Network Working Group
Internet-Draft: draft-main-ipaddr-text-rep-01
Category: Standards-Track
Expires: 2004-04-22

A. Main
Black Ops Ltd

2003-10-22

Textual Representation of IPv4 and IPv6 Addresses

Status of this Memo

Abstract

Historically, the conventional textual representations of IPv4 and IPv6 addresses have been poorly specified. This document gives precise definitions of these conventions, together with advice for implementors.

Change Log

Changes from -00 to -01:

- Discussion of the ABNF grammar for IPv4 addresses given in [MTP] and [SMTP-1].
- Explanation of the faulty ABNF grammar in [IPV6-AA-2].
- Mention of another faulty grammar for IPv6 addresses in [SMTP-2].
- Specific references for the enclosing of IP addresses in brackets.

- Changed intended category from Informational to Standards-Track.
- A couple of minor wording changes.
- Updated author's address.

1 Introduction

For as long as IP has existed, there has been a need to represent IP addresses in textual contexts, but the nature of these requirements has changed. IP addresses are textually represented much more widely than appears to have been originally envisioned; in particular, such representation has become a part of many network protocols. There is an increasing need for interoperability in IP address textual representations, for it is more commonly software than humans that read and write addresses in this format.

Historically, the definitions of IP address textual representations have been loose, underspecifying the syntax. They have also always been a minor part of a standard whose main focus is some other problem. This makes them difficult to locate and inconvenient to cite. With IPv6 address textual representation incorporating the IPv4 format by reference, the IPv6 format has not previously been completely specified in a single RFC.

This document collects together the complete syntax for textual representation of IPv4 and IPv6 addresses, clarifying the underspecified parts. It is intended to be a complete and unambiguous specification of these address formats, located together in a single document for ease of reference.

Section 2 of this document discusses the history of the specification and implementation of textual representation of IP addresses. Section 3 gives the complete syntax. Section 4 gives some advice for implementors.

1.1 Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [REQ-TERM].

1.2 Augmented BNF Notation

Syntax specifications in this document use augmented BNF notation as defined in [ABNF]. The `core rules' in appendix A of [ABNF] are used as defined there.

2 History

2.1 IPv4 Dotted Octet Format

2.1.1 Early Practice

The original IPv4 "dotted octet" format was never fully defined in any RFC, so it is necessary to look at usage, rather than merely find an authoritative definition, to determine what the effective syntax was. The first mention of dotted octets in the RFC series is in [MTP], a predecessor of SMTP, which interestingly mentions two address formats that evidently by then had some currency:

> One form is a decimal integer prefixed by a pound sign, "#", which indicates the number is the address of the host. Another form is four small decimal integers separated by dots and enclosed by brackets, e.g., "[123.255.37.321]", which indicates a 32 bit ARPA Internet Address in four eight bit fields.

[MTP] also gives a partial ABNF specification for these address formats. The grammar requires four dot-separated parts, each of which consists of "three digits representing an integer value in the range 0 through 255". Presumably the exclusion of one- and two-digit parts is a mistake, since it clashes with the example quoted above. [MTP]'s descendent [SMTP-1] gives the same partial ABNF but requires each part to consist of "one, two, or three digits representing a decimal integer value in the range 0 through 255".

A few months later, [IPV4-NUMB] (the "Assigned Numbers" RFC published at the same time as [IPV4]) gave, for the first time, a table of assigned IP addresses. (Previous "Assigned Numbers" RFCs, predating classful addressing, had merely had a table of "network numbers". Although the new table retained the title "assigned network numbers", it was actually expressed in terms of address blocks.) This table used dotted decimal format, zero-filling each encoded octet to three digits. The notes accompanying the table said:

> One notation for internet host addresses commonly used divides the 32-bit address into four 8-bit fields and specifies the value of each field as a decimal number with the fields separated by periods. For example, the internet address of ISIF is 010.020.000.052. This notation will be used in the listing of assigned network numbers.

Shortly thereafter, [NCP-TCP] gave a handful of live IP addresses without comment on the format, for example, "ARPANET/SATNET gateway at BBN (10.3.0.40)".

The next description of dotted octet notation is in [HOST-TBL-2], defining the host table file format, which describes the notation as "four decimal numbers separated by a period. Each decimal number represents 1 octet.". One of its example host table entries was "GATEWAY : 10.0.0.77, 18.8.0.4 : MIT-GW :: MOS : IP/GW :".

It is at this point in the history that [SMTP-1] reiterated the formal grammar given in [MTP], with the emendation to unambiguously allow decimal numbers of fewer than three digits.

[HREQ-APP], a much later and more significant standard, describes IP address text representation, in the course of recommending that applications allow users to specify IP addresses directly as well as via DNS host names. It merely describes the format as "dotted-decimal ("#.#.#.#") form". It gives no example of an address in this format.

So far we have seen dotted octet format in five different types of situation: a network protocol (machine-parsed email address), a table of address blocks, English text (discussion the NCP to TCP/IP switch), a machine-readable database (the host table), and human interfaces to network applications. All are consistent about dividing the address into octets and representing each octet purely in decimal, but there are two variants of the format due to a more subtle issue. The explicit descriptions of the format given so far have been silent about the permissibility of leading zeroes in octet representations; only one example, a human-oriented table of addresses, used leading zeroes.

This variation in the format, presumably initially intended to be of no consequence, lives on today. The direct descendent of [IPV4-NUMB]'s "assigned network numbers" table is the IANA-maintained "ipv4-address-space" table, which at the date of this document still shows octet values in zero-filled three-digit decimal. In all non-table contexts in which IPv4 addresses appear, including anything intended to be machine-readable, almost universally leading zeroes are suppressed. (Curiously, a different IANA-maintained table, the "multicast-addresses" table of IPv4 multicast addresses, uses a mixture of zero-filled and zero-suppressed octet values.)

Meanwhile, a very popular implementation of IP networking went off in its own direction. 4.2BSD introduced a function inet_aton(), whose job was to interpret character strings as IP addresses. It interpreted both of the syntaxes mentioned in [MTP] (see above): a single number giving the entire 32-bit address, and dot-separated octet values. It also interpreted two intermediate syntaxes: octet-dot-octet-dot-16bits, intended for class B addresses, and octet-dot-24bits, intended for class A addresses. It also allowed some

flexibility in how the individual numeric parts were specified: it allowed octal and hexadecimal in addition to decimal, distinguishing these radices by using the C language syntax involving a prefix "0" or "0x", and allowed the numbers to be arbitrarily long.

The 4.2BSD inet_aton() has been widely copied and imitated, and so is a de facto standard for the textual representation of IPv4 addresses. Nevertheless, these alternative syntaxes have now fallen out of use (if they ever had significant use). The only practical use that they now see is for deliberate obfuscation of addresses: giving an IPv4 address as a single 32-bit decimal number is favoured among people wishing to conceal the true location that is encoded in a URL. All the forms except for decimal octets are seen as non-standard (despite being quite widely interoperable) and undesirable.

2.1.2 Revision From IPv6 Work

When the textual format for IPv6 addresses was developed, part of the syntax involved representing an embedded IPv4 address by embedding an IPv4 address textual representation in the IPv6 textual format. [IPV6-AA-1], describing the IPv6 format for the first time, referred simply to "decimal values of the four low-order 8-bit pieces of the

address (standard IPv4 representation)", giving "::13.1.68.3" as an example of the format in practice.

[IPV6-AA-2] added an ABNF grammar, giving the first formal specification of IPv4 textual address syntax in the RFC series. This grammar showed dot-separated segments of one to three decimal digits each. Unfortunately, there were some errors in related bits of the grammar, and even with errors corrected the IPv6 address grammar was loose, syntactically permitting addresses of the wrong length. This, together with the similar looseness of the IPv4 address grammar (which would match "123.456.789.999"), left open the question of whether the grammar's acceptance of leading zeroes in IPv4 addresses was an intentional feature, an error, or deliberate looseness. [IPV6-AA-3], rather than correct the errors, withdrew the grammar.

The IPv6 effort also had an opportunity to advance the other branch of development of IPv4 address representation. [BSI-IPV6-1] doesn't attempt to modify inet_aton(), but defines a new function inet_pton(), which, in handling IPv4 addresses, accepts dotted decimal octets where each octet is encoded as "a one to three digit decimal number between 0 and 255". The variant forms traditionally accepted by inet_aton() are explicitly excluded. This definition is still not explicit about the handling of leading zeroes, but it seems to be intended to allow them, and it is being implemented accordingly.

2.1.3 Finale

So far we've seen two parallel versions of IPv4 address textual syntax, which we may label the IETF version and the BSD version. The difference has persisted for so long because the two are just sufficiently interoperable: they both handle in the same way the overwhelmingly dominant syntax, dotted decimal octets with leading zeroes suppressed. In all the other address forms they support they disagree: the IETF syntax makes nothing of most of the variants that BSD allows, and the two interpret differently a large group of representations involving leading zeroes, which is why zeroes have been mentioned so much in the foregoing history.

As of this writing, IPv4 addresses written with leading zeroes are de facto ambiguous. Although all IETF output that expresses an opinion

has consistently indicated that these should be interpreted as decimal, implementations that interpret them as octal are far too widespread to ignore. For this reason it is not safe to generate such addresses; the only way to generate an interoperable textual IPv4 address is to suppress leading zeroes. Overwhelmingly popular practice is, indeed, to avoid leading zeroes.

The most recent version of the URI syntax [URI] attempts to reconcile these variants in order to give a precise definition for acceptable IP address syntax in a URL. (Its predecessors had incorporated the traditionally ambiguous syntax by reference.) [URI] is the first RFC to require a completely rigorous definition of IP address syntax. The approach taken was to standardise the safe common subset of the IETF and BSD syntaxes, which achieves standardisation on IETF-like syntax while also retaining backward compatibility with existing BSD-based implementations.

This document, in section 3.1, presents the IPv4 address grammar from [URI].

2.2 IPv6 Presentation Format

The development of the IPv6 address presentation format has been simpler than the IPv4 history. The divergence between specification and implementation has been less significant, and there has been conscious effort to fully specify the format rather than leave it as oral tradition.

The first appearance of IPv6 address textual format in the RFC series is the specification of the format in [IPV6-AA-1]. This specification's relevant features are: a basic format of eight colon-separated 16-bit pieces; each piece represented in hexadecimal, with leading zeroes "not necessary" (examples are given both with and

without leading zeroes); optional use of "::", once in an address, to indicate a run of zero-valued 16-bit pieces; optional use of "standard IPv4 representation" for the least-significant 32 bits of the address.

Note that this doesn't say what the maximum length of a piece representation is, or whether "::" can be used in an address where all 16-bit pieces are given explicitly (the "::" would represent a

sequence of zero consecutive zero-valued pieces).

[IPV6-AA-2] didn't substantially modify the description of the syntax, but augmented it with an ABNF grammar. The grammar specified that a 16-bit piece could be represented in one to four case-insensitive hexadecimal digits, ruling out the use of more than four digits per piece. There were some errors in the grammar, making it inappropriate as a reference, and some looseness that makes it impossible to clear up any other syntactic uncertainty from it. These errors resulted from a "rough cut" grammar getting into the RFC largely unchanged, clearly without serious analysis.

[SMTP-2] gives an ABNF grammar for IPv6 address literals in email addresses. Unfortunately this grammar, also, is faulty and loose, but it concurs with the [IPV6-AA-2] grammar in requiring each piece to consist of one to four hexadecimal digits.

[IPV6-AA-3] dropped the faulty grammar from [IPV6-AA-2], and amended the format description to say that "::" represents "one or more" 16-bit pieces. This amended description leaves unclear only the issue of whether a 16-bit piece is permitted to be written with more than four hexadecimal digits; fortunately the intended answer (which is that it is not permitted) is known from the [IPV6-AA-2] and [SMTP-2] ABNF grammars. This document, in section 3.2, presents this syntax.

3 Syntax and Semantics

3.1 IPv4 Dotted Octet Format

A 32-bit IPv4 address is divided into four octets. Each octet is represented numerically in decimal, using the minimum possible number of digits (leading zeroes are not used, except in the case of 0 itself). The four encoded octets are given most-significant first, separated by period characters.

```
IPv4address = d8 "." d8 "." d8 "." d8

d8          = DIGIT                ; 0-9
            / %x31-39 DIGIT        ; 10-99
            / "1" 2DIGIT           ; 100-199
            / "2" %x30-34 DIGIT    ; 200-249
            / "25" %x30-35         ; 250-255
```

3.2 IPv6 Presentation Format

A 128-bit IPv6 address is divided into eight 16-bit pieces. Each piece is represented numerically in case-insensitive hexadecimal, using one to four hexadecimal digits (leading zeroes are permitted). The eight encoded pieces are given most-significant first, separated by colon characters. Optionally, the least-significant two pieces may instead be represented in IPv4 address textual format (the <IPv4address> production given above). Optionally, once in the address, a sequence of one or more consecutive zero-valued 16-bit pieces may be elided, omitting all their digits and leaving exactly two consecutive colons in their place to mark the elision.

```
IPv6address =                            6(h16 ":") ls32
            /                         "::" 5(h16 ":") ls32
            / [             h16 ] "::" 4(h16 ":") ls32
            / [ *1(h16 ":") h16 ] "::" 3(h16 ":") ls32
            / [ *2(h16 ":") h16 ] "::" 2(h16 ":") ls32
            / [ *3(h16 ":") h16 ] "::"   h16 ":"  ls32
            / [ *4(h16 ":") h16 ] "::"            ls32
            / [ *5(h16 ":") h16 ] "::"             h16
            / [ *6(h16 ":") h16 ] "::"

ls32        = h16 ":" h16 / IPv4address

h16         = 1*4HEXDIG
```

4 Recommendations

4.1 Be Stringent in What You Accept

Interpreting textual network addresses is a case where being liberal in what one receives is not a virtue. In addition to the well-known problem of interoperability testing against a liberal implementation leading to insufficiently conservative sending behaviour, variations on the address syntaxes tend to result in strings whose intended meaning is unclear. Since a misinterpreted network address is quite useless, whereas partial misinterpretation of most other things is forgivable, it is particularly important to reject any address whose interpretation is in question.

For backward compatibility, some applications will wish to continue supporting some of the variations discussed in section 2. New applications, however, SHOULD accept only the syntax given in section 3. Regardless of any alternative syntax that is supported, the standard syntax given in section 3 MUST be interpreted exactly as described there.

4.2 Generation of Representations of IPv6 Addresses

The standard format for IPv6 addresses has several options, granting some discretion in the choice of representation. The choices available are:

- which case to use for hexadecimal digits above 9;
- whether to use leading zeroes in the representation of 16-bit pieces whose upper four bits are all zero;
- whether to represent the least-significant 32 bits as two pieces in hexadecimal or in IPv4 format;
- whether to elide a sequence of zero-valued pieces, and which such sequence to elide.

For specific applications there may be needs that dictate some of these choices. For example, if laying out IPv6 addresses vertically in a table, comparison is eased by using a fixed format by including all leading zeroes and not eliding zero-valued pieces.

For general-purpose use, common practice is to use lowercase, use nearly the shortest possible representation, and to represent IPv4-compatible and IPv4-mapped addresses using the embedded IPv4

address representation. Experience has shown this format to be nearly optimal for human comprehension of an address presented in

isolation, and so is RECOMMENDED when there are no strong considerations promoting a different format. To generate this format:

- Use the embedded IPv4 address format for addresses in ::ffff:0:0/96 (IPv4-mapped addresses), and in ::/96 (IPv4-compatible addresses) except for :: (the unspecified address) and ::1 (the loopback address) which are not IPv4-compatible addresses.

- Omit all optional leading zeroes in the representations of 16-bit pieces.

- If there are any sequences of consecutive zero-valued pieces, elide the longest such sequence. In case of a tie, it seems to be most common to pick the leftmost candidate.

4.3 Delimitation

Textually-represented IPv4 and IPv6 addresses have a sufficiently narrow format that delimitation is rarely a problem. In human-readable text they look sufficiently like words that additional delimitation is usually not required; adjacent punctuation mostly wouldn't be a valid character in the address, and even with punctuation that can appear in the addresses (period and colon) trailing punctuation creates no ambiguity due to the restricted use of punctuation in the addresses.

A significant area where there is a delimitation issue is when an IP address is presented together with an alphanumeric subaddress such as a TCP port number. Some applications separate an IP address and port number using a period, which, particularly in the case of IPv4, makes the port number visually appear to be part of the address. This is particularly tricky to read if a bare IP address without port number might appear in the same context. Some applications use a colon to separate IP address and port number, which is good for IPv4 but in IPv6 it creates the same kind of problem that the period did in IPv4, and can actually give an ambiguous result if a bare IPv6 address is permitted in the same context. Applications SHOULD, therefore, pick

some other character to separate IP addresses and port numbers; BIND, for example, uses "#". "/" is not recommended, due to a clash with address prefix syntax.

In contexts where an IP address needs to be distinguished from similar-looking data that can appear in the same place, there is precedent for enclosing an IP address in brackets ("[]") as a distinguisher. Precedents are email addresses [SMTP-2] and URIs [URI].

5 Security Considerations

In a network protocol, representation of network addresses in a textual format raises no inherent issues over representation in a binary format. Care should be taken to ensure that textual addresses are parsed safely, so that bad syntax will not cause unwanted behaviour. Where a textually-represented address is expected, it should be decoded by a subroutine that will decode only the expected address format and will not do anything (besides report an error) if given some other input such as a host name.

In applications, the capability for the user to specify a network node by address as well as by name is both powerful and potentially dangerous. If an application does not intend to let the user specify absolutely any network resource, then it should either have only a more restrictive means of identifying network nodes or apply reasonableness checks on the address that the user enters.

6 Acknowledgements

This document is a spin-off from the development of [URI], which was the first RFC to give such a precise definition of IP address textual syntax as is given here. The ABNF rules in section 3 were developed collaboratively by Roy T. Fielding (author of [URI]) and the author of this document.

Brian Carpenter confessed to being the perpetrator of the faulty grammar that got into [IPV6-AA-2], and supplied a copy of the historically-interesting email message in which he originally proposed it. He also made some useful comments.

7 Normative References

[ABNF] D. Crocker, Ed., P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[REQ-TERM] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

8 Informative References

[BSI-IPV6-1] R. Gilligan, S. Thomson, J. Bound, W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 2133, April 1997.

[HOST-TBL-2] E.J. Feinler, K. Harrenstien, Z. Su, V. White, "DoD Internet host table specification", RFC 810, Mar-01-1982.

[HREQ-APP] R.T. Braden, "Requirements for Internet hosts - application and support", STD 3, RFC 1123, Oct-01-1989.

[IPV4] J. Postel, "Internet Protocol", RFC 791, Sep-01-1981.

[IPV4-NUMB] J. Postel, "Assigned numbers", RFC 790, Sep-01-1981.

[IPV6-AA-1] R. Hinden, S. Deering, Eds., "IP Version 6 Addressing Architecture", RFC 1884, December 1995.

[IPV6-AA-2] R. Hinden, S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[IPV6-AA-3] R. Hinden, S. Deering, "Internet Protocol Version 6 (IPv6) Addressing Architecture", RFC 3513, April 2003.

[MTP] S. Sluizer, J. Postel, "Mail Transfer Protocol", RFC 780, May-01-1981.

[NCP-TCP] J. Postel, "NCP/TCP transition plan", RFC 801, Nov-01-1981.

[SMTP-1] J. Postel, "Simple Mail Transfer Protocol", STD 10, RFC 821, Aug-01-1982.

[SMTP-2] J. Klensin, Ed., "Simple Mail Transfer Protocol", RFC 2821, April 2001.

[URI] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", draft-fielding-uri-rfc2396bis-01, March 3, 2003.

9 Author's Address

Andrew Main
Black Ops Ltd
Flat 2
84 Isledon Road
London
N7 7JS
United Kingdom

Phone: +44 7887 945779
EMail: zefram@fysh.org