lpwan Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 7, 2020

A. Minaburo
Acklio
L. Toutain
IMT-Atlantique
C. Gomez
Universitat Politecnica de Catalunya
D. Barthel
Orange Labs
JC. Zuniga
SIGFOX

December 05, 2019

Static Context Header Compression (SCHC) and fragmentation for LPWAN, application to UDP/IPv6
draft-ietf-lpwan-ipv6-static-context-hc-24

Abstract

This document defines the Static Context Header Compression (SCHC) framework, which provides both a header compression mechanism and an optional fragmentation mechanism. SCHC has been designed for Low Power Wide Area Networks (LPWAN).

SCHC compression is based on a common static context stored both in the LPWAN device and in the network infrastructure side. This document defines a generic header compression mechanism and its application to compress IPv6/UDP headers.

This document also specifies an optional fragmentation and reassembly mechanism. It can be used to support the IPv6 MTU requirement over the LPWAN technologies. Fragmentation is needed for IPv6 datagrams that, after SCHC compression or when such compression was not possible, still exceed the layer-2 maximum payload size.

The SCHC header compression and fragmentation mechanisms are independent of the specific LPWAN technology over which they are used. This document defines generic functionalities and offers flexibility with regard to parameter settings and mechanism choices. This document standardizes the exchange over the LPWAN between two SCHC entities. Settings and choices specific to a technology or a product are expected to be grouped into profiles, which are specified in other documents. Data models for the context and profiles are out of scope.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines the Static Context Header Compression (SCHC) framework, which provides both a header compression mechanism and an optional fragmentation mechanism. SCHC has been designed for Low Power Wide Area Networks (LPWAN).

LPWAN technologies impose some strict limitations on traffic. For instance, devices sleep most of the time and may only receive data during short periods of time after transmission, in order to preserve battery. LPWAN technologies are also characterized by a greatly reduced data unit and/or payload size (see [RFC8376]).

Header compression is needed for efficient Internet connectivity to a node within an LPWAN network. The following properties of LPWAN networks can be exploited to get an efficient header compression:

- The network topology is star-oriented, which means that all packets between the same source-destination pair follow the same path. For the needs of this document, the architecture can simply be described as Devices (Dev) exchanging information with LPWAN Application Servers (App) through a Network Gateway (NGW).

- Because devices embed built-in applications, the traffic flows to be compressed are known in advance. Indeed, new applications are less frequently installed in an LPWAN device, than they are in a general-purpose computer or smartphone.

SCHC compression uses a Context (a set of Rules) in which information about header fields is stored. This Context is static: the values of the header fields and the actions to do compression/decompression do not change over time. This avoids the need for complex resynchronization mechanisms. Indeed, a return path may be more restricted/expensive, sometimes completely unavailable [RFC8376]. A compression protocol that relies on feedback is not compatible with the characteristics of such LPWANs.

In most cases, a small Rule identifier is enough to represent the full IPv6/UDP headers. The SCHC header compression mechanism is independent of the specific LPWAN technology over which it is used.

Furthermore, some LPWAN technologies do not provide a fragmentation functionality; to support the IPv6 MTU requirement of 1280 bytes [RFC8200], they require a fragmentation protocol at the adaptation layer below IPv6. Accordingly, this document defines an optional fragmentation/reassembly mechanism for LPWAN technologies to support the IPv6 MTU requirement.

This document defines generic functionality and offers flexibility with regard to parameters settings and mechanism choices. Technology-specific settings are expected to be grouped into Profiles specified in other documents.

2. Requirements Notation

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. LPWAN Architecture

LPWAN network architectures are similar among them, but each LPWAN technology names architecture elements differently. In this document, we use terminology from [RFC8376], which identifies the following entities in a typical LPWAN network (see Figure 1):

o Devices (Dev) are the end-devices or hosts (e.g., sensors, actuators, etc.). There can be a very high density of devices per radio gateway.

o The Radio Gateway (RGW) is the end point of the constrained link.

o The Network Gateway (NGW) is the interconnection node between the Radio Gateway and the Internet.

o Application Server (App) is the end point of the application level protocol on the Internet side.

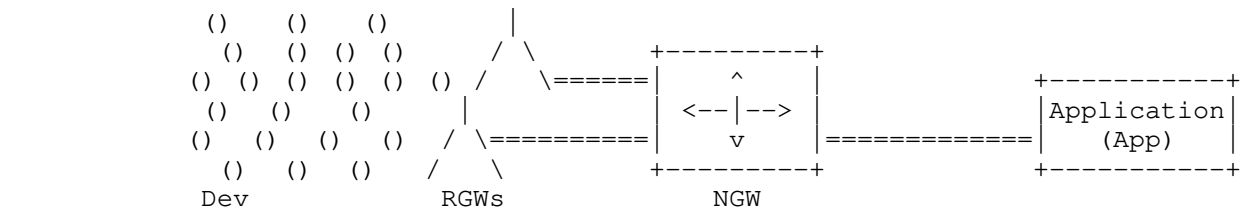


Figure 1: LPWAN Architecture, simplified from that shown in RFC8376

4. Terminology

This section defines the terminology and acronyms used in this document. It extends the terminology of [RFC8376].

The SCHC acronym is pronounced like "sheek" in English (or "chic" in French). Therefore, this document writes "a SCHC Packet" instead of "an SCHC Packet".

- App: LPWAN Application, as defined by [RFC8376]. An application sending/receiving packets to/from the Dev.

- AppIID: Application Interface Identifier. The IID that identifies the application server interface.

- Bi: Bidirectional. Characterizes a Field Descriptor that applies to headers of packets traveling in either direction (Up and Dw, see this glossary).

- CDA: Compression/Decompression Action. Describes the pair of actions that are performed at the compressor to compress a header field and at the decompressor to recover the original value of the header field.

- Compression Residue. The bits that remain to be sent (beyond the Rule ID itself) after applying the SCHC compression.

- Context: A set of Rules used to compress/decompress headers.

- Dev: Device, as defined by [RFC8376].

- DevIID: Device Interface Identifier. The IID that identifies the Dev interface.

- DI: Direction Indicator. This field tells which direction of packet travel (Up, Dw or Bi) a Field Description applies to. This allows for asymmetric processing, using the same Rule.

- Dw: Downlink direction for compression/decompression, from SCHC C/D in the network to SCHC C/D in the Dev.

- Field Description. A tuple containing identifier, value, matching operator and actions to be applied to a field.

- FID: Field Identifier. This identifies the protocol and field a Field Description applies to.

- FL: Field Length is the length of the original packet header field. It is expressed as a number of bits for header fields of fixed lengths or as a type (e.g., variable, token length, ...) for field lengths that are unknown at the time of Rule creation. The length of a header field is defined in the corresponding protocol specification (such as IPv6 or UDP).

- FP: when a Field is expected to appear multiple times in a header, Field Position specifies the occurrence this Field Description applies to (for example, first uri-path option, second uri-path, etc. in a CoAP header), counting from 1. The value 0 is special and means "don't care", see Section 7.3.

- IID: Interface Identifier. See the IPv6 addressing architecture [RFC7136].

- L2: Layer two. The immediate lower layer SCHC interfaces with. It is provided by an underlying LPWAN technology. It does not necessarily correspond to the OSI model definition of Layer 2.

- L2 Word: this is the minimum subdivision of payload data that the L2 will carry. In most L2 technologies, the L2 Word is an octet. In bit-oriented radio technologies, the L2 Word might be a single bit. The L2 Word size is assumed to be constant over time for each device.

- MO: Matching Operator. An operator used to match a value contained in a header field with a value contained in a Rule.

- Padding (P). Extra bits that may be appended by SCHC to a data unit that it passes to the underlying Layer 2 for transmission. SCHC itself operates on bits, not bytes, and does not have any alignment prerequisite. See Section 9.

- Profile: SCHC offers variations in the way it is operated, with a number of parameters listed in Appendix D. A Profile indicates a particular setting of all these parameters. Both ends of a SCHC communication must be provisioned with the same Profile information and with the same set of Rules before the communication starts, so that there is no ambiguity in how they expect to communicate.

- Rule: A set of Field Descriptions.

- Rule ID (Rule Identifier): An identifier for a Rule. SCHC C/D on both sides share the same Rule ID for a given packet. A set of Rule IDs are used to support SCHC F/R functionality.

- SCHC C/D: SCHC Compressor/Decompressor. A mechanism used on both sides, at the Dev and at the network, to achieve Compression/ Decompression of headers.

- SCHC F/R: SCHC Fragmentation / Reassembly. A mechanism used on both sides, at the Dev and at the network, to achieve Fragmentation / Reassembly of SCHC Packets.

- SCHC Packet: A packet (e.g., an IPv6 packet) whose header has been compressed as per the header compression mechanism defined in this document. If the header compression process is unable to actually compress the packet header, the packet with the uncompressed header is still called a SCHC Packet (in this case, a Rule ID is used to indicate that the packet header has not been compressed). See Section 7 for more details.

- TV: Target value. A value contained in a Rule that will be matched with the value of a header field.

- Up: Uplink direction for compression/decompression, from the Dev SCHC C/D to the network SCHC C/D.

Additional terminology for the optional SCHC Fragmentation / Reassembly mechanism (SCHC F/R) is found in Section 8.2.

5. SCHC overview

SCHC can be characterized as an adaptation layer between an upper layer (typically, IPv6) and an underlying layer (typically, an LPWAN technology). SCHC comprises two sublayers (i.e. the Compression sublayer and the Fragmentation sublayer), as shown in Figure 2.

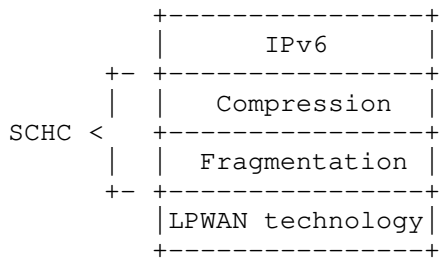
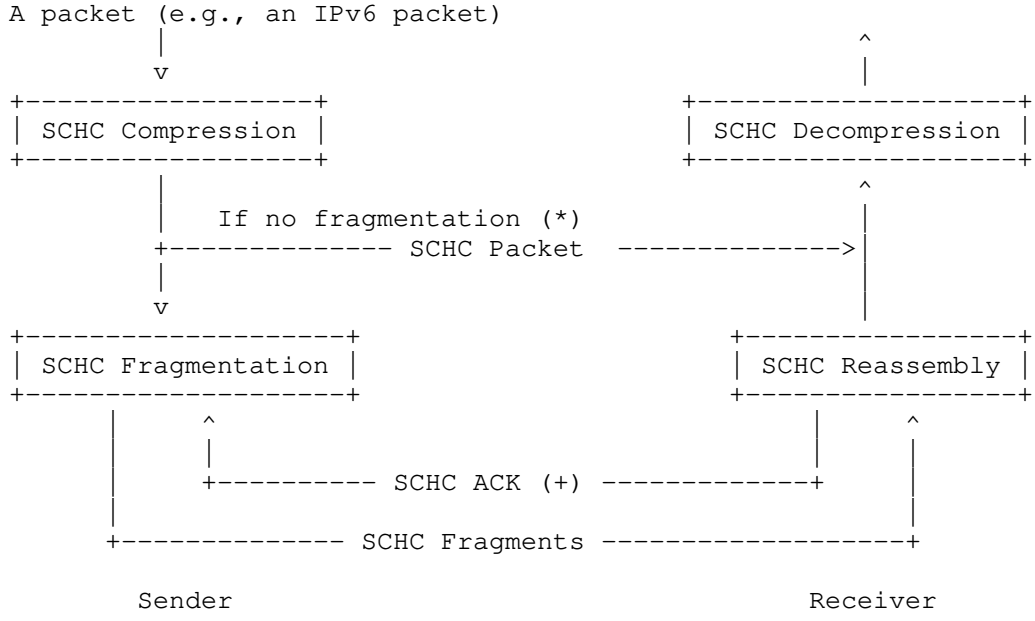


Figure 2: Protocol stack comprising IPv6, SCHC and an LPWAN technology

Before an upper layer packet (e.g., an IPv6 packet) is transmitted to the underlying layer, header compression is first attempted. The resulting packet is called a SCHC Packet, whether or not any compression is performed. If needed by the underlying layer, the optional SCHC Fragmentation MAY be applied to the SCHC Packet. The inverse operations take place at the receiver. This process is illustrated in Figure 3.



*: the decision to not use SCHC Fragmentation is left to each Profile.
+: optional, depends on Fragmentation mode.

Figure 3: SCHC operations at the Sender and the Receiver

5.1. SCHC Packet format

The SCHC Packet is composed of the Compressed Header followed by the payload from the original packet (see Figure 4). The Compressed Header itself is composed of the Rule ID and a Compression Residue, which is the output of compressing the packet header with that Rule (see Section 7). The Compression Residue may be empty. Both the Rule ID and the Compression Residue potentially have a variable size, and are not necessarily a multiple of bytes in size.

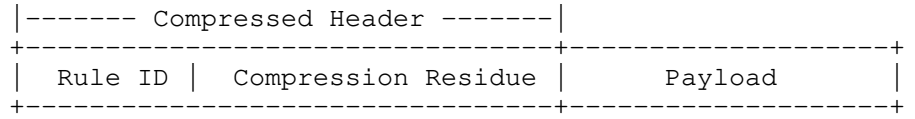


Figure 4: SCHC Packet

5.2. Functional mapping

Figure 5 maps the functional elements of Figure 3 onto the LPWAN architecture elements of Figure 1.

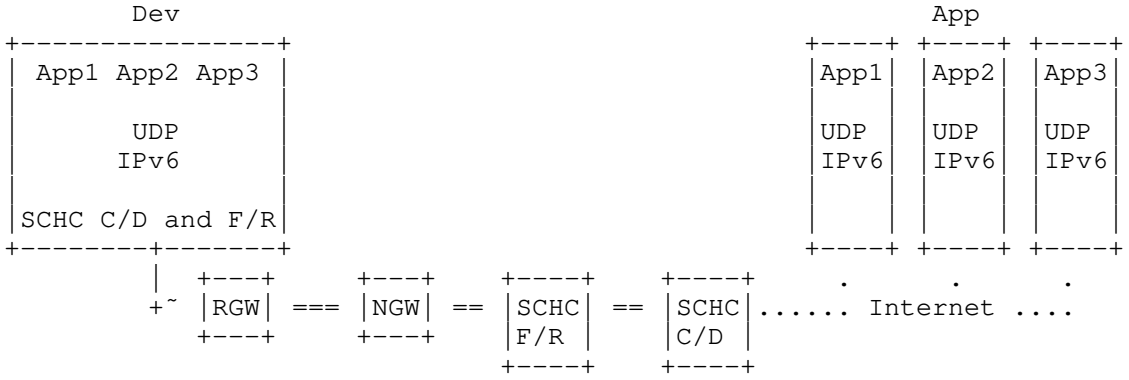


Figure 5: Architecture

SCHC C/D and SCHC F/R are located on both sides of the LPWAN transmission, hereafter called "the Dev side" and "the Network infrastructure side".

The operation in the Uplink direction is as follows. The Device application uses IPv6 or IPv6/UDP protocols. Before sending the packets, the Dev compresses their headers using SCHC C/D and, if the SCHC Packet resulting from the compression needs to be fragmented by SCHC, SCHC F/R is performed (see Section 8). The resulting SCHC Fragments are sent to an LPWAN Radio Gateway (RGW) which forwards them to a Network Gateway (NGW). The NGW sends the data to a SCHC F/R for re-assembly (if needed) and then to the SCHC C/D for decompression. After decompression, the packet can be sent over the Internet to one or several LPWAN Application Servers (App).

The SCHC F/R and C/D on the Network infrastructure side can be part of the NGW, or located in the Internet as long as a tunnel is established between them and the NGW. For some LPWAN technologies, it may be suitable to locate the SCHC F/R functionality nearer the NGW, in order to better deal with time constraints of such technologies.

The SCHC C/Ds on both sides MUST share the same set of Rules. So MUST the SCHC F/Rs on both sides.

The operation in the Downlink direction is similar to that in the Uplink direction, only reversing the order in which the architecture elements are traversed.

6. Rule ID

Rule IDs identify the Rules used for Compression/Decompression or for Fragmentation/Reassembly.

The scope of the Rule ID of a Compression/Decompression Rule is the link between the SCHC C/D in a given Dev and the corresponding SCHC C/D in the Network infrastructure side. The scope of the Rule ID of a Fragmentation/Reassembly Rule is the link between the SCHC F/R in a given Dev and the corresponding SCHC F/R in the Network infrastructure side. If such a link is bidirectional, the scope includes both directions.

Inside their scopes, Rules for Compression/Decompression and Rules for Fragmentation/Reassembly share the same Rule ID space.

The size of the Rule IDs is not specified in this document, as it is implementation-specific and can vary according to the LPWAN technology and the number of Rules, among others. It is defined in Profiles.

The Rule IDs are used:

- For SCHC C/D, to identify the Rule (i.e., the set of Field Descriptions) that is used to compress a packet header.
  - At least one Rule ID MUST be allocated to tagging packets for which SCHC compression was not possible (i.e., no matching compression Rule was found).
- In SCHC F/R, to identify the specific mode and settings of F/R for one direction of traffic (Up or Dw).
  - When F/R is used for both communication directions, at least two Rule ID values are needed for F/R, one per direction of traffic. This is because F/R may entail control messages flowing in the reverse direction compared to data traffic.

7. Compression/Decompression

Compression with SCHC is based on using a set of Rules, called the Context, to compress or decompress headers. SCHC avoids Context synchronization traffic, which consumes considerable bandwidth in other header compression mechanisms such as RoHC [RFC5795]. Since

the content of packets is highly predictable in LPWAN networks, static Contexts can be stored beforehand. The Contexts MUST be stored at both ends, and they can be learned by a provisioning protocol or by out of band means, or they can be pre-provisioned. The way the Contexts are provisioned is out of the scope of this document.

7.1. SCHC C/D Rules

The main idea of the SCHC compression scheme is to transmit the Rule ID to the other end instead of sending known field values. This Rule ID identifies a Rule that matches the original packet values. Hence, when a value is known by both ends, it is only necessary to send the corresponding Rule ID over the LPWAN network. The manner by which Rules are generated is out of the scope of this document. The Rules MAY be changed at run-time but the mechanism is out of scope of this document.

The Context is a set of Rules. See Figure 6 for a high level, abstract representation of the Context. The formal specification of the representation of the Rules is outside the scope of this document.

Each Rule itself contains a list of Field Descriptions composed of a Field Identifier (FID), a Field Length (FL), a Field Position (FP), a Direction Indicator (DI), a Target Value (TV), a Matching Operator (MO) and a Compression/Decompression Action (CDA).

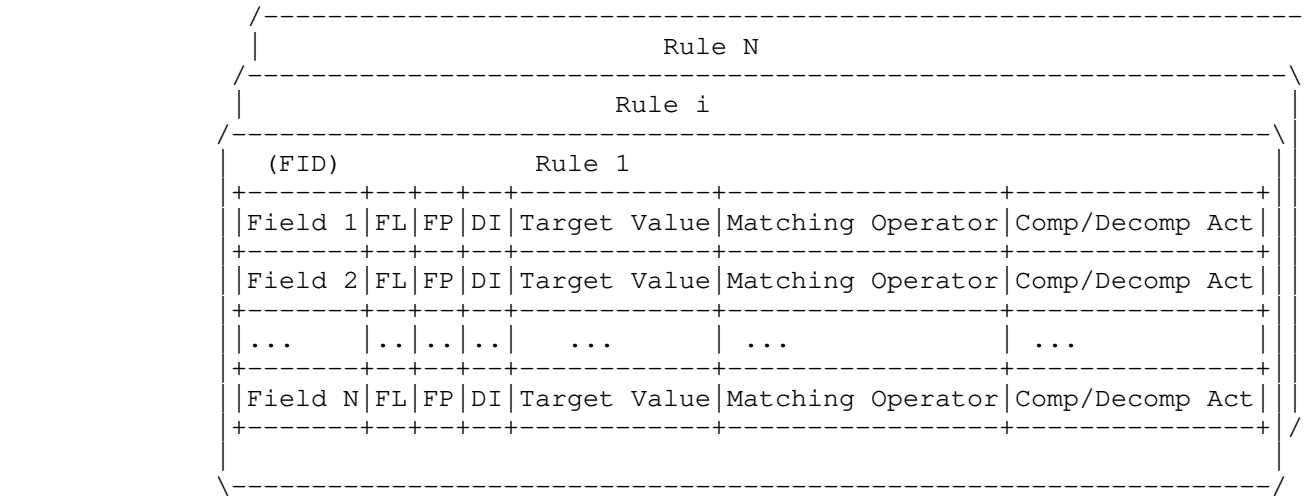


Figure 6: A Compression/Decompression Context

A Rule does not describe how the compressor parses a packet header to find and identify each field (e.g., the IPv6 Source Address, the UDP Destination Port or a CoAP URI path option). It is assumed that there is a protocol parser alongside SCHC that is able to identify all the fields encountered in the headers to be compressed, and to label them with a Field ID. Rules only describe the compression/ decompression behavior for each header field, after it has been identified.

In a Rule, the Field Descriptions are listed in the order in which the fields appear in the packet header. The Field Descriptions describe the header fields with the following entries:

- Field ID (FID) designates a protocol and field (e.g., UDP Destination Port), unambiguously among all protocols that a SCHC compressor processes. In the presence of protocol nesting, the Field ID also identifies the nesting.

- Field Length (FL) represents the length of the original field. It can be either a fixed value (in bits) if the length is known when the Rule is created or a type if the length is variable. The length of a header field is defined by its own protocol specification (e.g., IPv6 or UDP). If the length is variable, the type defines the process to compute the length and its unit (bits, bytes...).

- Field Position (FP): most often, a field only occurs once in a packet header. However, some fields may occur multiple times. An example is the uri-path of CoAP. FP indicates which occurrence this Field Description applies to. If FP is not specified in the Field Description, it takes the default value of 1. The value 1 designates the first occurrence. The value 0 is special. It means "don't care", see Section 7.3.

- A Direction Indicator (DI) indicates the packet direction(s) this Field Description applies to. Three values are possible:

  * UPLINK (Up): this Field Description is only applicable to packets sent by the Dev to the App,

  * DOWNLINK (Dw): this Field Description is only applicable to packets sent from the App to the Dev,

  * BIDIRECTIONAL (Bi): this Field Description is applicable to packets traveling both Up and Dw.

- Target Value (TV) is the value used to match against the packet header field. The Target Value can be a scalar value of any type

(integer, strings, etc.) or a more complex structure (array, list, etc.). The types and representations are out of scope for this document.

- Matching Operator (MO) is the operator used to match the Field Value and the Target Value. The Matching Operator may require some parameters. MO is only used during the compression phase. The set of MOs defined in this document can be found in Section 7.4.

- Compression Decompression Action (CDA) describes the compression and decompression processes to be performed after the MO is applied. Some CDAs might use parameter values for their operation. CDAs are used in both the compression and the decompression functions. The set of CDAs defined in this document can be found in Section 7.5.

7.2. Rule ID for SCHC C/D

Rule IDs are sent by the compression function in one side and are received for the decompression function in the other side. In SCHC C/D, the Rule IDs are specific to the Context related to one Dev. Hence, multiple Dev instances, which refer to different header compression Contexts, MAY reuse the same Rule ID for different Rules. On the Network infrastructure side, in order to identify the correct Rule to be applied, the SCHC Decompressor needs to associate the Rule ID with the Dev identifier. Similarly, the SCHC Compressor on the Network infrastructure side first identifies the destination Dev before looking for the appropriate compression Rule (and associated Rule ID) in the Context of that Dev.

7.3. Packet processing

The compression/decompression process follows several phases:

- Compression Rule selection: the general idea is to browse the Rule set to find a Rule that has a matching Field Descriptor (given the DI and FP) for all and only those header fields that appear in the packet being compressed. The detailed algorithm is the following:

  * The first step is to check the Field Identifiers (FID). If any header field of the packet being examined cannot be matched with a Field Description with the correct FID, the Rule MUST be disregarded. If any Field Description in the Rule has a FID that cannot be matched to one of the header fields of the packet being examined, the Rule MUST be disregarded.

* The next step is to match the Field Descriptions by their direction, using the Direction Indicator (DI). If any field of the packet header cannot be matched with a Field Description with the correct FID and DI, the Rule MUST be disregarded.

* Then the Field Descriptions are further selected according to Field Position (FP). If any field of the packet header cannot be matched with a Field Description with the correct FID, DI and FP, the Rule MUST be disregarded.

  The value 0 for FP means "don't care", i.e. the comparison of this Field Description's FP with the position of the field of the packet header being compressed returns True, whatever that position. FP=0 can be useful to build compression Rules for protocols headers in which some fields order is irrelevant. An example could be uri-queries in CoAP. Care needs to be exercised when writing Rules containing FP=0 values. Indeed, it may result in decompressed packets having fields ordered differently compared to the original packet.

* Once each header field has been associated with a Field Description with matching FID, DI and FP, each packet field's value is then compared to the corresponding Target Value (TV) stored in the Rule for that specific field, using the matching operator (MO). If every field in the packet header satisfies the corresponding matching operators (MO) of a Rule (i.e. all MO results are True), that Rule is valid for use to compress the header. Otherwise, the Rule MUST be disregarded.

  This specification does not prevent multiple Rules from matching the above steps and therefore being valid for use. Which Rule to use among multiple valid Rules is left to the implementation. As long as the same Rule set is installed at both ends, this degree of freedom does not constitute an interoperability issue.

* If no valid compression Rule is found, then the packet MUST be sent uncompressed using the Rule ID dedicated to this purpose (see Section 6). The entire packet header is the Compression Residue (see Figure 4). Sending an uncompressed header is likely to require SCHC F/R.

o Compression: if a valid Rule was found, each field of the header is compressed according to the Compression/Decompression Actions (CDAs) of the Rule. The fields are compressed in the order that the Field Descriptions appear in the Rule. The compression of each field results in a residue, which may be empty. The Compression Residue for the packet header is the concatenation of

the non-empty residues for each field of the header, in the order the Field Descriptions appear in the Rule. The order in which the Field Descriptions appear in the Rule is therefore semantically important.

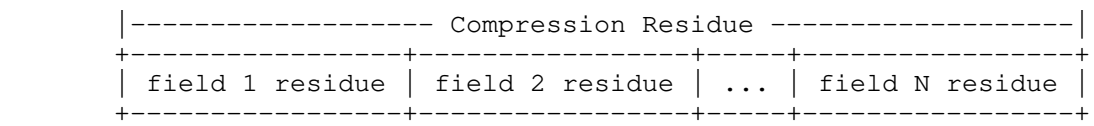


Figure 7: Compression Residue structure

- Sending: The Rule ID is sent to the other end followed by the Compression Residue (which could be empty) or the uncompressed header, and directly followed by the payload (see Figure 4). The way the Rule ID is sent will be specified in the Profile and is out of the scope of the present document. For example, it could be included in an L2 header or sent as part of the L2 payload.

- Decompression: when decompressing, on the Network infrastructure side the SCHC C/D needs to find the correct Rule based on the L2 address of the Dev; in this way, it can use the DevIID and the Rule ID. On the Dev side, only the Rule ID is needed to identify the correct Rule since the Dev typically only holds Rules that apply to itself.

  This Rule describes the compressed header format. From this, the decompressor determines the order of the residues, the fixed-sized or variable-sized nature of each residue (see Section 7.5.2), and the size of the fixed-sized residues.

  From the received compressed header, it can therefore retrieve all the residue values and associate them to the corresponding header fields.

  For each field in the header, the receiver applies the CDA action associated to that field in order to reconstruct the original header field value. The CDA application order can be different from the order in which the fields are listed in the Rule. In particular, Compute-* MUST be applied after the application of the CDAs of all the fields it computes on.

7.4. Matching operators

Matching Operators (MOs) are functions used by both SCHC C/D endpoints. They are not typed and can be applied to integer, string

or any other data type. The result of the operation can either be True or False. MOs are defined as follows:

- equal: The match result is True if the field value in the packet matches the TV.
- ignore: No matching is attempted between the field value in the packet and the TV in the Rule. The result is always true.
- MSB(x): A match is obtained if the most significant (leftmost) x bits of the packet header field value are equal to the TV in the Rule. The x parameter of the MSB MO indicates how many bits are involved in the comparison. If the FL is described as variable, the x parameter must be a multiple of the FL unit. For example, x must be multiple of 8 if the unit of the variable length is bytes.
- match-mapping: With match-mapping, the Target Value is a list of values. Each value of the list is identified by an index. Compression is achieved by sending the index instead of the original header field value. This operator matches if the header field value is equal to one of the values in the target list.

7.5. Compression Decompression Actions (CDA)

The Compression Decompression Action (CDA) describes the actions taken during the compression of header fields and the inverse action taken by the decompressor to restore the original value.

Action	Compression	Decompression
not-sent	elided	use TV stored in Rule
value-sent	send	use received value
mapping-sent	send index	retrieve value from TV list
LSB	send LSB	concat. TV and received value
compute-*	elided	recompute at decompressor
DevIID	elided	build IID from L2 Dev addr
AppIID	elided	build IID from L2 App addr

Table 1: Compression and Decompression Actions

Table 1 summarizes the basic actions that can be used to compress and decompress a field. The first column shows the action's name. The second and third columns show the compression and decompression behaviors for each action.

7.5.1. processing fixed-length fields

If the field is identified in the Field Description as being of fixed length, then applying the CDA to compress this field results in a fixed amount of bits. The residue for that field is simply the bits resulting from applying the CDA to the field. This value may be empty (e.g., not-sent CDA), in which case the field residue is absent from the Compression Residue.

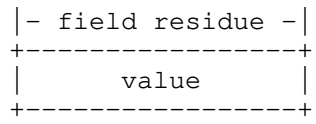


Figure 8: fixed sized field residue structure

7.5.2. processing variable-length fields

If the field is identified in the Field Description as being of variable length, then applying the CDA to compress this field may result in a value of fixed size (e.g., not-sent or mapping-sent) or of variable size (e.g., value-sent or LSB). In the latter case, the residue for that field is the bits that result from applying the CDA to the field, preceded with the size of the value. The most significant bit of the size is stored to the left (leftmost bit of the residue field).

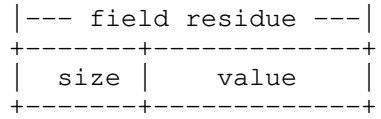


Figure 9: variable sized field residue structure

The size (using the unit defined in the FL) is encoded on 4, 12 or 28 bits as follows:

- If the size is between 0 and 14, it is encoded as a 4 bits unsigned integer.
- Sizes between 15 and 254 are encoded as 0b1111 followed by the 8 bits unsigned integer.
- Larger sizes are encoded as 0xfff followed by the 16 bits unsigned integer.

If the field is identified in the Field Description as being of variable length and this field is not present in the packet header being compressed, size 0 MUST be sent to denote its absence.

7.5.3. not-sent CDA

The not-sent action can be used when the field value is specified in a Rule and therefore known by both the Compressor and the Decompressor. This action SHOULD be used with the "equal" MO. If MO is "ignore", there is a risk to have a decompressed field value different from the original field that was compressed.

The compressor does not send any residue for a field on which not-sent compression is applied.

The decompressor restores the field value with the Target Value stored in the matched Rule identified by the received Rule ID.

7.5.4. value-sent CDA

The value-sent action can be used when the field value is not known by both the Compressor and the Decompressor. The field is sent in its entirety, using the same bit order as in the original packet header.

If this action is performed on a variable length field, the size of the residue value (using the units defined in FL) MUST be sent as described in Section 7.5.2.

This action is generally used with the "ignore" MO.

7.5.5. mapping-sent CDA

The mapping-sent action is used to send an index (the index into the Target Value list of values) instead of the original value. This action is used together with the "match-mapping" MO.

On the compressor side, the match-mapping Matching Operator searches the TV for a match with the header field value. The mapping-sent CDA then sends the corresponding index as the field residue. The most significant bit of the index is stored to the left (leftmost bit of the residue field).

On the decompressor side, the CDA uses the received index to restore the field value by looking up the list in the TV.

The number of bits sent is the minimal size for coding all the possible indices.

The first element in the list MUST be represented by index value 0, and successive elements in the list MUST have indices incremented by 1.

7.5.6. LSB CDA

The LSB action is used together with the "MSB(x)" MO to avoid sending the most significant part of the packet field if that part is already known by the receiving end.

The compressor sends the Least Significant Bits as the field residue value. The number of bits sent is the original header field length minus the length specified in the MSB(x) MO. The bits appear in the residue in the same bit order as in the original packet header.

The decompressor concatenates the x most significant bits of Target Value and the received residue value.

If this action is performed on a variable length field, the size of the residue value (using the units defined in FL) MUST be sent as described in Section 7.5.2.

7.5.7. DevIID, AppIID CDA

These actions are used to process respectively the Dev and the App Interface Identifiers (DevIID and AppIID) of the IPv6 addresses. AppIID CDA is less common since most current LPWAN technologies frames contain a single L2 address, which is the Dev’s address.

The IID value MAY be computed from the Device ID present in the L2 header, or from some other stable identifier. The computation is specific to each Profile and MAY depend on the Device ID size.

In the downlink direction (Dw), at the compressor, the DevIID CDA may be used to generate the L2 addresses on the LPWAN, based on the packet’s Destination Address.

7.5.8. Compute-*

Some fields can be elided at the compressor and recomputed locally at the decompressor.

Because the field is uniquely identified by its Field ID (e.g., UDP length), the relevant protocol specification unambiguously defines the algorithm for such computation.

Examples of fields that know how to recompute themselves are UDP length, IPv6 length and UDP checksum.

8. Fragmentation/Reassembly

8.1. Overview

In LPWAN technologies, the L2 MTU typically ranges from tens to hundreds of bytes. Some of these technologies do not have an internal fragmentation/reassembly mechanism.

The optional SCHC Fragmentation/Reassembly (SCHC F/R) functionality enables such LPWAN technologies to comply with the IPv6 MTU requirement of 1280 bytes [RFC8200]. It is OPTIONAL to implement per this specification, but Profiles may specify that it is REQUIRED.

This specification includes several SCHC F/R modes, which allow for a range of reliability options such as optional SCHC Fragment retransmission. More modes may be defined in the future.

The same SCHC F/R mode MUST be used for all SCHC Fragments of a given SCHC Packet. This document does not specify which mode(s) must be implemented and used over a specific LPWAN technology. That information will be given in Profiles.

SCHC allows transmitting non-fragmented SCHC Packet concurrently with fragmented SCHC Packets. In addition, SCHC F/R provides protocol elements that allow transmitting several fragmented SCHC Packets concurrently, i.e. interleaving the transmission of fragments from different fragmented SCHC Packets. A Profile MAY restrict the latter behavior.

The L2 Word size (see Section 4) determines the encoding of some messages. SCHC F/R usually generates SCHC Fragments and SCHC ACKs that are multiples of L2 Words.

8.2. SCHC F/R Protocol Elements

This subsection describes the different elements that are used to enable the SCHC F/R functionality defined in this document. These elements include the SCHC F/R messages, tiles, windows, bitmaps, counters, timers and header fields.

The elements are described here in a generic manner. Their application to each SCHC F/R mode is found in Section 8.4.

8.2.1. Messages

SCHC F/R defines the following messages:

- SCHC Fragment: A message that carries part of a SCHC Packet from the sender to the receiver.

- SCHC ACK: An acknowledgement for fragmentation, by the receiver to the sender. This message is used to indicate whether or not the reception of pieces of, or the whole of the fragmented SCHC Packet, was successful.

- SCHC ACK REQ: A request by the sender for a SCHC ACK from the receiver.

- SCHC Sender-Abort: A message by the sender telling the receiver that it has aborted the transmission of a fragmented SCHC Packet.

- SCHC Receiver-Abort: A message by the receiver to tell the sender to abort the transmission of a fragmented SCHC Packet.

The format of these messages is provided in Section 8.3.

8.2.2. Tiles, Windows, Bitmaps, Timers, Counters

8.2.2.1. Tiles

The SCHC Packet is fragmented into pieces, hereafter called tiles. The tiles MUST be non-empty and pairwise disjoint. Their union MUST be equal to the SCHC Packet.

See Figure 10 for an example.

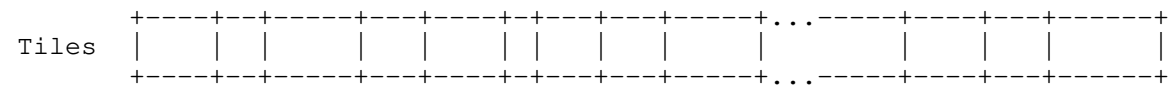


Figure 10: a SCHC Packet fragmented in tiles

Modes (see Section 8.4) MAY place additional constraints on tile sizes.

Each SCHC Fragment message carries at least one tile in its Payload, if the Payload field is present.

8.2.2.2. Windows

Some SCHC F/R modes may handle successive tiles in groups, called windows.

If windows are used

- all the windows of a SCHC Packet, except the last one, MUST contain the same number of tiles. This number is WINDOW_SIZE.
- WINDOW_SIZE MUST be specified in a Profile.
- the windows are numbered.
- their numbers MUST increment by 1 from 0 upward, from the start of the SCHC Packet to its end.
- the last window MUST contain WINDOW_SIZE tiles or less.
- tiles are numbered within each window.
- the tile indices MUST decrement by 1 from WINDOW_SIZE - 1 downward, looking from the start of the SCHC Packet toward its end.
- each tile of a SCHC Packet is therefore uniquely identified by a window number and a tile index within this window.

See Figure 11 for an example.

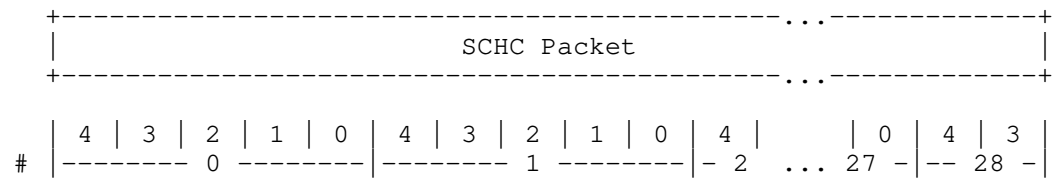


Figure 11: a SCHC Packet fragmented in tiles grouped in 29 windows, with WINDOW_SIZE = 5

Appendix E discusses the benefits of selecting one among multiple window sizes depending on the size of the SCHC Packet to be fragmented.

When windows are used

- Bitmaps (see Section 8.2.2.3) MAY be sent back by the receiver to the sender in a SCHC ACK message.
- A Bitmap corresponds to exactly one Window.

8.2.2.3. Bitmaps

Each bit in the Bitmap for a window corresponds to a tile in the window. Each Bitmap has therefore WINDOW_SIZE bits. The bit at the left-most position corresponds to the tile numbered WINDOW_SIZE - 1. Consecutive bits, going right, correspond to sequentially decreasing tile indices. In Bitmaps for windows that are not the last one of a SCHC Packet, the bit at the right-most position corresponds to the tile numbered 0. In the Bitmap for the last window, the bit at the right-most position corresponds either to the tile numbered 0 or to a tile that is sent/received as "the last one of the SCHC Packet" without explicitly stating its number (see Section 8.3.1.2).

At the receiver

- a bit set to 1 in the Bitmap indicates that a tile associated with that bit position has been correctly received for that window.

- a bit set to 0 in the Bitmap indicates that there has been no tile correctly received, associated with that bit position, for that window. Possible reasons include that the tile was not sent at all, not received, or received with errors.

8.2.2.4. Timers and counters

Some SCHC F/R modes can use the following timers and counters

- Inactivity Timer: a SCHC Fragment receiver uses this timer to abort waiting for a SCHC F/R message.

- Retransmission Timer: a SCHC Fragment sender uses this timer to abort waiting for an expected SCHC ACK.

- Attempts: this counter counts the requests for SCHC ACKs, up to MAX_ACK_REQUESTS.

8.2.3. Integrity Checking

The integrity of the fragmentation-reassembly process of a SCHC Packet MUST be checked at the receive end. A Profile MUST specify how integrity checking is performed.

It is RECOMMENDED that integrity checking be performed by computing a Reassembly Check Sequence (RCS) based on the SCHC Packet at the sender side and transmitting it to the receiver for comparison with the RCS locally computed after reassembly.

The RCS supports UDP checksum elision by SCHC C/D (see Section 10.11).

The CRC32 polynomial 0xEDB88320 (i.e., the reversed polynomial representation, which is used in the Ethernet standard [ETHERNET]) is RECOMMENDED as the default algorithm for computing the RCS.

The RCS MUST be computed on the full SCHC Packet concatenated with the padding bits, if any, of the SCHC Fragment carrying the last tile. The rationale is that the SCHC reassembler has no way of knowing the boundary between the last tile and the padding bits. Indeed, this requires decompressing the SCHC Packet, which is out of the scope of the SCHC reassembler.

The concatenation of the complete SCHC Packet and any padding bits, if present, of the last SCHC Fragment does not generally constitute an integer number of bytes. CRC libraries are usually byte-oriented. It is RECOMMENDED that the concatenation of the complete SCHC Packet and any last fragment padding bits be zero-extended to the next byte boundary and that the RCS be computed on that byte array.

8.2.4. Header Fields

The SCHC F/R messages contain the following fields (see the formats in Section 8.3):

- Rule ID: this field is present in all the SCHC F/R messages. The Rule identifies
  - that a SCHC F/R message is being carried, as opposed to an unfragmented SCHC Packet,
  - which SCHC F/R mode is used
  - in case this mode uses windows, what the value of WINDOW_SIZE is,
  - what other optional fields are present and what the field sizes are.

  The Rule tells apart a non-fragmented SCHC Packet from SCHC Fragments. It will also tell apart SCHC Fragments of fragmented SCHC Packets that use different SCHC F/R modes or different parameters. Interleaved transmission of these is therefore possible.

  All SCHC F/R messages pertaining to the same SCHC Packet MUST bear the same Rule ID.

- Datagram Tag (DTag). This field allows differentiating SCHC F/R messages belonging to different SCHC Packets that may be using the same Rule ID simultaneously. Hence, it allows interleaving fragments of a new SCHC Packet with fragments of a previous SCHC Packet under the same Rule ID.

  The size of the DTag field (called T, in bits) is defined by each Profile for each Rule ID. When T is 0, the DTag field does not appear in the SCHC F/R messages and the DTag value is defined as 0.

  When T is 0, there can be no more than one fragmented SCHC Packet in transit for each fragmentation Rule ID.

  If T is not 0, DTag

  * MUST be set to the same value for all the SCHC F/R messages related to the same fragmented SCHC Packet,

  * MUST be set to different values for SCHC F/R messages related to different SCHC Packets that are being fragmented under the same Rule ID, and whose transmission may overlap.

- W: The W field is optional. It is only present if windows are used. Its presence and size (called M, in bits) is defined by each SCHC F/R mode and each Profile for each Rule ID.

  This field carries information pertaining to the window a SCHC F/R message relates to. If present, W MUST carry the same value for all the SCHC F/R messages related to the same window. Depending on the mode and Profile, W may carry the full window number, or just the least significant bit or any other partial representation of the window number.

- Fragment Compressed Number (FCN). The FCN field is present in the SCHC Fragment Header. Its size (called N, in bits) is defined by each Profile for each Rule ID.

  This field conveys information about the progress in the sequence of tiles being transmitted by SCHC Fragment messages. For example, it can contain a partial, efficient representation of a larger-sized tile index. The description of the exact use of the FCN field is left to each SCHC F/R mode. However, two values are reserved for special purposes. They help control the SCHC F/R process:

  * The FCN value with all the bits equal to 1 (called All-1) signals that the very last tile of a SCHC Packet has been

transmitted. By extension, if windows are used, the last window of a packet is called the All-1 window.

* If windows are used, the FCN value with all the bits equal to 0 (called All-0) signals the last tile of a window that is not the last one of the SCHC packet. By extension, such a window is called an All-0 window.

- Reassembly Check Sequence (RCS). This field only appears in the All-1 SCHC Fragments. Its size (called U, in bits) is defined by each Profile for each Rule ID.

  See Section 8.2.3 for the RCS default size, default polynomial and details on RCS computation.

- C (integrity Check): C is a 1-bit field. This field is used in the SCHC ACK message to report on the reassembled SCHC Packet integrity check (see Section 8.2.3).

  A value of 1 tells that the integrity check was performed and is successful. A value of 0 tells that the integrity check was not performed, or that is was a failure.

- Compressed Bitmap. The Compressed Bitmap is used together with windows and Bitmaps (see Section 8.2.2.3). Its presence and size is defined for each F/R mode for each Rule ID.

  This field appears in the SCHC ACK message to report on the receiver Bitmap (see Section 8.3.2.1).

8.3. SCHC F/R Message Formats

This section defines the SCHC Fragment formats, the SCHC ACK format, the SCHC ACK REQ format and the SCHC Abort formats.

8.3.1. SCHC Fragment format

A SCHC Fragment conforms to the general format shown in Figure 12. It comprises a SCHC Fragment Header and a SCHC Fragment Payload. The SCHC Fragment Payload carries one or several tile(s).

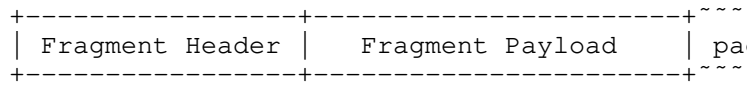


Figure 12: SCHC Fragment general format

8.3.1.1. Regular SCHC Fragment

The Regular SCHC Fragment format is shown in Figure 13. Regular SCHC Fragments are generally used to carry tiles that are not the last one of a SCHC Packet. The DTag field and the W field are OPTIONAL, their presence is specified by each mode and Profile.

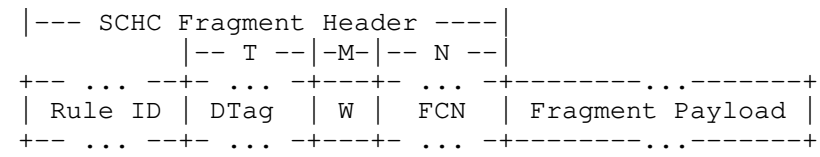


Figure 13: Detailed Header Format for Regular SCHC Fragments

The FCN field MUST NOT contain all bits set to 1.

Profiles MUST ensure that a SCHC Fragment with FCN equal to 0 (called an All-0 SCHC Fragment) is distinguishable by size, even in the presence of padding, from a SCHC ACK REQ message (see Section 8.3.3) with the same Rule ID value and with the same T, M and N values. This condition is met if the Payload is at least the size of an L2 Word. This condition is also met if the SCHC Fragment Header is a multiple of L2 Words.

8.3.1.2. All-1 SCHC Fragment

The All-1 SCHC Fragment format is shown in Figure 14. The sender uses the All-1 SCHC Fragment format for the message that completes the emission of a fragmented SCHC Packet. The DTag field, the W field, the RCS field and the Payload are OPTIONAL, their presence is specified by each mode and Profile. At least one of RCS field or Payload MUST be present. The FCN field is all ones.

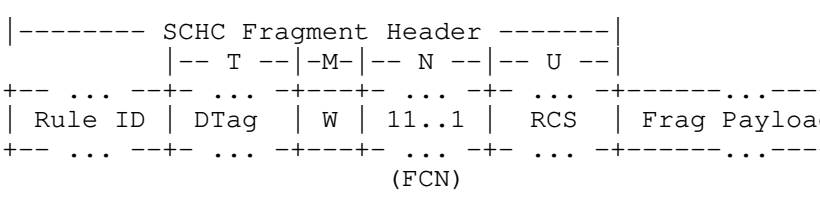


Figure 14: Detailed Header Format for the All-1 SCHC Fragment

Profiles MUST ensure that an All-1 SCHC Fragment message is distinguishable by size, even in the presence of padding, from a SCHC Sender-Abort message (see Section 8.3.4) with the same Rule ID value and with the same T, M and N values. This condition is met if the RCS is present and is at least the size of an L2 Word, or if the

Payload is present and at least the size an L2 Word. This condition is also met if the SCHC Sender-Abort Header is a multiple of L2 Words.

8.3.2. SCHC ACK format

The SCHC ACK message is shown in Figure 15. The DTag field and the W field are OPTIONAL, their presence is specified by each mode and Profile. The Compressed Bitmap field MUST be present in SCHC F/R modes that use windows, and MUST NOT be present in other modes.

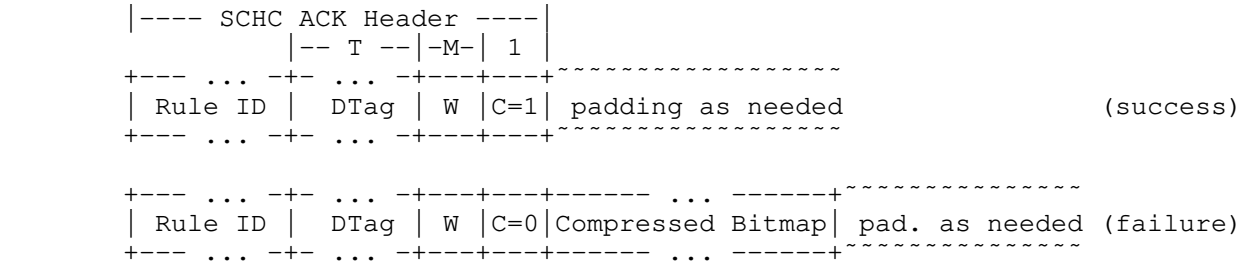


Figure 15: Format of the SCHC ACK message

The SCHC ACK Header contains a C bit (see Section 8.2.4).

If the C bit is set to 1 (integrity check successful), no Bitmap is carried.

If the C bit is set to 0 (integrity check not performed or failed) and if windows are used, a Compressed Bitmap for the window referred to by the W field is transmitted as specified in Section 8.3.2.1.

8.3.2.1. Bitmap Compression

For transmission, the Compressed Bitmap in the SCHC ACK message is defined by the following algorithm (see Figure 16 for a follow-along example):

- Build a temporary SCHC ACK message that contains the Header followed by the original Bitmap (see Section 8.2.2.3 for a description of Bitmaps).

- Position scissors at the end of the Bitmap, after its last bit.

- While the bit on the left of the scissors is 1 and belongs to the Bitmap, keep moving left, then stop. When this is done,

- While the scissors are not on an L2 Word boundary of the SCHC ACK message and there is a Bitmap bit on the right of the scissors, keep moving right, then stop.

- At this point, cut and drop off any bits to the right of the scissors

When one or more bits have effectively been dropped off as a result of the above algorithm, the SCHC ACK message is a multiple of L2 Words, no padding bits will be appended.

Because the SCHC Fragment sender knows the size of the original Bitmap, it can reconstruct the original Bitmap from the Compressed Bitmap received in the SCH ACK message.

Figure 16 shows an example where L2 Words are actually bytes and where the original Bitmap contains 17 bits, the last 15 of which are all set to 1.

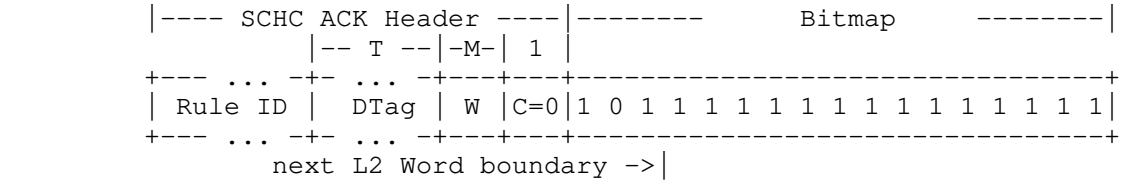


Figure 16: SCHC ACK Header plus uncompressed Bitmap

Figure 17 shows that the last 14 bits are not sent.

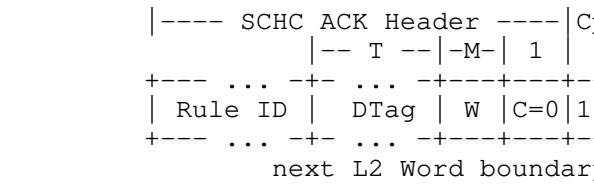


Figure 17: Resulting SCHC ACK message with Compressed Bitmap

Figure 18 shows an example of a SCHC ACK with tile indices ranging from 6 down to 0, where the Bitmap indicates that the second and the fourth tile of the window have not been correctly received.

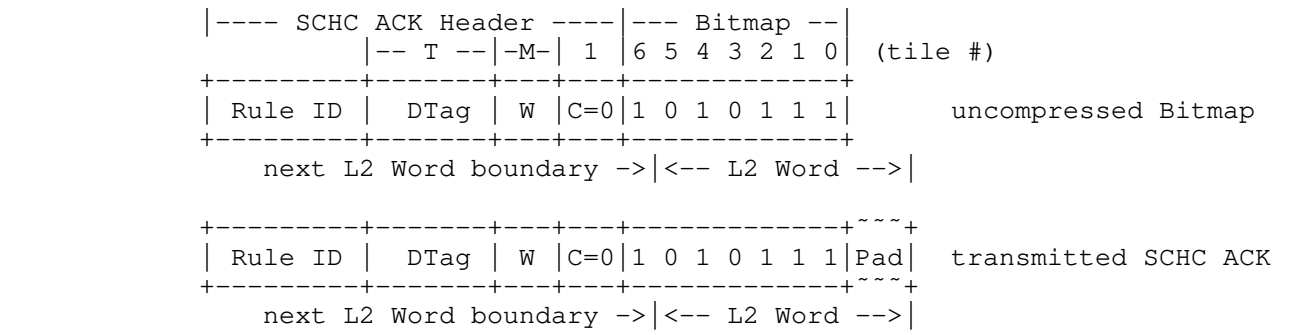


Figure 18: Example of a SCHC ACK message, missing tiles

Figure 19 shows an example of a SCHC ACK with FCN ranging from 6 down to 0, where integrity check has not been performed or has failed and the Bitmap indicates that there is no missing tile in that window.

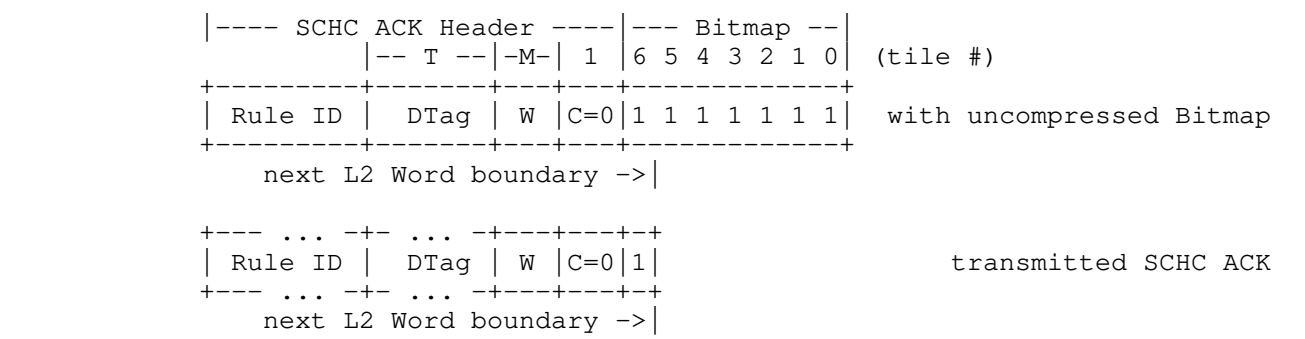


Figure 19: Example of a SCHC ACK message, no missing tile

8.3.3. SCHC ACK REQ format

The SCHC ACK REQ is used by a sender to request a SCHC ACK from the receiver. Its format is shown in Figure 20. The DTag field and the W field are OPTIONAL, their presence is specified by each mode and Profile. The FCN field is all zero.

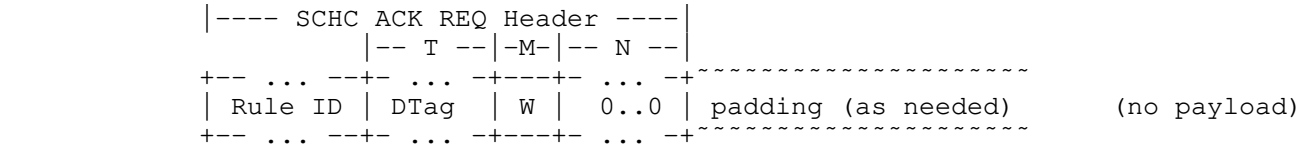


Figure 20: SCHC ACK REQ format

8.3.4. SCHC Sender-Abort format

When a SCHC Fragment sender needs to abort an on-going fragmented SCHC Packet transmission, it sends a SCHC Sender-Abort message to the SCHC Fragment receiver.

The SCHC Sender-Abort format is shown in Figure 21. The DTag field and the W field are OPTIONAL, their presence is specified by each mode and Profile. The FCN field is all ones.

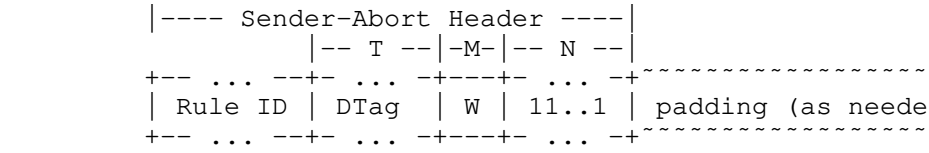


Figure 21: SCHC Sender-Abort format

If the W field is present,

- the fragment sender MUST set it to all ones. Other values are RESERVED.

- the fragment receiver MUST check its value. If the value is different from all ones, the message MUST be ignored.

The SCHC Sender-Abort MUST NOT be acknowledged.

8.3.5. SCHC Receiver-Abort format

When a SCHC Fragment receiver needs to abort an on-going fragmented SCHC Packet transmission, it transmits a SCHC Receiver-Abort message to the SCHC Fragment sender.

The SCHC Receiver-Abort format is shown in Figure 22. The DTag field and the W field are OPTIONAL, their presence is specified by each mode and Profile.

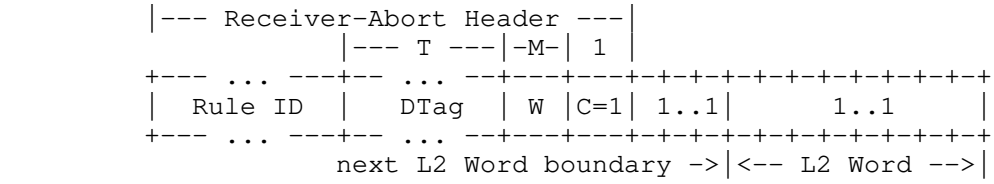


Figure 22: SCHC Receiver-Abort format

If the W field is present,

- the fragment receiver MUST set it to all ones. Other values are RESERVED.

- if the value is different from all ones, the fragment sender MUST ignore the message.

The SCHC Receiver-Abort has the same header as a SCHC ACK message. The bits that follow the SCHC Receiver-Abort Header MUST be as follows

- if the Header does not end at an L2 Word boundary, append bits set to 1 as needed to reach the next L2 Word boundary

- append exactly one more L2 Word with bits all set to ones

Such a bit pattern never occurs in a legitimate SCHC ACK. This is how the fragment sender recognizes a SCHC Receiver-Abort.

The SCHC Receiver-Abort MUST NOT be acknowledged.

8.4. SCHC F/R modes

This specification includes several SCHC F/R modes, which

- allow for a range of reliability options, such as optional SCHC Fragment retransmission

- support various LPWAN characteristics, such as links with variable MTU or unidirectional links.

More modes may be defined in the future.

Appendix B provides examples of fragmentation sessions based on the modes described hereafter.

Appendix C provides examples of Finite State Machines implementing the SCHC F/R modes described hereafter.

8.4.1. No-ACK mode

The No-ACK mode has been designed under the assumption that data unit out-of-sequence delivery does not occur between the entity performing fragmentation and the entity performing reassembly. This mode supports LPWAN technologies that have a variable MTU.

In No-ACK mode, there is no communication from the fragment receiver to the fragment sender. The sender transmits all the SCHC Fragments

without expecting any acknowledgement. Therefore, No-ACK does not require bidirectional links: unidirectional links are just fine.

In No-ACK mode, only the All-1 SCHC Fragment is padded as needed. The other SCHC Fragments are intrinsically aligned to L2 Words.

The tile sizes are not required to be uniform. Windows are not used. The Retransmission Timer is not used. The Attempts counter is not used.

Each Profile MUST specify which Rule ID value(s) correspond to SCHC F/R messages operating in this mode.

The W field MUST NOT be present in the SCHC F/R messages. SCHC ACK MUST NOT be sent. SCHC ACK REQ MUST NOT be sent. SCHC Sender-Abort MAY be sent. SCHC Receiver-Abort MUST NOT be sent.

The value of N (size of the FCN field) is RECOMMENDED to be 1.

Each Profile, for each Rule ID value, MUST define

- the size of the DTag field,
- the size and algorithm for the RCS field,
- the expiration time of the Inactivity Timer

Each Profile, for each Rule ID value, MAY define

- a value of N different from the recommended one,
- the meaning of values sent in the FCN field, for values different from the All-1 value.

For each active pair of Rule ID and DTag values, the receiver MUST maintain an Inactivity Timer. If the receiver is under-resourced to do this, it MUST silently drop the related messages.

8.4.1.1. Sender behavior

At the beginning of the fragmentation of a new SCHC Packet, the fragment sender MUST select a Rule ID and DTag value pair for this SCHC Packet.

Each SCHC Fragment MUST contain exactly one tile in its Payload. The tile MUST be at least the size of an L2 Word. The sender MUST transmit the SCHC Fragments messages in the order that the tiles appear in the SCHC Packet. Except for the last tile of a SCHC

Packet, each tile MUST be of a size that complements the SCHC Fragment Header so that the SCHC Fragment is a multiple of L2 Words without the need for padding bits. Except for the last one, the SCHC Fragments MUST use the Regular SCHC Fragment format specified in Section 8.3.1.1. The SCHC Fragment that carries the last tile MUST be an All-1 SCHC Fragment, described in Section 8.3.1.2.

The sender MAY transmit a SCHC Sender-Abort.

Figure 37 shows an example of a corresponding state machine.

8.4.1.2. Receiver behavior

Upon receiving each Regular SCHC Fragment,

- the receiver MUST reset the Inactivity Timer,
- the receiver assembles the payloads of the SCHC Fragments

On receiving an All-1 SCHC Fragment,

- the receiver MUST append the All-1 SCHC Fragment Payload and the padding bits to the previously received SCHC Fragment Payloads for this SCHC Packet
- the receiver MUST perform the integrity check
- if integrity checking fails, the receiver MUST drop the reassembled SCHC Packet
- the reassembly operation concludes.

On expiration of the Inactivity Timer, the receiver MUST drop the SCHC Packet being reassembled.

On receiving a SCHC Sender-Abort, the receiver MAY drop the SCHC Packet being reassembled.

Figure 38 shows an example of a corresponding state machine.

8.4.2. ACK-Always mode

The ACK-Always mode has been designed under the following assumptions

- Data unit out-of-sequence delivery does not occur between the entity performing fragmentation and the entity performing reassembly

- The L2 MTU value does not change while the fragments of a SCHC Packet are being transmitted.

- There is a feedback path from the reassembler to the fragmenter. See Appendix F for a discussion on using ACK-Always mode on quasi-bidirectional links.

In ACK-Always mode, windows are used. An acknowledgement, positive or negative, is transmitted by the fragment receiver to the fragment sender at the end of the transmission of each window of SCHC Fragments.

The tiles are not required to be of uniform size. In ACK-Always mode, only the All-1 SCHC Fragment is padded as needed. The other SCHC Fragments are intrinsically aligned to L2 Words.

Briefly, the algorithm is as follows: after a first blind transmission of all the tiles of a window, the fragment sender iterates retransmitting the tiles that are reported missing until the fragment receiver reports that all the tiles belonging to the window have been correctly received, or until too many attempts were made. The fragment sender only advances to the next window of tiles when it has ascertained that all the tiles belonging to the current window have been fully and correctly received. This results in a per-window lock-step behavior between the sender and the receiver.

Each Profile MUST specify which Rule ID value(s) correspond to SCHC F/R messages operating in this mode.

The W field MUST be present and its size M MUST be 1 bit.

Each Profile, for each Rule ID value, MUST define

- the value of N (size of the FCN field),

- the value of WINDOW_SIZE, which MUST be strictly less than 2^N,

- the size and algorithm for the RCS field,

- the size of the DTag field,

- the value of MAX_ACK_REQUESTS,

- the expiration time of the Retransmission Timer

- the expiration time of the Inactivity Timer

For each active pair of Rule ID and DTag values, the sender MUST maintain

- one Attempts counter
- one Retransmission Timer

For each active pair of Rule ID and DTag values, the receiver MUST maintain

- one Inactivity Timer
- one Attempts counter

8.4.2.1. Sender behavior

At the beginning of the fragmentation of a new SCHC Packet, the fragment sender MUST select a Rule ID and DTag value pair for this SCHC Packet.

Each SCHC Fragment MUST contain exactly one tile in its Payload. All tiles with the index 0, as well as the last tile, MUST be at least the size of an L2 Word.

In all SCHC Fragment messages, the W field MUST be filled with the least significant bit of the window number that the sender is currently processing.

For a SCHC Fragment that carries a tile other than the last one of the SCHC Packet,

- the Fragment MUST be of the Regular type specified in Section 8.3.1.1
- the FCN field MUST contain the tile index
- each tile MUST be of a size that complements the SCHC Fragment Header so that the SCHC Fragment is a multiple of L2 Words without the need for padding bits.

The SCHC Fragment that carries the last tile MUST be an All-1 SCHC Fragment, described in Section 8.3.1.2.

The fragment sender MUST start by transmitting the window numbered 0.

All message receptions being discussed in the rest of this section are to be understood as "matching the RuleID and DTag pair being processed", even if not spelled out, for brevity.

The sender starts by a "blind transmission" phase, in which it MUST transmit all the tiles composing the window, in decreasing tile index order.

Then, it enters a "retransmission phase" in which it MUST initialize an Attempts counter to 0, it MUST start a Retransmission Timer and it MUST await a SCHC ACK. Then,

- upon receiving a SCHC ACK,
  - if the SCHC ACK indicates that some tiles are missing at the receiver, then the sender MUST transmit all the tiles that have been reported missing, it MUST increment Attempts, it MUST reset the Retransmission Timer and MUST await the next SCHC ACK.
  - if the current window is not the last one and the SCHC ACK indicates that all tiles were correctly received, the sender MUST stop the Retransmission Timer, it MUST advance to the next fragmentation window and it MUST start a blind transmission phase as described above.
  - if the current window is the last one and the SCHC ACK indicates that more tiles were received than the sender sent, the fragment sender MUST send a SCHC Sender-Abort, and it MAY exit with an error condition.
  - if the current window is the last one and the SCHC ACK indicates that all tiles were correctly received yet integrity check was a failure, the fragment sender MUST send a SCHC Sender-Abort, and it MAY exit with an error condition.
  - if the current window is the last one and the SCHC ACK indicates that integrity checking was successful, the sender exits successfully.
- on Retransmission Timer expiration,
  - if Attempts is strictly less that MAX_ACK_REQUESTS, the fragment sender MUST send a SCHC ACK REQ and MUST increment the Attempts counter.
  - otherwise the fragment sender MUST send a SCHC Sender-Abort, and it MAY exit with an error condition.

At any time,

- on receiving a SCHC Receiver-Abort, the fragment sender MAY exit with an error condition.

- on receiving a SCHC ACK that bears a W value different from the W value that it currently uses, the fragment sender MUST silently discard and ignore that SCHC ACK.

Figure 39 shows an example of a corresponding state machine.

8.4.2.2. Receiver behavior

On receiving a SCHC Fragment with a Rule ID and DTag pair not being processed at that time

- the receiver SHOULD check if the DTag value has not recently been used for that Rule ID value, thereby ensuring that the received SCHC Fragment is not a remnant of a prior fragmented SCHC Packet transmission. The initial value of the Inactivity Timer is the RECOMMENDED lifetime for the DTag value at the receiver. If the SCHC Fragment is determined to be such a remnant, the receiver MAY silently ignore it and discard it.

- the receiver MUST start a process to assemble a new SCHC Packet with that Rule ID and DTag value pair.

- the receiver MUST start an Inactivity Timer for that RuleID and DTag pair. It MUST initialize an Attempts counter to 0 for that RuleID and DTag pair. It MUST initialize a window counter to 0. If the receiver is under-resourced to do this, it MUST respond to the sender with a SCHC Receiver Abort.

In the rest of this section, "local W bit" means the least significant bit of the window counter of the receiver.

On reception of any SCHC F/R message for the RuleID and DTag pair being processed, the receiver MUST reset the Inactivity Timer pertaining to that RuleID and DTag pair.

All message receptions being discussed in the rest of this section are to be understood as "matching the RuleID and DTag pair being processed", even if not spelled out, for brevity.

The receiver MUST first initialize an empty Bitmap for the first window, then enter an "acceptance phase", in which

- on receiving a SCHC Fragment or a SCHC ACK REQ, either one having the W bit different from the local W bit, the receiver MUST silently ignore and discard that message.

- on receiving a SCHC ACK REQ with the W bit equal to the local W bit, the receiver MUST send a SCHC ACK for this window.

- on receiving a SCHC Fragment with the W bit equal to the local W bit, the receiver MUST assemble the received tile based on the window counter and on the FCN field in the SCHC Fragment and it MUST update the Bitmap.

  * if the SCHC Fragment received is an All-0 SCHC Fragment, the current window is determined to be a not-last window, the receiver MUST send a SCHC ACK for this window and it MUST enter the "retransmission phase" for this window.

  * if the SCHC Fragment received is an All-1 SCHC Fragment, the padding bits of the All-1 SCHC Fragment MUST be assembled after the received tile, the current window is determined to be the last window, the receiver MUST perform the integrity check and it MUST send a SCHC ACK for this window. Then,

    + If the integrity check indicates that the full SCHC Packet has been correctly reassembled, the receiver MUST enter the "clean-up phase" for this window.

    + If the integrity check indicates that the full SCHC Packet has not been correctly reassembled, the receiver enters the "retransmission phase" for this window.

In the "retransmission phase":

- if the window is a not-last window

  * on receiving a SCHC Fragment that is not All-0 or All-1 and that has a W bit different from the local W bit, the receiver MUST increment its window counter and allocate a fresh Bitmap, it MUST assemble the tile received and update the Bitmap and it MUST enter the "acceptance phase" for that new window.

  * on receiving a SCHC ACK REQ with a W bit different from the local W bit, the receiver MUST increment its window counter and allocate a fresh Bitmap, it MUST send a SCHC ACK for that new window and it MUST enter the "acceptance phase" for that new window.

  * on receiving a SCHC All-0 Fragment with a W bit different from the local W bit, the receiver MUST increment its window counter and allocate a fresh Bitmap, it MUST assemble the tile received and update the Bitmap, it MUST send a SCHC ACK for that new

window and it MUST stay in the "retransmission phase" for that new window.

* on receiving a SCHC All-1 Fragment with a W bit different from the local W bit, the receiver MUST increment its window counter and allocate a fresh Bitmap, it MUST assemble the tile received, including the padding bits, it MUST update the Bitmap and perform the integrity check, it MUST send a SCHC ACK for the new window, which is determined to be the last window. Then,

  + If the integrity check indicates that the full SCHC Packet has been correctly reassembled, the receiver MUST enter the "clean-up phase" for that new window.

  + If the integrity check indicates that the full SCHC Packet has not been correctly reassembled, the receiver enters the "retransmission phase" for that new window.

* on receiving a SCHC Fragment with a W bit equal to the local W bit,

  + if the SCHC Fragment received is an All-1 SCHC Fragment, the receiver MUST silently ignore it and discard it.

  + otherwise, the receiver MUST assemble the tile received and update the Bitmap. If the Bitmap becomes fully populated with 1's or if the SCHC Fragment is an All-0, the receiver MUST send a SCHC ACK for this window.

* on receiving a SCHC ACK REQ with the W bit equal to the local W bit, the receiver MUST send a SCHC ACK for this window.

o if the window is the last window

* on receiving a SCHC Fragment or a SCHC ACK REQ, either one having a W bit different from the local W bit, the receiver MUST silently ignore and discard that message.

* on receiving a SCHC ACK REQ with the W bit equal to the local W bit, the receiver MUST send a SCHC ACK for this window.

* on receiving a SCHC Fragment with a W bit equal to the local W bit,

  + if the SCHC Fragment received is an All-0 SCHC Fragment, the receiver MUST silently ignore it and discard it.

+ otherwise, the receiver MUST update the Bitmap and it MUST assemble the tile received. If the SCHC Fragment received is an All-1 SCHC Fragment, the receiver MUST assemble the padding bits of the All-1 SCHC Fragment after the received tile, it MUST perform the integrity check and

  - if the integrity check indicates that the full SCHC Packet has been correctly reassembled, the receiver MUST send a SCHC ACK and it enters the "clean-up phase".

  - if the integrity check indicates that the full SCHC Packet has not been correctly reassembled,

    o if the SCHC Fragment received was an All-1 SCHC Fragment, the receiver MUST send a SCHC ACK for this window.

In the "clean-up phase":

- On receiving an All-1 SCHC Fragment or a SCHC ACK REQ, either one having the W bit equal to the local W bit, the receiver MUST send a SCHC ACK.

- Any other SCHC Fragment received MUST be silently ignored and discarded.

At any time, on sending a SCHC ACK, the receiver MUST increment the Attempts counter.

At any time, on incrementing its window counter, the receiver MUST reset the Attempts counter.

At any time, on expiration of the Inactivity Timer, on receiving a SCHC Sender-Abort or when Attempts reaches MAX_ACK_REQUESTS, the receiver MUST send a SCHC Receiver-Abort and it MAY exit the receive process for that SCHC Packet.

Figure 40 shows an example of a corresponding state machine.

8.4.3. ACK-on-Error mode

The ACK-on-Error mode supports LPWAN technologies that have variable MTU and out-of-order delivery. It operates with links that provide a feedback path from the reassembler to the fragmenter. See Appendix F for a discussion on using ACK-on-Error mode on quasi-bidirectional links.

In ACK-on-Error mode, windows are used.

All tiles, but the last one and the penultimate one, MUST be of equal size, hereafter called "regular". The size of the last tile MUST be smaller than or equal to the regular tile size. Regarding the penultimate tile, a Profile MUST pick one of the following two options:

- The penultimate tile size MUST be the regular tile size
- or the penultimate tile size MUST be either the regular tile size or the regular tile size minus one L2 Word.

A SCHC Fragment message carries one or several contiguous tiles, which may span multiple windows. A SCHC ACK reports on the reception of exactly one window of tiles.

See Figure 23 for an example.

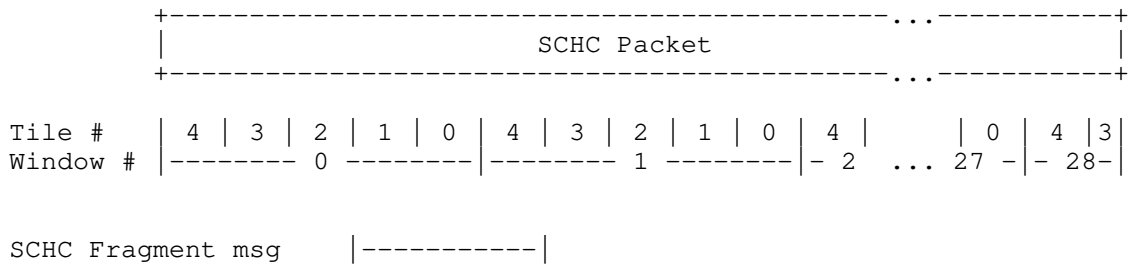


Figure 23: a SCHC Packet fragmented in tiles, ACK-on-Error mode

The W field is wide enough that it unambiguously represents an absolute window number. The fragment receiver sends SCHC ACKs to the fragment sender about windows for which tiles are missing. No SCHC ACK is sent by the fragment receiver for windows that it knows have been fully received.

The fragment sender retransmits SCHC Fragments for tiles that are reported missing. It can advance to next windows even before it has ascertained that all tiles belonging to previous windows have been correctly received, and can still later retransmit SCHC Fragments with tiles belonging to previous windows. Therefore, the sender and the receiver may operate in a decoupled fashion. The fragmented SCHC Packet transmission concludes when

- integrity checking shows that the fragmented SCHC Packet has been correctly reassembled at the receive end, and this information has been conveyed back to the sender,
- or too many retransmission attempts were made,

- or the receiver determines that the transmission of this fragmented SCHC Packet has been inactive for too long.

Each Profile MUST specify which Rule ID value(s) correspond to SCHC F/R messages operating in this mode.

The W field MUST be present in the SCHC F/R messages.

Each Profile, for each Rule ID value, MUST define

- the tile size (a tile does not need to be multiple of an L2 Word, but it MUST be at least the size of an L2 Word)
- the value of M (size of the W field),
- the value of N (size of the FCN field),
- the value of WINDOW_SIZE, which MUST be strictly less than 2^N,
- the size and algorithm for the RCS field,
- the size of the DTag field,
- the value of MAX_ACK_REQUESTS,
- the expiration time of the Retransmission Timer
- the expiration time of the Inactivity Timer
- whether the last tile is carried in a Regular SCHC Fragment or an All-1 SCHC Fragment (see Section 8.4.3.1)
- if the penultimate tile MAY be one L2 Word smaller than the regular tile size. In this case, the regular tile size MUST be at least twice the L2 Word size.

For each active pair of Rule ID and DTag values, the sender MUST maintain

- one Attempts counter
- one Retransmission Timer

For each active pair of Rule ID and DTag values, the receiver MUST maintain

- one Inactivity Timer

- one Attempts counter

8.4.3.1. Sender behavior

At the beginning of the fragmentation of a new SCHC Packet,

- the fragment sender MUST select a Rule ID and DTag value pair for this SCHC Packet. A Rule MUST NOT be selected if the values of M and WINDOW_SIZE for that Rule are such that the SCHC Packet cannot be fragmented in (2^M) * WINDOW_SIZE tiles or less.

- the fragment sender MUST initialize the Attempts counter to 0 for that Rule ID and DTag value pair.

A Regular SCHC Fragment message carries in its payload one or more tiles. If more than one tile is carried in one Regular SCHC Fragment

- the selected tiles MUST be contiguous in the original SCHC Packet

- they MUST be placed in the SCHC Fragment Payload adjacent to one another, in the order they appear in the SCHC Packet, from the start of the SCHC Packet toward its end.

Tiles that are not the last one MUST be sent in Regular SCHC Fragments specified in Section 8.3.1.1. The FCN field MUST contain the tile index of the first tile sent in that SCHC Fragment.

In a Regular SCHC Fragment message, the sender MUST fill the W field with the window number of the first tile sent in that SCHC Fragment.

Depending on the Profile, the last tile of a SCHC Packet MUST be sent either

- in a Regular SCHC Fragment, alone or as part of a multi-tiles Payload

- alone in an All-1 SCHC Fragment

In an All-1 SCHC Fragment message, the sender MUST fill the W field with the window number of the last tile of the SCHC Packet.

The fragment sender MUST send SCHC Fragments such that, all together, they contain all the tiles of the fragmented SCHC Packet.

The fragment sender MUST send at least one All-1 SCHC Fragment.

The fragment sender MUST listen for SCHC ACK messages after having sent

- an All-1 SCHC Fragment
- or a SCHC ACK REQ.

A Profile MAY specify other times at which the fragment sender MUST listen for SCHC ACK messages. For example, this could be after sending a complete window of tiles.

Each time a fragment sender sends an All-1 SCHC Fragment or a SCHC ACK REQ,

- it MUST increment the Attempts counter
- it MUST reset the Retransmission Timer

On Retransmission Timer expiration

- if Attempts is strictly less than MAX_ACK_REQUESTS, the fragment sender MUST send either the All-1 SCHC Fragment or a SCHC ACK REQ with the W field corresponding to the last window,
- otherwise the fragment sender MUST send a SCHC Sender-Abort and it MAY exit with an error condition.

All message receptions being discussed in the rest of this section are to be understood as "matching the RuleID and DTag pair being processed", even if not spelled out, for brevity.

On receiving a SCHC ACK,

- if the W field in the SCHC ACK corresponds to the last window of the SCHC Packet,
  - if the C bit is set, the sender MAY exit successfully
  - otherwise,
    - if the Profile mandates that the last tile be sent in an All-1 SCHC Fragment,
      - if the SCHC ACK shows no missing tile at the receiver, the sender
        - MUST send a SCHC Sender-Abort
        - MAY exit with an error condition
      - otherwise

- the fragment sender MUST send SCHC Fragment messages containing all the tiles that are reported missing in the SCHC ACK.

- if the last of these SCHC Fragment messages is not an All-1 SCHC Fragment, then the fragment sender MUST in addition send after it a SCHC ACK REQ with the W field corresponding to the last window.

+ otherwise,

- if the SCHC ACK shows no missing tile at the receiver, the sender MUST send the All-1 SCHC Fragment

- otherwise

- the fragment sender MUST send SCHC Fragment messages containing all the tiles that are reported missing in the SCHC ACK.

- the fragment sender MUST then send either the All-1 SCHC Fragment or a SCHC ACK REQ with the W field corresponding to the last window.

o otherwise, the fragment sender

* MUST send SCHC Fragment messages containing the tiles that are reported missing in the SCHC ACK

* then it MAY send a SCHC ACK REQ with the W field corresponding to the last window

See Figure 41 for one among several possible examples of a Finite State Machine implementing a sender behavior obeying this specification.

8.4.3.2. Receiver behavior

On receiving a SCHC Fragment with a Rule ID and DTag pair not being processed at that time

o the receiver SHOULD check if the DTag value has not recently been used for that Rule ID value, thereby ensuring that the received SCHC Fragment is not a remnant of a prior fragmented SCHC Packet transmission. The initial value of the Inactivity Timer is the RECOMMENDED lifetime for the DTag value at the receiver. If the SCHC Fragment is determined to be such a remnant, the receiver MAY silently ignore it and discard it.

- the receiver MUST start a process to assemble a new SCHC Packet with that Rule ID and DTag value pair. The receiver MUST start an Inactivity Timer for that Rule ID and DTag value pair. It MUST initialize an Attempts counter to 0 for that Rule ID and DTag value pair. If the receiver is under-resourced to do this, it MUST respond to the sender with a SCHC Receiver Abort.

On reception of any SCHC F/R message for the RuleID and DTag pair being processed, the receiver MUST reset the Inactivity Timer pertaining to that RuleID and DTag pair.

All message receptions being discussed in the rest of this section are to be understood as "matching the RuleID and DTag pair being processed", even if not spelled out, for brevity.

On receiving a SCHC Fragment message, the receiver determines what tiles were received, based on the payload length and on the W and FCN fields of the SCHC Fragment.

- if the FCN is All-1, if a Payload is present, the full SCHC Fragment Payload MUST be assembled including the padding bits. This is because the size of the last tile is not known by the receiver, therefore padding bits are indistinguishable from the tile data bits, at this stage. They will be removed by the SCHC C/D sublayer. If the size of the SCHC Fragment Payload exceeds or equals the size of one regular tile plus the size of an L2 Word, this SHOULD raise an error flag.

- otherwise, tiles MUST be assembled based on the a priori known tile size.

  * If allowed by the Profile, the end of the payload MAY contain the last tile, which may be shorter. Padding bits are indistinguishable from the tile data bits, at this stage.

  * the payload may contain the penultimate tile that, if allowed by the Profile, MAY be exactly one L2 Word shorter than the regular tile size.

  * Otherwise, padding bits MUST be discarded. The latter is possible because

    + the size of the tiles is known a priori,

    + tiles are larger than an L2 Word

    + padding bits are always strictly less than an L2 Word

On receiving a SCHC ACK REQ or an All-1 SCHC Fragment,

- if the receiver knows of any windows with missing tiles for the packet being reassembled, it MUST return a SCHC ACK for the lowest-numbered such window,
- otherwise,
  - if it has received at least one tile, it MUST return a SCHC ACK for the highest-numbered window it currently has tiles for
  - otherwise it MUST return a SCHC ACK for window numbered 0

A Profile MAY specify other times and circumstances at which a receiver sends a SCHC ACK, and which window the SCHC ACK reports about in these circumstances.

Upon sending a SCHC ACK, the receiver MUST increase the Attempts counter.

After receiving an All-1 SCHC Fragment, a receiver MUST check the integrity of the reassembled SCHC Packet at least every time it prepares for sending a SCHC ACK for the last window.

Upon receiving a SCHC Sender-Abort, the receiver MAY exit with an error condition.

Upon expiration of the Inactivity Timer, the receiver MUST send a SCHC Receiver-Abort and it MAY exit with an error condition.

On the Attempts counter exceeding MAX_ACK_REQUESTS, the receiver MUST send a SCHC Receiver-Abort and it MAY exit with an error condition.

Reassembly of the SCHC Packet concludes when

- a Sender-Abort has been received
- or the Inactivity Timer has expired
- or the Attempts counter has exceeded MAX_ACK_REQUESTS
- or when at least an All-1 SCHC Fragment has been received and integrity checking of the reassembled SCHC Packet is successful.

See Figure 42 for one among several possible examples of a Finite State Machine implementing a receiver behavior obeying this specification, and that is meant to match the sender Finite State Machine of Figure 41.

9. Padding management

SCHC C/D and SCHC F/R operate on bits, not bytes. SCHC itself does not have any alignment prerequisite. The size of SCHC Packets can be any number of bits.

If the layer below SCHC constrains the payload to align to some boundary, called L2 Words (for example, bytes), the SCHC messages MUST be padded. When padding occurs, the number of appended bits MUST be strictly less than the L2 Word size.

If a SCHC Packet is sent unfragmented (see Figure 24), it is padded as needed for transmission.

If a SCHC Packet needs to be fragmented for transmission, it is not padded in itself. Only the SCHC F/R messages are padded as needed for transmission. Some SCHC F/R messages are intrinsically aligned to L2 Words.

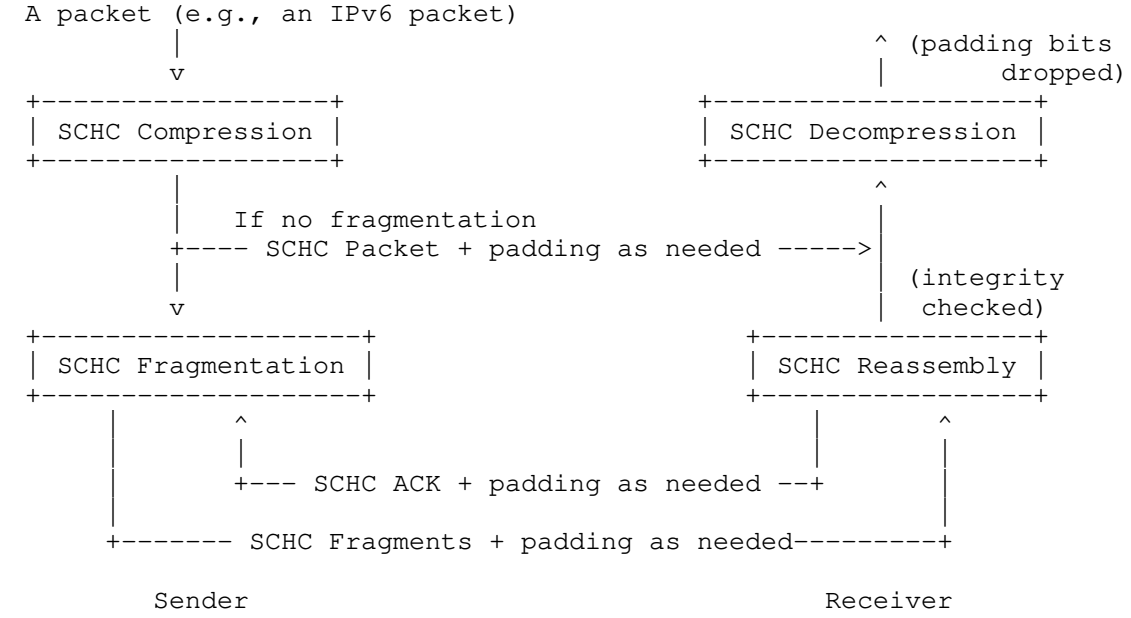


Figure 24: SCHC operations, including padding as needed

Each Profile MUST specify the size of the L2 Word. The L2 Word might actually be a single bit, in which case no padding will take place at all.

A Profile MAY define the value of the padding bits. The RECOMMENDED value is 0.

10. SCHC Compression for IPv6 and UDP headers

This section lists the IPv6 and UDP header fields and describes how they can be compressed. An example of a set of Rules for UDP/IPv6 header compression is provided in Appendix A.

10.1. IPv6 version field

The IPv6 version field is labeled by the protocol parser as being the "version" field of the IPv6 protocol. Therefore, it only exists for IPv6 packets. In the Rule, TV is set to 6, MO to "ignore" and CDA to "not-sent".

10.2. IPv6 Traffic class field

If the DiffServ field does not vary and is known by both sides, the Field Descriptor in the Rule SHOULD contain a TV with this well-known value, an "equal" MO and a "not-sent" CDA.

Otherwise (e.g., ECN bits are to be transmitted), two possibilities can be considered depending on the variability of the value:

- One possibility is to not compress the field and send the original value. In the Rule, TV is not set to any particular value, MO is set to "ignore" and CDA is set to "value-sent".

- If some upper bits in the field are constant and known, a better option is to only send the LSBs. In the Rule, TV is set to a value with the stable known upper part, MO is set to MSB(x) and CDA to LSB.

  ECN functionality depends on both bits of the ECN field, which are the 2 LSBs of this field, hence sending only a single LSB of this field is NOT RECOMMENDED.

10.3. Flow label field

If the flow label is not set, i.e. its value is zero, the Field Descriptor in the Rule SHOULD contain a TV set to zero, an "equal" MO and a "not-sent" CDA.

If the flow label is set to a pseudo-random value according to [RFC6437], in the Rule, TV is not set to any particular value, MO is set to "ignore" and CDA is set to "value-sent".

If the flow label is set according to some prior agreement, i.e. by a flow state establishment method as allowed by [RFC6437], the Field Descriptor in the Rule SHOULD contain a TV with this agreed-upon value, an "equal" MO and a "not-sent" CDA.

10.4. Payload Length field

This field can be elided for the transmission on the LPWAN network. The SCHC C/D recomputes the original payload length value. In the Field Descriptor, TV is not set, MO is set to "ignore" and CDA is "compute-*".

10.5. Next Header field

If the Next Header field does not vary and is known by both sides, the Field Descriptor in the Rule SHOULD contain a TV with this Next Header value, the MO SHOULD be "equal" and the CDA SHOULD be "not-sent".

Otherwise, TV is not set in the Field Descriptor, MO is set to "ignore" and CDA is set to "value-sent". Alternatively, a matching-list MAY also be used.

10.6. Hop Limit field

The field behavior for this field is different for uplink (Up) and downlink (Dw). In Up, since there is no IP forwarding between the Dev and the SCHC C/D, the value is relatively constant. On the other hand, the Dw value depends on Internet routing and can change more frequently. The Direction Indicator (DI) can be used to distinguish both directions:

- in the Up, elide the field: the TV in the Field Descriptor is set to the known constant value, the MO is set to "equal" and the CDA is set to "not-sent".

- in the Dw, the Hop Limit is elided for transmission and forced to 1 at the receiver, by setting TV to 1, MO to "ignore" and CDA to "not-sent". This prevents any further forwarding.

10.7. IPv6 addresses fields

As in 6LoWPAN [RFC4944], IPv6 addresses are split into two 64-bit long fields; one for the prefix and one for the Interface Identifier (IID). These fields SHOULD be compressed. To allow for a single Rule being used for both directions, these values are identified by their role (Dev or App) and not by their position in the header (source or destination).

10.7.1. IPv6 source and destination prefixes

Both ends MUST be configured with the appropriate prefixes. For a specific flow, the source and destination prefixes can be unique and stored in the Context. In that case, the TV for the source and destination prefixes contain the values, the MO is set to "equal" and the CDA is set to "not-sent".

If the Rule is intended to compress packets with different prefix values, match-mapping SHOULD be used. The different prefixes are listed in the TV, the MO is set to "match-mapping" and the CDA is set to "mapping-sent". See Figure 26.

Otherwise, the TV is not set, the MO is set to "ignore" and the CDA is set to "value-sent".

10.7.2. IPv6 source and destination IID

If the Dev or App IID are based on an LPWAN address, then the IID can be reconstructed with information coming from the LPWAN header. In that case, the TV is not set, the MO is set to "ignore" and the CDA is set to "DevIID" or "AppIID". On LPWAN technologies where the frames carry a single identifier (corresponding to the Dev.), AppIID cannot be used.

As described in [RFC8065], it may be undesirable to build the Dev IPv6 IID out of the Dev address. Another static value is used instead. In that case, the TV contains the static value, the MO operator is set to "equal" and the CDA is set to "not-sent".

If several IIDs are possible, then the TV contains the list of possible IIDs, the MO is set to "match-mapping" and the CDA is set to "mapping-sent".

It may also happen that the IID variability only expresses itself on a few bytes. In that case, the TV is set to the stable part of the IID, the MO is set to "MSB" and the CDA is set to "LSB".

Finally, the IID can be sent in its entirety on the LPWAN. In that case, the TV is not set, the MO is set to "ignore" and the CDA is set to "value-sent".

10.8. IPv6 extension headers

This document does not provide recommendations on how to compress IPv6 extension headers.

10.9. UDP source and destination ports

To allow for a single Rule being used for both directions, the UDP port values are identified by their role (Dev or App) and not by their position in the header (source or destination). The SCHC C/D MUST be aware of the traffic direction (Uplink, Downlink) to select the appropriate field. The following Rules apply for Dev and App port numbers.

If both ends know the port number, it can be elided. The TV contains the port number, the MO is set to "equal" and the CDA is set to "not-sent".

If the port variation is on few bits, the TV contains the stable part of the port number, the MO is set to "MSB" and the CDA is set to "LSB".

If some well-known values are used, the TV can contain the list of these values, the MO is set to "match-mapping" and the CDA is set to "mapping-sent".

Otherwise the port numbers are sent over the LPWAN. The TV is not set, the MO is set to "ignore" and the CDA is set to "value-sent".

10.10. UDP length field

The UDP length can be computed from the received data. The TV is not set, the MO is set to "ignore" and the CDA is set to "compute-*".

10.11. UDP Checksum field

The UDP checksum operation is mandatory with IPv6 for most packets but there are exceptions [RFC8200].

For instance, protocols that use UDP as a tunnel encapsulation may enable zero-checksum mode for a specific port (or set of ports) for sending and/or receiving. [RFC8200] requires any node implementing zero-checksum mode to follow the requirements specified in "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums" [RFC6936].

6LoWPAN Header Compression [RFC6282] also specifies that a UDP checksum can be elided by the compressor and re-computed by the decompressor when an upper layer guarantees the integrity of the UDP payload and pseudo-header. A specific example of this is when a message integrity check protects the compressed message between the compressor that elides the UDP checksum and the decompressor that

computes it, with a strength that is identical or better to the UDP checksum.

Similarly, a SCHC compressor MAY elide the UDP checksum when another layer guarantees at least equal integrity protection for the UDP payload and the pseudo-header. In this case, the TV is not set, the MO is set to "ignore" and the CDA is set to "compute-*".

In particular, when SCHC fragmentation is used, a fragmentation RCS of 2 bytes or more provides equal or better protection than the UDP checksum; in that case, if the compressor is collocated with the fragmentation point and the decompressor is collocated with the packet reassembly point, and if the SCHC Packet is fragmented even when it would fit unfragmented in the L2 MTU, then the compressor MAY verify and then elide the UDP checksum. Whether and when the UDP Checksum is elided is to be specified in the Profile.

Since the compression happens before the fragmentation, implementers should understand the risks when dealing with unprotected data below the transport layer and take special care when manipulating that data.

In other cases, the checksum SHOULD be explicitly sent. The TV is not set, the MO is set to "ignore" and the CDA is set to "value-sent".

11. IANA Considerations

This document has no request to IANA.

12. Security considerations

As explained in Section 5, SCHC is expected to be implemented on top of LPWAN technologies, which are expected to implement security measures.

In this section, we analyze the potential security threats that could be introduced into an LPWAN by adding the SCHC functionalities.

12.1. Security considerations for SCHC Compression/Decompression

12.1.1. Forged SCHC Packet

Let’s assume that an attacker is able to send a forged SCHC Packet to a SCHC Decompressor.

Let’s first consider the case where the Rule ID contained in that forged SCHC Packet does not correspond to a Rule allocated in the

Rule table. An implementation should detect that the Rule ID is invalid and should silently drop the offending SCHC Packet.

Let's now consider that the Rule ID corresponds to a Rule in the table. With the CDAs defined in this document, the reconstructed packet is at most a constant number of bits bigger than the SCHC Packet that was received. This assumes that the compute-* decompression actions produce a bounded number of bits, irrespective of the incoming SCHC Packet. This property is true for IPv6 Length, UDP Length and UDP Checksum, for which the compute-* CDA is recommended by this document.

As a consequence, SCHC Decompression does not amplify attacks, beyond adding a bounded number of bits to the SCHC Packet received. This bound is determined by the Rule stored in the receiving device.

As a general safety measure, a SCHC Decompressor should never reconstruct a packet larger than MAX_PACKET_SIZE (defined in a Profile, with 1500 bytes as generic default).

12.1.2. Compressed packet size as a side channel to guess a secret token

Some packet compression methods are known to be susceptible to attacks, such as BREACH and CRIME. The attack involves injecting arbitrary data into the packet and observing the resulting compressed packet size. The observed size potentially reflects correlation between the arbitrary data and some content that was meant to remain secret, such as a security token, thereby allowing the attacker to get at the secret.

By contrast, SCHC Compression takes place header field by header field, with the SCHC Packet being a mere concatenation of the compression residues of each of the individual field. Any correlation between header fields does not result in a change in the SCHC Packet size compressed under the same Rule.

If SCHC C/D is used to compress packets that include a secret information field, such as a token, the Rule set should be designed so that the size of the compression residue for the field to remain secret is the same irrespective of the value of the secret information. This is achieved by e.g., sending this field in extenso with the "ignore" MO and the "value-sent" CDA. This recommendation is disputable if it is ascertained that the Rule set itself will remain secret.

12.1.3. Decompressed packet different from the original packet

As explained in Section 7.3, using FPs with value 0 in Field Descriptors in a Rule may result in header fields appearing in the decompressed packet in an order different from that in the original packet. Likewise, as stated in Section 7.5.3, using an "ignore" MO together with a "not-sent" CDA will result in the header field taking the TV value, which is likely to be different from the original value.

Depending on the protocol, the order of header fields in the packet may be functionally significant or not.

Furthermore, if the packet is protected by a checksum or a similar integrity protection mechanism, and if the checksum is transmitted instead of being recomputed as part of the decompression, these situations may result in the packet being considered corrupt and dropped.

12.2. Security considerations for SCHC Fragmentation/Reassembly

12.2.1. Buffer reservation attack

Let’s assume that an attacker is able to send a forged SCHC Fragment to a SCHC Reassembler.

A node can perform a buffer reservation attack: the receiver will reserve buffer space for the SCHC Packet. If the implementation has only one buffer, other incoming fragmented SCHC Packets will be dropped while the reassembly buffer is occupied during the reassembly timeout. Once that timeout expires, the attacker can repeat the same procedure, and iterate, thus creating a denial of service attack. An implementation may have multiple reassembly buffers. The cost to mount this attack is linear with the number of buffers at the target node. Better, the cost for an attacker can be increased if individual fragments of multiple SCHC Packets can be stored in the reassembly buffer. The finer grained the reassembly buffer (downto the smallest tile size), the higher the cost of the attack. If buffer overload does occur, a smart receiver could selectively discard SCHC Packets being reassembled based on the sender behavior, which may help identify which SCHC Fragments have been sent by the attacker. Another mild counter-measure is for the target to abort the fragmentation/reassembly session as early as it detects a non-identical SCHC Fragment duplicate, anticipating for an eventual corrupt SCHC Packet, so as to save the sender the hassle of sending the rest of the fragments for this SCHC Packet.

12.2.2. Corrupt Fragment attack

Let's assume that an attacker is able to send a forged SCHC Fragment to a SCHC Reassembler. The malicious node is additionally assumed to be able to hear an incoming communication destined to the target node.

It can then send a forged SCHC Fragment that looks like it belongs to a SCHC Packet already being reassembled at the target node. This can cause the SCHC Packet to be considered corrupt and be dropped by the receiver. The amplification happens here by a single spoofed SCHC Fragment rendering a full sequence of legit SCHC Fragments useless. If the target uses ACK-Always or ACK-on-Error mode, such a malicious node can also interfere with the acknowledgement and repetition algorithm of SCHC F/R. A single spoofed ACK, with all bitmap bits set to 0, will trigger the repetition of WINDOW_SIZE tiles. This protocol loop amplification depletes the energy source of the target node and consumes the channel bandwidth. Similarly, a spoofed ACK REQ will trigger the sending of a SCHC ACK, which may be much larger than the ACK REQ if WINDOW_SIZE is large. These consequences should be borne in mind when defining profiles for SCHC over specific LPWAN technologies.

12.2.3. Fragmentation as a way to bypass Network Inspection

Fragmentation is known for potentially allowing to force through a Network Inspection device (e.g., firewall) packets that would be rejected if unfragmented. This involves sending overlapping fragments to rewrite fields whose initial value led the Network Inspection device to allow the flow go through.

SCHC F/R is expected to be used over one LPWAN link, where no Network Inspection device is expected to sit. As described in Section 5.2, even if the SCHC F/R on the Network infrastructure side is located in the Internet, a tunnel is to be established between it and the NGW.

12.2.4. Privacy issues associated with SCHC header fields

SCHC F/R allocates a DTag value to fragments belonging to the same SCHC Packet. Concerns were raised that, if DTag is a wide counter that is incremented in a predictable fashion for each new fragmented SCHC Packet, it might lead to a privacy issue, such as enabling tracking of a device across LPWANs.

However, SCHC F/R is expected to be used over exactly one LPWAN link. As described in Section 5.2, even if the SCHC F/R on the Network infrastructure side is located in the Internet, a tunnel is to be established between it and the NGW. Therefore, assuming the tunnel

provides confidentiality, neither the DTag field nor any other SCHC-introduced field is visible over the Internet.

13. Acknowledgements

Thanks to (in alphabetical order) Sergio Aguilar Romero, David Black, Carsten Bormann, Deborah Brungard, Brian Carpenter, Philippe Clavier, Alissa Cooper, Roman Danyliw, Daniel Ducuara Beltran, Diego Dujovne, Eduardo Ingles Sanchez, Rahul Jadhav, Benjamin Kaduk, Arunprabhu Kandasamy, Suresh Krishnan, Mirja Kuehlewind, Barry Leiba, Sergio Lopez Bernal, Antoni Markovski, Alexey Melnikov, Georgios Papadopoulos, Alexander Pelov, Charles Perkins, Edgar Ramos, Alvaro Retana, Adam Roach, Shoichi Sakane, Joseph Salowey, Pascal Thubert, and Eric Vyncke for useful design considerations, reviews and comments.

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Educacion, Cultura y Deporte) through the Jose Castillejo grant CAS15/00336, and by the ERDF and the Spanish Government through project TEC2016-79988-P. Part of his contribution to this work has been carried out during his stay as a visiting scholar at the Computer Laboratory of the University of Cambridge.

14. References

14.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, DOI 10.17487/RFC6936, April 2013, <https://www.rfc-editor.org/info/rfc6936>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[RFC8376] Farrell, S., Ed., "Low-Power Wide Area Network (LPWAN) Overview", RFC 8376, DOI 10.17487/RFC8376, May 2018, <https://www.rfc-editor.org/info/rfc8376>.

14.2. Informative References

[ETHERNET] "IEEE Standard for Ethernet", IEEE standard, DOI 10.1109/ieeestd.2018.8457469, n.d..

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.

[RFC5795] Sandlund, K., Pelletier, G., and L-E. Jonsson, "The RObust Header Compression (ROHC) Framework", RFC 5795, DOI 10.17487/RFC5795, March 2010, <https://www.rfc-editor.org/info/rfc5795>.

[RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <https://www.rfc-editor.org/info/rfc6282>.

[RFC6437] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme, "IPv6 Flow Label Specification", RFC 6437, DOI 10.17487/RFC6437, November 2011, <https://www.rfc-editor.org/info/rfc6437>.

[RFC7136] Carpenter, B. and S. Jiang, "Significance of IPv6 Interface Identifiers", RFC 7136, DOI 10.17487/RFC7136, February 2014, <https://www.rfc-editor.org/info/rfc7136>.

[RFC8065] Thaler, D., "Privacy Considerations for IPv6 Adaptation-Layer Mechanisms", RFC 8065, DOI 10.17487/RFC8065, February 2017, <https://www.rfc-editor.org/info/rfc8065>.

Appendix A. Compression Examples

This section gives some scenarios of the compression mechanism for IPv6/UDP. The goal is to illustrate the behavior of SCHC.

The mechanisms defined in this document can be applied to a Dev that embeds some applications running over CoAP. In this example, three flows are considered. The first flow is for the device management based on CoAP using Link Local IPv6 addresses and UDP ports 123 and 124 for Dev and App, respectively. The second flow will be a CoAP

server for measurements done by the Dev (using ports 5683) and Global IPv6 Address prefixes alpha::IID/64 to beta::1/64. The last flow is for legacy applications using different ports numbers, the destination IPv6 address prefix is gamma::1/64.

Figure 25 presents the protocol stack. IPv6 and UDP are represented with dotted lines since these protocols are compressed on the radio link.

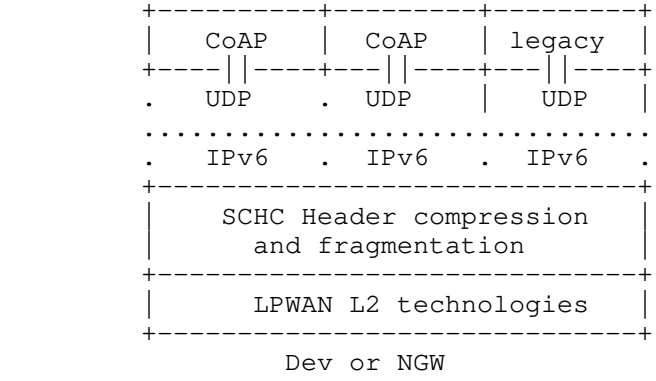


Figure 25: Simplified Protocol Stack for LP-WAN

In some LPWAN technologies, only the Devs have a device ID. When such technologies are used, it is necessary to statically define an IID for the Link Local address for the SCHC C/D.

Rule 0
Special Rule ID used to tag an uncompressed UDP/IPV6 packet.

Rule 1

Field	FL	FP	DI	Value	Match Opera.	Comp Decomp Action	Sent [bits]
IPv6 Version	4	1	Bi	6	ignore	not-sent	
IPv6 DiffServ	8	1	Bi	0	equal	not-sent	
IPv6 Flow Label	20	1	Bi	0	equal	not-sent	
IPv6 Length	16	1	Bi		ignore	compute-*	
IPv6 Next Header	8	1	Bi	17	equal	not-sent	
IPv6 Hop Limit	8	1	Bi	255	ignore	not-sent	
IPv6 DevPrefix	64	1	Bi	FE80::/64	equal	not-sent	
IPv6 DevIID	64	1	Bi		ignore	DevIID	
IPv6 AppPrefix	64	1	Bi	FE80::/64	equal	not-sent	

IPv6 AppIID	64	1	Bi	::1	equal	not-sent		
UDP DevPort	16	1	Bi	123	equal	not-sent		
UDP AppPort	16	1	Bi	124	equal	not-sent		
UDP Length	16	1	Bi		ignore	compute-*		
UDP checksum	16	1	Bi		ignore	compute-*		

Rule 2

Field	FL	FP	DI	Value	Match Opera.	Action Action		Sent [bits]
IPv6 Version	4	1	Bi	6	ignore	not-sent		
IPv6 DiffServ	8	1	Bi	0	equal	not-sent		
IPv6 Flow Label	20	1	Bi	0	equal	not-sent		
IPv6 Length	16	1	Bi		ignore	compute-*		
IPv6 Next Header	8	1	Bi	17	equal	not-sent		
IPv6 Hop Limit	8	1	Bi	255	ignore	not-sent		
IPv6 DevPrefix	64	1	Bi	[alpha/64, fe80::/64]	match-mapping	mapping-sent		1
IPv6 DevIID	64	1	Bi		ignore	DevIID		
IPv6 AppPrefix	64	1	Bi	[beta/64, alpha/64, fe80::64]	match-mapping	mapping-sent		2
IPv6 AppIID	64	1	Bi	::1000	equal	not-sent		
UDP DevPort	16	1	Bi	5683	equal	not-sent		
UDP AppPort	16	1	Bi	5683	equal	not-sent		
UDP Length	16	1	Bi		ignore	compute-*		
UDP checksum	16	1	Bi		ignore	compute-*		

Rule 3

Field	FL	FP	DI	Value	Match Opera.	Action Action		Sent [bits]
IPv6 Version	4	1	Bi	6	ignore	not-sent		
IPv6 DiffServ	8	1	Bi	0	equal	not-sent		
IPv6 Flow Label	20	1	Bi	0	equal	not-sent		
IPv6 Length	16	1	Bi		ignore	compute-*		
IPv6 Next Header	8	1	Bi	17	equal	not-sent		
IPv6 Hop Limit	8	1	Up	255	ignore	not-sent		
IPv6 Hop Limit	8	1	Dw		ignore	value-sent		8
IPv6 DevPrefix	64	1	Bi	alpha/64	equal	not-sent		
IPv6 DevIID	64	1	Bi		ignore	DevIID		
IPv6 AppPrefix	64	1	Bi	gamma/64	equal	not-sent		

IPv6 AppIID	64	1	Bi	::1000	equal	not-sent		
UDP DevPort	16	1	Bi	8720	MSB(12)	LSB		4
UDP AppPort	16	1	Bi	8720	MSB(12)	LSB		4
UDP Length	16	1	Bi		ignore	compute-*		
UDP checksum	16	1	Bi		ignore	compute-*		

Figure 26: Context Rules

Figure 26 describes a example of a Rule set.

In this example, 0 was chosen as the special Rule ID that tags packets that cannot be compressed with any compression Rule.

All the fields described in Rules 1-3 are present in the IPv6 and UDP headers. The DevIID-DID value is found in the L2 header.

Rules 2-3 use global addresses. The way the Dev learns the prefix is not in the scope of the document.

Rule 3 compresses each port number to 4 bits.

Appendix B. Fragmentation Examples

This section provides examples for the various fragment reliability modes specified in this document. In the drawings, Bitmaps are shown in their uncompressed form.

Figure 27 illustrates the transmission in No-ACK mode of a SCHC Packet that needs 11 SCHC Fragments. FCN is 1 bit wide.

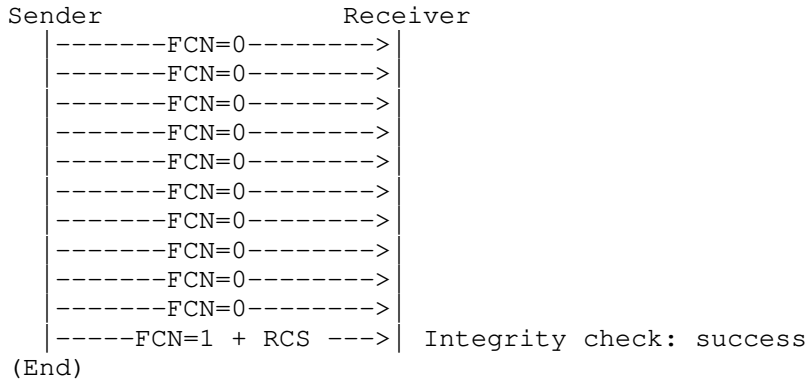


Figure 27: No-ACK mode, 11 SCHC Fragments

In the following examples, N (the size of the FCN field) is 3 bits. The All-1 FCN value is 7.

Figure 28 illustrates the transmission in ACK-on-Error mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment, WINDOW_SIZE=7 and no lost SCHC Fragment.

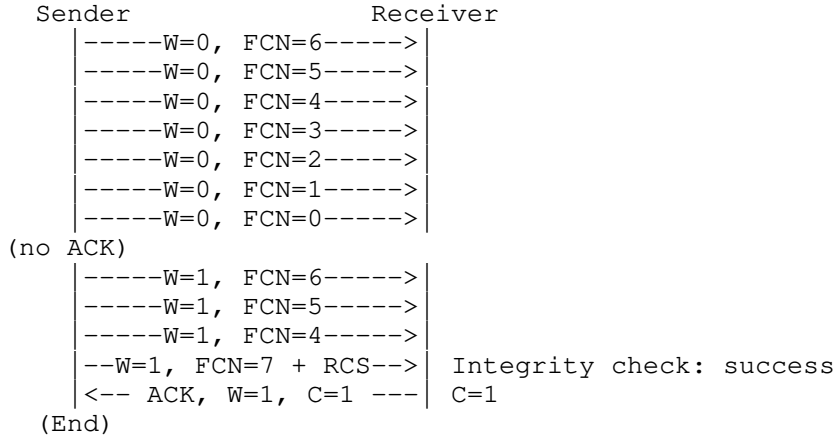


Figure 28: ACK-on-Error mode, 11 tiles, one tile per SCHC Fragment, no lost SCHC Fragment.

Figure 29 illustrates the transmission in ACK-on-Error mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment, WINDOW_SIZE=7 and three lost SCHC Fragments.

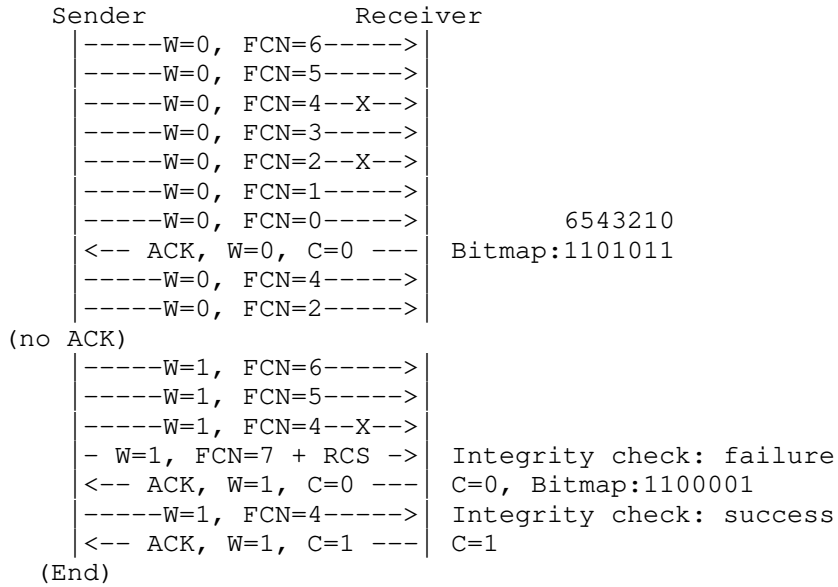


Figure 29: ACK-on-Error mode, 11 tiles, one tile per SCHC Fragment, lost SCHC Fragments.

Figure 30 shows an example of a transmission in ACK-on-Error mode of a SCHC Packet fragmented in 73 tiles, with N=5, WINDOW_SIZE=28, M=2 and 3 lost SCHC Fragments.

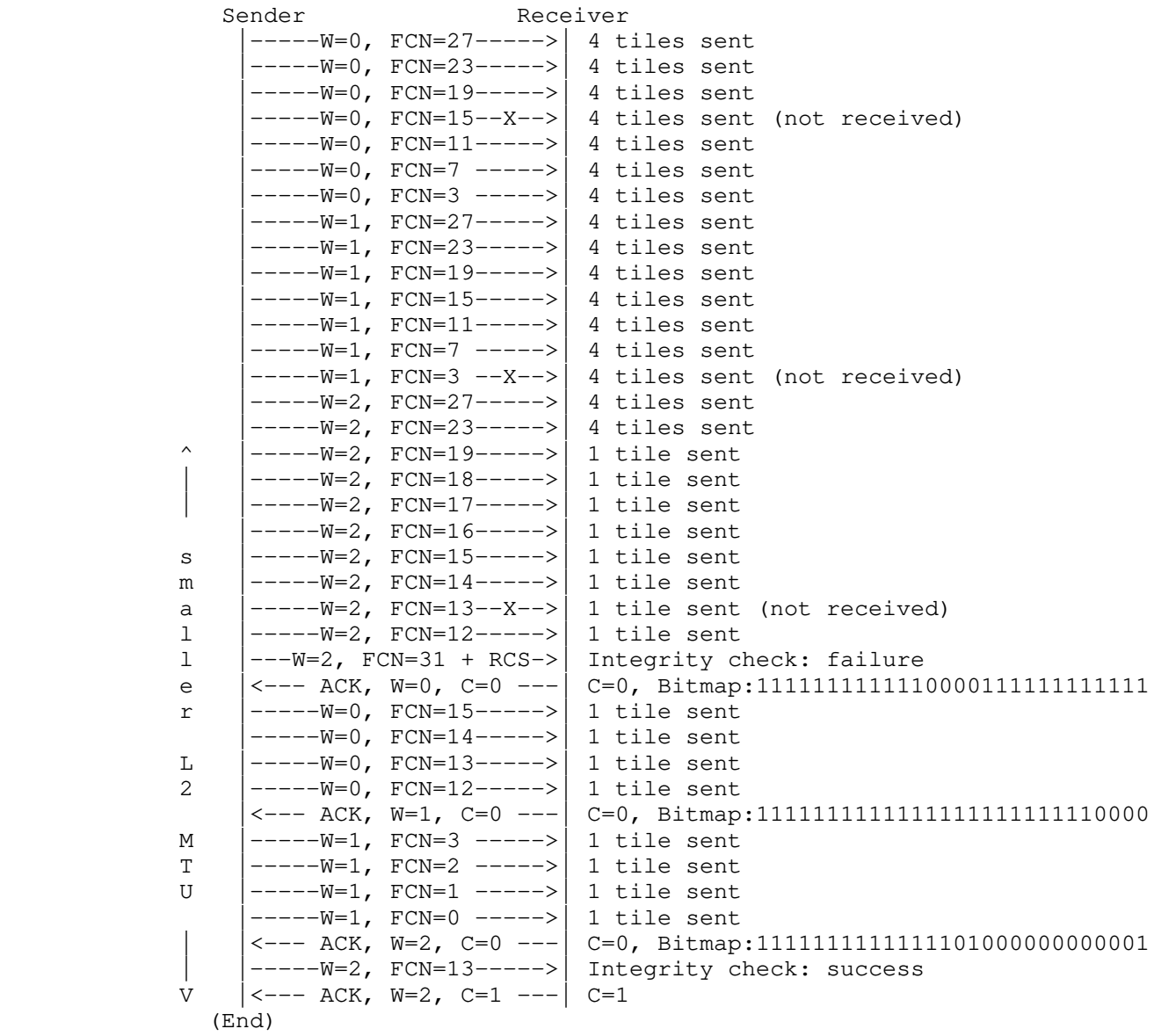


Figure 30: ACK-on-Error mode, variable MTU.

In this example, the L2 MTU becomes reduced just before sending the "W=2, FCN=19" fragment, leaving space for only 1 tile in each forthcoming SCHC Fragment. Before retransmissions, the 73 tiles are carried by a total of 25 SCHC Fragments, the last 9 being of smaller size.

Note: other sequences of events (e.g., regarding when ACKs are sent by the Receiver) are also allowed by this specification. Profiles may restrict this flexibility.

Figure 31 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment, with N=3, WINDOW_SIZE=7 and no loss.

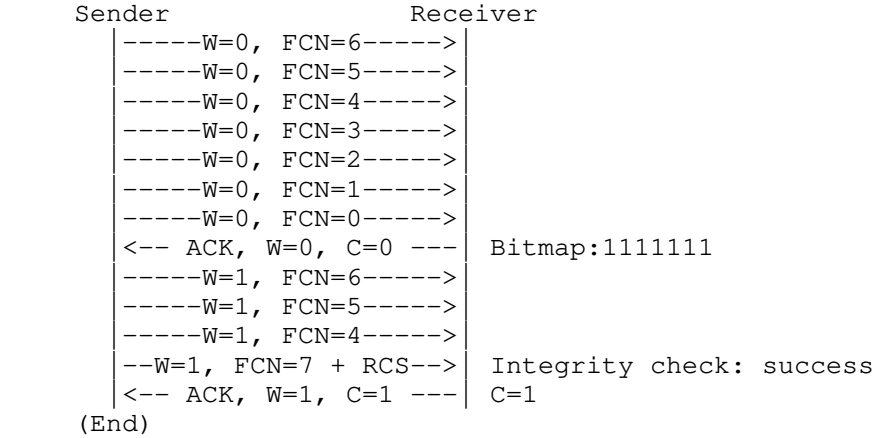


Figure 31: ACK-Always mode, 11 tiles, one tile per SCHC Fragment, no loss.

Figure 32 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment, N=3, WINDOW_SIZE=7 and three lost SCHC Fragments.

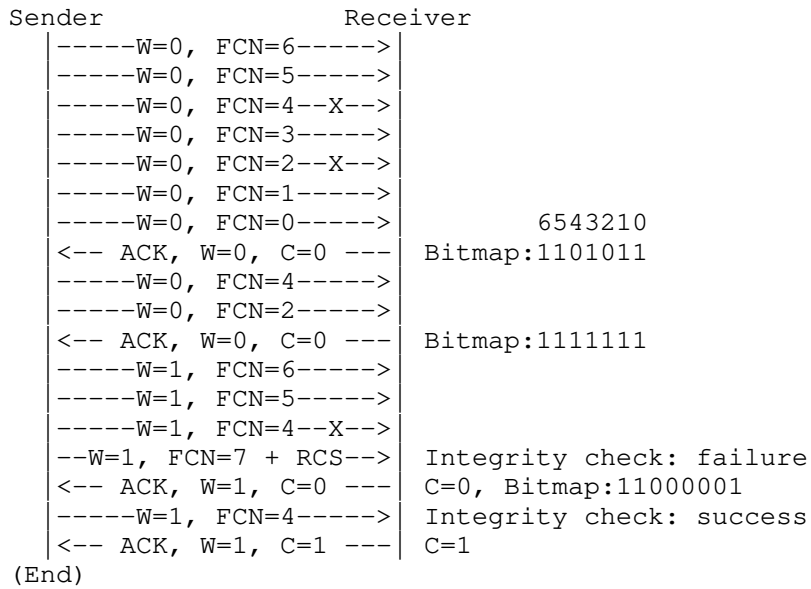


Figure 32: ACK-Always mode, 11 tiles, one tile per SCHC Fragment, three lost SCHC Fragments.

Figure 33 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 6 tiles, with one tile per SCHC Fragment, N=3, WINDOW_SIZE=7, three lost SCHC Fragments and only one retry needed to recover each lost SCHC Fragment.

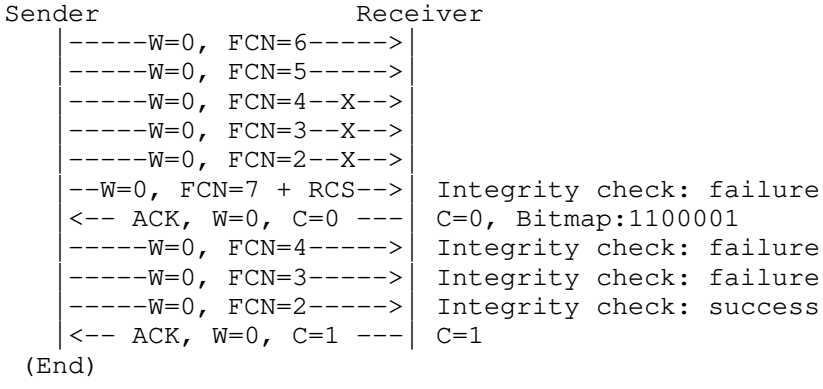


Figure 33: ACK-Always mode, 6 tiles, one tile per SCHC Fragment, three lost SCHC Fragments.

Figure 34 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 6 tiles, with one tile per SCHC Fragment, N=3,

WINDOW_SIZE=7, three lost SCHC Fragments, and the second SCHC ACK lost.

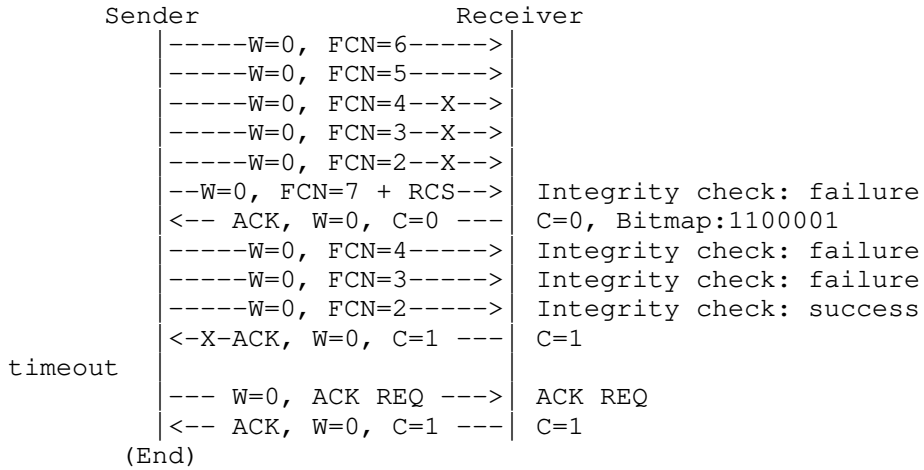


Figure 34: ACK-Always mode, 6 tiles, one tile per SCHC Fragment, SCHC ACK loss.

Figure 35 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 6 tiles, with N=3, WINDOW_SIZE=7, with three lost SCHC Fragments, and one retransmitted SCHC Fragment lost again.

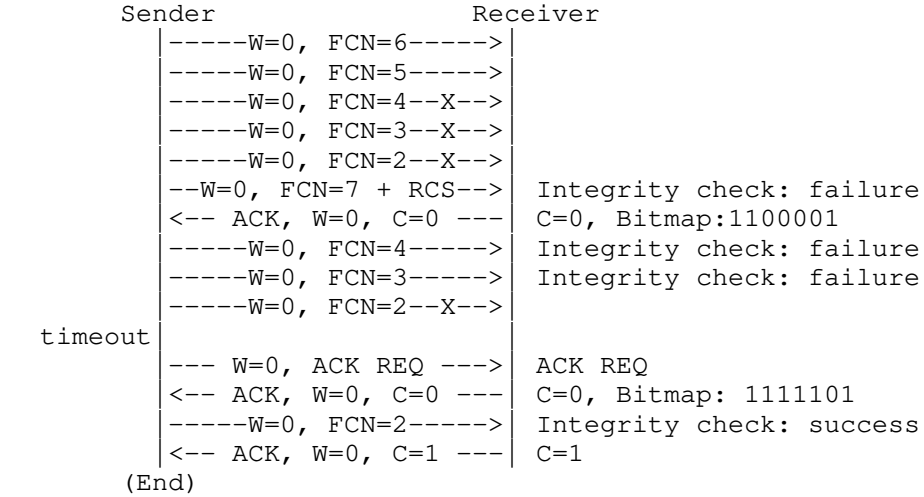


Figure 35: ACK-Always mode, 6 tiles, retransmitted SCHC Fragment lost again.

Figure 36 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 28 tiles, with one tile per SCHC Fragment, N=5, WINDOW_SIZE=24 and two lost SCHC Fragments.

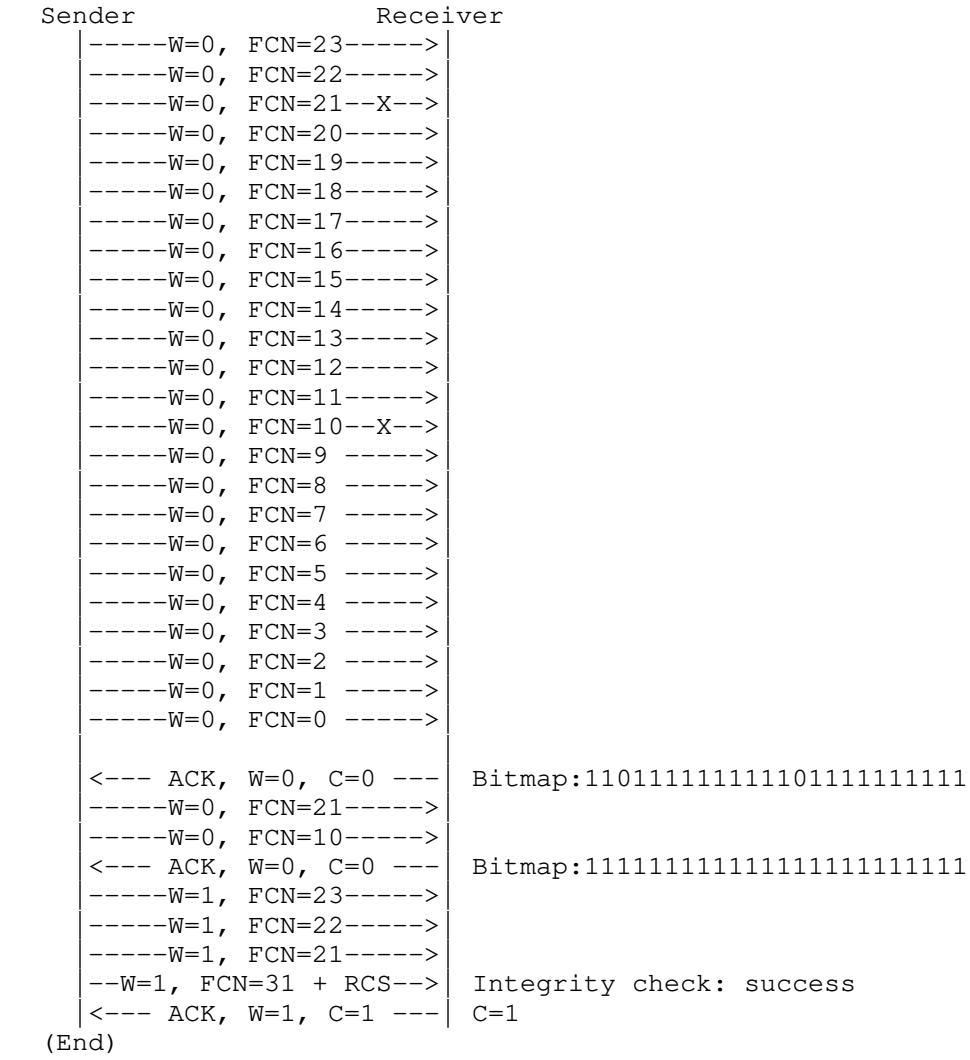


Figure 36: ACK-Always mode, 28 tiles, one tile per SCHC Fragment, lost SCHC Fragments.

Appendix C. Fragmentation State Machines

The fragmentation state machines of the sender and the receiver, one for each of the different reliability modes, are described in the following figures:

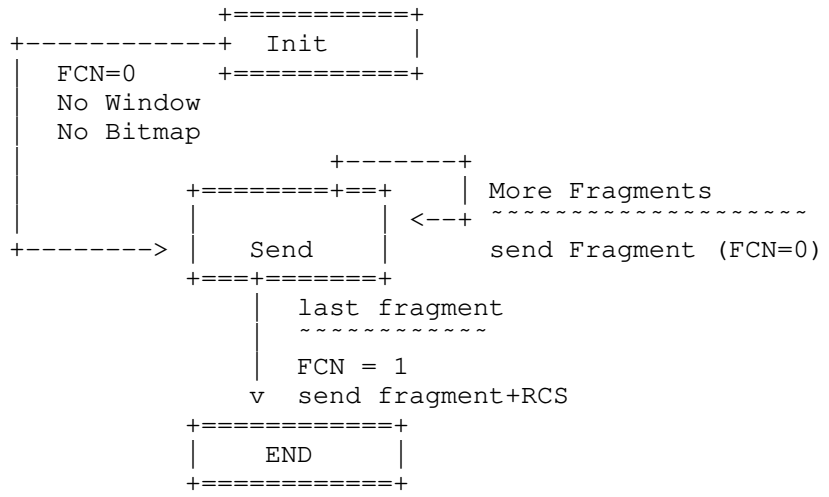


Figure 37: Sender State Machine for the No-ACK Mode

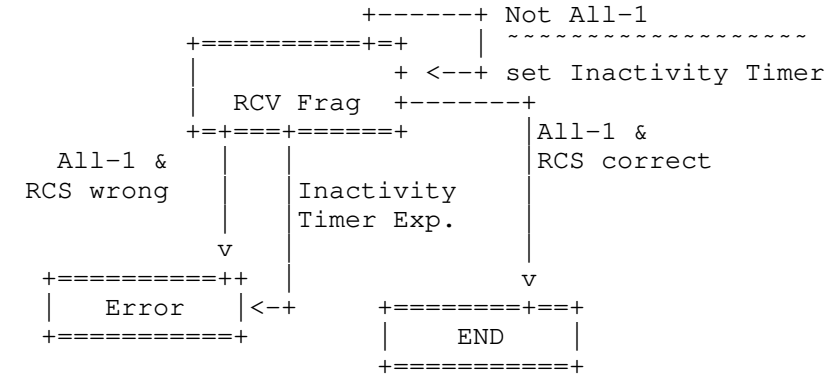


Figure 38: Receiver State Machine for the No-ACK Mode

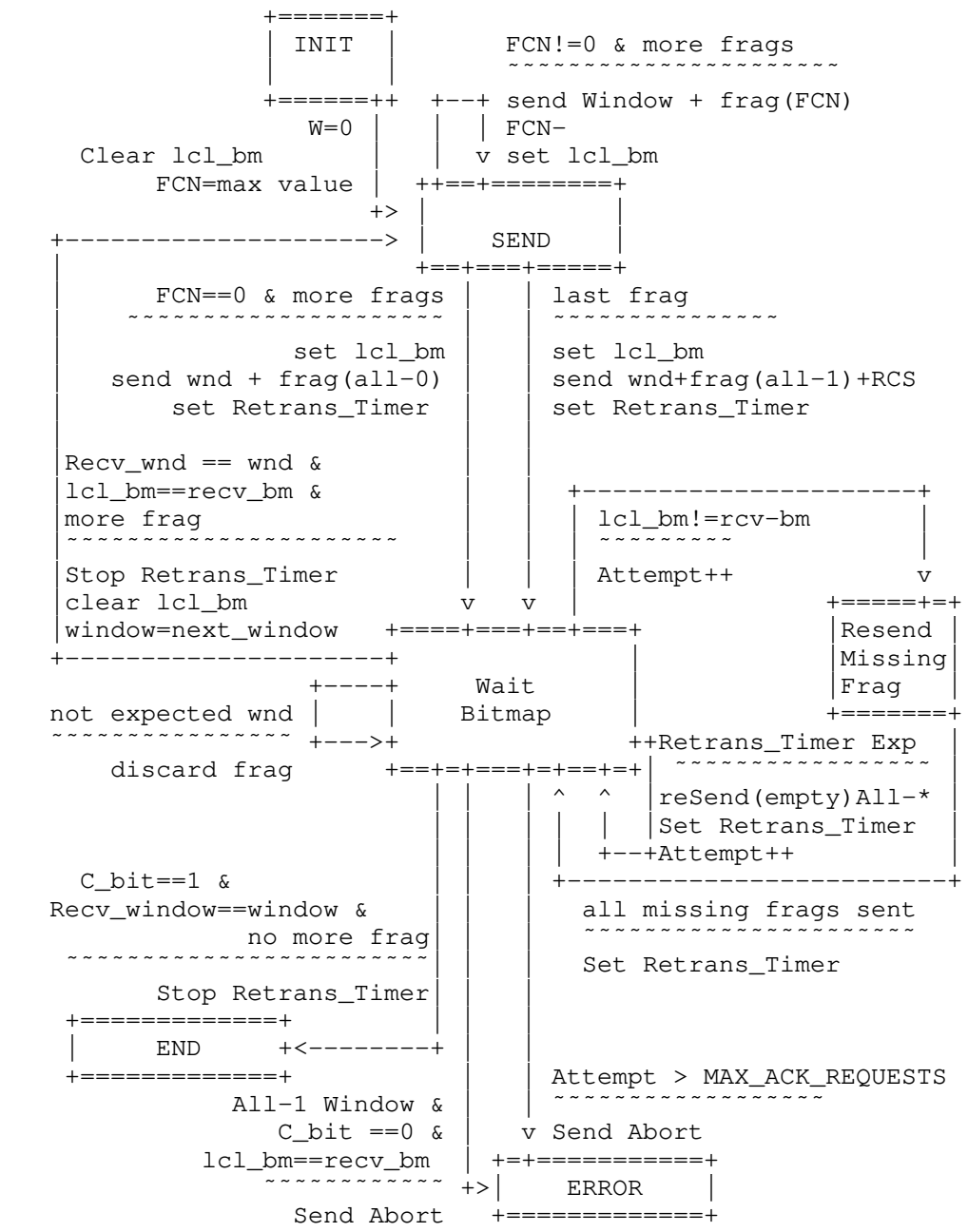
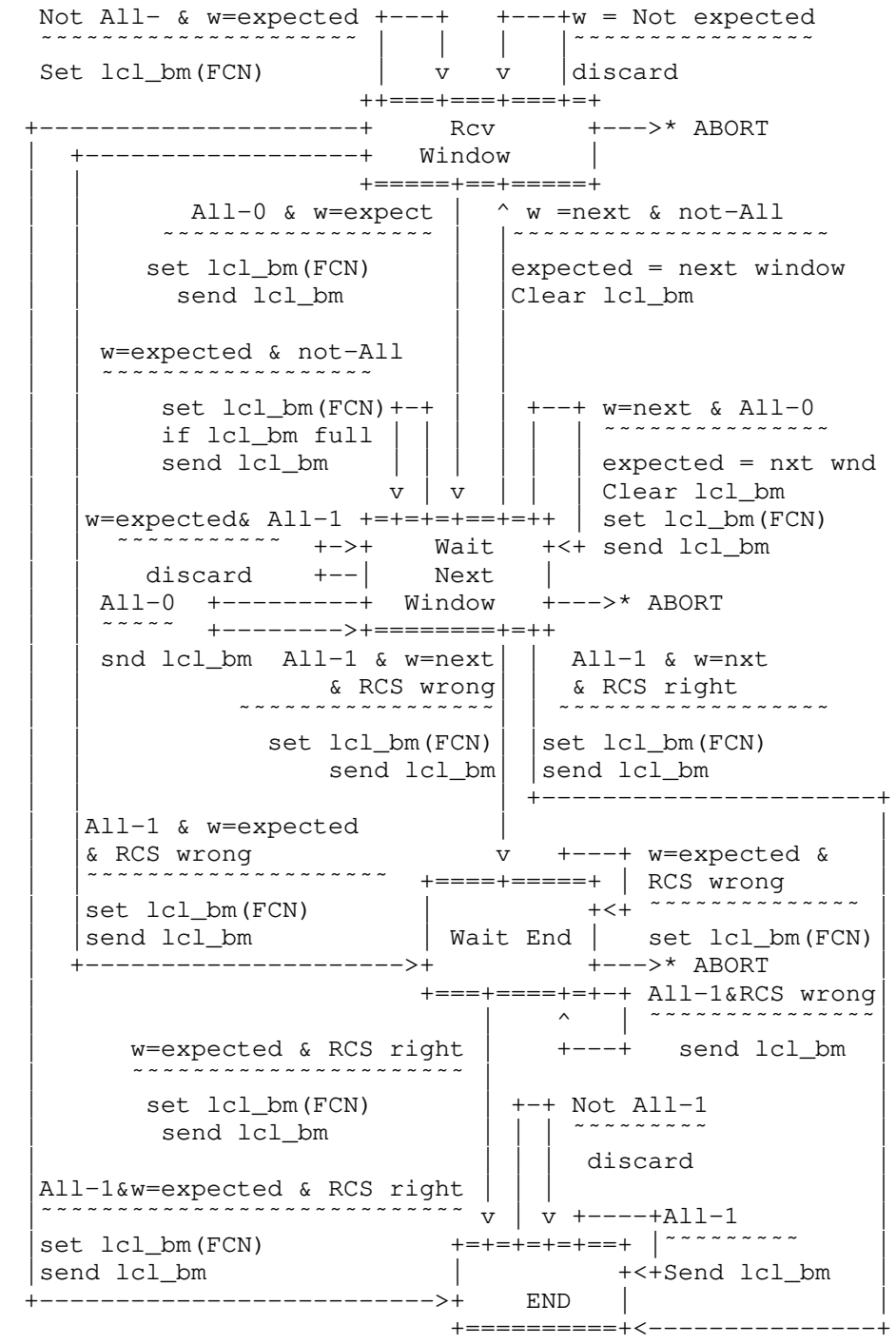


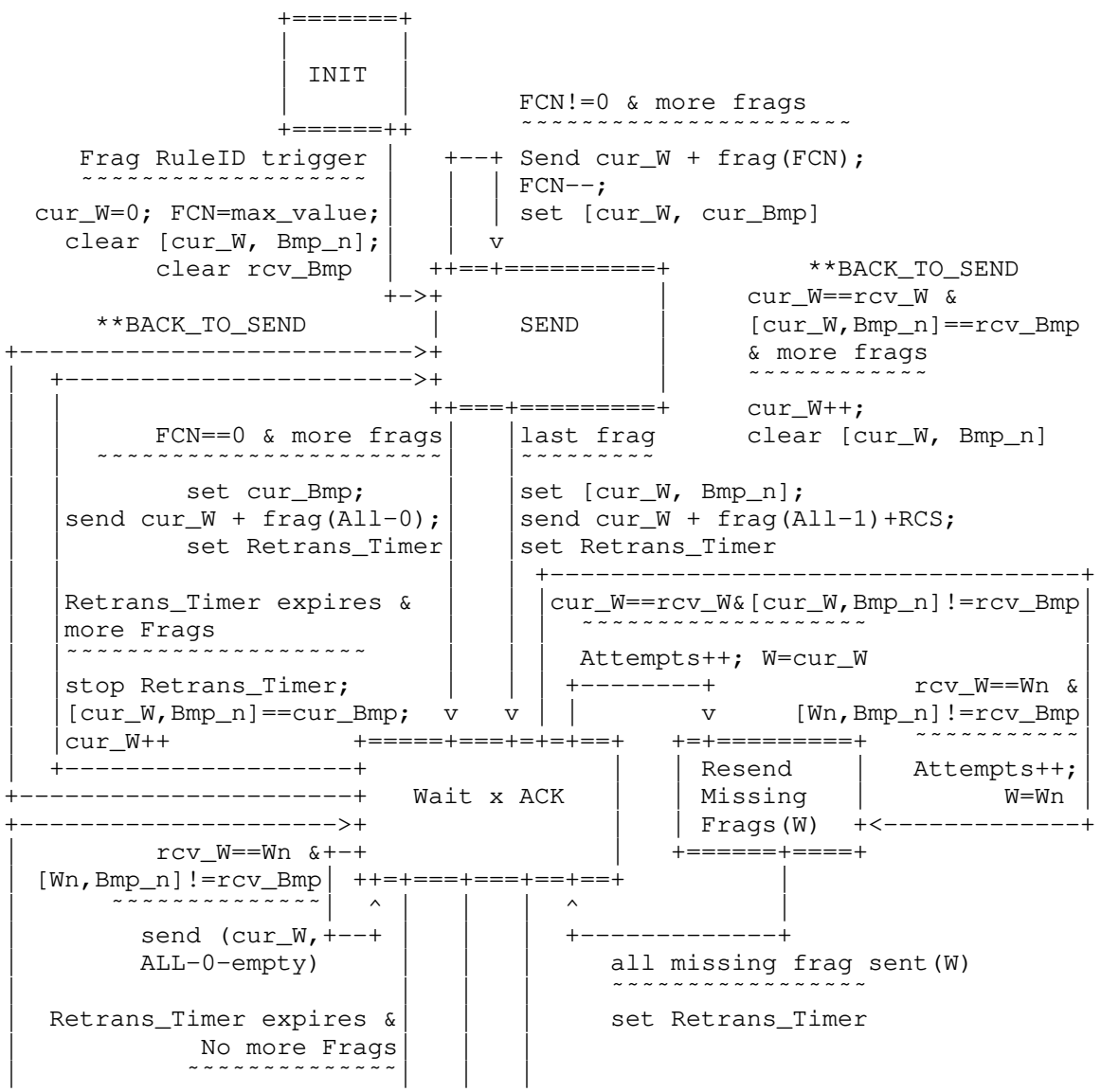
Figure 39: Sender State Machine for the ACK-Always Mode


Not All- & w=expected
Set lcl_bm(FCN)
w = Not expected
discard
Rcv
Window
* ABORT
All-0 & w=expect
set lcl_bm(FCN)
send lcl_bm
^ w =next & not-All
expected = next window
Clear lcl_bm
w=expected & not-All
set lcl_bm(FCN)
if lcl_bm full
send lcl_bm
w=next & All-0
expected = nxt wnd
Clear lcl_bm
set lcl_bm(FCN)
send lcl_bm
w=expected& All-1
discard
Wait
Next
All-0
Window
* ABORT
snd lcl_bm
All-1 & w=next
& RCS wrong
All-1 & w=nxt
& RCS right
set lcl_bm(FCN)
send lcl_bm
set lcl_bm(FCN)
send lcl_bm
All-1 & w=expected
& RCS wrong
w=expected &
RCS wrong
set lcl_bm(FCN)
send lcl_bm
Wait End
set lcl_bm(FCN)
* ABORT
All-1&RCS wrong
send lcl_bm
w=expected & RCS right
set lcl_bm(FCN)
send lcl_bm
Not All-1
discard
All-1&w=expected & RCS right
set lcl_bm(FCN)
send lcl_bm
All-1
Send lcl_bm
END

```
         --->* ABORT

         In any state
            on receiving a SCHC ACK REQ
               Send a SCHC ACK for the current window
```

Figure 40: Receiver State Machine for the ACK-Always Mode



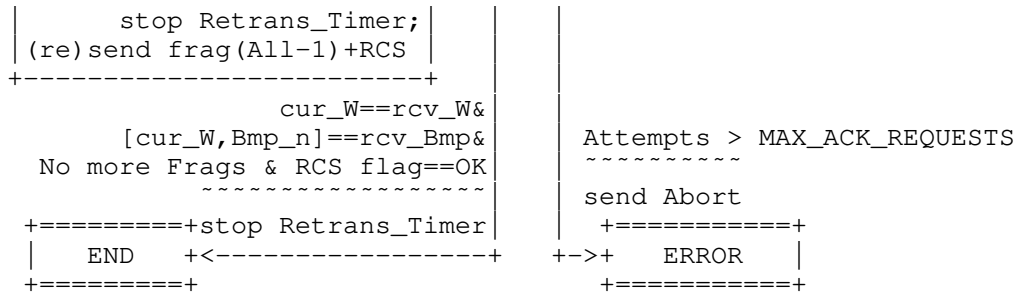
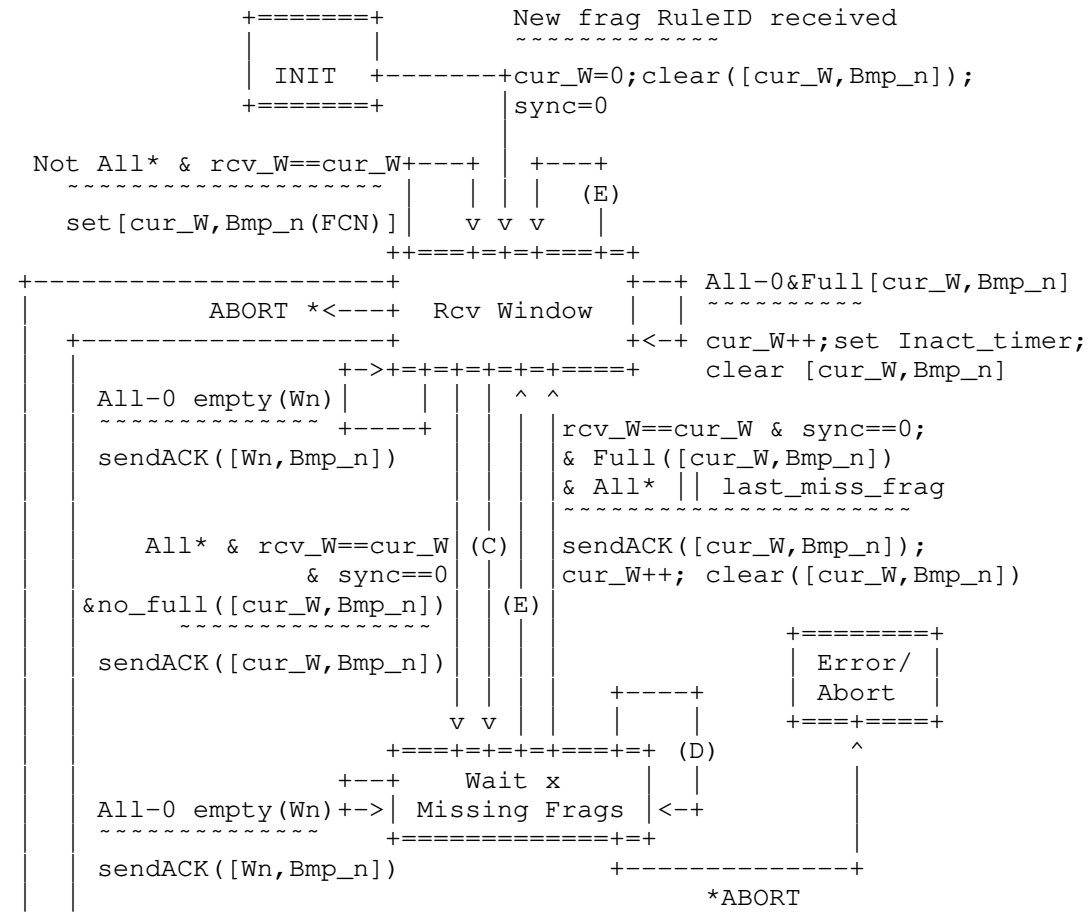


Figure 41: Sender State Machine for the ACK-on-Error Mode

This is an example only. It is not normative. The specification in Section 8.4.3.1 allows for sequences of operations different from the one shown here.



(C) All* & sync>0

Wn=oldest[not full(W)];
sendACK([Wn,Bmp_n])

(D) All* || last_miss_frag
& rcv_W!=cur_W & sync>0
& Full([rcv_W,Bmp_n])

Wn=oldest[not full(W)];
sendACK([Wn,Bmp_n]);sync--

ABORT-->* Uplink Only &
Inact_Timer expires
|| Attempts > MAX_ACK_REQUESTS

send Abort

(E) Not All* & rcv_W!=cur_W

sync++; cur_W=rcv_W;
set[cur_W,Bmp_n(FCN)]

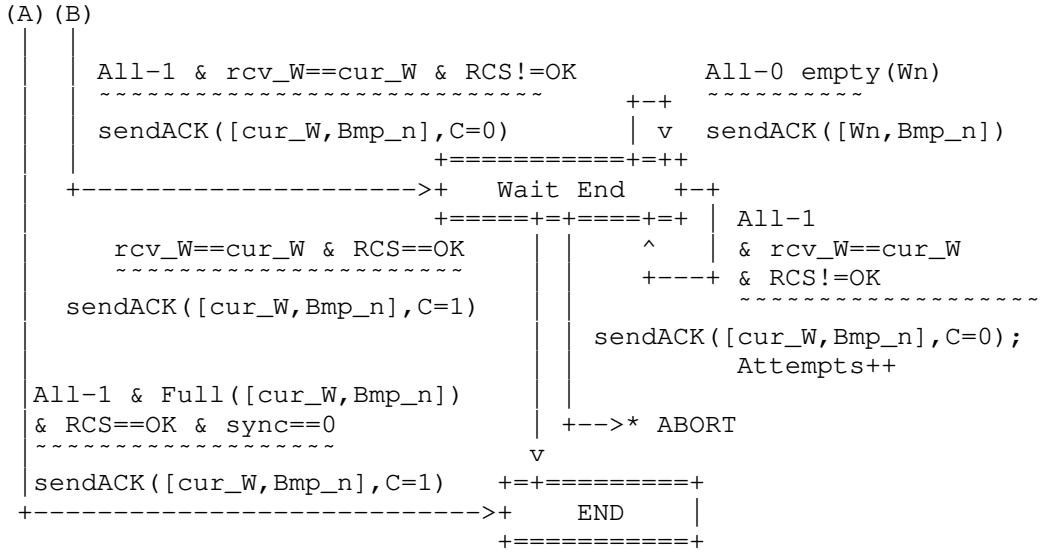


Figure 42: Receiver State Machine for the ACK-on-Error Mode

Appendix D. SCHC Parameters

This section lists the information that needs to be provided in the LPWAN technology-specific documents.

- Most common uses cases, deployment scenarios
- Mapping of the SCHC architectural elements onto the LPWAN architecture
- Assessment of LPWAN integrity checking
- Various potential channel conditions for the technology and the corresponding recommended use of SCHC C/D and F/R

This section lists the parameters that need to be defined in the Profile.

- Rule ID numbering scheme, fixed-sized or variable-sized Rule IDs, number of Rules, the way the Rule ID is transmitted
- maximum packet size that should ever be reconstructed by SCHC Decompression (MAX_PACKET_SIZE). See Section 12.

- Padding: size of the L2 Word (for most LPWAN technologies, this would be a byte; for some technologies, a bit)

- Decision to use SCHC fragmentation mechanism or not. If yes, the document must describe:

  * reliability mode(s) used, in which cases (e.g., based on link channel condition)

  * Rule ID values assigned to each mode in use

  * presence and number of bits for DTag (T) for each Rule ID value, lifetime of DTag at the receiver

  * support for interleaved packet transmission, to what extent

  * WINDOW_SIZE, for modes that use windows

  * number of bits for W (M) for each Rule ID value, for modes that use windows

  * number of bits for FCN (N) for each Rule ID value

  * what makes an All-0 SCHC Fragment and a SCHC ACK REQ distinguishable (see Section 8.3.1.1).

  * what makes an All-1 SCHC Fragment and a SCHC Sender-Abort distinguishable (see Section 8.3.1.2).

  * size of RCS and algorithm for its computation, for each Rule ID, if different from the default CRC32. Byte fill-up with zeroes or other mechanism, to be specified.

  * Retransmission Timer duration for each Rule ID value, if applicable to the SCHC F/R mode

  * Inactivity Timer duration for each Rule ID value, if applicable to the SCHC F/R mode

  * MAX_ACK_REQUESTS value for each Rule ID value, if applicable to the SCHC F/R mode

- if L2 Word is wider than a bit and SCHC fragmentation is used, value of the padding bits (0 or 1). This is needed because the padding bits of the last fragment are included in the RCS computation.

A Profile may define a delay to be added after each SCHC message transmission for compliance with local regulations or other constraints imposed by the applications.

- In some LPWAN technologies, as part of energy-saving techniques, downlink transmission is only possible immediately after an uplink transmission. In order to avoid potentially high delay in the downlink transmission of a fragmented SCHC Packet, the SCHC Fragment receiver may perform an uplink transmission as soon as possible after reception of a SCHC Fragment that is not the last one. Such uplink transmission may be triggered by the L2 (e.g., an L2 ACK sent in response to a SCHC Fragment encapsulated in a L2 PDU that requires an L2 ACK) or it may be triggered from an upper layer. See Appendix F.

- the following parameters need to be addressed in documents other than this one but not necessarily in the LPWAN technology-specific documents:

  * The way the Contexts are provisioned

  * The way the Rules are generated

Appendix E. Supporting multiple window sizes for fragmentation

For ACK-Always or ACK-on-Error, implementers may opt to support a single window size or multiple window sizes. The latter, when feasible, may provide performance optimizations. For example, a large window size should be used for packets that need to be split into a large number of tiles. However, when the number of tiles required to carry a packet is low, a smaller window size, and thus a shorter Bitmap, may be sufficient to provide reception status on all tiles. If multiple window sizes are supported, the Rule ID signals the window size in use for a specific packet transmission.

Appendix F. ACK-Always and ACK-on-Error on quasi-bidirectional links

The ACK-Always and ACK-on-Error modes of SCHC F/R are bidirectional protocols: they require a feedback path from the reassembler to the fragmenter.

Some LPWAN technologies provide quasi-bidirectional connectivity, whereby a downlink transmission from the Network Infrastructure can only take place right after an uplink transmission by the Dev.

When using SCHC F/R to send fragmented SCHC Packets downlink over these quasi-bidirectional links, the following situation may arise: if an uplink SCHC ACK is lost, the SCHC ACK REQ message by the sender

could be stuck indefinitely in the downlink queue at the Network Infrastructure, waiting for a transmission opportunity.

There are many ways by which this deadlock can be avoided. The Dev application might be sending recurring uplink messages such as keep-alive, or the Dev application stack might be sending other recurring uplink messages as part of its operation. However, these are out of the control of this generic SCHC specification.

In order to cope with quasi-bidirectional links, a SCHC-over-foo specification may want to amend the SCHC F/R specification to add a timer-based retransmission of the SCHC ACK. Below is an example of the suggested behavior for ACK-Always mode. Because it is an example, [RFC2119] language is deliberately not used here.

For downlink transmission of a fragmented SCHC Packet in ACK-Always mode, the SCHC Fragment receiver may support timer-based SCHC ACK retransmission. In this mechanism, the SCHC Fragment receiver initializes and starts a timer (the UplinkACK Timer) after the transmission of a SCHC ACK, except when the SCHC ACK is sent in response to the last SCHC Fragment of a packet (All-1 fragment). In the latter case, the SCHC Fragment receiver does not start a timer after transmission of the SCHC ACK.

If, after transmission of a SCHC ACK that is not an All-1 fragment, and before expiration of the corresponding UplinkACK timer, the SCHC Fragment receiver receives a SCHC Fragment that belongs to the current window (e.g., a missing SCHC Fragment from the current window) or to the next window, the UplinkACK timer for the SCHC ACK is stopped. However, if the UplinkACK timer expires, the SCHC ACK is resent and the UplinkACK timer is reinitialized and restarted.

The default initial value for the UplinkACK Timer, as well as the maximum number of retries for a specific SCHC ACK, denoted MAX_ACK_REQUESTS, is to be defined in a Profile. The initial value of the UplinkACK timer is expected to be greater than that of the Retransmission timer, in order to make sure that a (buffered) SCHC Fragment to be retransmitted finds an opportunity for that transmission. One exception to this recommendation is the special case of the All-1 SCHC Fragment transmission.

When the SCHC Fragment sender transmits the All-1 SCHC Fragment, it starts its Retransmission Timer with a large timeout value (e.g., several times that of the initial UplinkACK Timer). If a SCHC ACK is received before expiration of this timer, the SCHC Fragment sender retransmits any lost SCHC Fragments as reported by the SCHC ACK, or if the SCHC ACK confirms successful reception of all SCHC Fragments of the last window, the transmission of the fragmented SCHC Packet is

considered complete. If the timer expires, and no SCHC ACK has been received since the start of the timer, the SCHC Fragment sender assumes that the All-1 SCHC Fragment has been successfully received (and possibly, the last SCHC ACK has been lost: this mechanism assumes that the Retransmission Timer for the All-1 SCHC Fragment is long enough to allow several SCHC ACK retries if the All-1 SCHC Fragment has not been received by the SCHC Fragment receiver, and it also assumes that it is unlikely that several ACKs become all lost).

Authors' Addresses

Ana Minaburo
Acklio
1137A avenue des Champs Blancs
35510 Cesson-Sevigne Cedex
France

Email: ana@ackl.io

Laurent Toutain
IMT-Atlantique
2 rue de la Chataigneraie
CS 17607
35576 Cesson-Sevigne Cedex
France

Email: Laurent.Toutain@imt-atlantique.fr

Carles Gomez
Universitat Politecnica de Catalunya
C/Esteve Terradas, 7
08860 Castelldefels
Spain

Email: carlesgo@entel.upc.edu

Dominique Barthel
Orange Labs
28 chemin du Vieux Chene
38243 Meylan
France

Email: dominique.barthel@orange.com

Juan Carlos Zuniga
SIGFOX
425 rue Jean Rostand
Labege 31670
France

Email: JuanCarlos.Zuniga@sigfox.com

Network Working Group
Internet-Draft
Obsoletes: 1226 (if approved)
Intended status: Experimental
Expires: November 27, 2020

I. Learmonth
HamBSD

May 26, 2020

Internet Protocol Encapsulation of AX.25 Frames
draft-learmonth-intarea-rfc1226-bis-01

Abstract

This document describes a method for the encapsulation of AX.25 Link Access Protocol for Amateur Packet Radio frames within IPv4 and IPv6 packets. Obsoletes RFC1226.

Note

Comments are solicited and should be addressed to the author(s).

The sources for this draft are at:

https://github.com/irl/draft-rfc1226-bis

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on November 27, 2020.

1. Introduction

This document describes a method for the encapsulation of AX.25 Link Access Protocol for Amateur Packet Radio [AX.25] frames within IPv4 and IPv6 packets. It obsoletes [RFC1226].

AX.25 is a data link layer protocol originally derived from layer 2 of the X.25 protocol suite and designed for use by amateur radio operators. It is used extensively by amateur packet radio networks worldwide.

In addition to specifying how packets should be encapsulated, it gives recommendations for DiffServ codepoint marking of the encapsulating headers based on the AX.25 frame content and provides security considerations for the use of this encapsulation method.

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 [RFC2119].

3. Internet Protocol Encapsulation

Each AX.25 frame is encapsulated in one IPv4 or IPv6 datagram using protocol number 93 as assigned in the Assigned Internet Protocol Numbers registry [protocol-numbers]. For AX.25 version 2.0, the maximum frame size expected is 330 bytes and implementations MUST be prepared to handle frames of this size. Higher frame sizes can be negotiated by AX.25 version 2.2 and so this is a minimum requirement and not a limit.

HDLC framing elements (flags and zero-stuffing) are omitted, as the IP datagram adequately delimits the beginning and end of each AX.25 frame. The CRC-16-CCITT frame check sequence (normally generated by the HDLC transmission hardware) is included trailing the information field. In all other respects, AX.25 frames are encapsulated unaltered.

3.1. Priority Frames

In normal operation, the DiffServ codepoint field [RFC2474] in the encapsulating IP header SHOULD be set to best effort (BE). The exception to this is "priority frames" as specified for AX.25 version 2.2, including acknowledgement and digipeat frames, which SHOULD have the DiffServ codepoint set to AF21 [RFC2597]. A slot is reserved on the radio channel for the transmission of these frames and the use of this codepoint will permit the frames to arrive promptly at the station for transmission.

For the avoidance of doubt: on decapsulation the AX.25 frame MUST NOT be modified based on the DiffServ codepoint on the received encapsulating IP header. The receiver MUST NOT use the DiffServ codepoint to infer anything about the nature of the encapsulated packet. It has been shown that while the AF21 codepoint may be remarked while crossing administrative boundaries, it is unlikely that priority inversion will occur due to remarking where such remarking occurs [Cust18].

3.2. Automatic Packet Reporting System

Automatic Packet Reporting System [APRS] is an amateur radio-based system for real time digital communications for local situational awareness. APRS uses AX.25 frames for addressing, and additionally assigns special meaning to some of the reserved bits of an AX.25 frame header.

As a special case, when used with the Automatic Packet Reporting System [APRS], priority frames will not occur. If a tunnel is configured as carrying APRS data, the DiffServ codepoint SHOULD by default be set to AF11 [RFC2597]. Where the "Precedence Bit" [RR-bits] is set (i.e. it is zero) in an APRS packet, the DiffServ codepoint should be set to BE. Where the "Operator Present Bit" [RR-bits] is set (i.e. it is zero), the DiffServ codepoint MAY be set to AF21 [RFC2597].

Again, for the avoidance of doubt: on decapsulation the AX.25 frame MUST NOT be modified based on the DiffServ codepoint on the received encapsulating IP header. The receiver MUST NOT use the DiffServ codepoint to infer anything about the nature of the encapsulated packet. It has been shown that while AF codepoints may be remarked while crossing administrative boundaries, it is unlikely that priority inversion will occur, either with the BE traffic or between AF PHBs due to remarking where such remarking occurs [Cust18].

It is possible depending on the nature of the tunnel that decapsulated packets would need to be treated as third-party traffic

according to the APRS specification [APRS]. In this case, the Third-Party Network Identifier "IPENC" SHOULD be used. This is to differentiate traffic using IP encapsulation from APRS-IS traffic [APRS-IS] and other third-party networks.

4. Security Considerations

With the exception of control signals exchanged between earth command stations and space stations in the amateur-satellite service, amateur radio transmissions cannot be encoded for the purpose of obscuring their meaning. In essence, this means that cryptography that requires the use of secrets to decipher a message cannot be used where the possibility exists that a packet will be transmitted by an amateur radio station [Part97.113][OfcomTerms].

The CRC-16-CCITT provides for an integrity check but does not guarantee the authenticity of the packet. In many jurisdictions it is a requirement for amateur radio stations that are Internet connected that they verify that packets for transmission have originated from licensed radio amateurs [Part97.111][OfcomTerms].

In order to provide this guarantee, IPSec [RFC4301] SHOULD be employed to provide authentication of packets. The neogtiated SA SHOULD use transport mode with ESP [RFC4303] to limit the packet size overhead incurred by use of IPSec. The traffic selector MUST match packets with IP protocol number 93. An authentication algorithm MUST be selected to provide data origin authentication.

The encryption algorithm MUST NOT provide confidentiality for tunnels that will traverse an amateur radio link (i.e. the encapsulated packets will be transmitted by an amateur radio station). The use of the NULL algorithm [RFC2410] is RECOMMENDED for tunnels that will traverse an amateur radio link. In cases where traffic can be known or reasonably expected to not traverse an amateur radio link, an encryption algorithm that provides confidentiality is RECOMMENDED.

When transmitted by an amateur radio station, many propagation modes will permit wide reception of a packet. As such, receivers MUST implement anti-replay protection by verifying received sequence numbers [RFC4303]. The size of the anti-replay window may need to be scaled to account not only for the speed of the link, but also for packet loss that may occur on amateur radio links. Following extended packet loss a sender may have advanced the sequence number beyond the window size allowed. Dead peer detection [RFC7296] can be used to renegotiate SAs in this case and so SHOULD be enabled for any SA expected to traverse an amateur radio link that is expected to have varying propagation charachteristics.

Given the need for anti-replay protection, it is not possible to manually key the SAs. IKEv2 [RFC7296] SHOULD be used to establish SAs. Beyond the above, the exact details of the automatic keying protocol to use and its paramaters are not specified in this document.

5. IANA Considerations

Protocol number 93 is assigned in [protocol-numbers] and should be updated to point to this document.

6. Acknowledgements

The author would like to acknowledge the work of Brian Kantor who authored the original specification [RFC1226] that this document updates.

7. References

7.1. Normative References

[AX.25] Tucson Amateur Packet Radio Corporation, "AX.25 Link Access Protocol for Amateur Packet Radio Version 2.2", July 1998, <https://www.tapr.org/pdf/AX25.2.2.pdf>.

[protocol-numbers] IANA, "Assigned Internet Protocol Numbers", <http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2410] Glenn, R. and S. Kent, "The NULL Encryption Algorithm and Its Use With IPsec", RFC 2410, DOI 10.17487/RFC2410, November 1998, <https://www.rfc-editor.org/info/rfc2410>.

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <https://www.rfc-editor.org/info/rfc2474>.

[RFC2597] Heinanen, J., Baker, F., Weiss, W., and J. Wroclawski, "Assured Forwarding PHB Group", RFC 2597, DOI 10.17487/RFC2597, June 1999, <https://www.rfc-editor.org/info/rfc2597>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/info/rfc4301>.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <https://www.rfc-editor.org/info/rfc4303>.

[RR-bits] Bruninga, B., "APRS Future Use of AX.25 SSID RR Bits", December 2012, <http://aprs.org/aprs12/RR-bits.txt>.

7.2. Informative References

[APRS] Wade, I., Ed., "APRS Protocol Reference", August 2000, <http://www.aprs.org/doc/APRS101.PDF>.

[APRS-IS] Loveall, P., "APRS-IS", <http://www.aprs-is.net/>.

[Cust18] Custura, A., Secchi, R., and G. Fairhurst, "Exploring DSCP modification pathologies in the Internet", Computer Communications Vol. 127, pp. 86-94, DOI 10.1016/j.comcom.2018.05.016, September 2018.

[OfcomTerms] Ofcom, "UK Amateur Radio Licence Section 2", <https://www.ofcom.org.uk/__data/assets/pdf_file/0027/62991/amateur-terms.pdf>.

[Part97.111] e-CFR, "Electronic Code of Federal Regulations Title 47, Part 97.111 - Authorized transmissions", <https://www.ecfr.gov/cgi-bin/text-idx?node=pt47.5.97&rgn=div5#se47.5.97_1111>.

[Part97.113] e-CFR, "Electronic Code of Federal Regulations Title 47, Part 97.113 - Prohibited transmissions", <https://www.ecfr.gov/cgi-bin/text-idx?node=pt47.5.97&rgn=div5#se47.5.97_1113>.

[RFC1226] Kantor, B., "Internet protocol encapsulation of AX.25 frames", RFC 1226, DOI 10.17487/RFC1226, May 1991, <https://www.rfc-editor.org/info/rfc1226>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/info/rfc7296>.

Author's Address

Iain R. Learmonth
HamBSD

Email: irl@hambsd.org

Network Working Group
Internet-Draft
Obsoletes: 1226 (if approved)
Intended status: Experimental
Expires: May 23, 2021

I. Learmonth
HamBSD

November 19, 2020

Internet Protocol Encapsulation of AX.25 Frames
draft-learmonth-intarea-rfc1226-bis-02

Abstract

This document describes a method for the encapsulation of AX.25 Link Access Protocol for Amateur Packet Radio frames within IPv4 and IPv6 packets. Obsoletes RFC1226.

Note

Comments are solicited and should be addressed to the author(s).

The sources for this draft are at:

https://github.com/irl/draft-rfc1226-bis

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on May 23, 2021.

Copyright Notice

1. Introduction

This document describes a method for the encapsulation of AX.25 Link Access Protocol for Amateur Packet Radio [AX.25] frames within IPv4 and IPv6 packets. It obsoletes [RFC1226].

AX.25 is a data link layer protocol originally derived from layer 2 of the X.25 protocol suite and designed for use by amateur radio operators. It is used extensively by amateur packet radio networks worldwide.

In addition to specifying how packets should be encapsulated, it gives recommendations for DiffServ codepoint marking of the encapsulating headers based on the AX.25 frame content and provides security considerations for the use of this encapsulation method.

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 [RFC2119].

3. Internet Protocol Encapsulation

Each AX.25 frame is encapsulated in one IPv4 or IPv6 datagram using protocol number 93 as assigned in the Assigned Internet Protocol Numbers registry [protocol-numbers]. For AX.25 version 2.0, the maximum frame size expected is 330 bytes and implementations MUST be prepared to handle frames of this size. Higher frame sizes can be negotiated by AX.25 version 2.2 and so this is a minimum requirement and not a limit.

HDLC framing elements (flags and zero-stuffing) are omitted, as the IP datagram adequately delimits the beginning and end of each AX.25 frame. The CRC-16-CCITT frame check sequence (normally generated by the HDLC transmission hardware) is included trailing the information field. In all other respects, AX.25 frames are encapsulated unaltered.

3.1. Priority Frames

In normal operation, the DiffServ codepoint field [RFC2474] in the encapsulating IP header SHOULD be set to best effort (BE). The exception to this is "priority frames" as specified for AX.25 version 2.2, including acknowledgement and digipeat frames, which SHOULD have the DiffServ codepoint set to AF21 [RFC2597]. A slot is reserved on the radio channel for the transmission of these frames and the use of this codepoint will permit the frames to arrive promptly at the station for transmission.

For the avoidance of doubt: on decapsulation the AX.25 frame MUST NOT be modified based on the DiffServ codepoint on the received encapsulating IP header. The receiver MUST NOT use the DiffServ codepoint to infer anything about the nature of the encapsulated packet. It has been shown that while the AF21 codepoint may be remarked while crossing administrative boundaries, it is unlikely that priority inversion will occur due to remarking where such remarking occurs [Cust18].

3.2. Automatic Packet Reporting System

Automatic Packet Reporting System [APRS] is an amateur radio-based system for real time digital communications for local situational awareness. APRS uses AX.25 frames for addressing, and additionally assigns special meaning to some of the reserved bits of an AX.25 frame header.

As a special case, when used with the Automatic Packet Reporting System [APRS], priority frames will not occur. If a tunnel is configured as carrying APRS data, the DiffServ codepoint SHOULD by default be set to AF11 [RFC2597]. Where the "Precedence Bit" [RR-bits] is set (i.e. it is zero) in an APRS packet, the DiffServ codepoint should be set to BE. Where the "Operator Present Bit" [RR-bits] is set (i.e. it is zero), the DiffServ codepoint MAY be set to AF21 [RFC2597].

Again, for the avoidance of doubt: on decapsulation the AX.25 frame MUST NOT be modified based on the DiffServ codepoint on the received encapsulating IP header. The receiver MUST NOT use the DiffServ codepoint to infer anything about the nature of the encapsulated packet. It has been shown that while AF codepoints may be remarked while crossing administrative boundaries, it is unlikely that priority inversion will occur, either with the BE traffic or between AF PHBs due to remarking where such remarking occurs [Cust18].

It is possible depending on the nature of the tunnel that decapsulated packets would need to be treated as third-party traffic

according to the APRS specification [APRS]. In this case, the Third-Party Network Identifier "IPENC" SHOULD be used. This is to differentiate traffic using IP encapsulation from APRS-IS traffic [APRS-IS] and other third-party networks.

4. Security Considerations

With the exception of control signals exchanged between earth command stations and space stations in the amateur-satellite service, amateur radio transmissions cannot be encoded for the purpose of obscuring their meaning. In essence, this means that cryptography that requires the use of secrets to decipher a message cannot be used where the possibility exists that a packet will be transmitted by an amateur radio station [Part97.113][OfcomTerms].

The CRC-16-CCITT provides for an integrity check but does not guarantee the authenticity of the packet. In many jurisdictions it is a requirement for amateur radio stations that are Internet connected that they verify that packets for transmission have originated from licensed radio amateurs [Part97.111][OfcomTerms].

In order to provide this guarantee, IPSec [RFC4301] SHOULD be employed to provide authentication of packets. The neogtiated SA SHOULD use transport mode with ESP [RFC4303] to limit the packet size overhead incurred by use of IPSec. The traffic selector MUST match packets with IP protocol number 93. An authentication algorithm MUST be selected to provide data origin authentication.

The encryption algorithm MUST NOT provide confidentiality for tunnels that will traverse an amateur radio link (i.e. the encapsulated packets will be transmitted by an amateur radio station). The use of the NULL algorithm [RFC2410] is RECOMMENDED for tunnels that will traverse an amateur radio link. In cases where traffic can be known or reasonably expected to not traverse an amateur radio link, an encryption algorithm that provides confidentiality is RECOMMENDED.

Wrapped ESP [RFC5840] MAY be used to explicitly indicate where "integrity-only" security is provided without data confidentiality.

When transmitted by an amateur radio station, many propagation modes will permit wide reception of a packet. As such, receivers MUST implement anti-replay protection by verifying received sequence numbers [RFC4303]. The size of the anti-replay window may need to be scaled to account not only for the speed of the link, but also for packet loss that may occur on amateur radio links. Following extended packet loss a sender may have advanced the sequence number beyond the window size allowed. Dead peer detection [RFC7296] can be used to renegotiate SAs in this case and so SHOULD be enabled for any

SA expected to traverse an amateur radio link that is expected to have varying propagation charachteristics.

Given the need for anti-replay protection, it is not possible to manually key the SAs. IKEv2 [RFC7296] SHOULD be used to establish SAs. Beyond the above, the exact details of the automatic keying protocol to use and its paramaters are not specified in this document.

5. IANA Considerations

Protocol number 93 is assigned in [protocol-numbers] and should be updated to point to this document.

6. Acknowledgements

The author would like to acknowledge the work of Brian Kantor who authored the original specification [RFC1226] that this document updates.

7. References

7.1. Normative References

[AX.25] Tucson Amateur Packet Radio Corporation, "AX.25 Link Access Protocol for Amateur Packet Radio Version 2.2", July 1998, <https://www.tapr.org/pdf/AX25.2.2.pdf>.

[protocol-numbers] IANA, "Assigned Internet Protocol Numbers", <http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2410] Glenn, R. and S. Kent, "The NULL Encryption Algorithm and Its Use With IPsec", RFC 2410, DOI 10.17487/RFC2410, November 1998, <https://www.rfc-editor.org/info/rfc2410>.

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <https://www.rfc-editor.org/info/rfc2474>.

[RFC2597] Heinanen, J., Baker, F., Weiss, W., and J. Wroclawski, "Assured Forwarding PHB Group", RFC 2597, DOI 10.17487/RFC2597, June 1999, <https://www.rfc-editor.org/info/rfc2597>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/info/rfc4301>.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <https://www.rfc-editor.org/info/rfc4303>.

[RFC5840] Grewal, K., Montenegro, G., and M. Bhatia, "Wrapped Encapsulating Security Payload (ESP) for Traffic Visibility", RFC 5840, DOI 10.17487/RFC5840, April 2010, <https://www.rfc-editor.org/info/rfc5840>.

[RR-bits] Bruninga, B., "APRS Future Use of AX.25 SSID RR Bits", December 2012, <http://aprs.org/aprs12/RR-bits.txt>.

7.2. Informative References

[APRS] Wade, I., Ed., "APRS Protocol Reference", August 2000, <http://www.aprs.org/doc/APRS101.PDF>.

[APRS-IS] Loveall, P., "APRS-IS", <http://www.aprs-is.net/>.

[Cust18] Custura, A., Secchi, R., and G. Fairhurst, "Exploring DSCP modification pathologies in the Internet", Computer Communications Vol. 127, pp. 86-94, DOI 10.1016/j.comcom.2018.05.016, September 2018.

[OfcomTerms] Ofcom, "UK Amateur Radio Licence Section 2", <https://www.ofcom.org.uk/__data/assets/pdf_file/0027/62991/amateur-terms.pdf>.

[Part97.111] e-CFR, "Electronic Code of Federal Regulations Title 47, Part 97.111 - Authorized transmissions", <https://www.ecfr.gov/cgi-bin/text-idx?node=pt47.5.97&rgn=div5#se47.5.97_1111>.

[Part97.113]
e-CFR, "Electronic Code of Federal Regulations Title 47, Part 97.113 - Prohibited transmissions", <https://www.ecfr.gov/cgi-bin/text-idx?node=pt47.5.97&rgn=div5#se47.5.97_1113>.

[RFC1226] Kantor, B., "Internet protocol encapsulation of AX.25 frames", RFC 1226, DOI 10.17487/RFC1226, May 1991, <https://www.rfc-editor.org/info/rfc1226>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/info/rfc7296>.

Author's Address

Iain R. Learmonth
HamBSD

Email: irl@hambsd.org

Internet Area Working Group V. Olteanu
Internet-Draft D. Niculescu
Intended status: Experimental University Politehnica of Bucharest
Expires: January 14, 2021 July 13, 2020

SOCKS Protocol Version 6
draft-olteanu-intarea-socks-6-10

Abstract

The SOCKS protocol is used primarily to proxy TCP connections to arbitrary destinations via the use of a proxy server. Under the latest version of the protocol (version 5), it takes 2 RTTs (or 3, if authentication is used) before data can flow between the client and the server.

This memo proposes SOCKS version 6, which reduces the number of RTTs used, takes full advantage of TCP Fast Open, and adds support for 0-RTT authentication.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on January 14, 2021.

Table of Contents

1. Introduction

Versions 4 and 5 [RFC1928] of the SOCKS protocol were developed two decades ago and are in widespread use for circuit level gateways or as circumvention tools, and enjoy wide support and usage from various software, such as web browsers, SSH clients, and proxifiers. However, their design needs an update in order to take advantage of the new features of transport protocols, such as TCP Fast Open [RFC7413], or to better assist newer transport protocols, such as MPTCP [RFC6824].

One of the main issues faced by SOCKS version 5 is that, when taking into account the TCP handshake, method negotiation, authentication, connection request and grant, it may take up to 5 RTTs for a data exchange to take place at the application layer. This is especially costly in networks with a large delay at the access layer, such as 3G, 4G, or satelite.

The desire to reduce the number of RTTs manifests itself in the design of newer security protocols. TLS version 1.3 [RFC8446] defines a zero round trip (0-RTT) handshake mode for connections if the client and server had previously communicated.

TCP Fast Open [RFC7413] is a TCP option that allows TCP to send data in the SYN and receive a response in the first ACK, and aims at obtaining a data response in one RTT. The SOCKS protocol needs to concern itself with at least two TFO deployment scenarios: First, when TFO is available end-to-end (at the client, at the proxy, and at the server); second, when TFO is active between the client and the proxy, but not at the server.

This document describes the SOCKS protocol version 6. The key improvements over SOCKS version 5 are:

- The client sends as much information upfront as possible, and does not wait for the authentication process to conclude before requesting the creation of a socket.

- The connection request also mimics the semantics of TCP Fast Open [RFC7413]. As part of the connection request, the client can supply the potential payload for the initial SYN that is sent out to the server.
- The protocol can be extended via options without breaking backward-compatibility.
- The protocol can leverage the aforementioned options to support 0-RTT authentication schemes.

1.1. Revision log

Typos and minor clarifications are not listed.

draft-10

- Removed untrusted sessions
- IP DF
- UDP relay:
  - Support ICMPv6 Too Big
  - Shifted some fields in the error messages
  - RTP support

draft-09

- Revamped UDP relay
  - Support for ICMP errors: host/net unreachable, TTL exceeded
  - Datagrams can be sent over TCP
  - Timeout for the receipt of the initial datagram
- TTL stack option (intended use: traceroute)
- Added the "Privacy Considerations" section
- SOCKS-provided DNS: the proxy may provide a valid bind address and port

draft-08

- Removed Address Resolution options
- Happy Eyeballs options
- DNS provided by SOCKS

draft-07

- All fields are now alignned.
- Eliminated version minors
- Lots of changes to options
  - 2-byte option kinds
  - Flattened option kinds/types/reply codes; also renamed some options
  - Socket options
    - Proxies MUST always answer them (Clients can probe for support)
    - MPTCP Options: expanded functionality ("please do/don't do MPTCP on my behalf")
    - MPTCP Scheduler options removed
    - Listen Backlog options: code changed to 0x03
  - Revamped Idempotence options
  - Auth data options limited to one per method
- Authentication Reply: all authentication-related information is now in the options
  - Authentication replies no longer have a field indicating the chosen auth. method
  - Method that must proceed (or whereby authentication succeeded) indicated in options
  - Username/password authentication: proxy now sends reply in option

- Removed requirements w.r.t. caching authentication methods by multihomed clients
- UDP: 8-byte association IDs
- Sessions
  - The proxy is now free to terminate ongoing connections along with the session.
  - The session-terminating request is not part of the session that it terminated.
- Address Resolution options

draft-06

- Session options
- Options now have a 2-byte length field.
- Stack options
  - Stack options can no longer contain duplicate information.
  - TFO: Better payload size semantics
  - TOS: Added missing code field.
  - MPTCP Scheduler options:
    - Removed support for round-robin
    - "Default" renamed to "Lowest latency first"
  - Listen Backlog options: now tied to sessions, instead of an authenticated user
- Idempotence options
  - Now used in the context of a session (no longer tied to an authenticated user)
  - Idempotence options have a different codepoint: 0x05. (Was 0x04.)
  - Clarified that implementations that support Idempotence Options must support all Idempotence Option Types.

    * Shifted Idempotence Option Types by 1. (Makes implementation easier.)

- Shrunk vendor-specific option range to 32 (down from 64).
- Removed reference to dropping initial data. (It could no longer be done as of -05.)
- Initial data size capped at 16KB.
- Application data is never encrypted by SOCKS 6. (It can still be encrypted by the TLS layer under SOCKS.)
- Messages now carry the total length of the options, rather than the number of options. Limited options length to 16KB.
- Security Considerations
    * Updated the section to reflect the smaller maximum message size.
    * Added a subsection on resource exhaustion.

draft-05

- Limited the "slow" authentication negociations to one (and Authentication Replies to 2)
- Revamped the handling of the first bytes in the application data stream
    * False starts are now recommended. (Added the "False Start" section.)
    * Initial data is only available to clients willing to do "slow" authentication. Moved the "Initial data size" field from Requests to Authentication Method options.
    * Initial data size capped at 2^13. Initial data can no longer be dropped by the proxy.
    * The TFO option can hint at the desired SYN payload size.
- Request: clarified the meaning of the Address and Port fields.
- Better reverse TCP proxy support: optional listen backlog for TCP BIND

- TFO options can no longer be placed inside Operation Replies.
- IP TOS stack option
- Suggested a range for vendor-specific options.
- Revamped UDP functionality
  - Now using fixed UDP ports
  - DTLS support
- Stack options: renamed Proxy-Server leg to Proxy-Remote leg

draft-04

- Moved Token Expenditure Replies to the Authentication Reply.
- Shifted the Initial Data Size field in the Request, in order to make it easier to parse.

draft-03

- Shifted some fields in the Operation Reply to make it easier to parse.
- Added connection attempt timeout response code to Operation Replies.
- Proxies send an additional Authentication Reply after the authentication phase. (Useful for token window advertisements.)
- Renamed the section "Connection Requests" to "Requests"
- Clarified the fact that proxies don’t need to support any command in particular.
- Added the section "TCP Fast Open on the Client-Proxy Leg"
- Options:
  - Added constants for option kinds
  - Salt options removed, along with the relevant section from Security Considerations. (TLS 1.3 Makes AEAD mandatory.)
  - Limited Authentication Data options to one per method.

* Relaxed proxy requirements with regard to handling multiple Authentication Data options. (When the client violates the above bullet point.)

* Removed interdependence between Authentication Method and Authentication Data options.

* Clients SHOULD omit advertising the "No authentication required" option. (Was MAY.)

* Idempotence options:

  + Token Window Advertisements are now part of successful Authentication Replies (so that the proxy-server RTT has no impact on their timeliness).

  + Proxies can’t advetise token windows of size 0.

  + Tweaked token expenditure response codes.

  + Support no longer mandatory on the proxy side.

* Revamped Socket options

  + Renamed Socket options to Stack options.

  + Banned contradictory socket options.

  + Added socket level for generic IP. Removed the "socket" socket level.

  + Stack options no longer use option codes from setsockopt().

  + Changed MPTCP Scheduler constants.

draft-02

- Made support for Idempotence options mandatory for proxies.

- Clarified what happens when proxies can not or will not issue tokens.

- Limited token windows to 2^31 - 1.

- Fixed definition of "less than" for tokens.

- NOOP commands now trigger Operation Replies.

- Renamed Authentication options to Authentication Data options.
- Authentication Data options are no longer mandatory.
- Authentication methods are now advertised via options.
- Shifted some Request fields.
- Option range for vendor-specific options.
- Socket options.
- Password authentication.
- Salt options.

draft-01

- Added this section.
- Support for idempotent commands.
- Removed version numbers from operation replies.
- Request port number for SOCKS over TLS. Deprecate encryption/ encapsulation within SOCKS.
- Added Version Mismatch Replies.
- Renamed the AUTH command to NOOP.
- Shifted some fields to make requests and operation replies easier to parse.

2. Requirements language

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

3. Mode of operation

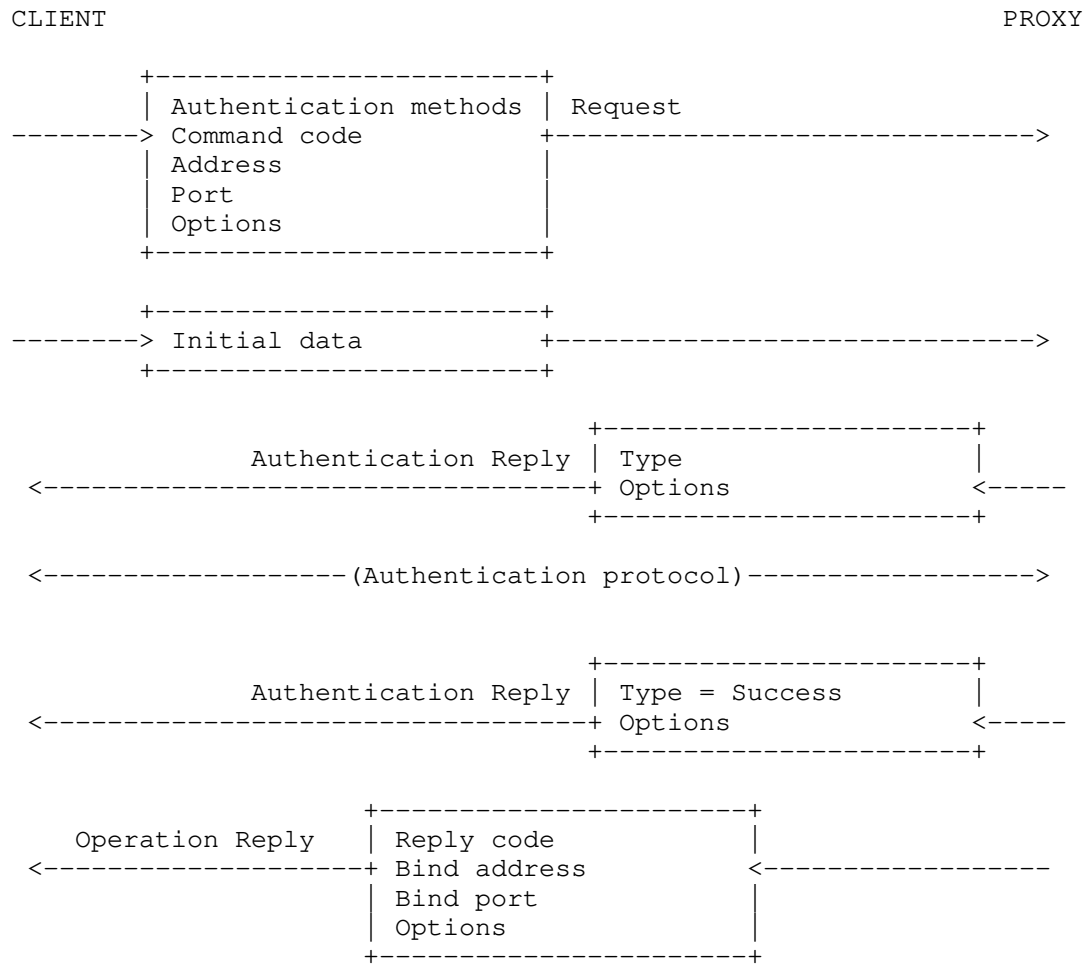


Figure 1: The SOCKS version 6 protocol message exchange

When a TCP-based client wishes to establish a connection to a server, it must open a TCP connection to the appropriate SOCKS port on the SOCKS proxy. The client then enters a negotiation phase, by sending the request in Figure 1, that contains, in addition to fields present in SOCKS 5 [RFC1928], fields that facilitate low RTT usage and faster authentication negotiation.

Next, the server sends an authentication reply. If the request did not contain the necessary authentication information, the proxy indicates an authentication method that must proceed. This may trigger a longer authentication sequence that could include tokens

for ulterior faster authentications. The part labeled "Authentication protocol" is specific to the authentication method employed and is not expected to be employed for every connection between a client and its proxy server. The authentication protocol typically takes up 1 RTT or more.

If the authentication is successful, an operation reply is generated by the proxy. It indicates whether the proxy was successful in creating the requested socket or not.

In the fast case, when authentication is properly set up, the proxy attempts to create the socket immediately after the receipt of the request, thus achieving an operational conection in one RTT (provided TFO functionality is available at the client, proxy, and server).

4. Requests

The client starts by sending a request to the proxy.

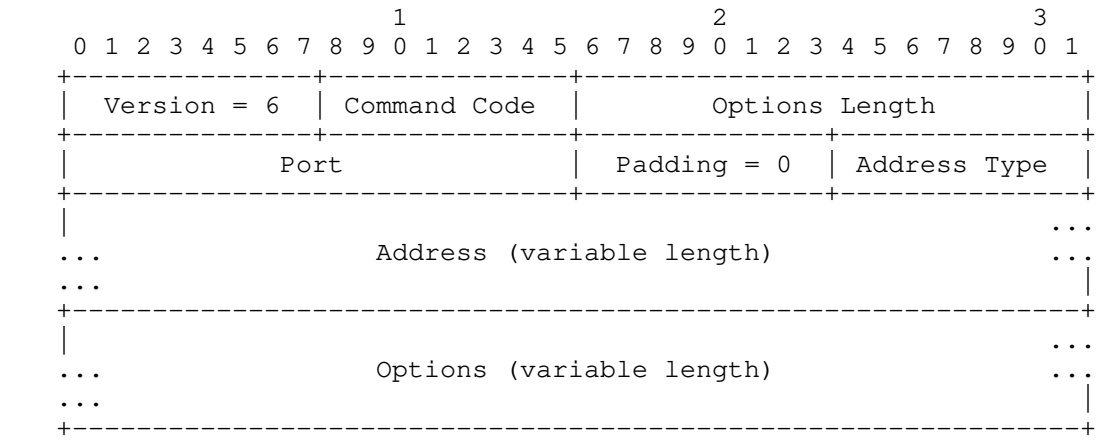


Figure 2: SOCKS 6 Request

- Version: 6
- Command Code:
  - 0x00 NOOP: does nothing.
  - 0x01 CONNECT: requests the establishment of a TCP connection. TFO MUST NOT be used unless explicitly requested.
  - 0x02 BIND: requests the establishment of a TCP port binding.

  * 0x03 UDP ASSOCIATE: requests a UDP port association.

- Address Type:
  * 0x01: IPv4
  * 0x03: Domain Name
  * 0x04: IPv6
- Address: this field's format depends on the address type:
  * IPv4: a 4-byte IPv4 address
  * Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated, but padded by NUL characters, if needed.
  * IPv6: a 16-byte IPv6 address
- Port: the port in network byte order.
- Padding: set to 0
- Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- Options: see Section 8.

The Address and Port fields have different meanings based on the Command Code:

- NOOP: The fields have no meaning. The Address Type field MUST be either 0x01 (IPv4) or 0x04 (IPv6). The Address and Port fields MUST be 0.
- CONNECT: The fields signify the address and port to which the client wishes to connect.
- BIND, UDP ASSOCIATE: The fields indicate the desired bind address and port. If the client does not require a certain address, it can set the Address Type field to 0x01 (IPv4) or 0x04 (IPv6), and the Address field to 0. Likewise, if the client does not require a certain port, it can set the Port field to 0.

Clients can advertise their supported authentication methods by including an Authentication Method Advertisement option (see Section 8.2).

5. Version Mismatch Replies

Upon receipt of a request starting with a version number other than 6, the proxy sends the following response:

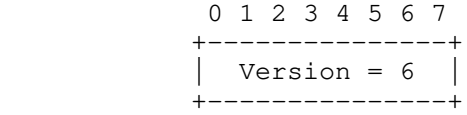


Figure 3: SOCKS 6 Version Mismatch Reply

- Version: 6

A client MUST close the connection after receiving such a reply.

6. Authentication Replies

Upon receipt of a valid request, the proxy sends an Authentication Reply:

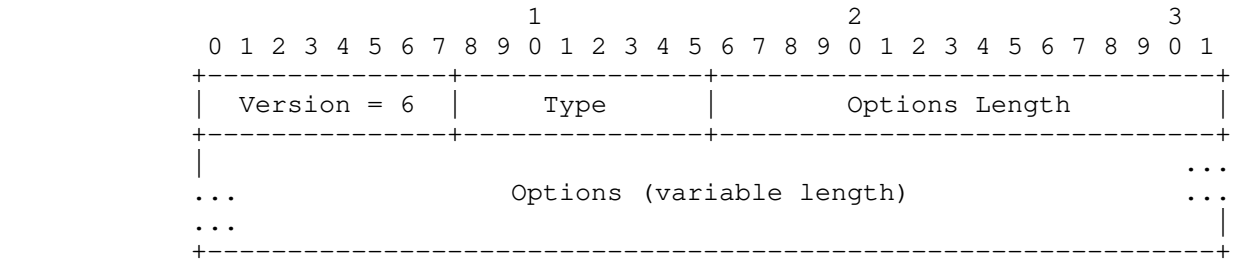


Figure 4: SOCKS 6 Authentication Reply

- Version: 6
- Type:
  - 0x00: authentication successful.
  - 0x01: authentication failed.
- Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- Options: see Section 8.

If the server signals that the authentication has failed and does not signal that any authentication negotiation can continue (via an Authentication Method Selection option), the client MUST close the connection.

The client and proxy begin a method-specific negotiation. During such negotiations, the proxy MAY supply information that allows the client to authenticate a future request using an Authentication Data option. Application data is not subject to any encryption negotiated during this phase. Descriptions of such negotiations are beyond the scope of this memo.

When the negotiation is complete (either successfully or unsuccessfully), the proxy sends a second Authentication Reply. The second Authentication Reply MUST NOT allow for further negotiations.

7. Operation Replies

After the authentication negotiations are complete, the proxy sends an Operation Reply:

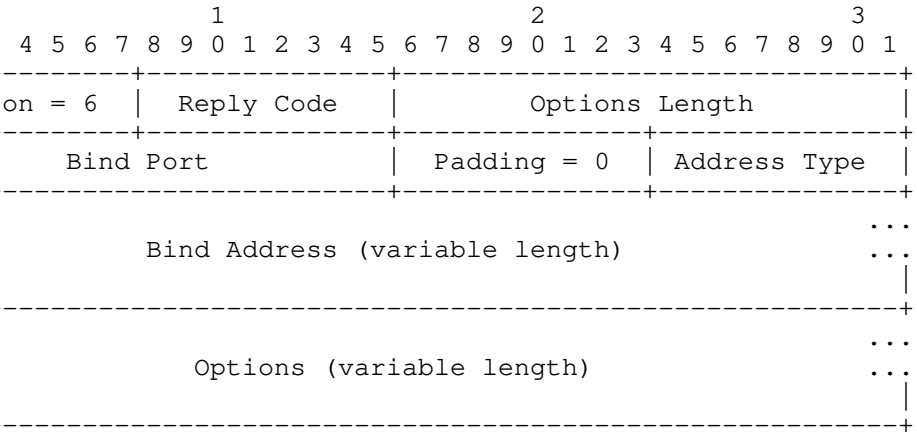


Figure 5: SOCKS 6 Operation Reply

- Version: 6
- Reply Code:
  - 0x00: Succes
  - 0x01: General SOCKS server failure

    * 0x02: Connection not allowed by ruleset
    * 0x03: Network unreachable
    * 0x04: Host unreachable
    * 0x05: Connection refused
    * 0x06: TTL expired
    * 0x07: Command not supported
    * 0x08: Address type not supported
    * 0x09: Connection attempt timed out

- Bind Port: the proxy bound port in network byte order.
- Padding: set to 0
- Address Type:
    * 0x01: IPv4
    * 0x03: Domain Name
    * 0x04: IPv6
- Bind Address: the proxy bound address in the following format:
    * IPv4: a 4-byte IPv4 address
    * Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated, but padded by NUL characters, if needed.
    * IPv6: a 16-byte IPv6 address
- Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- Options: see Section 8.

Proxy implementations MAY support any subset of the client commands listed in Section 4.

If the proxy returns a reply code other than "Success", the client MUST close the connection.

If the client issued an NOOP command, the client MUST close the connection after receiving the Operation Reply.

7.1. Handling CONNECT

In case the client has issued a CONNECT request, data can now pass.

7.2. Handling BIND

In case the client has issued a BIND request, it must wait for a second Operation reply from the proxy, which signifies that a host has connected to the bound port. The Bind Address and Bind Port fields contain the address and port of the connecting host. Afterwards, application data may pass.

7.3. Handling UDP ASSOCIATE

Proxies offering UDP functionality may be configured with a UDP port used for relaying UDP datagrams to and from the client, and/or a port used for relaying datagrams over DTLS.

Following a successful Operation Reply, the client and the proxy begin exchanging messages with the following header:

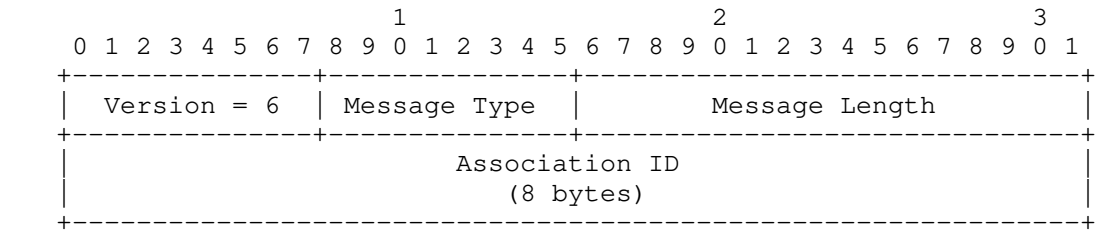


Figure 6: UDP Association Header

- Message Type:
  - 0x01: Association Initialization
  - 0x02: Association Confirmation
  - 0x03: Datagram
  - 0x04: Error
- Message Length: the total length of the message

o Association ID: the identifier of the UDP association

First, the proxy picks an Association ID sends a an Association Initialization message:

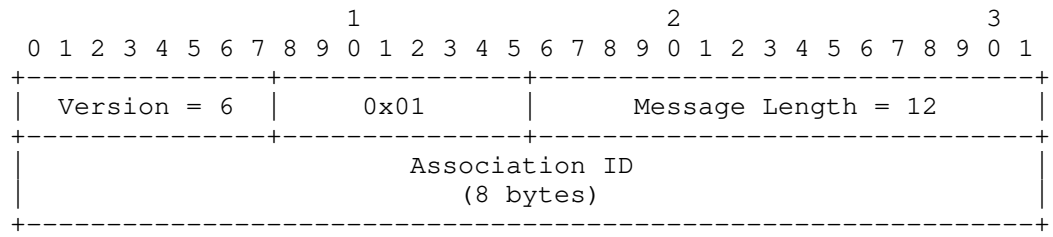


Figure 7: UDP Association Initialization

Proxy implementations SHOULD generate Association IDs randomly or pseudo-randomly.

Clients may start sending datagrams to the proxy either:

o over the TCP connection,

o in plaintext, using the proxy's configured UDP port(s), or

o over an established DTLS session.

A client's datagrams are prefixed by a Datagram Header, indicating the remote host's address and port:

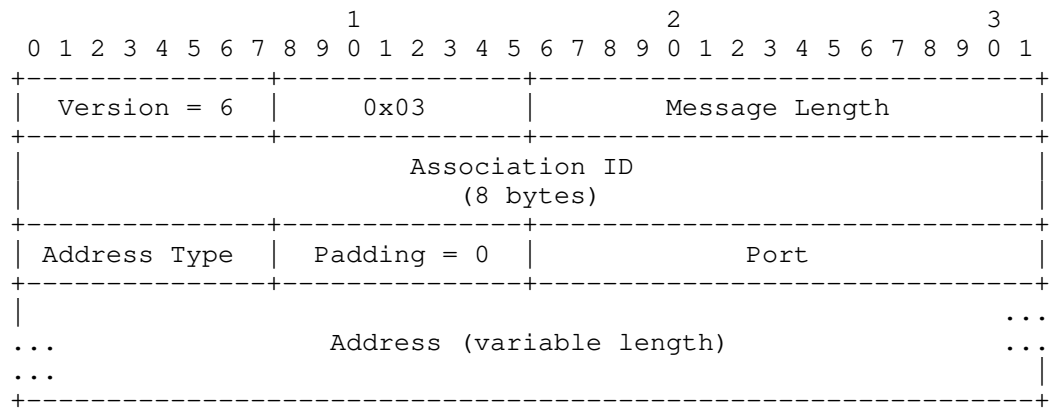


Figure 8: Datagram Header

- Version: 0x06
- Association ID: the identifier of the UDP association
- Address Type:
  - 0x01: IPv4
  - 0x03: Domain Name
  - 0x04: IPv6
- Address: this field's format depends on the address type:
  - IPv4: a 4-byte IPv4 address
  - Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated.
  - IPv6: a 16-byte IPv6 address
- Port: the port in network byte order.

Datagrams sent over UDP MAY be padded with arbitrary data (i. e., the Message Length MAY be smaller than the actual UDP/DTLS payload). Client and proxy implementations MUST ignore the padding. If the Message Length is larger than the size of the UDP or DTLS payload, the message MUST be silently ignored.

Following the receipt of the first datagram from the client, the proxy makes a one-way mapping between the Association ID and:

- the TCP connection, if it was received over TCP, or
- the 5-tuple of the UDP conversation, if the datagram was received over plain UDP, or
- the DTLS connection, if the datagram was received over DTLS. The DTLS connection is identified either by its 5-tuple, or some other mechanism, like [I-D.ietf-tls-dtls-connection-id].

The proxy SHOULD close the TCP connection if the initial datagram is not received after a timeout.

Further datagrams carrying the same Association ID, but not matching the established mapping, are silently dropped.

The proxy then sends an UDP Association Confirmation message over the TCP connection with the client:

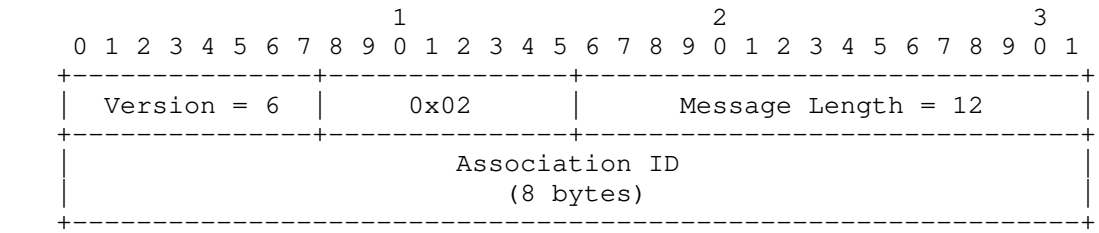


Figure 9: UDP Association Confirmation

Following the confirmation message, UDP packets bound for the proxy's bind address and port are relayed to the client, also prefixed by a Datagram Header.

The UDP association remains active for as long as the TCP connection between the client and the proxy is kept open.

7.3.1. Proxying UDP servers

Under some circumstances (e.g. when hosting a server), the SOCKS client expects the remote host to send UDP datagrams first. As such, the SOCKS client must trigger a UDP Association Confirmation without having the proxy relay any datagrams on its behalf.

To that end, it sends an empty datagram prefixed by a Datagram Header with an IP address and port consisting of zeroes. If it is using UDP, the client SHOULD resend the empty datagram if an UDP Association Confirmation is not received after a timeout.

7.3.2. Proxying multicast traffic

The use of multicast addessses is permitted for UDP traffic only.

7.3.3. Reporting ICMP Errors

If a client has opted in (see Section 8.1.8), the proxy MAY relay information contained in some ICMP Error packets. The message format is as follows:

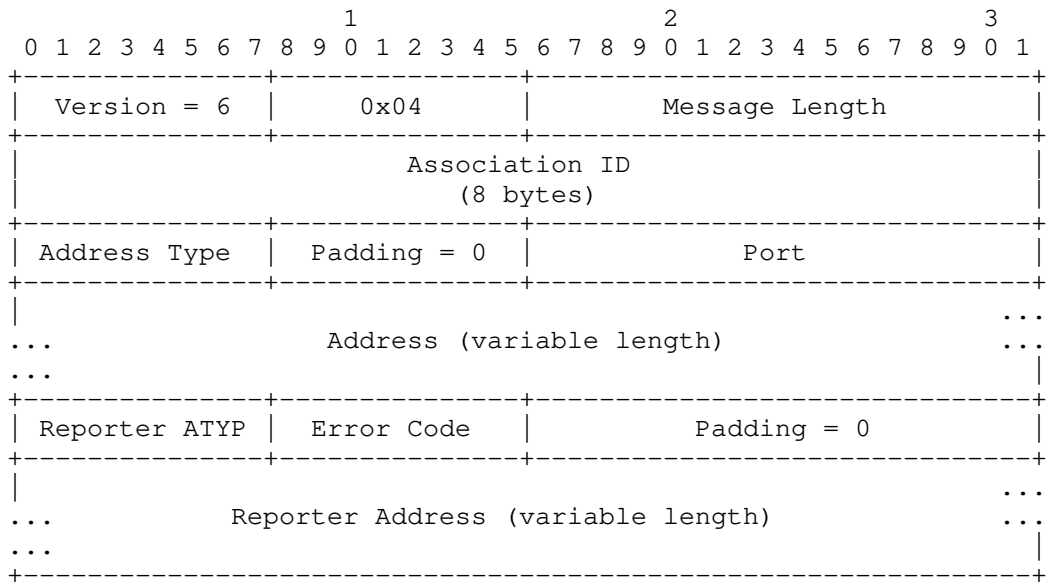


Figure 10: Datagram Error Message

- Address: The destination address of the IP header contained in the ICMP payload
- Address Type: Either 0x01 (IPv4) or 0x04 (IPv6)
- Port: The destination port of the UDP header contained in the ICMP payload
- Reporter Address: The IP address of the host that issued the ICMP error
- Reporter Address Type (ATYP): Either 0x01 (IPv4) or 0x04 (IPv6)
- Error code:
  - 0x01: Network unreachable
  - 0x02: Host unreachable
  - 0x03: TTL expired
  - 0x04: Datagram too big (IPv6 only)

It is possible for ICMP Error packets to be spurious, and not be related to any UDP packet that was sent out. The proxy is not required to check the validity of ICMP Error packets before reporting them to the client.

Clients MUST NOT send Datagram Error messages to the proxy. Proxies MUST NOT send Error messages unless the clients have opted in.

8. SOCKS Options

SOCKS options have the following format:

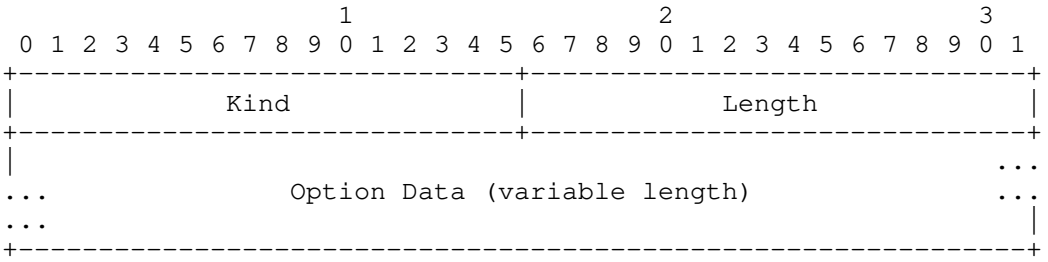


Figure 11: SOCKS 6 Option

- Kind: Allocated by IANA. (See Section 15.)
- Length: The total length of the option. MUST be a multiple of 4.
- Option Data: The contents are specific to each option kind.

Unless otherwise noted, client and proxy implementations MAY omit supporting any of the options described in this document. Upon encountering an unsupported option, a SOCKS endpoint MUST silently ignore it.

8.1. Stack options

Stack options can be used by clients to alter the behavior of the protocols on top of which SOCKS is running, as well the protcols used by the proxy to communicate with the remote host (i.e. IP, TCP, UDP). A Stack option can affect either the proxy's protocol on the client-proxy leg or on the proxy-remote leg. Clients can only place Stack options inside SOCKS Requests.

Proxies MAY choose not to honor any Stack options sent by the client.

Proxies include Stack options in their Operation Replies to signal their behavior, and MUST do so for every supported Stack option sent by the client. Said options MAY also be unsolicited, i. e. the proxy MAY send them to signal behaviour that was not explicitly requested by the client.

If a particular Stack option is unsupported, the proxy MUST silently ignore it.

In case of UDP ASSOCIATE, the stack options refer to the UDP traffic relayed by the proxy.

Stack options that are part of the same message MUST NOT contradict one another or contain duplicate information.

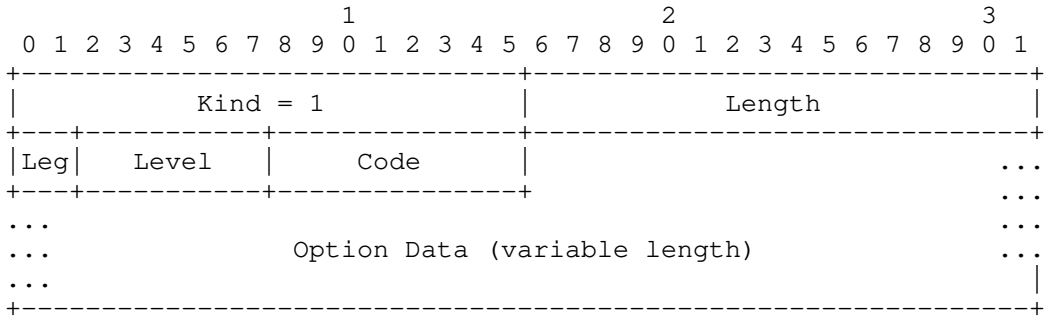


Figure 12: Stack Option

- Leg:
  - 1: Client-Proxy Leg
  - 2: Proxy-Remote Leg
  - 3: Both Legs
- Level:
  - 1: IP: options that apply to either IPv4 or IPv6
  - 2: IPv4
  - 3: IPv6
  - 4: TCP

* 5: UDP

- Code: Option code

- Option Data: Option-specific data

8.1.1. IP TOS options

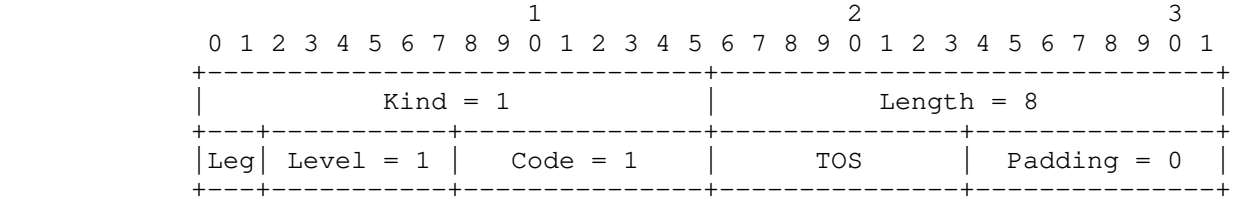


Figure 13: IP TOS Option

- TOS: The IP TOS code

The client can use IP TOS options to request that the proxy use a certain value for the IP TOS field. Likewise, the proxy can use IP TOS options to advertise the TOS values being used.

8.1.2. Happy Eyeballs options

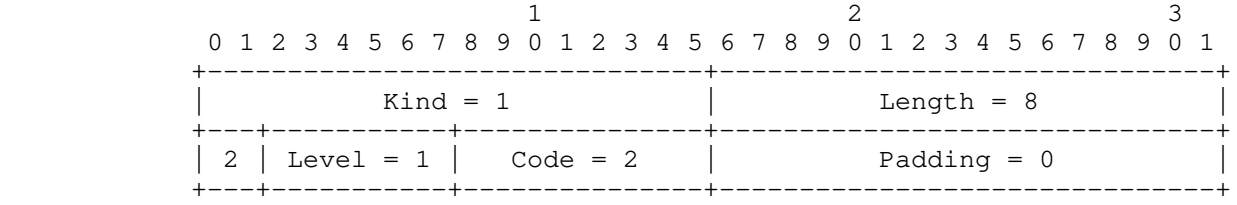


Figure 14: Happy Eyeballs Option

This memo provides enough features for clients to implement a mechanism analogous to Happy Eyeballs [RFC8305] over SOCKS. However, when the delay between the client and the proxy, or the proxy's vantage point, is high, doing so can become impractical or inefficient.

In such cases, the client can instruct the proxy to employ the Happy Eyeballs technique on its behalf when connecting to a remote host.

The client MUST supply a Domain Name as part of its Request. Otherwise, the proxy MUST silently ignore the option.

TODO: Figure out which knobs to include.

8.1.3. TTL options

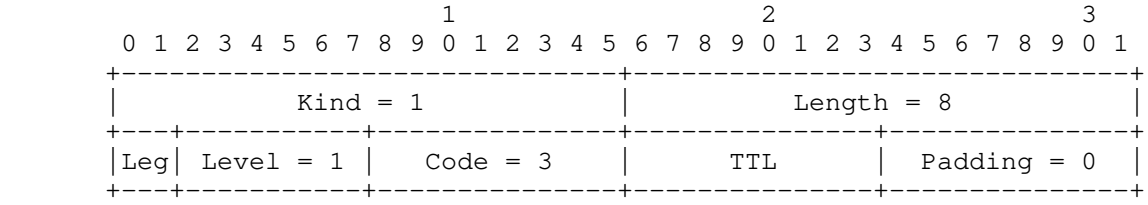


Figure 15: IP TTL Option

- TTL: The IP TTL or Hop Limit

8.1.4. No Fragmentaion options

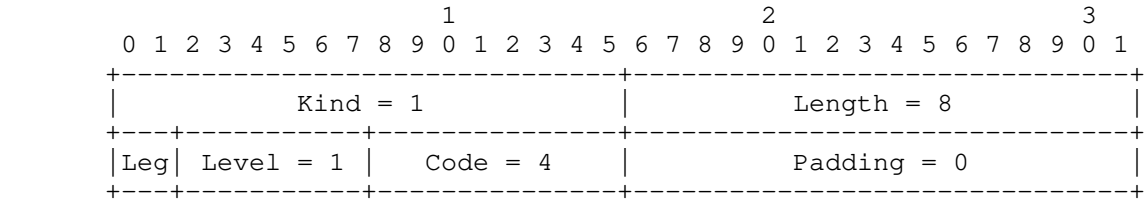


Figure 16: No Fragmentaion Option

A No Fragmentation option instructs the proxy to avoid IP fragmentation. In the case of IPv4, this also entails setting the DF bit on outgoing packets.

8.1.5. TFO options

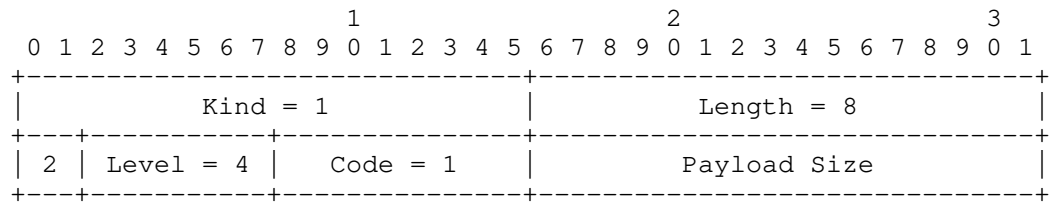


Figure 17: TFO Option

- Payload Size: The desired payload size of the TFO SYN. Ignored in case of a BIND command.

If a SOCKS Request contains a TFO option, the proxy SHOULD attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command. Otherwise, the proxy MUST NOT attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command.

In case of a CONNECT command, the client can indicate the desired payload size of the SYN. If the field is 0, the proxy can use an arbitrary payload size. If the field is non-zero, the proxy MUST NOT use a payload size larger than the one indicated. The proxy MAY use a smaller payload size than the one indicated.

8.1.6. Multipath options

In case of a CONNECT or BIND command, the client can inform the proxy whether MPTCP is desired on the proxy-remote leg by sending a Multipath option.

Conversely, the proxy can use a Multipath option to convey the following information:

- whether or not the connection uses MPTCP or not, when replying to a CONNECT command, or in the second Operation reply to a BIND command, or

- whether an MPTCP connection will be accepted, when first replying to a BIND command.

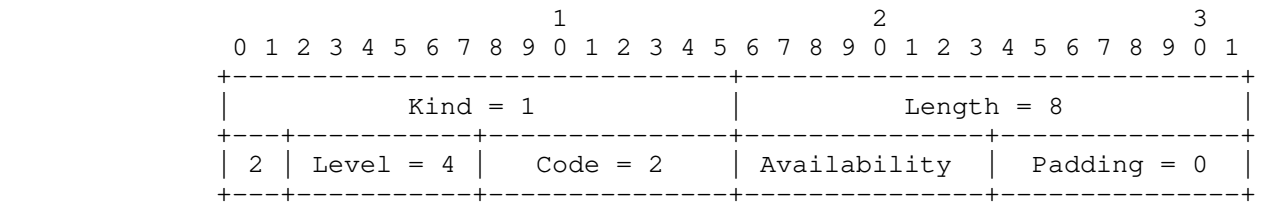


Figure 18: Multipath Option

- Availability:
  - 0x01: MPTCP is not desired or available
  - 0x02: MPTCP is desired or available

In the absence of such an option, the proxy SHOULD NOT enable MPTCP.

8.1.7. Listen Backlog options

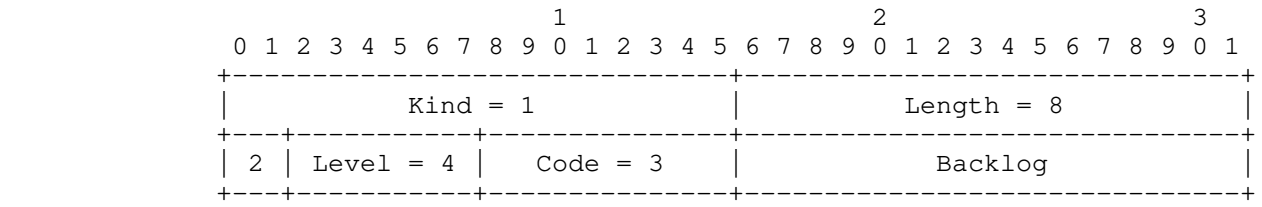


Figure 19: Listen Backlog Option

- Backlog: The length of the listen backlog.

The default behavior of the BIND does not allow a client to simultaneously handle multiple connections to the same bind address. A client can alter BIND's behavior by adding a TCP Listen Backlog Option to a BIND Request, provided that the Request is part of a Session.

In response, the proxy sends a TCP Listen Backlog Option as part of the Operation Reply, with the Backlog field signalling the actual backlog used. The proxy SHOULD NOT use a backlog longer than requested.

Following the successful negotiation of a backlog, the proxy listens for incoming connections for as long as the initial connection stays

open. The initial connection is not used to relay data between the client and a remote host.

To accept connections, the client issues further BIND Requests using the bind address and port supplied by the proxy in the initial Operation Reply. Said BIND requests must belong to the same Session as the original Request.

If no backlog is issued, the proxy signals a backlog length of 0, and BIND's behavior remains unaffected.

8.1.8. UDP Error options

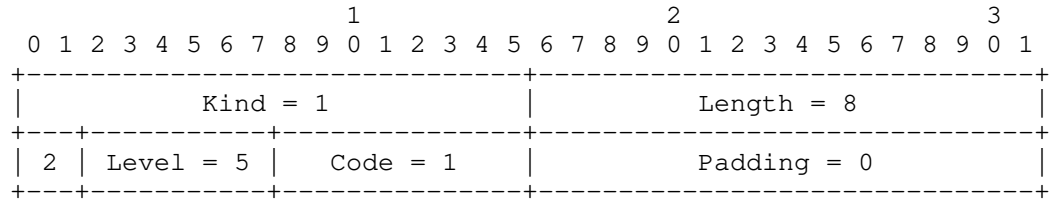


Figure 20: UDP Error Option

Clients can use this option to turn on error reporting for a particular UDP association. See Section 7.3.3.

8.1.9. Port Parity options

The RTP specification [RFC3550] recommends runnng the protocol on consecutive UDP ports, where the even port is the lower of the two.

SOCKS clients can specify the desired port parity when issuing a UDP ASSOCIATE command, and request that the port's counterpart be reserved.

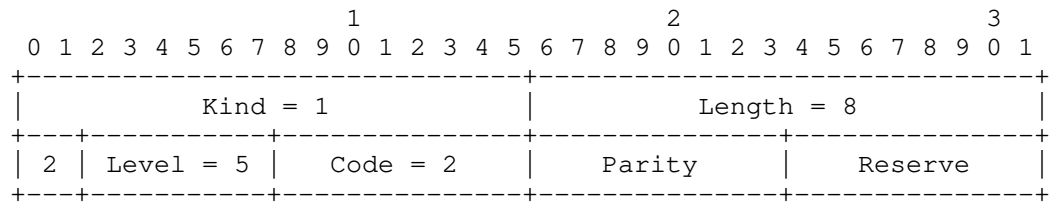


Figure 21: Port Parity Option

o Parity:

    * 0x00: Even

    * Ox01: Odd

  o Reserve: whether or not to reserve the port's counterpart

    * 0x00: Don't reserve

    * 0x01: Reserve

  If the UDP ASSOCIATE request does not have the Port field set to 0 (indicating that an arbitrary port can be chosen), the proxy MUST silently ignore this option.

  A port's counterpart is determined as follows:

  o for even ports, it is the next higher port and

  o for odd ports, it is the next lower port.

  If the proxy can not or will not comply with the requested parity, it does so silently, and also does not reserve the allocated port's counterpart. If it can not or will not comply with the reservation request, the reply MUST have its Reserve field set to 0.

  Port reservations are in place until either:

  o the original association ends, or

  o an association involving the reserved port is made.

  An association involving a reserved port can only be made if a client explicitly requests said port. Further, if the original association is part of a session (see Section 8.4), the reserved port can only be claimed from within the same session.

8.2. Authentication Method options

  A client that is willing to go through the authentication phase MUST include an Authentication Method Advertisement option in its Request. In case of a CONNECT Request, the option is also used to specify the amount of initial data supplied before any method-specific authentication negotiations take place.

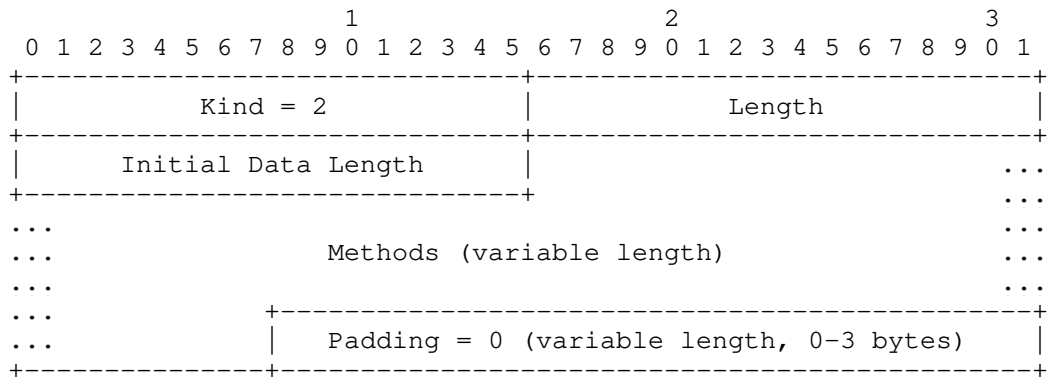


Figure 22: Authentication Method Advertisement Option

- Initial Data Size: A two-byte number in network byte order. In case of CONNECT, this is the number of bytes of initial data that are supplied by the client immediately following the Request. This number MUST NOT be larger than 2^14.

- Methods: One byte per advertised method. Method numbers are assigned by IANA.

- Padding: A minimally-sized sequence of zeroes, such that the option length is a multiple of 4. Note that 0 coincides with the value for "No Authentication Required".

Clients MUST support the "No authentication required" method. Clients SHOULD omit advertising the "No authentication required" option.

The proxy indicates which authentication method must proceed by sending an Authentication Method Selection option in the corresponding Authentication Reply:

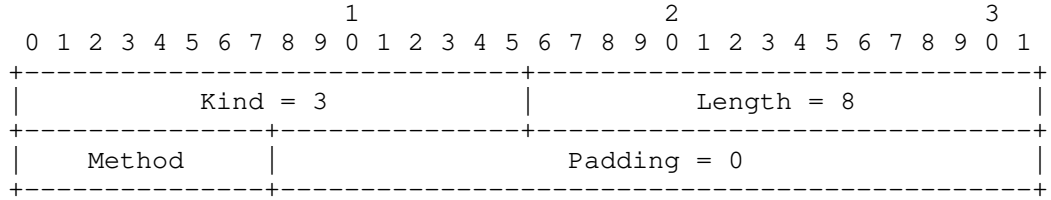


Figure 23: Authentication Method Selection Option

- Method: The selected method.

If the proxy selects "No Acceptable Methods", the client MUST close the connection.

If authentication is successful via some other means, or not required at all, the proxy silently ignores the Authentication Method Advertisement option.

8.3. Authentication Data options

Authentication Data options carry method-specific authentication data. They can be part of SOCKS Requests and Authentication Replies.

Authentication Data options have the following format:

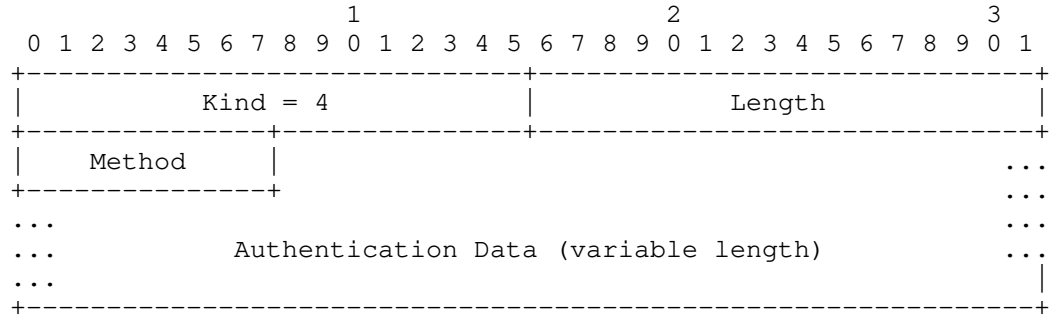


Figure 24: Authentication Data Option

- Method: The number of the authentication method. These numbers are assigned by IANA.

- Authentication Data: The contents are specific to each method.

Clients MUST only place one Authentication Data option per authentication method.

8.4. Session options

Clients and proxies can establish SOCKS sessions, which span one or more Requests. All session-related negotiations are done via Session Options, which are placed in Requests and Authentication Replies by the client and, respectively, by the proxy.

Client and proxy implementations MUST either support all Session Option Types, or none.

8.4.1. Session initiation

A client can initiate a session by sending a Session Request Option:

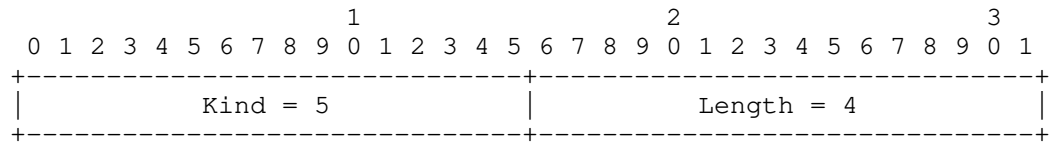


Figure 25: Session Request Option

The proxy then replies with a Session ID Option in the successful Operation Reply:

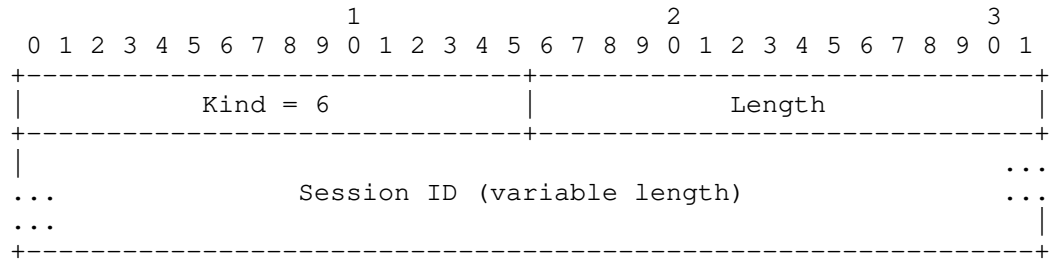


Figure 26: Session ID Option

- Session ID: An opaque sequence of bytes specific to the session. The size MUST be a multiple of 4. MUST NOT be empty.

The Session ID serves to identify the session and is opaque to the client.

The credentials, or lack thereof, used to initiate the session are tied to the session.

The SOCKS Request that initiated the session is considered part of the session. A client MUST NOT attempt to initiate a session from within a different session.

If the proxy can not or will not honor the Session Request, it does so silently.

8.4.2. Further SOCKS Requests

Any further SOCKS Requests that are part of the session MUST include a Session ID Option (as seen in Figure 26). The proxy MUST silently ignore any authentication attempt in the Request, and MUST NOT require any authentication.

The proxy then replies by placing a Session OK option in the successful Authentication Reply:

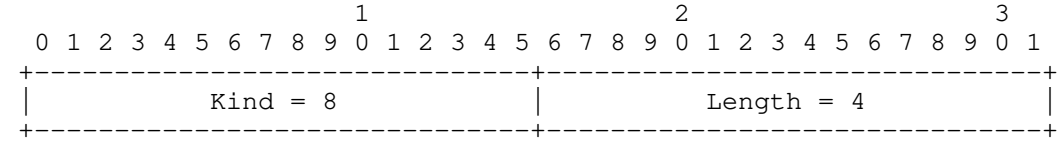


Figure 27: Session OK Option

If the Session ID is invalid, the first Authentication Reply MUST signal that authentication failed and can not continue (by setting the Type field to 0x01). Further, it SHALL contain a Session Invalid option:

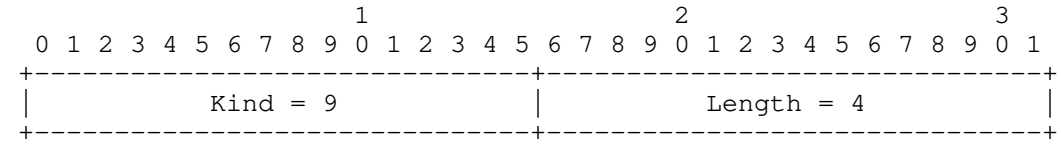


Figure 28: Session Invalid Option

8.4.3. Tearing down the session

Proxies can, at their discretion, tear down a session and free all associated state. Proxy implementations SHOULD feature a timeout mechanism that destroys sessions after a period of inactivity. When a session is terminated, the proxy MAY close all connections associated with said session.

Clients can signal that a session is no longer needed, and can be torn down, by sending a Session Teardown option in addition to the Session ID option:

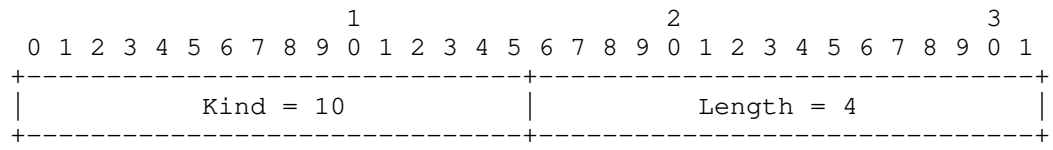


Figure 29: Session Teardown Option

After sending such an option, the client MUST assume that the session is no longer valid. The proxy MUST treat the session-terminating request as if it were not part of any session.

8.5. Idempotence options

To protect against duplicate SOCKS Requests, clients can request, and then spend, idempotence tokens. A token can only be spent on a single SOCKS request.

Tokens are 4-byte unsigned integers in a modular 4-byte space. Therefore, if x and y are tokens, x is less than y if $0 < (y - x) < 2^{31}$ in unsigned 32-bit arithmetic.

Proxies grant contiguous ranges of tokens called token windows. Token windows are defined by their base (the first token in the range) and size.

All token-related operations are done via Idempotence options.

Idempotence options are only valid in the context of a SOCKS Session. If a SOCKS Request is not part of a Session (either by supplying a valid Session ID or successfully initiating one via a Session Request), the proxy MUST silently ignore any Idempotence options.

Token windows are tracked by the proxy on a per-session basis. There can be at most one token window for every session and its tokens can only be spent from within said session.

Client and proxy implementations MUST either support all Idempotence Option Types, or none.

8.5.1. Requesting a token window

A client can obtain a window of tokens by sending an Idempotence Request option as part of a SOCKS Request:

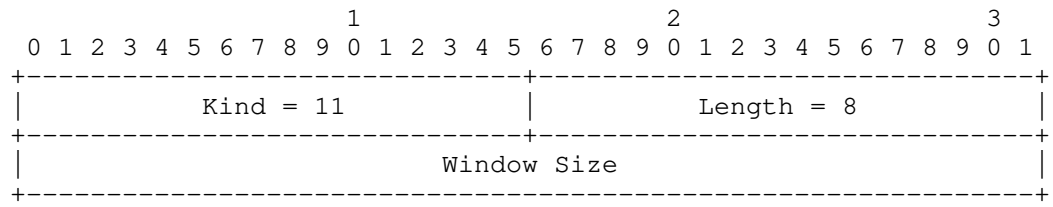


Figure 30: Token Request

- Window Size: The requested window size.

Once a token window is issued, the proxy MUST include an Idempotence Window option in all subsequent successful Authentication Replies:

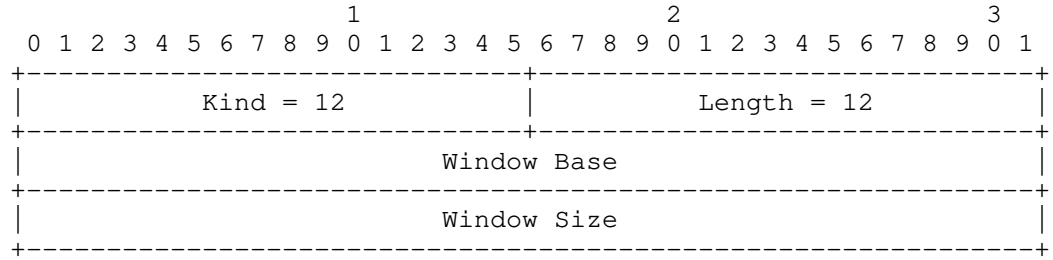


Figure 31: Idempotence Window

- Window Base: The first token in the window.

- Window Size: The window size. This value MAY differ from the requested window size. Window sizes MUST be less than 2^31. Window sizes MUST NOT be 0.

If no token window is issued, the proxy MUST silently ignore the Token Request. If there is already a token window associated with the session, the proxy MUST NOT issue a new window.

8.5.2. Spending a token

The client can attempt to spend a token by including a Idempotence Expenditure option in its SOCKS request:

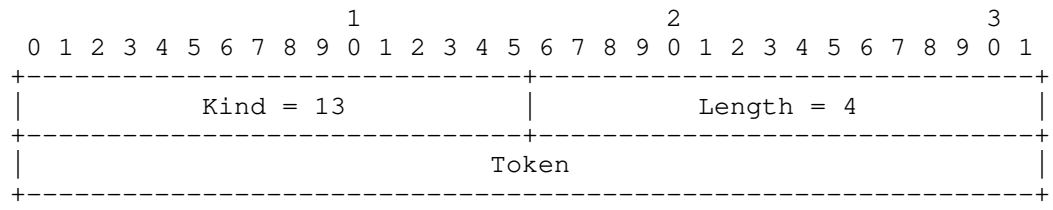


Figure 32: Idempotence Expenditure

- Kind: 13 (Idempotence Expenditure option)
- Length: 8
- Token: The token being spent.

Clients SHOULD prioritize spending the smaller tokens.

The proxy responds by sending either an Idempotence Accepted or Rejected option as part of the Authentication Reply:

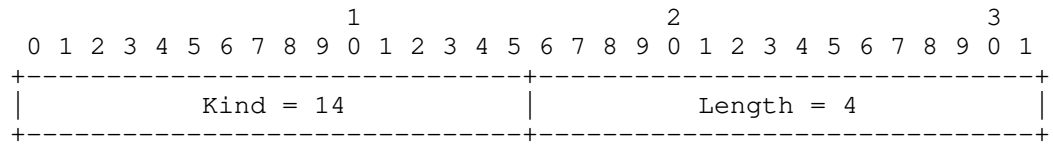


Figure 33: Idempotence Accepted

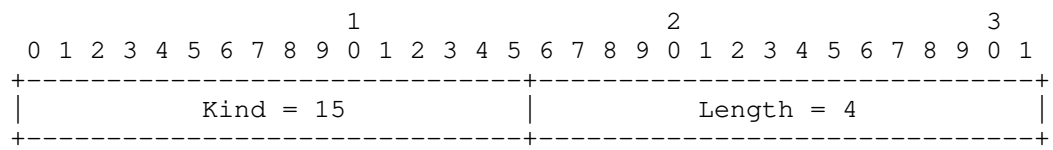


Figure 34: Idempotence Rejected

If eligible, the token is spent before attempting to honor the Request. If the token is not eligible for spending, the Authentication Reply MUST indicate failure.

8.5.3. Shifting windows

Windows can be shifted (i. e. have their base increased, while retaining their size) unilaterally by the proxy.

Proxy implementations SHOULD shift the window: * as soon as the lowest-order token in the window is spent and * when a sufficiently high-order token is spent.

Proxy implementations SHOULD NOT shift the window's base beyond the highest unspent token.

8.5.4. Out-of-order Window Advertisements

Even though the proxy increases the window's base monotonically, there is no mechanism whereby a SOCKS client can receive the Token Window Advertisements in order. As such, clients SHOULD disregard Token Window Advertisements with a Window Base less than the previously known value.

9. Username/Password Authentication

Username/Password authentication is carried out as in [RFC1929].

Clients can also attempt to authenticate by placing the Username/ Password request in an Authentication Data Option.

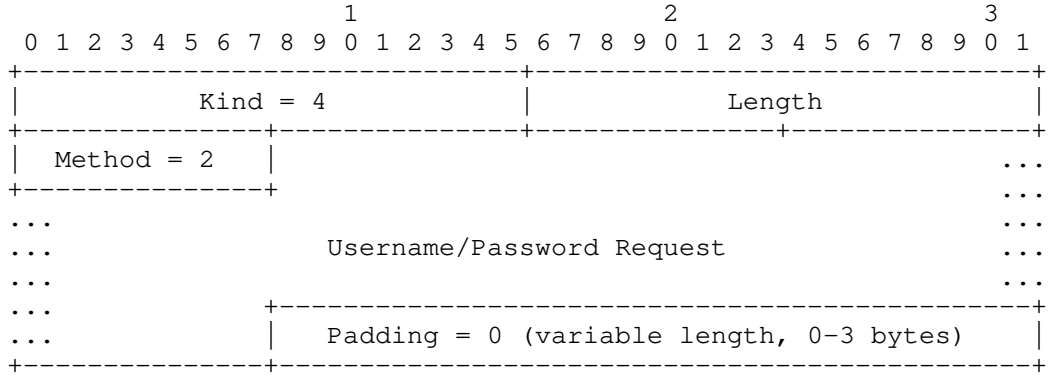


Figure 35: Password authentication via a SOCKS Option

- Username/Password Request: The Username/Password Request, as described in [RFC1929].

Proxies reply by including a Authentication Data Option in the next Authentication Reply which contains the Username/Password reply:

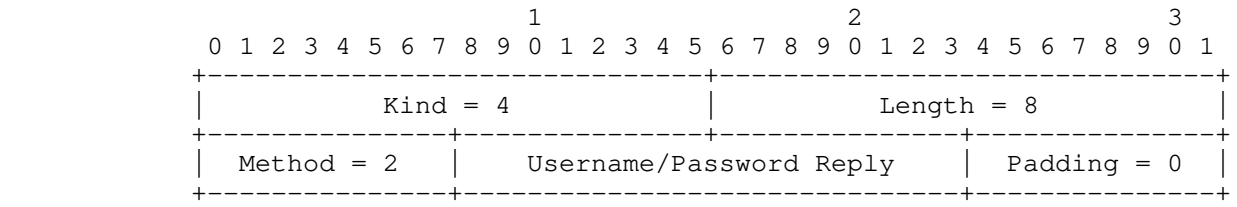


Figure 36: Reply to password authentication via a SOCKS Option

- Username/Password Reply: The Username/Password Reply, as described in [RFC1929].

10. TCP Fast Open on the Client-Proxy Leg

TFO breaks TCP semantics, causing replays of the data in the SYN's payload under certain rare circumstances [RFC7413]. A replayed SOCKS Request could itself result in a replayed connection on behalf of the client.

As such, client implementations SHOULD NOT use TFO on the client-proxy leg unless:

- The protocol running on top of SOCKS tolerates the risks of TFO, or
- The SYN's payload does not contain any application data (so that no data is replayed to the server, even though duplicate connections are still possible), or
- The client uses Idempotence Options, making replays very unlikely, or
- SOCKS is running on top of TLS and Early Data is not used.

11. False Starts

In case of CONNECT Requests, the client MAY start sending application data as soon as possible, as long as doing so does not incur the risk of breaking the SOCKS protocol.

Clients must work around the authentication phase by doing any of the following:

- If the Request does not contain an Authentication Method Advertisement option, the authentication phase is guaranteed not to happen. In this case, application data MAY be sent immediately after the Request.

- Application data MAY be sent immediately after receiving an Authentication Reply indicating success.

- When performing a method-specific authentication sequence, application data MAY be sent immediately after the last client message.

12. DNS provided by SOCKS

Clients may require information typically obtained from DNS servers, albeit from the proxy's vantage point.

While the CONNECT command can work with domain names, some clients' workflows require that addresses be resolved as a separate step prior to connecting. Moreover, the SOCKS Datagram Header, as described in Section 7.3, can be reduced in size by providing the resolved destination IP address, rather than the FQDN.

Emerging techniques may also make use of DNS to deliver server-specific information to clients. For example, Encrypted SNI [I-D.ietf-tls-esni] relies on DNS to publish encryption keys.

Proxy implementations MAY provide a default plaintext DNS service. A client looking to make use of it issues a CONNECT Request to IP address 0.0.0.0 or 0:0:0:0:0:0:0:0 on port 53. Following successful authentication, the Operation Reply MAY indicate an unspecified bind address (0.0.0.0 or ::) and port (0). The client and proxy then behave as per [RFC7766].

The service itself can be provided directly by the proxy daemon, or by proxying the client's request to a pre-configured DNS server.

If the proxy does not implement such functionality, it MAY return an error code signalling "Connection refused".

13. Security Considerations

13.1. Large requests

Given the format of the request message, a malicious client could craft a request that is in excess of 16 KB and proxies could be prone to DDoS attacks.

To mitigate such attacks, proxy implementations SHOULD be able to incrementally parse the requests. Proxies MAY close the connection to the client if:

- the request is not fully received after a certain timeout, or
- the number of options or their size exceeds an imposed hard cap.

13.2. Replay attacks

In TLS 1.3, early data (which is likely to contain a full SOCKS request) is prone to replay attacks.

While Token Expenditure options can be used to mitigate replay attacks, anything prior to the initial Token Request is still vulnerable. As such, client implementations SHOULD NOT make use of TLS early data unless the Request attempts to spend a token.

13.3. Resource exhaustion

Malicious clients can issue a large number of Session Requests, forcing the proxy to keep large amounts of state.

To mitigate this, the proxy MAY implement policies restricting the number of concurrent sessions on a per-IP or per-user basis, or barring unauthenticated clients from establishing sessions.

14. Privacy Considerations

The timing of Operation Replies can reveal some information about a proxy's recent usage:

- The DNS resolver used by the proxy may cache the answer to recent queries. As such, subsequent connection attempts to the same hostname are likely to be slightly faster, even if requested by different clients.
- Likewise, the proxy's OS typically caches TFO cookies. Repeated TFO connection attempts tend to be sped up, regardless of the client.

15. IANA Considerations

This document requests that IANA allocate 2-byte option kinds for SOCKS 6 options. Further, this document requests the following option kinds:

- Unassigned: 0

- Stack: 1
- Authentication Method Advertisement: 2
- Authentication Method Selection: 3
- Authentication Data: 4
- Session Request: 5
- Session ID: 6
- Session OK: 8
- Session Invalid: 9
- Session Teardown: 10
- Idempotence Request: 11
- Idempotence Window: 12
- Idempotence Expenditure: 13
- Idempotence Accepted: 14
- Idempotence Rejected: 15
- Resolution Request: 16
- IPv4 Resolution: 17
- IPv6 Resolution: 18
- Vendor-specific: 64512-0xFFFF

This document also requests that IANA allocate a TCP and UDP port for SOCKS over TLS and DTLS, respectively.

16. Acknowledgements

The protocol described in this draft builds upon and is a direct continuation of SOCKS 5 [RFC1928].

17. References

17.1. Normative References

[RFC1929] Leech, M., "Username/Password Authentication for SOCKS V5", RFC 1929, DOI 10.17487/RFC1929, March 1996, <https://www.rfc-editor.org/info/rfc1929>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7766] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", RFC 7766, DOI 10.17487/RFC7766, March 2016, <https://www.rfc-editor.org/info/rfc7766>.

[RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/info/rfc8305>.

17.2. Informative References

[I-D.ietf-tls-dtls-connection-id] Rescorla, E., Tschofenig, H., and T. Fossati, "Connection Identifiers for DTLS 1.2", draft-ietf-tls-dtls-connection-id-07 (work in progress), October 2019.

[I-D.ietf-tls-esni] Rescorla, E., Oku, K., Sullivan, N., and C. Wood, "TLS Encrypted Client Hello", draft-ietf-tls-esni-07 (work in progress), June 2020.

[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, DOI 10.17487/RFC1928, March 1996, <https://www.rfc-editor.org/info/rfc1928>.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <https://www.rfc-editor.org/info/rfc3550>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <https://www.rfc-editor.org/info/rfc6824>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

Authors' Addresses

Vladimir Olteanu
University Politehnica of Bucharest
313 Splaiul Independentei, Sector 6
Bucharest
Romania

Email: vladimir.olteanu@cs.pub.ro

Dragos Niculescu
University Politehnica of Bucharest
313 Splaiul Independentei, Sector 6
Bucharest
Romania

Email: dragos.niculescu@cs.pub.ro

Internet Area Working Group V. Olteanu
Internet-Draft D. Niculescu
Intended status: Standards Track University Politehnica of Bucharest
Expires: May 6, 2021 November 02, 2020

SOCKS Protocol Version 6
draft-olteanu-intarea-socks-6-11

Abstract

The SOCKS protocol is used primarily to proxy TCP connections to arbitrary destinations via the use of a proxy server. Under the latest version of the protocol (version 5), it takes 2 RTTs (or 3, if authentication is used) before data can flow between the client and the server.

This memo proposes SOCKS version 6, which reduces the number of RTTs used, takes full advantage of TCP Fast Open, and adds support for 0-RTT authentication.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on May 6, 2021.

Table of Contents

1. Introduction

Versions 4 and 5 [RFC1928] of the SOCKS protocol were developed two decades ago and are in widespread use for circuit level gateways or as circumvention tools, and enjoy wide support and usage from various software, such as web browsers, SSH clients, and proxifiers. However, their design needs an update in order to take advantage of the new features of transport protocols, such as TCP Fast Open [RFC7413], or to better assist newer transport protocols, such as MPTCP [RFC6824].

One of the main issues faced by SOCKS version 5 is that, when taking into account the TCP handshake, method negotiation, authentication, connection request and grant, it may take up to 5 RTTs for a data exchange to take place at the application layer. This is especially costly in networks with a large delay at the access layer, such as 3G, 4G, or satellite.

The desire to reduce the number of RTTs manifests itself in the design of newer security protocols. TLS version 1.3 [RFC8446] defines a zero round trip (0-RTT) handshake mode for connections if the client and server had previously communicated.

TCP Fast Open [RFC7413] is a TCP option that allows TCP to send data in the SYN and receive a response in the first ACK, and aims at obtaining a data response in one RTT. The SOCKS protocol needs to concern itself with at least two TFO deployment scenarios: First, when TFO is available end-to-end (at the client, at the proxy, and at the server); second, when TFO is active between the client and the proxy, but not at the server.

This document describes the SOCKS protocol version 6. The key improvements over SOCKS version 5 are:

- The client sends as much information upfront as possible, and does not wait for the authentication process to conclude before requesting the creation of a socket.

- The connection request also mimics the semantics of TCP Fast Open [RFC7413]. As part of the connection request, the client can supply the potential payload for the initial SYN that is sent out to the server.

- The protocol can be extended via options without breaking backward-compatibility.

- The protocol can leverage the aforementioned options to support 0-RTT authentication schemes.

1.1. Revision log

Typos and minor clarifications are not listed.

draft-11

- Changed intended status to Standards Track

- Renamed Vendor-specific option range to Experimental

- Stack options:

  * Fixed some instances where an unsupported option was indistinguishable from a case where the proxy couldn't or wouldn't honor it (offenders: Happy Eyeballs, IP Fragmentation, UDP Error, Port Parity)

  * MPTCP: changed semantics w.r.t. TCP BIND: the absence of such an option SHOULD no longer lead the proxy to refuse MPTCP

  * Port Parity: relaxed restrictions in case the client supplies a specific port

draft-10

- Removed untrusted sessions

- IP DF

- UDP relay:

  * Support ICMPv6 Too Big

  * Shifted some fields in the error messages

  * RTP support

draft-09

- Revamped UDP relay
  - Support for ICMP errors: host/net unreachable, TTL exceeded
  - Datagrams can be sent over TCP
  - Timeout for the receipt of the initial datagram
- TTL stack option (intended use: traceroute)
- Added the "Privacy Considerations" section
- SOCKS-provided DNS: the proxy may provide a valid bind address and port

draft-08

- Removed Address Resolution options
- Happy Eyeballs options
- DNS provided by SOCKS

draft-07

- All fields are now aligned.
- Eliminated version minors
- Lots of changes to options
  - 2-byte option kinds
  - Flattened option kinds/types/reply codes; also renamed some options
  - Socket options
    - Proxies MUST always answer them (Clients can probe for support)
    - MPTCP Options: expanded functionality ("please do/don't do MPTCP on my behalf")
    - MPTCP Scheduler options removed

    + Listen Backlog options: code changed to 0x03

  * Revamped Idempotence options

  * Auth data options limited to one per method

- Authentication Reply: all authentication-related information is now in the options

  * Authentication replies no longer have a field indicating the chosen auth. method

  * Method that must proceed (or whereby authentication succeeded) indicated in options

  * Username/password authentication: proxy now sends reply in option

- Removed requirements w.r.t. caching authentication methods by multihomed clients

- UDP: 8-byte association IDs

- Sessions

  * The proxy is now free to terminate ongoing connections along with the session.

  * The session-terminating request is not part of the session that it terminated.

- Address Resolution options

draft-06

- Session options

- Options now have a 2-byte length field.

- Stack options

  * Stack options can no longer contain duplicate information.

  * TFO: Better payload size semantics

  * TOS: Added missing code field.

  * MPTCP Scheduler options:

        + Removed support for round-robin

        + "Default" renamed to "Lowest latency first"

      * Listen Backlog options: now tied to sessions, instead of an authenticated user

  o Idempotence options

      * Now used in the context of a session (no longer tied to an authenticated user)

      * Idempotence options have a different codepoint: 0x05. (Was 0x04.)

      * Clarified that implementations that support Idempotence Options must support all Idempotence Option Types.

      * Shifted Idempotence Option Types by 1. (Makes implementation easier.)

  o Shrunk vendor-specific option range to 32 (down from 64).

  o Removed reference to dropping initial data. (It could no longer be done as of -05.)

  o Initial data size capped at 16KB.

  o Application data is never encrypted by SOCKS 6. (It can still be encrypted by the TLS layer under SOCKS.)

  o Messages now carry the total length of the options, rather than the number of options. Limited options length to 16KB.

  o Security Considerations

      * Updated the section to reflect the smaller maximum message size.

      * Added a subsection on resource exhaustion.

  draft-05

  o Limited the "slow" authentication negotiations to one (and Authentication Replies to 2)

  o Revamped the handling of the first bytes in the application data stream

    * False starts are now recommended. (Added the "False Start" section.)

    * Initial data is only available to clients willing to do "slow" authentication. Moved the "Initial data size" field from Requests to Authentication Method options.

    * Initial data size capped at 2^13. Initial data can no longer be dropped by the proxy.

    * The TFO option can hint at the desired SYN payload size.

- Request: clarified the meaning of the Address and Port fields.
- Better reverse TCP proxy support: optional listen backlog for TCP BIND
- TFO options can no longer be placed inside Operation Replies.
- IP TOS stack option
- Suggested a range for vendor-specific options.
- Revamped UDP functionality
    * Now using fixed UDP ports
    * DTLS support
- Stack options: renamed Proxy-Server leg to Proxy-Remote leg

draft-04

- Moved Token Expenditure Replies to the Authentication Reply.
- Shifted the Initial Data Size field in the Request, in order to make it easier to parse.

draft-03

- Shifted some fields in the Operation Reply to make it easier to parse.
- Added connection attempt timeout response code to Operation Replies.
- Proxies send an additional Authentication Reply after the authentication phase. (Useful for token window advertisements.)

- Renamed the section "Connection Requests" to "Requests"
- Clarified the fact that proxies don't need to support any command in particular.
- Added the section "TCP Fast Open on the Client-Proxy Leg"
- Options:
  - Added constants for option kinds
  - Salt options removed, along with the relevant section from Security Considerations. (TLS 1.3 Makes AEAD mandatory.)
  - Limited Authentication Data options to one per method.
  - Relaxed proxy requirements with regard to handling multiple Authentication Data options. (When the client violates the above bullet point.)
  - Removed interdependence between Authentication Method and Authentication Data options.
  - Clients SHOULD omit advertising the "No authentication required" option. (Was MAY.)
  - Idempotence options:
    - Token Window Advertisements are now part of successful Authentication Replies (so that the proxy-server RTT has no impact on their timeliness).
    - Proxies can't advertise token windows of size 0.
    - Tweaked token expenditure response codes.
    - Support no longer mandatory on the proxy side.
  - Revamped Socket options
    - Renamed Socket options to Stack options.
    - Banned contradictory socket options.
    - Added socket level for generic IP. Removed the "socket" socket level.
    - Stack options no longer use option codes from setsockopt().

+ Changed MPTCP Scheduler constants.

draft-02

- Made support for Idempotence options mandatory for proxies.
- Clarified what happens when proxies can not or will not issue tokens.
- Limited token windows to 2^31 - 1.
- Fixed definition of "less than" for tokens.
- NOOP commands now trigger Operation Replies.
- Renamed Authentication options to Authentication Data options.
- Authentication Data options are no longer mandatory.
- Authentication methods are now advertised via options.
- Shifted some Request fields.
- Option range for vendor-specific options.
- Socket options.
- Password authentication.
- Salt options.

draft-01

- Added this section.
- Support for idempotent commands.
- Removed version numbers from operation replies.
- Request port number for SOCKS over TLS. Deprecate encryption/ encapsulation within SOCKS.
- Added Version Mismatch Replies.
- Renamed the AUTH command to NOOP.
- Shifted some fields to make requests and operation replies easier to parse.

2. Requirements language

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

3. Mode of operation

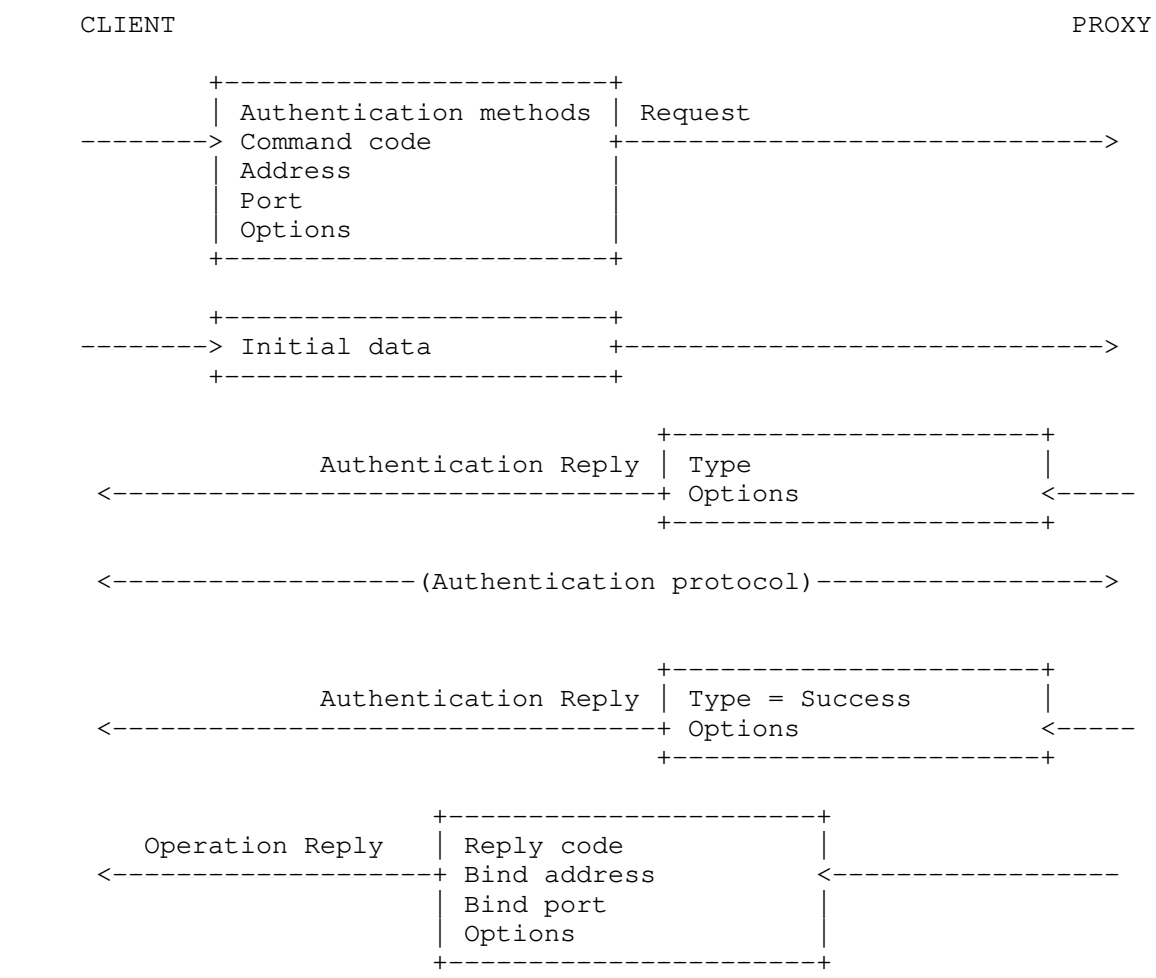


Figure 1: The SOCKS version 6 protocol message exchange

When a TCP-based client wishes to establish a connection to a server, it must open a TCP connection to the appropriate SOCKS port on the

SOCKS proxy. The client then enters a negotiation phase, by sending the request in Figure 1, that contains, in addition to fields present in SOCKS 5 [RFC1928], fields that facilitate low RTT usage and faster authentication negotiation.

Next, the server sends an authentication reply. If the request did not contain the necessary authentication information, the proxy indicates an authentication method that must proceed. This may trigger a longer authentication sequence that could include tokens for ulterior faster authentications. The part labeled "Authentication protocol" is specific to the authentication method employed and is not expected to be employed for every connection between a client and its proxy server. The authentication protocol typically takes up 1 RTT or more.

If the authentication is successful, an operation reply is generated by the proxy. It indicates whether the proxy was successful in creating the requested socket or not.

In the fast case, when authentication is properly set up, the proxy attempts to create the socket immediately after the receipt of the request, thus achieving an operational connection in one RTT (provided TFO functionality is available at the client, proxy, and server).

4. Requests

The client starts by sending a request to the proxy.

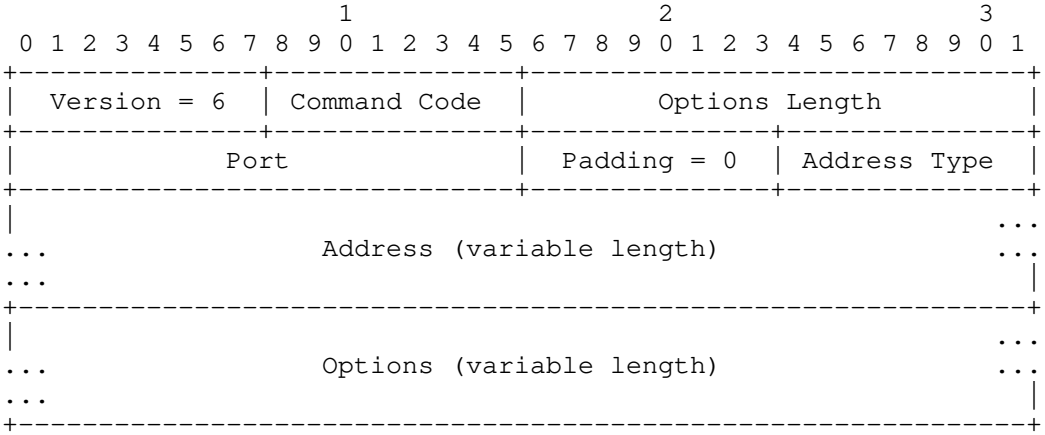


Figure 2: SOCKS 6 Request

- Version: 6
- Command Code:
  - 0x00 NOOP: does nothing.
  - 0x01 CONNECT: requests the establishment of a TCP connection. TFO MUST NOT be used unless explicitly requested.
  - 0x02 BIND: requests the establishment of a TCP port binding.
  - 0x03 UDP ASSOCIATE: requests a UDP port association.
- Address Type:
  - 0x01: IPv4
  - 0x03: Domain Name
  - 0x04: IPv6
- Address: this field's format depends on the address type:
  - IPv4: a 4-byte IPv4 address
  - Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated, but padded by NUL characters, if needed.
  - IPv6: a 16-byte IPv6 address
- Port: the port in network byte order.
- Padding: set to 0
- Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- Options: see Section 8.

The Address and Port fields have different meanings based on the Command Code:

- NOOP: The fields have no meaning. The Address Type field MUST be either 0x01 (IPv4) or 0x04 (IPv6). The Address and Port fields MUST be 0.

- CONNECT: The fields signify the address and port to which the client wishes to connect.

- BIND, UDP ASSOCIATE: The fields indicate the desired bind address and port. If the client does not require a certain address, it can set the Address Type field to 0x01 (IPv4) or 0x04 (IPv6), and the Address field to 0. Likewise, if the client does not require a certain port, it can set the Port field to 0.

Clients can advertise their supported authentication methods by including an Authentication Method Advertisement option (see Section 8.2).

5. Version Mismatch Replies

Upon receipt of a request starting with a version number other than 6, the proxy sends the following response:

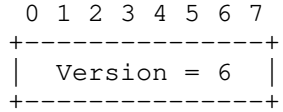


Figure 3: SOCKS 6 Version Mismatch Reply

- Version: 6

A client MUST close the connection after receiving such a reply.

6. Authentication Replies

Upon receipt of a valid request, the proxy sends an Authentication Reply:

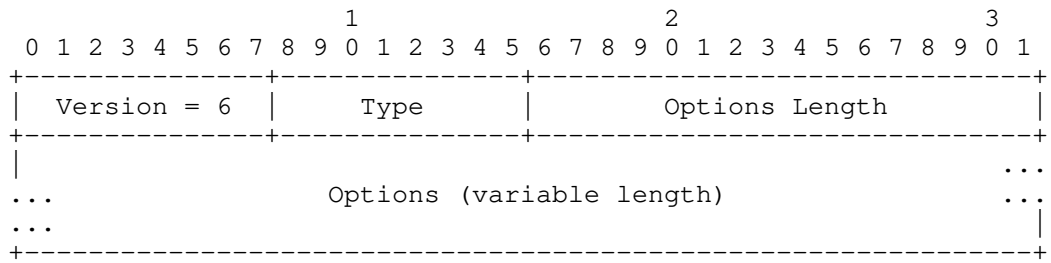


Figure 4: SOCKS 6 Authentication Reply

- Version: 6
- Type:
  - 0x00: authentication successful.
  - 0x01: authentication failed.
- Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.
- Options: see Section 8.

If the server signals that the authentication has failed and does not signal that any authentication negotiation can continue (via an Authentication Method Selection option), the client MUST close the connection.

The client and proxy begin a method-specific negotiation. During such negotiations, the proxy MAY supply information that allows the client to authenticate a future request using an Authentication Data option. Application data is not subject to any encryption negotiated during this phase. Descriptions of such negotiations are beyond the scope of this memo.

When the negotiation is complete (either successfully or unsuccessfully), the proxy sends a second Authentication Reply. The second Authentication Reply MUST NOT allow for further negotiations.

7. Operation Replies

After the authentication negotiations are complete, the proxy sends an Operation Reply:

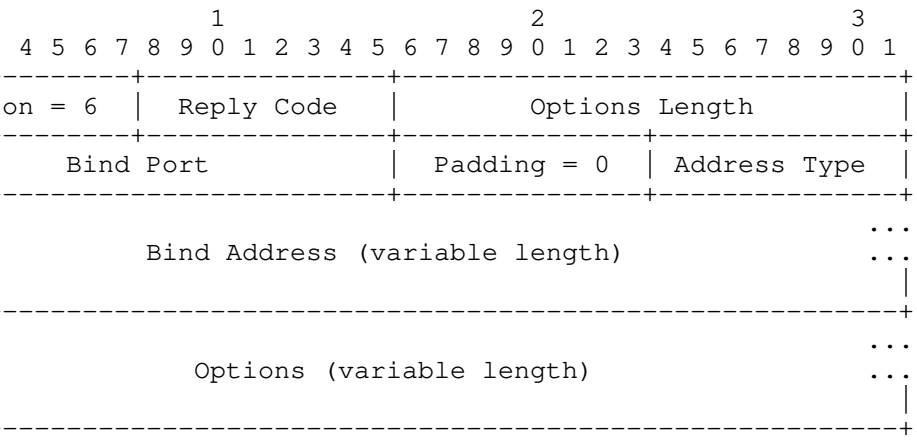


Figure 5: SOCKS 6 Operation Reply

- Version: 6
- Reply Code:
  - 0x00: Success
  - 0x01: General SOCKS server failure
  - 0x02: Connection not allowed by ruleset
  - 0x03: Network unreachable
  - 0x04: Host unreachable
  - 0x05: Connection refused
  - 0x06: TTL expired
  - 0x07: Command not supported
  - 0x08: Address type not supported
  - 0x09: Connection attempt timed out
- Bind Port: the proxy bound port in network byte order.
- Padding: set to 0
- Address Type:

    * 0x01: IPv4

    * 0x03: Domain Name

    * 0x04: IPv6

  o Bind Address: the proxy bound address in the following format:

    * IPv4: a 4-byte IPv4 address

    * Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated, but padded by NUL characters, if needed.

    * IPv6: a 16-byte IPv6 address

  o Options Length: the total size of the SOCKS options that appear in the Options field. MUST NOT exceed 16KB.

  o Options: see Section 8.

  Proxy implementations MAY support any subset of the client commands listed in Section 4.

  If the proxy returns a reply code other than "Success", the client MUST close the connection.

  If the client issued an NOOP command, the client MUST close the connection after receiving the Operation Reply.

7.1. Handling CONNECT

  In case the client has issued a CONNECT request, data can now pass.

7.2. Handling BIND

  In case the client has issued a BIND request, it must wait for a second Operation reply from the proxy, which signifies that a host has connected to the bound port. The Bind Address and Bind Port fields contain the address and port of the connecting host. Afterwards, application data may pass.

7.3. Handling UDP ASSOCIATE

  Proxies offering UDP functionality may be configured with a UDP port used for relaying UDP datagrams to and from the client, and/or a port used for relaying datagrams over DTLS.

Following a successful Operation Reply, the client and the proxy begin exchanging messages with the following header:

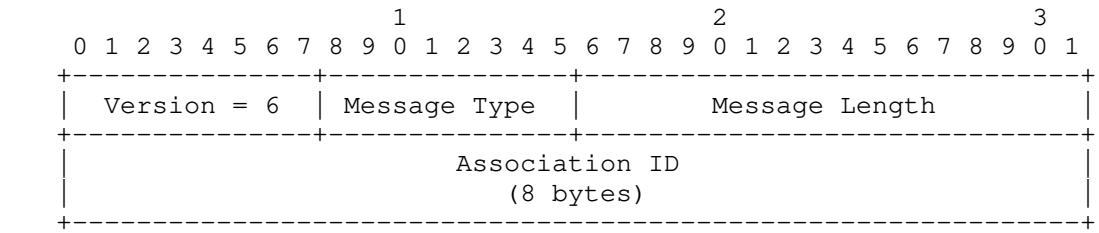


Figure 6: UDP Association Header

- Message Type:
  - 0x01: Association Initialization
  - 0x02: Association Confirmation
  - 0x03: Datagram
  - 0x04: Error
- Message Length: the total length of the message
- Association ID: the identifier of the UDP association

First, the proxy picks an Association ID sends a an Association Initialization message:

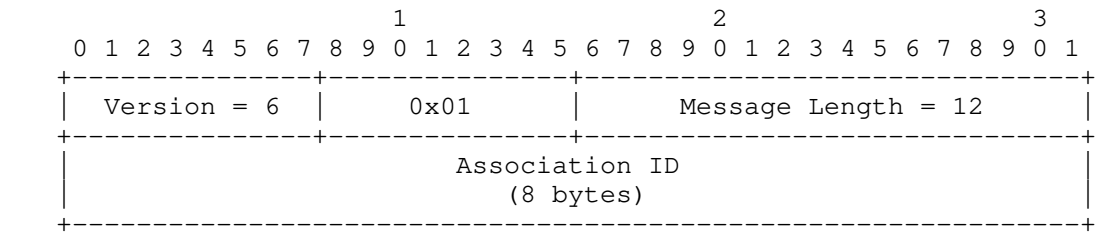


Figure 7: UDP Association Initialization

Proxy implementations SHOULD generate Association IDs randomly or pseudo-randomly.

Clients may start sending datagrams to the proxy either:

- over the TCP connection,
- in plaintext, using the proxy's configured UDP port(s), or
- over an established DTLS session.

A client's datagrams are prefixed by a Datagram Header, indicating the remote host's address and port:

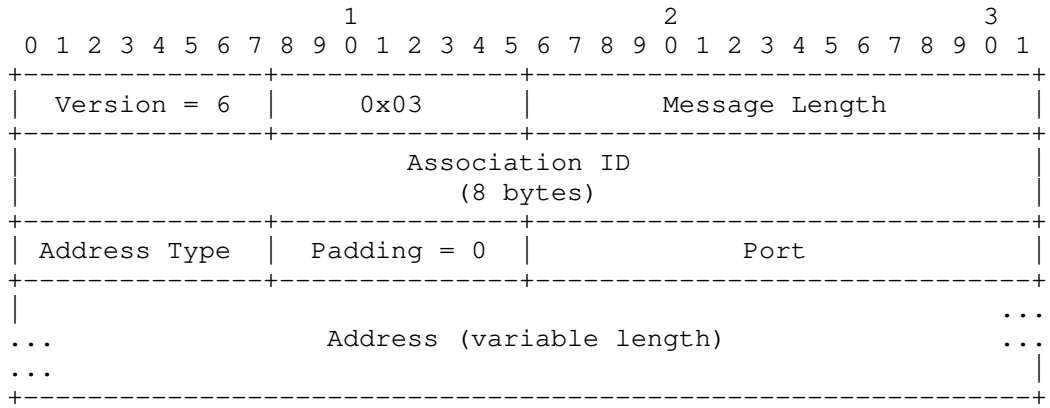


Figure 8: Datagram Header

- Version: 0x06
- Association ID: the identifier of the UDP association
- Address Type:
  - 0x01: IPv4
  - 0x03: Domain Name
  - 0x04: IPv6
- Address: this field's format depends on the address type:
  - IPv4: a 4-byte IPv4 address
  - Domain Name: one byte that contains the length of the FQDN, followed by the FQDN itself. The string is not NUL-terminated.
  - IPv6: a 16-byte IPv6 address

- Port: the port in network byte order.

Datagrams sent over UDP MAY be padded with arbitrary data (i. e., the Message Length MAY be smaller than the actual UDP/DTLS payload). Client and proxy implementations MUST ignore the padding. If the Message Length is larger than the size of the UDP or DTLS payload, the message MUST be silently ignored.

Following the receipt of the first datagram from the client, the proxy makes a one-way mapping between the Association ID and:

- the TCP connection, if it was received over TCP, or
- the 5-tuple of the UDP conversation, if the datagram was received over plain UDP, or
- the DTLS connection, if the datagram was received over DTLS. The DTLS connection is identified either by its 5-tuple, or some other mechanism, like [I-D.ietf-tls-dtls-connection-id].

The proxy SHOULD close the TCP connection if the initial datagram is not received after a timeout.

Further datagrams carrying the same Association ID, but not matching the established mapping, are silently dropped.

The proxy then sends an UDP Association Confirmation message over the TCP connection with the client:

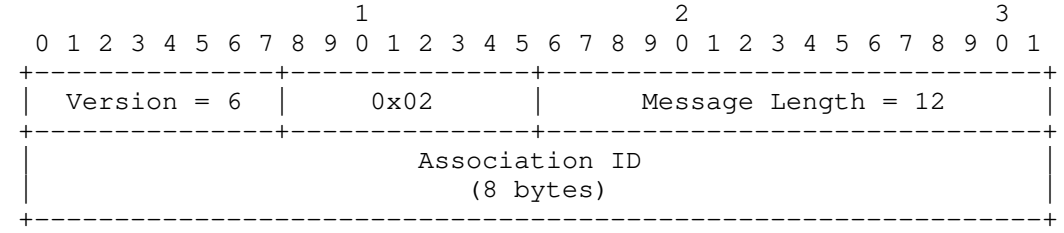


Figure 9: UDP Association Confirmation

Following the confirmation message, UDP packets bound for the proxy's bind address and port are relayed to the client, also prefixed by a Datagram Header.

The UDP association remains active for as long as the TCP connection between the client and the proxy is kept open.

7.3.1. Proxying UDP servers

Under some circumstances (e.g. when hosting a server), the SOCKS client expects the remote host to send UDP datagrams first. As such, the SOCKS client must trigger a UDP Association Confirmation without having the proxy relay any datagrams on its behalf.

To that end, it sends an empty datagram prefixed by a Datagram Header with an IP address and port consisting of zeroes. If it is using UDP, the client SHOULD resend the empty datagram if an UDP Association Confirmation is not received after a timeout.

7.3.2. Proxying multicast traffic

The use of multicast addresses is permitted for UDP traffic only.

7.3.3. Reporting ICMP Errors

If a client has opted in (see Section 8.1.8), the proxy MAY relay information contained in some ICMP Error packets. The message format is as follows:

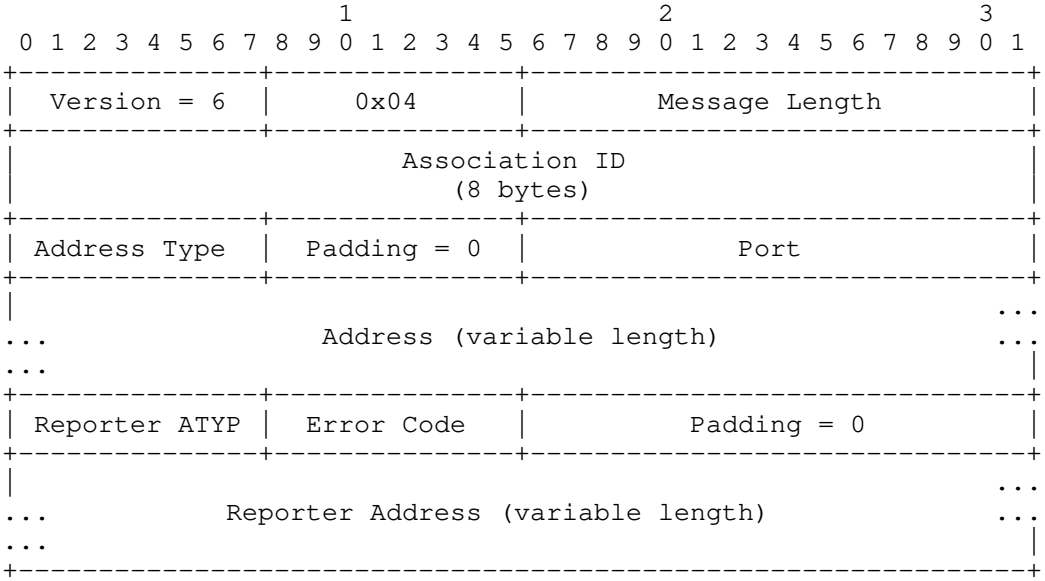


Figure 10: Datagram Error Message

- Address: The destination address of the IP header contained in the ICMP payload

- Address Type: Either 0x01 (IPv4) or 0x04 (IPv6)
- Port: The destination port of the UDP header contained in the ICMP payload
- Reporter Address: The IP address of the host that issued the ICMP error
- Reporter Address Type (ATYP): Either 0x01 (IPv4) or 0x04 (IPv6)
- Error code:
  - 0x01: Network unreachable
  - 0x02: Host unreachable
  - 0x03: TTL expired
  - 0x04: Datagram too big (IPv6 only)

It is possible for ICMP Error packets to be spurious, and not be related to any UDP packet that was sent out. The proxy is not required to check the validity of ICMP Error packets before reporting them to the client.

Clients MUST NOT send Datagram Error messages to the proxy. Proxies MUST NOT send Error messages unless the clients have opted in.

8. SOCKS Options

SOCKS options have the following format:

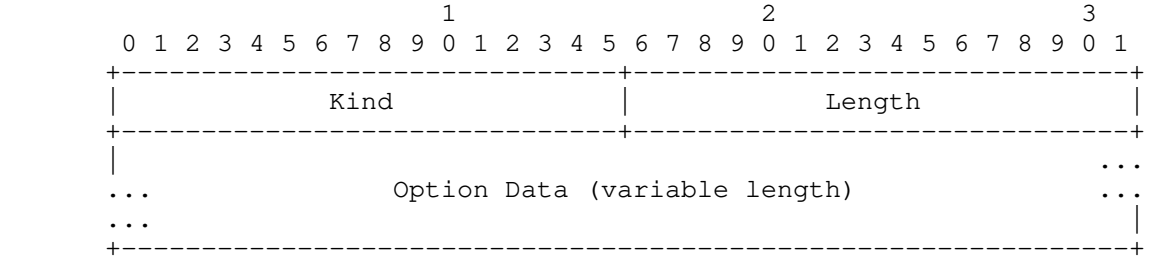


Figure 11: SOCKS 6 Option

- Kind: Allocated by IANA. (See Section 15.)
- Length: The total length of the option. MUST be a multiple of 4.

o Option Data: The contents are specific to each option kind.

Unless otherwise noted, client and proxy implementations MAY omit supporting any of the options described in this document. Upon encountering an unsupported option, a SOCKS endpoint MUST silently ignore it.

8.1. Stack options

Stack options can be used by clients to alter the behavior of the protocols on top of which SOCKS is running, as well the protocols used by the proxy to communicate with the remote host (i.e. IP, TCP, UDP). A Stack option can affect either the proxy's protocol on the client-proxy leg or on the proxy-remote leg. Clients can only place Stack options inside SOCKS Requests.

Proxies MAY choose not to honor any Stack options sent by the client.

Proxies include Stack options in their Operation Replies to signal their behavior, and MUST do so for every supported Stack option sent by the client. Said options MAY also be unsolicited, i. e. the proxy MAY send them to signal behavior that was not explicitly requested by the client.

If a particular Stack option is unsupported, the proxy MUST silently ignore it.

In case of UDP ASSOCIATE, the stack options refer to the UDP traffic relayed by the proxy.

Stack options that are part of the same message MUST NOT contradict one another or contain duplicate information.

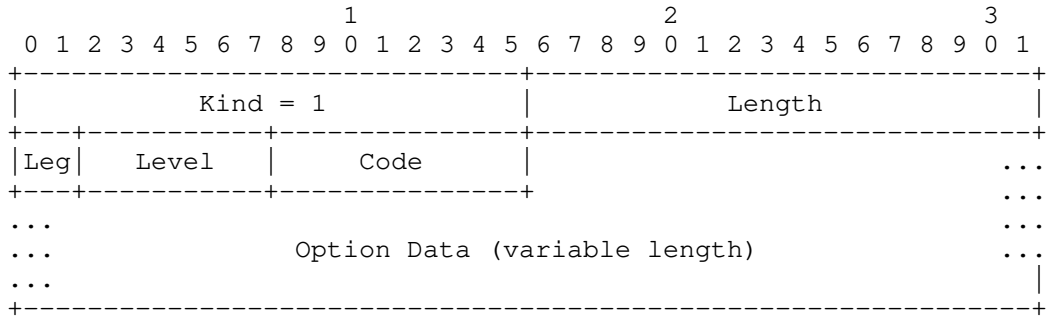


Figure 12: Stack Option

- Leg:
  - 1: Client-Proxy Leg
  - 2: Proxy-Remote Leg
  - 3: Both Legs
- Level:
  - 1: IP: options that apply to either IPv4 or IPv6
  - 2: IPv4
  - 3: IPv6
  - 4: TCP
  - 5: UDP
- Code: Option code
- Option Data: Option-specific data

8.1.1. IP TOS options

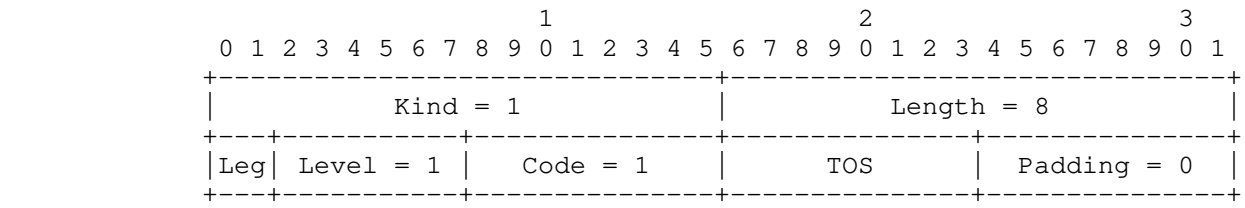


Figure 13: IP TOS Option

- TOS: The IP TOS code

The client can use IP TOS options to request that the proxy use a certain value for the IP TOS field. Likewise, the proxy can use IP TOS options to advertise the TOS values being used.

8.1.2. Happy Eyeballs options

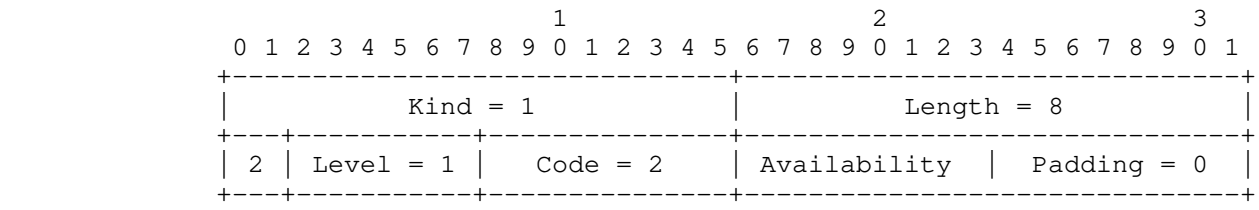


Figure 14: Happy Eyeballs Option

- Availability:
  - 0x01: Happy Eyeballs is not desired (client) or was not performed (proxy)
  - 0x02: Happy Eyeballs is desired (client) or was attempted (proxy)

This memo provides enough features for clients to implement a mechanism analogous to Happy Eyeballs [RFC8305] over SOCKS. However, when the delay between the client and the proxy, or the proxy's vantage point, is high, doing so can become impractical or inefficient.

In such cases, the client can instruct the proxy to employ the Happy Eyeballs technique on its behalf when connecting to a remote host.

The client MUST supply a Domain Name as part of its Request. Otherwise, the proxy MUST silently ignore the option.

TODO: Figure out which knobs to include.

8.1.3. TTL options

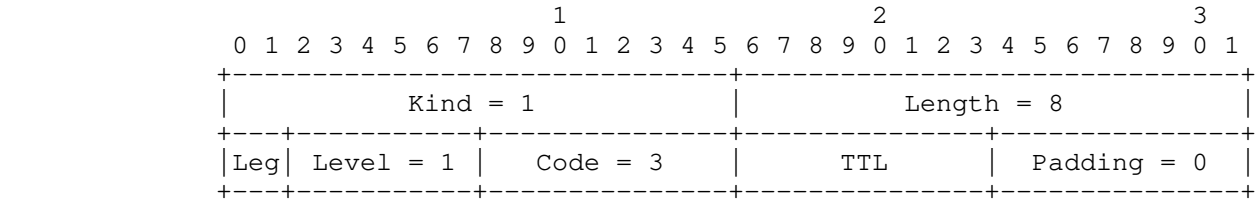


Figure 15: IP TTL Option

- TTL: The IP TTL or Hop Limit

8.1.4. No Fragmentation options

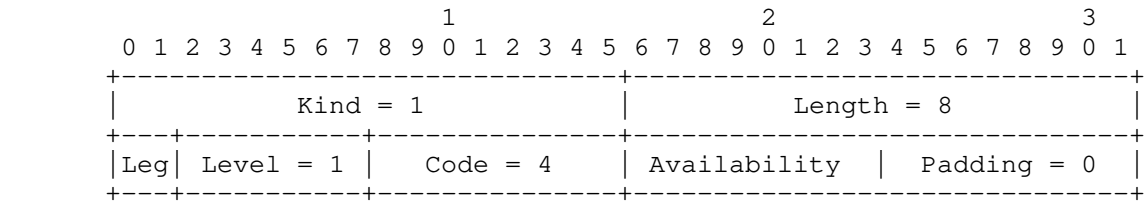


Figure 16: No Fragmentation Option

- Availability:
  - 0x01: IP fragmentation is allowed (client) or the lack thereof is not enforced (proxy)
  - 0x02: IP fragmentation is not desired (client) or avoidance of fragmentation is enforced (proxy)

A No Fragmentation option can be used to instruct the proxy to avoid IP fragmentation. In the case of IPv4, this also entails setting the DF bit on outgoing packets.

8.1.5. TFO options

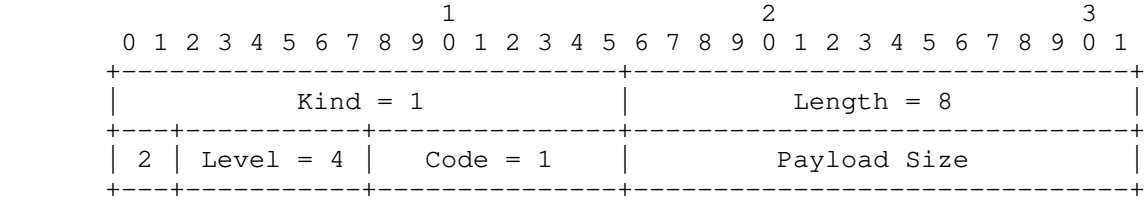


Figure 17: TFO Option

- Payload Size: The desired payload size of the TFO SYN. Ignored in case of a BIND command.

If a SOCKS Request contains a TFO option, the proxy SHOULD attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command. Otherwise, the proxy MUST NOT attempt to use TFO in case of a CONNECT command, or accept TFO in case of a BIND command.

In case of a CONNECT command, the client can indicate the desired payload size of the SYN. If the field is 0, the proxy can use an

arbitrary payload size. If the field is non-zero, the proxy MUST NOT use a payload size larger than the one indicated. The proxy MAY use a smaller payload size than the one indicated.

8.1.6. Multipath options

In case of a CONNECT or BIND command, the client can inform the proxy whether MPTCP is desired on the proxy-remote leg by sending a Multipath option.

Conversely, the proxy can use a Multipath option to convey the following information:

- whether or not the connection uses MPTCP or not, when replying to a CONNECT command, or in the second Operation reply to a BIND command, or
- whether an MPTCP connection will be accepted, when first replying to a BIND command.

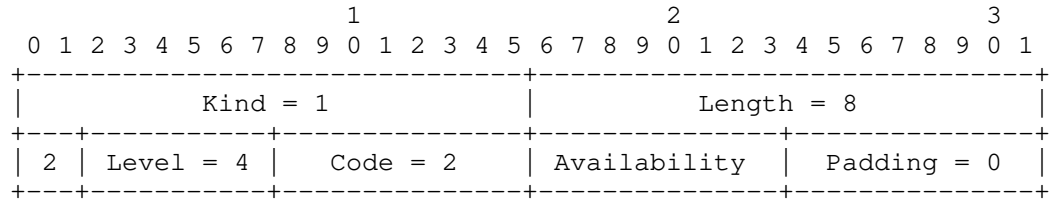


Figure 18: Multipath Option

- Availability:
  - 0x01: MPTCP is not desired (client) or available (proxy)
  - 0x02: MPTCP is desired (client) or available (proxy)

In the absence of such an option, the proxy SHOULD NOT enable MPTCP for CONNECT commands.

8.1.7. Listen Backlog options

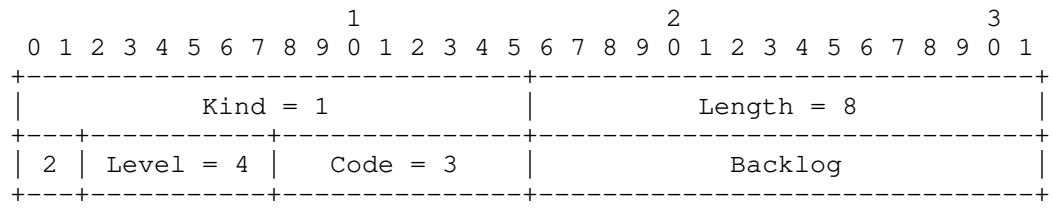


Figure 19: Listen Backlog Option

o Backlog: The length of the listen backlog.

The default behavior of the BIND does not allow a client to simultaneously handle multiple connections to the same bind address. A client can alter BIND's behavior by adding a TCP Listen Backlog Option to a BIND Request, provided that the Request is part of a Session.

In response, the proxy sends a TCP Listen Backlog Option as part of the Operation Reply, with the Backlog field signaling the actual backlog used. The proxy SHOULD NOT use a backlog longer than requested.

Following the successful negotiation of a backlog, the proxy listens for incoming connections for as long as the initial connection stays open. The initial connection is not used to relay data between the client and a remote host.

To accept connections, the client issues further BIND Requests using the bind address and port supplied by the proxy in the initial Operation Reply. Said BIND requests must belong to the same Session as the original Request.

If no backlog is issued, the proxy signals a backlog length of 0, and BIND's behavior remains unaffected.

8.1.8. UDP Error options

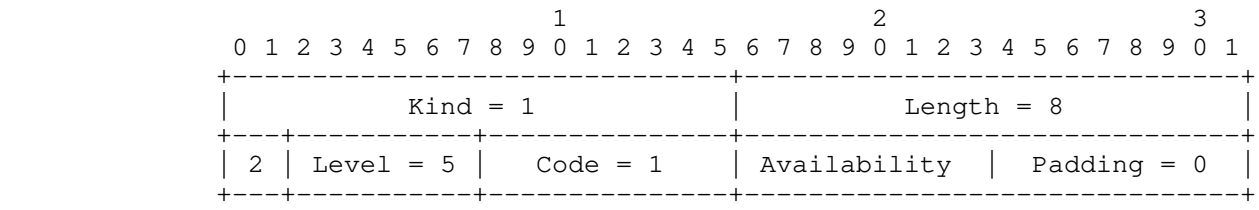


Figure 20: UDP Error Option

- Availability:
  - 0x01: Error reporting is not dersired (client) or will not be performed (proxy)
  - 0x02: Error reporting is dersired (client) or will be performed (proxy)

Clients can use this option to turn on error reporting for a particular UDP association. See Section 7.3.3.

8.1.9. Port Parity options

The RTP specification [RFC3550] recommends running the protocol on consecutive UDP ports, where the even port is the lower of the two.

SOCKS clients can specify the desired port parity when issuing a UDP ASSOCIATE command, and request that the port's counterpart be reserved.

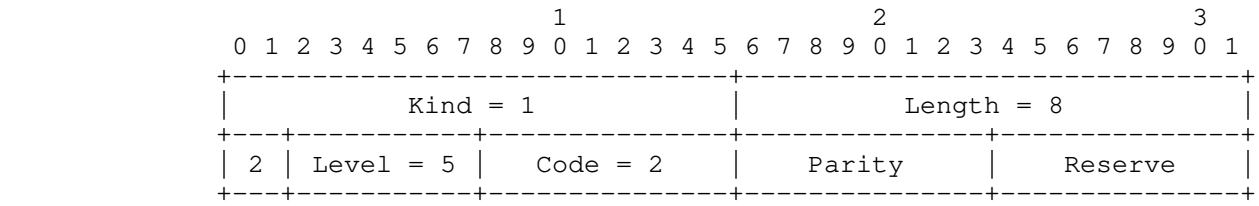


Figure 21: Port Parity Option

- Parity:
  - 0x00: No particular parity
  - 0x01: Even

    * 0x02: Odd

  o Reserve: whether or not to reserve the port's counterpart

    * 0x00: Don't reserve

    * 0x01: Reserve

  If the UDP ASSOCIATE request does not have the Port field set to 0 (indicating that an arbitrary port can be chosen), the proxy MUST ignore the suggested parity.

  A port's counterpart is determined as follows:

  o for even ports, it is the next higher port and

  o for odd ports, it is the next lower port.

  If the proxy can not or will not comply with the requested parity, it also does not reserve the allocated port's counterpart.

  Port reservations are in place until either:

  o the original association ends, or

  o an association involving the reserved port is made.

  An association involving a reserved port can only be made if a client explicitly requests said port. Further, if the original association is part of a session (see Section 8.4), the reserved port can only be claimed from within the same session.

8.2. Authentication Method options

  A client that is willing to go through the authentication phase MUST include an Authentication Method Advertisement option in its Request. In case of a CONNECT Request, the option is also used to specify the amount of initial data supplied before any method-specific authentication negotiations take place.

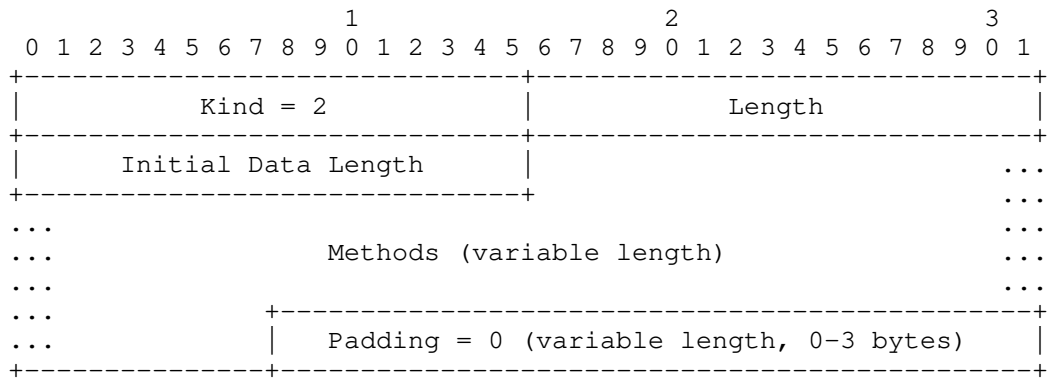


Figure 22: Authentication Method Advertisement Option

- Initial Data Size: A two-byte number in network byte order. In case of CONNECT, this is the number of bytes of initial data that are supplied by the client immediately following the Request. This number MUST NOT be larger than 2^14.

- Methods: One byte per advertised method. Method numbers are assigned by IANA.

- Padding: A minimally-sized sequence of zeroes, such that the option length is a multiple of 4. Note that 0 coincides with the value for "No Authentication Required".

Clients MUST support the "No authentication required" method. Clients SHOULD omit advertising the "No authentication required" option.

The proxy indicates which authentication method must proceed by sending an Authentication Method Selection option in the corresponding Authentication Reply:

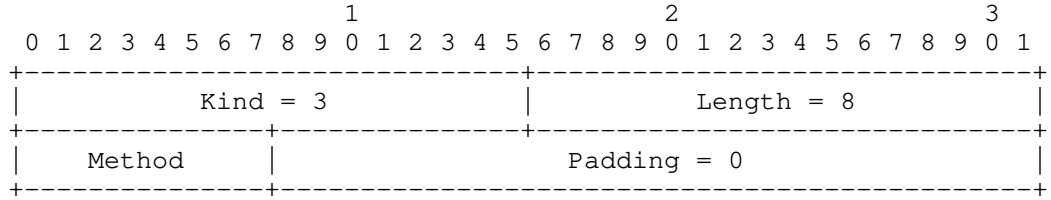


Figure 23: Authentication Method Selection Option

- Method: The selected method.

If the proxy selects "No Acceptable Methods", the client MUST close the connection.

If authentication is successful via some other means, or not required at all, the proxy silently ignores the Authentication Method Advertisement option.

8.3. Authentication Data options

Authentication Data options carry method-specific authentication data. They can be part of SOCKS Requests and Authentication Replies.

Authentication Data options have the following format:

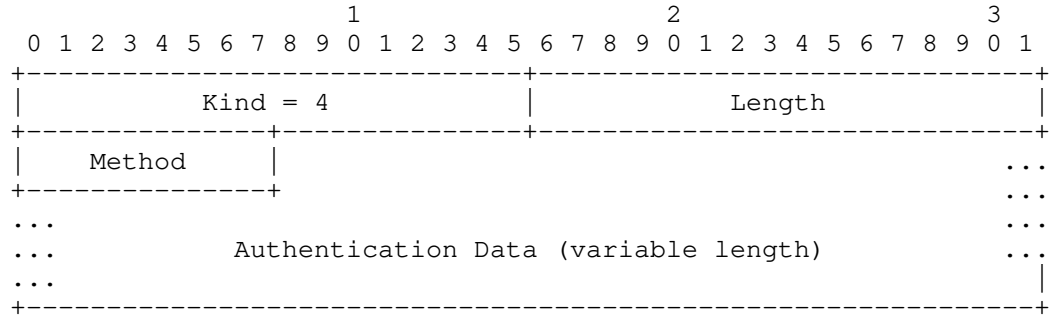


Figure 24: Authentication Data Option

- Method: The number of the authentication method. These numbers are assigned by IANA.

- Authentication Data: The contents are specific to each method.

Clients MUST only place one Authentication Data option per authentication method.

8.4. Session options

Clients and proxies can establish SOCKS sessions, which span one or more Requests. All session-related negotiations are done via Session Options, which are placed in Requests and Authentication Replies by the client and, respectively, by the proxy.

Client and proxy implementations MUST either support all Session Option Types, or none.

8.4.1. Session initiation

A client can initiate a session by sending a Session Request Option:

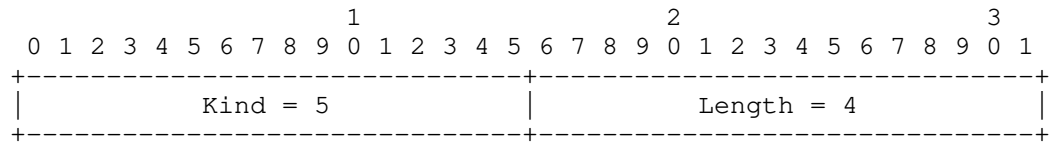


Figure 25: Session Request Option

The proxy then replies with a Session ID Option in the successful Operation Reply:

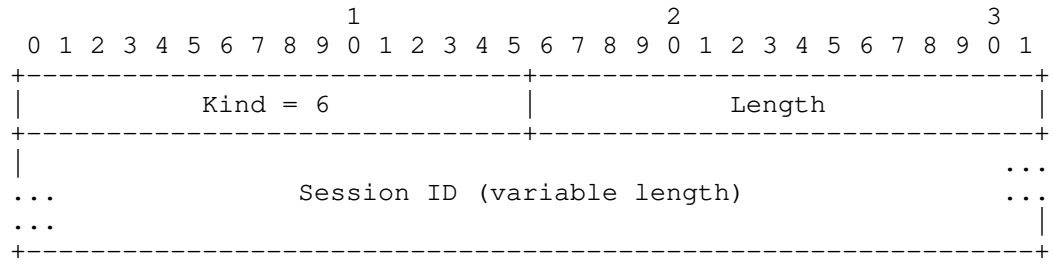


Figure 26: Session ID Option

- Session ID: An opaque sequence of bytes specific to the session. The size MUST be a multiple of 4. MUST NOT be empty.

The Session ID serves to identify the session and is opaque to the client.

The credentials, or lack thereof, used to initiate the session are tied to the session.

The SOCKS Request that initiated the session is considered part of the session. A client MUST NOT attempt to initiate a session from within a different session.

If the proxy can not or will not honor the Session Request, it does so silently.

8.4.2. Further SOCKS Requests

Any further SOCKS Requests that are part of the session MUST include a Session ID Option (as seen in Figure 26). The proxy MUST silently ignore any authentication attempt in the Request, and MUST NOT require any authentication.

The proxy then replies by placing a Session OK option in the successful Authentication Reply:

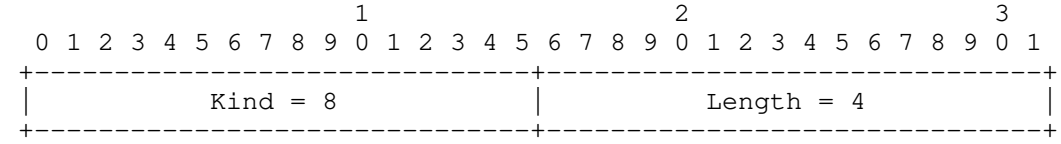


Figure 27: Session OK Option

If the Session ID is invalid, the first Authentication Reply MUST signal that authentication failed and can not continue (by setting the Type field to 0x01). Further, it SHALL contain a Session Invalid option:

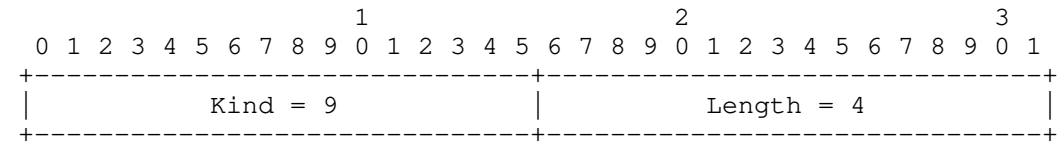


Figure 28: Session Invalid Option

8.4.3. Tearing down the session

Proxies can, at their discretion, tear down a session and free all associated state. Proxy implementations SHOULD feature a timeout mechanism that destroys sessions after a period of inactivity. When a session is terminated, the proxy MAY close all connections associated with said session.

Clients can signal that a session is no longer needed, and can be torn down, by sending a Session Teardown option in addition to the Session ID option:

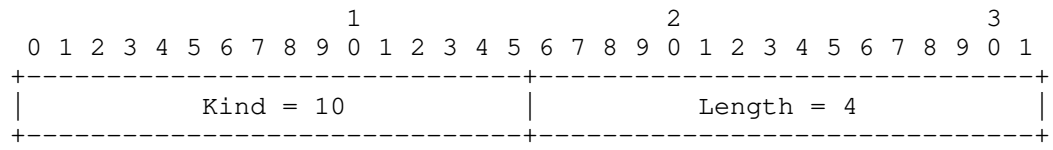


Figure 29: Session Teardown Option

After sending such an option, the client MUST assume that the session is no longer valid. The proxy MUST treat the session-terminating request as if it were not part of any session.

8.5. Idempotence options

To protect against duplicate SOCKS Requests, clients can request, and then spend, idempotence tokens. A token can only be spent on a single SOCKS request.

Tokens are 4-byte unsigned integers in a modular 4-byte space. Therefore, if x and y are tokens, x is less than y if $0 < (y - x) < 2^{31}$ in unsigned 32-bit arithmetic.

Proxies grant contiguous ranges of tokens called token windows. Token windows are defined by their base (the first token in the range) and size.

All token-related operations are done via Idempotence options.

Idempotence options are only valid in the context of a SOCKS Session. If a SOCKS Request is not part of a Session (either by supplying a valid Session ID or successfully initiating one via a Session Request), the proxy MUST silently ignore any Idempotence options.

Token windows are tracked by the proxy on a per-session basis. There can be at most one token window for every session and its tokens can only be spent from within said session.

Client and proxy implementations MUST either support all Idempotence Option Types, or none.

8.5.1. Requesting a token window

A client can obtain a window of tokens by sending an Idempotence Request option as part of a SOCKS Request:

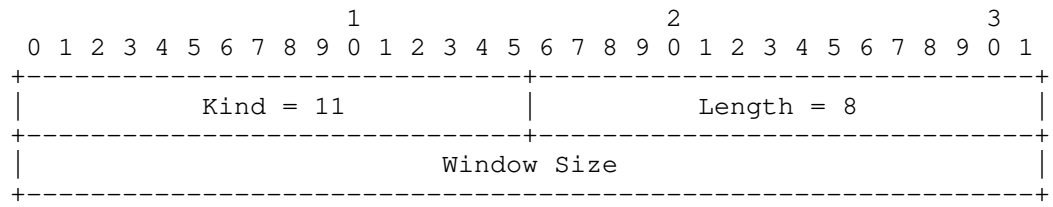


Figure 30: Token Request

- Window Size: The requested window size.

Once a token window is issued, the proxy MUST include an Idempotence Window option in all subsequent successful Authentication Replies:

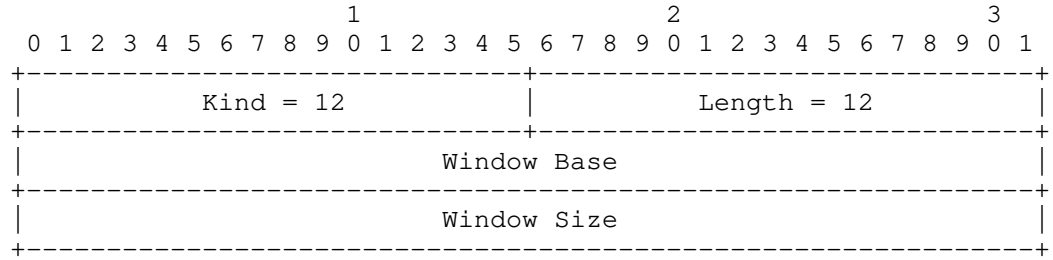


Figure 31: Idempotence Window

- Window Base: The first token in the window.

- Window Size: The window size. This value MAY differ from the requested window size. Window sizes MUST be less than 2^31. Window sizes MUST NOT be 0.

If no token window is issued, the proxy MUST silently ignore the Token Request. If there is already a token window associated with the session, the proxy MUST NOT issue a new window.

8.5.2. Spending a token

The client can attempt to spend a token by including a Idempotence Expenditure option in its SOCKS request:

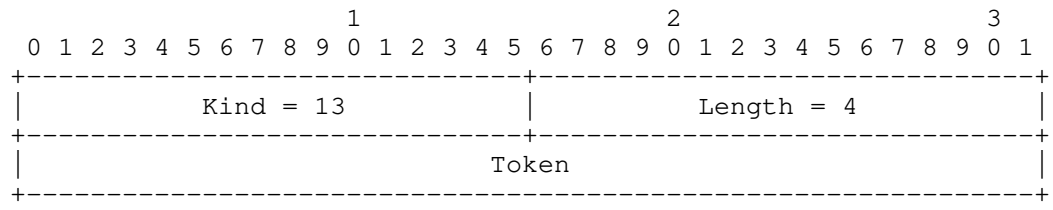


Figure 32: Idempotence Expenditure

- Kind: 13 (Idempotence Expenditure option)
- Length: 8
- Token: The token being spent.

Clients SHOULD prioritize spending the smaller tokens.

The proxy responds by sending either an Idempotence Accepted or Rejected option as part of the Authentication Reply:

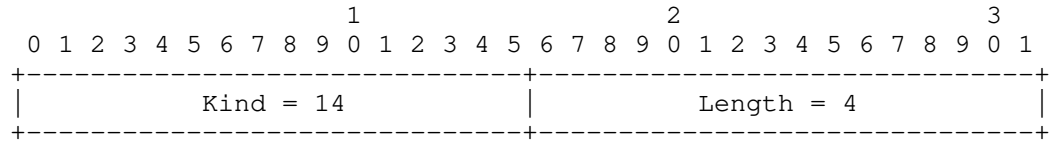


Figure 33: Idempotence Accepted

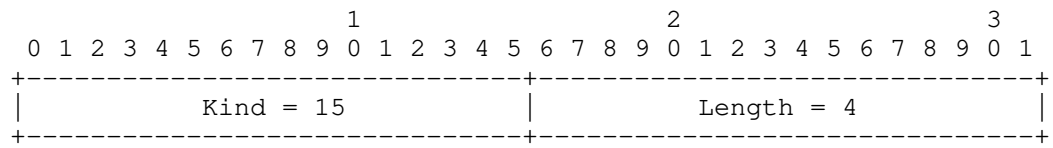


Figure 34: Idempotence Rejected

If eligible, the token is spent before attempting to honor the Request. If the token is not eligible for spending, the Authentication Reply MUST indicate failure.

8.5.3. Shifting windows

Windows can be shifted (i. e. have their base increased, while retaining their size) unilaterally by the proxy.

Proxy implementations SHOULD shift the window: * as soon as the lowest-order token in the window is spent and * when a sufficiently high-order token is spent.

Proxy implementations SHOULD NOT shift the window's base beyond the highest unspent token.

8.5.4. Out-of-order Window Advertisements

Even though the proxy increases the window's base monotonically, there is no mechanism whereby a SOCKS client can receive the Token Window Advertisements in order. As such, clients SHOULD disregard Token Window Advertisements with a Window Base less than the previously known value.

9. Username/Password Authentication

Username/Password authentication is carried out as in [RFC1929].

Clients can also attempt to authenticate by placing the Username/Password request in an Authentication Data Option.

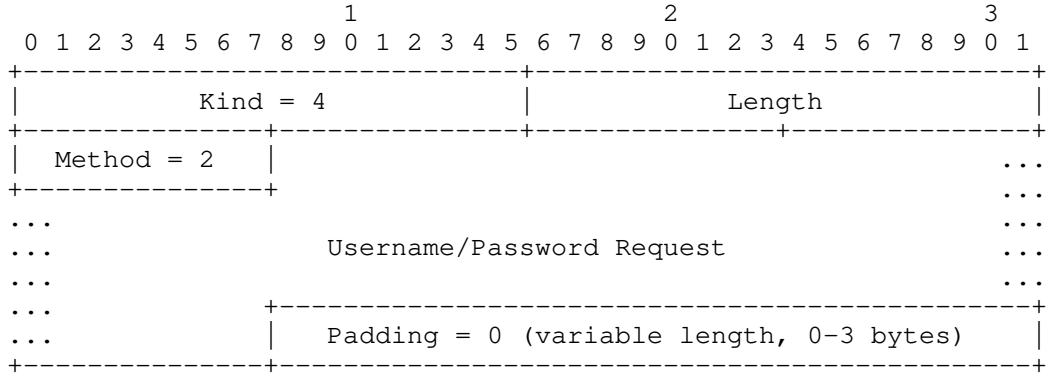


Figure 35: Password authentication via a SOCKS Option

- Username/Password Request: The Username/Password Request, as described in [RFC1929].

Proxies reply by including a Authentication Data Option in the next Authentication Reply which contains the Username/Password reply:

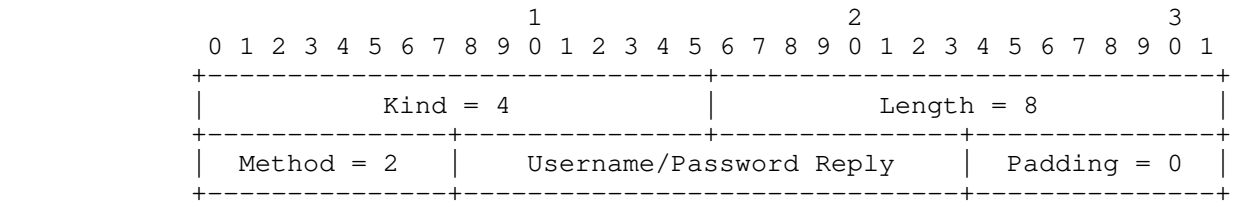


Figure 36: Reply to password authentication via a SOCKS Option

- Username/Password Reply: The Username/Password Reply, as described in [RFC1929].

10. TCP Fast Open on the Client-Proxy Leg

TFO breaks TCP semantics, causing replays of the data in the SYN's payload under certain rare circumstances [RFC7413]. A replayed SOCKS Request could itself result in a replayed connection on behalf of the client.

As such, client implementations SHOULD NOT use TFO on the client-proxy leg unless:

- The protocol running on top of SOCKS tolerates the risks of TFO, or
- The SYN's payload does not contain any application data (so that no data is replayed to the server, even though duplicate connections are still possible), or
- The client uses Idempotence Options, making replays very unlikely, or
- SOCKS is running on top of TLS and Early Data is not used.

11. False Starts

In case of CONNECT Requests, the client MAY start sending application data as soon as possible, as long as doing so does not incur the risk of breaking the SOCKS protocol.

Clients must work around the authentication phase by doing any of the following:

- If the Request does not contain an Authentication Method Advertisement option, the authentication phase is guaranteed not to happen. In this case, application data MAY be sent immediately after the Request.

- Application data MAY be sent immediately after receiving an Authentication Reply indicating success.

- When performing a method-specific authentication sequence, application data MAY be sent immediately after the last client message.

12. DNS provided by SOCKS

Clients may require information typically obtained from DNS servers, albeit from the proxy's vantage point.

While the CONNECT command can work with domain names, some clients' workflows require that addresses be resolved as a separate step prior to connecting. Moreover, the SOCKS Datagram Header, as described in Section 7.3, can be reduced in size by providing the resolved destination IP address, rather than the FQDN.

Emerging techniques may also make use of DNS to deliver server-specific information to clients. For example, Encrypted SNI [I-D.ietf-tls-esni] relies on DNS to publish encryption keys.

Proxy implementations MAY provide a default plaintext DNS service. A client looking to make use of it issues a CONNECT Request to IP address 0.0.0.0 or 0:0:0:0:0:0:0:0 on port 53. Following successful authentication, the Operation Reply MAY indicate an unspecified bind address (0.0.0.0 or ::) and port (0). The client and proxy then behave as per [RFC7766].

The service itself can be provided directly by the proxy daemon, or by proxying the client's request to a pre-configured DNS server.

If the proxy does not implement such functionality, it MAY return an error code signaling "Connection refused".

13. Security Considerations

13.1. Large requests

Given the format of the request message, a malicious client could craft a request that is in excess of 16 KB and proxies could be prone to DDoS attacks.

To mitigate such attacks, proxy implementations SHOULD be able to incrementally parse the requests. Proxies MAY close the connection to the client if:

- the request is not fully received after a certain timeout, or
- the number of options or their size exceeds an imposed hard cap.

13.2. Replay attacks

In TLS 1.3, early data (which is likely to contain a full SOCKS request) is prone to replay attacks.

While Token Expenditure options can be used to mitigate replay attacks, anything prior to the initial Token Request is still vulnerable. As such, client implementations SHOULD NOT make use of TLS early data unless the Request attempts to spend a token.

13.3. Resource exhaustion

Malicious clients can issue a large number of Session Requests, forcing the proxy to keep large amounts of state.

To mitigate this, the proxy MAY implement policies restricting the number of concurrent sessions on a per-IP or per-user basis, or barring unauthenticated clients from establishing sessions.

14. Privacy Considerations

The timing of Operation Replies can reveal some information about a proxy's recent usage:

- The DNS resolver used by the proxy may cache the answer to recent queries. As such, subsequent connection attempts to the same hostname are likely to be slightly faster, even if requested by different clients.
- Likewise, the proxy's OS typically caches TFO cookies. Repeated TFO connection attempts tend to be sped up, regardless of the client.

15. IANA Considerations

This document requests that IANA allocate 2-byte option kinds for SOCKS 6 options. Further, this document requests the following option kinds:

- Unassigned: 0

- Stack: 1
- Authentication Method Advertisement: 2
- Authentication Method Selection: 3
- Authentication Data: 4
- Session Request: 5
- Session ID: 6
- Session OK: 8
- Session Invalid: 9
- Session Teardown: 10
- Idempotence Request: 11
- Idempotence Window: 12
- Idempotence Expenditure: 13
- Idempotence Accepted: 14
- Idempotence Rejected: 15
- Resolution Request: 16
- IPv4 Resolution: 17
- IPv6 Resolution: 18
- Experimental: 64512-0xFFFF

This document also requests that IANA allocate a TCP and UDP port for SOCKS over TLS and DTLS, respectively.

16. Acknowledgments

The protocol described in this draft builds upon and is a direct continuation of SOCKS 5 [RFC1928].

17. References

17.1. Normative References

[RFC1929] Leech, M., "Username/Password Authentication for SOCKS V5", RFC 1929, DOI 10.17487/RFC1929, March 1996, <https://www.rfc-editor.org/info/rfc1929>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC7766] Dickinson, J., Dickinson, S., Bellis, R., Mankin, A., and D. Wessels, "DNS Transport over TCP - Implementation Requirements", RFC 7766, DOI 10.17487/RFC7766, March 2016, <https://www.rfc-editor.org/info/rfc7766>.

[RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/info/rfc8305>.

17.2. Informative References

[I-D.ietf-tls-dtls-connection-id] Rescorla, E., Tschofenig, H., and T. Fossati, "Connection Identifiers for DTLS 1.2", draft-ietf-tls-dtls-connection-id-07 (work in progress), October 2019.

[I-D.ietf-tls-esni] Rescorla, E., Oku, K., Sullivan, N., and C. Wood, "TLS Encrypted Client Hello", draft-ietf-tls-esni-08 (work in progress), October 2020.

[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, DOI 10.17487/RFC1928, March 1996, <https://www.rfc-editor.org/info/rfc1928>.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <https://www.rfc-editor.org/info/rfc3550>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <https://www.rfc-editor.org/info/rfc6824>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

Authors' Addresses

Vladimir Olteanu
University Politehnica of Bucharest
313 Splaiul Independentei, Sector 6
Bucharest
Romania

Email: vladimir.olteanu@cs.pub.ro

Dragos Niculescu
University Politehnica of Bucharest
313 Splaiul Independentei, Sector 6
Bucharest
Romania

Email: dragos.niculescu@cs.pub.ro

LPWAN
Internet-Draft
Updates: 5172 (if approved)
Intended status: Standards Track
Expires: 2 January 2021

P. Thubert, Ed.
Cisco Systems

1 July 2020

SCHC over PPP
draft-thubert-intarea-schc-over-ppp-00

Abstract

This document extends RFC 5172 to signal the use of SCHC as the compression method between a pair of nodes over PPP. Combined with RFC 2516, this enables the use of SCHC over Ethernet and Wi-Fi.

Table of Contents

1. Introduction

The Point-to-Point Protocol (PPP) [RFC5172] provides a standard method of encapsulating network-layer protocol information over point-to-point links. "A Method for Transmitting PPP Over Ethernet (PPPoE)" [RFC2516] transports PPP over Ethernet between a pair of nodes. It is compatible with a translating bridge to Wi-Fi, and therefore enables PPP over Wi-Fi as well.

PPP also defines an extensible Link Control Protocol, and proposes a family of Network Control Protocols (NCPs) for establishing and configuring different network-layer protocols. "IP Version 6 over PPP" [RFC5072] defines the IPv6 Control Protocol (IPV6CP) , which is an NCP for a PPP link, and allows for the negotiation of desirable parameters for an IPv6 interface over PPP.

"Negotiation for IPv6 Datagram Compression Using IPv6 Control Protocol" [RFC5172] defines the IPv6 datagram compression option that can be negotiated by a node on the link through the IPV6CP. The "Static Context Header Compression (SCHC) and fragmentation for LPWAN, application to UDP/IPv6" [SCHC] is a compression and fragmentation technique that was defined after the publication of [RFC5172]. In order to enable SCHC over PPP and therefore Ethernet and Wi-Fi, [RFC5172] must be extended to signal SCHC as an additional compression method for use over PPP.

2. BCP 14

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 BCP 14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Extending RFC 5172

With this specification, a PPP session defines a vitual link where a SCHC context is established with a particular set of Rules, which is indicated at the set up of the PPP session as follows:

[RFC5172] defines an IPV6CP option called the IPv6-Compression-Protocol Configuration option with a type of 2. The option contains an IPv6-Compression-Protocol field value that indicates a compression protocol and an optional data field as shown in Figure 1:

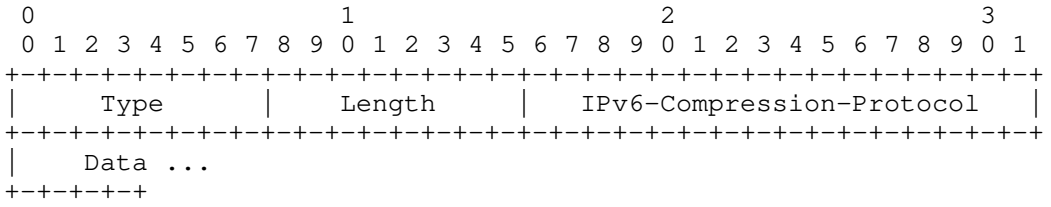


Figure 1: The IPv6-Compression-Protocol Configuration Option

This specification indicates a new IPv6-Compression-Protocol field value for [SCHC] (see Section 5), and enables to transport a Uniform Resource Identifier (URI) [RFC3986] of the set of rules in the optional data. The default format for the set of rules is YANG using the "Data Model for SCHC" [SCHC_DATA_MODEL] encoded in JSON as specified in [RFC7951]. The size of the URL is computed based on the Length of the option as Length-4. If the encoding is asymetrical, the initiator of the session is considered downstream, playing the role of the device in an LPWAN network.

4. Profiling SCHC for high speed links

Appendix D of [SCHC] specifies the profile information that technology specifications such as this must provide. The following section address this requirement.

4.1. Mapping the SCHC Architecture

This specification leverages SCHC between an end point that is an IP Host and possibly a serial DTE (Data Terminal Equipment), and another that is an IP Node (either another IP Host or a Router) and possibly a serial DCE (Data Control Equipment), or a more modern physical or emulated endpoint, e.g., Ethernet devices that echange IP packets over PPPoE.

Both endpoints MUST support the function of SCHC Compressor/ Decompressor (C/D) as shown in Figure 2.

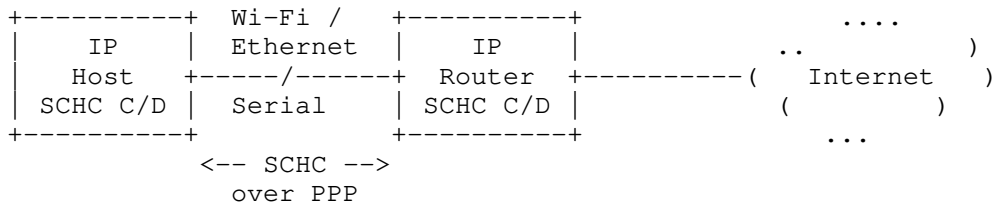


Figure 2: Typical Deployment

The assumption for this document is that the SCHC Fragmenter/ Reassembler (F/R) is generally not needed, because the maximum transmission unit (MTU) is expected to be large enough and SCHC only reduces the frame size vs. native IP.

An example use case for SCHC over PPP over Ethernet (PPPoE) would be to apply SCHC to streamline traffic by reducing the size of the frames and maintain them to a constant size and constant rate, e.g., to simplify the scheduling of [DetNet] packets over TSN or one of the [RAW Technologies]. Scheduling on DetNet links introduces a form of duty cycle, but that does not affect the SCHC operation, since fragmentation is not provided.

A context may be generated for a particular upper layer application, such as a control loop using an industrial automation protocol, to protect the particular flow with a DetNet service. The context can be asymetric, e.g., when connecting a master and a slave, a client and a server, or a programmable logic controller with a sensor or an actuator.

4.2. SCHC Parameters

Compared to typical LPWANs, most serial links and emulations such as PPPoE are very fast and most of the constraints can be alleviated. For this reason, the SCHC profile for PPP is defined as follows:

RuleID numbering scheme: Rules are of a fixed size of two bytes, which allows for more than 65000 different Rules within one session. Since this specification does not leverage the SCHC fragmentation, a SCHC packet is always in the form:

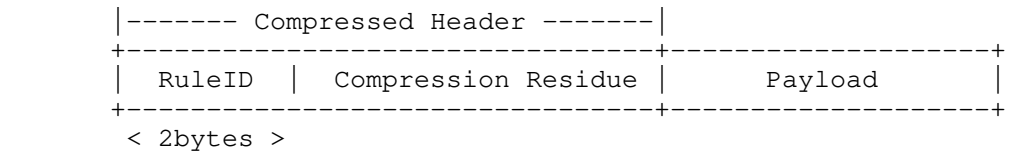


Figure 3: SCHC Packet

Maximum packet size: MAX_PACKET_SIZE is aligned to the PPP Link MTU.

Padding: The L2 word is one byte, so padding is up to the next byte boundary. The padding bit is a 0.

4.2.1. Resulting Packet Format

In the case of PPPoE, the sequence of compression and encapsulation is as follows:

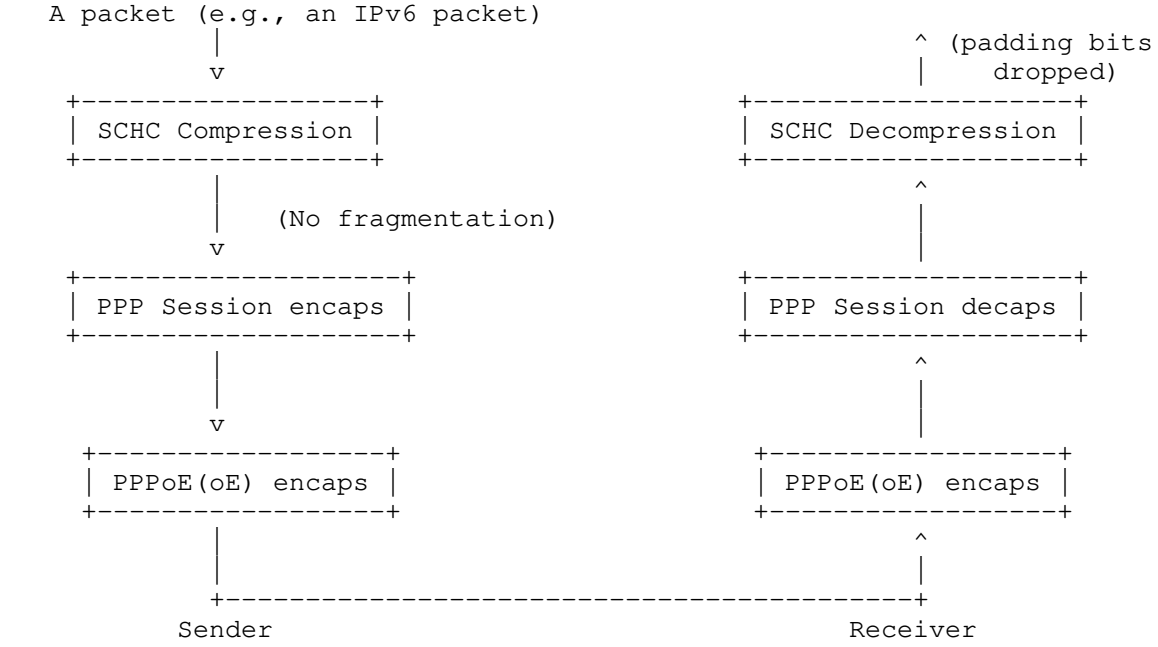


Figure 4: Stack Operation

In the case of PPPoE, a frame that transports an IPv6 packet compressed with SCHC shows as follows:

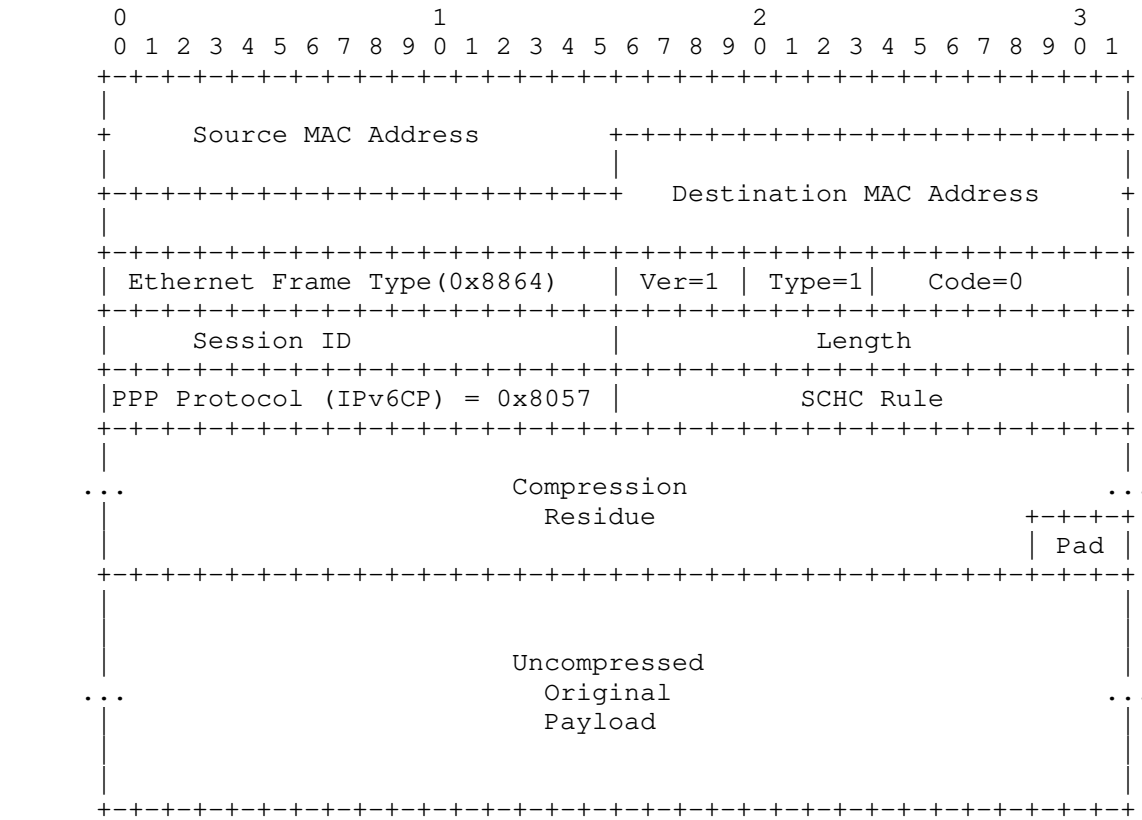


Figure 5: SCHC over PPP over Ethernet Format

4.3. Security Considerations

This draft enables to use the SCHC compression and fragmentation over PPP and therefore Ethernet and Wi-Fi with PPPoE. It inherits the possible threats against SCHC listed in the "Security considerations" section of [SCHC].

5. IANA Considerations

This document requests the allocation of a new value in the registry "IPv6-Compression-Protocol Types" for "SCHC". A suggested value is proposed in Table 1:

Value	Description	Reference
4	Static Context Header Compression (SCHC)	This document

Table 1: IP Header Compression Configuration Option Suboption Types

6. Acknowledgments

7. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2516] Mamakos, L., Lidl, K., Evarts, J., Carrel, D., Simone, D., and R. Wheeler, "A Method for Transmitting PPP Over Ethernet (PPPoE)", RFC 2516, DOI 10.17487/RFC2516, February 1999, <https://www.rfc-editor.org/info/rfc2516>.

[RFC5072] Varada, S., Ed., Haskins, D., and E. Allen, "IP Version 6 over PPP", RFC 5072, DOI 10.17487/RFC5072, September 2007, <https://www.rfc-editor.org/info/rfc5072>.

[RFC5172] Varada, S., Ed., "Negotiation for IPv6 Datagram Compression Using IPv6 Control Protocol", RFC 5172, DOI 10.17487/RFC5172, March 2008, <https://www.rfc-editor.org/info/rfc5172>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[SCHC] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zúñiga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724, DOI 10.17487/RFC8724, April 2020, <https://www.rfc-editor.org/info/rfc8724>.

8. Informative References

[SCHC_DATA_MODEL]
Minaburo, A. and L. Toutain, "Data Model for Static Context Header Compression (SCHC)", Work in Progress, Internet-Draft, draft-ietf-lpwan-schc-yang-data-model-02, 28 February 2020, <https://tools.ietf.org/html/draft-ietf-lpwan-schc-yang-data-model-02>.

[RAW Technologies]
Thubert, P., Cavalcanti, D., Vilajosana, X., Schmitt, C., and J. Farkas, "Reliable and Available Wireless Technologies", Work in Progress, Internet-Draft, draft-thubert-raw-technologies-05, 18 May 2020, <https://tools.ietf.org/html/draft-thubert-raw-technologies-05>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

[DetNet] Finn, N., Thubert, P., Varga, B., and J. Farkas, "Deterministic Networking Architecture", RFC 8655, DOI 10.17487/RFC8655, October 2019, <https://www.rfc-editor.org/info/rfc8655>.

Author's Address

Pascal Thubert (editor)
Cisco Systems, Inc
Building D
45 Allee des Ormes - BP1200
06254 Mougins - Sophia Antipolis
France

Phone: +33 497 23 26 34
Email: pthubert@cisco.com

LPWAN
Internet-Draft
Updates: 5172 (if approved)
Intended status: Standards Track
Expires: 26 March 2021

P. Thubert, Ed.
Cisco Systems

22 September 2020

SCHC over PPP
draft-thubert-intarea-schc-over-ppp-02

Abstract

This document extends RFC 5172 to signal the use of SCHC as the compression method between a pair of nodes over PPP. Combined with RFC 2516, this enables the use of SCHC over Ethernet and Wi-Fi.

Table of Contents

1. Introduction

The Point-to-Point Protocol (PPP) [RFC5172] provides a standard method of encapsulating network-layer protocol information over serial (point-to-point and bus) links. "A Method for Transmitting PPP Over Ethernet (PPPoE)" [RFC2516] transports PPP over Ethernet between a pair of nodes. It is compatible with a translating bridge to Wi-Fi, and therefore enables PPP over Wi-Fi as well.

PPP also proposes an extensible Link Control Protocol and a family of Network Control Protocols (NCPs) for establishing and configuring different network-layer protocols. "IP Version 6 over PPP" [RFC5072] specifies the IPv6 Control Protocol (IPV6CP), which is an NCP for a PPP link, and allows for the negotiation of desirable parameters for an IPv6 interface over PPP. "Negotiation for IPv6 Datagram Compression Using IPv6 Control Protocol" [RFC5172] defines the IPv6 datagram compression option that can be negotiated by a node on the link through the IPV6CP.

The "Static Context Header Compression (SCHC) and fragmentation for LPWAN, application to UDP/IPv6" [SCHC] is a new technology that can provide an extreme compression performance but requires a same state to be provisionned on both ends before it can be operated. To enable SCHC over PPP and therefore Ethernet and Wi-Fi, this specification extends [RFC5172] to signal SCHC as an additional compression method for use over PPP.

An example use case for SCHC over PPP over Ethernet (SCHCoPPPoE) is to apply SCHC to periodic flows and maintain them at a protocol-independant size and rate. The constant size may be too small for a particular flow or protocol. The SCHC fragmentation can then be used to transport a protocol data unit (PDU) as N compressed SCHC fragments, in which case the effective PDU rate is the TSN frame rate divided by N.

This can be useful to streamline the frames and simplifies the scheduling of Deterministic Networking [DetNet] and Operational Technology (OT) control flows over IEEE Std 802.1 Time-Sensitive Networking (TSN) [IEEE802.1TSNTG] or one of the [RAW Technologies].

2. BCP 14

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 BCP 14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Extending RFC 5172

With this specification, a PPP session defines a vitual link where a SCHC context is established with a particular set of Rules, which is indicated at the set up of the PPP session as follows:

[RFC5172] defines an IPV6CP option called the IPv6-Compression-Protocol Configuration option with a type of 2. The option contains an IPv6-Compression-Protocol field value that indicates a compression protocol and an optional data field as shown in Figure 1:

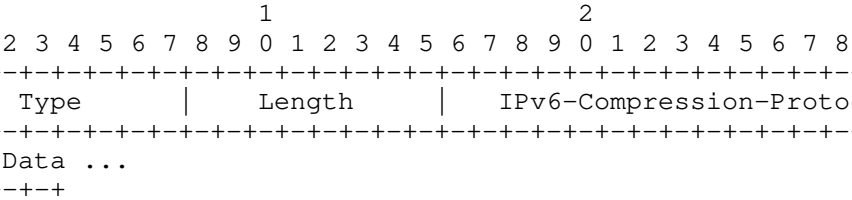


Figure 1: The IPv6-Compression-Protocol Configuration Option

This specification indicates a new IPv6-Compression-Protocol field value for [SCHC] (see Section 5), and enables to transport a Uniform Resource Identifier (URI) [RFC3986] of the set of rules in the optional data. The default format for the set of rules is YANG using the "Data Model for SCHC" [SCHC_DATA_MODEL] encoded in JSON as specified in [RFC7951]. The size of the URL is computed based on the

Length of the option as Length-4. If the encoding is asymetrical, the initiator of the session is considered downstream, playing the role of the device in an LPWAN network.

4. Profiling SCHC for high speed links

Appendix D of [SCHC] specifies the profile information that technology specifications such as this must provide. The following section address this requirement.

4.1. Mapping the SCHC Architecture

This specification leverages SCHC between an end point that is an IP Host and possibly a serial DTE (Data Terminal Equipment), and another that is an IP Node (either another IP Host or a Router) and possibly a serial DCE (Data Control Equipment), or a more modern physical or emulated endpoint, e.g., Ethernet devices that echange IP packets over PPPoE.

Both endpoints MUST support the function of SCHC Compressor/ Decompressor (C/D) as shown in Figure 2.

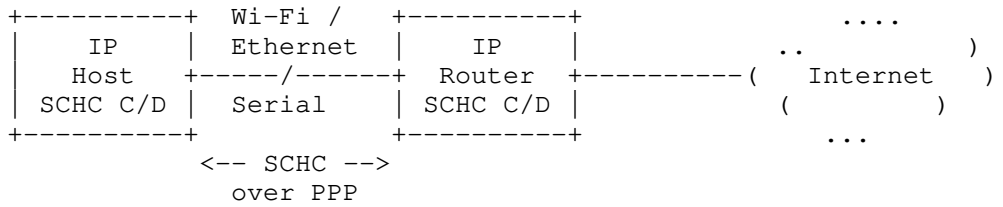


Figure 2: Typical Deployment

The SCHC Fragmenter/Reassembler (F/R) is generally not needed, because the maximum transmission unit (MTU) is expected to be large enough and SCHC only reduces the frame size vs. native IP. But it may be used to obtain a small protocol-independant frame size for the compressed packets, possibly way smaller than MTU.

A context may be generated for a particular upper layer application, such as a control loop using an industrial automation protocol, to protect the particular flow with a DetNet service. The context can be asymetric, e.g., when connecting a primary and a secondary endpoints, a client and a server, or a programmable logic controller with a sensor or an actuator.

4.2. SCHC Parameters

Compared to typical LPWANs, most serial links and emulations such as PPPoE are very fast and most of the constraints can be alleviated. For this reason, the SCHC profile for PPP is defined as follows:

RuleID numbering scheme: The RuleID for a compression rule is expressed as 2 bytes. The first (leftmost) 2 bits of that RuleId MUST be set to 0 This leaves 14 bits to index the rule. A SCHC compressed packet is always in the form:

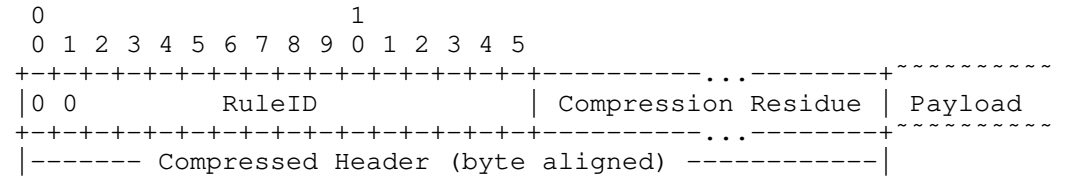


Figure 3: SCHC Compressed Packet

This specification only supports the No-ACK Mode of SCHC fragmentation as specified in section 8.4.1 of [SCHC]. The SCHC Fragment Header is 2 bytes long.

The RuleID for a fragmenation rule is expressed as 4 bits. The bits MUST all set to 1 for a fragmentation rule in No-ACK Mode. The DTag field is 11 bits long (T=11) and the FCN field is one bit (N=1), which is set to 1 on the last fragment as illustrated in Appendix B of [SCHC] and to 0 otherwise. There is no W field (M=0).

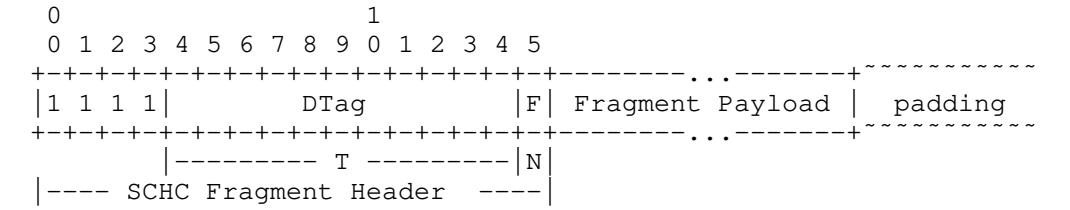


Figure 4: SCHC Fragment

The No-ACK mode has been designed under the assumption that data unit out-of-sequence delivery does not occur between the entity performing fragmentation and the entity performing reassembly and a DetNet PREOF function might be needed to reorder the fragments.

Maximum packet size: MAX_PACKET_SIZE is aligned to the PPP Link MTU.

Padding: The Compression Residue MUST be aligned to the L2 word. For Ethernet, the L2 word is one byte, so padding is needed up to the next byte boundary. If a compression rule produces a residue that is not byte aligned, then it is implicitly terminated with a statement that indicates padding till the next byte boundary. The padding bit is 0.

4.2.1. Resulting Packet Format

In the case of PPPoE, the sequence of compression and encapsulation is as follows:

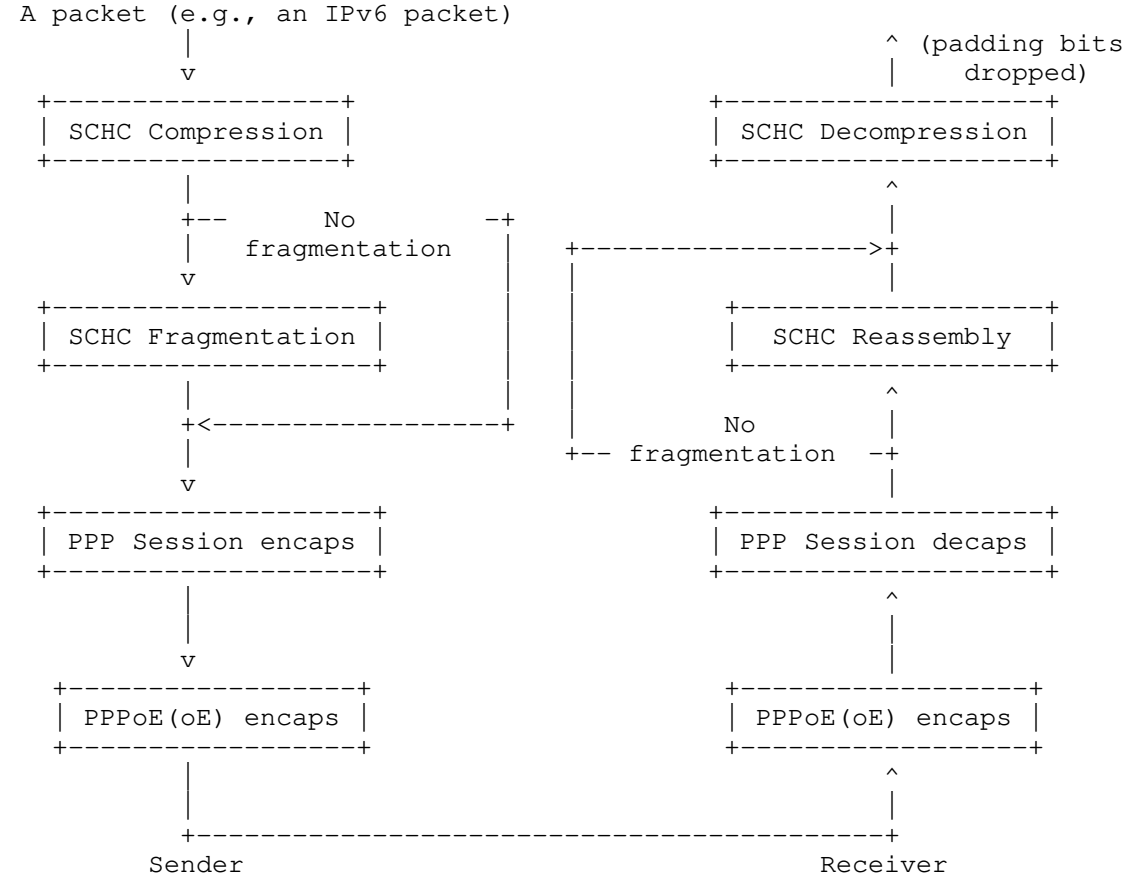


Figure 5: Stack Operation (no fragment)

In the case of PPPoE, a frame that transports an IPv6 packet compressed with SCHC with no fragmentation shows as follows:

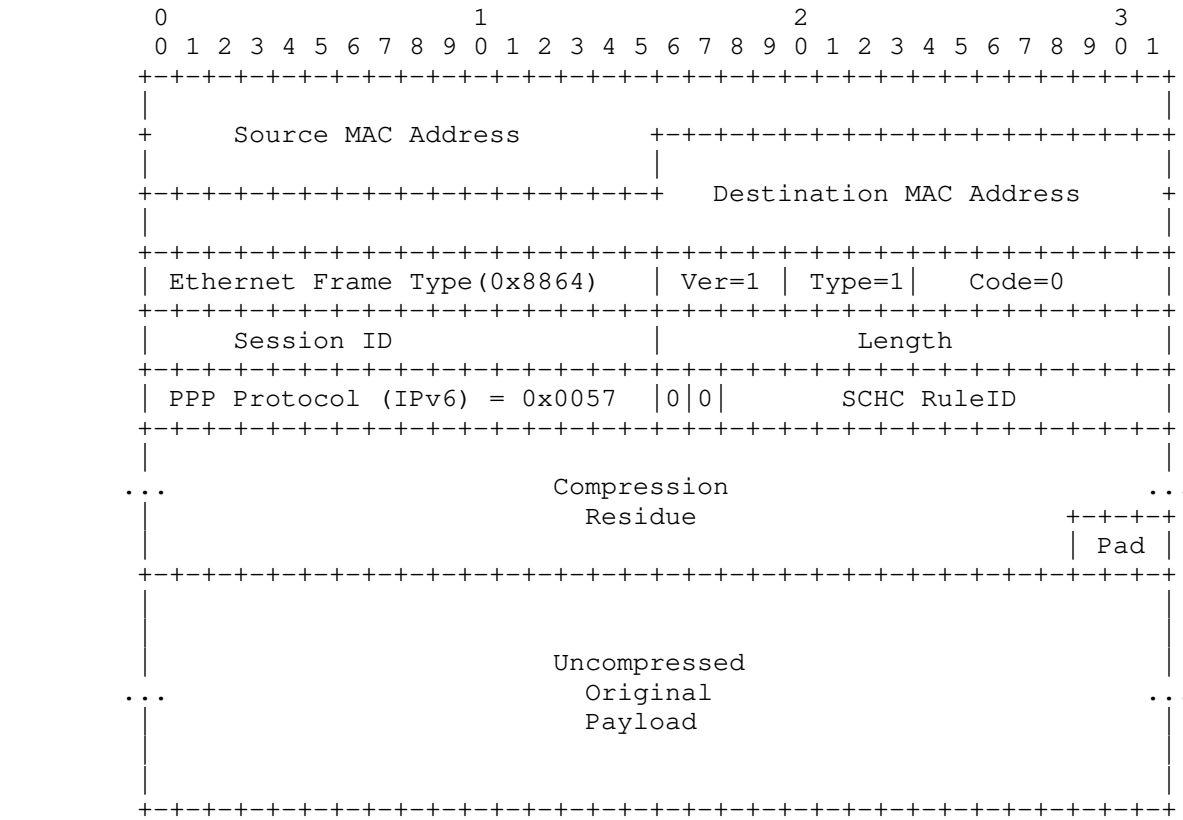


Figure 6: SCHC over PPP over Ethernet Format

4.3. Security Considerations

This draft enables to use the SCHC compression and fragmentation over PPP and therefore Ethernet and Wi-Fi with PPPoE. It inherits the possible threats against SCHC listed in the "Security considerations" section of [SCHC].

5. IANA Considerations

This document requests the allocation of a new value in the registry "IPv6-Compression-Protocol Types" for "SCHC". A suggested value is proposed in Table 1:

Value	Description	Reference
4	Static Context Header Compression (SCHC)	This document

Table 1: IP Header Compression Configuration Option Suboption Types

6. Acknowledgments

7. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2516] Mamakos, L., Lidl, K., Evarts, J., Carrel, D., Simone, D., and R. Wheeler, "A Method for Transmitting PPP Over Ethernet (PPPoE)", RFC 2516, DOI 10.17487/RFC2516, February 1999, <https://www.rfc-editor.org/info/rfc2516>.

[RFC5072] Varada, S., Ed., Haskins, D., and E. Allen, "IP Version 6 over PPP", RFC 5072, DOI 10.17487/RFC5072, September 2007, <https://www.rfc-editor.org/info/rfc5072>.

[RFC5172] Varada, S., Ed., "Negotiation for IPv6 Datagram Compression Using IPv6 Control Protocol", RFC 5172, DOI 10.17487/RFC5172, March 2008, <https://www.rfc-editor.org/info/rfc5172>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[SCHC] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zúñiga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724, DOI 10.17487/RFC8724, April 2020, <https://www.rfc-editor.org/info/rfc8724>.

8. Informative References

[SCHC_DATA_MODEL]
Minaburo, A. and L. Toutain, "Data Model for Static Context Header Compression (SCHC)", Work in Progress, Internet-Draft, draft-ietf-lpwan-schc-yang-data-model-03, 10 July 2020, <https://tools.ietf.org/html/draft-ietf-lpwan-schc-yang-data-model-03>.

[RAW Technologies]
Thubert, P., Cavalcanti, D., Vilajosana, X., Schmitt, C., and J. Farkas, "Reliable and Available Wireless Technologies", Work in Progress, Internet-Draft, draft-thubert-raw-technologies-05, 18 May 2020, <https://tools.ietf.org/html/draft-thubert-raw-technologies-05>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

[DetNet] Finn, N., Thubert, P., Varga, B., and J. Farkas, "Deterministic Networking Architecture", RFC 8655, DOI 10.17487/RFC8655, October 2019, <https://www.rfc-editor.org/info/rfc8655>.

[IEEE802.1TSNTG]
IEEE, "Time-Sensitive Networking (TSN) Task Group", <https://1.ieee802.org/tsn/>.

Author's Address

Pascal Thubert (editor)
Cisco Systems, Inc
Building D
45 Allee des Ormes - BP1200
06254 Mougins - Sophia Antipolis
France

Phone: +33 497 23 26 34
Email: pthubert@cisco.com