in itself have any security implications. There are no IPv6 security considerations that are specific to Grid. IPv6 intrinsically supports the same security features and is largely exposed to the same security threats as IPv4. All of the RFCs defining IPv6, including [RFC2460], contain relevant security considerations, to which the reader is referred.

Author Information

Tim Chown
Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ
United Kingdom
Email: tjc@ecs.soton.ac.uk
Phone: +44 23 8059 3257

Jim Bound
Hewlett Packard
110 Spitbrook Road
Nashua, NH 03062
USA
Email: jim.bound@hp.com
Phone: +1 603.884.0062

Piers O'Hanlon
Computer Science Department
University College London
Gower Street
London WC1E 6BT
United Kingdom
Email: p.ohanlon@cs.ucl.ac.uk
Phone: +44 20 7679 3670

Sheng Jiang
Computer Science Department
University College London
Gower Street
London WC1E 6BT
United Kingdom
Email: s.jiang@cs.ucl.ac.uk
Phone: +44 20 7679 3670

Acknowledgements

The authors would like to thank members of the GGF IPv6 WG and wider Grid community for their comments, including Matt Crawford, Brian Carpenter, Jon Crowcroft, William Allcock and Fred Templin.

Intellectual Property Statement

Full Copyright Notice

References

[APP-ASPECTS] M. Shin, Y. Hong, J. Hagino, P. Savola, E. Castro, *Application Aspects of IPv6 Transition*, IETF Internet Draft, February 2004 (work in progress).

[JDKv6] *Java Development Kit 1.5.0, IPv6 Guide*, http://java.sun.com/j2se/1.5.0/docs/guide/net/ipv6_guide

[LONG-PORTING] T. de Miguel, E. M. Castro, *Programming guidelines on transition to IPv6,* LONG Project, http://www.ist-long.com/, January 2003.

[RFC1918] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. deGroot, E. Lear, *Address Allocations for Private Internets*, IETF RFC, February 1996.

[RFC1981] J. McCann, S. Deering, J. Mogul, *Path MTU Discovery for IPv6*, IETF RFC, August 1996.

[RFC2396] T. Berners-Lee, R. Fielding, L. Masinter, *Uniform Resource Identifiers (URI): Generic Syntax Uniform Resource Identifiers (URI): Generic Syntax*. IETF RFC, August 1998.

[RFC2401] S. Kent, R. Atkinson, *Security Architecture for the Internet Protocol*, IETF RFC, November 1998.

[RFC2460] S. Deering, R. Hinden, *Internet Protocol Version 6 Specification*, IETF RFC, December 1998.

[RFC2474] K. Nichols, S. Blake, F. Baker, D. Black, *Definition of the Differentiated Services Field in the IPv4 and IPv6 Headers*, IETF RFC, December 1998.

[RFC2526] D. Johnson, S. Deering, *Reserved IPv6 Subnet Anycast Address*, IETF RFC, March 1999.

[RFC2732] R. Hinden, B. Carpenter, L. Masinter, *Format for Literal IPv6 Addresses in URL's*. IETF RFC, December 1999.

[RFC2766] G. Tsirtsis, P. Srisuresh, *Network Address Translation - Protocol Translation (NAT-PT)*. IETF RFC, February 2000.

[RFC3022] K. Egevang, P. Srisuresh, *Traditional IP Network Address Translator,* IETF RFC, January 2001.

[RFC3041] T. Narten, R. Draves, *Privacy Extensions for Stateless Address Autoconfiguration in IPv6*, IETF RFC, January 2001

[RFC3152] R. Bush, *Delegation of IP6.ARPA*, IETF RFC, August 2001.

[RFC3484] R. Draves, *Default Address Selection for Internet Protocol Version 6*, IETF RFC, February 2003.

[RFC3493] R. Gilligan, S. Thompson, J. Bound, J. McCann, W. Stevens, *Basic Socket Interface Extensions for IPv6*, IETF RFC (obsoletes RFC2553), February 2003.

[RFC3513] S. Deering, R. Hinden, *IP Version 6 Addressing Architecture*, IETF RFC (obsoletes RFC2373), April 2003.

[RFC3542] R. Stevens, M. Thomas, E. Nordmark, T. Jinmei, *Advanced Sockets Applications Program Interface (API) for IPv6*, IETF RFC (obsoletes RFC2292), May 2003.

[RFC3697] J. Rajahalme, B. Carpenter, A. Conta, S. Deering, *IPv6 Flow Label Specification*, IETF RFC, March 2004.

[RFC3849] G. Huston, A. Lord, P. Smith, *IPv6 Address Prefix Reserved for Documentation*, IETF RFC, July 2004

[TRANSMECH] E. Nordmark, R. E. Gilligan, *Basic Transition Mechanisms for Hosts and Routers*, IETF Internet Draft, September 2004, (work in progress)

[UNIQUE-LOCAL] R. Hinden, B. Haberman, *Unique Local IPv6 Unicast Addresses*, IETF Internet Draft, November 2004, (work in progress)