Stream: Internet Engineering Task Force (IETF)
RFC: 8931
Updates: 4944
Category: Standards Track
Published: November 2020
ISSN: 2070-1721
Author: P. Thubert, Ed.
Cisco Systems

RFC 8931
IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Selective Fragment Recovery

Abstract

This document updates RFC 4944 with a protocol that forwards individual fragments across a route-over mesh and recovers them end to end, with congestion control capabilities to protect the network.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8931.

Copyright Notice

Table of Contents

1. Introduction

In most Low-Power and Lossy Network (LLN) applications, the bulk of the traffic consists of small chunks of data (on the order of a few bytes to a few tens of bytes) at a time. Given that an IEEE Std 802.15.4 [IEEE.802.15.4] frame can carry a payload of 74 bytes or more, fragmentation is usually not required. However, and though this happens only occasionally, a number of mission-critical applications do require the capability to transfer larger chunks of data, for instance, to support the firmware upgrade of the LLN nodes or the extraction of logs from LLN nodes.

In the former case, the large chunk of data is transferred to the LLN node, whereas in the latter case, the large chunk flows away from the LLN node. In both cases, the size can be on the order of 10 KB or more, and an end-to-end reliable transport is required.

"Transmission of IPv6 Packets over IEEE 802.15.4 Networks" [RFC4944] defines the original IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) datagram fragmentation mechanism for LLNs. One critical issue with this original design is that routing an IPv6 [RFC8200] packet across a route-over mesh requires the reassembly of the packet at each hop. "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4" [6TiSCH] indicates that this may cause latency along a path and impact critical resources such as memory and battery; to alleviate those undesirable effects, it recommends using a 6LoWPAN Fragment Forwarding (6LFF) technique.

"On Forwarding 6LoWPAN Fragments over a Multihop IPv6 Network" [RFC8930] specifies the generic behavior that all 6LFF techniques including this specification follow, and it presents the associated caveats. In particular, the routing information is fully indicated in the first fragment, which is always forwarded first. With this specification, the first fragment is identified by a Sequence of 0 as opposed to a dispatch type in [RFC4944]. A state is formed and used to forward all the next fragments along the same path. The Datagram_Tag is locally significant to the Layer 2 source of the packet and is swapped at each hop; see Section 6. This specification encodes the Datagram_Tag in 1 byte, which will saturate if more than 256 datagrams transit in fragmented form over a single hop at the same time. This is not realistic at the time of this writing. Should this happen in a new 6LoWPAN technology, a node will need to use several link-layer addresses to increase its indexing capacity.

"Virtual reassembly buffers in 6LoWPAN" [LWIG-FRAG] proposes a 6LFF technique that is compatible with [RFC4944] without the need to define a new protocol. However, adding that capability alone to the local implementation of the original 6LoWPAN fragmentation would not address the inherent fragility of fragmentation (see [RFC8900]), in particular, the issues of resources locked on the reassembling endpoint and the wasted transmissions due to the loss of a single fragment in a whole datagram. [Kent] compares the unreliable delivery of fragments with a mechanism it calls "selective acknowledgments" that recovers the loss of a fragment individually. The paper illustrates the benefits that can be derived from such a method; see Figures 1, 2, and 3 in Section 2.3 of [Kent]. [RFC4944] has no selective recovery, and the whole datagram fails when one fragment is not delivered to the reassembling endpoint. Constrained memory resources are blocked on the reassembling endpoint until it times out, possibly causing the loss of subsequent packets that cannot be received for the lack of buffers.

That problem is exacerbated when forwarding fragments over multiple hops since a loss at an intermediate hop will not be discovered by either the fragmenting or the reassembling endpoints. Should this happen, the source will keep on sending fragments, wasting even more resources in the network since the datagram cannot arrive in its entirety, which possibly contributes to the condition that caused the loss. [RFC4944] is lacking a congestion control to avoid participating in a saturation that may have caused the loss of the fragment. It has no signaling to abort a multi-fragment transmission at any time and from either end, and if the capability to forward fragments is implemented, clean up the related state in the network.

This specification provides a method to forward fragments over, typically, a few hops in a route-over 6LoWPAN mesh and a selective acknowledgment to recover individual fragments between 6LoWPAN endpoints. The method can help limit the congestion loss in the network and addresses the requirements in Appendix B. Flow control is out of scope since the endpoints are expected to be able to store the full datagram. Deployments are expected to be managed and homogeneous, and an incremental transition requires a flag day.

2. Terminology

2.1. Requirements Language

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.

2.2. Background

This document uses 6LoWPAN terms and concepts that are presented in "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals" [RFC4919]; "Transmission of IPv6 Packets over IEEE 802.15.4 Networks" [RFC4944]; and "Problem Statement and Requirements for IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing" [RFC6606].

[RFC8930] discusses the generic concept of a Virtual Reassembly Buffer (VRB) and specifies behaviors and caveats that are common to a large family of 6LFF techniques including the mechanism specified by this document, which is fully inherited from that specification. It also defines terms used in this document: Compressed Form, Datagram_Tag, Datagram_Size, Fragment_Offset, and 6LoWPAN Fragment Forwarding endpoint (commonly abbreviated as only "endpoint").

Past experience with fragmentation has shown that misassociated or lost fragments can lead to poor network behavior and, occasionally, trouble at the application layer. The reader is encouraged to read "IPv4 Reassembly Errors at High Data Rates" [RFC4963] and follow the references for more information. That experience led to the definition of the "Path MTU Discovery for IP version 6" [RFC8201] protocol that limits fragmentation over the Internet. Specifically, in the case of UDP, valuable additional information can be found in "UDP Usage Guidelines" [RFC8085].

"The Benefits of Using Explicit Congestion Notification (ECN)" [RFC8087] provides useful information on the potential benefits and pitfalls of using ECN.

Quoting "Multiprotocol Label Switching Architecture" [RFC3031]:

> With MPLS, "packets are "labeled" before they are forwarded [along a Label Switched Path (LSP)]. At subsequent hops, there is no further analysis of the packet's network layer header. Rather, the label is used as an index into a table which specifies the next hop, and a new label".

[RFC8930] leverages MPLS to forward fragments that actually do not have a network-layer header, since the fragmentation occurs below IP, and this specification makes it reversible so the reverse path can be followed as well.

2.3. Other Terms

This specification uses the following terms:

RFRAG: Recoverable Fragment

RFRAG-ACK: Recoverable Fragment Acknowledgment

RFRAG Acknowledgment Request: An RFRAG with the Acknowledgment Request flag ("X" flag) set.

NULL bitmap: Refers to a bitmap with all bits set to zero.

FULL bitmap: Refers to a bitmap with all bits set to one.

Reassembling endpoint: The receiving endpoint.

Fragmenting endpoint: The sending endpoint.

Forward direction: The direction of a path, which is followed by the RFRAG.

Reverse direction: The reverse direction of a path, which is taken by the RFRAG-ACK.

3. Updating RFC 4944

This specification updates the fragmentation mechanism that is specified in [RFC4944] for use in route-over LLNs by providing a model where fragments can be forwarded end to end across a 6LoWPAN LLN and where fragments that are lost on the way can be recovered individually. A new format for fragments is introduced, and new dispatch types are defined in Section 5.

[RFC8138] allows modifying the size of a packet en route by removing the consumed hops in a compressed Routing Header. This requires that Fragment_Offset and Datagram_Size (defined in Section 5.1) also be modified en route, which is difficult to do in the uncompressed form. This specification expresses those fields in the compressed form and allows modifying them en route easily (more in Section 4.4).

To be consistent with Section 2 of [RFC6282], for the fragmentation mechanism described in Section 5.3 of [RFC4944], any header that cannot fit within the first fragment **MUST NOT** be compressed when using the fragmentation mechanism described in this specification.

4. Extending RFC 8930

This specification implements the generic 6LFF technique defined in [RFC8930] and provides end-to-end fragment recovery and congestion control mechanisms.

4.1. Slack in the First Fragment

[RFC8930] allows for a refragmentation operation in intermediate nodes, whereby the trailing bytes from a given fragment may be left in the VRB to be added as the heading bytes in the next fragment. This solves the case when the outgoing fragment needs more space than the incoming fragment; that case may arise when the 6LoWPAN header compression is not as efficient on the outgoing link or if the Link MTU is reduced.

This specification cannot allow that refragmentation operation since the fragments are recovered end to end based on a sequence number. The Fragment_Size **MUST** be tailored to fit the minimal MTU along the path, and the first fragment that contains a 6LoWPAN compressed header **MUST** have enough slack to enable a less-efficient compression in the next hops to still fit within the Link MTU.

For instance, if the fragmenting endpoint is also the 6LoWPAN compression endpoint, it will elide the Interface ID (IID) of the source IPv6 address when it matches the link-layer address [RFC6282]. In that case, it **MUST** leave slack in the first fragment as the if MTU on the first hop was 8 bytes less, so the next hop can expand the IID within the same fragment within MTU.

4.2. Gap between Frames

[RFC8930] requires that a configurable interval of time be inserted between transmissions to the same next hop and, in particular, between fragments of a same datagram. In the case of half duplex interfaces, this inter-frame gap ensures that the next hop is done forwarding the previous frame and is capable of receiving the next one.

In the case of a mesh operating at a single frequency with omnidirectional antennas, a larger inter-frame gap is required to protect the frame against hidden terminal collisions with the previous frame of the same flow that is still progressing along a common path.

The inter-frame gap is useful even for unfragmented datagrams, but it becomes a necessity for fragments that are typically generated in a fast sequence and are all sent over the exact same path.

4.3. Congestion Control

The inter-frame gap is the only protection that [RFC8930] imposes by default. This document enables grouping fragments in windows and requesting intermediate acknowledgments, so the number of in-flight fragments can be bounded. This document also adds an ECN mechanism that can be used to protect the network by adapting the size of the window, the size of the fragments, and/or the inter-frame gap.

This specification enables the fragmenting endpoint to apply a congestion control mechanism to tune those parameters, but the mechanism itself is out of scope. In most cases, the expectation is that most datagrams will require only a few fragments, and that only the last fragment will be acknowledged. A basic implementation of the fragmenting endpoint is NOT **REQUIRED** to vary the size of the window, the duration of the inter-frame gap, or the size of a fragment in the middle of the transmission of a datagram, and it **MAY** ignore the ECN signal or simply reset the window to 1 (see Appendix C) until the end of this datagram upon detecting a congestion.

An intermediate node that experiences a congestion **MAY** set the ECN bit in a fragment, and the reassembling endpoint echoes the ECN bit at most once at the next opportunity to acknowledge back.

The size of the fragments is typically computed from the Link MTU to maximize the size of the resulting frames. The size of the window and the duration of the inter-frame gap **SHOULD** be configurable, to reduce the chances of congestion and to follow the general recommendations in [RFC8930], respectively.

4.4. Modifying the First Fragment

The compression of the hop limit, of the source and destination addresses in the IPv6 header, and of the Routing Header, which are all in the first fragment, may change en route in a route-over mesh LLN. If the size of the first fragment is modified, then the intermediate node **MUST** adapt the Datagram_Size, encoded in the Fragment_Size field, to reflect that difference.

The intermediate node **MUST** also save the difference of Datagram_Size of the first fragment in the VRB and add it to the Fragment_Offset of all the subsequent fragments that it forwards for that datagram. In the case of a Source Routing Header 6LoWPAN Routing Header (SRH-6LoRH) [RFC8138] being consumed and thus reduced, that difference is negative, meaning that the Fragment_Offset is decremented by the number of bytes that were consumed.

5. New Dispatch Types and Headers

This document specifies an alternative to the 6LoWPAN fragmentation sub-layer [RFC4944] to emulate a Link MTU up to 2048 bytes for the upper layer, which can be the 6LoWPAN header compression sub-layer that is defined in "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks" [RFC6282]. This specification also provides a reliable transmission of the fragments over a multi-hop 6LoWPAN route-over mesh network and a minimal congestion control to reduce the chances of congestion loss.

A 6LoWPAN Fragment Forwarding [RFC8930] technique derived from MPLS enables the forwarding of individual fragments across a 6LoWPAN route-over mesh without reassembly at each hop. The Datagram_Tag is used as a label; it is locally unique to the node that owns the source link-layer address of the fragment, so together the link-layer address and the label can identify the fragment globally within the lifetime of the datagram. A node may build the Datagram_Tag in its own locally significant way, as long as the chosen Datagram_Tag stays unique to the particular datagram for its lifetime. The result is that the label does not need to be globally unique, but it must be swapped at each hop as the source link-layer address changes.

In the following sections, a Datagram_Tag extends the semantics defined in "Fragmentation Type and Header" (see Section 5.3 of [RFC4944]). The Datagram_Tag is a locally unique identifier for the datagram from the perspective of the sender. This means that the Datagram_Tag identifies a datagram uniquely in the network when associated with the source of the datagram. As the datagram gets forwarded, the source changes, and the Datagram_Tag must be swapped as detailed in [RFC8930].

This specification extends [RFC4944] with two new dispatch types for RFRAG and the RFRAG-ACK that is received back. The new 6LoWPAN dispatch types are taken from [RFC8025], as indicated in Table 1 of Section 9.

5.1. Recoverable Fragment Dispatch Type and Header

In this specification, if the packet is compressed, the size and offset of the fragments are expressed with respect to the compressed form of the packet, as opposed to the uncompressed (native) form.

The format of the fragment header is shown in Figure 1. It is the same for all fragments even though the Fragment_Offset is overloaded. The format has a length and an offset, as well as a Sequence field. This would be redundant if the offset was computed as the product of the Sequence by the length, but this is not the case. The position of a fragment in the reassembly buffer is correlated with neither the value of the Sequence field nor the order in which the fragments are received. This enables splitting fragments to cope with an MTU deduction; see the example of fragment Sequence 5 that is retried end to end as smaller fragment Sequences 13 and 14 in Section 6.2.

The first fragment is recognized by a Sequence of 0; it carries its Fragment_Size and the Datagram_Size of the compressed packet before it is fragmented, whereas the other fragments carry their Fragment_Size and Fragment_Offset. The last fragment for a datagram is recognized when its Fragment_Offset and its Fragment_Size add up to the stored Datagram_Size of the packet identified by the sender link-layer address and the Datagram_Tag.

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
                                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                |1 1 1 0 1 0 0|E|  Datagram_Tag |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|X| Sequence|   Fragment_Size   |       Fragment_Offset         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                                     X set == Ack-Request
```

Figure 1: RFRAG Dispatch Type and Header

X: 1 bit; Ack-Request. When set, the fragmenting endpoint requires an RFRAG Acknowledgment from the reassembling endpoint.

E: 1 bit; Explicit Congestion Notification. The "E" flag is cleared by the source of the fragment and set by intermediate routers to signal that this fragment experienced congestion along its path.

Fragment_Size: 10-bit unsigned integer. The size of this fragment in a unit that depends on link-layer technology. Unless overridden by a more specific specification, that unit is the byte, which allows fragments up to 1023 bytes.

Datagram_Tag: 8 bits. An identifier of the datagram that is locally unique to the link-layer sender.

Sequence: 5-bit unsigned integer. The sequence number of the fragment in the acknowledgment bitmap. Fragments are numbered as [0..N], where N is in [0..31]. A Sequence of 0 indicates the first fragment in a datagram, but non-zero values are not indicative of the position in the reassembly buffer.

Fragment_Offset: 16-bit unsigned integer.

When the Fragment_Offset is set to a non-zero value, its semantics depend on the value of the Sequence field as follows:

- For a first fragment (i.e., with a Sequence of 0), this field indicates the Datagram_Size of the compressed datagram, to help the reassembling endpoint allocate an adapted buffer for the reception and reassembly operations. The fragment may be stored for local reassembly. Alternatively, it may be routed based on the destination IPv6 address. In that case, a VRB state must be installed as described in Section 6.1.1.
- When the Sequence is not 0, this field indicates the offset of the fragment in the compressed form of the datagram. The fragment may be added to a local reassembly buffer or forwarded based on an existing VRB as described in Section 6.1.2.

A Fragment_Offset that is set to a value of 0 indicates an abort condition, and all states regarding the datagram should be cleaned up once the processing of the fragment is complete; the processing of the fragment depends on whether there is a VRB already established for this datagram and if the next hop is still reachable:

- if a VRB already exists and the next hop is still reachable, the fragment is to be forwarded along the associated LSP as described in Section 6.1.2, without checking the value of the Sequence field.
- else, if the Sequence is 0, then the fragment is to be routed as described in Section 6.1.1, but no state is conserved afterwards. In that case, the session, if it exists, is aborted, and the packet is also forwarded in an attempt to clean up the next hops along the path indicated by the IPv6 header (possibly including a Routing Header).
- else (the Sequence is non-zero and either no VRB exists or the next hop is unavailable), the fragment cannot be forwarded or routed; the fragment is discarded and an abort RFRAG-ACK is sent back to the source as described in Section 6.1.2.

Recoverable Fragments are sequenced, and a bitmap is used in the RFRAG Acknowledgment to indicate the received fragments by setting the individual bits that correspond to their sequence.

There is no requirement on the reassembling endpoint to check that the received fragments are consecutive and non-overlapping. This may be useful, in particular, in the case where the MTU changes and a fragment Sequence is retried with a smaller Fragment_Size, with the remainder of the original fragment being retried with new Sequence values. The fragmenting endpoint knows that the datagram is fully received when the acknowledged fragments cover the whole datagram, which is implied by a FULL bitmap.

5.2. RFRAG Acknowledgment Dispatch Type and Header

This specification also defines a 4-byte RFRAG Acknowledgment Bitmap that is used by the reassembling endpoint to selectively confirm the reception of individual fragments. A given offset in the bitmap maps one to one with a given sequence number and indicates which fragment is acknowledged as follows:

```
                      1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |            RFRAG Acknowledgment Bitmap                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  ^                 ^
  |                 |    bitmap indicating whether:
  |                 +----- Fragment with Sequence 9 was received
  +----------------------- Fragment with Sequence 0 was received
```

Figure 2: RFRAG Acknowledgment Bitmap Encoding

Figure 3 shows an example RFRAG Acknowledgment Bitmap that indicates that all fragments from Sequence 0 to 20 were received, except for fragments 1, 2, and 16, which were lost and must be retried.

```
                      1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |1|0|0|1|1|1|1|1|1|1|1|1|1|1|1|1|0|1|1|1|1|0|0|0|0|0|0|0|0|0|0|0|
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: Example RFRAG Acknowledgment Bitmap

The RFRAG Acknowledgment Bitmap is included in an RFRAG Acknowledgment header, as follows:

```
                      1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
                                 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                 |1 1 1 0 1 0 1|E|  Datagram_Tag |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |          RFRAG Acknowledgment Bitmap (32 bits)                |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: RFRAG Acknowledgment Dispatch Type and Header

E: 1 bit; Explicit Congestion Notification Echo.

When set, the fragmenting endpoint indicates that at least one of the acknowledged fragments was received with an Explicit Congestion Notification, indicating that the path followed by the fragments is subject to congestion. See more details in Appendix C.

Datagram_Tag: 8 bits; an identifier of the datagram that is locally unique to the link-layer recipient.

RFRAG Acknowledgment Bitmap: An RFRAG Acknowledgment Bitmap, whereby setting the bit at offset x indicates that fragment x was received, as shown in Figure 2. A NULL bitmap indicates that the fragmentation process is aborted. A FULL bitmap indicates that the fragmentation process is complete; all fragments were received at the reassembly endpoint.

6. Fragment Recovery

The RFRAG header is used to transport a fragment and optionally request an RFRAG-ACK that confirms the reception of one or more fragments. An RFRAG-ACK is carried as a standalone fragment header (i.e., with no 6LoWPAN payload) in a message that is propagated back to the fragmenting endpoint. To achieve this, each hop that performed an MPLS-like operation on fragments reverses that operation for the RFRAG-ACK by sending a frame from the next hop to the previous hop as known by its link-layer address in the VRB. The Datagram_Tag in the RFRAG-ACK is unique to the reassembling endpoint and is enough information for an intermediate hop to locate the VRB that contains the Datagram_Tag used by the previous hop and the Layer 2 information associated with it (interface and link-layer address).

The fragmenting endpoint (i.e., the node that fragments the packets at the 6LoWPAN level) also controls the number of acknowledgments by setting the Ack-Request flag in the RFRAG packets.

The fragmenting endpoint may set the Ack-Request flag on any fragment to perform congestion control by limiting the number of outstanding fragments, which are the fragments that have been sent but for which reception or loss was not positively confirmed by the reassembling endpoint. The maximum number of outstanding fragments is controlled by the Window-Size. It is configurable and may vary in case of ECN notification. When the endpoint that reassembles the packets at the 6LoWPAN level receives a fragment with the Ack-Request flag set, it **MUST** send an RFRAG-ACK back to the originator to confirm reception of all the fragments it has received so far.

The Ack-Request ("X") set in an RFRAG marks the end of a window. This flag **MUST** be set on the last fragment if the fragmenting endpoint wishes to perform an automatic repeat request (ARQ) process for the datagram, and it **MAY** be set in any intermediate fragment for the purpose of congestion control.

This ARQ process **MUST** be protected by a Retransmission Timeout (RTO) timer, and the fragment that carries the "X" flag **MAY** be retried upon a timeout for a configurable number of times (see Section 7.1) with an exponential backoff. Upon exhaustion of the retries, the fragmenting endpoint may either abort the transmission of the datagram or resend the first fragment with an "X" flag set in order to establish a new path for the datagram and obtain the list of fragments that were received over the old path in the acknowledgment bitmap. When the fragmenting endpoint knows that an underlying link-layer mechanism protects the fragments, it may refrain from using the RFRAG Acknowledgment mechanism and never set the Ack-Request bit.

The reassembling endpoint **MAY** issue unsolicited acknowledgments. An unsolicited acknowledgment signals to the fragmenting endpoint that it can resume sending in case it has reached its maximum number of outstanding fragments. Another use is to inform the fragmenting endpoint that the reassembling endpoint aborted the processing of an individual datagram.

The RFRAG Acknowledgment carries an ECN indication for congestion control (see Appendix C). The reassembling endpoint of a fragment with the "E" (ECN) flag set **MUST** echo that information at most once by setting the "E" (ECN) flag in the next RFRAG-ACK.

In order to protect the datagram, the fragmenting endpoint transfers a controlled number of fragments and flags to the last fragment of a window with an RFRAG Acknowledgment Request. The reassembling endpoint **MUST** acknowledge a fragment with the acknowledgment request bit set. If any fragment immediately preceding an acknowledgment request is still missing, the reassembling endpoint **MAY** intentionally delay its acknowledgment to allow in-transit fragments to arrive. Because it might defeat the round-trip time computation, delaying the acknowledgment should be configurable and not enabled by default.

When enough fragments are received to cover the whole datagram, the reassembling endpoint reconstructs the packet, passes it to the upper layer, sends an RFRAG-ACK on the reverse path with a FULL bitmap, and arms a short timer, e.g., on the order of an average round-trip time in the network. The FULL bitmap is used as opposed to a bitmap that acknowledges only the received fragments to let the intermediate nodes know that the datagram is fully received. As the timer runs, the reassembling endpoint absorbs the fragments that were still in flight for that datagram without creating a new state, acknowledging the ones that bear an Ack-Request with an FRAG Acknowledgment and the FULL bitmap. The reassembling endpoint aborts the communication if fragments with a matching source and Datagram-Tag continue to be received after the timer expires.

Note that acknowledgments might consume precious resources, so the use of unsolicited acknowledgments **SHOULD** be configurable and not enabled by default.

An observation is that streamlining the forwarding of fragments generally reduces the latency over the LLN mesh, providing room for retries within existing upper-layer reliability mechanisms. The fragmenting endpoint protects the transmission over the LLN mesh with a retry timer that is configured for a use case and may be adapted dynamically, e.g., according to the method detailed in [RFC6298]. It is expected that the upper-layer retry mechanism obeys the recommendations in [RFC8085], in which case a single round of fragment recovery should fit within the upper-layer recovery timers.

Fragments **MUST** be sent in a round-robin fashion: the sender **MUST** send all the fragments for a first time before it retries any lost fragment; lost fragments **MUST** be retried in sequence, oldest first. This mechanism enables the receiver to acknowledge fragments that were delayed in the network before they are retried.

When a single radio frequency is used by contiguous hops, the fragmenting endpoint **SHOULD** insert a delay between the frames (e.g., carrying fragments) that are sent to the same next hop. The delay **SHOULD** cover multiple transmissions so as to let a frame progress a few hops and avoid hidden terminal issues. This precaution is not required on channel hopping technologies such as Time-Slotted Channel Hopping (TSCH) [RFC6554], where nodes that communicate at Layer 2 are scheduled to send and receive, respectively, and different hops operate on different channels.

6.1. Forwarding Fragments

This specification inherits from [RFC8930] and proposes a Virtual Reassembly Buffer technique to forward fragments with no intermediate reconstruction of the entire datagram.

The IPv6 header **MUST** be placed in the first fragment in full to enable the routing decision. The first fragment is routed and creates an LSP from the fragmenting endpoint to the reassembling endpoint. The next fragments are label switched along that LSP. As a consequence, the next fragments can only follow the path that was set up by the first fragment; they cannot follow an alternate route. The Datagram_Tag is used to carry the label, which is swapped in each hop.

If the first fragment is too large for the path MTU, it will repeatedly fail and never establish an LSP. In that case, the fragmenting endpoint **MAY** retry the same datagram with a smaller Fragment_Size, in which case it **MUST** abort the original attempt and use a new Datagram_Tag for the new attempt.

6.1.1. Receiving the First Fragment

In route-over mode, the source and destination link-layer addresses in a frame change at each hop. The label that is formed and placed in the Datagram_Tag by the sender is associated with the source link-layer address and only valid (and temporarily unique) for that source link-layer address.

Upon receiving the first fragment (i.e., with a Sequence of 0), an intermediate router creates a VRB and the associated LSP state indexed by the incoming interface, the previous-hop link-layer address, and the Datagram_Tag and forwards the fragment along the IPv6 route that matches the destination IPv6 address in the IPv6 header until it reaches the reassembling endpoint, as prescribed by [RFC8930]. The LSP state enables matching the next incoming fragments of a datagram to the abstract forwarding information of the next interface, source and next-hop link-layer addresses, and the swapped Datagram_Tag.

In addition, the router also forms a reverse LSP state indexed by the interface to the next hop, the link-layer address the router uses as source for that datagram, and the swapped Datagram_Tag. This reverse LSP state enables matching the tuple (interface, destination link-layer address, Datagram_Tag) found in an RFRAG-ACK to the abstract forwarding information (previous interface, previous link-layer address, Datagram_Tag) used to forward the RFRAG-ACK back to the fragmenting endpoint.

6.1.2. Receiving the Next Fragments

Upon receiving the next fragment (i.e., with a non-zero Sequence), an intermediate router looks up an LSP indexed by the tuple (incoming interface, previous-hop link-layer address, Datagram_Tag) found in the fragment. If it is found, the router forwards the fragment using the associated VRB as prescribed by [RFC8930].

If the VRB for the tuple is not found, the router builds an RFRAG-ACK to abort the transmission of the packet. The resulting message has the following information:

- The source and destination link-layer addresses are swapped from those found in the fragment, and the same interface is used
- The Datagram_Tag is set to the Datagram_Tag found in the fragment
- A NULL bitmap is used to signal the abort condition

At this point, the router is all set and can send the RFRAG-ACK back to the previous router. The RFRAG-ACK should normally be forwarded all the way to the source using the reverse LSP state in the VRBs in the intermediate routers as described in the next section.

[RFC8930] indicates that the reassembling endpoint stores "the actual packet data from the fragments received so far, in a form that makes it possible to detect when the whole packet has been received and can be processed or forwarded". How this is computed is implementation specific, but it relies on receiving all the bytes up to the Datagram_Size indicated in the first fragment. An implementation may receive overlapping fragments as the result of retries after an MTU change.

6.2. Receiving RFRAG Acknowledgments

Upon receipt of an RFRAG-ACK, the router looks up a reverse LSP indexed by the interface and destination link-layer address of the received frame and the received Datagram_Tag in the RFRAG-ACK. If it is found, the router forwards the fragment using the associated VRB as prescribed by [RFC8930], but it uses the reverse LSP so that the RFRAG-ACK flows back to the fragmenting endpoint.

If the reverse LSP is not found, the router **MUST** silently drop the RFRAG-ACK message.

Either way, if the RFRAG-ACK indicates that the fragment was entirely received (FULL bitmap), it arms a short timer, and upon timeout, the VRB and all the associated states are destroyed. Until the timer elapses, fragments of that datagram may still be received, e.g., if the RFRAG-ACK was lost on the path back, and the source retried the last fragment. In that case, the router generates an RFRAG-ACK with a FULL bitmap back to the fragmenting endpoint if an acknowledgment was requested; else, it silently drops the fragment.

This specification does not provide a method to discover the number of hops or the minimal value of MTU along those hops. In a typical case, the MTU is constant and is the same across the network. But should the minimal MTU along the path decrease, it is possible to retry a long fragment (say a Sequence of 5) with several shorter fragments with a Sequence that was not used before (e.g., 13 and 14). Fragment 5 is marked as abandoned and will not be retried anymore. Note that when this mechanism is in place, it is hard to predict the total number of fragments that will be needed or the final shape of the bitmap that would cover the whole packet. This is why the FULL bitmap is used when the reassembling endpoint gets the whole datagram regardless of which fragments were actually used to do so. Intermediate nodes will know unambiguously that the process is complete. Note that Path MTU Discovery is out of scope for this document.

6.3. Aborting the Transmission of a Fragmented Packet

A reset is signaled on the forward path with a pseudo fragment that has the Fragment_Offset set to 0. The sender of a reset **SHOULD** also set the Sequence and Fragment_Size field to 0.

When the fragmenting endpoint or a router on the path decides that a packet should be dropped and the fragmentation process aborted, it generates a reset pseudo fragment and forwards it down the fragment path.

Each router along the path forwards the pseudo fragment in turn based on the VRB state. If an acknowledgment is not requested, the VRB and all associated states are destroyed.

Upon reception of the pseudo fragment, the reassembling endpoint cleans up all resources for the packet associated with the Datagram_Tag. If an acknowledgment is requested, the reassembling endpoint responds with a NULL bitmap.

On the other hand, the reassembling endpoint might need to abort the processing of a fragmented packet for internal reasons, for instance, if it is out of reassembly buffers, already uses all 256 possible values of the Datagram_Tag, or keeps receiving fragments beyond a reasonable time while it considers that this packet is already fully reassembled and was passed to the upper layer. In that case, the reassembling endpoint **SHOULD** indicate so to the fragmenting endpoint with a NULL bitmap in an RFRAG-ACK.

The RFRAG-ACK is forwarded all the way back to the source of the packet and cleans up all resources on the path. Upon an acknowledgment with a NULL bitmap, the fragmenting endpoint **MUST** abort the transmission of the fragmented datagram with one exception: in the particular case of the first fragment, it **MAY** decide to retry via an alternate next hop instead.

6.4. Applying Recoverable Fragmentation along a Diverse Path

The text above can be read with the assumption of a serial path between a source and a destination. The IPv6 over the TSCH mode of IEEE 802.15.4e (6TiSCH) architecture (see Section 4.5.3 of [6TiSCH]) defines the concept of a Track that can be a complex path between a source and a destination with Packet ARQ, Replication, Elimination, and Overhearing (PAREO) along the Track. This specification can be used along any subset of the complex Track where the first fragment is flooded. The last RFRAG Acknowledgment is flooded on that same subset in the reverse direction. Intermediate RFRAG Acknowledgments can be flooded on any sub-subset of that reverse subset that reaches back to the source.

7. Management Considerations

This specification extends [RFC8930] and requires the same parameters in the reassembling endpoint and on intermediate nodes. There is no new parameter as echoing ECN is always on. These parameters typically include the reassembly timeout at the reassembling endpoint, an inactivity cleanup timer on the intermediate nodes, and the number of messages that can be processed in parallel in all nodes.

The configuration settings introduced by this specification only apply to the fragmenting endpoint, which is in full control of the transmission. LLNs vary a lot in size (there can be thousands of nodes in a mesh), in speed (from 10 Kbps to several Mbps at the PHY layer), in traffic density, and in optimizations that are desired (e.g., the selection of a Routing Protocol for LLNs (RPL) [RFC6550] Objective Function [RFC6552] impacts the shape of the routing graph).

For that reason, only very generic guidance can be given on the settings of the fragmenting endpoint and on whether complex algorithms are needed to perform congestion control or to estimate the round-trip time. To cover the most complex use cases, this specification enables the fragmenting endpoint to vary the fragment size, the window size, and the inter-frame gap based on the number of losses, the observed variations of the round-trip time, and the setting of the ECN bit.

7.1. Protocol Parameters

The management system **SHOULD** be capable of providing the parameters listed in this section, and an implementation **MUST** abide by those parameters and, in particular, never exceed the minimum and maximum configured boundaries.

An implementation should consider the generic recommendations from the IETF in the matter of congestion control and rate management for IP datagrams in [RFC8085]. An implementation may perform congestion control by using a dynamic value of the window size (Window_Size), adapting the fragment size (Fragment_Size), and potentially reducing the load by inserting an inter-frame gap that is longer than necessary. In a large network where nodes contend for the bandwidth, a larger Fragment_Size consumes less bandwidth but also reduces fluidity and incurs higher chances of loss in transmission.

This is controlled by the following parameters:

inter-frame gap: The inter-frame gap indicates the minimum amount of time between transmissions. The inter-frame gap controls the rate at which fragments are sent, the ratio of air time, and the amount of memory in intermediate nodes that a particular datagram will use. It can be used as a flow control, a congestion control, and/or a collision control measure. It **MUST** be set at a minimum to a value that protects the propagation of one transmission against collision with next [RFC8930]. In a wireless network that uses the same frequency along a path, this may represent the time for a frame to progress over multiple hops (see more in Section 4.2). It **SHOULD** be augmented beyond this as necessary to protect the network against congestion.

MinFragmentSize: The MinFragmentSize is the minimum value for the Fragment_Size. It **MUST** be lower than the minimum value of smallest 1-hop MTU that can be encountered along the path.

OptFragmentSize: The OptFragmentSize is the value for the Fragment_Size that the fragmenting endpoint should use to start with. It is greater than or equal to MinFragmentSize. It is less than or equal to MaxFragmentSize. For the first fragment, it must account for the expansion of the IPv6 addresses and of the Hop Limit field within MTU. For all fragments, it is a balance between the expected fluidity and the overhead of link-layer and 6LoWPAN headers. For a small MTU, the idea is to keep it close to the maximum, whereas for larger MTUs, it might make sense to keep it short enough so that the duty cycle of the transmitter is bounded, e.g., to transmit at least 10 frames per second.

MaxFragmentSize: The MaxFragmentSize is the maximum value for the Fragment_Size. It **MUST** be lower than the maximum value of the smallest 1-hop MTU that can be encountered along the path. A large value augments the chances of buffer bloat and transmission loss. The value **MUST** be less than 512 if the unit that is defined for the PHY layer is the byte.

Window_Size: The Window_Size **MUST** be at least 1 and less than 33.

- If the round-trip time is known, the Window_Size **SHOULD** be set to the round-trip time divided by the time per fragment; that is, the time to transmit a fragment plus the inter-frame gap.

Otherwise:

- A window_size of 32 indicates that only the last fragment is to be acknowledged in each round. This is the **RECOMMENDED** value in a half-duplex LLN where the fragment acknowledgment consumes roughly the same bandwidth on the same links as the fragments themselves.
- If it is set to a smaller value, more acks are generated. In a full-duplex network, the load on the forward path will be lower, and a small value of 3 **SHOULD** be configured.

An implementation may perform its estimate of the RTO or use a configured one. The ARQ process is controlled by the following parameters:

MinARQTimeOut: The minimum amount of time a node should wait for an RFRAG Acknowledgment before it takes the next action. It **MUST** be more than the maximum expected round-trip time in the respective network.

OptARQTimeOut: The initial value of the RTO, which is the amount of time that a fragmenting endpoint should wait for an RFRAG Acknowledgment before it takes the next action. It is greater than or equal to MinARQTimeOut. It is less than or equal to MaxARQTimeOut. See Appendix C for recommendations on computing the round-trip time. By default, a value of 3 times the maximum expected round-trip time in the respective network is **RECOMMENDED**.

MaxARQTimeOut: The maximum amount of time a node should wait for the RFRAG Acknowledgment before it takes the next action. It must cover the longest expected round-trip time and be several times less than the timeout that covers the recomposition buffer at the reassembling endpoint, which is typically on the order of the minute. An upper bound can be estimated to ensure that the datagram is either fully transmitted or dropped before an upper layer decides to retry it.

MaxFragRetries: The maximum number of retries for a particular fragment. A default value of 3 is **RECOMMENDED**. An upper bound can be estimated to ensure that the datagram is either fully transmitted or dropped before an upper layer decides to retry it.

MaxDatagramRetries: The maximum number of retries from scratch for a particular datagram. A default value of 1 is **RECOMMENDED**. An upper bound can be estimated to ensure that the datagram is either fully transmitted or dropped before an upper layer decides to retry it.

An implementation may be capable of performing congestion control based on ECN; see Appendix C. This is controlled by the following parameter:

UseECN: Indicates whether the fragmenting endpoint should react to ECN. The fragmenting endpoint may react to ECN by varying the Window_Size between MinWindowSize and MaxWindowSize, varying the Fragment_Size between MinFragmentSize and MaxFragmentSize, and/or increasing or reducing the inter-frame gap. With this specification, if UseECN is set and a fragmenting endpoint detects a congestion, it may apply a congestion control method until the end of the datagram, whereas if UseECN is reset, the endpoint does not react to congestion. Future specifications may provide additional parameters and capabilities.

7.2. Observing the Network

The management system should monitor the number of retries and ECN settings that can be observed from the perspective of the fragmenting endpoint with respect to the reassembling endpoint and reciprocally. It may then tune the optimum size of Fragment_Size and of Window_Size, OptFragmentSize, and OptWindowSize, respectively, at the fragmenting endpoint towards a particular reassembling endpoint, which is applicable to the next datagrams. It will preferably tune the inter-frame gap to increase the spacing between fragments of the same datagram and reduce the buffer bloat in the intermediate node that holds one or more fragments of that datagram.

8. Security Considerations

This document specifies an instantiation of a 6LFF technique and inherits from the generic description in [RFC8930]. The considerations in the Security Considerations section of [RFC8930] equally apply to this document.

In addition to the threats detailed therein, an attacker that is on path can prematurely end the transmission of a datagram by sending a RFRAG Acknowledgment to the fragmenting endpoint. It can also cause extra transmissions of fragments by resetting bits in the RFRAG Acknowledgment Bitmap and of RFRAG Acknowledgments by forcing the Ack-Request bit in fragments that it forwards.

As indicated in [RFC8930], secure joining and link-layer security are **REQUIRED** to protect against those attacks, as the fragmentation protocol does not include any native security mechanisms.

This specification does not recommend a particular algorithm for the estimation of the duration of the RTO that covers the detection of the loss of a fragment with the "X" flag set; regardless, an attacker on the path may slow down or discard packets, which in turn can affect the throughput of fragmented packets.

Compared to [RFC4944], this specification reduces the Datagram_Tag to 8 bits, and the tag wraps faster than with [RFC4944]. But for a constrained network where a node is expected to be able to hold only one or a few large packets in memory, 256 is still a large number. Also, the acknowledgment mechanism allows cleaning up the state rapidly once the packet is fully transmitted or aborted.

The abstract Virtual Recovery Buffer from [RFC8930] may be used to perform a Denial-of-Service (DoS) attack against the intermediate routers since the routers need to maintain a state per flow. The particular VRB implementation technique described in [LWIG-FRAG] allows realigning which data goes in which fragment; this causes the intermediate node to store a portion of the data, which adds an attack vector that is not present with this specification. With this specification, the data that is transported in each fragment is conserved, and the state to keep does not include any data that would not fit in the previous fragment.

9. IANA Considerations

This document allocates two patterns for a total of four dispatch values for Recoverable Fragments from the "Dispatch Type Field" registry that was created by [RFC4944] and reformatted by "IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Paging Dispatch" [RFC8025].

Bit Pattern	Page	Header Type	Reference
11 10100x	0	RFRAG - Recoverable Fragment	RFC 8931

Bit Pattern	Page	Header Type	Reference
11 10100x	1-14	Unassigned	
11 10100x	15	Reserved for Experimental Use	RFC 8025
11 10101x	0	RFRAG-ACK - RFRAG Acknowledgment	RFC 8931
11 10101x	1-14	Unassigned	
11 10101x	15	Reserved for Experimental Use	RFC 8025

Table 1: Additional Dispatch Value Bit Patterns

10. References

10.1. 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>.

[RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <https://www.rfc-editor.org/info/rfc4919>.

[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>.

[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>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

[RFC6606] Kim, E., Kaspar, D., Gomez, C., and C. Bormann, "Problem Statement and Requirements for IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing", RFC 6606, DOI 10.17487/RFC6606, May 2012, <https://www.rfc-editor.org/info/rfc6606>.

[RFC8025] Thubert, P., Ed. and R. Cragie, "IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Paging Dispatch", RFC 8025, DOI 10.17487/RFC8025, November 2016, <https://www.rfc-editor.org/info/rfc8025>.

[RFC8138] Thubert, P., Ed., Bormann, C., Toutain, L., and R. Cragie, "IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing Header", RFC 8138, DOI 10.17487/RFC8138, April 2017, <https://www.rfc-editor.org/info/rfc8138>.

[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>.

[RFC8930] Watteyne, T., Ed., Thubert, P., Ed., and C. Bormann, "On Forwarding 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Network) Fragments over a Multi-Hop IPv6 Network", RFC 8930, DOI 10.17487/RFC8930, November 2020, <https://www.rfc-editor.org/info/rfc8930>.

10.2. Informative References

[6TiSCH] Thubert, P., "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4", Work in Progress, Internet-Draft, draft-ietf-6tisch-architecture-29, 27 August 2020, <https://tools.ietf.org/html/draft-ietf-6tisch-architecture-29>.

[IEEE.802.15.4] IEEE, "IEEE Standard for Low-Rate Wireless Networks", IEEE Standard 802.15.4-2015, DOI 10.1109/IEEESTD.2016.7460875, April 2016, <http://ieeexplore.ieee.org/document/7460875/>.

[Kent] Kent, C. and J. Mogul, "Fragmentation Considered Harmful", SIGCOMM '87: Proceedings of the ACM workshop on Frontiers in computer communications technology, pp. 390-401, DOI 10.1145/55483.55524, August 1987, <http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-87-3.pdf>.

[LWIG-FRAG] Bormann, C. and T. Watteyne, "Virtual reassembly buffers in 6LoWPAN", Work in Progress, Internet-Draft, draft-ietf-lwig-6lowpan-virtual-reassembly-02, 9 March 2020, <https://tools.ietf.org/html/draft-ietf-lwig-6lowpan-virtual-reassembly-02>.

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, DOI 10.17487/RFC2914, September 2000, <https://www.rfc-editor.org/info/rfc2914>.

[RFC3031] Rosen, E., Viswanathan, A., and R. Callon, "Multiprotocol Label Switching Architecture", RFC 3031, DOI 10.17487/RFC3031, January 2001, <https://www.rfc-editor.org/info/rfc3031>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, DOI 10.17487/RFC4963, July 2007, <https://www.rfc-editor.org/info/rfc4963>.

[RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, DOI 10.17487/RFC5033, August 2007, <https://www.rfc-editor.org/info/rfc5033>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <https://www.rfc-editor.org/info/rfc6550>.

[RFC6552] Thubert, P., Ed., "Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL)", RFC 6552, DOI 10.17487/RFC6552, March 2012, <https://www.rfc-editor.org/info/rfc6552>.

[RFC6554] Hui, J., Vasseur, JP., Culler, D., and V. Manral, "An IPv6 Routing Header for Source Routes with the Routing Protocol for Low-Power and Lossy Networks (RPL)", RFC 6554, DOI 10.17487/RFC6554, March 2012, <https://www.rfc-editor.org/info/rfc6554>.

[RFC7554] Watteyne, T., Ed., Palattella, M., and L. Grieco, "Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT): Problem Statement", RFC 7554, DOI 10.17487/RFC7554, May 2015, <https://www.rfc-editor.org/info/rfc7554>.

[RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <https://www.rfc-editor.org/info/rfc7567>.

[RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/info/rfc8085>.

[RFC8087] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC 8087, DOI 10.17487/RFC8087, March 2017, <https://www.rfc-editor.org/info/rfc8087>.

[RFC8201] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, RFC 8201, DOI 10.17487/RFC8201, July 2017, <https://www.rfc-editor.org/info/rfc8201>.

[RFC8900] Bonica, R., Baker, F., Huston, G., Hinden, R., Troan, O., and F. Gont, "IP Fragmentation Considered Fragile", BCP 230, RFC 8900, DOI 10.17487/RFC8900, September 2020, <https://www.rfc-editor.org/info/rfc8900>.

Appendix A. Rationale

There are a number of uses for large packets in Wireless Sensor Networks. Such usages may not be the most typical or represent the largest amount of traffic over the LLN; however, the associated functionality can be critical enough to justify extra care for ensuring effective transport of large packets across the LLN.

The list of those usages includes:

Towards the LLN node:

Firmware update: For example, a new version of the LLN node software is downloaded from a system manager over unicast or multicast services. Such a reflashing operation typically involves updating a large number of similar LLN nodes over a relatively short period of time.

Packages of commands: A number of commands or a full configuration can be packaged as a single message to ensure consistency and enable atomic execution or complete rollback. Until such commands are fully received and interpreted, the intended operation will not take effect.

From the LLN node:

Waveform captures: A number of consecutive samples are measured at a high rate for a short time and then are transferred from a sensor to a gateway or an edge server as a single large report.

Data logs: LLN nodes may generate large logs of sampled data for later extraction. LLN nodes may also generate system logs to assist in diagnosing problems on the node or network.

Large data packets: Rich data types might require more than one fragment.

Uncontrolled firmware download or waveform upload can easily result in a massive increase of the traffic and saturate the network.

When a fragment is lost in transmission, the lack of recovery in the original fragmentation system of RFC 4944 implies that all fragments would need to be resent, further contributing to the congestion that caused the initial loss and potentially leading to congestion collapse.

This saturation may lead to excessive radio interference or random early discard (leaky bucket) in relaying nodes. Additional queuing and memory congestion may result while waiting for a low-power next hop to emerge from its sleep state.

Considering that RFC 4944 defines an MTU as 1280 bytes, and that in most incarnations (except 802.15.4g) an IEEE Std 802.15.4 frame can limit the link-layer payload to as few as 74 bytes, a packet might be fragmented into at least 18 fragments at the 6LoWPAN shim layer. Taking into account the worst-case header overhead for 6LoWPAN Fragmentation and Mesh Addressing headers will increase the number of required fragments to around 32. This level of fragmentation is much higher than that traditionally experienced over the Internet with IPv4 fragments. At the same time, the use of radios increases the probability of transmission loss, and mesh-under techniques compound that risk over multiple hops.

Mechanisms such as TCP or application-layer segmentation could be used to support end-to-end reliable transport. One option to support bulk data transfer over a frame-size-constrained LLN is to set the Maximum Segment Size to fit within the link maximum frame size. However, doing so can add significant header overhead to each 802.15.4 frame and cause extraneous acknowledgments across the LLN compared to the method in this specification.

Appendix B. Requirements

For one-hop communications, a number of LLN link layers propose a local acknowledgment mechanism that is enough to detect and recover the loss of fragments. In a multi-hop environment, an end-to-end fragment recovery mechanism might be a good complement to a hop-by-hop Medium Access Control (MAC) recovery. This document introduces a simple protocol to recover individual fragments between 6LFF endpoints that may be multiple hops away.

The method addresses the following requirements of an LLN:

Number of fragments: The recovery mechanism must support highly fragmented packets, with a maximum of 32 fragments per packet.

Minimum acknowledgment overhead: Because the radio is half duplex, and because of silent time spent in the various medium access mechanisms, an acknowledgment consumes roughly as many resources as a data fragment.

The new end-to-end fragment recovery mechanism should be able to acknowledge multiple fragments in a single message and not require an acknowledgment at all if fragments are already protected at a lower layer.

Controlled latency: The recovery mechanism must succeed or give up within the time boundary imposed by the recovery process of the upper-layer protocols.

Optional congestion control: The aggregation of multiple concurrent flows may lead to the saturation of the radio network and congestion collapse.

The recovery mechanism should provide means for controlling the number of fragments in transit over the LLN.

Appendix C. Considerations on Congestion Control

Considering that a multi-hop LLN can be a very sensitive environment due to the limited queuing capabilities of a large population of its nodes, this document recommends a simple and conservative approach to congestion control, based on TCP congestion avoidance.

Congestion on the forward path is assumed in case of packet loss, and packet loss is assumed upon timeout. This document allows controlling the number of outstanding fragments that have been transmitted, but for which an acknowledgment was not yet received, and that are still covered by the ARQ timer.

Congestion on the forward path can also be indicated by an ECN mechanism. Though whether and how ECN [RFC3168] is carried out over the LoWPAN is out of scope, this document provides a way for the destination endpoint to echo an ECN indication back to the fragmenting endpoint in an acknowledgment message as represented in Figure 4 in Section 5.2.

While the support of echoing the ECN at the reassembling endpoint is mandatory, this specification only provides a minimalistic behavior on the fragmenting endpoint. If an "E" flag is received, the window **SHOULD** be reduced at least by 1 and at max to 1. Halving the window for each "E" flag received could be a good compromise, but it needs further experimentation. A very simple implementation may just reset the window to 1, so the fragments are sent and acknowledged one by one.

Note that any action that has been performed upon detection of congestion only applies for the transmission of one datagram, and the next datagram starts with the configured Window_Size again.

The exact use of the Acknowledgment Request flag and of the window are left to implementation. An optimistic implementation could send all the fragments up to Window_Size, setting the Acknowledgment Request "X" flag only on the last fragment; wait for the bitmap, which means a gap of half a round-trip time; and resend the losses. A pessimistic implementation could set the "X" flag on the first fragment to check that the path works and open the window only upon receiving the RFRAG-ACK. It could then set an "X" flag again on the second fragment and use the window as a credit to send up to Window_Size before it is blocked. In that case, if the RFRAG-ACK comes back before the window starves, the gating factor is the inter-frame gap. If the RFRAG-ACK does not arrive in time, the Window_Size is the gating factor, and the transmission of the datagram is delayed.

It must be noted that even though the inter-frame gap can be used as a flow control or a congestion control measure, it also plays a critical role in wireless collision avoidance. In particular, when a mesh operates on the same channel over multiple hops, the forwarding of a fragment over a certain hop may collide with the forwarding of the next fragment that is following over a previous hop but that is in the same interference domain. To prevent this, the fragmenting endpoint is required to pace individual fragments within a transmit window with an inter-frame gap. This is needed to ensure that a given fragment is sent only when the previous fragment has had a chance to progress beyond the interference domain of this hop. In the case of 6TiSCH [6TiSCH], which operates over the Time-Slotted Channel Hopping (TSCH) mode of operation of IEEE 802.15.4 [RFC7554], a fragment is forwarded over a different channel at a different time, and it makes full sense to transmit the next fragment as soon as the previous fragment has had its chance to be forwarded at the next hop.

Depending on the setting of the Window_Size and the inter-frame gap, how the window is used, and the number of hops, the Window_Size may or may not become the gating factor that blocks the transmission. If the sender uses the Window_Size as a credit:

- a conservative Window_Size of, say, 3 will be the gating factor that limits the transmission rate of the sender -- and causes transmission gaps longer than the inter-frame gap -- as soon as the number of hops exceeds 3 in a TSCH network and 5-9 in a single frequency mesh. The more hops the more the starving window will add to latency of the transmission.
- The recommendation to align the Window-Size to the round-trip time divided by the time per fragment aligns the Window-Size to the time it takes to get the RFAG_ACK before the window starves. A Window-Size that is higher than that increases the chances of a congestion but does not improve the forward throughput. Considering that the RFRAG-ACK takes the same path as the fragment with the assumption that it travels at roughly the same speed, an inter-frame gap that separates fragments by 2 hops leads to a Window_Size that is roughly the number of hops.
- Setting the Window-Size to 32 minimizes the cost of the acknowledgment in a constrained network and frees bandwidth for the fragments in a half-duplex network. Using it increases the risk of congestion if a bottleneck forms, but it optimizes the use of resources under normal conditions. When it is used, the only protection for the network is the inter-frame gap, which must be chosen wisely to prevent the formation of a bottleneck.

From the standpoint of a source 6LoWPAN endpoint, an outstanding fragment is a fragment that was sent but for which no explicit acknowledgment was yet received. This means that the fragment might be on the path or received but not yet acknowledged, or the acknowledgment might be on the path back. It is also possible that either the fragment or the acknowledgment was lost on the way.

From the fragmenting endpoint standpoint, all outstanding fragments might still be in the network and contribute to its congestion. There is an assumption, though, that after a certain amount of time, a frame is either received or lost, so it is not causing congestion anymore. This amount of time can be estimated based on the round-trip time between the 6LoWPAN endpoints. For the lack of a more adapted technique, the method detailed in "Computing TCP's Retransmission Timer" [RFC6298] may be used for that computation.

This specification provides the necessary tools for the fragmenting endpoint to take congestion control actions and protect the network, but it leaves the implementation free to select the action to be taken. The intention is to use it to build experience and specify more precisely the congestion control actions in one or more future specifications. "Congestion Control Principles" [RFC2914] and "Specifying New Congestion Control Algorithms" [RFC5033] provide indications and wisdom that should help through this process.

[RFC7567] and [RFC5681] provide deeper information on why congestion control is needed and how TCP handles it. Basically, the goal here is to manage the number of fragments present in the network; this is achieved by reducing the number of outstanding fragments over a congested path by throttling the sources.

Acknowledgments

The author wishes to thank Michel Veillette, Dario Tedeschi, Laurent Toutain, Carles Gomez Montenegro, Thomas Watteyne, and Michael Richardson for their in-depth reviews and comments. Also, many thanks to Roman Danyliw, Peter Yee, Colin Perkins, Tirumaleswar Reddy.K, Éric Vyncke, Warren Kumari, Magnus Westerlund, Erik Nordmark, and especially Benjamin Kaduk and Mirja Kühlewind for their careful reviews and help during the IETF Last Call and IESG review process. Thanks to Jonathan Hui, Jay Werb, Christos Polyzois, Soumitri Kolavennu, Pat Kinney, Margaret Wasserman, Richard Kelsey, Carsten Bormann, and Harry Courtice for their various contributions in the long process that lead to this document.

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