Mobile Ad hoc Networking (MANET)
Internet-Draft
Intended status: Standards Track
Expires: March 4, 2008

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

September 2007

Generalized MANET Packet/Message Format
draft-ietf-manet-packetbb-10

Status of this Memo

This Internet-Draft will expire on March 4, 2008.

Copyright Notice

Abstract

This document specifies a multi-message packet format that may be used by mobile ad hoc network routing and other protocols.

Table of Contents

1. Introduction

This document specifies the syntax of a general purpose multi-message packet format for information exchange between MANET 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 regular expressions.

This document specifies:

- A packet format, allowing zero or more messages to be contained within a single transmission, and optionally including a packet header. A packet with zero messages may be sent in case the only information to exchange is contained in the packet header (such as a "keep alive" sequence number).

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

- A message header format, which may allow a node to make praocessing and forwarding decisions based on the node's present state and the message header, without inspecting and parsing the message body. Message header information permits single- and multi-hop message diffusion.

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

- An address block format, where an address block represents sets of addresses in a compact (compressed) form.

- A generalized type-length-value (TLV) format representing attributes. Multiple TLVs can be included and each associated with a packet, a message, an address, or a set of addresses.

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

Parsing logic - the regular expression specification facilitates generic, protocol independent, parsing logic.

Extensibility - packets and messages defined by a protocol using this specification are extensible through defining new message types and new TLVs. Existing protocol specifications 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. 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, unless indicated otherwise. The structure of packet and message encoding allows parsing, verifying, and identifying individual elements in a single linear pass.

Separation of forwarding and processing - duplicate detection and controlled scope message forwarding decisions can be made using information contained in the message header, without processing the message body.

2. 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 RFC 2119. [1].

Additionally, this document uses the following terminology:

Packet - the top level entity in this specification. Packets are transmitted over a single logical hop and are not forwarded. A packet contains zero or more messages, and may contain a packet header.

Message - the fundamental entity carrying protocol information, in the form of addresses and TLVs. Messages are transmitted in packets, and may be forwarded based on their header information.

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.

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.

Element - a syntactic entity defined in the regular expression specification, represented using the notation <foo>.

<foo> - if <foo> is an 8 or 16 bit field then <foo> is also used to represent the value of that field.

? - zero or one occurrences of the preceding element.

* - zero or more occurrences of the preceding element.

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

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

3. Applicability Statement

This specification describes a generic multi-message packet format, for carrying MANET routing protocol signals. The specification has been developed from that used by OLSR (The Optimized Link State Routing Protocol) [4].

The specification is designed with IP (IPv4/IPv6) in mind. All addresses within a control message are assumed to be of the same size, deduced from IP. In the case of mixed IPv6 and IPv4 addresses, IPv4 addresses are represented as IPv4-mapped IPv6 addresses as specified in [2].

The messages defined by this specification are designed to carry routing protocol signals between MANET routers, and to support scope limited diffusion, as well as point to point signaling in a multi-hop network.

The packets defined by this specification are designed to carry a number of messages in a single transmission. The packets may be unicast or multicast, may be transmitted as appropriate to the protocol using this specification and may travel over a single logical hop which might consist of one or more IP hops.

This specification is particularly appropriate for extensible protocols. It offers external extensibility in the form of new message types. It offers internal extensibility in the form of TLVs, which may be added to existing message types.

A protocol using the multi-message packet format defined by this specification may constrain the syntax (for example requiring a specific set of message header fields) and features (for example specifying the suggested diffusion mechanism) that the protocol will employ.

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 or other protocol.

5. Signaling Framework

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 in the form of regular expressions. Illustrations of specified elements are given in Appendix B.

5.1. Packets

<packet> is defined by:

```
<packet> = {<pkt-header><pad-octet>*}?
           {<message><pad-octet>*}*
```

where <message> is defined in Section 5.2, and <pad-octet> is defined in Section 5.7. Successful parsing is terminated when all octets (defined using the variable pkt-size below) are used. A packet MUST contain either a <pkt-header> or at least one <message>.

<pkt-header> is defined by:

```
<pkt-header> = <zero>
               <pkt-semantics>
               <pkt-version>?
               <pkt-size>?
               <pkt-seq-num>?
               <tlv-block>?
```

where:

<zero> is an 8 bit field with all bits cleared ('0'). This field serves to identify that the packet starts with a <pkt-header>.

<pkt-semantics> is an 8 bit field, specifying the composition of the <pkt-header>:

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

bit 1 (phassize): if cleared ('0'), then <pkt-size> is not included in the <pkt-header>. if set ('1'), then <pkt-size> is included in the <pkt-header>.

bit 2 (pnoseqnum): if cleared ('0'), then <pkt-seq-num> is included in the <pkt-header>. If set ('1'), then <pkt-seq-num> is not included in the <pkt-header>.

bit 3 (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>.

bits 4-7: are RESERVED, and MUST each be cleared ('0') to be in conformance with this version of the specification.

<pkt-version> is omitted if the phasversion bit is cleared ('0'), otherwise is an 8 bit field, specifying the version of this specification.

<pkt-size> is omitted if the phassize bit is cleared ('0'), otherwise is a 16 bit field, specifying the size of the <packet> counted in octets.

<pkt-seq-num> is omitted if the pnoseqnum bit is set ('1'), otherwise is a 16 bit field, specifying a packet sequence number.

<tlv-block> is omitted if the phastlv bit is cleared ('0'), and is otherwise as defined in Section 5.5.

Note that since the message type zero is not used (see Section 7), the presence or absence of a packet header can be determined by inspecting the first octet of the packet.

pkt-version is a variable, defined to equal <pkt-version> if present, or 0 otherwise. See Section 5.3.

pkt-size is a variable, defined to equal <pkt-size> if present, or the size of the payload of the transport protocol employed otherwise.

5.2. Messages

Information is carried through 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 addresses.

o A TLV block, containing zero or more TLVs, following each address block.

<message> is defined by:

```
<message>    = <msg-header>
               <tlv-block>
               {<addr-block><tlv-block>}*

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

where:

<tlv-block> is as defined in Section 5.5.

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

<msg-type> is an 8 bit field, specifying the type of message. A type with all bits cleared ('0') MUST NOT be used.

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

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

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

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

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

bit 4 (mnoseqnum): If cleared ('0'), then <msg-seq-num> is included in the <msg-header>. If set ('1'), then <msg-seq-num> is not included in the <msg-header>.

bit 5 (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 bit MUST be cleared ('0') if the mnoorig bit is set ('1').

bits 6-7: are RESERVED and MUST each be cleared ('0') to be in conformance with this version of the specification.

If bit 1 (mnoorig) and bit 4 (mnoseqnum) are both cleared, then the message header provides for duplicate suppression.

If bit 2 (mnohoplimit) is cleared, then the message header provides for scope-delimited forwarding.

<msg-version> is omitted if the mhasversion bit is cleared ('0'), otherwise is an 8 bit field, specifying the version of this specification.

<msg-size> is a 16 bit field, specifying the size of the <message>, counted in octets.

<msg-orig-addr> is omitted if the mnoorig bit is set ('1'), otherwise is an identifier of length equal to address-length, which serves to uniquely identify the node that originated the message.

<msg-hop-limit> is omitted if the mnohoplimit bit is set ('1'), otherwise is an 8 bit field, which contains the maximum number of logical hops that the message should be further transmitted.

<msg-hop-count> is omitted if the mnohopcount bit is set ('1'), otherwise is an 8 bit field, which contains the number of logical hops that the message has traveled.

<msg-seq-num> is omitted if the mnoseqnum bit is set ('1'), otherwise is a 16 bit field, which contains a unique number, generated by the message's originator node. The <msg-orig-addr>, <msg-seq-num>, and, if the mistypedep bit in the <msg-semantics> field is set, the <msg-type> of a message serves to uniquely identify the message in the network (within the time period until <msg-seq-num> wraps around to a matching value).

msg-version is a variable, defined to equal <msg-version> if present, or to pkt-version otherwise. See Section 5.3.

5.3. Packet and Message Versioning

This specification defines packets and messages of version 0 (zero), so that pkt-version and each msg-version MUST equal zero. This SHOULD be indicated by clearing the phasversion and mhasversion bits in <pkt-semantics> and <msg-semantics>, respectively.

A protocol using this specification, or any future version (i.e. where pkt-version or msg-version are different from zero) of this specification MUST specify appropriate behavior in the case where an incoming packet or message indicates a pkt-version or msg-version different from the one used by that protocol, e.g. by discarding the packet or message.

5.4. Address Blocks

An address is specified as a sequence of address-length octets of the form head:mid:tail. An address block is an ordered set of addresses sharing the same head and tail, and having individual mids. Network addresses may be represented using the PREFIX_LENGTH TLV defined in Section 6.

<address-block> is defined by:

```
    <address-block> = <num-addr>
                      <addr-semantics>
                      <head-length>?
                      <head>?
                      <tail-length>?
                      <tail>?
                      <mid>*
```

where:

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

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

bit 0 (anohead): if cleared ('0'), then <head-length> is included in the <address-block>, and <head> may be included in the <address-block>. If set ('1'), then <head-length> and <head> are not included in the <address-block>.

bit 1 (anotail) and bit 2 (ahaszerotail): MUST NOT both be set ('1'). Otherwise, they are interpreted according to Table 1.

anotail	ahaszerotail	<tail-length>	<tail>
0	0	included	may be included
0	1	included	not included
1	0	not included	not included

Table 1

bits 3-7: are RESERVED and MUST each be cleared ('0') to be in accordance with this version of the specification.

<head-length> if present is an 8 bit field, which contains the total length (in octets) of the head of all of the addresses.

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

<head> is omitted if head-length == 0, otherwise it is a field of the head-length leftmost octets of all the addresses.

<tail-length> if present is an 8 bit field, which contains the total length (in octets) of the tail of all of the addresses.

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

<tail> is omitted if tail-length == 0 or if the ahaszerotail bit is set ('1'), otherwise it is a field of the tail-length rightmost octets of all the addresses. If the ahaszerotail bit is set ('1') then the tail-length rightmost octets of all the addresses 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 == 0, otherwise each <mid> is a field of length mid-length octets, representing the mid of the corresponding address in the address block.

5.5. TLVs and TLV Blocks

A TLV block is defined by:

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

where:

<tlvs-length> is a 16 bit field, which contains the total length (in octets) of all of the immediately following <tlv> elements.

<tlv> is as defined in Section 5.5.1.

5.5.1. TLVs

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

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

<tlv> is defined by:

```
    <tlv> = <tlv-type>
            <tlv-semantics>
            <tlv-type-ext>?
            <index-start>?
            <index-stop>?
            <length>?
            <value>?
```

where:

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

<tlv-semantics> 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 (tisextended) and bit 2 (tnovalue): MUST NOT both be set ('1'). Otherwise, they are interpreted according to Table 2.

tisextended	tnovalue	<length>	<value>
0	0	8 bits	included
0	1	not included	not included
1	0	16 bits	included

Table 2

bit 3 (tnoindex) and bit 4 (thassingleindex): MUST NOT both be set ('1'). The former MUST be set ('1') and the latter MUST be cleared ('0') for packet or message TLVs. They are interpreted according to Table 3.

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

Table 3

bit 5 (tismultivalue): this bit serves to specify how the <value> field is interpreted, as specified below. This bit MUST be cleared ('0') for packet or message TLVs, if the thassingleindex bit is set ('1'), or if the tnovalue bit is set ('1').

bits 6-7: are RESERVED and MUST each be cleared ('0') to be in accordance with this version of the specification.

<tlv-type-ext> is an 8 bit 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 are each an 8 bit field, interpreted as follows.

index-start and index-stop are variables, defined according to Table 4. 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

tnoindex	thassingleindex	index-start =	index-stop =
0	0	<index-start>	<index-stop>
0	1	<index-start>	<index-start>
1	0	0	end-index

Table 4

For an address block TLV, the TLV applies to the addresses from position index-start to position index-stop (inclusive) in the address block, where the first address has position zero.

number-values is a variable, defined by:

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

<length> is omitted or is an 8 or 16 bit field according to Table 2. If the tismultivalue bit 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 bit is cleared ('0'), then the variable single-length is defined by:

* single-length = <length>

<value> if present (see Table 2) is a field of length <length> octets. In an address block TLV, <value> is associated with the addresses from index-start to index-stop, inclusive. If the tismultivalue bit is cleared ('0') then the whole of this field is associated with each of the indicated addresses. If the tismultivalue bit 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 addresses.

5.5.2. TLV Inclusion and Constraints

TLVs associates attributes to entities, where entities are individual addresses in an address block, ranges of addresses in an address block, messages or packets. Inclusion of a TLV serves to associate an attribute to a given entity; the processing of this attribute and the relationship (e.g. interpretation, order of processing) between multiple different attributes associated to the same entity is to be defined by the protocol which uses this specification.

The following constraints on TLV ordering MUST be respected:

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

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

- Two or more TLVs with the same tlv-fulltype in the same address block TLV block MUST NOT be associated with the same copy of an address (i.e. they must not have overlapping index ranges).

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

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 linear 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, which may apply to ranges of addresses, ensure that encoded information (entity-attributes) can be unambiguously decoded, and that verification of such is a linear operation.

5.6. Malformed Elements

An element is malformed if it cannot be parsed according to its syntactical specification (including if there are insufficient octets available when a length is specified, in particular if there are fewer than pkt-size octets overall) or if a constraint (including, but not only, those specified in Section 5.5.2) 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.

5.7. Padding

Packet headers and messages MAY be padded to ensure 32 bit alignment of each message contained within the packet and of the overall packet length. Padding MAY also be used to ensure that all packets and/or messages have the same size.

All elements are an integer multiple of octets, hence padding can be accomplished by inserting an integer number of <pad-octet> elements after the element that is to be 32 bit aligned.

The number of <pad-octet> elements required to achieve this 32 bit alignment is the smallest number (0 to 3) that, when added to the size of the preceding elements, produces an integer multiple of 4.

<pad-octet> is an 8 bit field with all bits cleared ('0').

There is no need to indicate if padding is included, since a <pad-octet> will always precede either a message or the end of the packet. In the former case, the start of a message is indicated by the next non-zero octet parsed.

The padding after a message may be freely changed when a message is forwarded without affecting the message.

6. TLV specification

This document specifies one address block TLV, which is included to allow a standardized way of representing network addresses.

6.1. Address Block TLV Specification

Name	Type	Type extension	Length	Value
PREFIX_LENGTH	0	0	8 bits	Indicates that the address is a network address, rather than a host address. The value is the length of the prefix/netmask, in bits.

Table 5

An address in an address block without an associated PREFIX_LENGTH TLV may be considered to have a prefix length equal to the address length in bits (i.e. 8 * address-length).

7. IANA Considerations

7.1. Message Types

A new registry for message types must be created with initial assignments as specified in Table 6. Future values in the range 5-127 can be allocated using standards action [3]. Additionally, values in the range 128-255 are reserved for private/local use.

Type	Description
0	MUST NOT be allocated.
1-4	RESERVED

Table 6

Message type 0 MUST NOT be allocated because a zero-octet signifies a packet header and zero-octets are used for padding. Message types 1 to 4 are reserved because they are used by OLSR [4], which uses a compatible packet/message header format.

7.2. Packet TLV Types

A new registry for packet TLV types must be created, with no initial assignments.

Future values in the range 0-127 can be allocated using standards action [3], respecting the following guidelines:

values 0-7 - MUST NOT be assigned 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.

values 8-127 - no constraints.

Additionally, values in the range 128-255 are reserved for private/local use. If a packet TLV type is allocated 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 [3].

7.3. Message TLV Types

A new registry for message TLV types must be created with no initial assignments.

Future values in the range 0-127 can be allocated using standards action [3], respecting the following guidelines:

values 0-7 - MUST NOT be assigned 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.

values 8-127 - no constraints.

Additionally, values in the range 128-255 are reserved for private/local use. If a message TLV type is allocated 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 [3].

7.4. Address Block TLV Types

A new registry for address block TLV types must be created with initial assignments as specified in Table 7.

Mnemonic	Type	Type extension	Description
PREFIX_LENGTH	0	0	Indicates that associated addresses are network addresses, with given prefix length, in bits.
	0	1-255	RESERVED

Table 7

Future values in the range 1-127 can be allocated using standards action [3], respecting the following guidelines:

values 1-7 - MUST NOT be assigned 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 appearing 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.

values 8-127 - no constraints.

Additionally, values in the range 128-255 are reserved for private/ local use. If an address block TLV type is allocated then a new registry for type extensions of that type must be created. A document which defines an address block TLV type MUST also specify the mechanism by which its type extensions are allocated, from among those in [3].

8. 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 two mechanisms which are enabled by this specification, and may form part of a security approach in a protocol using this specification, are described in Section 8.1. There is however no requirement that a protocol using this specification should use either.

8.1. Security Suggestions

The security suggestions made here, are based on that:

- Messages are designed to be carriers of protocol information and MAY, at each hop, be forwarded and/or processed according to the information in the message header 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 security 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 may 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 zero ('0') if present.

- Hop-by-hop packet level security 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.

9. References

9.1. Normative References

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

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

[3] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 2434, BCP 26, October 1998.

9.2. Informative References

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

Appendix A. Examples

This appendix contains some examples of parts of this specification.

A.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 anohead and anotail bits of the semantics octet are set, and hence head-length = 0, tail-length = 0, mid-length = address-length, and the address block consists of the number of addresses, the semantics octet with value 3, and a list of the uncompressed addresses. This is also 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-semantics> has anotail set (value 2);
  - <tail-length> and <tail> are omitted.

  The address block is then 3 2 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-semantics> has anohead set (value 1);
  - <head-length> and <head> are omitted.

The address block is then 2 1 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-semantics> = 0.

  The address block is then 2 0 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-semantics> has ahaszerotail set (value 4);
  - <tail> is omitted.

  The address block is then 3 4 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-semantics> has anohead and ahaszerotail set (value 5);
  - <head> and <tail> are omitted.

  The address block is then 2 5 2 a b c d (7 octets).

A.2. TLV Examples

If network addresses a.b.0.0/16, c.d.0.0/16, c.d.e.0/24 and c.d.e.f/32 are to be represented using a single address block containing a.b.0.0, c.d.0.0, c.d.e.0 and c.d.e.f, with the prefix lengths added using one or more address block TLVs of type

PREFIX_LENGTH (0, with no type extension), then this can be done in a number of ways. Possible examples are:

- Using one multivalue TLV covering all of the addresses:
  - <tlv-semantics> has tnoindex and tismultivalue set (value 40);
  - <index-start> and <index-stop> are omitted;
  - <length> = 4 (single-length = 1).
  - The TLV is then 0 40 4 16 16 24 32 (7 octets).
- Using one multivalue TLV omitting the last address (a prefix length of 32 is the default):
  - <tlv-semantics> has tismultivalue set (value 32);
  - <index-start> = 0;
  - <index-stop> = 2
  - <length> = 3 (single-length = 1).
  - The TLV is then 0 32 0 2 3 16 16 24 (8 octets).
- Using two single value TLVs, omitting the last address. First:
  - <tlv-semantics> = 0;
  - <index-start> = 0;
  - <index-stop> = 1;
  - <length> = 1;
  - <value> = 16.
  - The TLV is then 0 0 0 1 1 16 (6 octets).

  Second:
  - <tlv-semantics> has thassingleindex set (value 16);
  - <index-start> = 2;
  - <index-stop> is omitted;

* <length> = 1;

* <value> = 24.

* The TLV is then 0 16 2 1 24 (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 EXAMPLE (and no type extension) which has no value (it is sufficient to simply indicate which addresses are examples), for the same address block, with the second and third addresses being examples, this can be indicated with a single TLV:

o <tlv-semantics> has tnovalue set (value 4);

o <index-start> = 1;

o <index-stop> = 2;

o <length> and <value> are omitted.

o The TLV is then EXAMPLE 4 1 2 (4 octets).

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

o <tlv-semantics> has tnoindex set (value 8);

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

o <length> = 8.

o The TLV is then DATA 8 8 a b c d e f g h (11 octets).

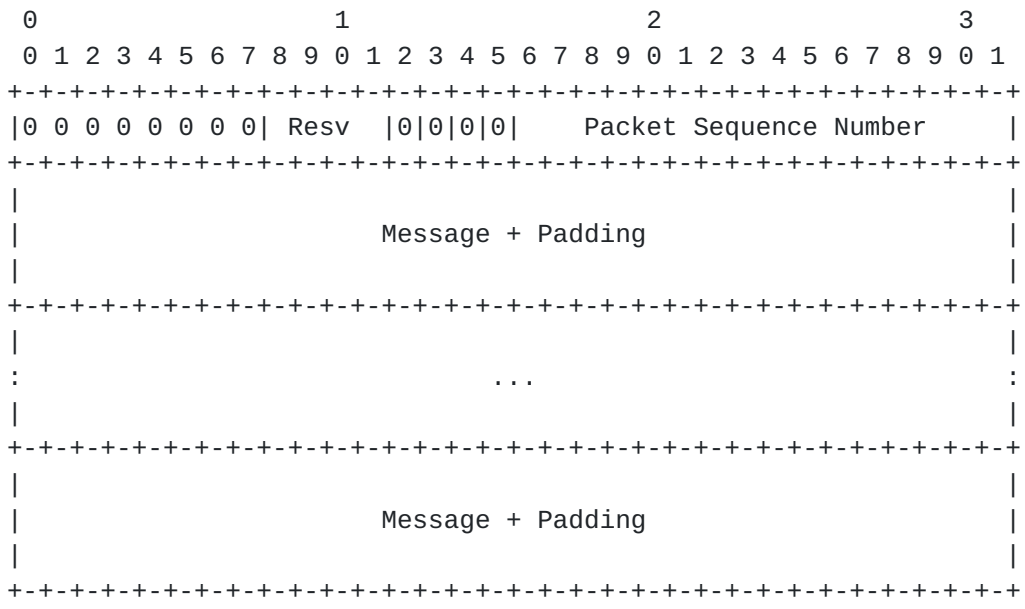
If, in this example, the number of data octets were 256 or greater then <tlv-semantics> would also have tisextended set and have value 10. The length would require two octets (most significant first). The TLV length would be 4 + N octets, where N is the number of data octets (it can be 3 + N octets if N is 255 or less).

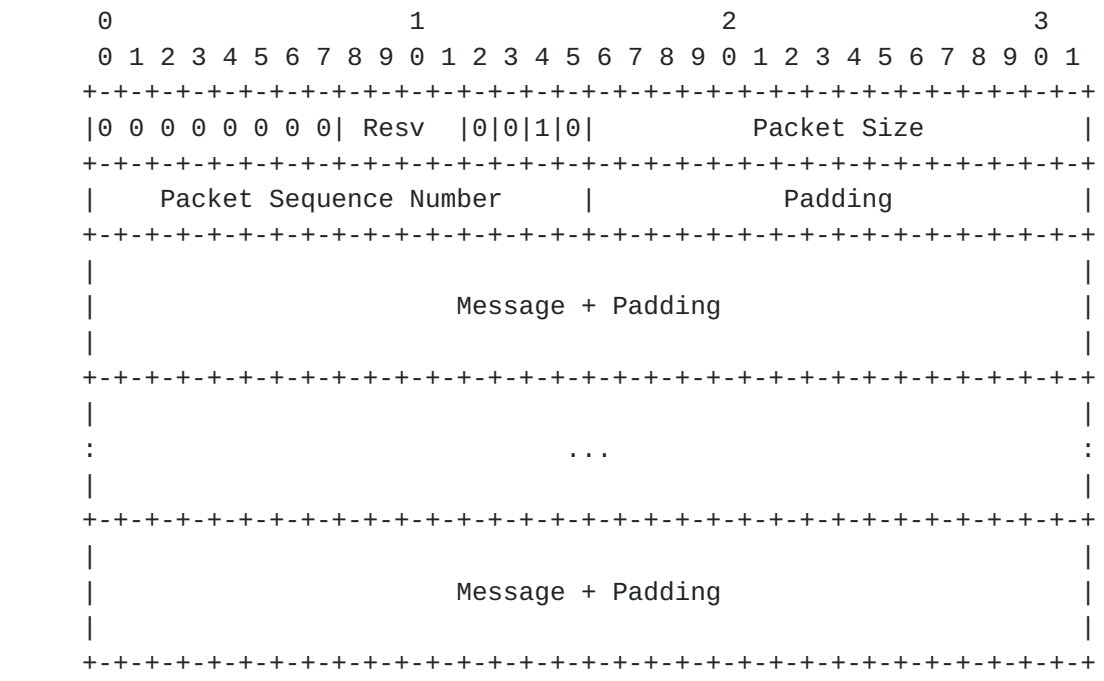
Appendix B. Illustrations

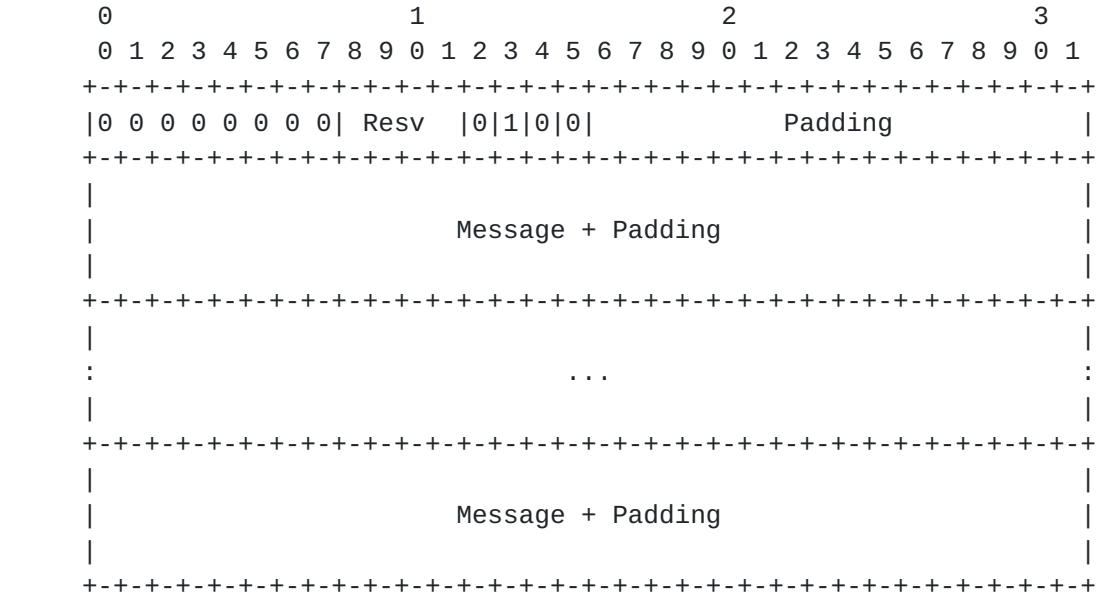
This informative appendix illustrates the elements, which are normatively specified in Section 5 using regular expressions. Examples with version fields are omitted, as not needed when using this version of this specification (although permitted).

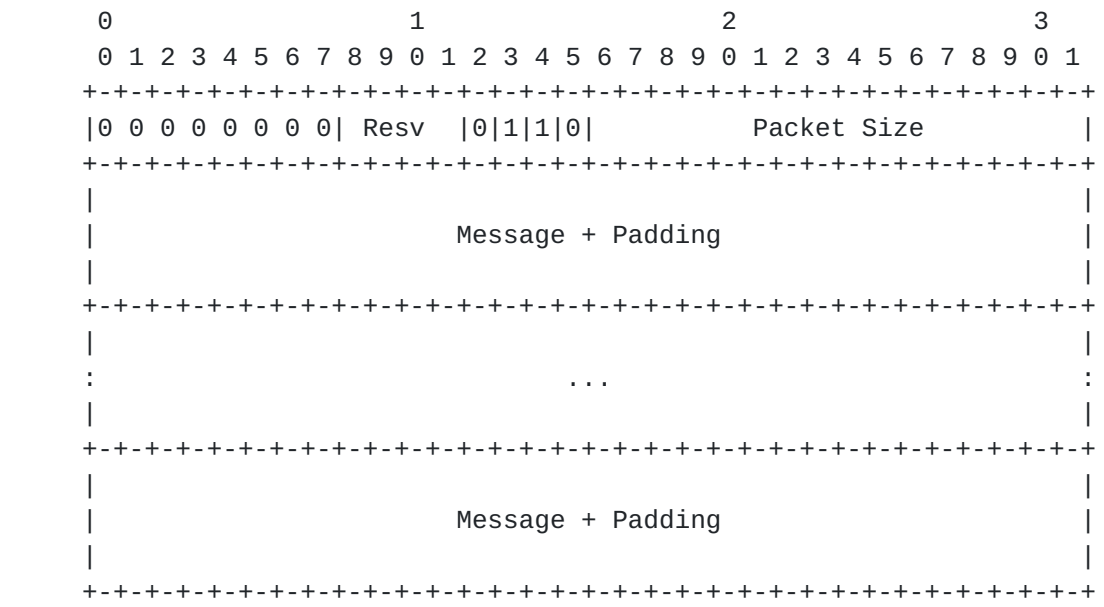
Bits labeled Resv, or Rsv are cleared ('0'). Bits labeled N or M may be cleared ('0') or set ('1'). Octets labeled Padding must be zero ('0'), and are optional. (They have been omitted where not needed for alignment.)

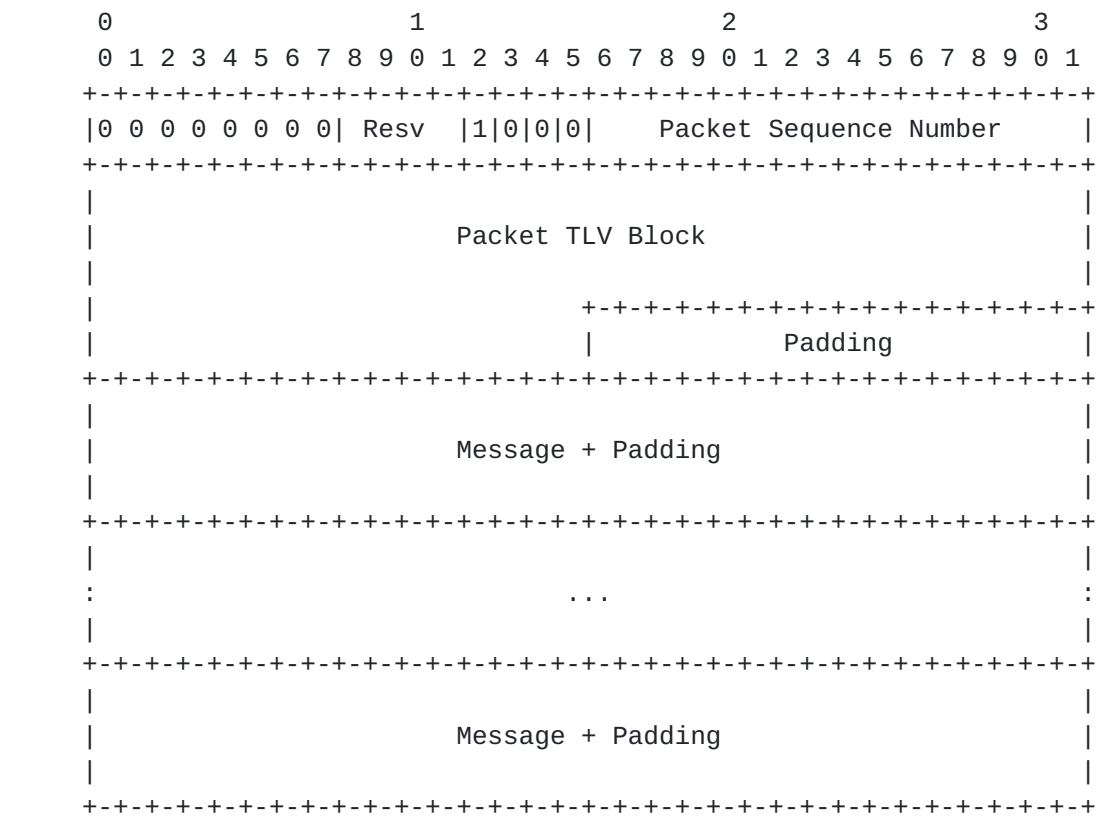
Appendix B.1. Packet

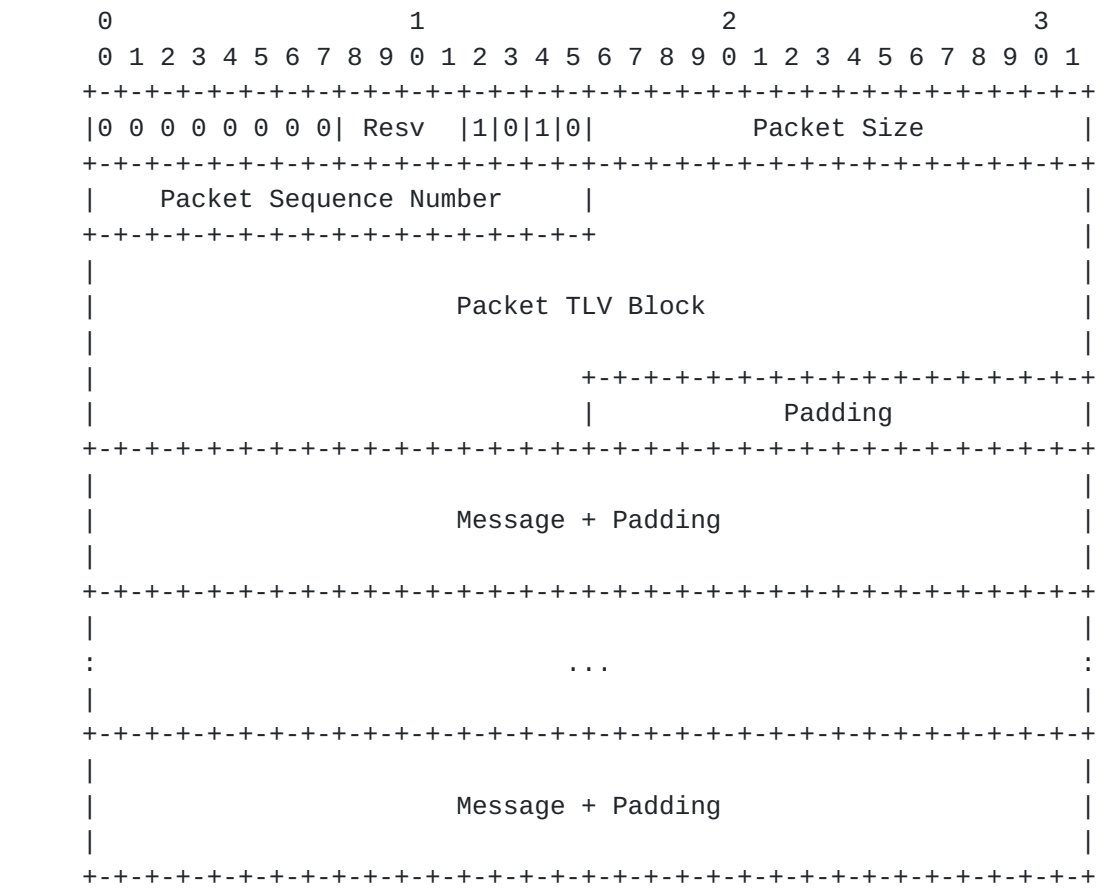


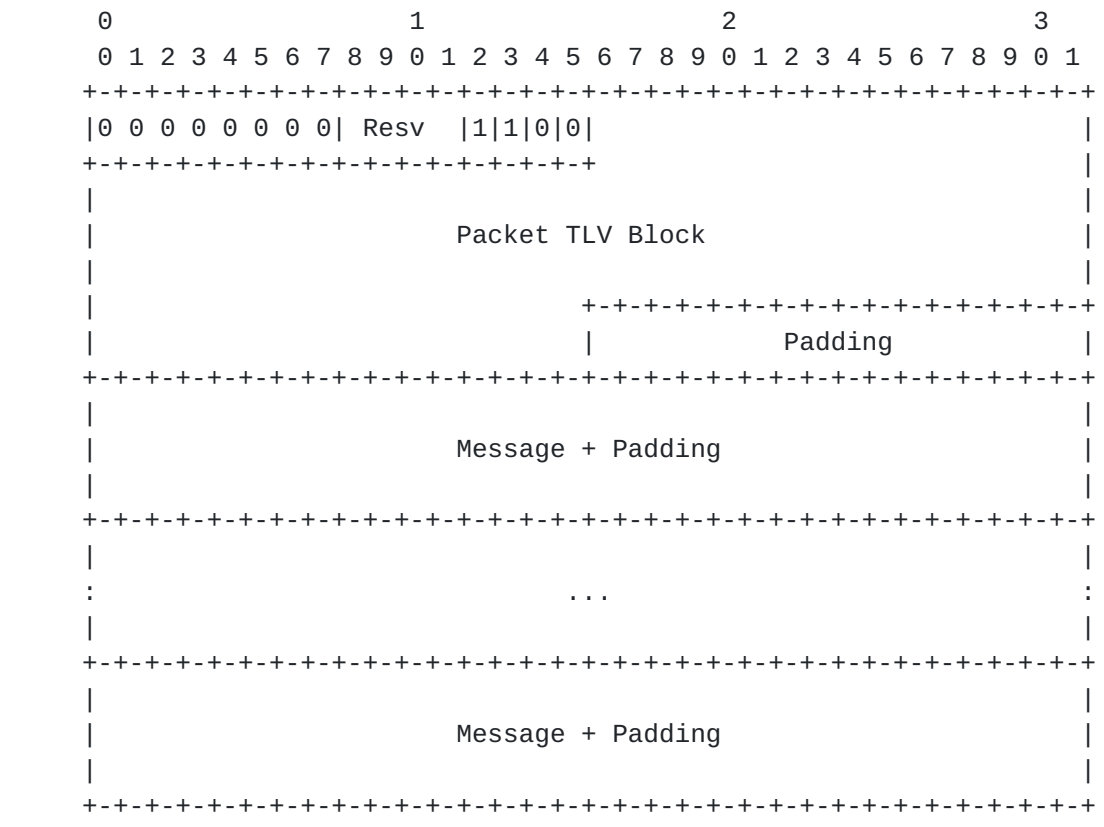

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
0 0 0 0 0 0 0 0 Resv 0 0 1 0 Packet Size
Packet Sequence Number Padding
Message + Padding
...
Message + Padding

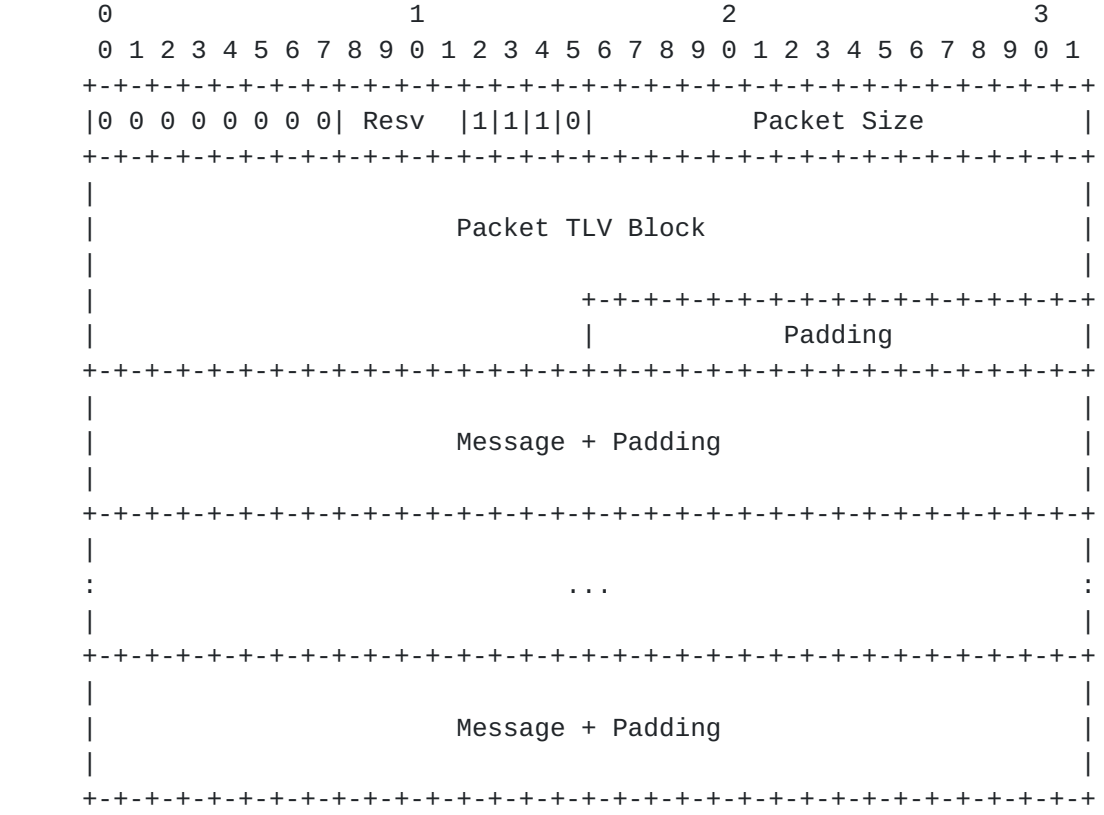

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
0 0 0 0 0 0 0 0 Resv 0 1 0 0 Padding
Message + Padding
...
Message + Padding

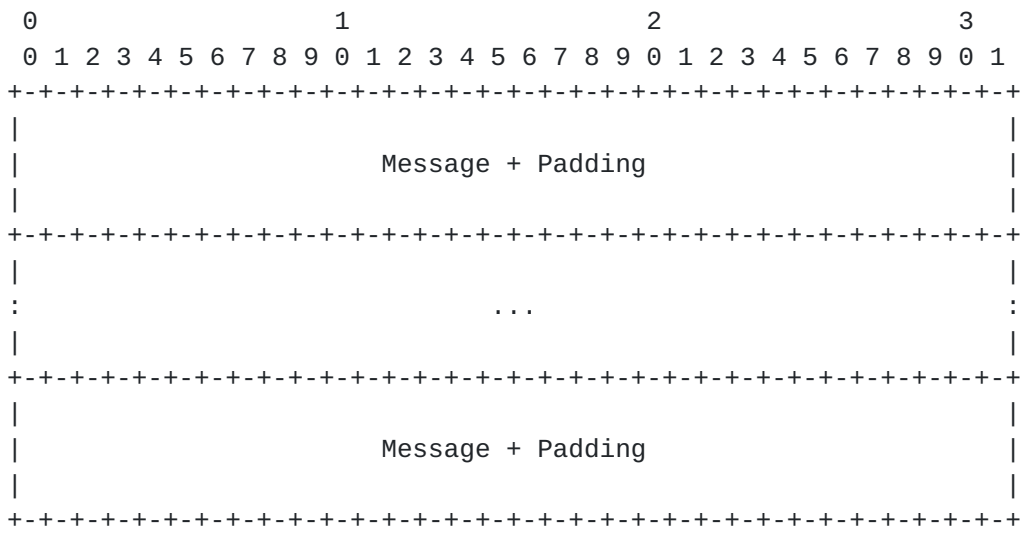

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
0 0 0 0 0 0 0 0 Resv 0 1 1 0 Packet Size
Message + Padding
...
Message + Padding


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 0 0 0 Resv 1 0 0 0 Packet Sequence Number
Packet TLV Block
Padding
Message + Padding
...
Message + Padding

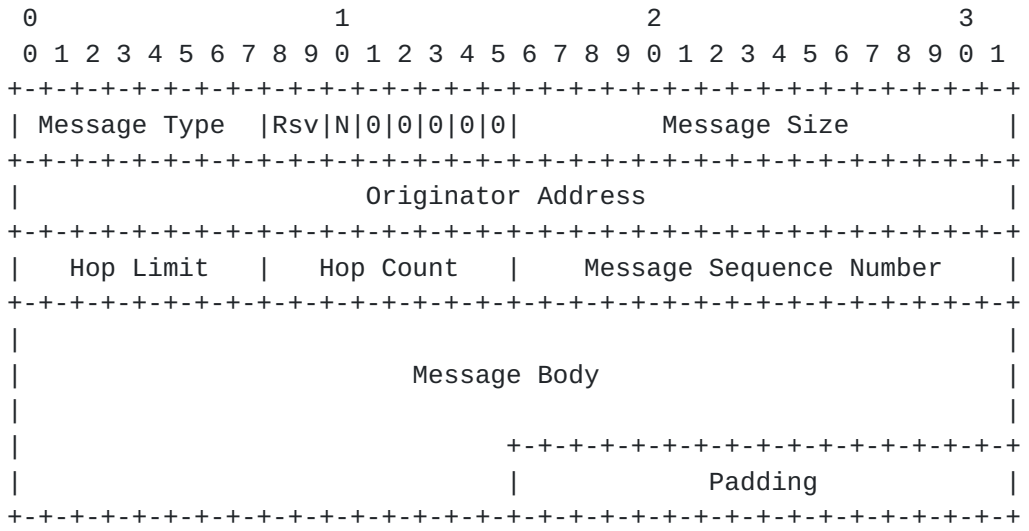

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
|0 0 0 0 0 0 0 0| Resv |1|0|1|0| Packet Size
Packet Sequence Number
Packet TLV Block
Padding
Message + Padding
...
Message + Padding

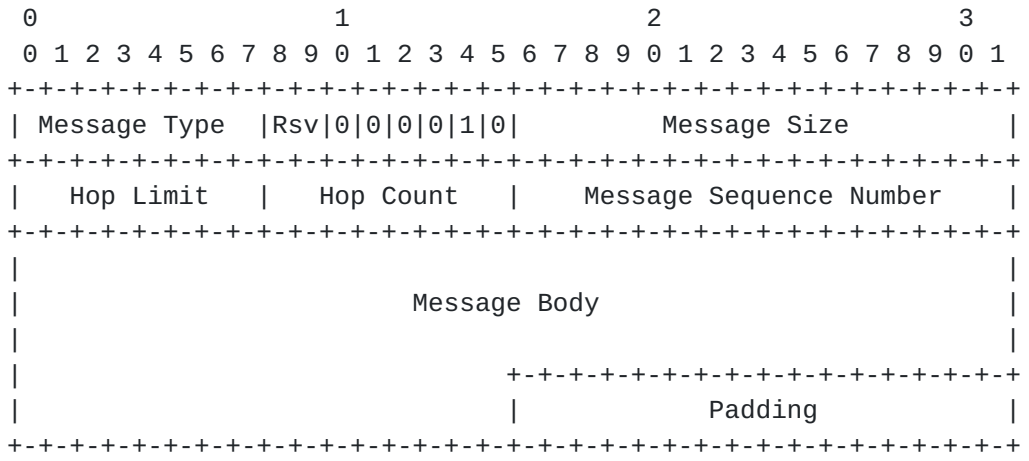

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
0 0 0 0 0 0 0 0 Resv 1 1 0 0
Packet TLV Block
Padding
Message + Padding
...
Message + Padding

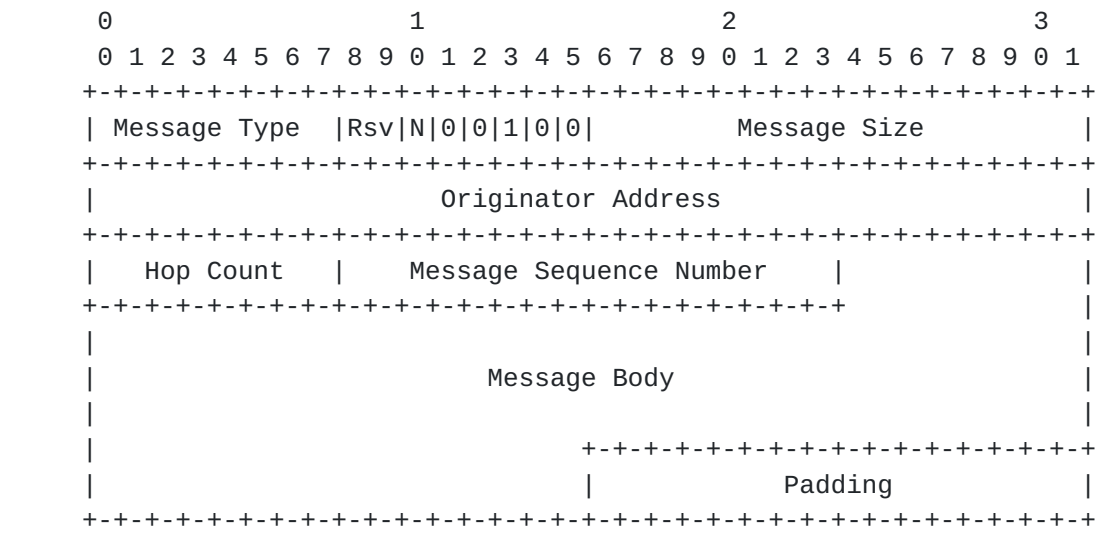

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
0 0 0 0 0 0 0 0 Resv 1 1 1 0 Packet Size
Packet TLV Block
Padding
Message + Padding
...
Message + Padding

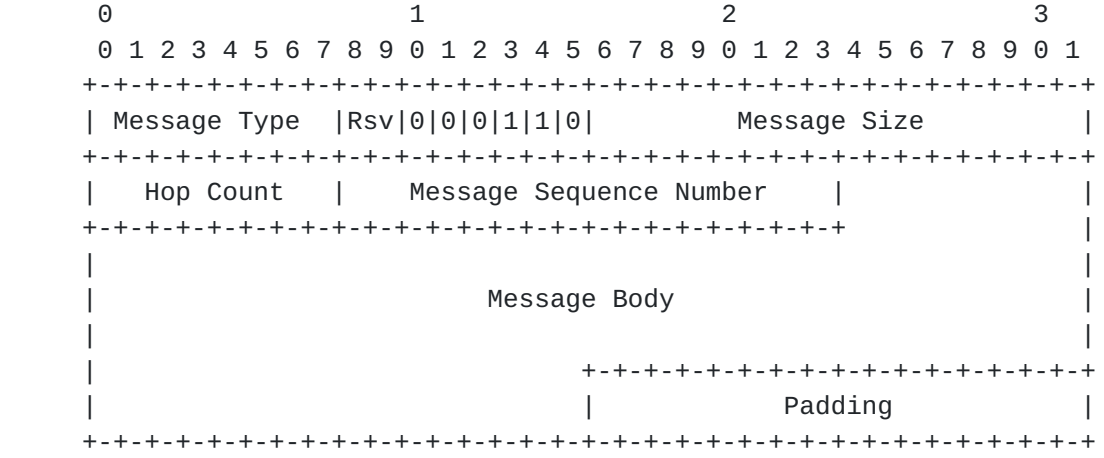


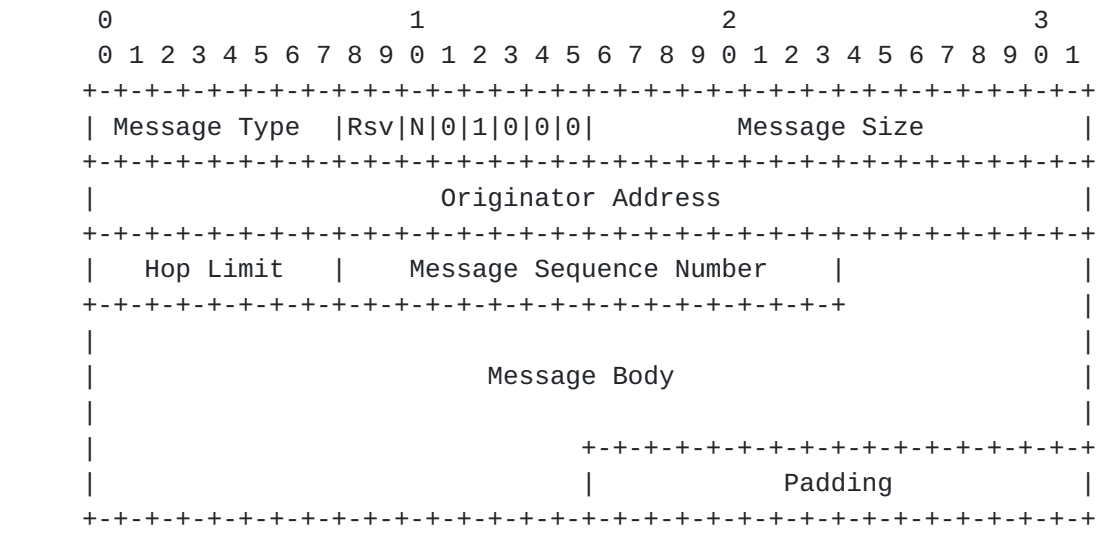
Appendix B.2. Message and Padding

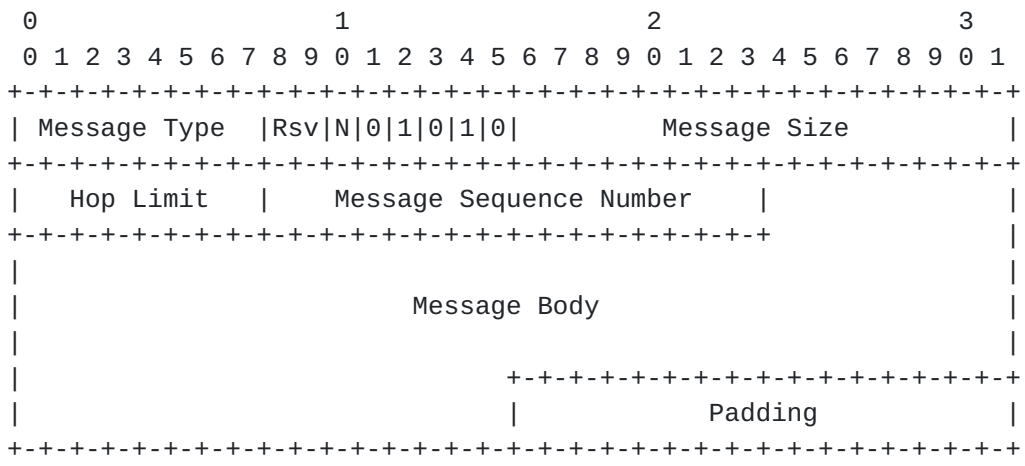


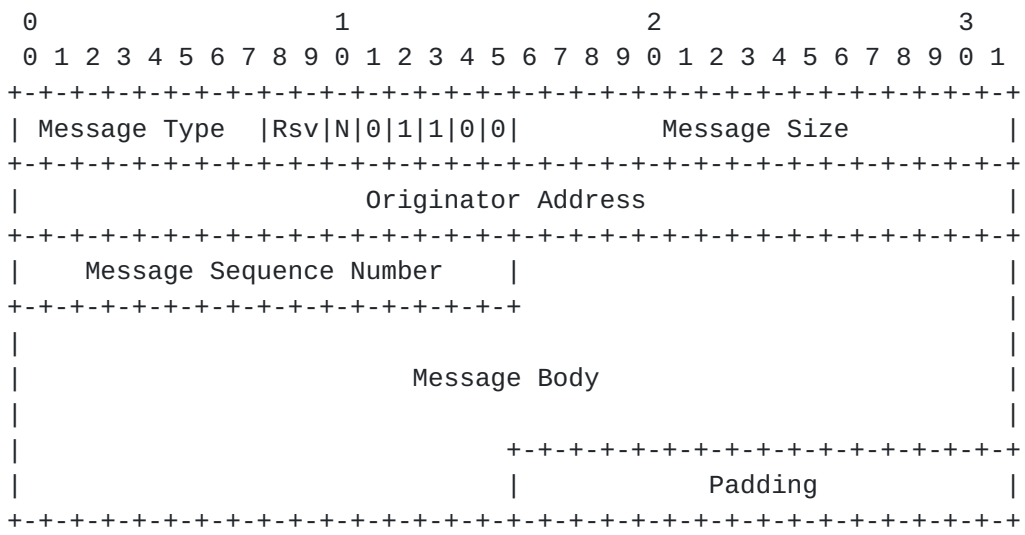


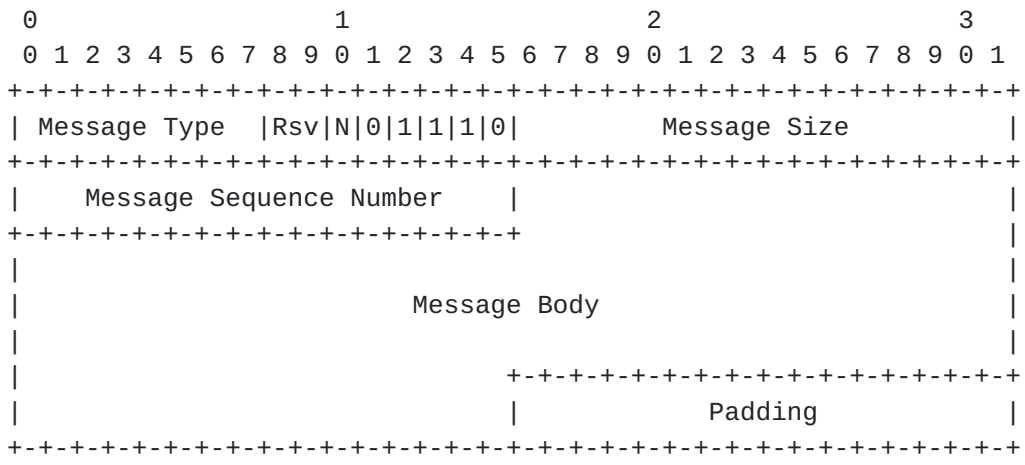

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type
Rsv
N
0 0 1 0 0
Message Size
Originator Address
Hop Count
Message Sequence Number
Message Body
Padding

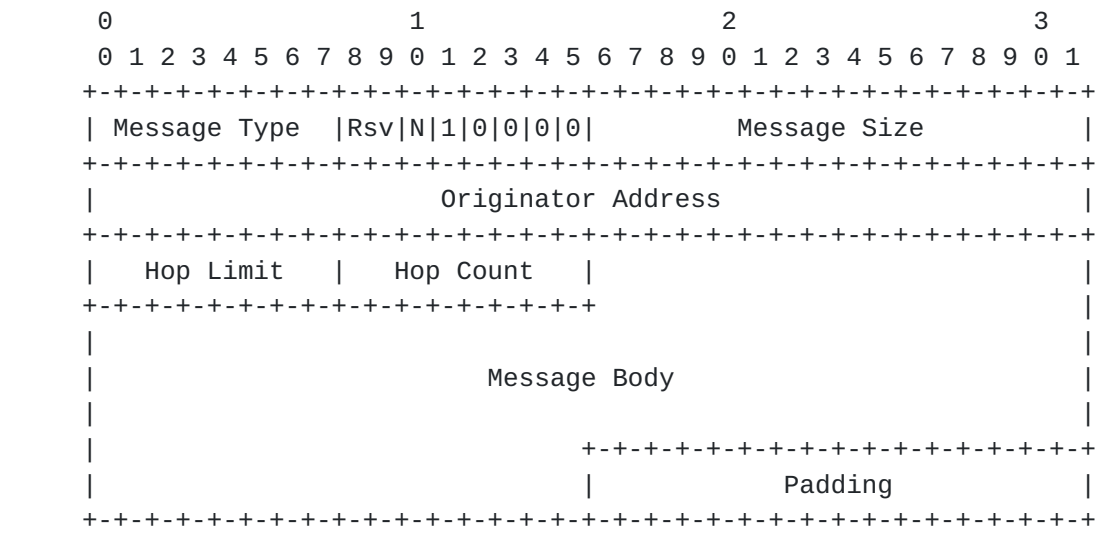

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type
Rsv
N
0 0 0 1 1 0
Message Size
Hop Count
Message Sequence Number
Message Body
Padding

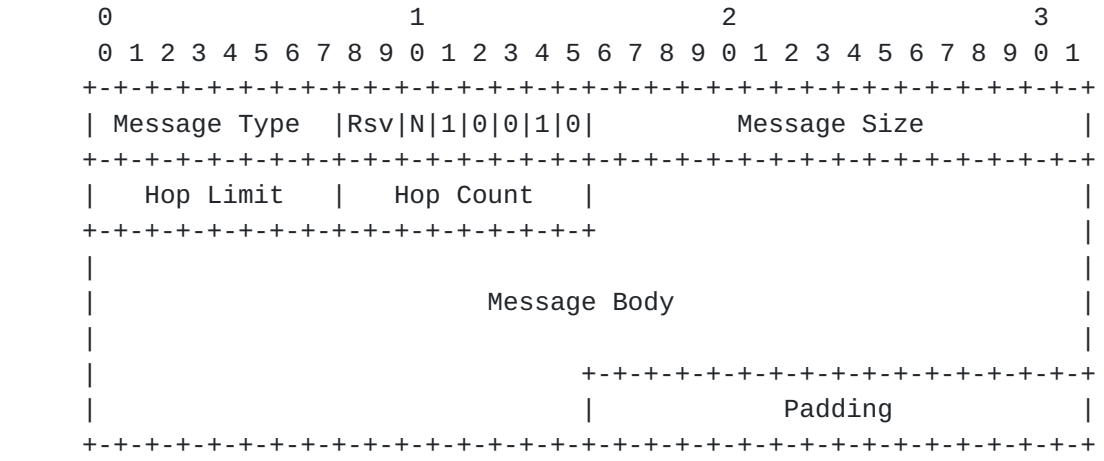

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type
Rsv
N
0 1 0 0 0
Message Size
Originator Address
Hop Limit
Message Sequence Number
Message Body
Padding

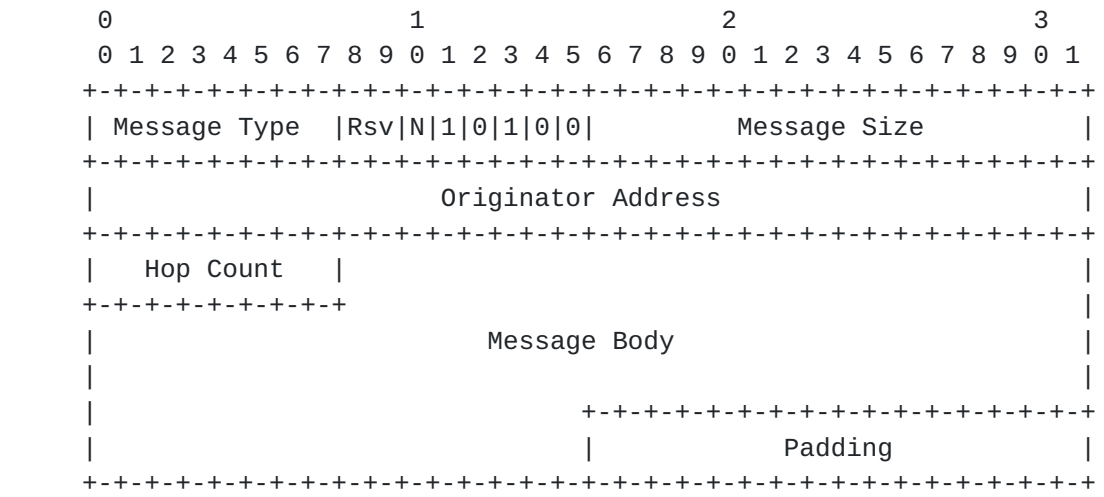

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type
Rsv
N
0
1
0
1
0
Message Size
Hop Limit
Message Sequence Number
Message Body
Padding

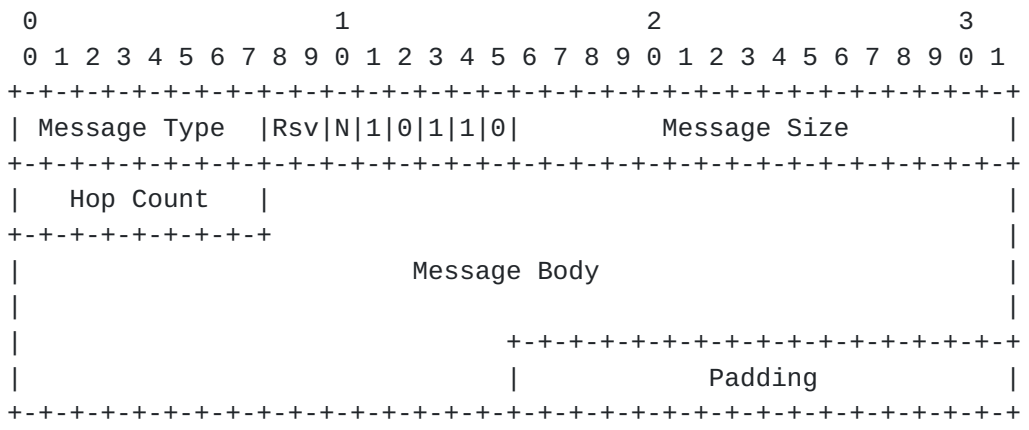

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type
Rsv
N
0
1
1
0
0
Message Size
Originator Address
Message Sequence Number
Message Body
Padding

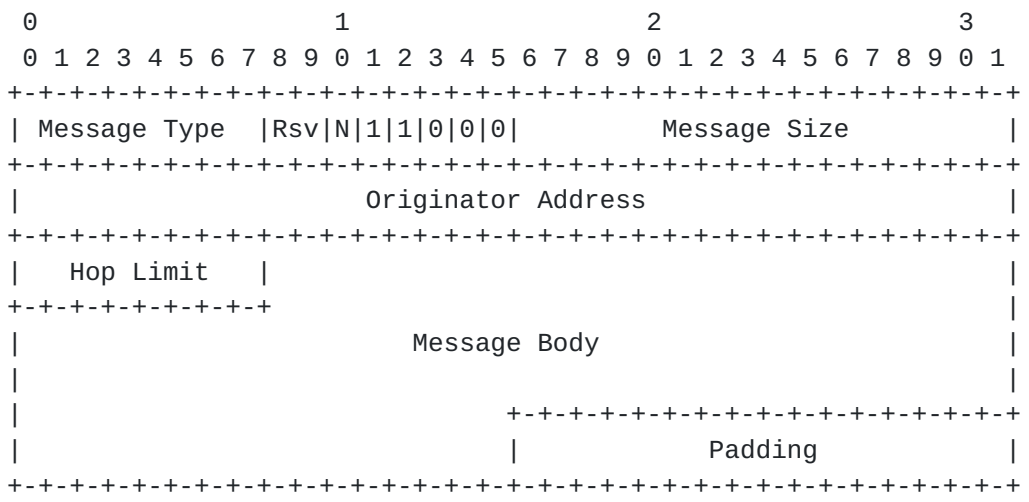

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type
Rsv
N
0
1
1
1
0
Message Size
Message Sequence Number
Message Body
Padding

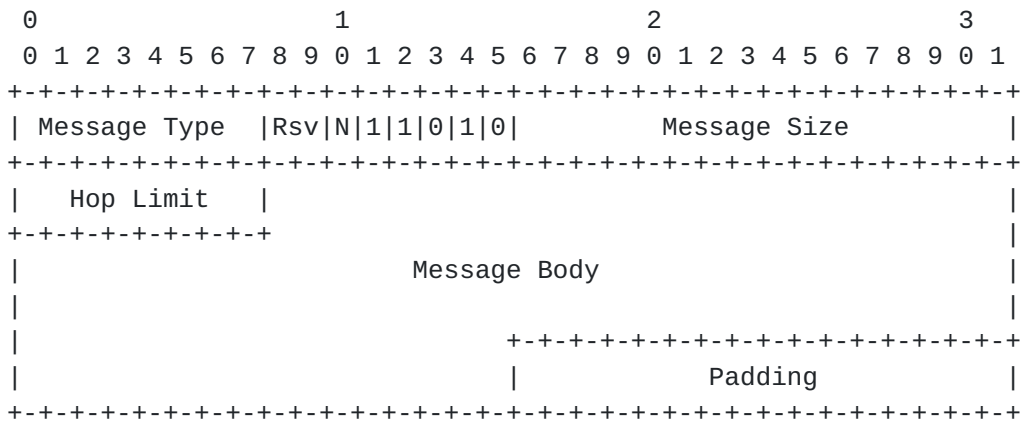

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type
Rsv
N
1
0
0
0
0
Message Size
Originator Address
Hop Limit
Hop Count
Message Body
Padding

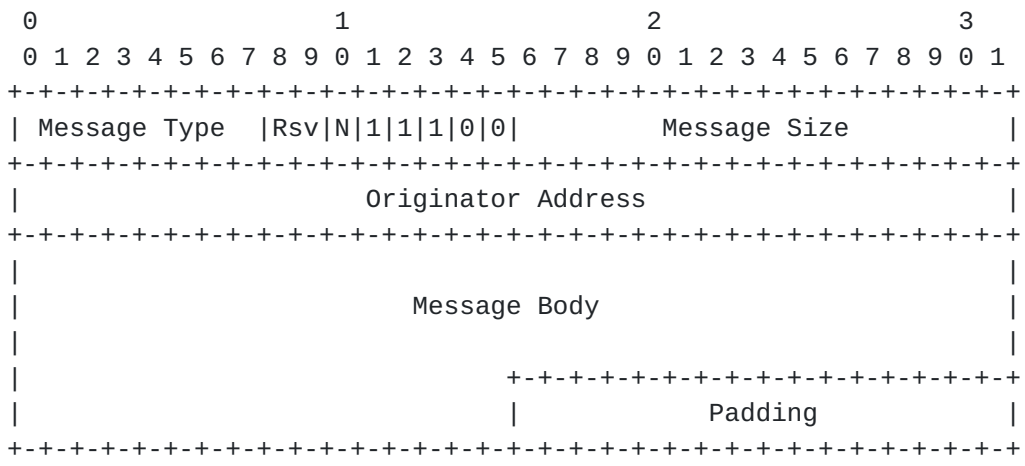

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type
Rsv
N
1
0
0
1
0
Message Size
Hop Limit
Hop Count
Message Body
Padding

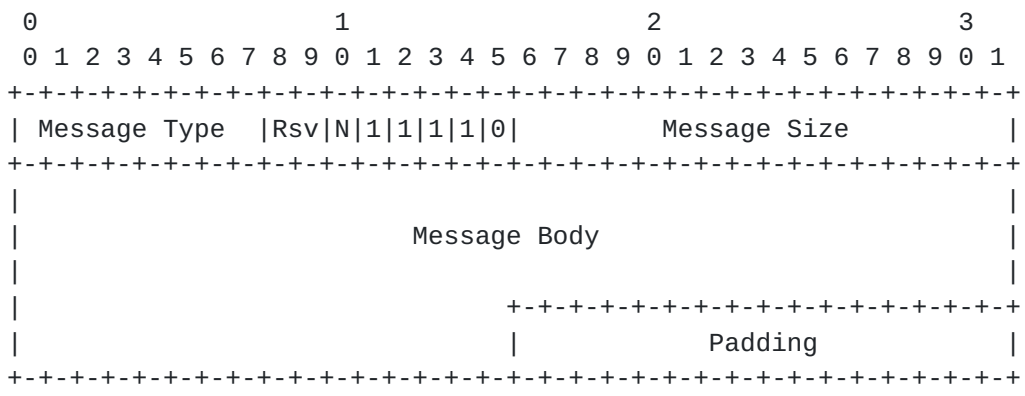

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type
Rsv
N
1
0
1
0
0
Message Size
Originator Address
Hop Count
Message Body
Padding


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
Rsv
N
1
0
1
1
0
Message Size
Hop Count
Message Body
Padding

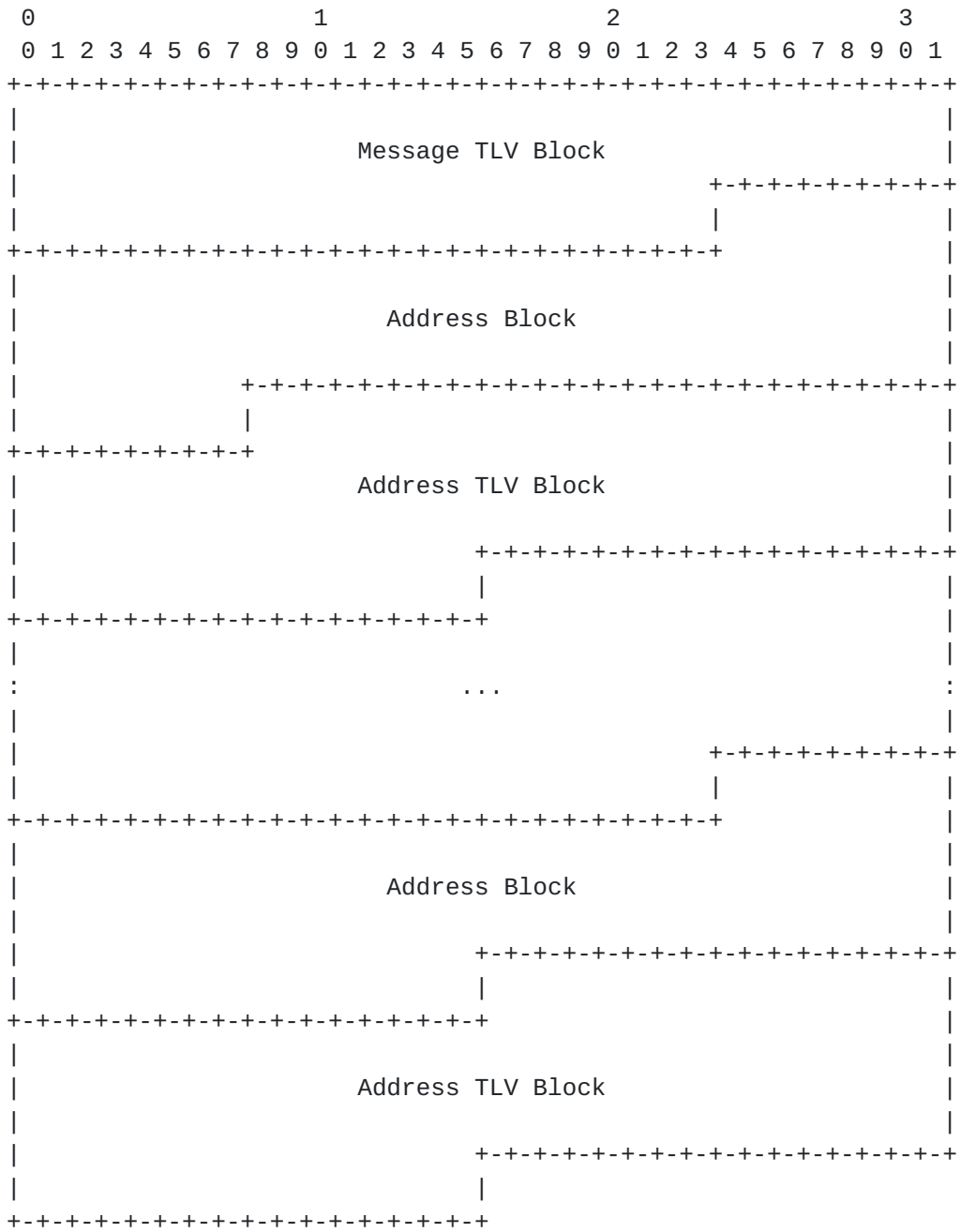

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type
Rsv
N
1
1
0
0
0
Message Size
Originator Address
Hop Limit
Message Body
Padding


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
Rsv
N
1
1
0
1
0
Message Size
Hop Limit
Message Body
Padding

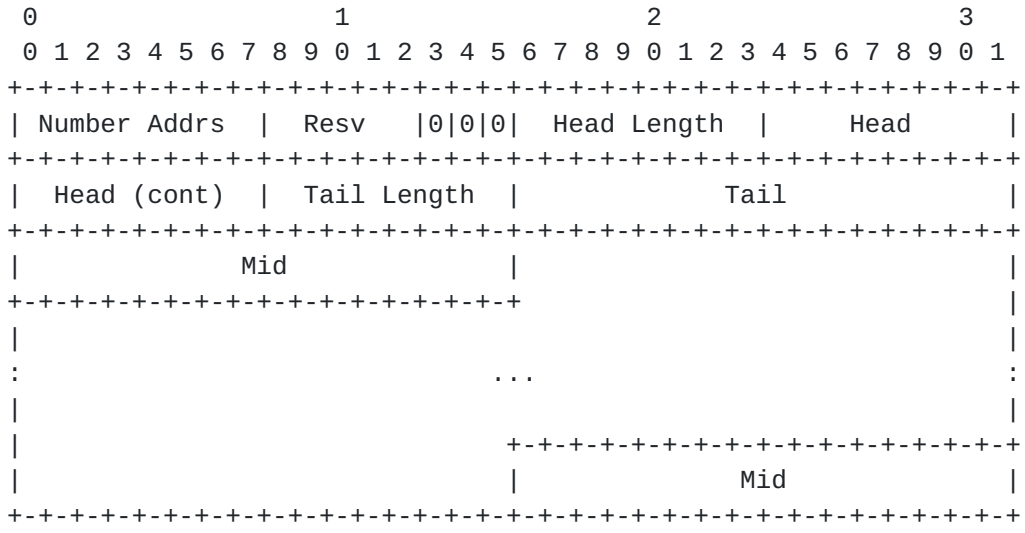

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type
Rsv
N
1
1
1
0
0
Message Size
Originator Address
Message Body
Padding

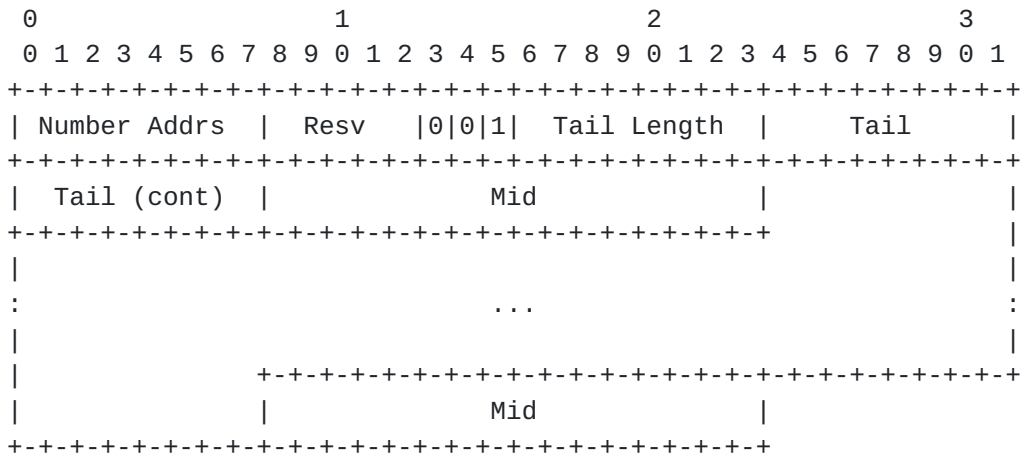

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Message Type
Rsv
N
1
1
1
1
0
Message Size
Message Body
Padding

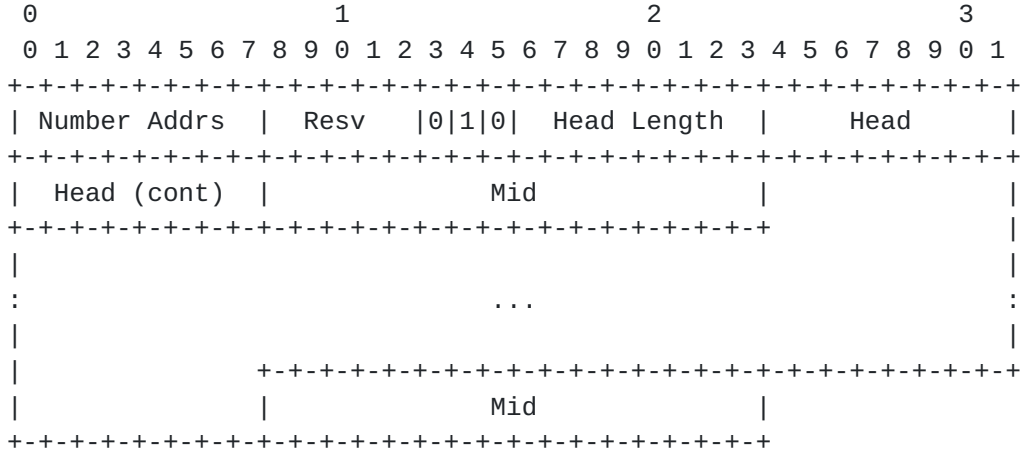
Appendix B.3. Message Body

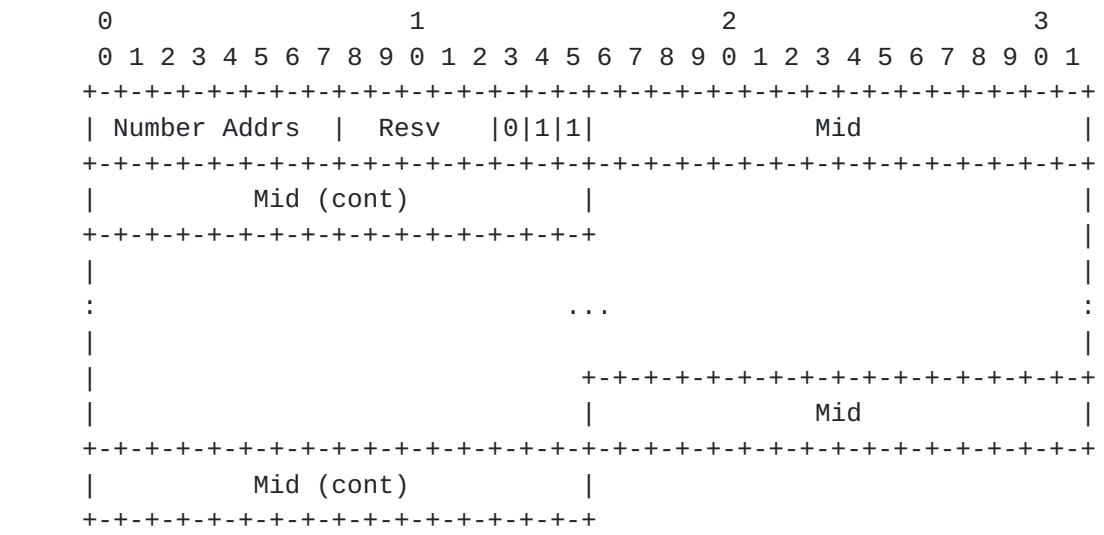


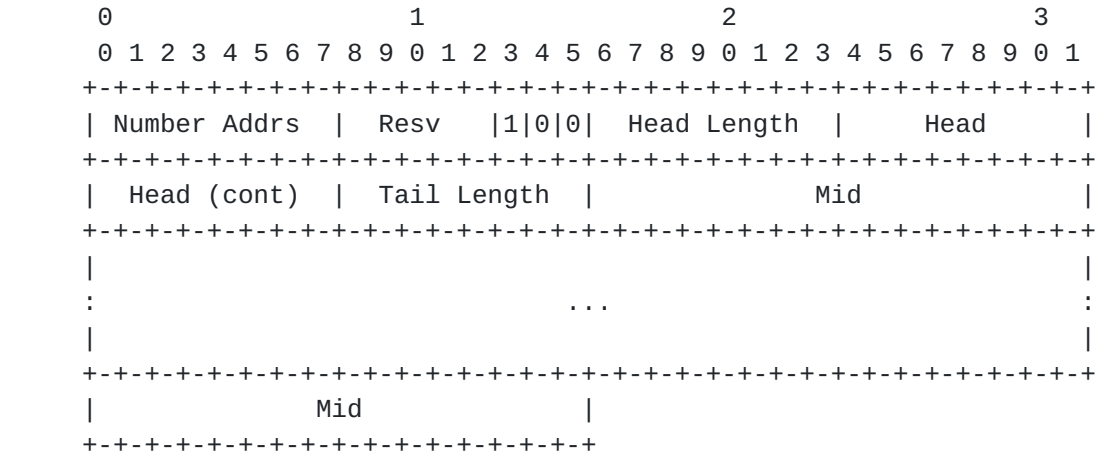
Appendix B.4. Address Block

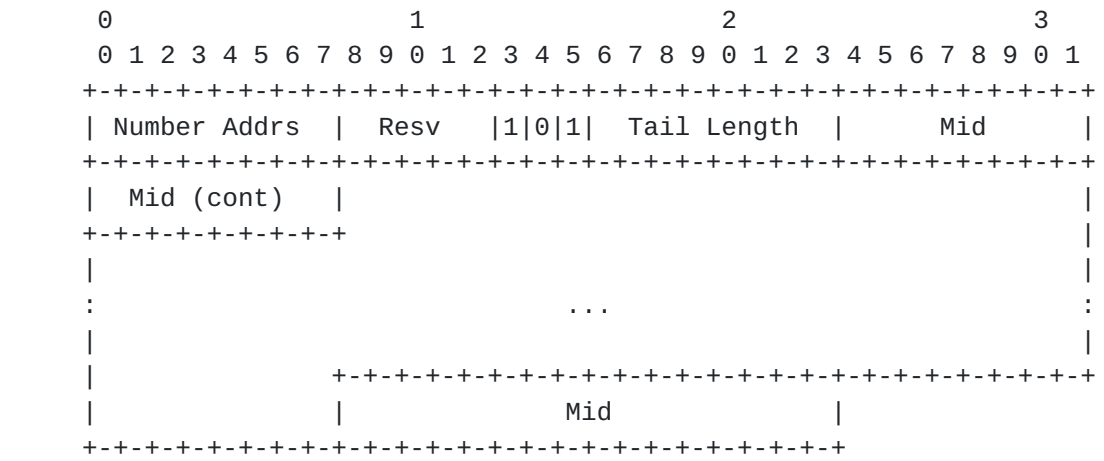




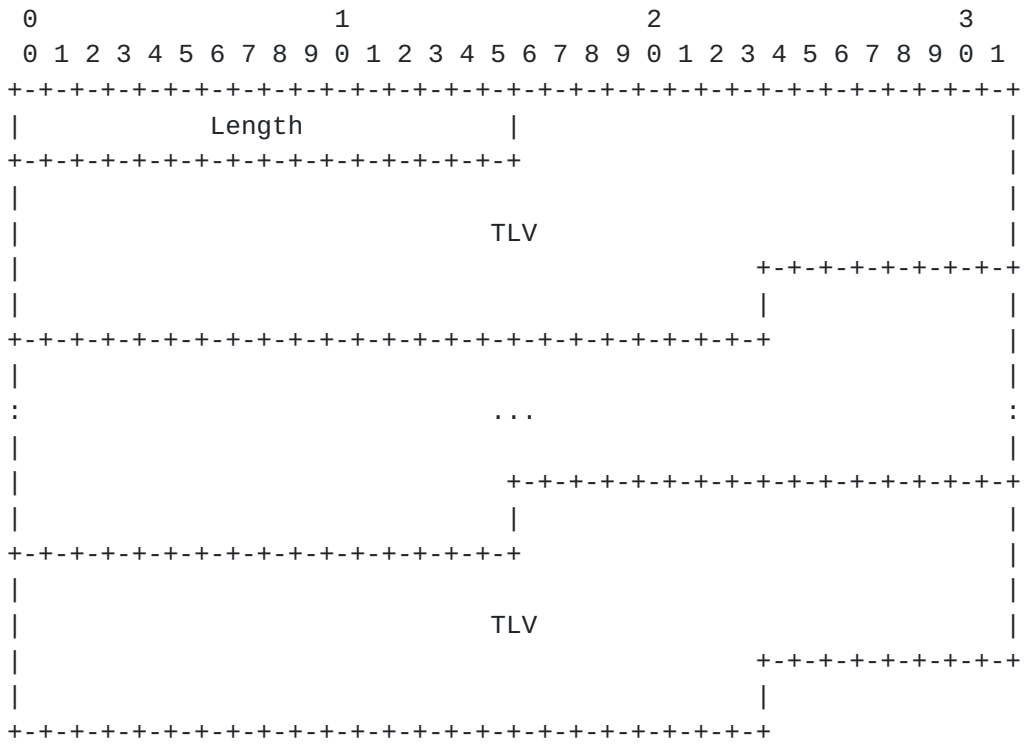



0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
Resv
0|1|1|
Mid
Mid (cont)
...
Mid
Mid (cont)

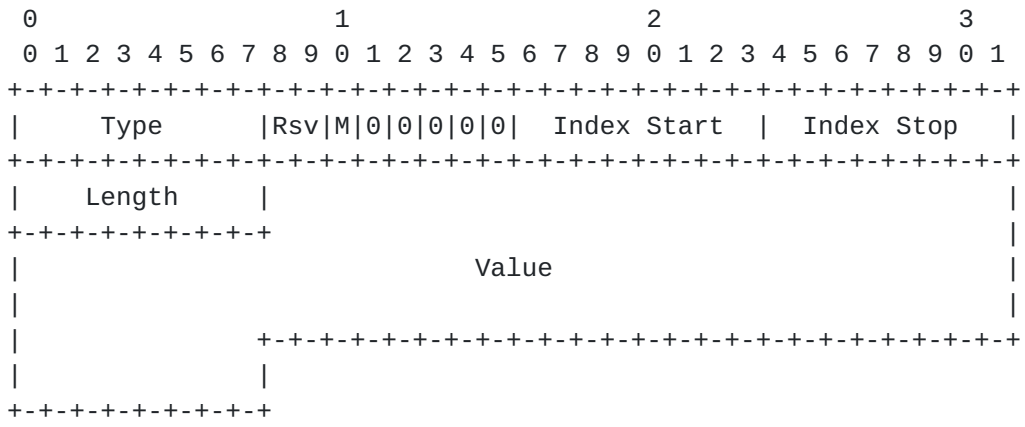

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
Resv
1|0|0|
Head Length
Head
Head (cont)
Tail Length
Mid
...
Mid

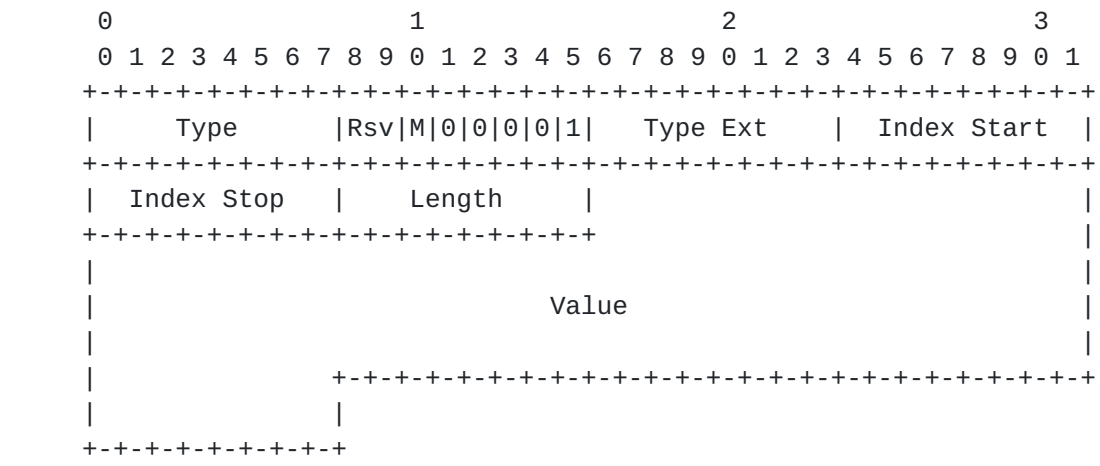

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Number Addrs
Resv
1|0|1|
Tail Length
Mid
Mid (cont)
...
Mid

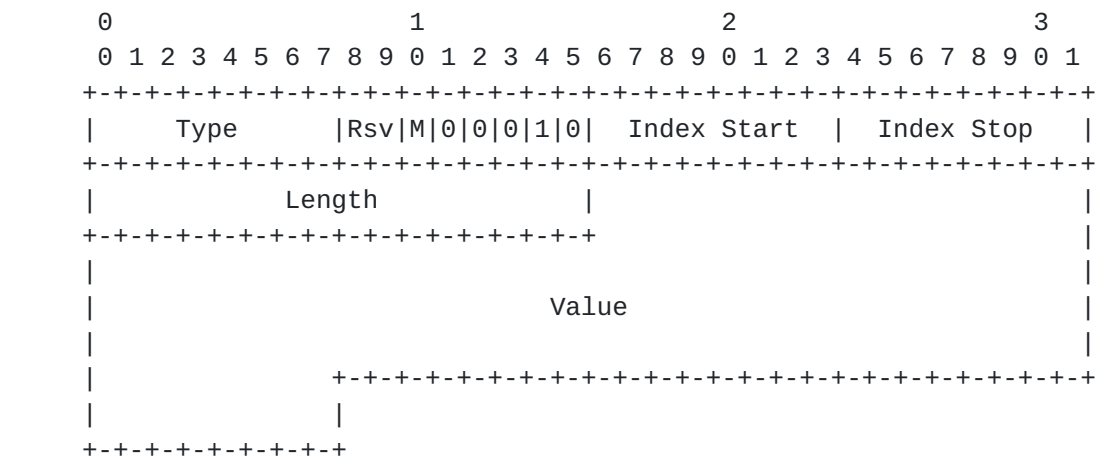
Appendix B.5. TLV Block

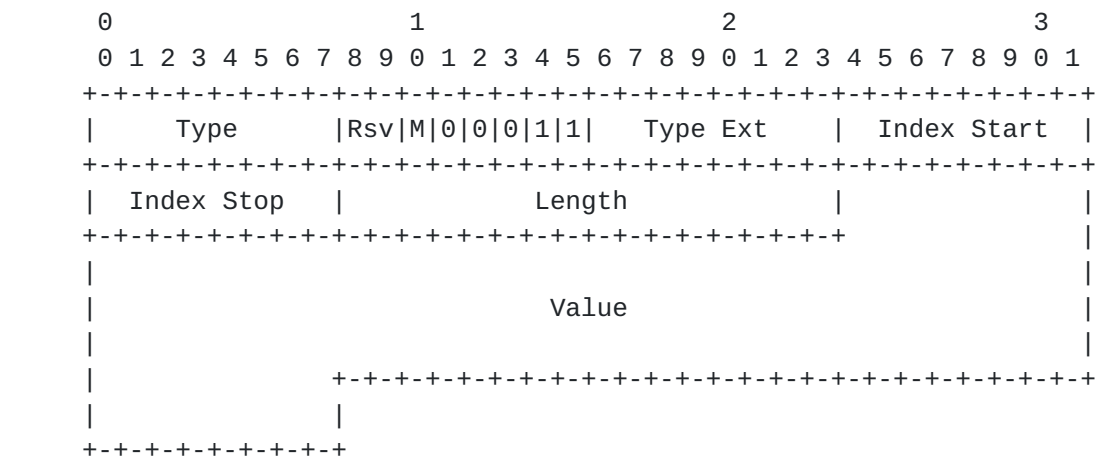


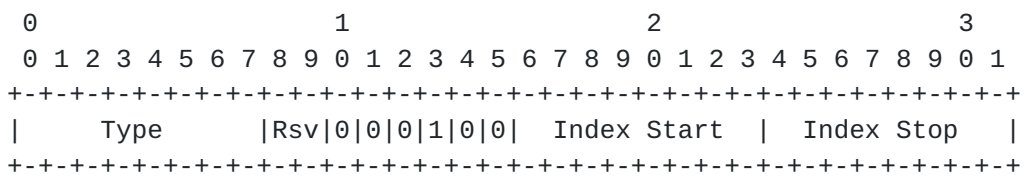
Appendix B.6. TLV

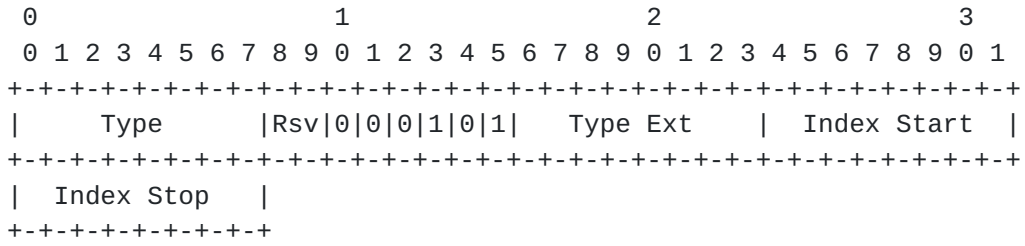


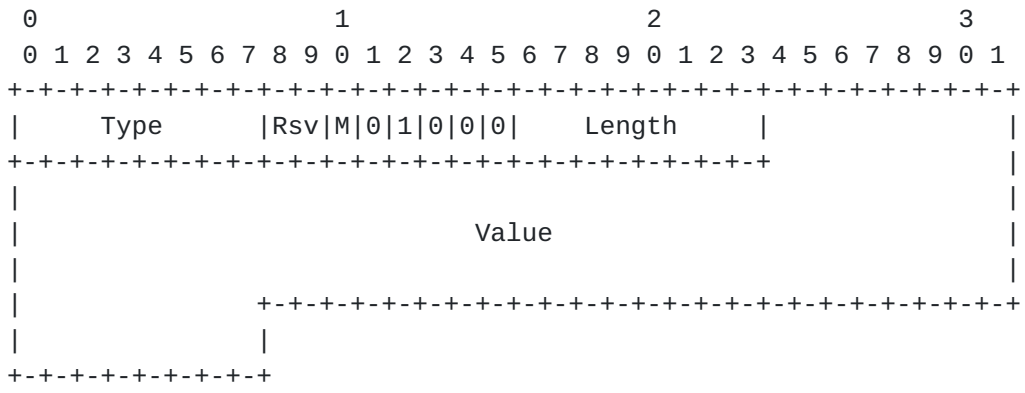

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
Rsv
M
0
0
0
0
1
Type Ext
Index Start
Index Stop
Length
Value

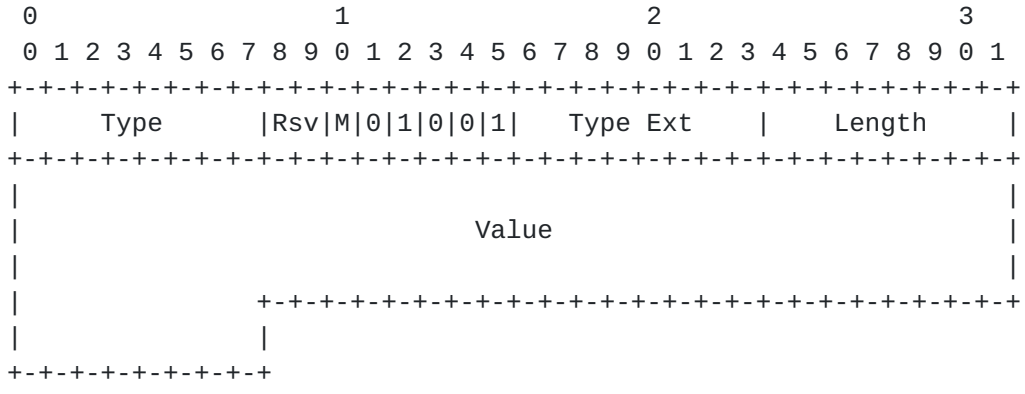

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
Rsv
M
0
0
0
1
0
Index Start
Index Stop
Length
Value

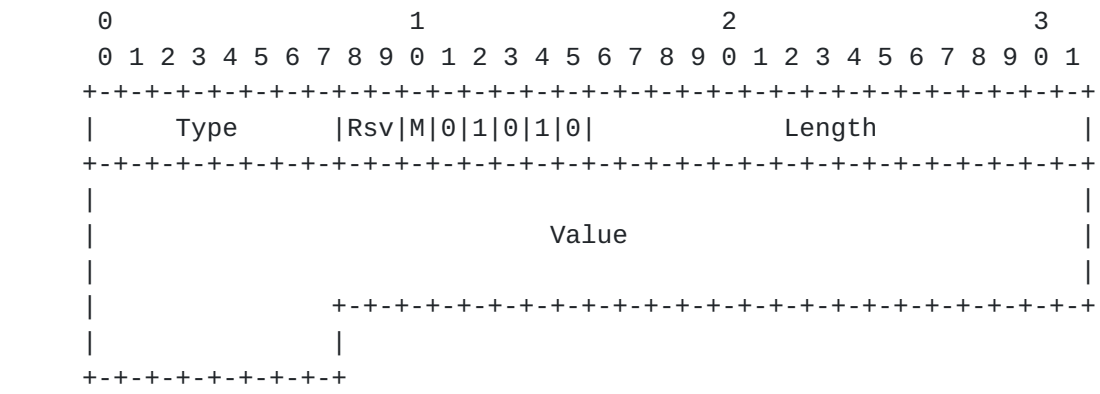

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
Rsv
M
0
0
0
1
1
Type Ext
Index Start
Index Stop
Length
Value

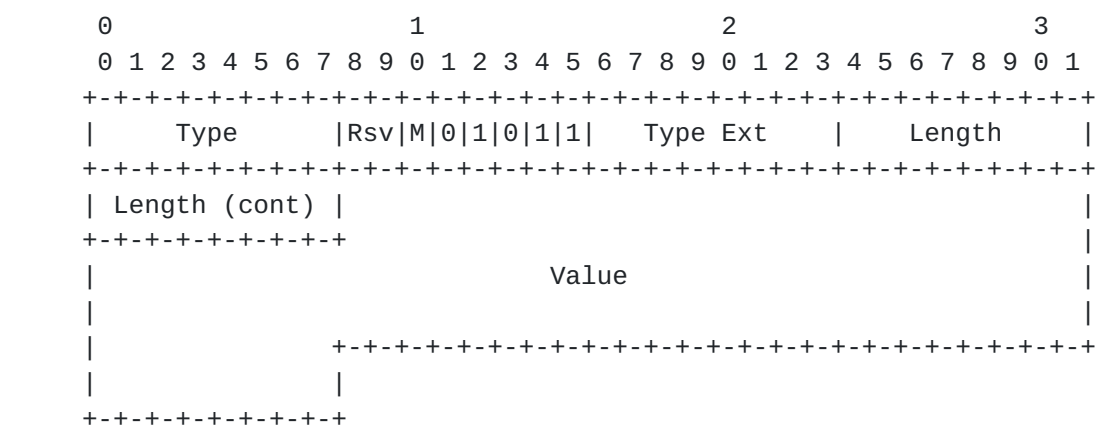

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
Rsv|0|0|0|1|0|0|
Index Start
Index Stop

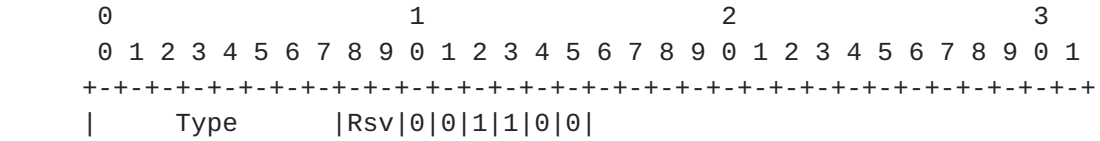

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
Rsv|0|0|0|1|0|1|
Type Ext
Index Start
Index Stop

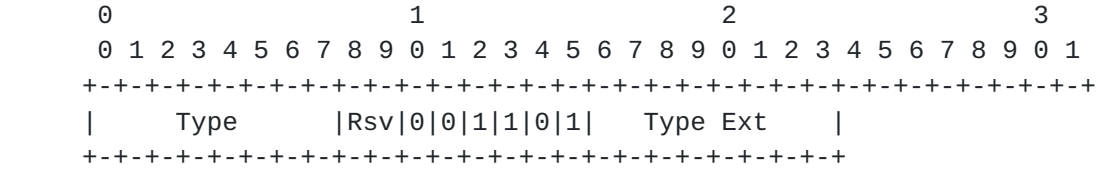

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
Rsv|M|0|1|0|0|0|
Length
Value

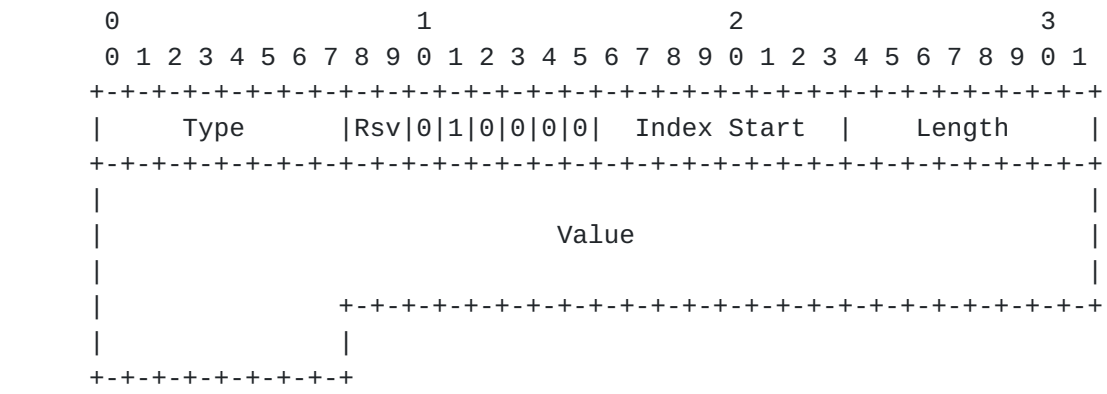

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
Rsv|M|0|1|0|0|1|
Type Ext
Length
Value

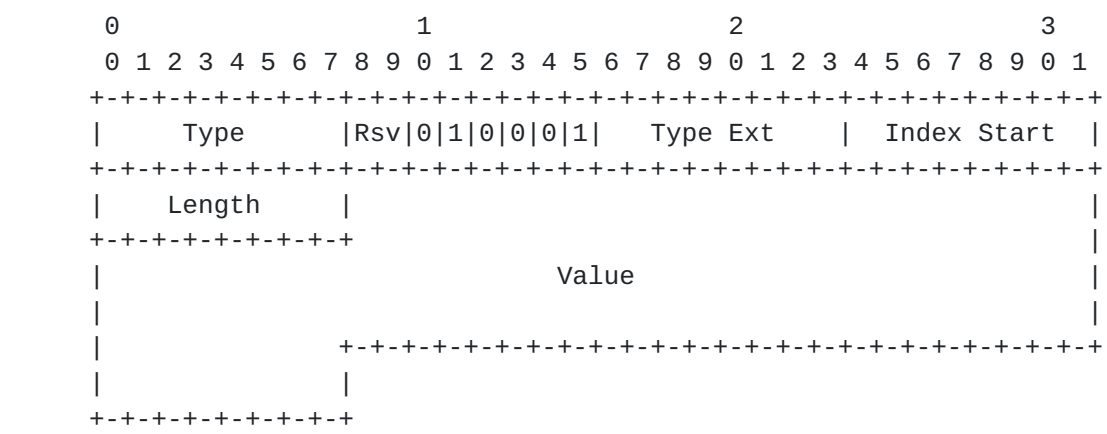

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
Rsv|M|0|1|0|1|0|
Length
Value

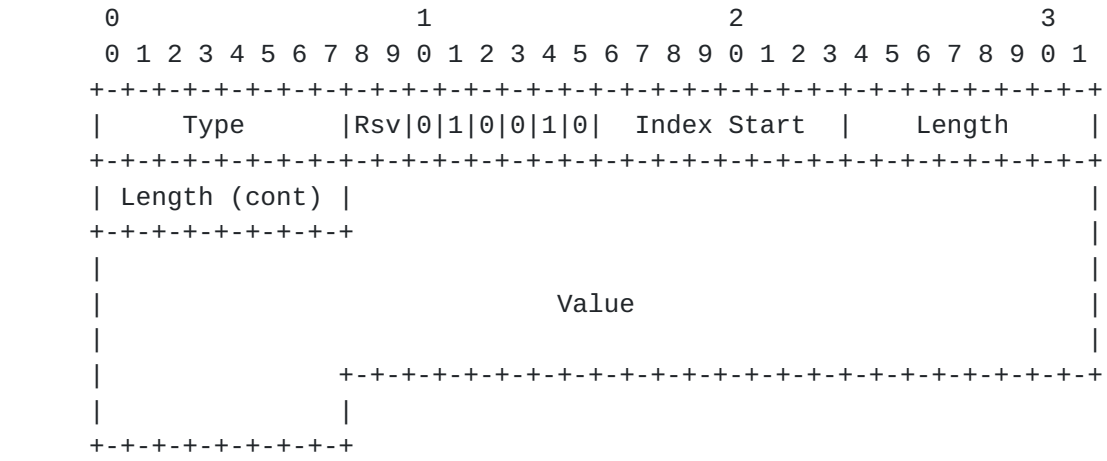

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
Rsv|M|0|1|0|1|1|
Type Ext
Length
Length (cont)
Value

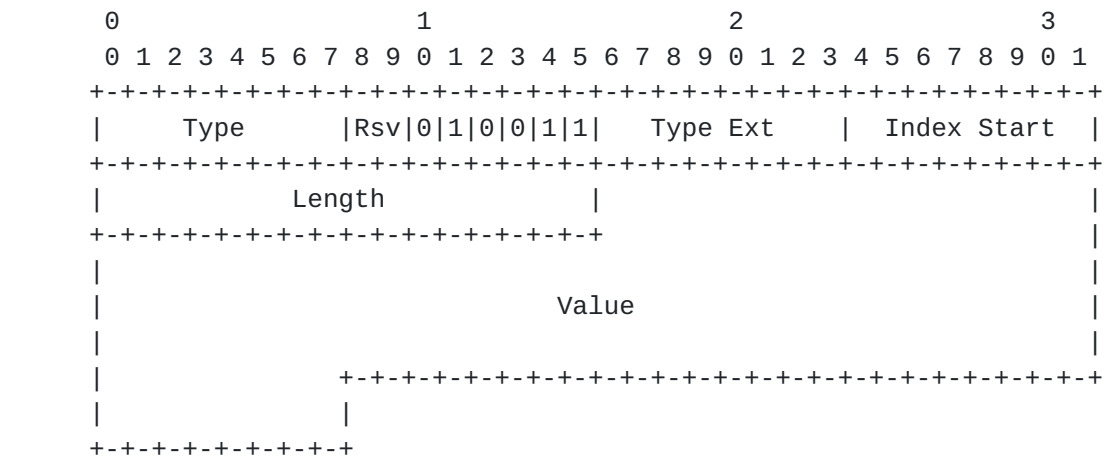

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
Rsv|0|0|1|1|0|0|

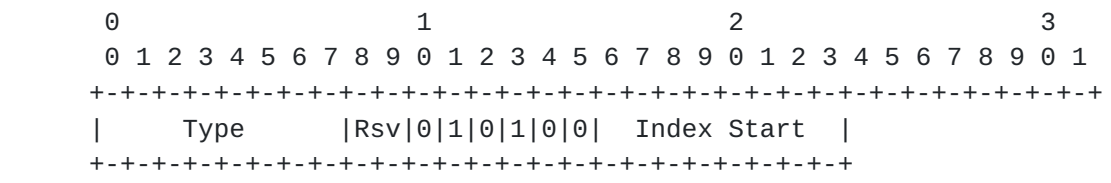

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
Rsv|0|0|1|1|0|1|
Type Ext

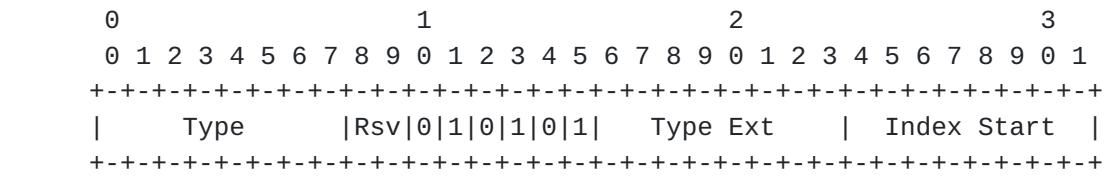

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Type
Rsv|0|1|0|0|0|0|
Index Start
Length
Value


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
Rsv|0|1|0|0|0|1|
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
Rsv|0|1|0|0|1|0|
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
Rsv|0|1|0|0|1|1|
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
Rsv|0|1|0|1|0|0|
Index Start


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
Rsv|0|1|0|1|0|1|
Type Ext
Index Start

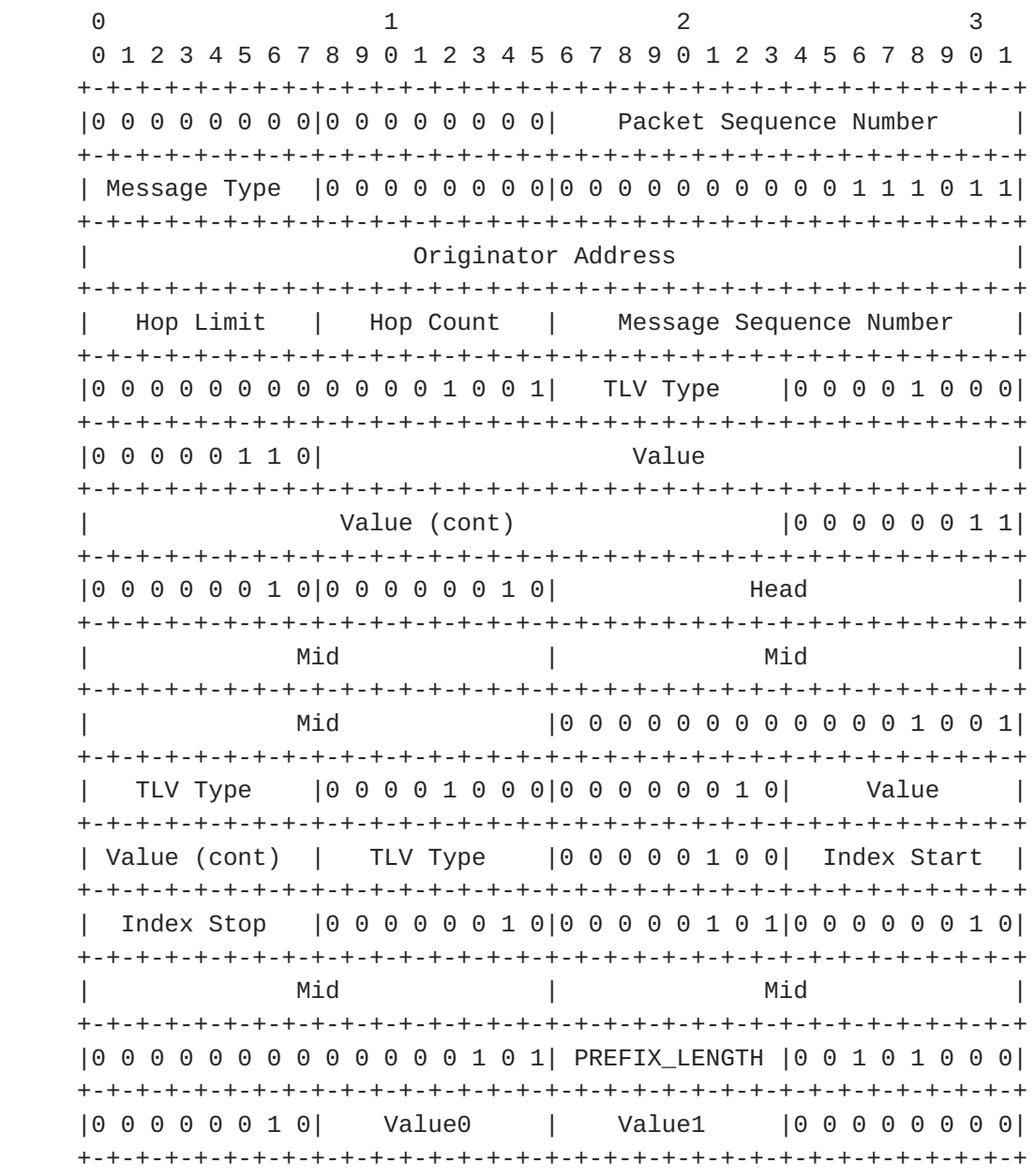
Appendix C. Complete Example

An example packet, using IPv4 addresses (length four octets) is shown. This packet has a header, indicated by the initial octet 0. The packet header has semantics octet 0, and hence has version zero, a packet sequence number, but no packet TLV block.

The packet contains a single message. This message has semantics octet 0, and hence has a complete message header, other than version (which is zero), including originator address, hop limit, hop count and message sequence number (which is type independent). The message has a message TLV block with content length 9 octets, containing a single TLV. This TLV has the tnoindex bit of its semantics octet 8 set, and has value length 6 octets. The message then has two address blocks each with a following TLV block.

The first address block contains 3 addresses. It has the anotail bit of its semantics octet 2 set, and has head length 2 octets, hence mid length two octets. It is followed by a TLV block, with content length 9 octets, containing two TLVs. The first of these TLVs has the tnoindex bit of its semantics octet 8 set, 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 tnovalue bit of its semantics octet 4 set, and hence has no length or value fields. It does have index fields, which indicate those addresses this TLV applies to.

The second address block contains 2 addresses. It has the anohead and ahaszerotail bits of its semantics octet 5 set, and has tail length 2 octets, hence mid length two octets. The two tail octets of each address are zero. It is followed by a TLV block with content length 5 octets. This TLV block contains a single TLV of type PREFIX_LENGTH that has the tismultivalue and tnoindex bits of its semantics octet 40 set, and a value field length of 2 octets, indicating two values each of one octet length. There is one final padding octet 0 that is not included in the message length of 59 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
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 Packet Sequence Number
Message Type 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1
Originator Address
Hop Limit Hop Count Message Sequence Number
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 TLV Type 0 0 0 0 1 0 0 0
0 0 0 0 0 1 1 0 Value
Value (cont) 0 0 0 0 0 0 1 1
0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 Head
Mid Mid
Mid 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1
TLV Type 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 Value
Value (cont) TLV Type 0 0 0 0 0 1 0 0 Index Start
Index Stop 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0
Mid Mid
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 PREFIX_LENGTH 0 0 1 0 1 0 0 0
0 0 0 0 0 0 1 0 Value0 Value1 0 0 0 0 0 0 0 0

Appendix D. 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, <h-satoh.yj@sdl.hitachi.co.jp>
- Philippe Jacquet, INRIA, France, <Philippe.Jacquet@inria.fr>
- Monden Kazuya, Hitachi SDL, Japan, <monden.vw@sdl.hitachi.co.jp>

Appendix E. Acknowledgements

The authors would like to acknowledge the team behind OLSRv1, as specified in RFC 3626, including Anis Laouiti, Pascale Minet, Laurent Viennot (all 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: Joe Macker (NRL), Alan Cullen (BAE Systems), Ian Chakeres (Motorola), Charlie E. Perkins (Nokia), Andreas Schjonhaug (LIX), Florent Brunneau (LIX), Yasunori Owada (Niigata University) 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

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).