Network Working Group
Request for Comments: 5511
Category: Standards Track

A. Farrel
Old Dog Consulting

April 2009

Routing Backus-Naur Form (RBNF): A Syntax Used to Form Encoding Rules in Various Routing Protocol Specifications

Status of This Memo

Copyright Notice

Abstract

Several protocols have been specified in the Routing Area of the IETF using a common variant of the Backus-Naur Form (BNF) of representing message syntax. However, there is no formal definition of this version of BNF.

There is value in using the same variant of BNF for the set of protocols that are commonly used together. This reduces confusion and simplifies implementation.

Updating existing documents to use some other variant of BNF that is already formally documented would be a substantial piece of work.

This document provides a formal definition of the variant of BNF that has been used (that we call Routing BNF) and makes it available for use by new protocols.

Table of Contents

1. Introduction

Backus-Naur Form (BNF) has been used to specify the message formats of several protocols within the Routing Area of the IETF. Unfortunately, these specifications are not based on any specific formal definition of BNF, and they differ slightly from the definitions provided in other places.

It is clearly valuable to have a formal definition of the syntax-defining language that is used. It would be possible to convert all existing specifications to use an established specification of BNF (for example, Augmented BNF or ABNF [RFC5234]); however, this would require a lot of work. It should be noted that in ABNF the terminals are integers (characters/bytes), while in the BNF form used to define message formats, the terminals are "objects" (some kind of message elements, but not individual bytes or characters) or entire "messages". This means that converting existing specifications to use an established BNF specification would also require extensions to that BNF specification.

On the other hand, the variant of BNF used by the specifications in question (which is similar to a subset of Extended BNF [EBNF]) is consistent and has only a small number of constructs. It makes sense, therefore, to provide a definition of this variant of BNF to allow ease of interpretation of existing documents and to facilitate the development of new protocol specifications using the same variant of BNF. A specification will also facilitate automated verification of the formal definitions used in future documents.

This document provides such a specification and names the BNF variant Routing BNF (RBNF).

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.2. Existing Uses

The first notable use of the variant of BNF that concerns us is in the specification of the Resource Reservation Protocol (RSVP) [RFC2205]. RSVP has been extended for use in Multiprotocol Label Switching (MPLS) networks to provide signaling for Traffic Engineering (TE) [RFC3209], and this has been developed for use as the signaling protocol in Generalized MPLS (GMPLS) networks [RFC3473].

Each of these three uses of RSVP has given rise to a large number of specifications of protocol extensions to provide additional features over and above those in the base documents. Each new feature is defined in its own document using the common variant of BNF.

New protocols have also been specified using the same variant of BNF. This has arisen partly because the developers were familiar with the BNF used in [RFC2205], etc., but also because of the overlap between the protocols, especially with respect to the network objects controlled and operated.

Notable among these additional protocols are the Link Management Protocol (LMP) [RFC4204] and the Path Computation Element Protocol (PCEP) [RFC5440]. In both cases, further documents that specify protocol extensions also use the same variant of BNF.

1.3. Applicability Statement

RBNF as defined in this document is primarily applicable for the protocols listed in the previous section. The specification may be used to facilitate the interpretation of the pre-existing RFCs that are referenced. It should also be used in the specification of extensions to those protocols.

RBNF could also be used for the specification of new protocols. This is most appropriate for the development of new protocols that are closely related to those that already use RBNF. For example, PCEP is closely related to RSVP-TE, and when it was developed, the PCE working group chose to use the same form of BNF as was already used in the RSVP-TE specifications.

If a wholly new protocol is being developed and is not related to a protocol that already uses RBNF, the working group should consider carefully whether to use RBNF or to use a more formally specified and broader form of BNF such as ABNF [RFC5234].

The use of RBNF to specify extensions to protocols that do not already use RBNF (i.e., that use some other form of BNF) is not recommended.

2. Formal Definitions

The basic building blocks of BNF are rules and operators. At its simplest form, a rule in the context we are defining is a protocol object that is traditionally defined by a bit diagram in the protocol specification. Further and more complex rules are constructed by

combining other rules using operators. The most complex rule is the message that is constructed from an organization of protocol objects as specified by the operators.

An RBNF specification consists of a sequence of rule definitions using the operators defined in Section 2.2. One rule may be constructed from a set of other rules using operators. The order of definition of rules does not matter. That is, the subordinate rules MAY be defined first and then used in subsequent definitions of further rules, or the top-level rules MAY be defined first followed by a set of definitions of the subordinate rules.

Rule definitions are read left-to-right on any line, and the lines are read top-to-bottom on the page. This becomes particularly important when considering sequences of rules and operators.

2.1. Rule Definitions

No semantics should be assumed from special characters used in rule names. For example, it would be wrong to assume that a rule carries a decimal number because the rule name begins or ends with the letter "d". However, individual specifications MAY choose to assign rule names in any way that makes the human interpretation of the rule easier.

2.1.1. Rule Name Delimitation

All rule names are enclosed by angle brackets ("<" and ">"). Rule names MAY include any printable characters, but MUST NOT include tabs or line feeds/breaks.

Example:

```
<Path Message>
```

2.1.2. Objects

The most basic (indivisible) rule is termed an object. The definition of an object is derived from its context.

Objects are typically named in uppercase. They do not usually use spaces within the name, favoring underbars ("_").

Example:

```
<SENDER_TEMPLATE>
```

2.1.3. Constructs

Rules that are constructed from other rules using operators are termed constructs.

Constructs are named in lowercase, although capitals are commonly used to indicate acronyms. Spaces and hyphens are used between words within names.

Example:
<sender descriptor>

2.1.4. Messages

The final objective is the definition of messages. These are rules that are constructed from objects and constructs using operators. The only syntactic difference between a message and a construct is that no other rule is typically constructed from a message.

Messages are typically named in title case.

Example:
<Path Message>

2.2. Operators

Operators are used to build constructs and messages from objects and constructs.

2.2.1. Assignment

Assignment is used to form constructs and messages.

Meaning:
The named construct or message on the left-hand side is defined to be set equal to the right-hand side of the assignment.

Encoding:
colon, colon, equal sign ("::=")

Example:
<WF flow descriptor> ::= <FLOWSPEC>

Note:
The left-hand side of the assignment and the assignment operator MUST be present on the same line.

2.2.2. Concatenation

Objects and constructs can be combined as a sequence to form a new construct or a message.

Meaning:
The objects or constructs MUST be present in the order specified. The order of reading RBNF is stated in Section 2.

Encoding:
A sequence of objects and constructs usually separated by spaces. The objects in a sequence MAY be separated by line breaks.

Example:
<SE flow descriptor> ::= <FLOWSPEC> <filter spec list>

Note:
See Section 2.3.3 for further comments on the ordering of objects and constructs.

2.2.3. Optional Presence

Objects and constructs can be marked as optionally present.

Meaning:
The optional objects or constructs MAY be present or absent within the assignment. Unless indicated as optional, objects and constructs are mandatory and MUST be present. The optional operator can also be nested to give a hierarchical dependency of presence as shown in the example below.

Encoding:
Contained in square brackets ("[" and "]").

Example:

```
<PathTear Message> ::= <Common Header> [ <INTEGRITY> ]
                       <SESSION> <RSVP_HOP>
                       [ <sender descriptor> ]
```

Example of nesting:
The optional operator can be nested. For example,

```
<construct> ::= <MAND> [ <OPT_1> [ <OPT_2> ] ]
```

In this construction, the object OPT_2 can only be present if OPT_1 is also present.

Note:
The set of objects and constructs within the same pair of square brackets is treated as a unit (an unnamed construct). This means that when multiple objects and constructs are included within the same pair of square brackets, all MUST be included when one is included, unless nested square brackets are used as in the previous example.

2.2.4. Alternatives

Choices can be indicated within assignments.

Meaning:
Either one rule or the other MUST be present.

Encoding:
The pipe symbol ("|") is used between the objects or constructs that are alternatives.

Example:

```
<flow descriptor list> ::= <FF flow descriptor list>
                         | <SE flow descriptor>
```

Notes:

1. Use of explicit grouping (Section 2.2.6) is RECOMMENDED to avoid confusion. Implicit grouping using line breaks (Section 2.3.2) is often used, but gives rise to potential misinterpretation and SHOULD be avoided in new definitions.

2. Multiple members of alternate sets can give rise to confusion. For example:

   ```
   <flow descriptor list> ::=  <empty> |
                        <flow descriptor list> <flow descriptor>
   ```

   could be read to mean that an instance of <flow descriptor> must be present or that it is optional.

   To avoid this type of issue, explicit grouping (see Section 2.2.6), or an intermediary MUST be used in all new documents (existing uses are not deprecated, and automatic parsers need to handle existing RFCs). See also Section 2.4 for a description of precedence rules.

   Thus:

   ```
   <construct> ::= <ALT_A> <ALT_B> | <ALT_C> <ALT_D>
   ```

is not allowed in new documents and MUST be presented using grouping or using an intermediary construct. For example, and depending on intended meaning:

```
<construct> ::= ( <ALT_A> <ALT_B> ) | ( <ALT_C> <ALT_D> )

or

<construct> ::= <ALT_A> ( <ALT_B> | <ALT_C> ) <ALT_D>
```

or

```
<intermediary X> ::= <ALT_A> <ALT_B>
<intermediary Y> ::= <ALT_C> <ALT_D>
<construct> ::= <intermediary X> | <intermediary Y>

or

<intermediary Z> ::= <ALT_B> | <ALT_C>
<construct> ::= <ALT_A> <intermediary Z> <ALT_D>
```

2.2.5. Repetition

It could be the case that a sequence of identical objects or constructs is required within an assignment.

Meaning:
MAY repeat the preceding object, intermediate construct, or construct.

Encoding:
Three dots ("...").

Example:

```
<Path Message> ::= <Common Header> [ <INTEGRITY> ]
                   <SESSION> <RSVP_HOP>
                   <TIME_VALUES>
                   [ <POLICY_DATA> ... ]
                   [ <sender descriptor> ]
```

Notes:

1. A set of zero or more objects or constructs can be achieved by combining with the Optional concept as shown in the example above.

2. Sequences can also be encoded by building a recursive construct using the Alternative operator. For example:

```
<sequence> ::= <OBJECT> |
               ( <OBJECT> <sequence> )
```

3. Repetition can also be applied to a component of an assignment to indicate the optional repetition of that component. For example, the Notify message in [RFC3473] is defined as follows:

```
<Notify message> ::=
                 <Common Header> [<INTEGRITY>]
                 [ [<MESSAGE_ID_ACK> | <MESSAGE_ID_NACK>] ... ]
                 [ <MESSAGE_ID> ]
                 <ERROR_SPEC> <notify session list>
```

In this example, there is a sequence of zero or more instances of [<MESSAGE_ID_ACK> | <MESSAGE_ID_NACK>]. One could argue that the use of grouping (see Section 2.2.6) or a recursive construct (see Note 2, above) would be more clear.

2.2.6. Grouping

Meaning:
A group of objects or constructs to be treated together. This notation is not mandatory but is RECOMMENDED for clarity. See Section 2.4 on Precedence.

Encoding:
Round brackets ("(" and ")") enclosing a set of objects, constructs, and operators.

Example:

```
<group> ::= ( <this> <that> )
```

Notes:

1. The precedence rule in Section 2.4 means that the use of grouping is not necessary for the formal interpretation of the BNF representation. However, grouping can make the BNF easier to parse unambiguously. Either grouping or an intermediate construct MUST be used for multi-alternates (Section 2.2.4).

2. Line breaks (Section 2.3.2) are often used to clarify grouping as can be seen in the definition of <sequence> in Section 2.2.5, but these are open to misinterpretation, and explicit grouping is RECOMMENDED.

3. A practical alternative to grouping is the definition of intermediate constructs as illustrated in Note 2 of Section 2.2.4.

2.3. Editorial Conventions

2.3.1. White Space

White space (that is space characters) between operators, objects, and constructs is ignored but SHOULD be used for readability.

2.3.2. Line Breaks

Line breaks within an assignment are ignored but SHOULD be used for readability.

Line breaks are often used to imply grouping within the precedence rules set out in Section 2.4, but explicit grouping (Section 2.2.6) or intermediary constructs (Section 2.2.4) SHOULD be used in new definitions.

A line break MUST NOT be present between the left-hand side of an assignment and the assignment operator (see Section 2.2.1).

New assignments (i.e., new construct or message definitions) MUST begin on a new line.

2.3.3. Ordering

The ordering of objects and constructs in an assignment is explicit.

Protocol specifications MAY opt to state that ordering is only RECOMMENDED. In this case, elements of a list of objects and constructs MAY be received in any order.

2.4. Precedence

Precedence is the main opportunity for confusion in the use of this BNF. In particular, the use of alternatives mixed with concatenations can give rise to different interpretations of the BNF. Although precedence can be deduced from a "proper" reading of the BNF using the rules defined above and the precedence ordering shown below, authors are strongly RECOMMENDED to use grouping (Section 2.2.6) and ordering (Section 2.3.3) to avoid cases where the reader would otherwise be required to understand the precedence rules.

Automated readers are REQUIRED to parse rules correctly with or without this use of grouping.

The various mechanisms described in the previous sections have the following precedence, from highest (binding tightest) at the top, to lowest (and loosest) at the bottom:

```
   objects, constructs
   repetition
   grouping, optional
   concatenation
   alternative
```

Note:
Precedence is the main opportunity for confusion in the use of BNF. Authors are strongly RECOMMENDED to use grouping (Section 2.2.6) in all places where there is any scope for misinterpretation even when the meaning is obvious to the authors.

Example:

An example of the confusion in precedence can be found in Section 3.1.4 of [RFC2205] and is mentioned in Section 2.2.4.

```
<flow descriptor list> ::=  <empty> |
                <flow descriptor list> <flow descriptor>
```

The implementer MUST decide which of the following is intended:

a.
```
<flow descriptor list> ::= <empty> |
                      ( <flow descriptor list> <flow descriptor> )
```

b.
```
<flow descriptor list> ::= ( <empty> | <flow descriptor list> )
                             <flow descriptor>
```

The line break MAY be interpreted as implying grouping, but that is not an explicit rule. However, the precedence rules say that concatenation has higher precedence than the Alternative operator. Thus, the text in [RFC2205] SHOULD be interpreted as shown in formulation a.

Similarly (from the same section of [RFC2205]):

```
  <flow descriptor list> ::=
                  <FLOWSPEC>  <FILTER_SPEC>  |
                  <flow descriptor list> <FF flow descriptor>
```

SHALL be interpreted as:

```
  <flow descriptor list> ::=
                ( <FLOWSPEC> <FILTER_SPEC> ) |
                ( <flow descriptor list> <FF flow descriptor> )
```

The use of explicit grouping or intermediary constructs is strongly RECOMMENDED in new text to avoid confusion.

3. Automated Validation

RBNF would be appropriate for verification using automated validation tools. Validation tools need to be able to check for close conformance to the rules expressed in this document to be useful for verifying new documents, but should also be able to parse RBNF as used in existing RFCs. No tools are known at this time.

4. Security Considerations

This document does not define any network behavior and does not introduce or seek to solve any security issues.

It may be noted that clear and unambiguous protocol specifications reduce the likelihood of incompatible or defective implementations that might be exploited in security attacks.

5. Acknowledgments

Thanks to Magnus Westerlund, Nic Neate, Chris Newman, Alfred Hoenes, Lou Berger, Julien Meuric, Stuart Venters, Tom Petch, Sam Hartman, and Pasi Eronen for review and useful comments.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

6.2. Informative References

[RFC2205] Braden, R., Ed., Zhang, L., Berson, S., Herzog, S., and S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205, September 1997.

[RFC3209] Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V., and G. Swallow, "RSVP-TE: Extensions to RSVP for LSP Tunnels", RFC 3209, December 2001.

[RFC3473] Berger, L., Ed., "Generalized Multi-Protocol Label Switching (GMPLS) Signaling Resource ReserVation Protocol-Traffic Engineering (RSVP-TE) Extensions", RFC 3473, January 2003.

[RFC4204] Lang, J., Ed., "Link Management Protocol (LMP)", RFC 4204, October 2005.

[RFC5234] Crocker, D., Ed., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5440] Vasseur, JP., Ed., and JL. Le Roux, Ed., "Path Computation Element (PCE) Communication Protocol (PCEP)", RFC 5440, March 2009.

[EBNF] ISO/IEC 14977, "Information technology -- Syntactic metalanguage -- Extended BNF", 1996.

Author's Address

Adrian Farrel
Old Dog Consulting

EMail: adrian@olddog.co.uk