6lo
Internet-Draft
Intended status: Informational
Expires: September 7, 2018

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

March 06, 2018

LLN Minimal Fragment Forwarding
draft-watteyne-6lo-minimal-fragment-01

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 details the Virtual Reassembly Buffer (VRB) implementation technique which reduces the latency and increases end-to-end reliability in route-over forwarding, and discusses its limits.

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 September 7, 2018.

Table of Contents

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.

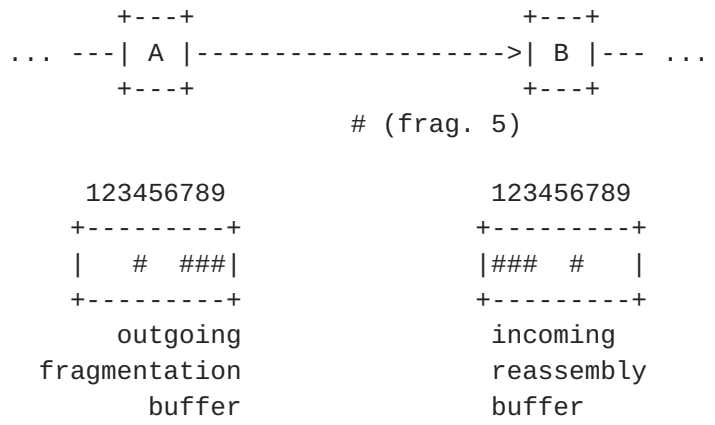

Figure 1: Fragmentation at node A, reassembly at node B.

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.

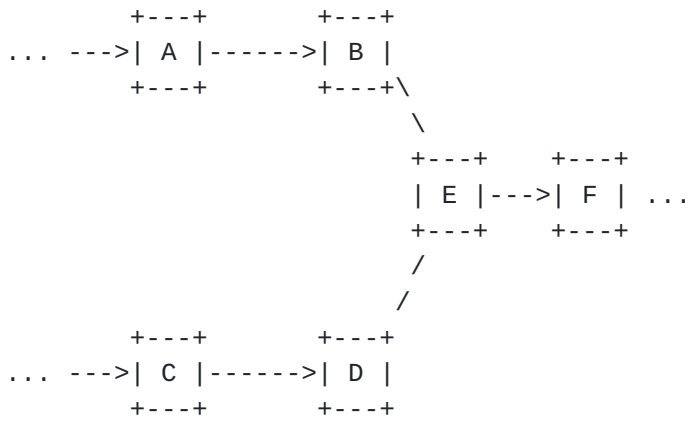

Figure 2: Illustrating the Memory Management Issue.

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

One implementation of 6LoWPAN fragmentation overcomes the limits listed in Section 2. The idea is for a node to immediately retransmit a fragment it receives, without fully reassembling the packet. This idea was introduced in Section 2.5.2 of [BOOK]. That is, a node may attempt to send out the data for a fragment in the form of a forwarded fragment, as soon as all necessary information for that is available.
Obviously, all fragments need to be sent with the same outgoing address (otherwise a full reassembly implementation would discard the fragments) and the same datagram_tag.

We use Figure 3 to illustrate VRB, and focus on the behavior of node E. With VRB, node E maintains a VRB table which functions similarly to a switching table: when receiving a fragment from node B with datagram_tag=2, forward it to node F with datagram_tag=8.

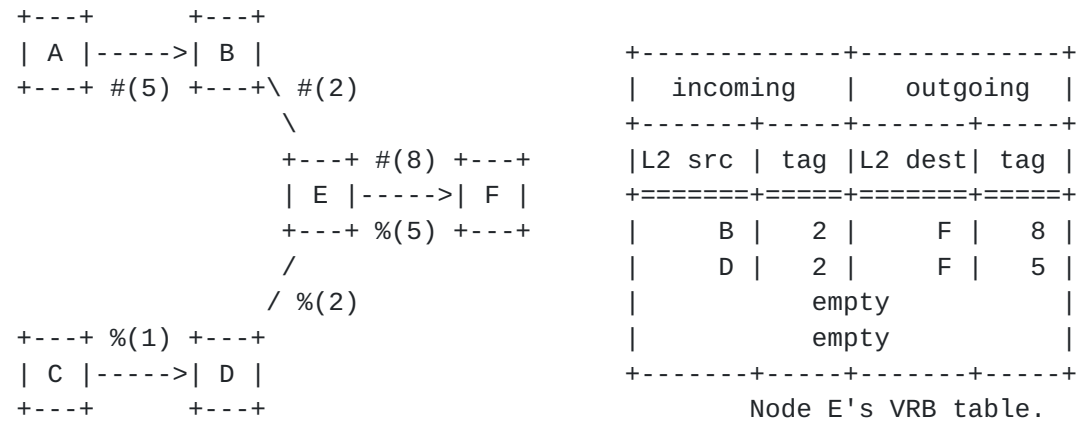

Figure 3: Illustrating VRB. #(5) and %(1) are fragments from packets coming from nodes A and C, with datagram_tag set to 5 and 1, respectively.

The VRB table is initially empty. An implementation might have for example pre-allocate memory for a VRB table with 4 entries (as in Figure 3), initially cleared.

When node E receives fragment 1 from node B with datagram_tag=2, it inspects the contents of the fragment and reads out the destination IPv6 address. When it is not destined to it, node E identifies the next hop to send this fragment to. It then creates an entry in the VRB table which contains 4 fields: (1) the link-layer address of the sender of the fragment it received, (2) the datagram_tag of the fragment it received, (3) the link-layer address of the next hop, (4) a datagram_tag for the fragments it will send. The latter datagram_tag must be locally unique.

Note that, if node E had multiple interfaces, the VRB table would also need additional column to identify the incoming and outgoing interface.

Any subsequent fragment that matches the "incoming" columns in the node's VRB table are immediately forwarded using the information in the "outgoing" columns. Note that, while this results in a behavior similar to link-layer switching, what is really happening is that the node has a virtual reassembly buffer. That is, it operates as if the packet were reassembled and fragmented, without ever actually holding a fully reassembled packet in memory.

Upon forwarding the last fragment of a packet, the VRB table entry can be cleared, and reused for a future packet. If the last fragment of a packet is dropped, the VRB table entry can be invalidated by timeout. Its timeout value is set to a maximum of 60 seconds as the reassembly timeout defined in [RFC4944].

A simple implementation may do away with any attempt to keep packet data in the virtual reassembly buffer. It then has to discard all non-first fragments for which a reassembly buffer is not already available (penalizing reordering, which however may be rare).

In case fragments can come out of order (a rare case, as all fragments of a packet are sent between the same neighbors), an implementation can use multiple the following two techniques. In case fragment 1 isn't received first, it can temporarily buffer fragments 2, 3, etc., until fragment 1 is received, and a next hop neighbor can be identified. Similarly, as the final fragment of the packet isn't necessarily received last, an implementation can maintain a bitmap of already forwarded fragments to know when all fragments have been forwarded (and the corresponding VRB entry can be cleared).

Note that the decision to do local processing of a packet needs to be taken with the first fragment - such packets of course do need to be fully reassembled (unless transport and application also can cope with fragments, which they rarely can in the presence of security).

It is possible for a network to be composed of some nodes that implement VRB, and others that don't. Nodes that do not implement VRB reassemble the packet.

[RFC6282] defines the header compression format for 6LoWPAN. One important impact of header compression is that the header is no longer of a fixed length. In particular, changes made by a forwarder may gain or lose the ability to use a more highly compressed variant, changing the length of the header in the packet.

If the change increases the size, the maximum frame size may be exceeded, leading to the need to re-fragment in the forwarder. This is less of a problem with full reassembly, but with virtual reassembly can lead to the need for sending an additional frame for each packet.

The well-known approach to minimize the probability of this need is for the original sender to put all slack in the frame sizes into the _first_ packet, making this the smallest fragment and not the last one as would be done in a naive implementation. (This also has other consequences related to delivery probability, which are not discussed here.) This makes sure an additional fragment only needs to be sent if the header expansion during forwarding would have created an additional fragment with full reassembly as well.

4. Critique of VRB

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.

5. 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.

6. IANA Considerations

No requests to IANA are made by this document.

7. Acknowledgments

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

8. Informative References

[BOOK] Shelby, Z. and C. Bormann, "6LoWPAN", John Wiley & Sons, Ltd monograph, DOI 10.1002/9780470686218, November 2009.

[I-D.thubert-6lo-fragment-recovery] Thubert, P., "6LoWPAN Selective Fragment Recovery", draft-thubert-6lo-fragment-recovery-00 (work in progress), February 2018.

[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