INTERNET-DRAFT
Intended Status: Standard
Expires: January 2020

T. Herbert
Intel

July 8, 2019

UDP Surplus Header
draft-herbert-udp-space-hdr-01

Abstract

This specification defines the UDP Surplus Header that is an extensible and generic format applied to the UDP surplus space. The UDP surplus space comprises the bytes between the end of the UDP datagram, as indicated by the UDP Length field, and the end of the IP packet, as indicated by IP packet or payload length. The UDP Surplus Header can be either a protocol trailer of the UDP datagram, or a protocol header which effectively serves as an extended UDP header.

Table of Contents

1 Introduction

As defined in [RFC768], the UDP header contains a UDP Length field. The UDP Length is not required to correlate with the IP payload length of a packet such that there may be bytes between the end of the UDP datagram and the end of the IP packet. This space is referred to as the UDP surplus space.

This specification defines the UDP Surplus Header (USH) to provide a common format for the UDP surplus space. The USH is comprised of a four byte base header and some variable amount of data. The base header contains a type field that determines how the header data is interpreted. This allows different formats and uses of the UDP surplus space. UDP options [UDPOPT] are one example of a type where the header data contains a list of options.

There are two use cases of USH:

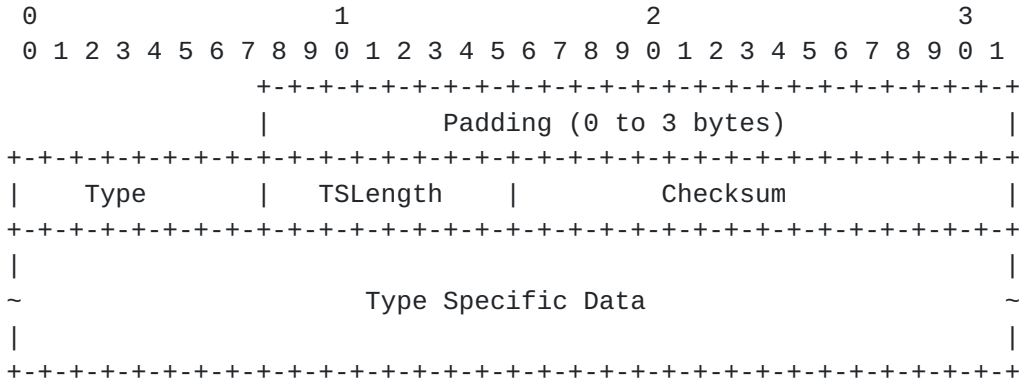
1) Protocol trailer (section 2.1)

2) Protocol header or Extended UDP Header (section 2.2)

The motivations for USH, include the motivations for protocol header format in USH, are described in section 4.

2 UDP Surplus Header format

The common format of the UDP Surplus Header (USH) is shown below:



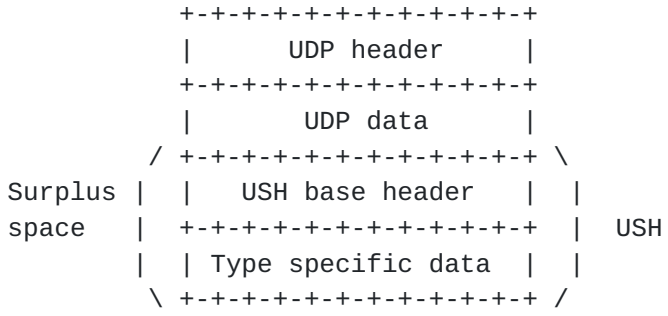
The fields are:

- Padding: Aligns the UDP Surplus Header to four bytes. The number of padding bytes required is: 3 - ((udp_length - 1) % 4), where the udp_length is the length of the UDP datagram as specified in the UDP Length field. Padding bytes MUST be set to zero on transmission, and MUST be verified to be zero when received.

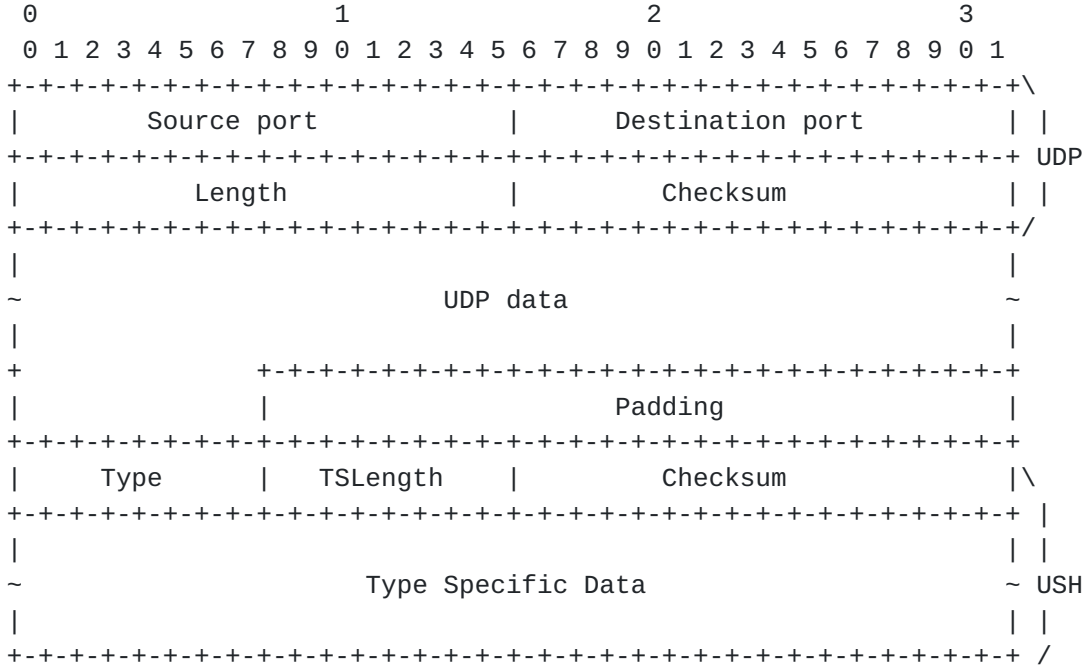
- Type: Indicates the format of the UDP surplus space and how the Type Specific Data is interpreted. Defined Type values are:
  - 0: Reserved
  - 1: UDP options
  - 2-127: Reserved
  - 128-255: Available for private use or experimentation
- TSLength: Length of the type specific data in units of four byte words. The length of the type specific data is thus zero to 1020 bytes.
- Checksum: The standard one's complement checksum that covers the UDP surplus area. The coverage starts from the first byte of Padding, or the Type field if no padding is present, through the end of the IP packet. If the number of Padding bytes is odd then a zero byte is logically prepended to surplus area for the checksum calculation.
- Type Specific Data: Variable length data that is considered part of the UDP Surplus Header. This data is interpreted per the value of the Type field.

2.1 Protocol trailer format

When used as a protocol trailer, the UDP Surplus Header immediately follows the UDP data. The logical protocol layering is:



The packet format of UDP Surplus Header as a protocol trailer is:



Notes:

- The offset of the UDP Surplus Header from the start of the UDP header, including possible padding for the USH, is equal the UDP Length.

- The number of padding bytes is 3 - ((udp_length - 1) % 4), where udp_length is equal to the UDP Length field. The offset of the Type field of the USH is 4 * ((udp_length - 1) / 4 + 1).

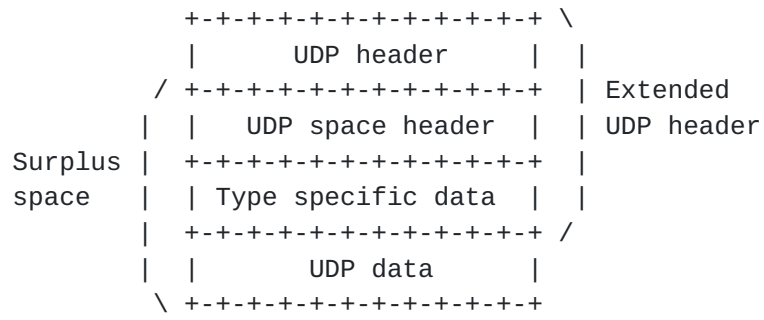
- If the size of the USH header (four plus four times TSLength) is less than the size of the UDP surplus space in a packet, then the USH is considered to be malformed (see section 3.2).

- The UDP checksum covers the UDP header and UDP data. The USH checksum covers the entire UDP surplus space.

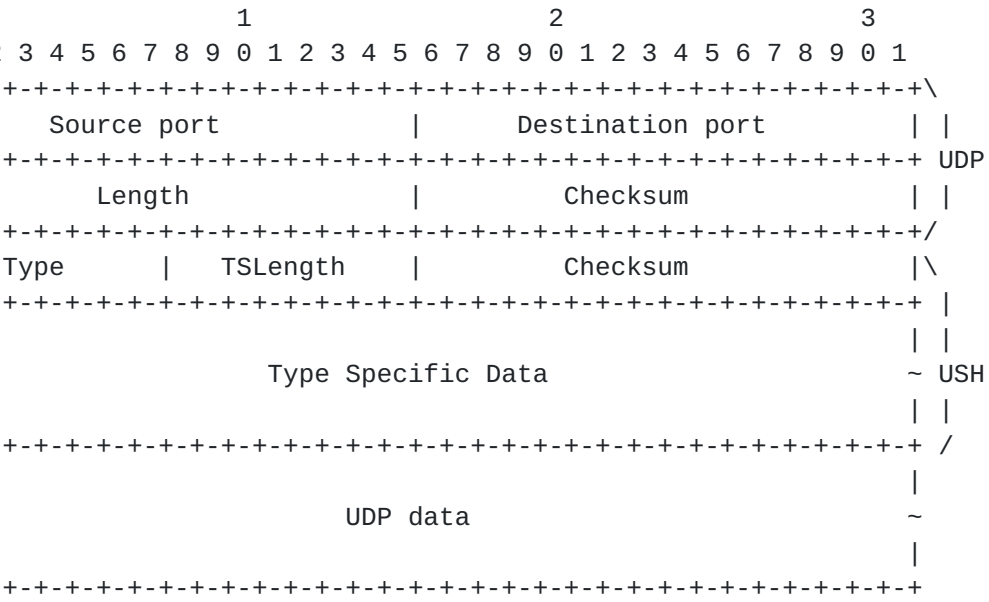
- A legacy receiver, one that does not understand the UDP Surplus Header, will ignore the contents of the UDP surplus space and process the UDP data as normal. Protocol data that cannot correctly be ignored by a receiver, such as the fragmentation option in the [UDPOPT], MUST NOT be in a surplus space trailer.

2.2 Protocol header format (Extended UDP header)

The UDP Surplus Header can be used as a protocol header. Effectively, this creates an extended UDP header format. The logical protocol layering is:



The packet format containing an extended UDP header is:



Notes:

- Since the UDP header is aligned and a multiple of four bytes, no padding for USH is necessary.

- The UDP length is fixed to be eight so that all bytes beyond the UDP header are contained in the surplus space.

- The UDP checksum covers the eight bytes of UDP header and the checksum pseudo header. The USH checksum covers the entire surplus space which includes the UDP Surplus Header and UDP data.

- The UDP data length is the IP payload length minus the size of the UDP header and the size of the UDP Surplus Header. That is:

  UDP_data_length = IP_payload_length - 12 - (4 * TSLength)

- If a legacy receiver, one that does not understand the UDP Surplus Header, receives a packet in protocol header format it will process it as a UDP datagram containing zero length data. Presumably, most applications will ignore such packets, however if an application applies semantics to zero length datagrams then a sender MUST NOT send packets with an extended UDP header to legacy receivers.

3 Operation

3.1 Sender operation

A sender sets a UDP Surplus Header in the surplus space when sending an IP packet. The UDP surplus header immediately follows the UDP packet at the offset of UDP Length from the start of the UDP header. The sender MUST insert up to three bytes of padding to align the offset of the Type field in the UDP Surplus Header to four bytes. Padding bytes MUST be set to zero.

If the USH is being used as a protocol trailer then the UDP Surplus Header follows the UDP data. If a protocol header is being set then the UDP Surplus Header follows the eight byte UDP header and the UDP data follows the UDP Surplus Header.

The IP Length field in the IPv4 header or Total Length field in the IPv6 header MUST be set to include the UDP datagram and the UDP surplus space. The UDP Length field MUST be set to size of the UDP header (eight) plus the size of the UDP data in the protocol trailer use case, and MUST be set to the size of the UDP header (eight) in the protocol header use case.

The TSLength field MUST be set to reflect the length of the Type Specific Data. The Type Specific Data MUST be padded if necessary to align its length to four bytes.

The USH Checksum MUST be set. To compute the checksum:

1) Set the Checksum field to zero. Compute the standard one's complement two byte checksum starting from the Type field through the end of the IP packet (end of the surplus space). If the length of the surplus space is odd then a zero byte is logically appended for the purposes of the calculation.

2) Set the value of the Checksum field to the bitwise "not" of the checksum computed in the previous step.

3.2 Receiver operation

The processing for a UDP packet with surplus space is:

1) Check for minimum length to contain a UDP Surplus Header. If the UDP surplus space length is less than 3 - ((udp_length - 1) % 4) + 4, then the UDP Surplus Header is considered invalid.

2) Check padding bytes. If the UDP Length is not a multiple of four bytes then verify that the padding bytes following the UDP payload are set to zero. The required number of padding bytes is 3 - ((udp_length - 1) % 4). If the padding bytes are not zero, the UDP Surplus Header is considered invalid.

3) Check the TSLength field. If the length determined from the TSLength field plus the starting offset of the Type Specific Data exceeds the length of the IP packet then the UDP Surplus Header is considered invalid.

4) Verify the checksum. Compute the one's complement checksum starting from the Type field through the end of the IP packet (the end of the surplus area). If the result of the computation is checksum zero (~0 or -0) then the checksum is verified. If the checksum is not verified then the UDP Surplus Header is considered invalid.

5) Check the Type. If the Type is unknown to the receiver then the surplus header is considered invalid.

6) Process the Type Specific Data per the Type in the UDP Surplus Header. If an error condition is encountered in the course of processing the Type Specific Data then the receiver SHOULD consider that the UDP Surplus Header is invalid.

7) In the protocol trailer use case, if there are additional bytes beyond the UDP Surplus Header, a receiver SHOULD ignore those bytes (with the exception that the excess bytes MUST be included in the USH Checksum computation).

8) If the UDP Surplus Header is validated and processed, deliver the UDP data to the application.

   In the case of a protocol trailer, the surplus area is discarded and the UDP data, which follows the UDP header and has length of UDP Length minus eight, is delivered to the

application.

In the case of protocol header, the UDP data delivered to the application immediately follows the UDP Surplus Header and has length of IP_payload_length - 12 - (4 * TSLength).

3.2.1 Error handling

If an error is encountered when processing the UDP space or UDP Surplus Header such that the UDP Surplus Header is considered invalid, then the following actions should be taken:

- In the protocol trailer case (UDP Length greater than eight), the UDP surplus area SHOULD be ignored per protocol processing convention. An implementation MAY allow configuration that would discard such packets. An implementation MUST either process the surplus space or ignore the whole space. In particular, the UDP Surplus Header MUST NOT be partially processed lest that leads to indeterminate results of processing an accepted packet.

- In the case of a protocol header (a UDP packet having exactly a length of eight), the receiver SHOULD discard packets with malformed UDP surplus space or UDP Surplus Header. A receiver MAY deliver the packet to the application in the unlikely scenario that the application applies semantics to zero length UDP datagrams and there is the possibility that the surplus space is a legacy use case (i.e. the sender set surplus space but doesn't use the UDP Surplus Header format).

4 Motivation

This section describes the motivations for the UDP Surplus Header and motivation for protocol headers.

- While the UDP surplus area was implicitly created by [RFC768], the space was never specifically reserved by IETF action. Prescribing a format enables interoperable and backwards compatible use of this space within the context of defined protocol specifications.

- A common header allows different uses and extensibility of the UDP surplus space within a common framework. This is achieved by inclusion of a Type field and Type Specific Data in the UDP Surplus Header. For instance, legacy uses of surplus space could be adapted to use the format and brought into conformance.

- Since the UDP surplus space was never reserved, there is a possibility that the UDP surplus space is already being used by

some implementations. Disambiguating these "legacy" use cases from a newly defined standard format is essential. The required Checksum field, and to a lesser extent the Type and TSLength fields, help disambiguate uses of the surplus area from legacy or accidental uses of the surplus area. Use of the extended UDP header format also reduces the chances of misinterpreting legacy uses.

- The USH checksum is checksum offload friendly. See appendix A for discussion on checksum offload and USH.

- The required checksum in the UDP Surplus Header properly compensates for those devices that incorrectly compute UDP checksum over the length of the IP payload as opposed to just the UDP length.

- A fixed checksum, as opposed to placing a checksum in options, avoids the problem that a checksum can't protect against corruption of the type field for the option containing the checksum.

- Protocols headers, such as those used in the Extended UDP Header format, are more implementation friendly than protocol trailers. See Appendix B for more discussion.

- Maintaining four byte alignment, as is common in IP protocols, is beneficial to implementations on several hardware architectures. See Appendix C for more discussion.

5 Security Considerations

The UDP Surplus Header does not address nor introduce any new security considerations. The Type Specific Data in a UDP Surplus Header may contain security protocol mechanisms or require additional security considerations. Security considerations for Type Specific Data is out of scope for this document.

6 IANA Considerations

IANA is requested to create a registry for the UDP Surplus Header Types.

7 References

7.1 Normative References

7.2 Informative References

[UDPOPT] Touch, J., "Transport Options for UDP", draft-ietf-tsvwg-udp-options-07

Appendix A: Checksum processing

This appendix is informational and does not constitute a normative part of this document.

Checksum offload is a ubiquitous feature of Network Interface Cards (NICs) that offloads checksum computation to hardware for performance. This section suggests some implementation techniques to best leverage checksum offload when UDP surplus space is being used.

Note that the USH checksum ensures that the checksum computed over the UDP surplus space sums to zero in one's complement arithmetic. This has the intended consequence that the UDP checksum calculation over just the UDP length results in the same value when the UDP checksum is computed over the UDP length and surplus space as well. This property can be exploited for efficient and interoperable processing.

A.1 Transmit Checksum processing

A UDP packet with a UDP Surplus Header has two checksum that may need to be set on transmission: the UDP checksum and the USH checksum. The UDP checksum is optional for IPv4 and is required for IPv6 except in very narrow circumstances described in [RFC6936]. The USH checksum is always required to be set.

Most devices only offload one checksum on transmit, so a design objective is to offload the checksum that covers the most bytes and hence provides the most benefit to offload. The checksum that is not offloaded is computed by the host CPU. Generally, the checksum that covers the UDP data is the one covers the most data and should be offloaded. That is, when USH is a protocol trailer the UDP checksum should be offloaded, and when the USH is a protocol header (i.e. extended UDP header) the USH checksum should be offloaded.

In generic checksum offload, for each packet the host indicates to the device the starting offset where the checksum calculation begins and the offset of the field to write the resultant checksum. The extent of the checksum coverage is assumed to be the end of the packet. In particular, this means that even if the UDP checksum is being offloaded, the UDP surplus space is included in the device's computation. Ensuring that the surplus space sums to zero in one's complement arithmetic avoids any ambiguity with checksum offload.

A.1.1 TX checksum for USH trailer

The recommended procedure for setting checksums when the UDP Surplus Header is a trailer is:

1) On the host set the USH checksum using the normal procedures for setting the checksum (section 3.1).

2) Arrange for the UDP checksum to be offloaded to the device. This is done by indicating the checksum start offset to be the first byte of UDP header, indicating the checksum field offset to be the offset of the UDP checksum field, and initializing the UDP checksum field to the "bitwise not" of the appropriate IP pseudo header.

Step 1) ensures that the surplus area sums to zero in one's complement arithmetic, so that in step 2) the value that the device sets in the UDP checksum field will be correct regardless of whether the device includes the surplus area in its computation or not.

Note that the USH padding must be set to zero so it does not affect the checksum computed in step 1). The USH checksum on transmission can be correctly computed by starting the checksum computation from the offset of USH Type field.

A.1.2 TX checksum for USH header

The recommended procedure for setting checksums when the UDP Surplus Header is a header is:

1) Set the UDP checksum on the host. This is normal procedures to set the UDP checksum for a UDP datagram with length of eight.

2) Arrange to offload the USH checksum. The USH checksum field is initialized to zero, the offset to start the checksum calculation is set to the offset of the Type field in the USH, and the checksum field offset is set to the offset of the USH checksum field.

A.2 Receive Checksum handling

In the most generic form of receive checksum offload, a device performs a running checksum calculation across a packet as it is received. That is, it performs a running ones complement addition over two byte words as they are received. The device then provides the computed value, referred to as the "checksum complete" value, to the host in the meta data (receive descriptor) for the packet. The host can use this value to verify one or more packet checksums contained in the packet.

A.2.1 Simultaneous verification

If a device provides a checksum complete value and the UDP checksum

is set, then both the UDP checksum and USH checksum can be simultaneously verified:

1) Pull up checksum to start of the UDP header. That is the checksum complete value is computed from the start of the UDP header through the end of the IP packet.

2) Verify the UDP checksum taking into account the pseudo header. If the UDP checksum is verified, then the USH checksum is also verified.

If the simultaneous verification fails then further work might be needed if checksum failure of the surplus space does not result in the packet being dropped. For instance, if the surplus space is to be ignored in the trailer use case.

A.2.2 RX checksum for USH trailer

The recommended procedure for independently verifying the UDP and USH checksums when the UDP Surplus Header is a protocol trailer is:

1) Compute the one's complement checksum across the UDP surplus space. If checksum zero is the result, then the USH checksum is verified.

2) Perform one's complement subtraction of the value derived in step 1) from the checksum complete value. The result is the checksum complete value across just the UDP header and UDP data.

3) Compute the IP pseudo header for the UDP checksum and one's complement add the result to that of step 2). If the result is checksum zero then the UDP checksum is verified.

If the UDP checksum is zero (unset) then only the USH checksum needs to be verified so steps 2) and 3) can be omitted.

A.2.3 RX checksum for USH header

The recommended procedure for independently verifying the UDP and USH checksums when the UDP Surplus Header is a protocol header is:

1) Compute the one's complement checksum across the UDP header.

2) Compute the IP pseudo header for the UDP checksum and one's complement add the result to that of step 1). If the result is checksum zero then the checksum of the UDP header (zero length datagram) is verified.

3) Perform one's complement subtraction of the value derived in step 1) from the checksum complete value. The result is the checksum complete value across just the UDP surplus space. If zero is the result, then the USH checksum is valid.

If the UDP checksum is zero (unset) then only the USH checksum needs to be verified, so step 2) can be omitted.

Appendix B: Protocol headers versus versus protocol trailers

This appendix is informational and does not constitute a normative part of this document.

Protocol headers by definition are data at the precede the payload of a packet, whereas protocol trailers follow the payload. By nearly universal convention, IP protocols specify protocol headers (e.g. IP, TCP, UDP, Extension headers) and not protocol trailers. A notable exception to this is ESP where the integrity check value is placed after the payload data.

Both software and hardware implementations are designed and optimized for processing protocol headers.

A common technique in software implementations is to "pull up" all the headers in a packet into a contiguous buffer as various protocol layers are processed. To process a protocol trailer, such as a UDP Surplus Header in the trailer use case, an alternate mechanism is needed. This may result in copying data from the end of the packet into a contiguous buffer. Another disadvantage of protocol trailers is that when they are processed a cache miss is almost certain. This will be especially noticeable with hardware techniques that attempt to pre-populate the CPU data cache with some number of header bytes (such as data Direct Data I/O).

High performance hardware devices that perform Deep Packet Inspection (DPI) will be even more sensitive to protocol trailers. Often such devices have a fixed length parsing buffer of X bytes (where X is commonly 64, 128, or 256 bytes). When a device receives a packet, the first X bytes of the packet are preloaded into the parsing buffer before processing commences. Protocol processing is performed on the bytes in the parsing buffer. If the protocol headers extend beyond the parsing buffer then either the device won't process the headers (which may mean they drop the packet) or the packet is relegated to a slow path. Neither behavior is desirable. Given that protocol trailers follow packet payload, it will be common that the protocol trailers for a packet are not contained with parsing buffer.

Appendix C: Protocol field alignment

This appendix is informational and does not constitute a normative part of this document.

It is often convenient to access multi-byte protocol fields in a protocol header in memory using CPU instructions to access a field as a word (two bytes) or double word (four bytes). When such accesses are done, the data being accessed can be "aligned" or "unaligned". An aligned data access happens when the address of the operation modulo the size of the operand is zero, and conversely an unaligned access occurs when the when the address of the operation modulo the size of the operand is non-zero. On certain CPU architectures including SPARC, older versions of ARM, some cases of RISC-V, and even a corner case in x86, an unaligned access may incur a substantial performance penalty compared to an aligned access. For instance, an unaligned access may result in a software trap and handling the memory access in software.

By convention, most IETF protocols are structured to ensure that multi-byte fields have an offset within the respective protocol header that is properly aligned per their field size. Additionally, most IP protocols are defined to have length that is a multiple of four bytes. These conventions, along with some implementation techniques, have mostly allowed software implementations to be reusable across different architectures without the sustaining performance hit of unaligned accesses.

The Padding field in UDP Surplus Header is important to maintain the benefits of aligned protocol headers.

Author's Address

Tom Herbert
Intel
Santa Clara, CA
USA

Email: tom@quantonium.net