6lo
Internet-Draft
Intended status: Informational
Expires: March 2, 2020

T. Watteyne, Ed.
Analog Devices
C. Bormann
Universitaet Bremen TZI
P. Thubert
Cisco

August 30, 2019

6LoWPAN Fragment Forwarding

draft-ietf-6lo-minimal-fragment-04

Abstract

This document provides a simple method to forwarding 6LoWPAN fragments. 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 always been possible with the original fragmentation design of RFC4944. This method reduces the latency and increases end-to-end reliability in route-over forwarding. It is the companion to the virtual Reassembly Buffer which is a pure implementation technique.

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 March 2, 2020.

Copyright Notice

Copyright (c) 2019 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 (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Overview of 6LoWPAN Fragmentation

The original 6LoWPAN fragmentation is defined in [RFC4944] and it is implicitly defined for use over a single IP hop though possibly multiple Layer-2 hops in a meshed 6LoWPAN Network. 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 the header compression mechanism 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.

The reassembly buffer for 6LoWPAN is indexed in node B by:

- a unique Identifier of Node A (e.g., Node A's link-layer address)
- the datagram_tag chosen by node A for this fragmented datagram

Because it may be hard for node B to correlate all possible link-layer addresses that node A may use (e.g., short vs. long addresses), node A must use the same link-layer address to send all the fragments of a same datagram to node B.

Conceptually, the reassembly buffer in node B contains, assuming that node B is neither the source nor the final destination:

- a datagram_tag as received in the incoming fragments, associated to link-layer address of node A for which the received datagram_tag is unique,
- the link-layer address that node B uses to forward the fragments
- the link-layer address of the next hop that is resolved on the first fragment
- a datagram_tag that node B uniquely allocated for this datagram and that is used when forwarding the fragments of the datagram
- 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,
- a datagram_size,
- a buffer for the remainder of a previous fragment left to be sent,
- a timer that allows discarding a partially reassembled packet after some 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 datagram can be uniquely identified by the sender link-layer addresses of the frame that carries it and the datagram_tag that the sender allocated for this datagram. Each fragment can be identified within its datagram by the datagram-offset.

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. See [ARTICLE] for detailed simulation results on both limits.

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 a case where each fragment is forwarded without per-hop reassembly.

2.2. Memory Management and Reliability

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

To illustrate this we use 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 do not 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: The abstract data in a VRB table entry contains at a minimum the MAC address of the predecessor and that of the successor, the datagram_tag used by the predecessor and the local datagram_tag that this node will swap with it. The VRB may need to store a few octets from the last fragment that may not have fit within MTU and that will be prepended to the next fragment. This yields a small footprint that is 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 the first fragment, because the IP header is required to route the fragment and is only present in the first fragment. A side effect is that the first fragment must always be forwarded first.

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 present and not acceptable for the application, future specifications may define new protocols to overcome these limits. One example is [I-D.ietf-6lo-fragment-recovery] which defines a protocol which allows fragment recovery.

4. Security Considerations

An attacker can perform a Denial-of-Service (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. Note that the VRB does not need to remember the full datagram as received so far but only possibly a few octets from the last fragment that could not fit in it. It is expected that an implementation protects itself to keep the number of VRBs within capacity, and that old VRBs are protected by a timer of a reasonable duration for the technology and destroyed upon timeout.

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. Also many thanks to Georgies Papadopoulos and Dominique Barthel for their own reviews.

7. Informative References

[ARTICLE] Tanaka, Y., Minet, P., and T. Watteyne, "6LoWPAN Fragment Forwarding", IEEE Communications Standards Magazine , 2019.

[I-D.ietf-6lo-fragment-recovery]
Thubert, P., "6LoWPAN Selective Fragment Recovery", draft-ietf-6lo-fragment-recovery-05 (work in progress), July 2019.

[I-D.ietf-lwig-6lowpan-virtual-reassembly]
Bormann, C. and T. Watteyne, "Virtual reassembly buffers in 6LoWPAN", draft-ietf-lwig-6lowpan-virtual-reassembly-01 (work in progress), March 2019.

[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>.

Authors' Addresses

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

Email: thomas.watteyne@analog.com

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Email: cabo@tzi.org

Pascal Thubert
Cisco Systems, Inc
Building D
45 Allee des Ormes - BP1200
MOUGINS - Sophia Antipolis 06254
France

Email: pthubert@cisco.com