Mobile Ad hoc Networking (MANET)
Internet-Draft
Intended status: Standards Track
Expires: December 26, 2008

T. Clausen
LIX, Ecole Polytechnique, France
C. Dearlove
BAE Systems Advanced Technology Centre
J. Dean
Naval Research Laboratory
C. Adjih
INRIA Rocquencourt

June 24, 2008

Generalized MANET Packet/Message Format
draft-ietf-manet-packetbb-13

Status of This Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on December 26, 2008.

Abstract

This document specifies a packet format capable of carrying multiple messages that may be used by mobile ad hoc network routing protocols.

Table of Contents

1. Introduction 3
2. Notation and Terminology 4
2.1. Notation 4
2.2. Terminology 5
3. Applicability Statement 5
4. Protocol Overview and Functioning 7
5. Syntactical Specification 7
5.1. Packets 7
5.2. Messages 8
5.3. Address Blocks 10
5.4. TLVs and TLV Blocks 13
5.4.1. TLVs 14
5.4.2. TLV Inclusion and Constraints 17
5.5. Malformed Elements 18
6. IANA Considerations 18
6.1. Expert Review: Evaluation Guidelines 18
6.2. Message Types 21
6.2.1. Message Type Specific TLV Registry Creation 21
6.3. Packet TLV Types 21
6.3.1. Packet TLV Type Extension Registry Creation 21
6.4. Message TLV Types 21
6.4.1. Message TLV Type Extension Registry Creation 22
6.5. Address Block TLV Types 22
6.5.1. Address Block TLV Type Extension Registry Creation . . 23
7. Security Considerations 23
7.1. Authentication and Integrity Suggestions 24
7.2. Confidentiality Suggestions 24
8. References 25
8.1. Normative References 25
8.2. Informative References 26
Appendix A. Intended Usage 26
Appendix B. Multiplexing and Demultiplexing 27
Appendix C. Examples 28
C.1. Address Block Examples 28
C.2. TLV Examples 30
Appendix D. Illustrations 32
D.1. Packet 33
D.2. Message 36
D.3. Message Body 42
D.4. Address Block 43
D.5. TLV Block 50
D.6. TLV 51
Appendix E. Complete Example 56
Appendix F. Contributors 57
Appendix G. Acknowledgments 58

1. Introduction

This document specifies the syntax of a packet format designed to carrying multiple messages for information exchange between MANET (Mobile Ad hoc NETwork) routers. Messages consist of a message header, which is designed for control of message dissemination, and a message body, which contains protocol information. Only the syntax of the packet and messages is specified. All syntactical entities, including packets and messages, are specified using a regular expression derived notation.

This document specifies:

- A packet format, allowing zero or more messages to be contained within a single transmission. A packet with zero messages may be sent in case the only information to exchange is contained in the packet header.

- A message format, where a message is composed of a message header and a message body.

- A message header format, which contains information which may be sufficient to allow a protocol using this specification to make processing and forwarding decisions.

- A message body format, containing attributes associated with the message or the originator of the message, as well as blocks of addresses, or address prefixes, with associated attributes.

- An address block format, where an address block represents sets of addresses, or address prefixes, in a compact form with aggregated addresses.

- A generalized type-length-value (TLV) format representing attributes. Each TLV can be associated with a packet, a message, or one or more addresses or address prefixes in a single address block. Multiple TLVs can be included and each associated with a packet, a message, and with the same, different or overlapping sets of addresses or address prefixes.

The specification has been explicitly designed with the following properties, listed in no particular order, in mind:

Parsing logic - the regular expression derived notation used in this specification facilitates generic, protocol independent, parsing logic.

Extensibility - packets and messages defined by a protocol using this specification are extensible by defining new message types and new TLVs. Protocols using this specification will be able to correctly identify and skip such new message types and TLVs, while correctly parsing the remainder of the packet and message.

Efficiency - when reported addresses share common bit sequences (e.g. address prefixes or IPv6 interface identifiers) the address block representation allows for a compact representation. Compact message headers are ensured through permitting inclusion of only required message header elements. The structure of packet and message encoding allows parsing, verifying, and identifying individual elements in a single pass.

Separation of forwarding and processing - a protocol using this specification can be designed such that duplicate detection and controlled scope message forwarding decisions can be made using information contained in the message header, without processing the message body.

2. Notation and Terminology

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

Additionally, this document uses the notation in Section 2.1, and the terminology in Section 2.2.

2.1. Notation

This specification uses a derivative of the regular expression language from [SingleUNIX], [POSIX] or [ECMAScript], using only the following notation:

Element - A group of any number of consecutive bits which, together, forms a syntactic entity, represented using the notation <foo>.

<foo> - If <foo> is an unsigned integer field then <foo> is also used to represent the value of that field.

bar - A variable, usually obtained through calculations based on the value(s) of element(s). Variables are introduced into the specification solely as a means to clarify the description.

:= - Indicates that the element or variable on the left hand side is defined as the expression on the right hand side.

(<element1><element2>) - Indicates a grouping of the elements enclosed by the parentheses.

? - Zero or one occurrences of the preceding element or group.

* - Zero or more occurrences of the preceding element or group.

address-length - A variable whose value is the length of an address in octets, it is 4 if using IPv4, or 16 if using IPv6.

2.2. Terminology

This document uses the following terminology:

Packet - The top level entity in this specification. A packet contains a packet header and zero or more messages.

Message - The fundamental entity carrying protocol information, in the form of address objects and TLVs.

Address - A number of octets of the same length as the source IP address in the IP datagram carrying the packet. The meaning of an address is defined by the protocol using this specification.

Address Prefix - An address plus a prefix length, with the prefix length being a number of address bits measured from the left/most significant end of the address.

Address Object - Either an address, or an address prefix, as specified in an address block in this specification.

TLV - A Type-Length-Value structure. This is a generic way in which an attribute can be represented and correctly parsed, without the parser having to understand the attribute.

3. Applicability Statement

This specification describes a generic packet format, designed for use by MANET routing protocols. The specification has been inspired by and extended from that used by OLSR (The Optimized Link State Routing protocol) [RFC3626].

MANETs are, commonly though not exclusively, characterized as being able to run over wireless network interfaces of limited to moderate capacity. MANETs are therefore less tolerant of wasted transmitted

octets than are most wired networks. This specification thus represents a tradeoff between sometimes competing attributes, specifically efficiency, extensibility, and ease of use.

Efficiency is supported by reducing packet size and by allowing multiple disjoint messages in a single packet. Reduced packet size is primarily supported by address aggregation, optional packet and message header fields, and optional fields in address blocks and TLVs. Supporting multi-message packets allows a reduction in the number of packets, each of which can incur significant bandwidth costs from transport, network and lower layers.

This specification provides both external and internal extensibility. External extensibility is supported by the ability to add packet TLVs and to define new message types. Internal extensibility is supported by the ability to add message TLVs and address block TLVs to existing messages. Protocols can define new TLV types, and hence the contents of their value fields (see Section 6.1), and new message types. Protocols can also reuse message and TLV type definitions from other protocols which also use this specification.

This specification aims at being sufficiently expressive and flexible to be able to accommodate both different classes of MANET routing protocols (e.g. proactive, reactive and hybrid routing protocols), as well as extensions thereto. Having a common packet and message format, and a common way of representing IP addresses and associated attributes, allows generic parsing code to be developed, regardless of the algorithm used by the routing protocol.

All addresses within a message are assumed to be of the same size, deduced from the IP header. In the case of mixed IPv6 and IPv4 addresses, IPv4 addresses can be represented as IPv4-mapped IPv6 addresses as specified in [RFC4291]. Packets may be unicast or multicast, and may use any appropriate transport protocol, or none.

The messages defined by this specification are designed to carry MANET routing protocol signaling between MANET routers. This specification includes elements which can support limited diffusion (flooding), as well as being usable for point to point delivery of MANET routing protocol signaling in a multi-hop network.

A MANET routing protocol using the packet format defined by this specification can constrain the syntax (for example requiring a specific set of message header fields) that the protocol will employ. Protocols with such restrictions need not be able to parse all possible packet structures as defined by this document but must be coherent in packet generation and reception of packets of which they define. If a protocol specifies which elements are included, then

direct indexing of the appropriate fields is possible, dependant on the syntax restrictions imposed by the protocol. Such protocols may have more limited extensibility.

4. Protocol Overview and Functioning

This specification does not describe a protocol. It describes a packet format, which may be used by any mobile ad hoc network routing protocol.

5. Syntactical Specification

This section provides syntactical specification of a packet, represented by the element <packet> and the elements from which it is composed. The specification is given using the notation in Section 2.1. Illustrations of specified elements are given in Appendix D.

This format uses network byte order (most significant octet first) for all fields. The most significant bit in an octet is numbered bit 0, and the least significant bit of an octet is numbered bit 7 [Stevens].

5.1. Packets

<packet> is defined by:

```
    <packet> := <pkt-header>
                <message>*
```

where <message> is as defined in Section 5.2. Successful parsing is terminated when all octets of the packet (as defined by the datagram containing the packet) are used.

<pkt-header> is defined by:

```
    <pkt-header> := <version>
                    <pkt-flags>
                    <pkt-seq-num>?
                    <tlv-block>?
```

where:

<version> is a 4 bit unsigned integer field and specifies the version of the specification on which the packet and the contained messages are constructed. This document specifies version 0.

<pkt-flags> is a 4 bit field, specifying the interpretation of the remainder of the packet header:

bit 0 (phasseqnum): If cleared ('0'), then <pkt-seq-num> is not included in the <pkt-header>. If set ('1'), then <pkt-seq-num> is included in the <pkt-header>.

bit 1 (phastlv): If cleared ('0'), then <tlv-block> is not included in the <pkt-header>. If set ('1'), then <tlv-block> is included in the <pkt-header>.

bit 2 (pnouniord): If cleared ('0'), then the packet TLV block MUST conform to the constraints in Section 5.4.2. If set ('1'), then the packet TLV block is not subject to the constraints in Section 5.4.2. Additional constraints MAY be imposed by a protocol using this specification.

bit 3: Is RESERVED, and SHOULD be cleared ('0') on transmission, and SHOULD be ignored on reception.

<pkt-seq-num> is omitted if the phasseqnum pkt-flag is cleared ('0'), otherwise is a 16 bit unsigned integer field, specifying a packet sequence number.

<tlv-block> is omitted if the phastlv flag is cleared ('0'), and is otherwise as defined in Section 5.4.

5.2. Messages

Packets may, in addition to the packet header, contain one or more messages. Messages contain:

- A message header.
- A message TLV block that contains zero or more TLVs, associated with the whole message.
- Zero or more address blocks, each containing one or more address objects.
- An address TLV block, containing zero or more TLVs, following each address block, through which addresses can be associated with additional attributes.

<message> is defined by:

```
      <message>    := <msg-header>
                      <tlv-block>
                      (<addr-block><tlv-block>)*

      <msg-header> := <msg-type>
                      <msg-flags>
                      <msg-size>
                      <msg-orig-addr>?
                      <msg-hop-limit>?
                      <msg-hop-count>?
                      <msg-seq-num>?
```

where:

<tlv-block> is as defined in Section 5.4.

<addr-block> is as defined in Section 5.3.

<msg-type> is an 8 bit unsigned integer field, specifying the type of the message.

<msg-flags> is an 8 bit field, specifying the interpretation of the remainder of the message header:

bit 0 (mhasorig): If cleared ('0'), then <msg-orig-addr> is not included in the <msg-header>. If set ('1'), then <msg-orig-addr> is included in the <msg-header>.

bit 1 (mhashoplimit): If cleared ('0'), then <msg-hop-limit> is not included in the <msg-header>. If set ('1'), then <msg-hop-limit> is included in the <msg-header>.

bit 2 (mhashopcount): If cleared ('0'), then <msg-hop-count> is not included in the <msg-header>. If set ('1'), then <msg-hop-count> is included in the <msg-header>.

bit 3 (mhasseqnum): If cleared ('0'), then <msg-seq-num> is not included in the <msg-header>. If set ('1'), then <msg-seq-num> is included in the <msg-header>.

bit 4 (mistypedep): If cleared ('0'), then the message sequence number in the message is type-independent. If set ('1'), then the message sequence number contained in the message is type dependent (the message originator maintains a sequence number specific to <msg-type>). This msg-flag MUST be cleared ('0') if the mhasorig msg-flag is cleared ('0').

bit 5 (mnouniord): If cleared ('0'), then the message TLV block and all address TLV blocks in the message MUST conform to the constraints in Section 5.4.2. If set ('1'), then the message TLV block and all address TLV blocks in the message are not subject to the constraints in Section 5.4.2. Additional constraints MAY be imposed by a protocol using this specification.

bit 6-7: Are RESERVED, and SHOULD each be cleared ('0') on transmission, and SHOULD be ignored on reception.

<msg-size> is a 16 bit unsigned integer field, specifying the number of octets that make up the <message>, including the <msg-header>.

<msg-orig-addr> is omitted if the mhasorig msg-flag is cleared ('0'), otherwise is an identifier with length equal to address-length, which can serve to uniquely identify the node that originated the message.

<msg-hop-limit> is omitted if the mhashoplimit msg-flag is cleared ('0'), otherwise is an 8 bit unsigned integer field, which can contain the maximum number of hops that the message should be further transmitted.

<msg-hop-count> is omitted if the mhashopcount msg-flag is cleared ('0'), otherwise is an 8 bit unsigned integer field, which can contain the number of hops that the message has traveled.

<msg-seq-num> is omitted if the mhasseqnum msg-flag is cleared ('0'), otherwise is a 16 bit unsigned integer field, which can contain a sequence number, generated by the message's originator node.

5.3. Address Blocks

An address block can specify one or more addresses. It can also specify prefix lengths that can be applied to all addresses in the address block. This allows an address block to specify either addresses or address prefixes. A protocol may specify that an address with a maximum prefix length (equal to the address length, in bits) is considered to be an address, rather than an address prefix, thus allowing an address block to contain a mixture of addresses and address prefixes. The common term "address object" is used in this specification to cover both of these; note that an address object in an address block always includes the prefix length, if present.

An address is specified as a sequence of address-length octets of the form head:mid:tail. There are no semantics associated with head, mid

or tail; this representation is solely to allow aggregation of addresses, which often have common parts (e.g. common prefixes or multiple IPv6 addresses on the same interface). An address block contains an ordered set of addresses all sharing the same head and the same tail, but having individual mids. Independently, head and tail may be empty, allowing for representation of address objects which do not have common heads or common tails. Detailed examples of address blocks are included in Appendix C.1.

An address block can specify address prefixes:

o with a single prefix length for all address prefixes; OR

o with a prefix length for each address prefix.

<address-block> is defined by:

```
    <address-block> := <num-addr>
                       <addr-flags>
                       (<head-length><head>?)?
                       (<tail-length><tail>?)?
                       <mid>*
                       <prefix-length>*
```

where:

<num-addr> is an 8 bit unsigned integer field containing the number of addresses represented in the address block, which MUST NOT be zero.

<addr-flags> is an 8 bit field specifying the interpretation of the remainder of the address block:

bit 0 (ahashead): If cleared ('0'), then <head-length> and <head> are not included in the <address-block>. If set ('1'), then <head-length> is included in the <address-block>, and <head> is included in the <address-block> unless <head-length> is zero.

bit 1 (ahasfulltail) and bit 2 (ahaszerotail): Are interpreted according to Table 1. A combination not shown in that table MUST NOT be used.

bit 3 (ahassingleprelen) and bit 4 (ahasmultiprelen): Are interpreted according to Table 2. A combination not shown in that table MUST NOT be used.

bits 5-7: Are RESERVED, and SHOULD each be cleared ('0') on transmission, and SHOULD be ignored on reception.

ahasfulltail	ahaszerotail	<tail-length>	<tail>
0	0	not included	not included
1	0	included	included unless <tail-length> is zero
0	1	included	not included

Table 1: Interpretation of the ahasfulltail and ahaszerotail addr-flags

ahassingle prelen	ahasmulti prelen	number of <prefix-length> fields	prefix length of the nth address prefix, in bits
0	0	0	8 * address-length
1	0	1	<prefix-length>
0	1	<num-addr>	nth <prefix-length>

Table 2: Interpretation of the ahassingleprelen and ahasmultiprelen addr-flags

<head-length> if present is an 8 bit unsigned integer field, which contains the number of octets in the head of all of the addresses in the address block.

head-length is a variable, defined to equal <head-length> if present, or 0 otherwise.

<head> is omitted if head-length is equal to 0, otherwise it is a field of the head-length leftmost octets common to all the addresses in the address block.

<tail-length> if present is an 8 bit unsigned integer field, which contains the number of octets in the tail of all of the addresses in the address block.

tail-length is a variable, defined to equal <tail-length> if present, or 0 otherwise.

<tail> is omitted if tail-length is equal to 0, or if the ahaszerotail bit is set ('1'), otherwise it is a field of the tail-length rightmost octets common to all the addresses in the address block. If the ahaszerotail bit is set ('1') then the tail-length rightmost octets of all the addresses in the address block are all 0.

mid-length is a variable, which MUST be non-negative, defined by:

* mid-length := address-length - head-length - tail-length

<mid> is omitted if mid-length is equal to 0, otherwise each <mid> is a field of length mid-length octets, representing the mid of the corresponding address in the address block. When not omitted, an address block contains exactly <num-addr> <mid> fields.

<prefix-length> is an 8 bit unsigned integer field containing the length, in bits, of an address prefix. If the ahassingleprelen addr-flag is set ('1') then a single <prefix-length> field is included, which contains the prefix length of all addresses in the address block. If the ahasmultiprelen addr-flag is set ('1') then <num-addr> <prefix-length> fields are included, each of which contains the prefix length of the corresponding address prefix in the address block (in the same order). Otherwise no <prefix-length> fields are present; each address object can be considered to have a prefix length equal to 8 * address-length bits. The address block is malformed if any <prefix-length> element has a value greater than 8 * address-length.

5.4. TLVs and TLV Blocks

A TLV allows association of an arbitrary attribute with a message or a packet, or with a single address or a contiguous set of addresses in an address block. The attribute (value) is made up from an integer number of consecutive octets. Different attributes have different types; attributes which are unknown when parsing can be skipped.

TLVs are grouped in TLV blocks, with all TLVs within a TLV block associating attributes with either the packet (for the TLV block in the packet header), the message (for the TLV block immediately following the message header) or to addresses in the immediately preceding address block. Individual TLVs in a TLV block immediately following an address block can associate attributes to a single address, a range of addresses or all addresses in that address block. When associating an attribute with more than one address, a TLV can include one value for all addresses, or one value per address. Detailed examples of TLVs are included in Appendix C.2.

A TLV block is defined by:

```
    <tlv-block> := <tlvs-length>
                   <tlv>*
```

where:

<tlvs-length> is a 16 bit unsigned integer field, which contains the total number of octets in all of the immediately following <tlv> elements (<tlvs-length> not included).

<tlv> is as defined in Section 5.4.1.

5.4.1. TLVs

There are three kinds of TLV, each represented by an element <tlv>:

- A packet TLV, included in the packet TLV block in a packet header.
- A message TLV, included in the message TLV block in a message, before any address blocks.
- An address block TLV, included in an address TLV block following an address block. An address block TLV applies to:
  - all address objects in the address block; OR
  - any continuous sequence of address objects in the address block; OR
  - a single address object in the address block.

<tlv> is defined by:

```
    <tlv> := <tlv-type>
             <tlv-flags>
             <tlv-type-ext>?
             (<index-start><index-stop>?)?
             (<length><value>?)?
```

where:

<tlv-type> is an 8 bit unsigned integer field, specifying the type of the TLV, specific to the TLV kind (i.e. packet, message, or address block TLV).

<tlv-flags> is an 8 bit field specifying the interpretation of the remainder of the TLV:

bit 0 (thastypeext): If cleared ('0'), then <tlv-type-ext> is not included in the <tlv>. If set ('1'), then <tlv-type-ext> is included in the <tlv>.

bit 1 (thassingleindex) and bit 2 (thasmultiindex): Are interpreted according to Table 3. A combination not shown in that table MUST NOT be used. Both of these tlv-flags MUST be cleared ('0') in packet and message TLVs.

bit 3 (thasvalue) and bit 4 (thasextlen): Are interpreted according to Table 4. A combination not shown in that table MUST NOT be used.

bit 5 (tismultivalue): This tlv-flag serves to specify how the <value> field is interpreted, as specified below. This tlv-flag MUST be cleared ('0') in packet and message TLVs, if the thasmultiindex tlv-flag is cleared ('0'), or if the thasvalue tlv-flag is cleared ('0').

bits 6-7: Are RESERVED, and SHOULD each be cleared ('0') on transmission, and SHOULD be ignored on reception.

thassingleindex	thasmultiindex	<index-start>	<index-stop>
0	0	not included	not included
1	0	included	not included
0	1	included	included

Table 3: Interpretation of the thassingleindex and thasmultiindex tlv-flags

thasvalue	thasextlen	<length>	<value>
0	0	not included	not included
1	0	8 bits	included unless <length> is zero
1	1	16 bits	included unless <length> is zero

Table 4: Interpretation of the thasvalue and thasextlen tlv-flags

<tlv-type-ext> is an 8 bit unsigned integer field, specifying an extension of the TLV type, specific to the TLV type and kind (i.e. packet, message, or address block TLV).

tlv-type-ext is a variable, defined to equal <tlv-type-ext> if present, or 0 otherwise.

tlv-fulltype is a variable, defined by:

* tlv-fulltype := 256 * <tlv-type> + tlv-type-ext

<index-start> and <index-stop> when present, in an address block TLV only, are each an 8 bit unsigned integer field.

index-start and index-stop are variables, defined according to Table 5. The variable end-index is defined as follows:

* For message and packet TLVs:

  + end-index := 0

* For address block TLVs:

  + end-index := <num-addr> - 1

An address block TLV applies to the address objects from position index-start to position index-stop (inclusive) in the address block, where the first address object has position zero.

thassingleindex	thasmultiindex	index-start :=	index-stop :=
0	0	0	end-index
1	0	<index-start>	<index-start>
0	1	<index-start>	<index-stop>

Table 5: Interpretation of the thassingleindex and thasmultiindex tlv-flags

number-values is a variable, defined by:

* number-values := index-stop - index-start + 1

<length> is omitted or is an 8 or 16 bit unsigned integer field according to Table 4. If the tismultivalue tlv-flag is set ('1') then <length> MUST be an integral multiple of number-values, and the variable single-length is defined by:

* single-length := <length> / number-values

If the tismultivalue tlv-flag is cleared ('0'), then the variable single-length is defined by:

* single-length := <length>

<value> if present (see Table 4) is a field of length <length> octets. In an address block TLV, <value> is associated with the address objects from positions index-start to index-stop, inclusive. If the tismultivalue tlv-flag is cleared ('0') then the whole of this field is associated with each of the indicated address objects. If the tismultivalue tlv-flag is set ('1') then this field is divided equally into number-values fields, each of length single-length octets, and these are associated, in order, with the indicated address objects.

5.4.2. TLV Inclusion and Constraints

A TLV associates an attribute with a packet, a message or one or more consecutive address objects in an address block. The interpretation and processing of this attribute, and the relationship (including order of processing) between different attributes associated with the same entity MUST be defined by a protocol which uses this specification.

If the appropriate flags (pnouniord for a packet TLV block, mnouniord for a message TLV block or an address block TLV block) are cleared ('0'), then the following constraints MUST be respected. These flags MUST always be cleared ('0') unless a protocol using this specification defines otherwise. Protocols SHOULD only define use of these flags when necessary, and then MUST indicate what constraints do apply if each of the indicated flags is set ('1'); each of these constraints SHOULD be retained unless otherwise necessary.

- TLVs in the same TLV block are sorted in non-descending tlv-fulltype order.

- Two or more address TLVs with the same tlv-fulltype in the same message are not associated with the same value of address object.

- TLVs with the same tlv-fulltype associated with the same address block are sorted in ascending index-start order.

In addition, packet and message TLVs MUST be defined so as to indicate whether two such TLVs with the same tlv-fulltype are, or are not, allowed in the same packet or message TLV block, respectively.

Note that the above constrains only the encoding of TLVs in a TLV block for transmission, and do specifically NOT mandate any given order of processing or interpretation by a protocol of the TLVs and the entities to which these are associated.

Respecting the constraints above allows parsing and verification of a TLV block in a single pass and allows terminating the search for a TLV with a specific type without traversing the entire TLV block.

The constraints on address block TLVs ensure that encoded information (entity-attributes) can be unambiguously decoded.

5.5. Malformed Elements

An element is malformed if it cannot be parsed according to its syntactical specification (including if there are insufficient octets available) or if a constraint (including, but not only, those specified in Section 5.4.2) specified as one which MUST be satisfied, is not. A protocol using this specification MUST specify the action, or choice of actions, to be taken when a packet containing such elements is received. Typical examples will include discarding any affected message(s), or discarding the complete packet.

6. IANA Considerations

This document introduces four namespaces that require registration: Message Types, Packet TLV Types, Message TLV Types and Address Block TLV Types. This section specifies IANA registries for these namespaces, and provides guidance to the Internet Assigned Numbers Authority regarding registrations in these namespaces.

The following terms are used with the meanings defined in [BCP26]: "Namespace", "Assigned Value", "Registration", "Unassigned", "Reserved", "Hierarchical Allocation", "Designated Expert".

The following policies are used with the meanings defined in [BCP26]: "Private Use", "Expert Review", "Standards Action".

6.1. Expert Review: Evaluation Guidelines

For registration requests where an Expert Review is required, the Designated Expert SHOULD take the following general recommendations into consideration:

- The purposes of these registries are to support Standard and Experimental MANET routing and related protocols, and extensions to the same.

- The intention is that all registrations will be accompanied by a published RFC.

- In order to allow for registration prior to the RFC being approved for publication, the Designated Expert can approve the registration once it seems clear that an RFC is expected to be published.

- The Designated Expert will post a request to the MANET WG mailing list, or to a successor hereto as designated by the Area Director, for comments and reviews. This request will include a reference to the Internet-Draft requesting the registration.

- Before a period of 30 days has passed, the Designated Expert will either approve or deny the registration request and publish a note of the decision to the MANET WG mailing list or its successor, as well as informing IANA and the IESG. A denial note MUST be justified by an explanation and, in cases where it is possible, suggestions as to how the request can be modified so as to become acceptable SHOULD be provided.

For the registry for Message Types, the following guidelines apply:

- Registration of a message type implies creation of two registries for message type specific message TLVs and message type specific address block TLVs. The document which requests the registration of the message type MUST indicate how these message type specific TLV types are to be allocated, from any options in [BCP26], and any initial allocations. The Designated Expert SHOULD take the allocation policies specified for these registries into consideration in reviewing the message type allocation request.

For the registries for Packet TLV Types, Message TLV Types and Address Block TLV Types, the following guidelines apply:

- These are Hierarchical Allocations, allocation of a type creates a registry for the extended types corresponding to that type. The document which requests the registration of the type MUST indicate how these extended types are to be allocated, from any options in [BCP26], and any initial allocations. Normally this allocation should also be Expert Review, but with the possible allocation of some type extensions as Reserved, Experimental and/or Private.

- TLV type values 0-7 MUST NOT be registered except when a documented need exists that TLVs of a given type are required to appear before all other TLVs in the TLV block as encoded for transmission. Note that the need for a TLV to be processed before other TLVs does not however automatically translate into a need for the TLV to appear before all other TLVs in the TLV block (the order in which TLVs are to be processed MUST be defined, if applicable, in the protocols using this specification).

- The request for a TLV type MUST include the specification of the permitted size, syntax of any internal structure of, and meaning of, the value field (if any) of the TLV.

For the registries for Message TLV Types and Address Block TLV Types, the following additional guidelines apply:

- TLV type values 0-127 are common for all message types. TLVs which receive registrations from the 0-127 interval SHOULD be modular in design to allow reuse among protocols.

- TLV type values 128-223 are message type specific TLV type values, relevant only in the context of the containing message type. Registration of TLV type values within the 128-223 interval require that a registry in the 128-223 interval exists for a specific message type value (see Section 6.2.1), and registrations are made in accordance with the allocation policies specified for these message type specific registries. Message type specific TLV types SHOULD be registered for TLVs which the Designated Expert deems too message type specific for registration of a 0-127 value. Multiple different TLV definitions MAY be assigned the same TLV type value within the 128-223 interval, given that they are associated with different message type specific TLV type registries. Where possible, existing global TLV definitions and modular global TLV definitions for registration in the 0-127 range SHOULD be used.

6.2. Message Types

A new registry for message types must be created, with initial assignments and allocation policies as specified in Table 6.

Type	Description	Allocation Policy
0-223	Unassigned	Expert Review
224-255	Unassigned	Experimental Use

Table 6: Message Types

6.2.1. Message Type Specific TLV Registry Creation

When a message type is registered then registries MUST be specified for both message type specific message TLVs (Table 8) and message type specific address block TLVs (Table 10). A document which creates a message type specific TLV registry MUST also specify the mechanism by which message specific TLV types are allocated, from among those in [BCP26].

6.3. Packet TLV Types

A new registry for packet TLV types must be created, with initial assignments and allocation policies as specified in Table 7.

Type	Description	Allocation Policy
0-223	Unassigned	Expert Review
224-255	Unassigned	Experimental Use

Table 7: Packet TLV Types

6.3.1. Packet TLV Type Extension Registry Creation

When a packet TLV type is registered then a new registry for type extensions of that type must be created. A document which defines a packet TLV type MUST also specify the mechanism by which its type extensions are allocated, from among those in [BCP26].

6.4. Message TLV Types

A new registry for message type independent message TLV types must be created, with initial assignments and allocation policies as

specified in Table 8.

Type	Description	Allocation Policy
0-127	Unassigned	Expert Review
128-223	Message Type Specific	Reserved, see Table 9
224-255	Unassigned	Experimental Use

Table 8: Message TLV Types

Message TLV Types 128-223 are reserved for message type specific Message TLVs, for which a new registry is created with the registration of a message type, and with initial assignments and allocation policies as specified in Table 9.

Type	Description	Allocation Policy
0-127	Common to all Message Types	Reserved
128-223	Message Type Specific	See Below
224-255	Common to all Message Types	Reserved

Table 9: Message Specific Message TLV Types

Allocation policies for message type specific message TLV types MUST be specified when creating the registry associated with the containing message type, see Section 6.2.1.

6.4.1. Message TLV Type Extension Registry Creation

If a message TLV type is registered then a new registry for type extensions of that type must be created. A document which defines a message TLV type MUST also specify the mechanism by which its type extensions are allocated, from among those in [BCP26].

6.5. Address Block TLV Types

A new registry for message type independent address block TLV types must be created, with initial assignments and allocation policies as specified in Table 10.

Type	Description	Allocation Policy
0-127	Unassigned	Expert Review
128-223	Message Type Specific	Reserved, see Table 11
224-255	Unassigned	Experimental Use

Table 10: Address Block TLV Types

Address Block TLV Types 128-223 are reserved for message type specific Address Block TLVs, for which a new registry is created with the registration of a message type, and with initial assignments and allocation policies as specified in Table 11.

Type	Description	Allocation Policy
0-127	Common to all Message Types	Reserved
128-223	Message Type Specific	See Below
224-255	Common to all Message Types	Reserved

Table 11: Message Specific Address Block TLV Types

Allocation policies for message type specific address block TLV types MUST be specified when creating the registry associated with the containing message type, see Section 6.2.1.

6.5.1. Address Block TLV Type Extension Registry Creation

When an address block TLV type is registered then a new registry for type extensions of that type must be created. A document which defines a message TLV type MUST also specify the mechanism by which its type extensions are allocated, from among those in [BCP26].

7. Security Considerations

This specification does not describe a protocol; it describes a packet format. As such it does not specify any security considerations, these are matters for a protocol using this specification. However some security mechanisms are enabled by this specification, and may form part of a protocol using this specification. Mechanisms which may form part of an authentication and integrity approach in a protocol using this specification, are described in Section 7.1. Mechanisms which may form part of a confidentiality approach in a protocol using this specification, are described in Section 7.2. There is however no requirement that a

protocol using this specification should use either.

7.1. Authentication and Integrity Suggestions

The authentication and integrity suggestions made here, are based on the intended usage in Appendix A, specifically that:

- Messages are designed to be carriers of protocol information and MAY, at each hop, be forwarded and/or processed by the protocol using this specification.

- Packets are designed to carry a number of messages between neighboring nodes in a single transmission and over a single logical hop.

Consequently:

- For forwarded messages where the message is unchanged by forwarding nodes, then end-to-end authentication and integrity MAY be implemented, between nodes with an existing security association, by including a suitable message TLV containing a cryptographic signature in the message. Since <hop-count> and <hop-limit> are the only fields that should be modified when such a message is forwarded in this manner, this signature can be calculated based on the entire message, including the message header, with the <hop-count> and <hop-limit> fields set to 0 if present.

- Hop-by-hop packet level authentication and integrity MAY be implemented, between nodes with an existing security association, by including a suitable packet TLV containing a cryptographic signature to the packet. Since packets are received as transmitted, this signature can be calculated based on the entire packet, or on parts thereof as appropriate.

7.2. Confidentiality Suggestions

This specification does not explicitly enable protecting packet/message confidentiality. Such confidentiality would normally, when required, be provided hop-by-hop either by link-layer mechanisms, or at the IP layer using [RFC4301], and would apply to a packet only.

It is possible, however, for a protocol using this specification to protect the confidentiality of information included in a packet, message or address block TLV by specifying that the value field of that TLV type be encrypted, as well as specifying the encryption mechanism.

In an extreme case, all information can be encrypted by defining either:

- A packet, consisting of only a packet header (with no messages), and containing a packet TLV, where the packet TLV type indicates that its value field contains one or more encrypted messages. Upon receipt, and once this packet TLV is successfully decrypted, these messages may then be parsed according to this specification and processed according to the protocol using this specification.

- A message, consisting of only a message header and a single message TLV, where the message TLV type indicates that its value field contains an encrypted version of the message's remaining message TLVs, address blocks and address block TLVs. Upon receipt, and once this message TLV is successfully decrypted, the complete message may then be parsed according to this specification and processed according to the protocol using this specification.

In either case, the protocol MUST define the encrypted TLV type, as well as the format of the encrypted data block contained in the value field of the TLV.

8. References

8.1. Normative References

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

[BCP26] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, BCP 26, May 2008.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, February 2006.

[SingleUNIX] IEEE Std 1003.1, The Open Group, and ISO/IEC JTC 1/SC22/WG15, "Single UNIX Specification, Version 3, 2004 Edition", April 2004.

[POSIX] ISO/IEC Standard 9945-1:2003, "Portable Operating System Interface (POSIX) - Part 1: Base Definitions", January 2003.

[ECMAScript] ISO/IEC Standard 9845-1:2003, "Ecma-262, ECMAScript Language Specification", January 2003.

8.2. Informative References

[RFC3626] Clausen, T. and P. Jacquet, "The Optimized Link State Routing Protocol", RFC 3626, October 2003.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[Stevens] Stevens, W., "TCP/IP Illustrated Volume 1 - The Protocols.", 1994.

Appendix A. Intended Usage

The packet and message format specified in this document is designed to carry MANET routing protocol signaling between MANET routers, and to support scope limited diffusion (flooding) as well as point-to-point delivery.

Specifically:

- Packets are transmitted over a single logical hop and are not forwarded. Messages are designed to be able to be forwarded over one or more logical hops, in a new packet for each logical hop. Each logical hop may consist of one or more IP hops. Packets may contain a different combination of messages for each logical hop. Packets may be formed from messages from more than one MANET routing protocol; the use of a single message type space for all such protocols allows these messages to be separated by protocol.

- Scope limited flooding is supported for messages thus:

  * If the mhasorig and mhasseqnum msg-flags are both set ('1') then the message header provides for duplicate suppression, using the identifier consisting of the message's <msg-orig-addr>, <msg-seq-num>, and, if the mistypedep flag in the <msg-flags> field is set ('1'), <msg-type>. These serve to uniquely identify the message in the network within the time period until <msg-seq-num> is repeated, i.e. wraps around to a matching value if used sequentially.

  * If the mhashoplimit msg-flag is set ('1'), then the message header provides the information to make forwarding decisions for scope limited diffusion (flooding). This may be by any appropriate flooding mechanism specified by a protocol using this specification.

  * <msg-hop-limit> SHOULD, if present, be decremented, usually by one, if the message is forwarded.

* <msg-hop-count> SHOULD, if present, be incremented, usually by one, if the message is forwarded.

Appendix B. Multiplexing and Demultiplexing

The packet and message format specified in this document is designed to allow zero or more messages to be contained within a single packet (e.g. a single IP datagram or UDP datagram). Such messages may be from the same or may be from different protocols. In the latter case, multiplexing of outgoing messages from different protocols and demultiplexing of incoming messages contained within a single packet, must be addressed.

Multiplexing messages of different message types from different protocols running on a given node into a single packet, rather than to have each protocol generate its own packet, is an optimization. Doing so may reduce the total number of octets as transmitted by that node, while not doing so should not have any adverse effects on protocol functioning.

Having two different protocols (or, two different instances of the same protocol) generate messages of the same message type on the same interface may, however, cause inconsistencies, unless this message generation is coordinated.

If a packet contains messages destined for multiple protocols, correct functioning of each of these protocols depends on the appropriate messages being delivered to each protocol.

If multiple protocols which are using the packet and message format specified in this document are running over the same UDP port or IP protocol number, multiplexing and demultiplexing of messages SHOULD be accomplished based on message types, ensuring a correspondence between message types and protocol instances thus:

- For each message type, a protocol - unless specified otherwise, the one making the IANA reservation for that message type - SHOULD be designated as the "owner" of that message type.

- On a given interface, and for a given UDP port or IP protocol number, only one protocol instance of a protocol which is the designated "owner" of a message type SHOULD be running,

- Any incoming messages of a given message type SHOULD be delivered to the protocol instance on the receiving interface which "owns" messages of that type. Specifically, any incoming message SHOULD NOT be delivered to any other protocol instances and SHOULD NOT be delivered to more than one protocol instance.

- Any outgoing messages of a given type SHOULD be generated and transmitted only by the protocol instance which "owns" that message type.

- If two protocols both wish to use the same message type (e.g. through inspecting information carried in messages of that type, or through inserting information into generated messages of that type) then this interaction SHOULD be specified by the protocol which is the designated "owner" of that message type.

Appendix C. Examples

This appendix contains some examples of parts of this specification.

C.1. Address Block Examples

The following examples illustrate how some combinations of addresses may be efficiently included in address blocks. These examples are for IPv4, with address-length equal to 4. a, b, c etc. represent distinct, non-zero, octet values.

Note that it is permissible to use a less efficient representation, in particular one in which the ahashead and ahasfulltail flags are cleared ('0'), and hence head-length = 0, tail-length = 0, mid-length = address-length, and (with no address prefixes) the address block consists of the number of addresses, <addr-flags> with value 0, and a list of the unaggregated addresses. This is the most efficient way to represent a single address, and the only way to represent, for example, a.b.c.d and e.f.g.h in one address block.

Examples:

- To include a.b.c.d, a.b.e.f and a.b.g.h:

  - head-length = 2;

  - tail-length = 0;

  - mid-length = 2;

  - <addr-flags> has ahashead set (value 128);

  - <tail-length> and <tail> are omitted.

  The address block is then 3 128 2 a b c d e f g h (11 octets).

- To include a.b.c.g and d.e.f.g:

* head-length = 0;

* tail-length = 1;

* mid-length = 3;

* <addr-flags> has ahasfulltail set (value 64);

* <head-length> and <head> are omitted.

The address block is then 2 64 1 g a b c d e f (10 octets).

- To include a.b.d.e and a.c.d.e:

  * head-length = 1;

  * tail-length = 2;

  * mid-length = 1;

  * <addr-flags> has ahashead and ahasfulltail set (value 192).

  The address block is then 2 192 1 a 2 d e b c (9 octets).

- To include a.b.0.0, a.c.0.0, and a.d.0.0:

  * head-length = 1;

  * tail-length = 2;

  * mid-length = 1;

  * <addr-flags> has ahashead and ahaszerotail set (value 160);

  * <tail> is omitted.

  The address block is then 3 160 1 a 2 b c d (8 octets).

- To include a.b.0.0 and c.d.0.0:

  * head-length = 0;

  * tail-length = 2;

  * mid-length = 2;

  * <addr-flags> has ahaszerotail set (value 32);

* <head> and <tail> are omitted.

The address block is then 2 32 2 a b c d (7 octets).

o To include a.b.0.0/n and c.d.0.0/n:

* head-length = 0;

* tail-length = 2;

* mid-length = 2;

* <addr-flags> has ahaszerotail and ahassingleprelen set (value 48);

* <head> and <tail> are omitted.

The address block is then 2 48 2 a b c d n (8 octets).

o To include a.b.0.0/n and c.d.0.0/m:

* head-length = 0;

* tail-length = 2;

* mid-length = 2;

* <addr-flags> has ahaszerotail and ahasmultiprelen set (value 40);

* <head> and <tail> are omitted.

The address block is then 2 40 2 a b c d n m (9 octets).

C.2. TLV Examples

Assuming the definition of an address block TLV with type EXAMPLE1 (and no type extension) which has single octet values per address, then if values a, a, b and c are to be associated with the four addresses in the preceding address block, where c is a default value that can be omitted, then this can be done in a number of ways.

Examples:

o Using one multivalue TLV covering all of the addresses:

* <tlv-flags> has thasvalue and tismultivalue set (value 20);

* <index-start> and <index-stop> are omitted;

* <length> = 4 (single-length = 1).

* The TLV is then EXAMPLE1 20 4 a a b c (7 octets).

o Using one multivalue TLV omitting the last address:

* <tlv-flags> has thasmultiindex, thasvalue and tismultivalue set (value 52);

* <index-start> = 0;

* <index-stop> = 2

* <length> = 3 (single-length = 1).

* The TLV is then EXAMPLE1 52 0 2 3 a a b (8 octets).

o Using two single value TLVs, omitting the last address. First:

* <tlv-flags> has thasmultiindex and thasvalue set (value 48);

* <index-start> = 0;

* <index-stop> = 1;

* <length> = 1;

* <value> = a.

* The TLV is then EXAMPLE1 48 0 1 1 a (6 octets).

Second:

* <tlv-flags> has thassingleindex and thasvalue set (value 80);

* <index-start> = 2;

* <index-stop> is omitted;

* <length> = 1;

* <value> = b.

* The TLV is then EXAMPLE1 80 2 1 b (5 octets).

Total length of TLVs is 11 octets.

In this case the first of these is the most efficient. In other cases patterns such as the others may be preferred. Regardless of efficiency, any of these may be used.

Assuming the definition of an address block TLV with type EXAMPLE2 (and no type extension) which has no value, which is to be associated with the second and third addresses in an address block, then this can be indicated with a single TLV:

- <tlv-flags> has thasmultiindex set (value 32);
- <index-start> = 1;
- <index-stop> = 2;
- <length> and <value> are omitted.
- The TLV is then EXAMPLE2 32 1 2 (4 octets).

Assuming the definition of a message TLV with type EXAMPLE3 (and no type extension) which can take a value field of any length, for such a TLV with 8 octets of data (a to h):

- <tlv-flags> has thasvalue set (value 16);
- <index-start> and <index-stop> are omitted;
- <length> = 8.
- The TLV is then EXAMPLE3 16 8 a b c d e f g h (11 octets).

If, in this example, the number of data octets were 256 or greater then <tlv-flags> would also have thasextlen set and have value 24. The length would require two octets (most significant first). The TLV length would be 4 + N octets, where N is the number of data octets (it can be 3 + N octets if N is 255 or less).

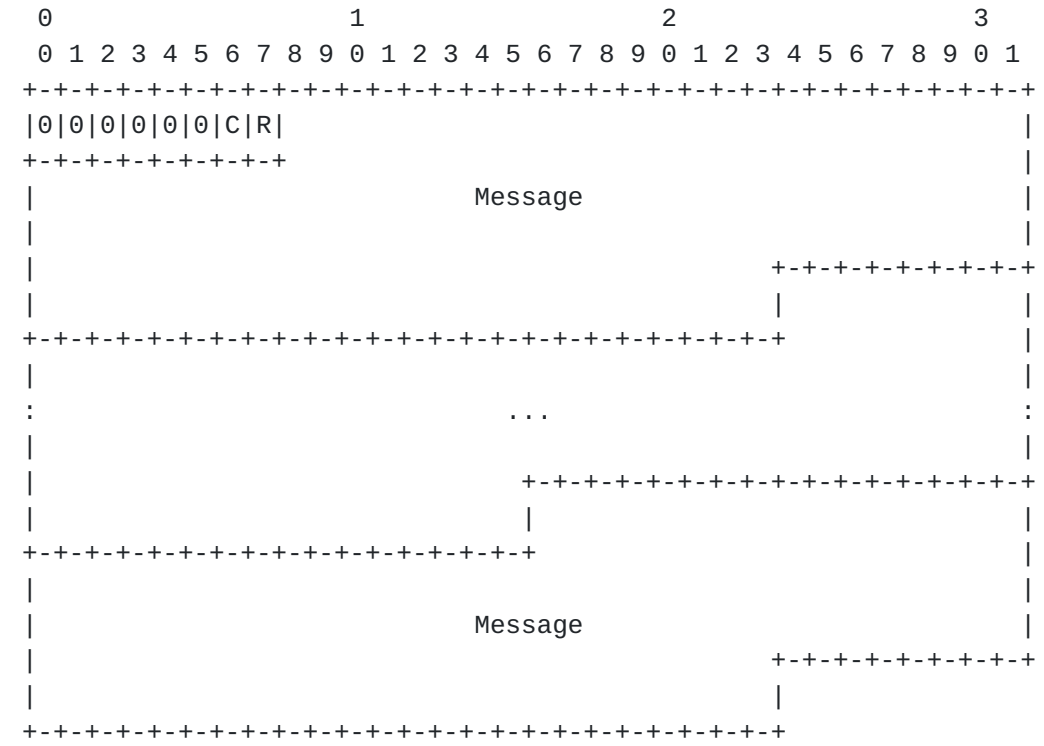
Appendix D. Illustrations

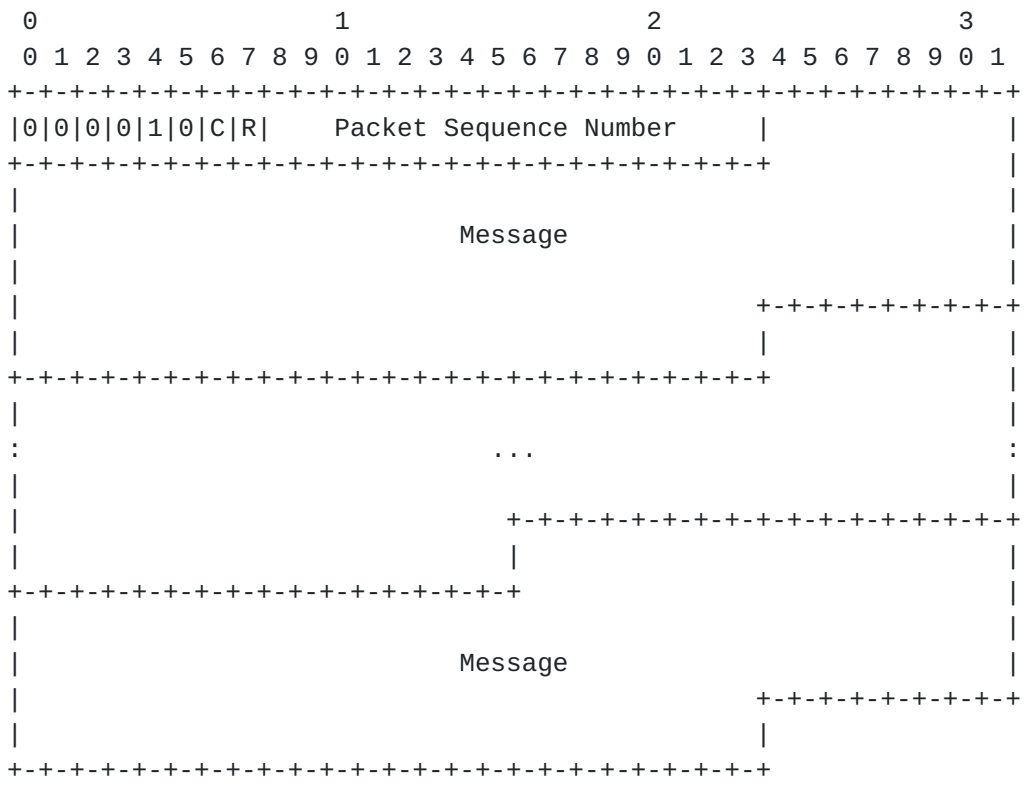
This informative appendix illustrates the elements which are normatively specified in Section 5.

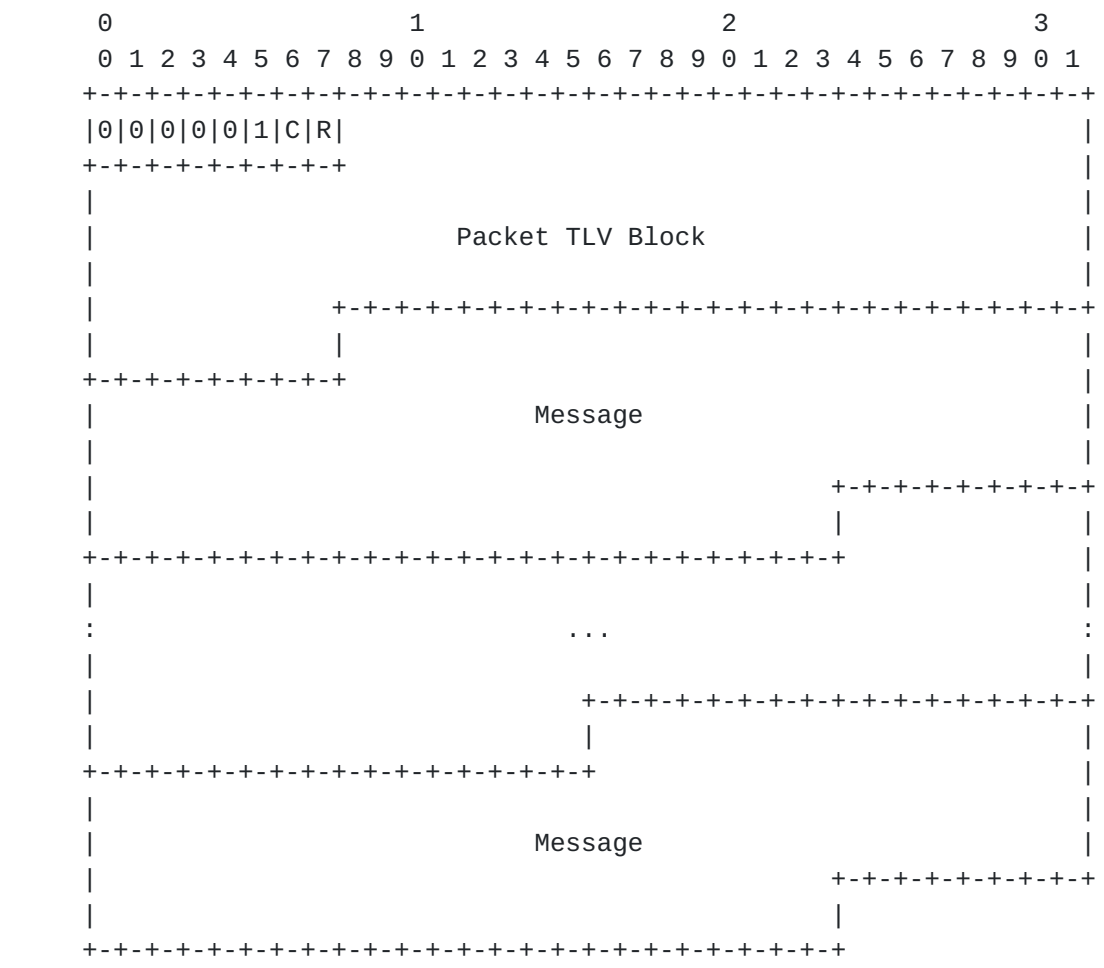
Bits labeled Rsv or R should be cleared ('0'). Bits labeled C, N, or M may be cleared ('0') or set ('1').

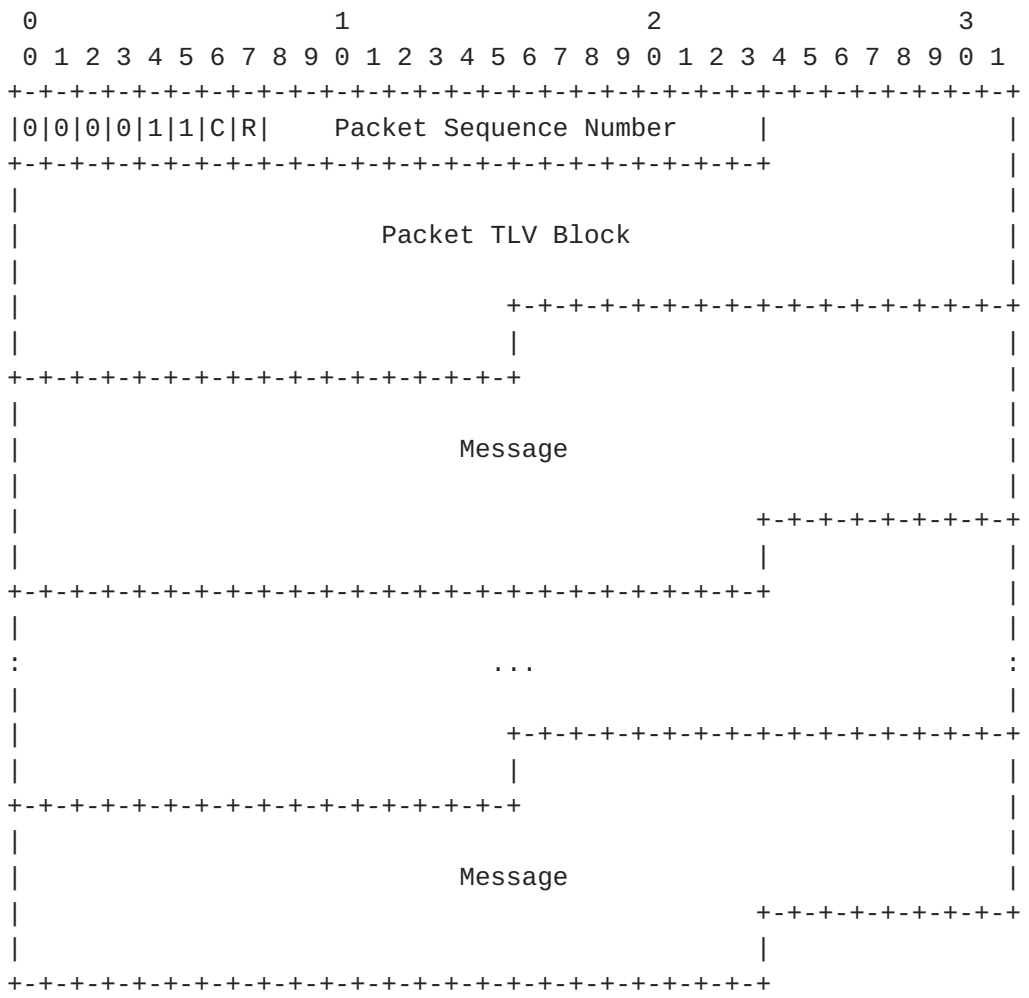
D.1. Packet

Possible options for the <packet> element. These are differentiated by the flags field in the first octet. The packet may include any number (zero or more) of Messages. The number of Messages is determined from when the packet is exhausted, given the packet length from the network layer.



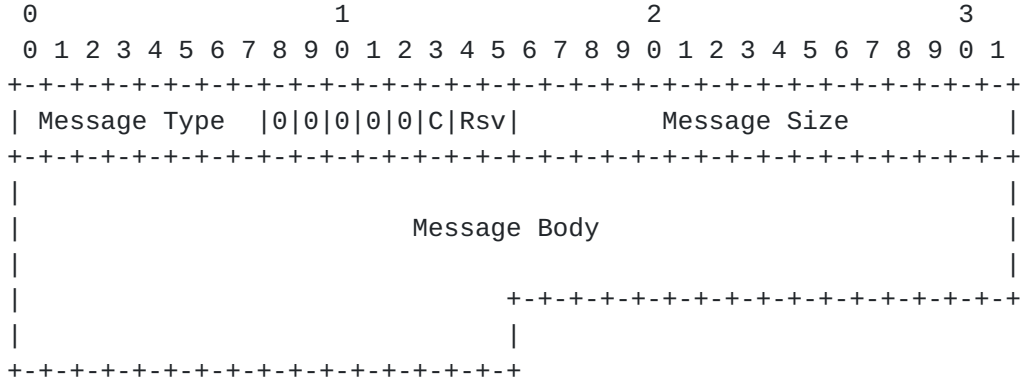

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
|0|0|0|0|1|0|C|R|
Packet Sequence Number
Message
...
Message


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
|0|0|0|0|0|1|C|R|
Packet TLV Block
Message
...
Message

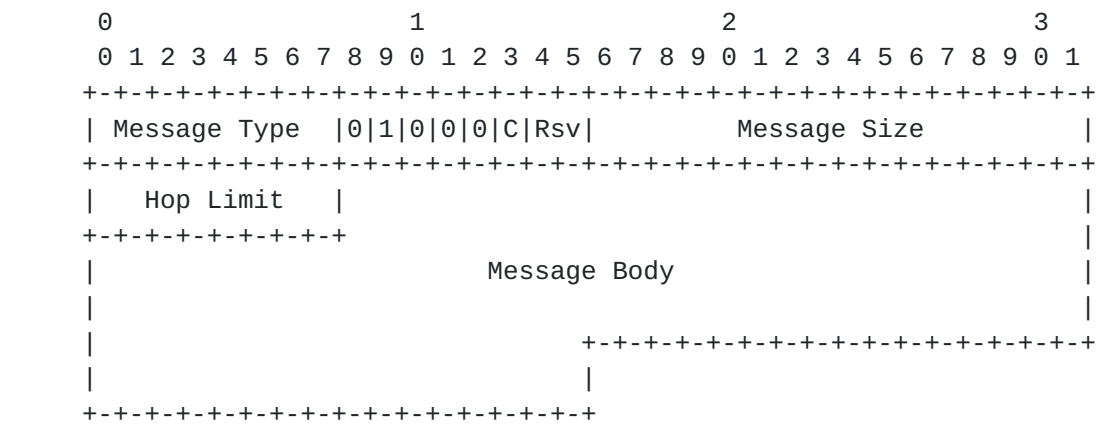


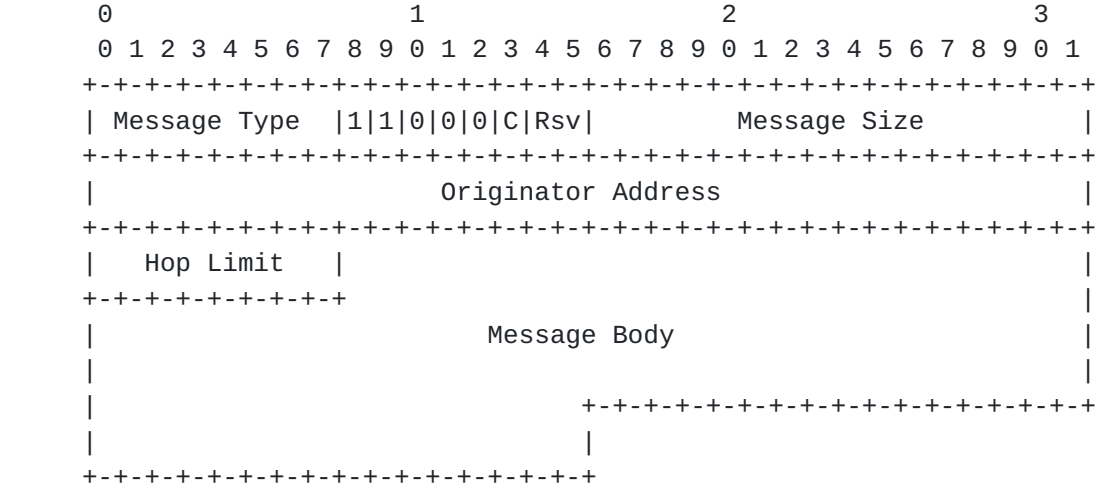
D.2. Message

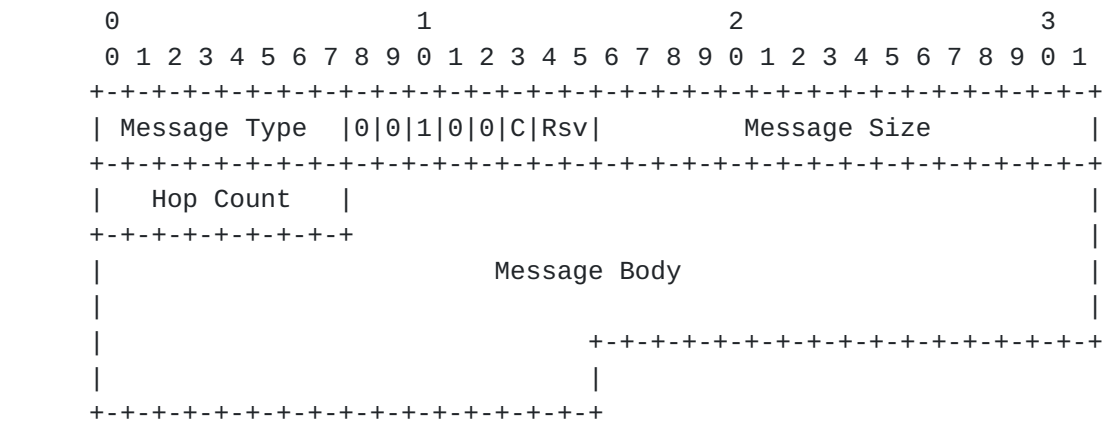
Possible options for the <message> element. These are differentiated by their second (flags) octet. The length of the Message Body is determined using the Message Size field, which is the combined length of all the fields shown.

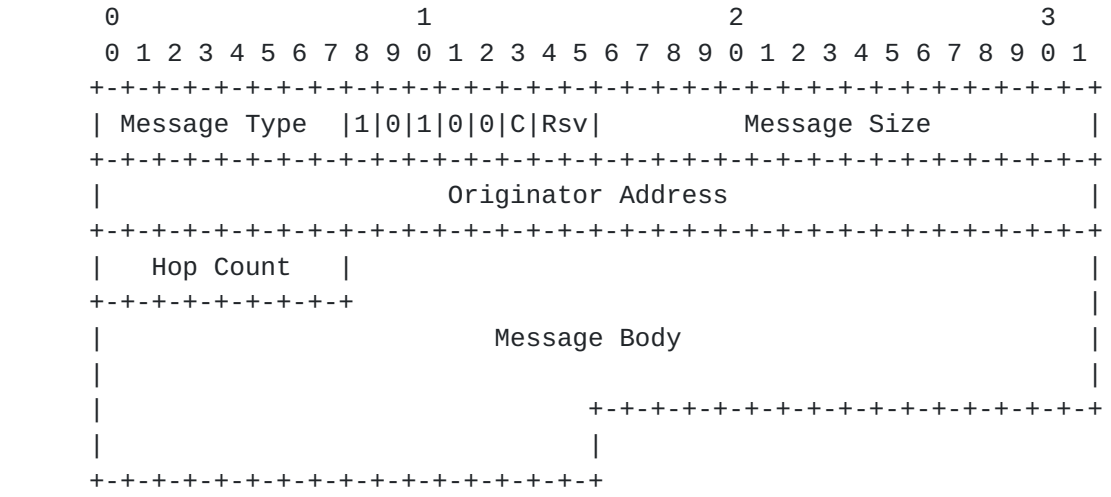


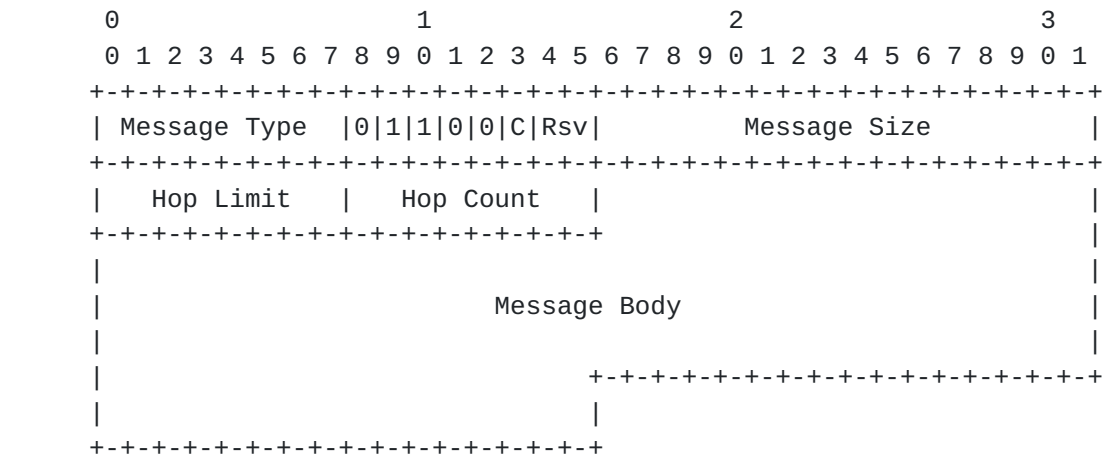

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type
1 0 0 0 0 C Rsv
Message Size
Originator Address
Message Body

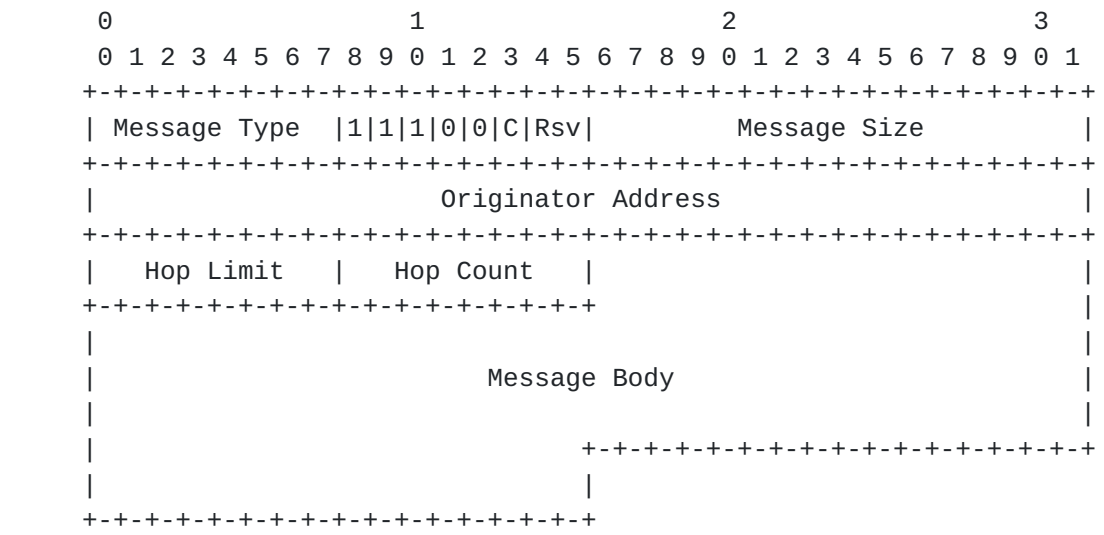

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type
0 1 0 0 0 C Rsv
Message Size
Hop Limit
Message Body

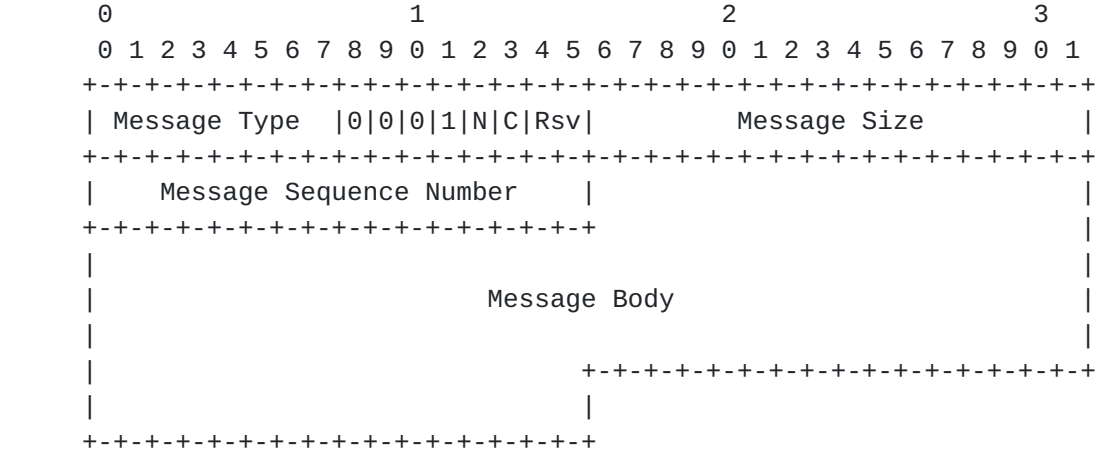

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type
1 1 0 0 0 C Rsv
Message Size
Originator Address
Hop Limit
Message Body

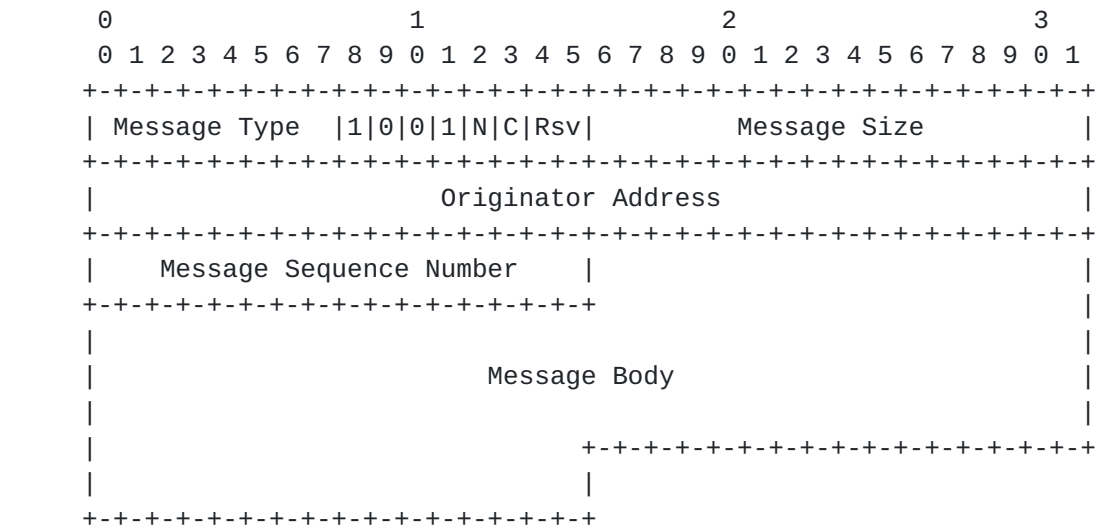

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type |0|0|1|0|0|C|Rsv| Message Size
Hop Count
Message Body

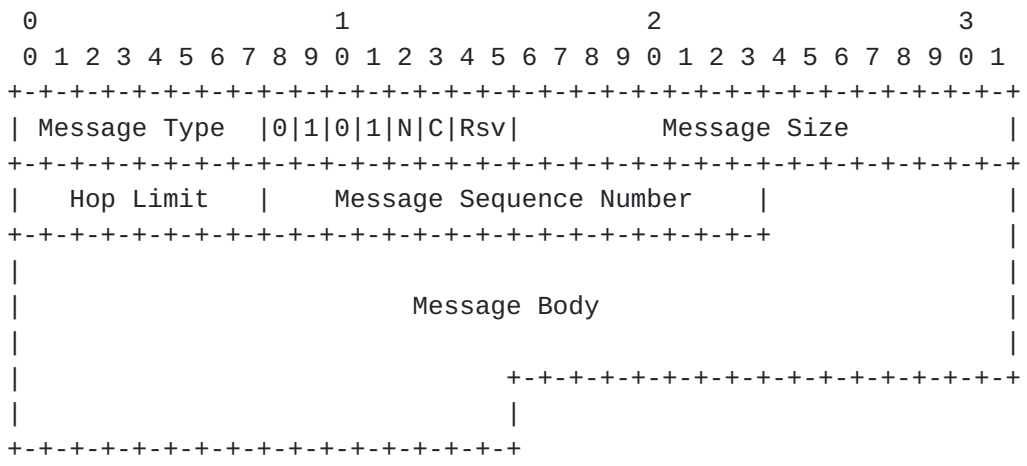

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type |1|0|1|0|0|C|Rsv| Message Size
Originator Address
Hop Count
Message Body

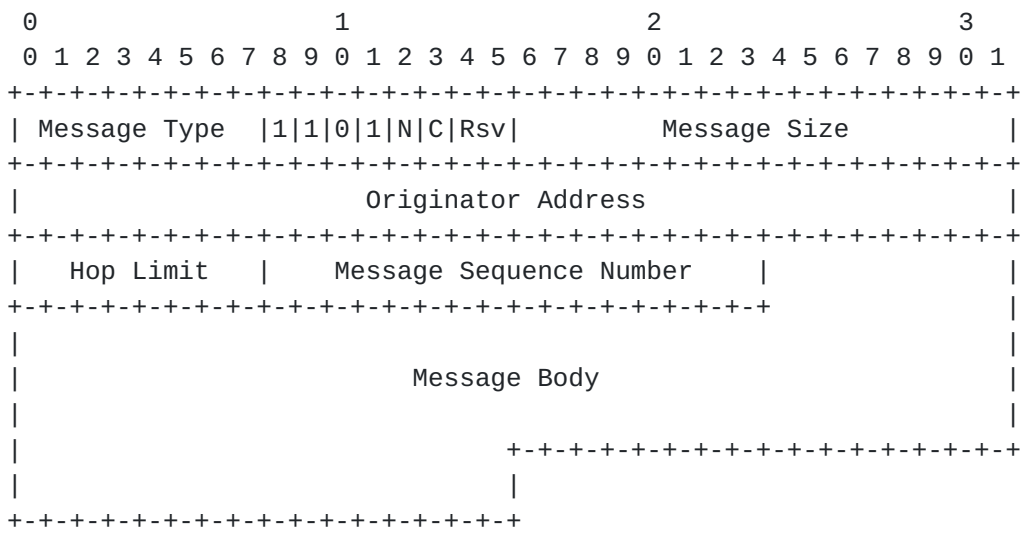

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type |0|1|1|0|0|C|Rsv| Message Size
Hop Limit
Hop Count
Message Body

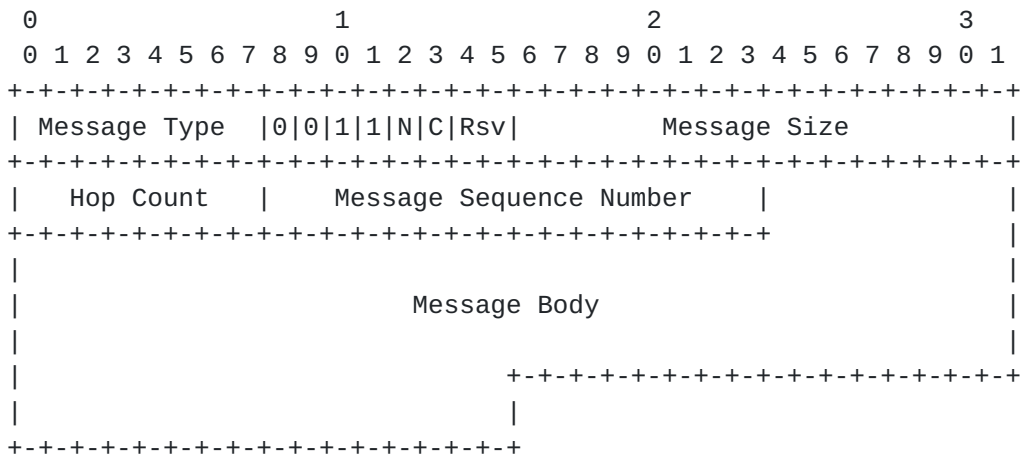

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type |1|1|1|0|0|C|Rsv| Message Size
Originator Address
Hop Limit
Hop Count
Message Body

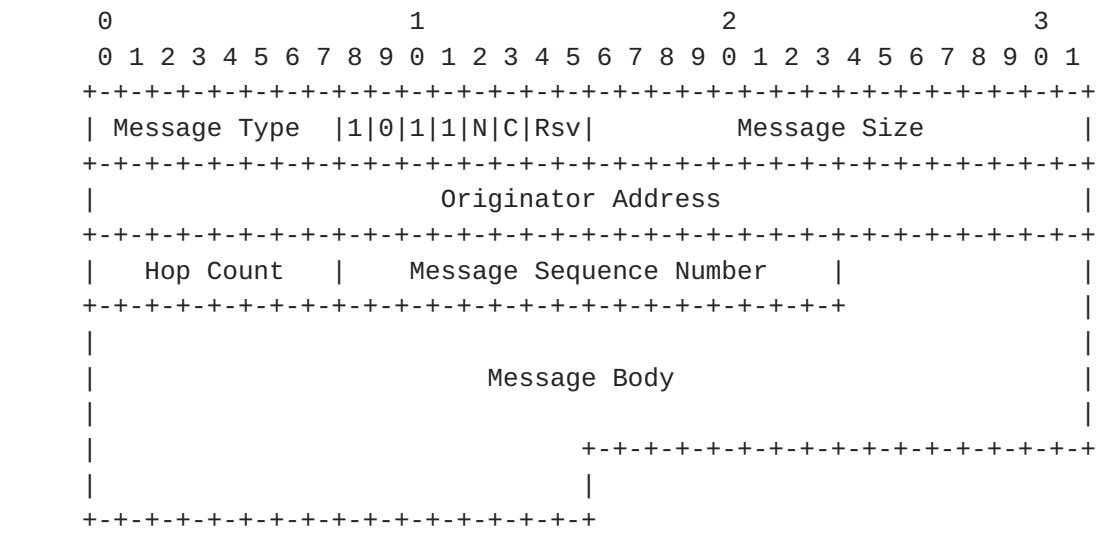

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type |0|0|0|1|N|C|Rsv| Message Size
Message Sequence Number
Message Body

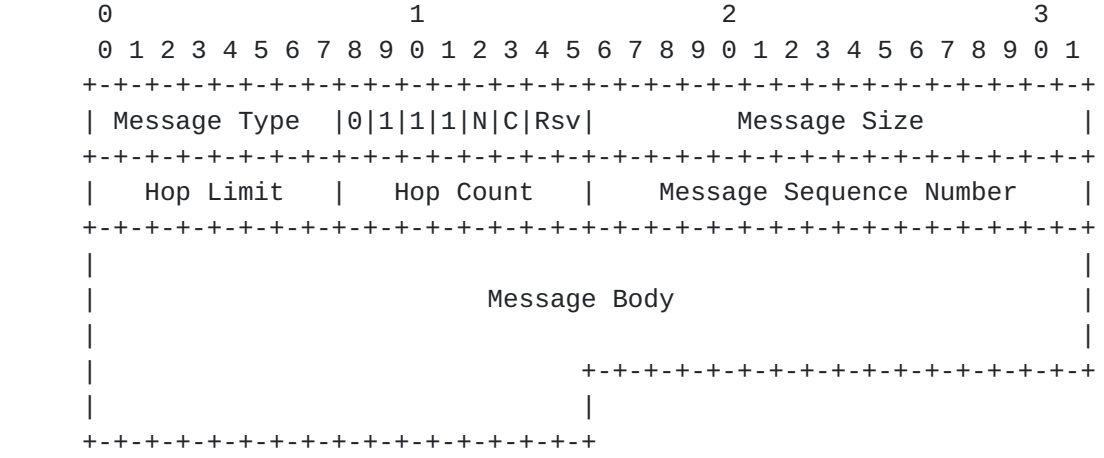

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type |1|0|0|1|N|C|Rsv| Message Size
Originator Address
Message Sequence Number
Message Body

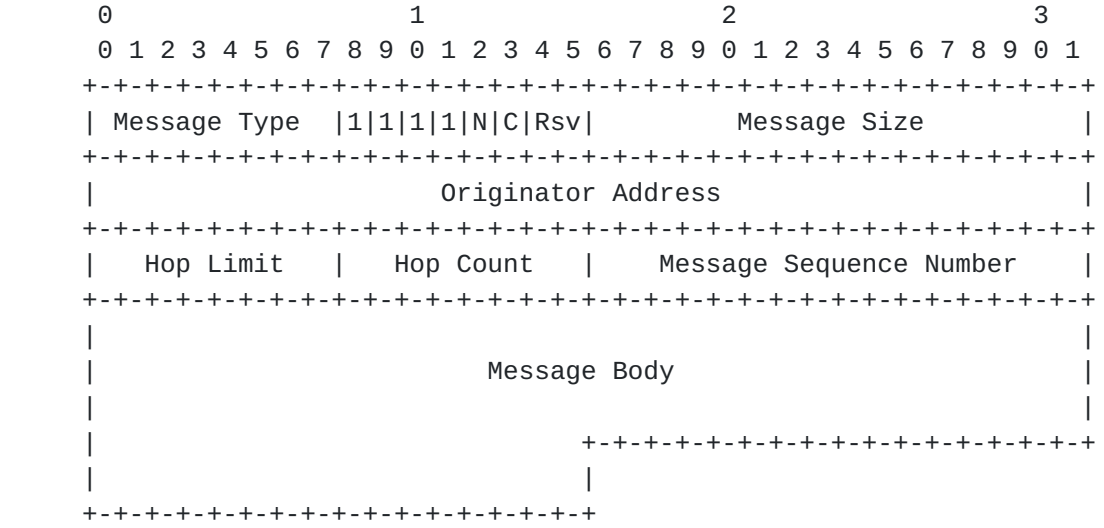

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type |0|1|0|1|N|C|Rsv| Message Size
Hop Limit | Message Sequence Number
Message Body


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type |1|1|0|1|N|C|Rsv| Message Size
Originator Address
Hop Limit | Message Sequence Number
Message Body


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type |0|0|1|1|N|C|Rsv| Message Size
Hop Count | Message Sequence Number
Message Body

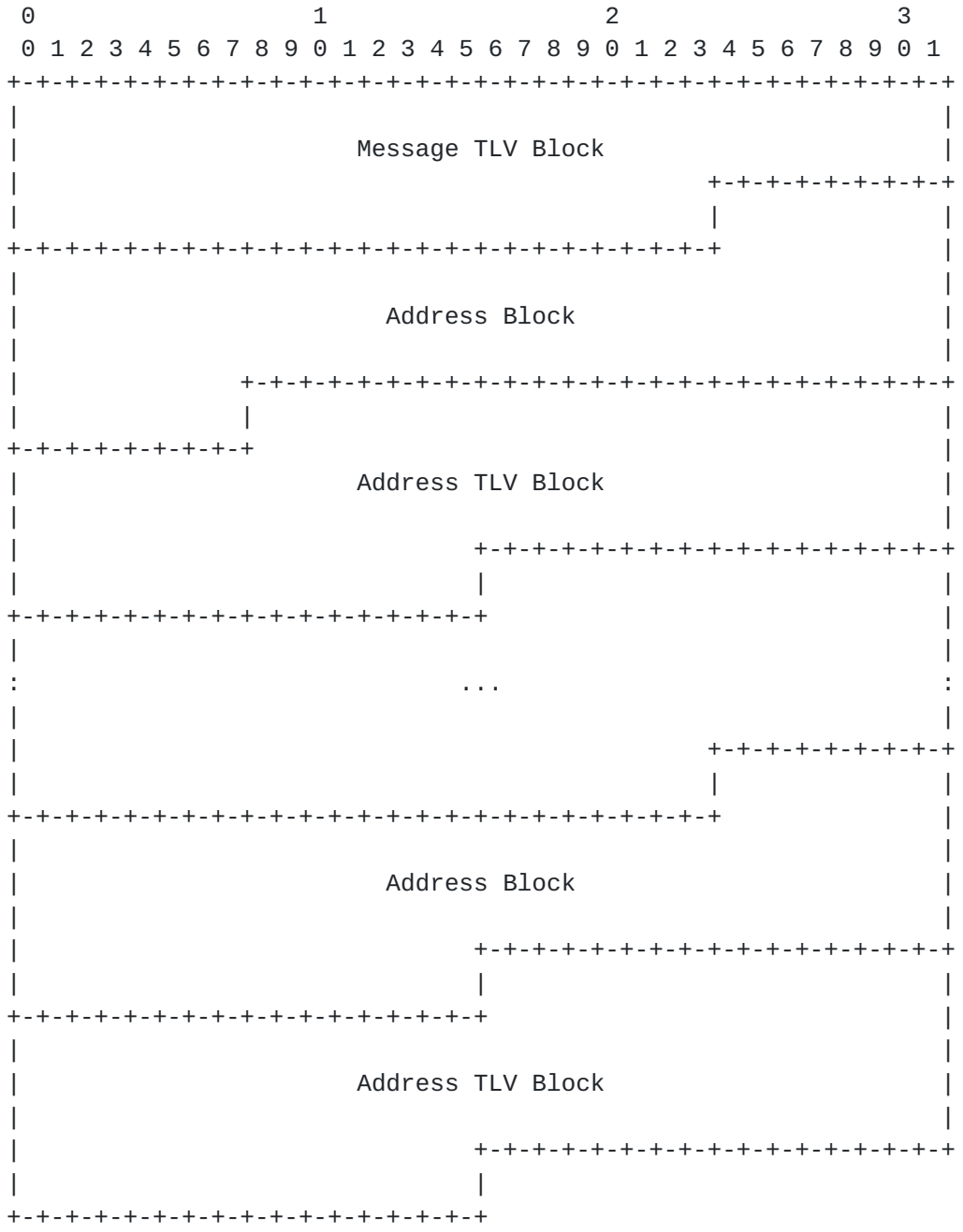

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type |1|0|1|1|N|C|Rsv| Message Size
Originator Address
Hop Count
Message Sequence Number
Message Body


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type |0|1|1|1|N|C|Rsv| Message Size
Hop Limit
Hop Count
Message Sequence Number
Message Body


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type |1|1|1|1|N|C|Rsv| Message Size
Originator Address
Hop Limit
Hop Count
Message Sequence Number
Message Body

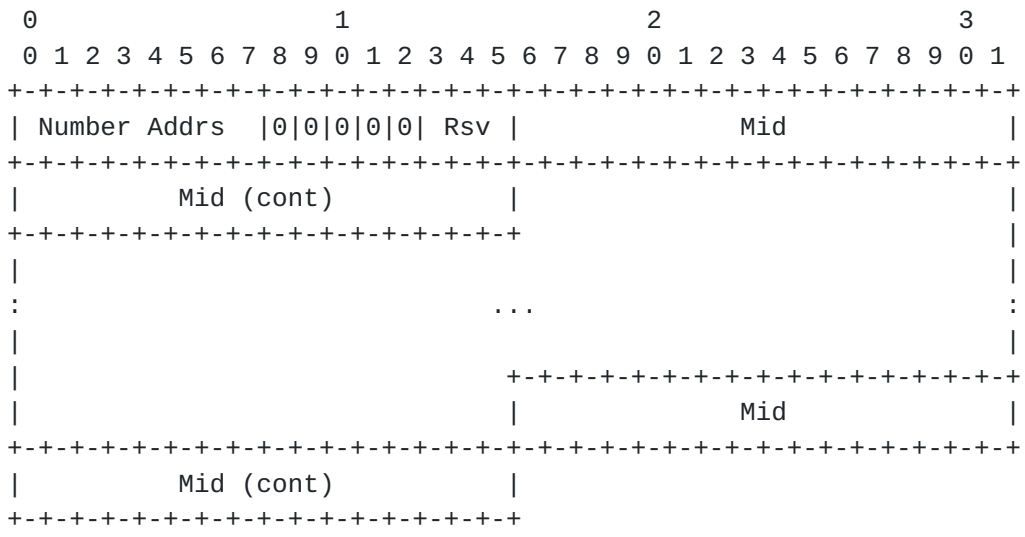
D.3. Message Body

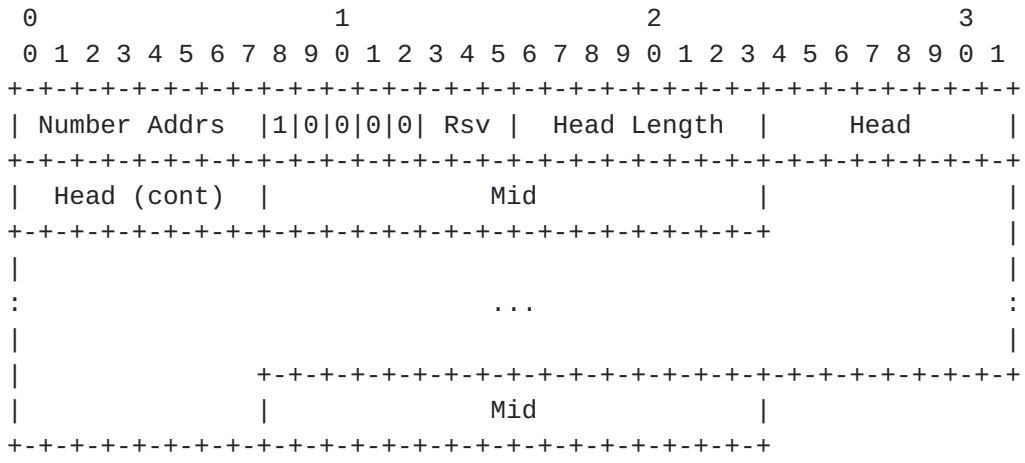
Format of a message body (the <message> element excluding its initial <msg-header> element). The message body includes one Message TLV Block (containing zero or more TLVs) and may include any number (zero or more) of Address Block and Address TLV Block pairs.

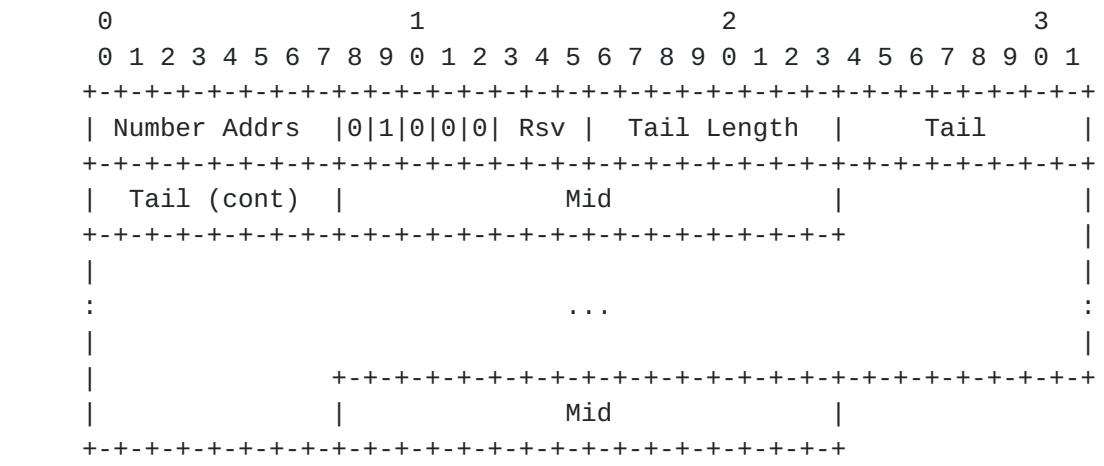


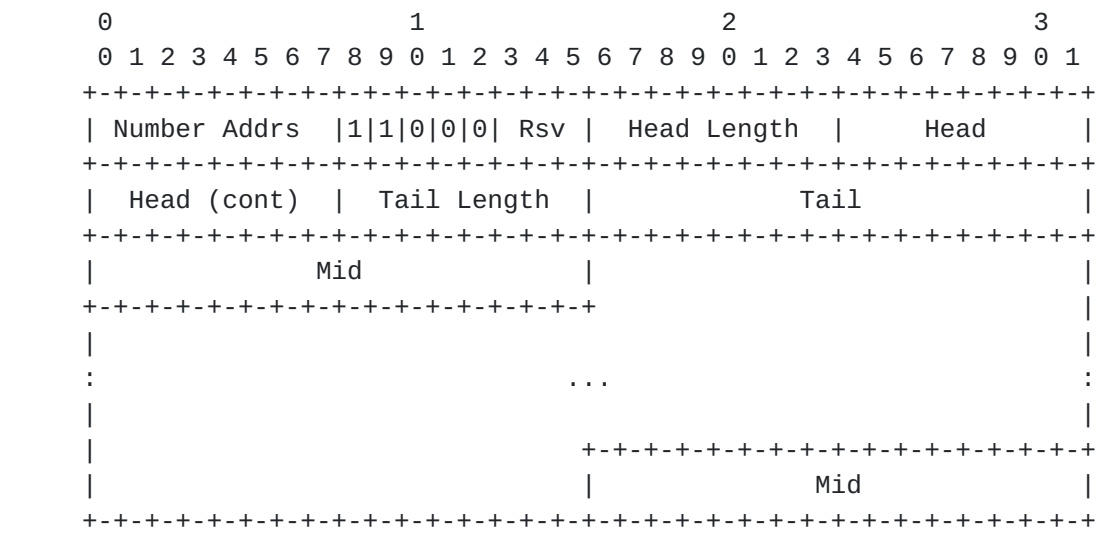
D.4. Address Block

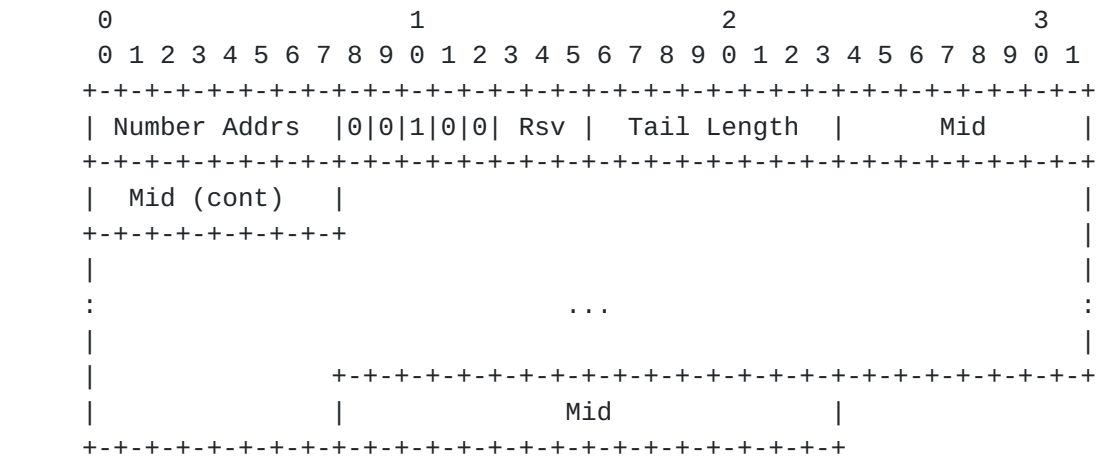
Possible options for the <addr-block> element. These are differentiated by their second (flags) octet. The number of Mid elements is equal to the number of addresses (except when mid-length is zero, when there are no Mid elements). Where a variable number of Prefix Length fields is shown, their number is equal to the number of addresses.

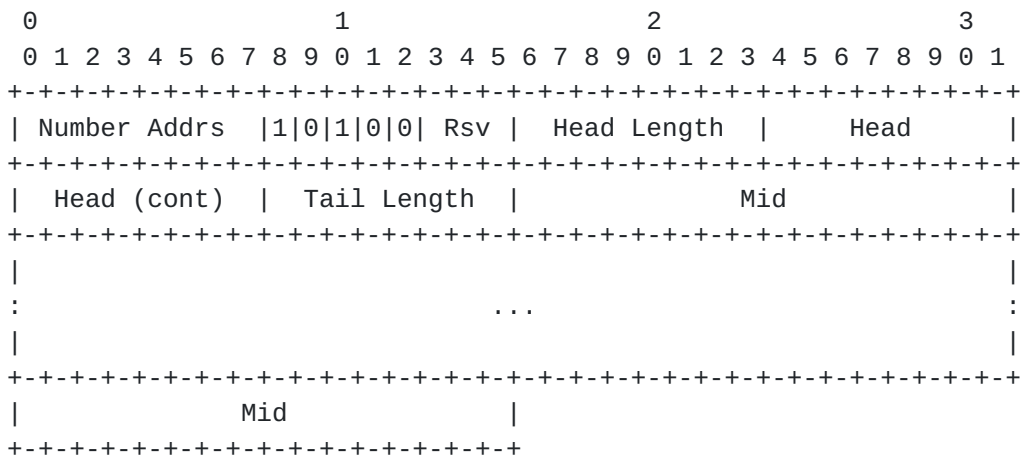


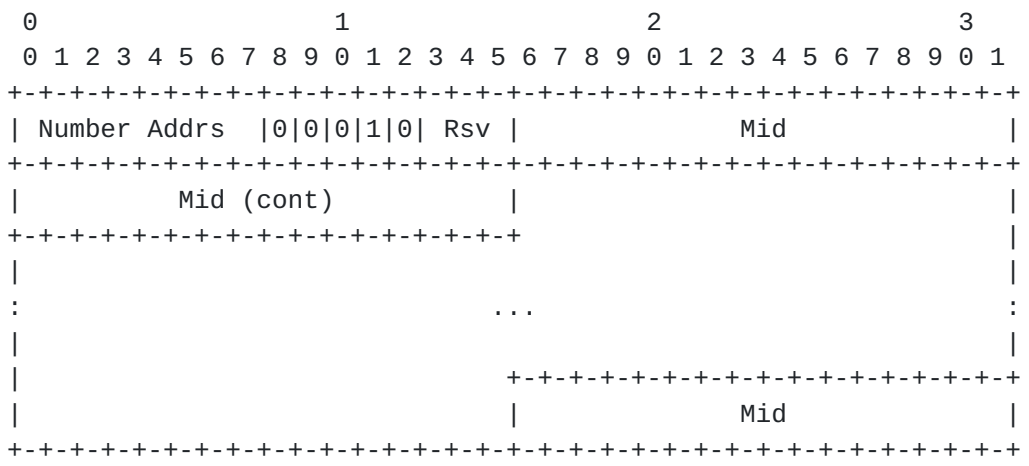


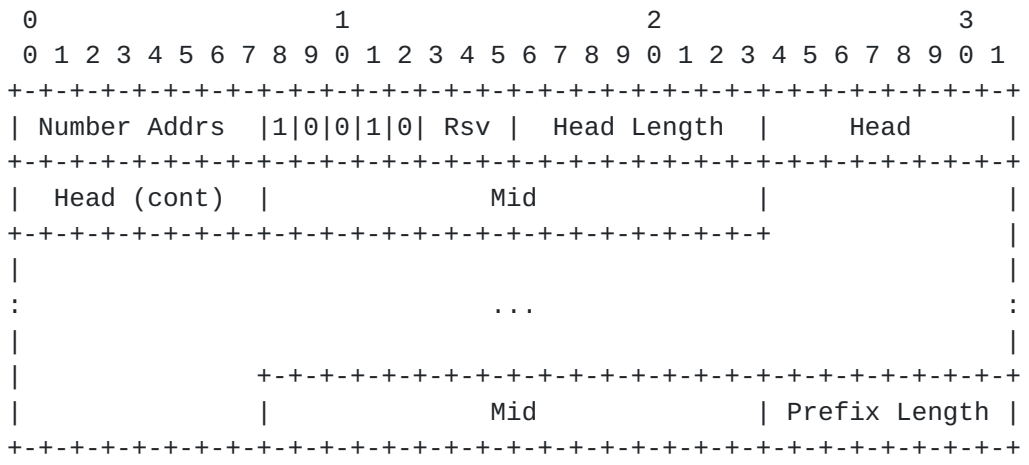

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
0 1 0 0 0
Rsv
Tail Length
Tail
Tail (cont)
Mid
...
Mid

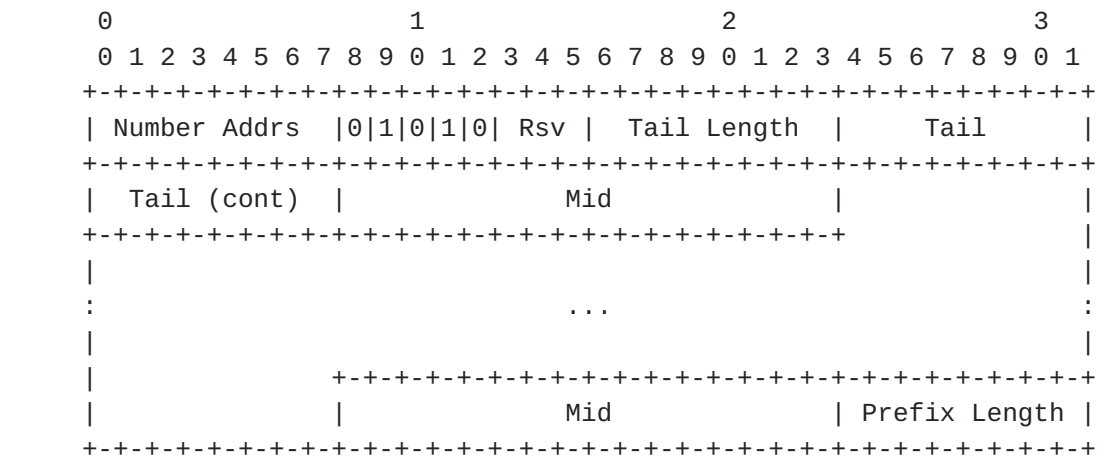

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
1 1 0 0 0
Rsv
Head Length
Head
Head (cont)
Tail Length
Tail
Mid
...
Mid

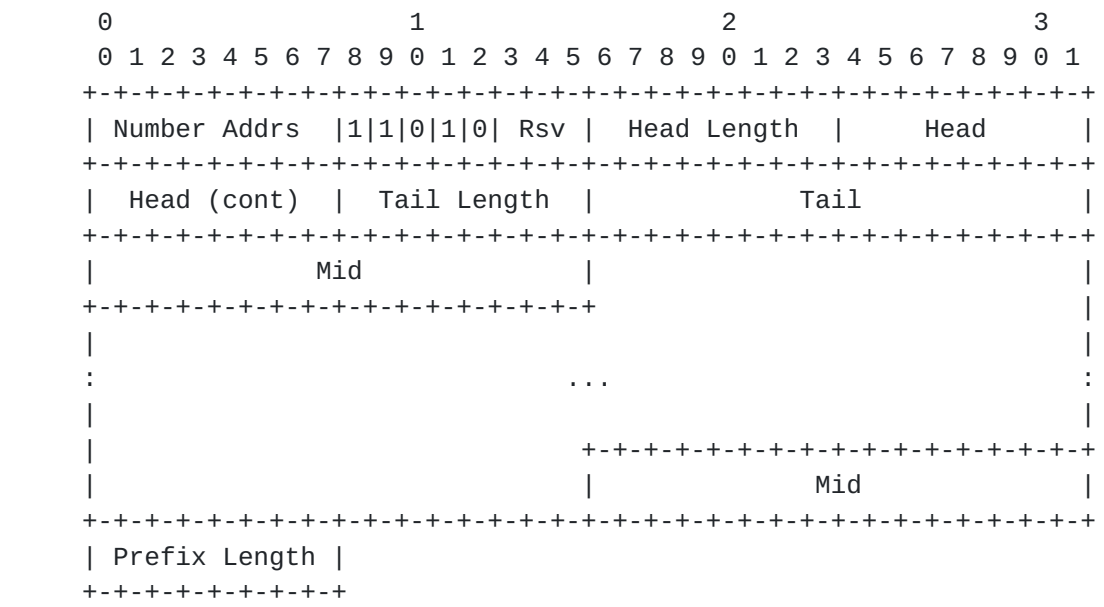

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
0 0 1 0 0
Rsv
Tail Length
Mid
Mid (cont)
...
Mid

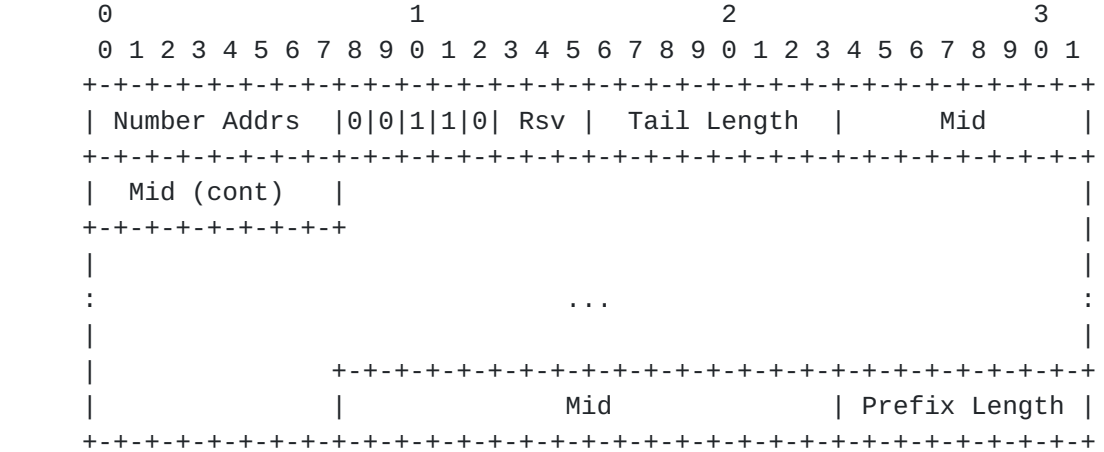

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
1 0 1 0 0
Rsv
Head Length
Head
Head (cont)
Tail Length
Mid
...
Mid

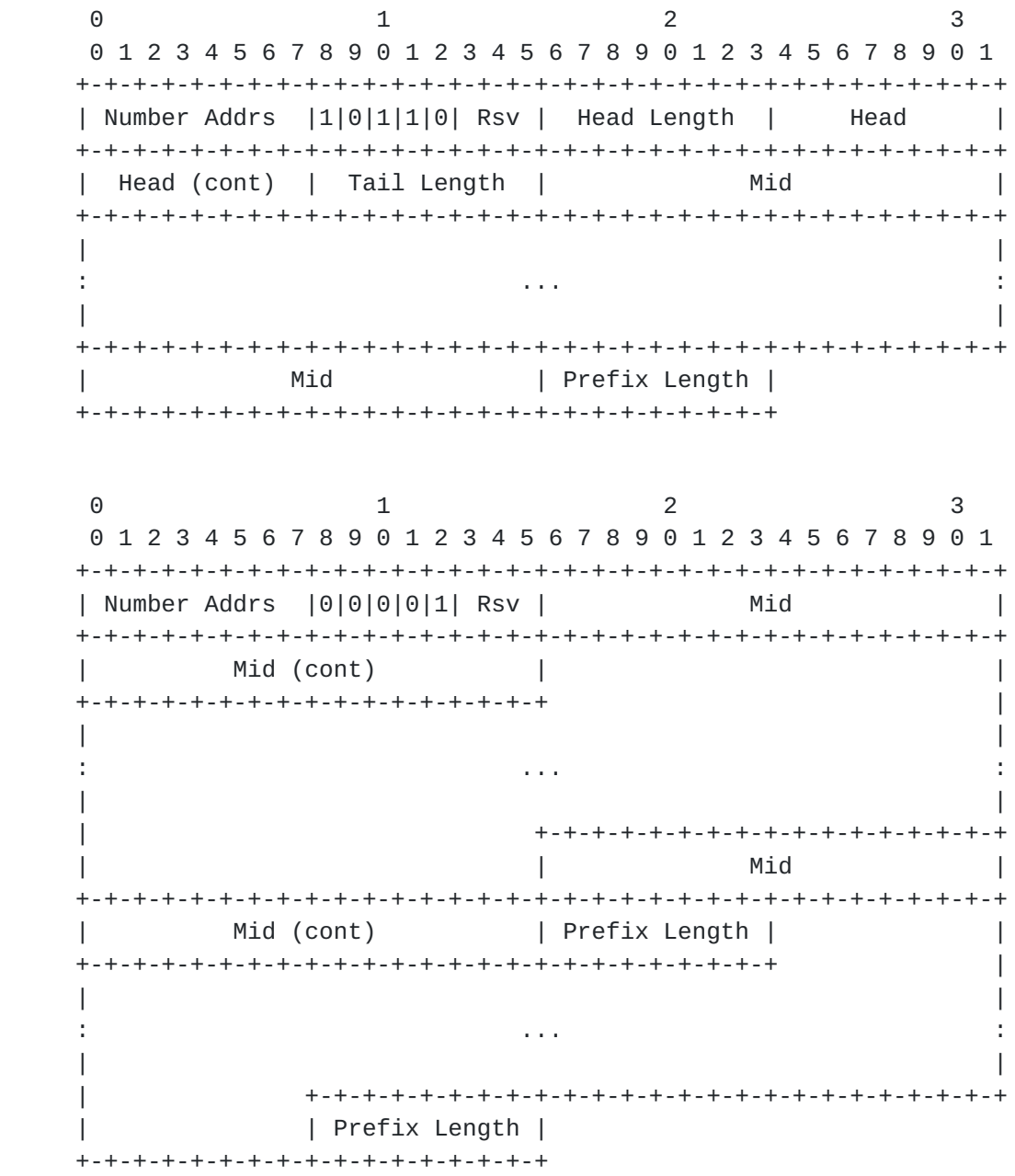

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
0 0 0 1 0
Rsv
Mid
Mid (cont)
...
Mid
Mid (cont)
Prefix Length

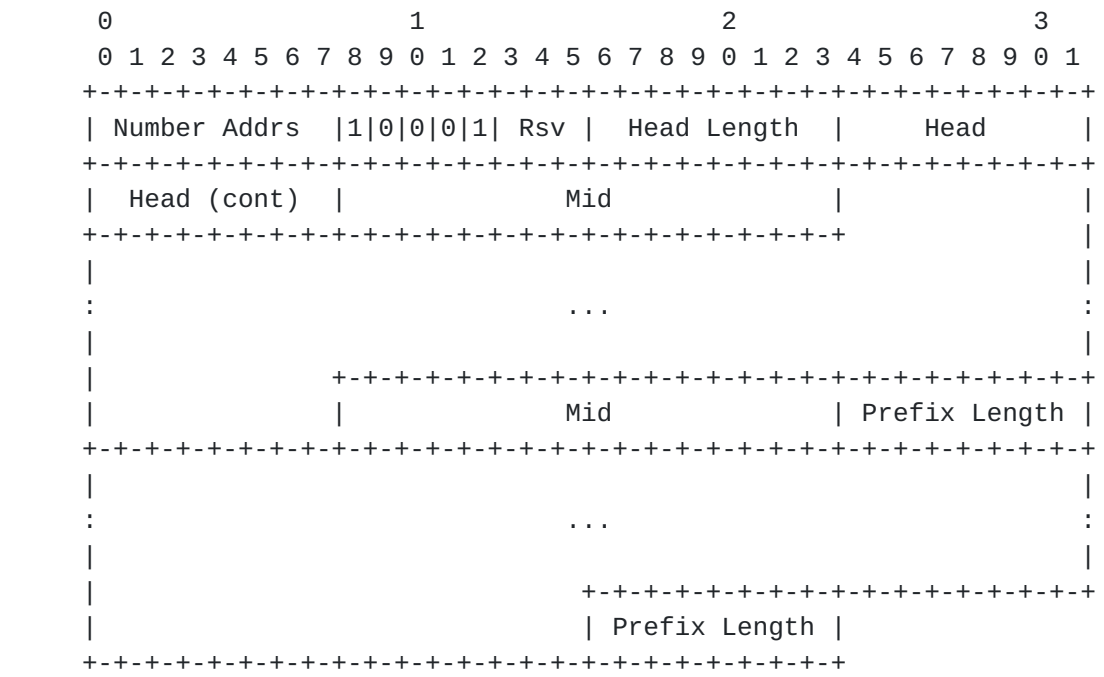

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
1 0 0 1 0
Rsv
Head Length
Head
Head (cont)
Mid
...
Mid
Prefix Length

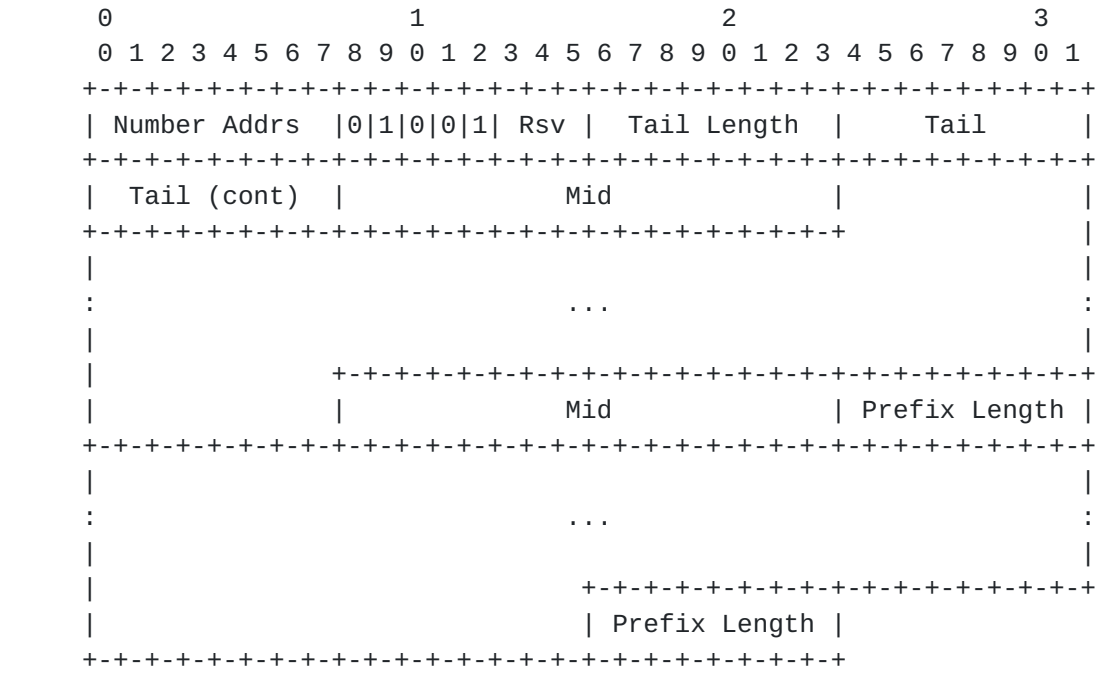

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
0|1|0|1|0|
Rsv
Tail Length
Tail
Tail (cont)
Mid
...
Mid
Prefix Length

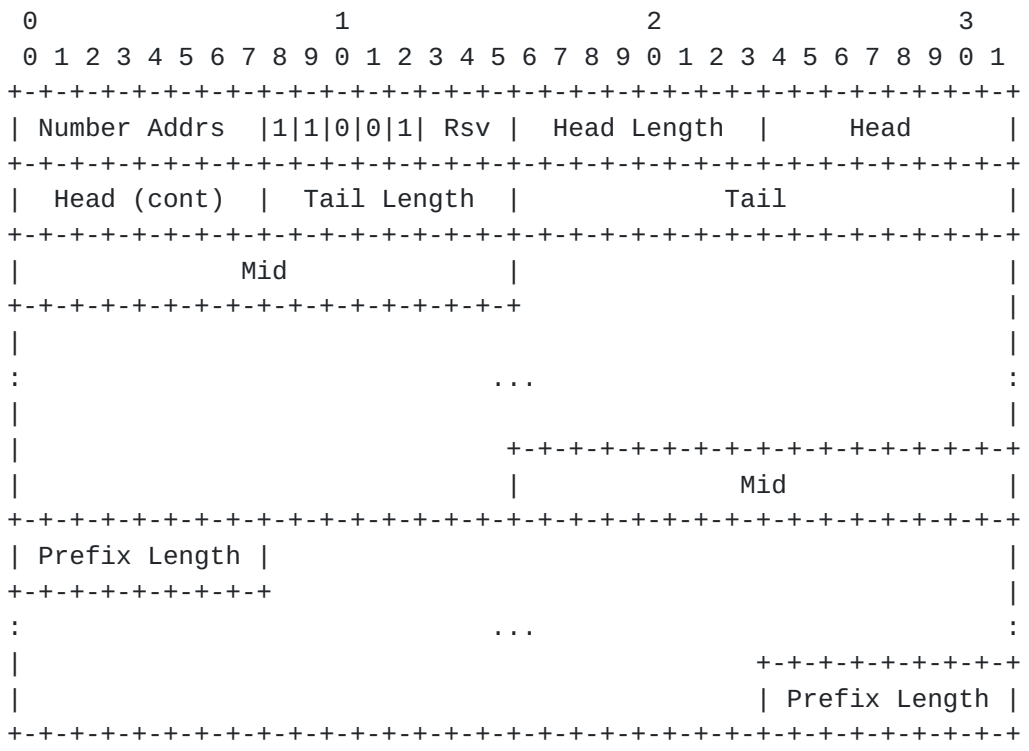

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
1|1|0|1|0|
Rsv
Head Length
Head
Head (cont)
Tail Length
Tail
Mid
...
Mid
Prefix Length

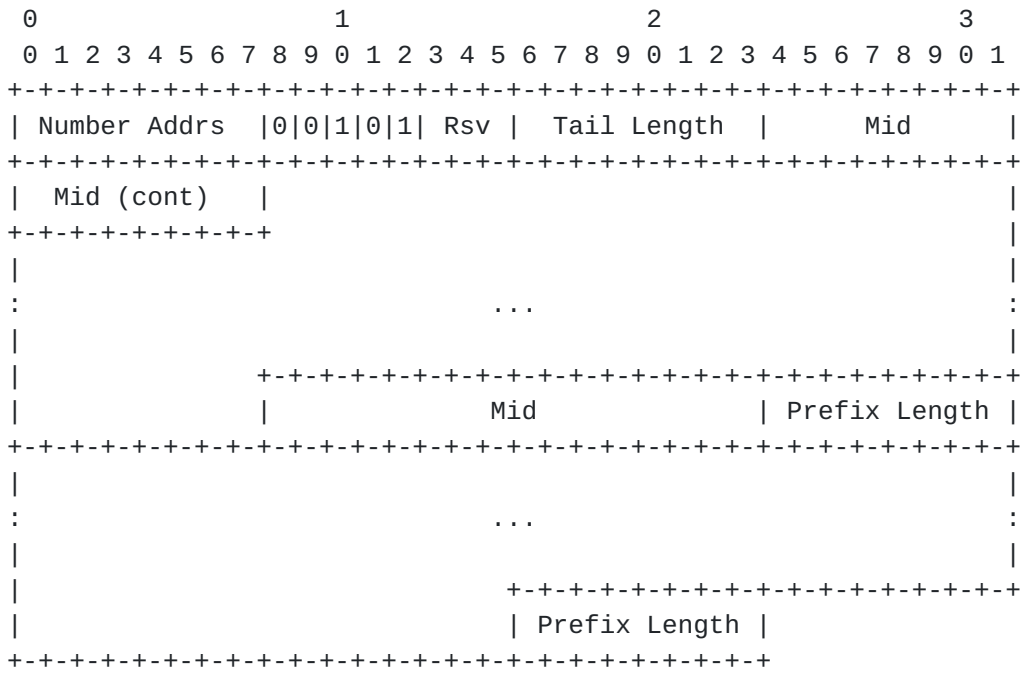

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
0|0|1|1|0|
Rsv
Tail Length
Mid
Mid (cont)
...
Mid
Prefix Length

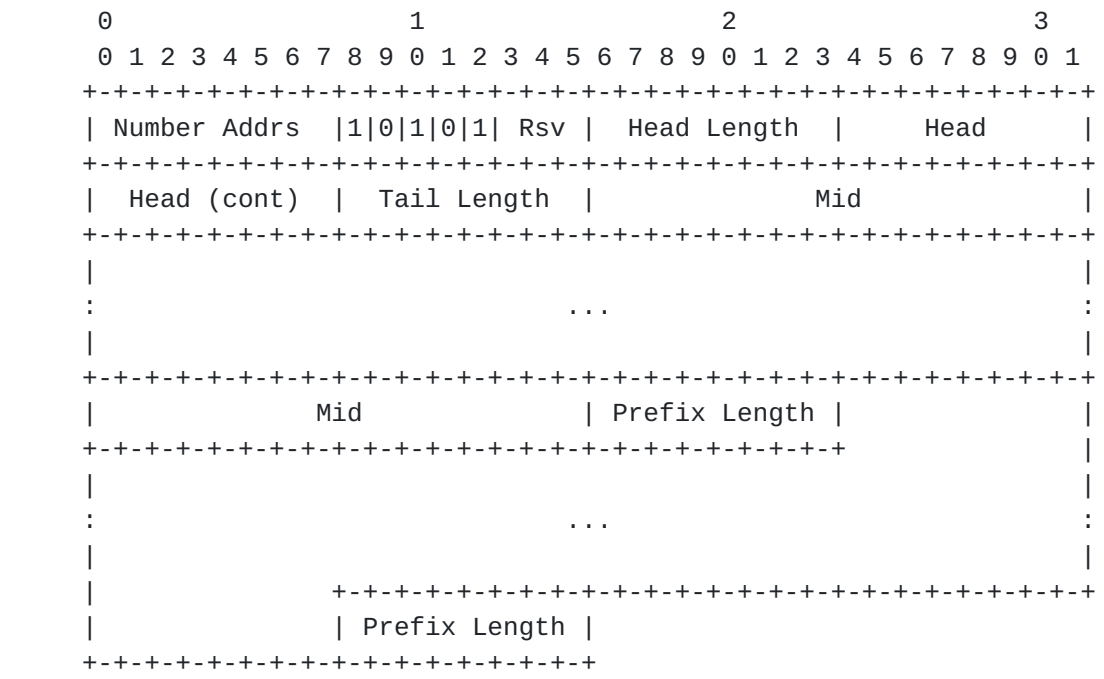

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
1 0 1 1 0
Rsv
Head Length
Head
Head (cont)
Tail Length
Mid
...
Mid
Prefix Length
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
0 0 0 0 1
Rsv
Mid
Mid (cont)
...
Mid
Mid (cont)
Prefix Length
...
Prefix Length


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
1 0 0 0 1
Rsv
Head Length
Head
Head (cont)
Mid
...
Mid
Prefix Length
...
Prefix Length


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
0 1 0 0 1
Rsv
Tail Length
Tail
Tail (cont)
Mid
...
Mid
Prefix Length
...
Prefix Length

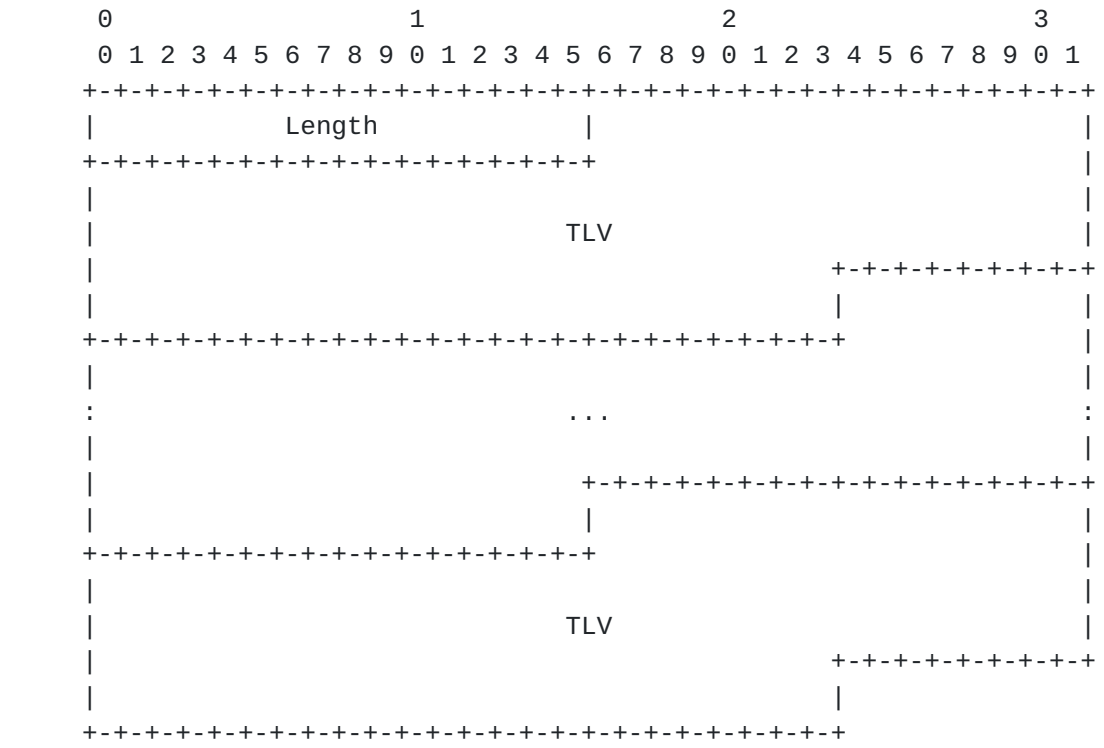

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
1 1 0 0 1
Rsv
Head Length
Head
Head (cont)
Tail Length
Tail
Mid
Prefix Length
...
Mid
Prefix Length


0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
0 0 1 0 1
Rsv
Tail Length
Mid
Mid (cont)
...
Mid
Prefix Length
...
Prefix Length



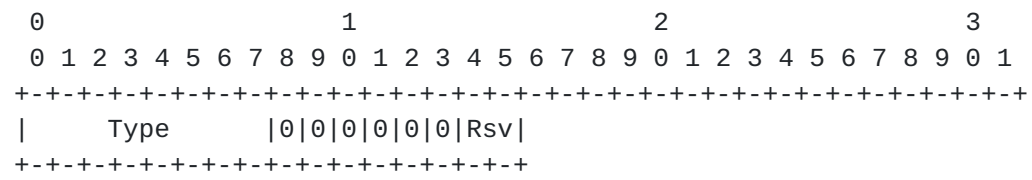
D.5. TLV Block

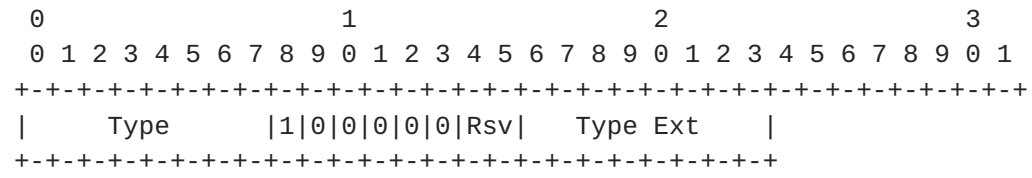
Format of a <tlv-block> element. There may be any number (zero or more) of TLVs, with total length of the TLVs (in octets) equal to the Length field.

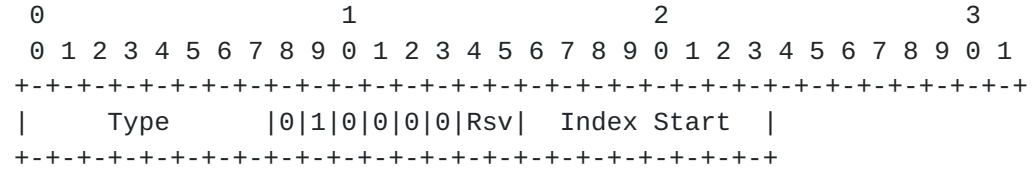


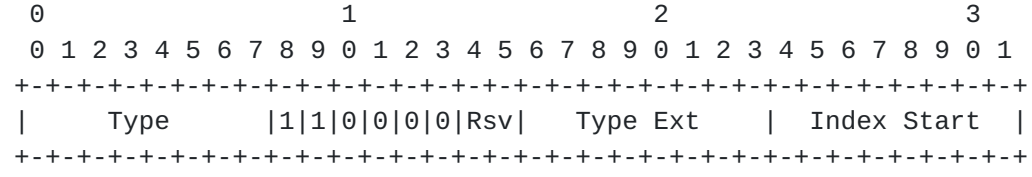
D.6. TLV

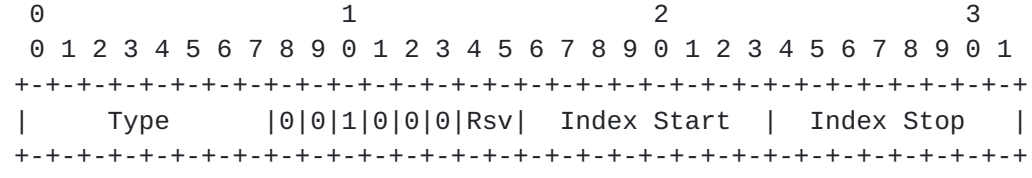
Possible options for the <tlv> element. These are differentiated by their second (flags) octet. If there are no index fields then this may be a packet, message or address block TLV, if there are one or two index fields then this must be an address block TLV. The Length field gives the length of the value field (in octets).

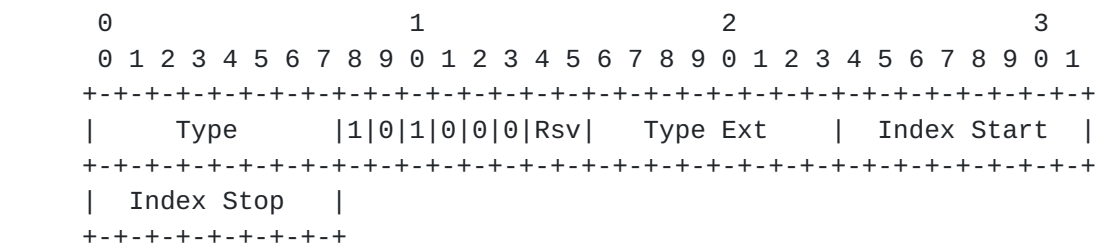


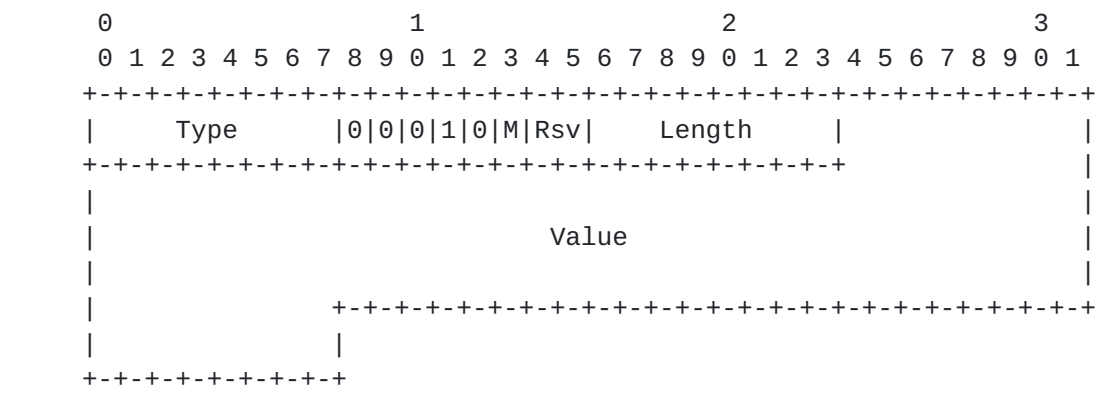


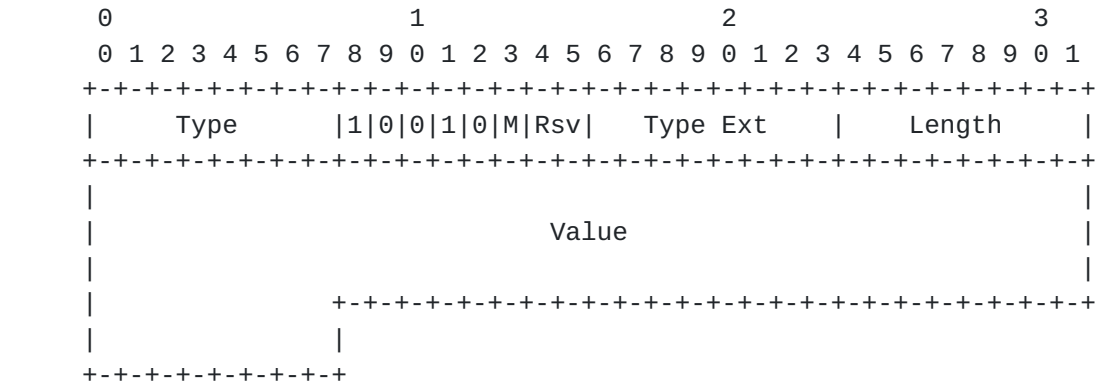


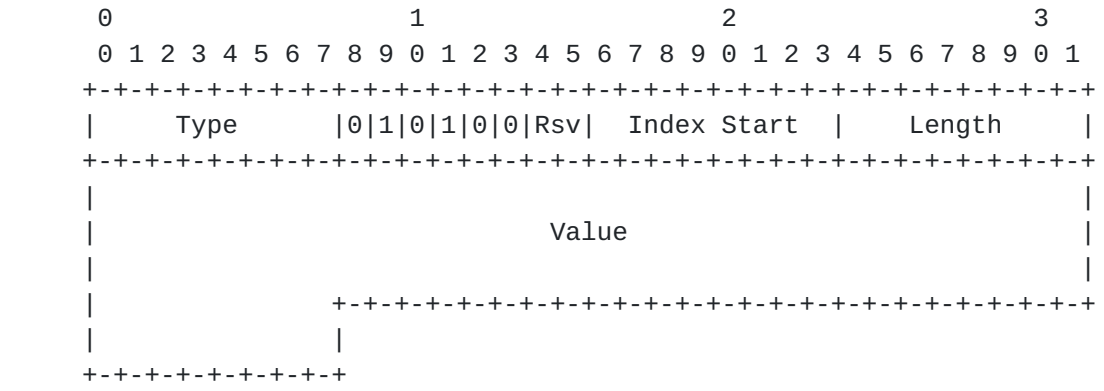


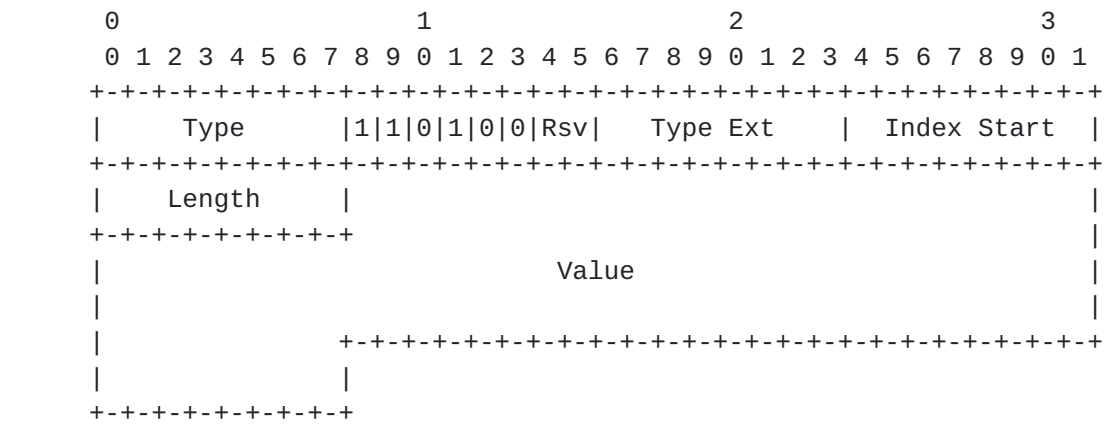


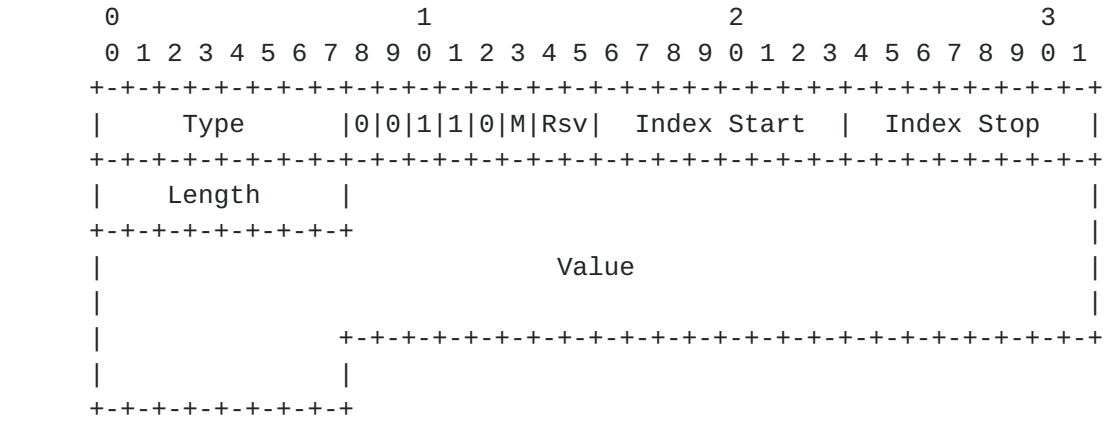

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
1 0 1 0 0 0 Rsv
Type Ext
Index Start
Index Stop

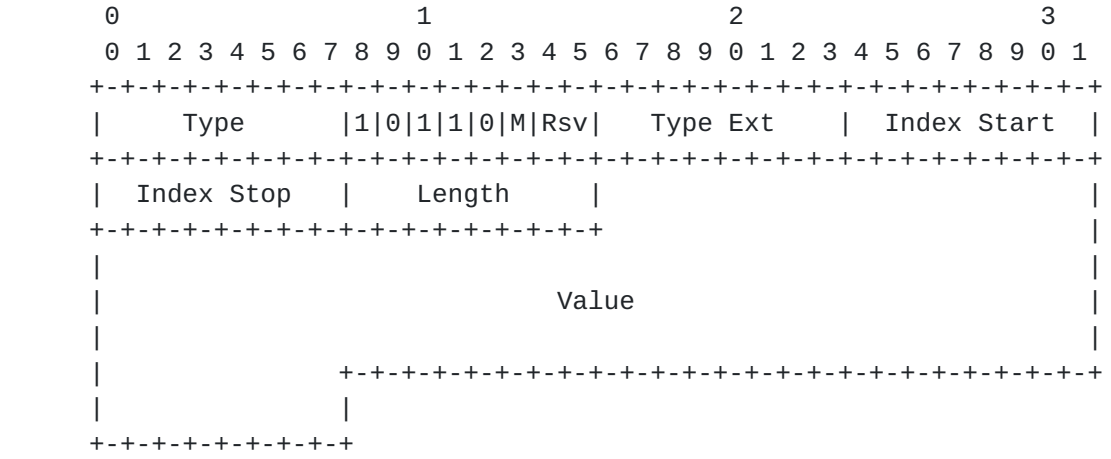

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
0 0 0 1 0 M Rsv
Length
Value

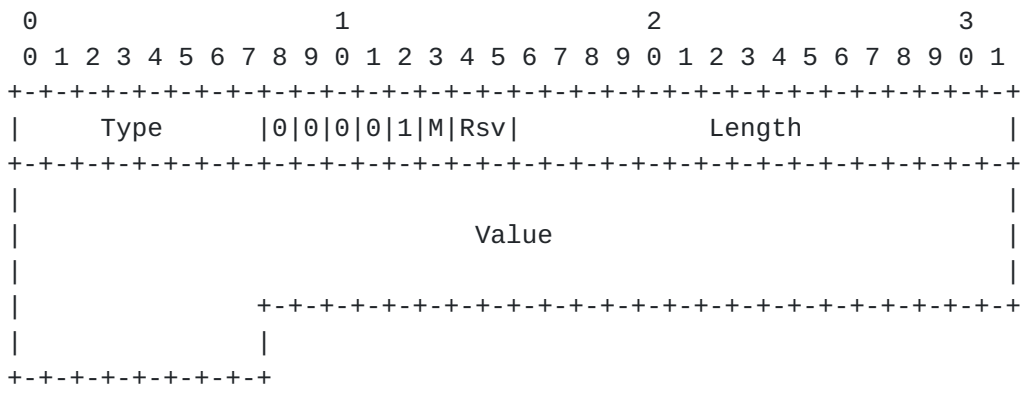

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
1 0 0 1 0 M Rsv
Type Ext
Length
Value

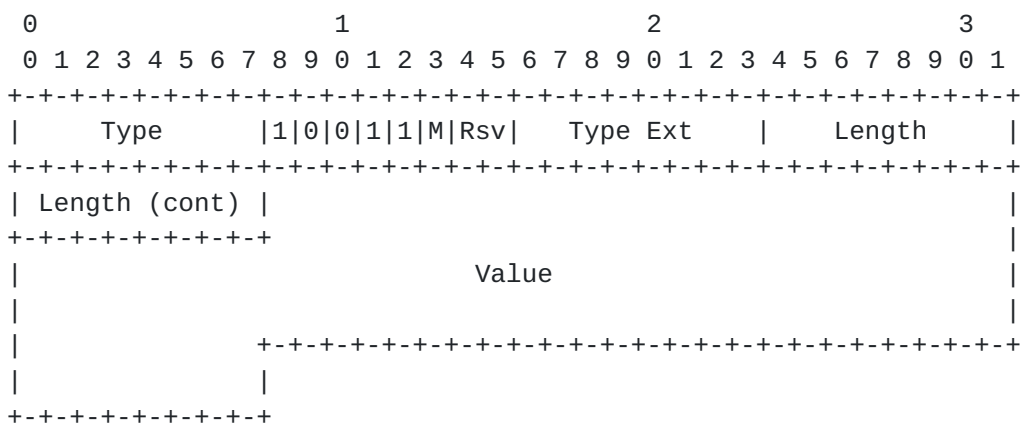

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
0 1 0 1 0 0 Rsv
Index Start
Length
Value

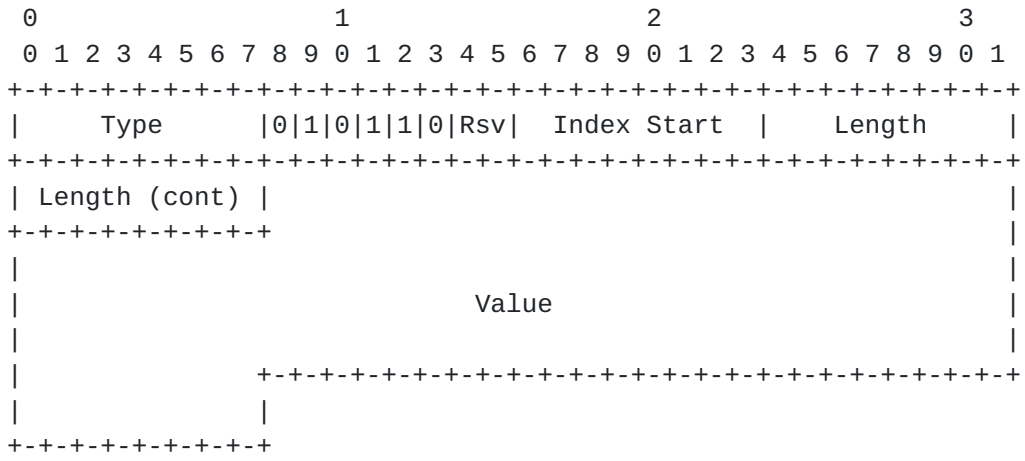

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
1 1 0 1 0 0 Rsv
Type Ext
Index Start
Length
Value

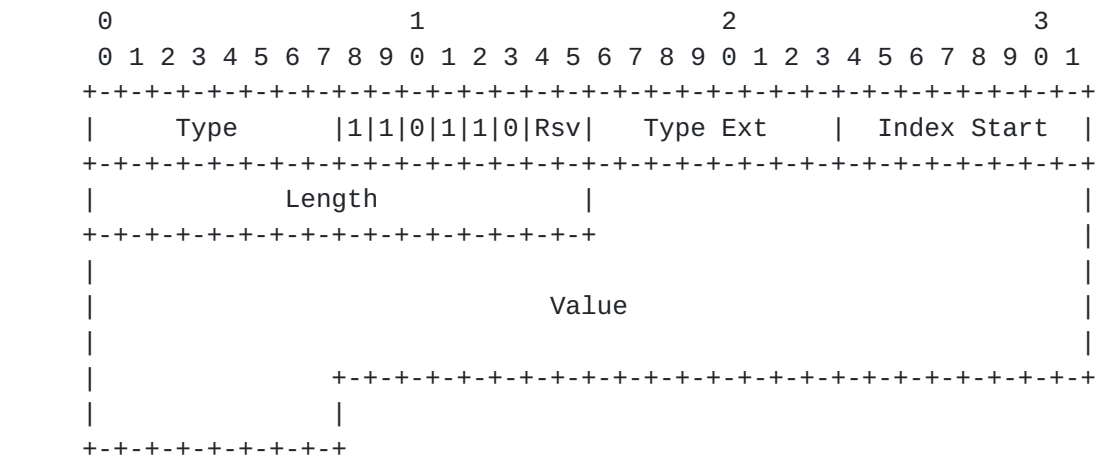

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
0 0 1 1 0 M Rsv
Index Start
Index Stop
Length
Value

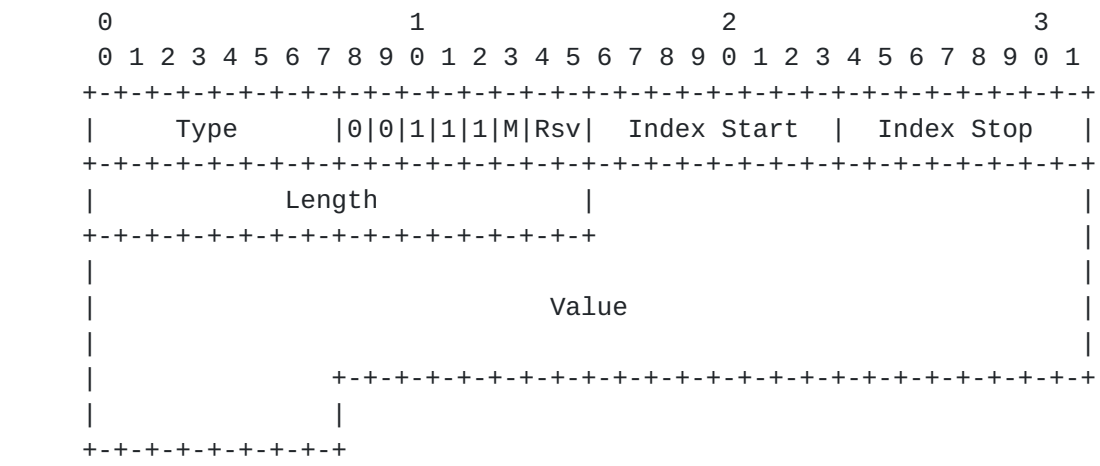

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
1 0 1 1 0 M Rsv
Type Ext
Index Start
Index Stop
Length
Value

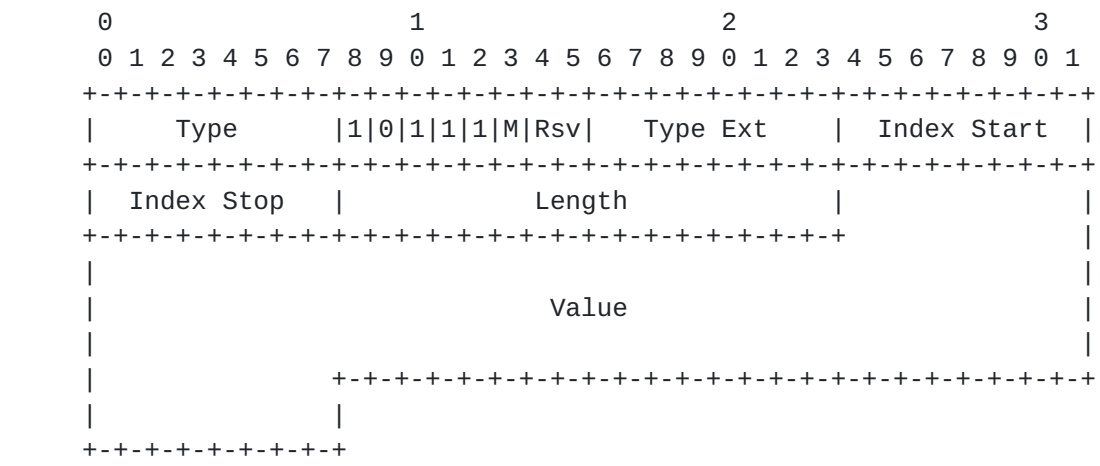

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
0 0 0 0 1 M Rsv
Length
Value


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
1 0 0 1 1 M Rsv
Type Ext
Length
Length (cont)
Value


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
0 1 0 1 1 0 Rsv
Index Start
Length
Length (cont)
Value


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
1 1 0 1 1 0 Rsv
Type Ext
Index Start
Length
Value


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
0 0 1 1 1 M Rsv
Index Start
Index Stop
Length
Value


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
1 0 1 1 1 M Rsv
Type Ext
Index Start
Index Stop
Length
Value

Appendix E. Complete Example

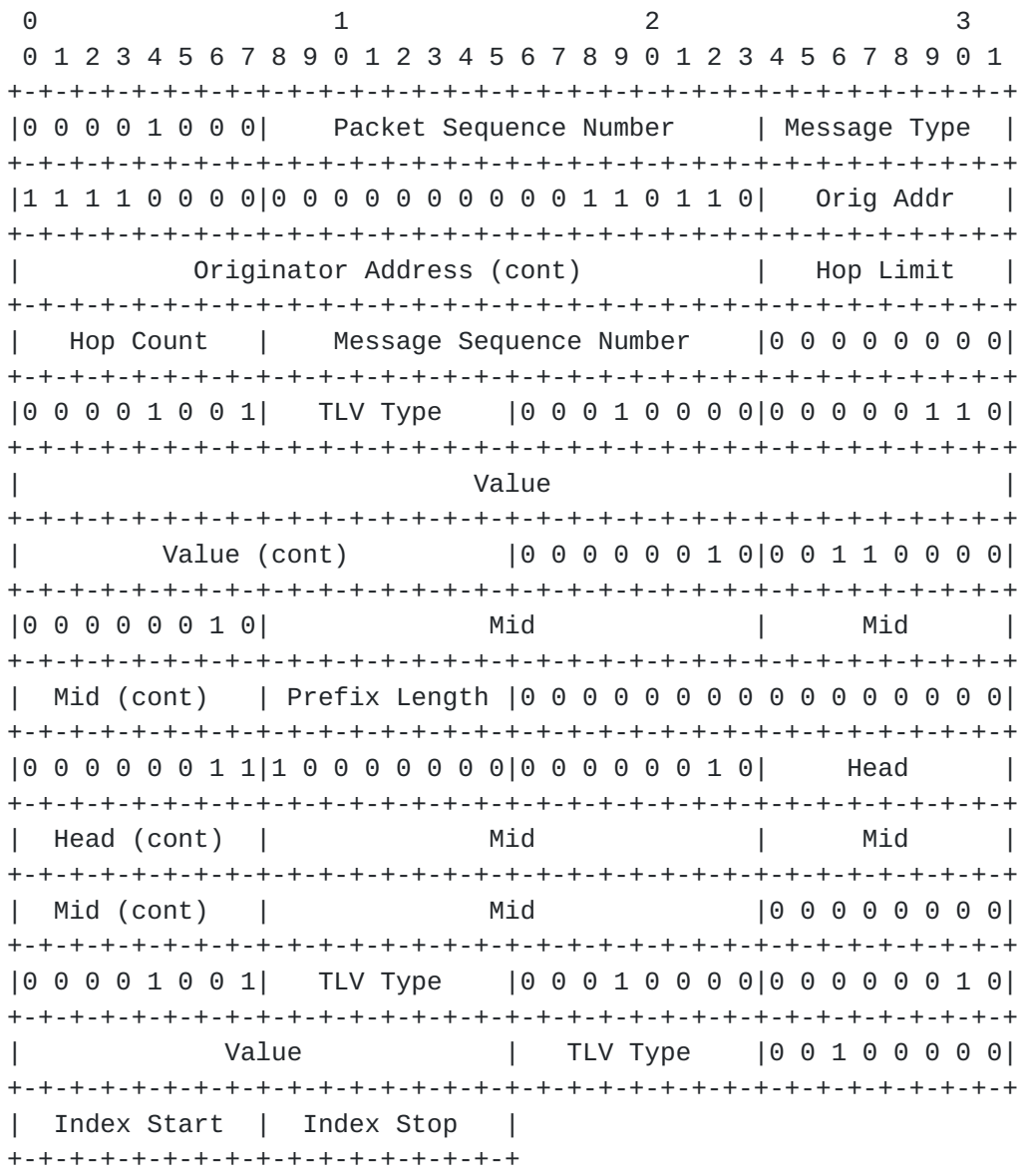
The following example packet, using IPv4 addresses (hence address-length is four octets) is included with the intent to exemplify how packet and message headers are constructed, and how addresses and attributes are encoded using address blocks and TLV blocks. This example is specifically not constructed to exhibit maximum message or packet size reduction. Appendix D contains illustrations of all syntactical elements.

The packet header has the phasseqnum flag set of its flags field set (value 8), and hence has a Packet Sequence Number, but no packet TLV block.

The packet contains a single message with length 54 octets. This message has the mhasorig, mhashoplimit, mhashopcount and mhasseqnum flags of its flags octet set (value 240), and hence includes an Originator Address, a Hop Limit, a Hop Count and a Message Sequence Number (which is type independent). The message has a message TLV block with content length 9 octets, containing a single message TLV. This TLV has the thasvalue flag of its flags octet set (value 16), and hence contains a Value field, with preceding value length 6 octets. The message then has two address blocks each with a following address TLV block.

The first address block contains 2 address prefixes. It has the ahaszerotail and ahassingleprelen flags of its flags octet set (value 48), and hence has no head (head-length is zero octets). It has a tail-length of 2 octets, hence mid-length is two octets. The two tail octets of each address are not included (since ahaszerotail is set) and have value zero. The address block has a single Prefix Length. The following address TLV block is empty (content length 0 octets).

The second address block contains 3 addresses. It has the ahashead flag of its flags octet set (value 128), and has head length 2 octets, no tail (tail-length is zero octets) and hence mid-length is two octets. It is followed by an address TLV block, with content length 9 octets, containing two address block TLVs. The first of these TLVs has the thasvalue flag of its flags octet set (value 16), and has a single Value of length 2 octets, which applies to all of the addresses in the address block. The second of these TLVs has the thasmultiindex flag of its flags octet set (value 32), and hence has no value length or value fields. It has two index fields (Index Start and Index Stop), which indicate those addresses this TLV applies to (inclusive range, counting from zero).



Appendix F. Contributors

This specification is the result of the joint efforts of the following contributors from the OLSRv2 Design Team, listed alphabetically:

- Cedric Adjih, INRIA, France, <Cedric.Adjih@inria.fr>
- Emmanuel Baccelli, INRIA, France, <Emmanuel.Baccelli@inria.fr>
- Thomas Heide Clausen, LIX, Ecole Polytechnique, France, <T.Clausen@computer.org>

- Justin W. Dean, NRL, USA<jdean@itd.nrl.navy.mil>
- Christopher Dearlove, BAE Systems, UK, <chris.dearlove@baesystems.com>
- Satoh Hiroki, Hitachi SDL, Japan, <hiroki.satoh.yj@sdl.hitachi.co.jp>
- Philippe Jacquet, INRIA, France, <Philippe.Jacquet@inria.fr>
- Monden Kazuya, Hitachi SDL, Japan, <kazuya.monden.vw@sdl.hitachi.co.jp>

Appendix G. Acknowledgments

The authors would like to acknowledge the team behind OLSR [RFC3626], including Anis Laouiti (INT, France), Pascale Minet, Laurent Viennot (both at INRIA, France), and Amir Qayyum (Center for Advanced Research in Engineering, Pakistan) for their contributions.

The authors would like to gratefully acknowledge the following people for intense technical discussions, early reviews and comments on the specification and its components (listed alphabetically):

- Brian Adamson (NRL)
- Teco Boot (Infinity Networks)
- Florent Brunneau (LIX)
- Ian Chakeres (Motorola)
- Alan Cullen (BAE Systems)
- Ulrich Herberg (LIX)
- Joe Macker (NRL)
- Yasunori Owada (Niigata University)
- Charlie E. Perkins (WiChorus)
- Andreas Schjonhaug (LIX)

and the entire IETF MANET working group.

Authors' Addresses

Thomas Heide Clausen
LIX, Ecole Polytechnique, France

Phone: +33 6 6058 9349
EMail: T.Clausen@computer.org
URI: http://www.thomasclausen.org/

Christopher M. Dearlove
BAE Systems Advanced Technology Centre

Phone: +44 1245 242194
EMail: chris.dearlove@baesystems.com
URI: http://www.baesystems.com/

Justin W. Dean
Naval Research Laboratory

Phone: +1 202 767 3397
EMail: jdean@itd.nrl.navy.mil
URI: http://pf.itd.nrl.navy.mil/

Cedric Adjih
INRIA Rocquencourt

Phone: +33 1 3963 5215
EMail: Cedric.Adjih@inria.fr
URI: http://menetou.inria.fr/~adjih/

Full Copyright Statement

Copyright (C) The IETF Trust (2008).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.