LLN Minimal Fragment Forwarding
draft-watteyne-6lo-minimal-fragment-02

Abstract

This document gives an overview of LLN Minimal Fragment Forwarding. When employing adaptation layer fragmentation in 6LoWPAN, it may be beneficial for a forwarder not to have to reassemble each packet in its entirety before forwarding it. This has been always possible with the original fragmentation design of RFC4944. This document is a companion document to [I-D.ietf-lwig-6lowpan-virtual-reassembly], which details the virtual Reassembly Buffer (VRB) implementation technique which reduces the latency and increases end-to-end reliability in route-over forwarding.

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 17, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents
1. Overview of 6LoWPAN Fragmentation

6LoWPAN fragmentation is defined in [RFC4944]. Although [RFC6282] updates [RFC4944], it does not redefine 6LoWPAN fragmentation.

We use Figure 1 to illustrate 6LoWPAN fragmentation. We assume node A forwards a packet to node B, possibly as part of a multi-hop route between IPv6 source and destination nodes which are neither A nor B.

```
+-----+                     +-----+
... ---| A |------------------->| B |--- ...
+-----+                     +-----+
# (frag. 5)
```

```
| 123456789 | 123456789 |
| ----------+----------|
| #       # | #       # |
```

outgoing    incoming
fragmentation reassembly
buffer      buffer

Figure 1: Fragmentation at node A, reassembly at node B.

Node A starts by compacting the IPv6 packet using header compression defined in [RFC6282]. If the resulting 6LoWPAN packet does not fit into a single link-layer frame, node A’s 6LoWPAN sublayer cuts it...
into multiple 6LoWPAN fragments, which it transmits as separate link-
layer frames to node B. Node B’s 6LoWPAN sublayer reassembles these
fragments, inflates the compressed header fields back to the original
IPv6 header, and hands over the full IPv6 packet to its IPv6 layer.

In Figure 1, a packet forwarded by node A to node B is cut into nine
fragments, numbered 1 to 9. Each fragment is represented by the ‘#’
symbol. Node A has sent fragments 1, 2, 3, 5, 6 to node B. Node B
has received fragments 1, 2, 3, 6 from node A. Fragment 5 is still
being transmitted at the link layer from node A to node B.

A reassembly buffer for 6LoWPAN contains:

- datagram_size,
- datagram_tag and link-layer sender and receiver addresses (to
  which the datagram_tag is local),
- 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,
- a timer that allows discarding the partial packet after a timeout.

A fragmentation header is added to each fragment; it indicates what
portion of the packet that fragment corresponds to. Section 5.3 of
[RFC4944] defines the format of the header for the first and
subsequent fragments. All fragments are tagged with a 16-bit
"datagram_tag", used to identify which packet each fragment belongs
to. Each fragment can be uniquely identified by the source and
destination link-layer addresses of the frame that carries it, and
the datagram_tag. The value of the datagram_tag only needs to be
locally unique to nodes A and B.

Node B’s typical behavior, per [RFC4944], is as follows. Upon
receiving a fragment from node A with a datagram_tag previously
unseen from node A, node B allocates a buffer large enough to hold
the entire packet. The length of the packet is indicated in each
fragment (the datagram_size field), so node B can allocate the buffer
even if the first fragment it receives is not fragment 1. As
fragments come in, node B fills the buffer. When all fragments have
been received, node B inflates the compressed header fields into an
IPv6 header, and hands the resulting IPv6 packet to the IPv6 layer.

This behavior typically results in per-hop fragmentation and
reassembly. That is, the packet is fully reassembled, then
(re)fragmented, at every hop.
2. Limits of Per-Hop Fragmentation and Reassembly

There are at least 2 limits to doing per-hop fragmentation and reassembly:

2.1. Latency

When reassembling, a node needs to wait for all the fragments to be received before being able to generate the IPv6 packet, and possibly forward it to the next hop. This repeats at every hop.

This may result in increased end-to-end latency compared to the case where each fragment would be forwarded without per-hop reassembly.

2.2. Memory Management and Reliability

Constrained nodes have limited memory. Assuming 1 kB reassembly buffers, typical nodes only have enough memory for 1-3 reassembly buffers.

Assuming the topology from Figure 2, where nodes A, B, C and D all send packets through node E. We further assume that node E’s memory can only hold 3 reassembly buffers.

```
+----+       +----+
... --|-- A |------|-- B |
+----+       +----+
     \                  |
      \                  | E |----|-- F | ...
     +----+       +----+
      /                  /
     +----+       +----+
... --|-- C |------|-- D |
+----+       +----+
```

Figure 2: Illustrating the Memory Management Issue.

When nodes A, B and C concurrently send fragmented packets, all 3 reassembly buffers in node E are occupied. If, at that moment, node D also sends a fragmented packet, node E has no option but to drop one of the packets, lowering end-to-end reliability.
3. Virtual Reassembly Buffer (VRB) Implementation

Virtual Reassembly Buffer (VRB) is the implementation technique described in [I-D.ietf-lwig-6lowpan-virtual-reassembly] in which a forwarder does not reassemble each packet in its entirety before forwarding it.

VRB overcomes the limits listed in Section 2. Nodes don't wait for the last fragment before forwarding, reducing end-to-end latency. Similarly, the memory footprint of VRB is just the VRB table, reducing the packet drop probability significantly.

There are, however, limits:

Non-zero Packet Drop Probability: Each VRB table entry can be 12 B (assuming 16-bit link-layer addresses). This is a footprint 2 orders of magnitude smaller compared to needing a 1280-byte reassembly buffer for each packet. Yet, the size of the VRB table necessarily remains finite. In the extreme case where a node is required to concurrently forward more packets that it has entries in its VRB table, packets are dropped.

No Fragment Recovery: There is no mechanism in VRB for the node that reassembles a packet to request a single missing fragment. Dropping a fragment requires the whole packet to be resent. This causes unnecessary traffic, as fragments are forwarded even when the destination node can never construct the original IPv6 packet.

No Per-Fragment Routing: All subsequent fragments follow the same sequence of hops from the source to the destination node as fragment 1.

The severity and occurrence of these limits depends on the link-layer used. Whether these limits are acceptable depends entirely on the requirements the application places on the network.

If the limits are both present and not accepted by the application, future specifications may define new protocols to overcome these limits. One example is [I-D.thubert-6lo-fragment-recovery] which defines a protocol which allows fragment recovery.

4. Security Considerations

An attacker can perform a DoS attack on a node implementing VRB by generating a large number of bogus "fragment 1" fragments without sending subsequent fragments. This causes the VRB table to fill up.

Secure joining and the link-layer security that it sets up protects against those attacks from network outsiders.
5. IANA Considerations

No requests to IANA are made by this document.

6. Acknowledgments

The authors would like to thank Yasuyuki Tanaka for his in-depth review of this document.

7. Informative References


Authors’ Addresses

Thomas Watteyne (editor)
Analog Devices
32990 Alvarado-Niles Road, Suite 910
Union City, CA 94587
USA

Email: thomas.watteyne@analog.com