TFS Framing Decapsulation Optimization

G. Paul Ziemba and Christian Hopps
Labn Consulting, LLC

September 21, 2020

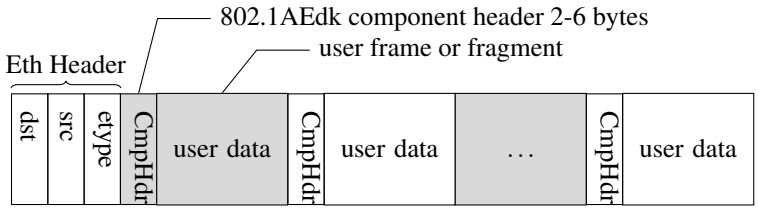
1 Overview

Current proposals for Ethernet [etf20] and IP Traffic Flow Security [Hop20] describe methods for tunneling user traffic (ethernet or IP packets) within TFS tunnel PDUs. These proposals allow user packets to be fragmented and/or aggregated within tunnel PDUs, and specify headers at the start of each user packet component within the tunnel PDU. A TFS receiver parses tunnel PDUs to recover the original user packets, splitting on component boundaries and reassembling fragments as needed. These proposals specify in-line frame headers, which requires that receivers read packet memory at each frame boundary in order to reconstitute tunneled user packets.

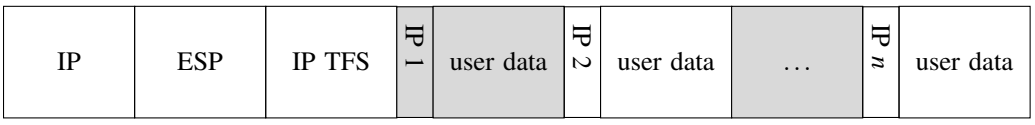
We consider modifying the protocol formats to collect framing headers into a single block in order to reduce memory I/O at the receiver and allow for increased performance. Based on our assumptions about user packet size statistics, the case for making this modification does not seem compelling.

2 Current Inline Framing

Ethernet TFS encapsulation uses the following format:



IP TFS encapsulation uses the following format:



In general, user packets are encapsulated in TFS tunnel PDUs in the order they were received. User packets may be fragmented across multiple TFS tunnel PDUs as needed to fill tunnel PDUs to a configured maximum size.

The Component Header or internal IP header includes a length field indicating the framing boundary and start of the next component. A receiver must parse each Component Header or internal IP header to divide the tunnel PDU into its constituent fragments/user packets for further processing.

3 Performance Drawbacks

Software-based data plane implementations read the first part of a received packet (*i.e.*, the header) to determine how to handle it, but generally do not read the rest of the packet data. Thus access to approximately one data cache-

line's worth of packet header data is a fixed minimum cost. Subsequent random-access reads incur additional latency penalties. Typical cache line sizes are 64 or 128 bytes (processor-specific).

The first-order problem we seek to address is the penalty due to gathering the framing information in the tunnel packet. The current TFS encapsulation formats distribute framing information across the tunnel packet data area; computing the locations of each frame boundary requires a probable data cache miss for each boundary.

A second-order problem is the alignment of the framing headers with respect to the data cache-line boundaries. If a header straddles a boundary, two cache misses occur before the header can be fully parsed.

4 Proposed Framing Trailer

We proposed to collect the component headers into a single region of the tunnel PDU, and thus reduce the processing latency for the tunnel receiver. This approach addresses the first-order problem described above by gathering the framing information into a single memory region.

Early discussions considered placing an aggregated offset array at the head of the tunnel packet, but this approach had several drawbacks:

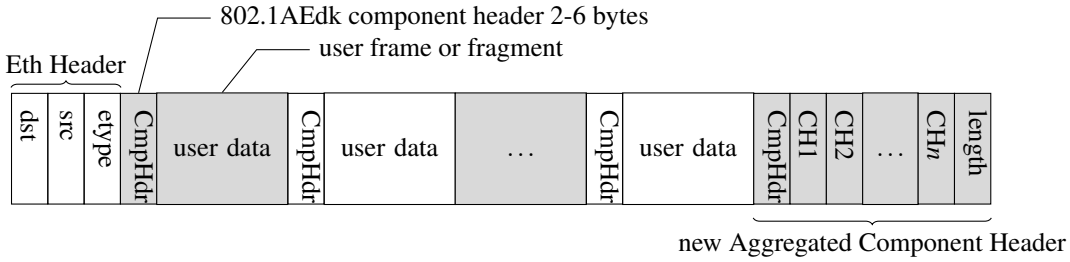
1. When constructing the tunnel PDU, an encapsulator does not necessarily know the makeup of all components when it places the first component, so the size of the complete set of component headers is not known at this time.

   Thus, either a speculative fixed-size header space must be set aside at the head of the packet, or the header (and its preceding headers) must be added after the payload is constructed.

   A speculative fixed-size header leads to less-efficient filling of tunnel packets. Adding a variable-length header after the payload is built has its own challenges, including alignment issues and cache inefficiency.

2. The 802.1 AEdk specification includes support for late insertion of a component into a tunnel PDU after the initial part of the PDU has been transmitted. A combined component header at the start of the PDU would be incompatible with this feature.

However, placing an aggregated component header at the end of the tunnel PDU addresses these issues.



The final field of the proposed Aggregated Component Header would indicate the trailer length to enable the receiver to find the start of the trailer.

It would be possible for the proposed aggregate component header to coexist with the previously-defined distributed component headers. In this case, receivers would be free to treat it as an optional optimization.

5 Cache Alignment

The second-order problem noted in Section 3, cache-line alignment, is more challenging. Tunnel PDUs are subject to various perturbations between the encapsulator and receiver such as label/tag insertions which could affect the memory alignment of the received packet. This problem is subject to many unknowns and it is not clear that a solution is possible.

The effect of cache alignment on an aggregated component header is +/-1 cache miss. By itself, such a small advantage might not be worth trying to address.

It may also be advantageous to align the start of each encapsulated user packet to optimize processing of decapsulation results. In the case of IP TFS, knowledge of the encapsulated packet type (IPv4 vs. IPv6) could come into play.

6 Expected Performance Benefit

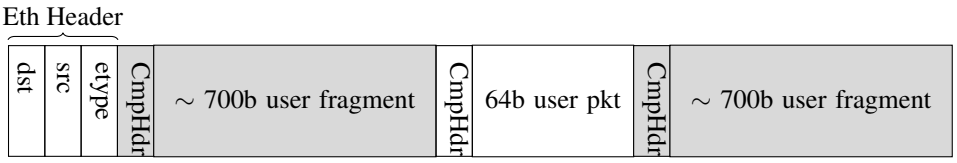
Internet packet size distribution today seems bimodal with approximately 40% 64-byte packets and 40% 1500-byte packets [EEM19].

To simplify analysis, we assume 50% 64-byte packets and 50% 1500-byte packets. We also consider scenarios involving 9000-byte packets as jumbo packets are becoming more common.

For the proposed trailer variations, we consider both non-cache-aligned (2 cache misses) and cache-aligned (1 cache miss) trailer cases.

6.1 1500-in-1500

A typical 1500-byte tunnel PDU carrying 1500-byte user packets is shown below. Statistically, we expect the tunnel to carry almost but not quite one 64-byte user packet per tunnel packet. We further expect almost but not quite one 1500-byte user packet per tunnel packet, and due to TFS opportunistic packing, we expect almost all tunnel packets to carry these 1500 byte user packets in the form of two fragments.

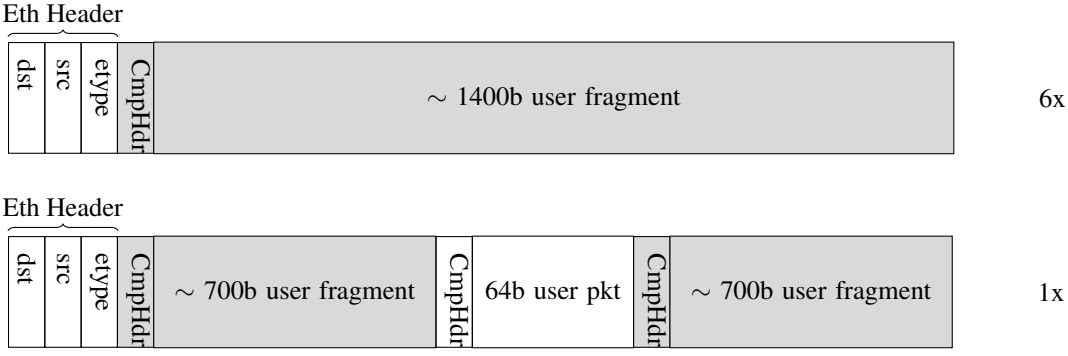


With distributed component headers, we expect that the first header would already be in the data cache due to adjacency to the ethernet header, but two additional cache-miss reads would be needed to access the remaining component headers, or a rate of $\frac{2}{1500} = 1.33$ cache-miss reads per kByte.

Using the proposed trailer, the non-cache-aligned case has the same rate, whereas the cache-aligned case has a rate of $\frac{1}{1500} = .67$ cache-miss reads per kByte.

6.2 9000-in-1500

In networks carrying 9000-byte user packets in a 1500-byte tunnel, we expect a distribution of roughly six tunnel packets carrying a single large user fragment for every tunnel packet carrying two large user fragments and a 64-byte user packet.

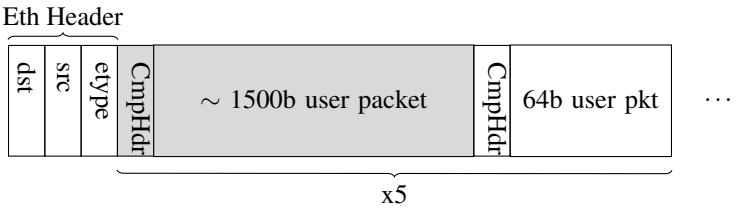


The upper tunnel PDU should not requre any cache-miss reads to access component headers, and the lower PDU should require two, resulting in an overall rate of

$\frac{(6\times0)+(1\times2)}{6+1} = .286$ cache-miss reads/tunnel PDU
or a rate of $\frac{.286}{1500} = .19$ cache-miss reads per kByte.
Using the proposed trailer, the non-cache-aligned case has the same rate, whereas the cache-aligned case has
$\frac{(6\times0)+(1\times1)}{6+1} = .143$ cache-miss reads/tunnel PDU
or a rate of $\frac{.143}{1500} = .1$ per kByte.

6.3 1500-in-9000

In networks carrying 1500-byte user packets in a 9000-byte tunnel, we expect a distribution of roughly six tunnel packets carrying a single large user fragment for every tunnel packet carrying two large user fragments and a 64-byte user packet.



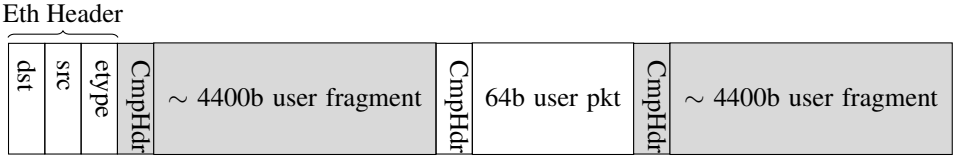
In this scenario, we expect about nine cache-miss reads per tunnel PDU or a rate of $\frac{1}{1000} = 1$ per kByte.

Using the proposed trailer, the non-cache-aligned case would have two cache-miss reads per tunnel PDU or a rate of $\frac{2}{9000} = .22$ per kByte.

The proposed trailer in the cache-aligned case would have one cache-miss read per tunnel PDU or a rate of .11 per kByte.

6.4 9000-in-9000

In networks carrying 9000-byte user packets in a 9000-byte tunnel, we expect the tunnel to carry almost but not quite one 64-byte user packet per tunnel packet. We further expect almost but not quite one 9000-byte user packet per tunnel packet, and due to TFS opportunistic packing, we expect almost all tunnel packets to carry these 9000 byte user packets in the form of two fragments.



With distributed component headers, we expect that the first header would already be in the data cache due to adjacency to the ethernet header, but two additional cache-miss reads would be needed to access the remaining component headers, or a rate of $\frac{2}{9000} = .22$ cache-miss reads per kByte.

Using the proposed trailer, the non-cache-aligned case has the same rate, whereas the cache-aligned case has a rate of $\frac{1}{9000} = .11$ cache-miss reads per kByte.

6.5 Comparison Summary

The cache-miss rate computations (values in misses per kByte) above are summarized here:

MTU scenario	Distributed	Aggregated	Aggregated and Aligned
1500-in-1500	1.33	1.33	.67
9000-in-1500	.19	.19	.10
1500-in-9000	1.00	.22	.11
9000-in-9000	.22	.22	.11

7 Conclusion

Given the statistical assumptions above, the benefits do not seem compelling. The proposed trailer would have greater advantage for user traffic mixes with many small packets.

References

[EEM19] Espinal, Albert, Estrada, Rebeca, and Monsalve, Carlos. Traffic model using a novel sniffer that ensures the user data privacy. *MATEC Web Conf.*, 292:03002, 2019.

[etf20] IEEE Draft Standard for Local and Metropolitan area networks – Media Access Control (MAC) Security, Amendment 4: MAC Privacy Protection. *Amendment to IEEE Std 802.1AE: 802-1AEdk-d0-2*, pages 134–140, 2020.

[Hop20] C. Hopps. IP Traffic Flow Security. (draft-ietf-ipsecme-iptfs-01), March 2020.