Network Working Group
Request for Comments: 5044
Category: Standards Track

P. Culley
Hewlett-Packard Company
U. Elzur
Broadcom Corporation
R. Recio
IBM Corporation
S. Bailey
Sandburst Corporation
J. Carrier
Cray Inc.

October 2007

Marker PDU Aligned Framing for TCP Specification

Status of This Memo

Abstract

Marker PDU Aligned Framing (MPA) is designed to work as an "adaptation layer" between TCP and the Direct Data Placement protocol (DDP) as described in RFC 5041. It preserves the reliable, in-order delivery of TCP, while adding the preservation of higher-level protocol record boundaries that DDP requires. MPA is fully compliant with applicable TCP RFCs and can be utilized with existing TCP implementations. MPA also supports integrated implementations that combine TCP, MPA and DDP to reduce buffering requirements in the implementation and improve performance at the system level.

Table of Contents

Table of Figures

1. Introduction

This section discusses the reason for creating MPA on TCP and a general overview of the protocol.

1.1. Motivation

The Direct Data Placement protocol [DDP], when used with TCP [RFC793], requires a mechanism to detect record boundaries. The DDP records are referred to as Upper Layer Protocol Data Units by this document. The ability to locate the Upper Layer Protocol Data Unit (ULPDU) boundary is useful to a hardware network adapter that uses DDP to directly place the data in the application buffer based on the control information carried in the ULPDU header. This may be done without requiring that the packets arrive in order. Potential benefits of this capability are the avoidance of the memory copy overhead and a smaller memory requirement for handling out-of-order or dropped packets.

Many approaches have been proposed for a generalized framing mechanism. Some are probabilistic in nature and others are deterministic. An example probabilistic approach is characterized by a detectable value embedded in the octet stream, with no method of preventing that value elsewhere within user data. It is probabilistic because under some conditions the receiver may incorrectly interpret application data as the detectable value. Under these conditions, the protocol may fail with unacceptable frequency. One deterministic approach is characterized by embedded controls at known locations in the octet stream. Because the receiver can guarantee it will only examine the data stream at locations that are known to contain the embedded control, the protocol can never misinterpret application data as being embedded control data. For unambiguous handling of an out-of-order packet, a deterministic approach is preferred.

The MPA protocol provides a framing mechanism for DDP running over TCP using the deterministic approach. It allows the location of the ULPDU to be determined in the TCP stream even if the TCP segments arrive out of order.

1.2. Protocol Overview

The layering of PDUs with MPA is shown in Figure 1, below.

```
            +-------------------+
            |     ULP client    |
            +-------------------+  <- Consumer messages
            |        DDP        |
            +-------------------+  <- ULPDUs
            |        MPA*       |
            +-------------------+  <- FPDUs (containing ULPDUs)
            |        TCP*       |
            +-------------------+  <- TCP Segments (containing FPDUs)
            |      IP etc.      |
            +-------------------+
             * These may be fully layered or optimized together.
```

Figure 1: ULP MPA TCP Layering

MPA is described as an extra layer above TCP and below DDP. The operation sequence is:

1. A TCP connection is established by ULP action. This is done using methods not described by this specification. The ULP may exchange some amount of data in streaming mode prior to starting MPA, but is not required to do so.

2. The Consumer negotiates the use of DDP and MPA at both ends of a connection. The mechanisms to do this are not described in this specification. The negotiation may be done in streaming mode, or by some other mechanism (such as a pre-arranged port number).

3. The ULP activates MPA on each end in the Startup Phase, either as an Initiator or a Responder, as determined by the ULP. This mode verifies the usage of MPA, specifies the use of CRC and Markers, and allows the ULP to communicate some additional data via a Private Data exchange. See Section 7.1, Connection Setup, for more details on the startup process.

4. At the end of the Startup Phase, the ULP puts MPA (and DDP) into Full Operation and begins sending DDP data as further described below. In this document, DDP data chunks are called ULPDUs. For a description of the DDP data, see [DDP].

Following is a description of data transfer when MPA is in Full Operation.

1. DDP determines the Maximum ULPDU (MULPDU) size by querying MPA for this value. MPA derives this information from TCP or IP, when it is available, or chooses a reasonable value.

2. DDP creates ULPDUs of MULPDU size or smaller, and hands them to MPA at the sender.

3. MPA creates a Framed Protocol Data Unit (FPDU) by prepending a header, optionally inserting Markers, and appending a CRC field after the ULPDU and PAD (if any). MPA delivers the FPDU to TCP.

4. The TCP sender puts the FPDUs into the TCP stream. If the sender is optimized MPA/TCP, it segments the TCP stream in such a way that a TCP Segment boundary is also the boundary of an FPDU. TCP then passes each segment to the IP layer for transmission.

5. The receiver may or may not be optimized. If it is optimized MPA/TCP, it may separate passing the TCP payload to MPA from passing the TCP payload ordering information to MPA. In either case, RFC-compliant TCP wire behavior is observed at both the sender and receiver.

6. The MPA receiver locates and assembles complete FPDUs within the stream, verifies their integrity, and removes MPA Markers (when present), ULPDU_Length, PAD, and the CRC field.

7. MPA then provides the complete ULPDUs to DDP. MPA may also separate passing MPA payload to DDP from passing the MPA payload ordering information.

A fully layered MPA on TCP is implemented as a data stream ULP for TCP and is therefore RFC compliant.

An optimized DDP/MPA/TCP uses a TCP layer that potentially contains some additional behaviors as suggested in this document. When DDP/MPA/TCP are cross-layer optimized, the behavior of TCP (especially sender segmentation) may change from that of the un-optimized implementation, but the changes are within the bounds permitted by the TCP RFC specifications, and will interoperate with an un-optimized TCP. The additional behaviors are described in Appendix A and are not normative; they are described at a TCP interface layer as a convenience. Implementations may achieve the described functionality using any method, including cross-layer optimizations between TCP, MPA, and DDP.

An optimized DDP/MPA/TCP sender is able to segment the data stream such that TCP segments begin with FPDUs (FPDU Alignment). This has significant advantages for receivers. When segments arrive with aligned FPDUs, the receiver usually need not buffer any portion of the segment, allowing DDP to place it in its destination memory immediately, thus avoiding copies from intermediate buffers (DDP’s reason for existence).

An optimized DDP/MPA/TCP receiver allows a DDP on MPA implementation to locate the start of ULPDUs that may be received out of order. It also allows the implementation to determine if the entire ULPDU has been received. As a result, MPA can pass out-of-order ULPDUs to DDP for immediate use. This enables a DDP on MPA implementation to save a significant amount of intermediate storage by placing the ULPDUs in the right locations in the application buffers when they arrive, rather than waiting until full ordering can be restored.

The ability of a receiver to recover out-of-order ULPDUs is optional and declared to the transmitter during startup. When the receiver declares that it does not support out-of-order recovery, the transmitter does not add the control information to the data stream needed for out-of-order recovery.

If the receiver is fully layered, then MPA receives a strictly ordered stream of data and does not deal with out-of-order ULPDUs. In this case, MPA passes each ULPDU to DDP when the last bytes arrive from TCP, along with the indication that they are in order.

MPA implementations that support recovery of out-of-order ULPDUs MUST support a mechanism to indicate the ordering of ULPDUs as the sender transmitted them and indicate when missing intermediate segments arrive. These mechanisms allow DDP to reestablish record ordering and report Delivery of complete messages (groups of records).

MPA also addresses enhanced data integrity. Some users of TCP have noted that the TCP checksum is not as strong as could be desired (see [CRCTCP]). Studies such as [CRCTCP] have shown that the TCP checksum indicates segments in error at a much higher rate than the underlying link characteristics would indicate. With these higher error rates, the chance that an error will escape detection, when using only the TCP checksum for data integrity, becomes a concern. A stronger integrity check can reduce the chance of data errors being missed.

MPA includes a CRC check to increase the ULPDU data integrity to the level provided by other modern protocols, such as SCTP [RFC4960]. It is possible to disable this CRC check; however, CRCs MUST be enabled unless it is clear that the end-to-end connection through the network has data integrity at least as good as an MPA with CRC enabled (for example, when IPsec is implemented end to end). DDP’s ULP expects this level of data integrity and therefore the ULP does not have to provide its own duplicate data integrity and error recovery for lost data.

2. Glossary

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Consumer - the ULPs or applications that lie above MPA and DDP. The Consumer is responsible for making TCP connections, starting MPA and DDP connections, and generally controlling operations.

CRC - Cyclic Redundancy Check.

Delivery - (Delivered, Delivers) - For MPA, Delivery is defined as the process of informing DDP that a particular PDU is ordered for use. A PDU is Delivered in the exact order that it was sent by the original sender; MPA uses TCP’s byte stream ordering to determine when Delivery is possible. This is specifically different from "passing the PDU to DDP", which may generally occur in any order, while the order of Delivery is strictly defined.

EMSS - Effective Maximum Segment Size. EMSS is the smaller of the TCP maximum segment size (MSS) as defined in RFC 793 [RFC793], and the current path Maximum Transmission Unit (MTU) [RFC1191].

FPDU - Framed Protocol Data Unit. The unit of data created by an MPA sender.

FPDU Alignment - The property that an FPDU is Header Aligned with the TCP segment, and the TCP segment includes an integer number of FPDUs. A TCP segment with an FPDU Alignment allows immediate processing of the contained FPDUs without waiting on other TCP segments to arrive or combining with prior segments.

FPDU Pointer (FPDUPTR) - This field of the Marker is used to indicate the beginning of an FPDU.

Full Operation (Full Operation Phase) - After the completion of the Startup Phase, MPA begins exchanging FPDUs.

Header Alignment - The property that a TCP segment begins with an FPDU. The FPDU is Header Aligned when the FPDU header is exactly at the start of the TCP segment (right behind the TCP headers on the wire).

Initiator - The endpoint of a connection that sends the MPA Request Frame, i.e., the first to actually send data (which may not be the one that sends the TCP SYN).

Marker - A four-octet field that is placed in the MPA data stream at fixed octet intervals (every 512 octets).

MPA-aware TCP - A TCP implementation that is aware of the receiver efficiencies of MPA FPDU Alignment and is capable of sending TCP segments that begin with an FPDU.

MPA-enabled - MPA is enabled if the MPA protocol is visible on the wire. When the sender is MPA-enabled, it is inserting framing and Markers. When the receiver is MPA-enabled, it is interpreting framing and Markers.

MPA Request Frame - Data sent from the MPA Initiator to the MPA Responder during the Startup Phase.

MPA Reply Frame - Data sent from the MPA Responder to the MPA Initiator during the Startup Phase.

MPA - Marker-based ULP PDU Aligned Framing for TCP protocol. This document defines the MPA protocol.

MULPDU - Maximum ULPDU. The current maximum size of the record that is acceptable for DDP to pass to MPA for transmission.

Node - A computing device attached to one or more links of a network. A Node in this context does not refer to a specific application or protocol instantiation running on the computer. A Node may consist of one or more MPA on TCP devices installed in a host computer.

PAD - A 1-3 octet group of zeros used to fill an FPDU to an exact modulo 4 size.

PDU - Protocol data unit

Private Data - A block of data exchanged between MPA endpoints during initial connection setup.

Protection Domain - An RDMA concept (see [VERBS-RDMA] and [RDMASEC]) that ties use of various endpoint resources (memory access, etc.) to the specific RDMA/DDP/MPA connection.

RDDP - A suite of protocols including MPA, [DDP], [RDMAP], an overall security document [RDMASEC], a problem statement [RFC4297], an architecture document [RFC4296], and an applicability document [APPL].

RDMA - Remote Direct Memory Access; a protocol that uses DDP and MPA to enable applications to transfer data directly from memory buffers. See [RDMAP].

Remote Peer - The MPA protocol implementation on the opposite end of the connection. Used to refer to the remote entity when describing protocol exchanges or other interactions between two Nodes.

Responder - The connection endpoint that responds to an incoming MPA connection request (the MAP Request Frame). This may not be the endpoint that awaited the TCP SYN.

Startup Phase - The initial exchanges of an MPA connection that serves to more fully identify MPA endpoints to each other and pass connection specific setup information to each other.

ULP - Upper Layer Protocol. The protocol layer above the protocol layer currently being referenced. The ULP for MPA is DDP [DDP].

ULPDU - Upper Layer Protocol Data Unit. The data record defined by the layer above MPA (DDP). ULPDU corresponds to DDP’s DDP segment.

ULPDU_Length - A field in the FPDU describing the length of the included ULPDU.

3. MPA’s Interactions with DDP

DDP requires MPA to maintain DDP record boundaries from the sender to the receiver. When using MPA on TCP to send data, DDP provides records (ULPDUs) to MPA. MPA will use the reliable transmission abilities of TCP to transmit the data, and will insert appropriate additional information into the TCP stream to allow the MPA receiver to locate the record boundary information.

As such, MPA accepts complete records (ULPDUs) from DDP at the sender and returns them to DDP at the receiver.

MPA MUST encapsulate the ULPDU such that there is exactly one ULPDU contained in one FPDU.

MPA over a standard TCP stack can usually provide FPDU Alignment with the TCP Header if the FPDU is equal to TCP’s EMSS. An optimized MPA/TCP stack can also maintain alignment as long as the FPDU is less than or equal to TCP’s EMSS. Since FPDU Alignment is generally desired by the receiver, DDP cooperates with MPA to ensure FPDUs’ lengths do not exceed the EMSS under normal conditions. This is done with the MULPDU mechanism.

MPA MUST provide information to DDP on the current maximum size of the record that is acceptable to send (MULPDU). DDP SHOULD limit each record size to MULPDU. The range of MULPDU values MUST be between 128 octets and 64768 octets, inclusive.

The sending DDP MUST NOT post a ULPDU larger than 64768 octets to MPA. DDP MAY post a ULPDU of any size between one and 64768 octets; however, MPA is not REQUIRED to support a ULPDU Length that is greater than the current MULPDU.

While the maximum theoretical length supported by the MPA header ULPDU_Length field is 65535, TCP over IP requires the IP datagram maximum length to be 65535 octets. To enable MPA to support FPDU Alignment, the maximum size of the FPDU must fit within an IP datagram. Thus, the ULPDU limit of 64768 octets was derived by taking the maximum IP datagram length, subtracting from it the maximum total length of the sum of the IPv4 header, TCP header, IPv4 options, TCP options, and the worst-case MPA overhead, and then rounding the result down to a 128-octet boundary.

Note that MULPDU will be significantly smaller than the theoretical maximum in most implementations for most circumstances, due to link MTUs, use of extra headers such as required for IPsec, etc.

On receive, MPA MUST pass each ULPDU with its length to DDP when it has been validated.

If an MPA implementation supports passing out-of-order ULPDUs to DDP, the MPA implementation SHOULD:

- Pass each ULPDU with its length to DDP as soon as it has been fully received and validated.

- Provide a mechanism to indicate the ordering of ULPDUs as the sender transmitted them. One possible mechanism might be providing the TCP sequence number for each ULPDU.

- Provide a mechanism to indicate when a given ULPDU (and prior ULPDUs) are complete (Delivered to DDP). One possible mechanism might be to allow DDP to see the current outgoing TCP ACK sequence number.

- Provide an indication to DDP that the TCP has closed or has begun to close the connection (e.g., received a FIN).

MPA MUST provide the protocol version negotiated with its peer to DDP. DDP will use this version to set the version in its header and to report the version to [RDMAP].

4. MPA Full Operation Phase

The following sections describe the main semantics of the Full Operation Phase of MPA.

4.1. FPDU Format

MPA senders create FPDUs out of ULPDUs. The format of an FPDU shown below MUST be used for all MPA FPDUs. For purposes of clarity, Markers are not shown in Figure 2.

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |          ULPDU_Length         |                               |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
      |                                                               |
      ~                                                               ~
      ~                             ULPDU                             ~
      |                                                               |
      |                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                               |          PAD (0-3 octets)     |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                             CRC                               |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: FPDU Format

ULPDU_Length: 16 bits (unsigned integer). This is the number of octets of the contained ULPDU. It does not include the length of the FPDU header itself, the pad, the CRC, or of any Markers that fall within the ULPDU. The 16-bit ULPDU Length field is large enough to support the largest IP datagrams for IPv4 or IPv6.

PAD: The PAD field trails the ULPDU and contains between 0 and 3 octets of data. The pad data MUST be set to zero by the sender and ignored by the receiver (except for CRC checking). The length of the pad is set so as to make the size of the FPDU an integral multiple of four.

CRC: 32 bits. When CRCs are enabled, this field contains a CRC32c check value, which is used to verify the entire contents of the FPDU, using CRC32c. See Section 4.4, CRC Calculation. When CRCs are not enabled, this field is still present, may contain any value, and MUST NOT be checked.

The FPDU adds a minimum of 6 octets to the length of the ULPDU. In addition, the total length of the FPDU will include the length of any Markers and from 0 to 3 pad octets added to round-up the ULPDU size.

4.2. Marker Format

The format of a Marker MUST be as specified in Figure 3:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |           RESERVED            |            FPDUPTR            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: Marker Format

RESERVED: The Reserved field MUST be set to zero on transmit and ignored on receive (except for CRC calculation).

FPDUPTR: The FPDU Pointer is a relative pointer, 16 bits long, interpreted as an unsigned integer that indicates the number of octets in the TCP stream from the beginning of the ULPDU Length field to the first octet of the entire Marker. The least significant two bits MUST always be set to zero at the transmitter, and the receivers MUST always treat these as zero for calculations.

4.3. MPA Markers

MPA Markers are used to identify the start of FPDUs when packets are received out of order. This is done by locating the Markers at fixed intervals in the data stream (which is correlated to the TCP sequence number) and using the Marker value to locate the preceding FPDU start.

All MPA Markers are included in the containing FPDU CRC calculation (when both CRCs and Markers are in use).

The MPA receiver’s ability to locate out-of-order FPDUs and pass the ULPDUs to DDP is implementation dependent. MPA/DDP allows those receivers that are able to deal with out-of-order FPDUs in this way to require the insertion of Markers in the data stream. When the receiver cannot deal with out-of-order FPDUs in this way, it may disable the insertion of Markers at the sender. All MPA senders MUST be able to generate Markers when their use is declared by the opposing receiver (see Section 7.1, Connection Setup).

When Markers are enabled, MPA senders MUST insert a Marker into the data stream at a 512-octet periodic interval in the TCP Sequence Number Space. The Marker contains a 16-bit unsigned integer referred to as the FPDUPTR (FPDU Pointer).

If the FPDUPTR's value is non-zero, the FPDU Pointer is a 16-bit relative back-pointer. FPDUPTR MUST contain the number of octets in the TCP stream from the beginning of the ULPDU Length field to the first octet of the Marker, unless the Marker falls between FPDUs. Thus, the location of the first octet of the previous FPDU header can be determined by subtracting the value of the given Marker from the current octet-stream sequence number (i.e., TCP sequence number) of the first octet of the Marker. Note that this computation MUST take into account that the TCP sequence number could have wrapped between the Marker and the header.

An FPDUPTR value of 0x0000 is a special case -- it is used when the Marker falls exactly between FPDUs (between the preceding FPDU CRC field and the next FPDU's ULPDU Length field). In this case, the Marker is considered to be contained in the following FPDU; the Marker MUST be included in the CRC calculation of the FPDU following the Marker (if CRCs are being generated or checked). Thus, an FPDUPTR value of 0x0000 means that immediately following the Marker is an FPDU header (the ULPDU Length field).

Since all FPDUs are integral multiples of 4 octets, the bottom two bits of the FPDUPTR as calculated by the sender are zero. MPA reserves these bits so they MUST be treated as zero for computation at the receiver.

When Markers are enabled (see Section 7.1, Connection Setup), the MPA Markers MUST be inserted immediately preceding the first FPDU of Full Operation Phase, and at every 512th octet of the TCP octet stream thereafter. As a result, the first Marker has an FPDUPTR value of 0x0000. If the first Marker begins at octet sequence number SeqStart, then Markers are inserted such that the first octet of the Marker is at octet sequence number SeqNum if the remainder of (SeqNum - SeqStart) mod 512 is zero. Note that SeqNum can wrap.

For example, if the TCP sequence number were used to calculate the insertion point of the Marker, the starting TCP sequence number is unlikely to be zero, and 512-octet multiples are unlikely to fall on a modulo 512 of zero. If the MPA connection is started at TCP sequence number 11, then the 1st Marker will begin at 11, and subsequent Markers will begin at 523, 1035, etc.

If an FPDU is large enough to contain multiple Markers, they MUST all point to the same point in the TCP stream: the first octet of the ULPDU Length field for the FPDU.

If a Marker interval contains multiple FPDUs (the FPDUs are small), the Marker MUST point to the start of the ULPDU Length field for the FPDU containing the Marker unless the Marker falls between FPDUs, in which case the Marker MUST be zero.

The following example shows an FPDU containing a Marker.

```
   0                   1                   2                   3
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |       ULPDU Length (0x0010)   |                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               +
   |                                                               |
   +                                                               +
   |                         ULPDU (octets 0-9)                    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |            (0x0000)           |        FPDU ptr (0x000C)      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        ULPDU (octets 10-15)                   |
   |                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                               |          PAD (2 octets:0,0)   |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                              CRC                              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: Example FPDU Format with Marker

MPA Receivers MUST preserve ULPDU boundaries when passing data to DDP. MPA Receivers MUST pass the ULPDU data and the ULPDU Length to DDP and not the Markers, headers, and CRC.

4.4. CRC Calculation

An MPA implementation MUST implement CRC support and MUST either:

(1) always use CRCs; the MPA provider is not REQUIRED to support an administrator’s request that CRCs not be used.

or

(2a) only indicate a preference not to use CRCs on the explicit request of the system administrator, via an interface not defined in this spec. The default configuration for a connection MUST be to use CRCs.

(2b) disable CRC checking (and possibly generation) if both the local and remote endpoints indicate preference not to use CRCs.

An administrative decision to have a host request CRC suppression SHOULD NOT be made unless there is assurance that the TCP connection involved provides protection from undetected errors that is at least as strong as an end-to-end CRC32c. End-to-end usage of an IPsec cryptographic integrity check is among the ways to provide such protection, and the use of channel bindings [NFSv4CHANNEL] by the ULP can provide a high level of assurance that the IPsec protection scope is end-to-end with respect to the ULP.

The process MUST be invisible to the ULP.

After receipt of an MPA startup declaration indicating that its peer requires CRCs, an MPA instance MUST continue generating and checking CRCs until the connection terminates. If an MPA instance has declared that it does not require CRCs, it MUST turn off CRC checking immediately after receipt of an MPA mode declaration indicating that its peer also does not require CRCs. It MAY continue generating CRCs. See Section 7.1, Connection Setup, for details on the MPA startup.

When sending an FPDU, the sender MUST include a CRC field. When CRCs are enabled, the CRC field in the MPA FPDU MUST be computed using the CRC32c polynomial in the manner described in the iSCSI Protocol [iSCSI] document for Header and Data Digests.

The fields which MUST be included in the CRC calculation when sending an FPDU are as follows:

1) If a Marker does not immediately precede the ULPDU Length field, the CRC-32c is calculated from the first octet of the ULPDU Length field, through all the ULPDU and Markers (if present), to the last octet of the PAD (if present), inclusive. If there is a Marker immediately following the PAD, the Marker is included in the CRC calculation for this FPDU.

2) If a Marker immediately precedes the first octet of the ULPDU Length field of the FPDU, (i.e., the Marker fell between FPDUs, and thus is required to be included in the second FPDU), the CRC-32c is calculated from the first octet of the Marker, through the ULPDU Length header, through all the ULPDU and Markers (if present), to the last octet of the PAD (if present), inclusive.

3) After calculating the CRC-32c, the resultant value is placed into the CRC field at the end of the FPDU.

When an FPDU is received, and CRC checking is enabled, the receiver MUST first perform the following:

1) Calculate the CRC of the incoming FPDU in the same fashion as defined above.

2) Verify that the calculated CRC-32c value is the same as the received CRC-32c value found in the FPDU CRC field. If not, the receiver MUST treat the FPDU as an invalid FPDU.

The procedure for handling invalid FPDUs is covered in Section 8, Error Semantics.

The following is an annotated hex dump of an example FPDU sent as the first FPDU on the stream. As such, it starts with a Marker. The FPDU contains a 42 octet ULPDU (an example DDP segment) which in turn contains 24 octets of the contained ULPDU, which is a data load that is all zeros. The CRC32c has been correctly calculated and can be used as a reference. See the [DDP] and [RDMAP] specification for definitions of the DDP Control field, Queue, MSN, MO, and Send Data.

```
       Octet Contents  Annotation
       Count

       0000    00      Marker: Reserved
       0001    00
       0002    00      Marker: FPDUPTR
       0003    00
       0004    00      ULPDU Length
       0005    2a
       0006    41      DDP Control Field, Send with Last flag set
       0007    43
       0008    00      Reserved (DDP STag position with no STag)
       0009    00
       000a    00
       000b    00
       000c    00      DDP Queue = 0
       000d    00
       000e    00
       000f    00
       0010    00      DDP MSN = 1
       0011    00
       0012    00
       0013    01
       0014    00      DDP MO = 0
       0015    00
       0016    00
       0017    00
       0018    00      DDP Send Data (24 octets of zeros)
       ...
       002f    00
       0030    52      CRC32c
       0031    23
       0032    99
       0033    83
```

Figure 5: Annotated Hex Dump of an FPDU

The following is an example sent as the second FPDU of the stream where the first FPDU (which is not shown here) had a length of 492 octets and was also a Send to Queue 0 with Last Flag set. This example contains a Marker.

```
Octet Contents  Annotation
Count

01ec    00      Length
01ed    2a
01ee    41      DDP Control Field: Send with Last Flag set
01ef    43
01f0    00      Reserved (DDP STag position with no STag)
01f1    00
01f2    00
01f3    00
01f4    00      DDP Queue = 0
01f5    00
01f6    00
01f7    00
01f8    00      DDP MSN = 2
01f9    00
01fa    00
01fb    02
01fc    00      DDP MO = 0
01fd    00
01fe    00
01ff    00
0200    00      Marker: Reserved
0201    00
0202    00      Marker: FPDUPTR
0203    14
0204    00      DDP Send Data (24 octets of zeros)
...
021b    00
021c    84      CRC32c
021d    92
021e    58
021f    98
```

Figure 6: Annotated Hex Dump of an FPDU with Marker

4.5. FPDU Size Considerations

MPA defines the Maximum Upper Layer Protocol Data Unit (MULPDU) as the size of the largest ULPDU fitting in an FPDU. For an empty TCP Segment, MULPDU is EMSS minus the FPDU overhead (6 octets) minus space for Markers and pad octets.

The maximum ULPDU Length for a single ULPDU when Markers are present MUST be computed as:

```
MULPDU = EMSS - (6 + 4 * Ceiling(EMSS / 512) + EMSS mod 4)
```

The formula above accounts for the worst-case number of Markers.

The maximum ULPDU Length for a single ULPDU when Markers are NOT present MUST be computed as:

```
MULPDU = EMSS - (6 + EMSS mod 4)
```

As a further optimization of the wire efficiency an MPA implementation MAY dynamically adjust the MULPDU (see Section 5 for latency and wire efficiency trade-offs). When one or more FPDUs are already packed into a TCP Segment, MULPDU MAY be reduced accordingly.

DDP SHOULD provide ULPDUs that are as large as possible, but less than or equal to MULPDU.

If the TCP implementation needs to adjust EMSS to support MTU changes or changing TCP options, the MULPDU value is changed accordingly.

In certain rare situations, the EMSS may shrink below 128 octets in size. If this occurs, the MPA on TCP sender MUST NOT shrink the MULPDU below 128 octets and is not required to follow the segmentation rules in Section 5.1 and Appendix A.

If one or more FPDUs are already packed into a TCP segment, such that the remaining room is less than 128 octets, MPA MUST NOT provide a MULPDU smaller than 128. In this case, MPA would typically provide a MULPDU for the next full sized segment, but may still pack the next FPDU into the small remaining room, provide that the next FPDU is small enough to fit.

The value 128 is chosen as to allow DDP designers room for the DDP Header and some user data.

5. MPA's interactions with TCP

The following sections describe MPA's interactions with TCP. This section discusses using a standard layered TCP stack with MPA attached above a TCP socket. Discussion of using an optimized MPA-aware TCP with an MPA implementation that takes advantage of the extra optimizations is done in Appendix A.

```
                   +-----------------------------------+
                   | +-----+       +-----------------+ |
                   | | MPA |       | Other Protocols | |
                   | +-----+       +-----------------+ |
                   |    ||                   ||        |
                   |  ----- socket API --------------  |
                   |            ||                     |
                   |         +-----+                   |
                   |         | TCP |                   |
                   |         +-----+                   |
                   |            ||                     |
                   |         +-----+                   |
                   |         | IP  |                   |
                   |         +-----+                   |
                   +-----------------------------------+
```

Figure 7: Fully Layered Implementation

The Fully layered implementation is described for completeness; however, the user is cautioned that the reduced probability of FPDU alignment when transmitting with this implementation will tend to introduce a higher overhead at optimized receivers. In addition, the lack of out-of-order receive processing will significantly reduce the value of DDP/MPA by imposing higher buffering and copying overhead in the local receiver.

5.1. MPA transmitters with a standard layered TCP

MPA transmitters SHOULD calculate a MULPDU as described in Section 4.5. If the TCP implementation allows EMSS to be determined by MPA, that value should be used. If the transmit side TCP implementation is not able to report the EMSS, MPA SHOULD use the current MTU value to establish a likely FPDU size, taking into account the various expected header sizes.

MPA transmitters SHOULD also use whatever facilities the TCP stack presents to cause the TCP transmitter to start TCP segments at FPDU boundaries. Multiple FPDUs MAY be packed into a single TCP segment as determined by the EMSS calculation as long as they are entirely contained in the TCP segment.

For example, passing FPDU buffers sized to the current EMSS to the TCP socket and using the TCP_NODELAY socket option to disable the Nagle [RFC896] algorithm will usually result in many of the segments starting with an FPDU.

It is recognized that various effects can cause an FPDU Alignment to be lost. Following are a few of the effects:

- ULPDUs that are smaller than the MULPDU. If these are sent in a continuous stream, FPDU Alignment will be lost. Note that careful use of a dynamic MULPDU can help in this case; the MULPDU for future FPDUs can be adjusted to re-establish alignment with the segments based on the current EMSS.

- Sending enough data that the TCP receive window limit is reached. TCP may send a smaller segment to exactly fill the receive window.

- Sending data when TCP is operating up against the congestion window. If TCP is not tracking the congestion window in segments, it may transmit a smaller segment to exactly fill the receive window.

- Changes in EMSS due to varying TCP options, or changes in MTU.

If FPDU Alignment with TCP segments is lost for any reason, the alignment is regained after a break in transmission where the TCP send buffers are emptied. Many usage models for DDP/MPA will include such breaks.

MPA receivers are REQUIRED to be able to operate correctly even if alignment is lost (see Section 6).

5.2. MPA receivers with a standard layered TCP

MPA receivers will get TCP data in the usual ordered stream. The receivers MUST identify FPDU boundaries by using the ULPDU_LENGTH field, as described in Section 6. Receivers MAY utilize markers to check for FPDU boundary consistency, but they are NOT required to examine the markers to determine the FPDU boundaries.

6. MPA Receiver FPDU Identification

An MPA receiver MUST first verify the FPDU before passing the ULPDU to DDP. To do this, the receiver MUST:

- locate the start of the FPDU unambiguously,

- verify its CRC (if CRC checking is enabled).

If the above conditions are true, the MPA receiver passes the ULPDU to DDP.

To detect the start of the FPDU unambiguously one of the following MUST be used:

1: In an ordered TCP stream, the ULPDU Length field in the current FPDU when FPDU has a valid CRC, can be used to identify the beginning of the next FPDU.

2: For optimized MPA/TCP receivers that support out-of-order reception of FPDUs (see Section 4.3, MPA Markers) a Marker can always be used to locate the beginning of an FPDU (in FPDUs with valid CRCs). Since the location of the Marker is known in the octet stream (sequence number space), the Marker can always be found.

3: Having found an FPDU by means of a Marker, an optimized MPA/TCP receiver can find following contiguous FPDUs by using the ULPDU Length fields (from FPDUs with valid CRCs) to establish the next FPDU boundary.

The ULPDU Length field (see Section 4) MUST be used to determine if the entire FPDU is present before forwarding the ULPDU to DDP.

CRC calculation is discussed in Section 4.4 above.

7. Connection Semantics

7.1. Connection Setup

MPA requires that the Consumer MUST activate MPA, and any TCP enhancements for MPA, on a TCP half connection at the same location in the octet stream at both the sender and the receiver. This is required in order for the Marker scheme to correctly locate the Markers (if enabled) and to correctly locate the first FPDU.

MPA, and any TCP enhancements for MPA are enabled by the ULP in both directions at once at an endpoint.

This can be accomplished several ways, and is left up to DDP’s ULP:

* DDP’s ULP MAY require DDP on MPA startup immediately after TCP connection setup. This has the advantage that no streaming mode negotiation is needed. An example of such a protocol is shown in Figure 10: Example Immediate Startup negotiation.

  This may be accomplished by using a well-known port, or a service locator protocol to locate an appropriate port on which DDP on MPA is expected to operate.

* DDP’s ULP MAY negotiate the start of DDP on MPA sometime after a normal TCP startup, using TCP streaming data exchanges on the same connection. The exchange establishes that DDP on MPA (as well as other ULPs) will be used, and exactly locates the point in the octet stream where MPA is to begin operation. Note that such a negotiation protocol is outside the scope of this specification. A simplified example of such a protocol is shown in Figure 9: Example Delayed Startup negotiation on page 33.

An MPA endpoint operates in two distinct phases.

The Startup Phase is used to verify correct MPA setup, exchange CRC and Marker configuration, and optionally pass Private Data between endpoints prior to completing a DDP connection. During this phase, specifically formatted frames are exchanged as TCP byte streams without using CRCs or Markers. During this phase a DDP endpoint need not be "bound" to the MPA connection. In fact, the choice of DDP endpoint and its operating parameters may not be known until the Consumer supplied Private Data (if any) has been examined by the Consumer.

The second distinct phase is Full Operation during which FPDUs are sent using all the rules that pertain (CRCs, Markers, MULPDU restrictions, etc.). A DDP endpoint MUST be "bound" to the MPA connection at entry to this phase.

When Private Data is passed between ULPs in the Startup Phase, the ULP is responsible for interpreting that data, and then placing MPA into Full Operation.

Note: The following text differentiates the two endpoints by calling them Initiator and Responder. This is quite arbitrary and is NOT related to the TCP startup (SYN, SYN/ACK sequence). The Initiator is the side that sends first in the MPA startup sequence (the MPA Request Frame).

Note: The possibility that both endpoints would be allowed to make a connection at the same time, sometimes called an active/active connection, was considered by the work group and rejected. There were several motivations for this decision. One was that applications needing this facility were few (none other than theoretical at the time of this document). Another was that the facility created some implementation difficulties, particularly with the "dual stack" designs described later on. A last issue was that dealing with rejected connections at startup would have required at least an additional frame type, and more recovery actions, complicating the protocol. While none of these issues was overwhelming, the group and implementers were not motivated to do the work to resolve these issues. The protocol includes a method of detecting these active/active startup attempts so that they can be rejected and an error reported.

The ULP is responsible for determining which side is Initiator or Responder. For client/server type ULPs, this is easy. For peer-peer ULPs (which might utilize a TCP style active/active startup), some mechanism (not defined by this specification) must be established, or some streaming mode data exchanged prior to MPA startup to determine which side starts in Initiator and which starts in Responder MPA mode.

7.1.1 MPA Request and Reply Frame Format

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   0  |                                                               |
      +         Key (16 bytes containing "MPA ID Req Frame")          +
   4  |      (4D 50 41 20 49 44 20 52 65 71 20 46 72 61 6D 65)        |
      +         Or  (16 bytes containing "MPA ID Rep Frame")          +
   8  |      (4D 50 41 20 49 44 20 52 65 70 20 46 72 61 6D 65)        |
      +                                                               +
   12 |                                                               |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   16 |M|C|R| Res     |     Rev       |          PD_Length            |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                                                               |
      ~                                                               ~
      ~                   Private Data                                ~
      |                                                               |
      |                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                               |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 8: MPA Request/Reply Frame

Key: This field contains the "key" used to validate that the sender is an MPA sender. Initiator mode senders MUST set this field to the fixed value "MPA ID Req Frame" or (in byte order) 4D 50 41 20 49 44 20 52 65 71 20 46 72 61 6D 65 (in hexadecimal). Responder mode receivers MUST check this field for the same value, and close the connection and report an error locally if any other value is detected. Responder mode senders MUST set this field to the fixed value "MPA ID Rep Frame" or (in byte order) 4D 50 41 20 49 44 20 52 65 70 20 46 72 61 6D 65 (in hexadecimal). Initiator mode receivers MUST check this field for the same value, and close the connection and report an error locally if any other value is detected.

M: This bit declares an endpoint’s REQUIRED Marker usage. When this bit is ’1’ in an MPA Request Frame, the Initiator declares that Markers are REQUIRED in FPDUs sent from the Responder. When set to ’1’ in an MPA Reply Frame, this bit declares that Markers are REQUIRED in FPDUs sent from the Initiator. When in a received MPA Request Frame or MPA Reply Frame and the value is ’0’, Markers MUST NOT be added to the data stream by that endpoint. When ’1’ Markers MUST be added as described in Section 4.3, MPA Markers.

C: This bit declares an endpoint’s preferred CRC usage. When this field is ’0’ in the MPA Request Frame and the MPA Reply Frame, CRCs MUST not be checked and need not be generated by either endpoint. When this bit is ’1’ in either the MPA Request Frame or MPA Reply Frame, CRCs MUST be generated and checked by both endpoints. Note that even when not in use, the CRC field remains present in the FPDU. When CRCs are not in use, the CRC field MUST be considered valid for FPDU checking regardless of its contents.

R: This bit is set to zero, and not checked on reception in the MPA Request Frame. In the MPA Reply Frame, this bit is the Rejected Connection bit, set by the Responders ULP to indicate acceptance ’0’, or rejection ’1’, of the connection parameters provided in the Private Data.

Res: This field is reserved for future use. It MUST be set to zero when sending, and not checked on reception.

Rev: This field contains the revision of MPA. For this version of the specification, senders MUST set this field to one. MPA receivers compliant with this version of the specification MUST check this field. If the MPA receiver cannot interoperate with the received version, then it MUST close the connection and report an error locally. Otherwise, the MPA receiver should report the received version to the ULP.

PD_Length: This field MUST contain the length in octets of the Private Data field. A value of zero indicates that there is no Private Data field present at all. If the receiver detects that the PD_Length field does not match the length of the Private Data field, or if the length of the Private Data field exceeds 512 octets, the receiver MUST close the connection and report an error locally. Otherwise, the MPA receiver should pass the PD_Length value and Private Data to the ULP.

Private Data: This field may contain any value defined by ULPs or may not be present. The Private Data field MUST be between 0 and 512 octets in length. ULPs define how to size, set, and validate this field within these limits. Private Data usage is further discussed in Section 7.1.4.

7.1.2. Connection Startup Rules

The following rules apply to MPA connection Startup Phase:

1. When MPA is started in the Initiator mode, the MPA implementation MUST send a valid MPA Request Frame. The MPA Request Frame MAY include ULP-supplied Private Data.

2. When MPA is started in the Responder mode, the MPA implementation MUST wait until an MPA Request Frame is received and validated before entering Full MPA/DDP Operation.

   If the MPA Request Frame is improperly formatted, the implementation MUST close the TCP connection and exit MPA.

   If the MPA Request Frame is properly formatted but the Private Data is not acceptable, the implementation SHOULD return an MPA Reply Frame with the Rejected Connection bit set to '1'; the MPA Reply Frame MAY include ULP-supplied Private Data; the implementation MUST exit MPA, leaving the TCP connection open. The ULP may close TCP or use the connection for other purposes.

   If the MPA Request Frame is properly formatted and the Private Data is acceptable, the implementation SHOULD return an MPA Reply Frame with the Rejected Connection bit set to '0'; the MPA Reply

Frame MAY include ULP-supplied Private Data; and the Responder SHOULD prepare to interpret any data received as FPDUs and pass any received ULPDUs to DDP.

Note: Since the receiver’s ability to deal with Markers is unknown until the Request and Reply Frames have been received, sending FPDUs before this occurs is not possible.

Note: The requirement to wait on a Request Frame before sending a Reply Frame is a design choice. It makes for a well-ordered sequence of events at each end, and avoids having to specify how to deal with situations where both ends start at the same time.

3. MPA Initiator mode implementations MUST receive and validate an MPA Reply Frame.

   If the MPA Reply Frame is improperly formatted, the implementation MUST close the TCP connection and exit MPA.

   If the MPA Reply Frame is properly formatted but is the Private Data is not acceptable, or if the Rejected Connection bit is set to ’1’, the implementation MUST exit MPA, leaving the TCP connection open. The ULP may close TCP or use the connection for other purposes.

   If the MPA Reply Frame is properly formatted and the Private Data is acceptable, and the Reject Connection bit is set to ’0’, the implementation SHOULD enter Full MPA/DDP Operation Phase; interpreting any received data as FPDUs and sending DDP ULPDUs as FPDUs.

4. MPA Responder mode implementations MUST receive and validate at least one FPDU before sending any FPDUs or Markers.

   Note: This requirement is present to allow the Initiator time to get its receiver into Full Operation before an FPDU arrives, avoiding potential race conditions at the Initiator. This was also subject to some debate in the work group before rough consensus was reached. Eliminating this requirement would allow faster startup in some types of applications. However, that would also make certain implementations (particularly "dual stack") much harder.

5. If a received "Key" does not match the expected value (see Section 7.1.1, MPA Request and Reply Frame Format) the TCP/DDP connection MUST be closed, and an error returned to the ULP.

6. The received Private Data fields may be used by Consumers at either end to further validate the connection and set up DDP or other ULP parameters. The Initiator ULP MAY close the TCP/MPA/DDP connection as a result of validating the Private Data fields. The Responder SHOULD return an MPA Reply Frame with the "Reject Connection" bit set to ’1’ if the validation of the Private Data is not acceptable to the ULP.

7. When the first FPDU is to be sent, then if Markers are enabled, the first octets sent are the special Marker 0x00000000, followed by the start of the FPDU (the FPDU’s ULPDU Length field). If Markers are not enabled, the first octets sent are the start of the FPDU (the FPDU’s ULPDU Length field).

8. MPA implementations MUST use the difference between the MPA Request Frame and the MPA Reply Frame to check for incorrect "Initiator/Initiator" startups. Implementations SHOULD put a timeout on waiting for the MPA Request Frame when started in Responder mode, to detect incorrect "Responder/Responder" startups.

9. MPA implementations MUST validate the PD_Length field. The buffer that receives the Private Data field MUST be large enough to receive that data; the amount of Private Data MUST not exceed the PD_Length or the application buffer. If any of the above fails, the startup frame MUST be considered improperly formatted.

10. MPA implementations SHOULD implement a reasonable timeout while waiting for the entire set of startup frames; this prevents certain denial-of-service attacks. ULPs SHOULD implement a reasonable timeout while waiting for FPDUs, ULPDUs, and application level messages to guard against application failures and certain denial-of-service attacks.

7.1.3. Example Delayed Startup Sequence

A variety of startup sequences are possible when using MPA on TCP. Following is an example of an MPA/DDP startup that occurs after TCP has been running for a while and has exchanged some amount of streaming data. This example does not use any Private Data (an example that does is shown later in Section 7.1.4.2, Example Immediate Startup Using Private Data), although it is perfectly legal to include the Private Data. Note that since the example does not use any Private Data, there are no ULP interactions shown between receiving "startup frames" and putting MPA into Full Operation.

```
         Initiator                                 Responder

  +---------------------------+
  |ULP streaming mode         |
  |  <Hello> request to       |
  |  transition to DDP/MPA    |           +---------------------------+
  |  mode (optional).         | --------> |ULP gets request;          |
  +---------------------------+           |  enables MPA Responder    |
                                          |  mode with last (optional)|
                                          |  streaming mode           |
                                          |  <Hello Ack> for MPA to   |
                                          |  send.                    |
  +---------------------------+           |MPA waits for incoming     |
  |ULP receives streaming     | <-------- |  <MPA Request Frame>.     |
  |  <Hello Ack>;             |           +---------------------------+
  |Enters MPA Initiator mode; |
  |MPA sends                  |
  |  <MPA Request Frame>;     |
  |MPA waits for incoming     |           +---------------------------+
  |  <MPA Reply Frame>.       | - - - - > |MPA receives               |
  +---------------------------+           |  <MPA Request Frame>.     |
                                          |Consumer binds DDP to MPA; |
                                          |MPA sends the              |
                                          |  <MPA Reply Frame>.       |
                                          |DDP/MPA enables FPDU       |
  +---------------------------+           |  decoding, but does not   |
  |MPA receives the           | < - - - - |  send any FPDUs.          |
  |  <MPA Reply Frame>        |           +---------------------------+
  |Consumer binds DDP to MPA; |
  |DDP/MPA begins Full        |
  |  Operation.               |
  |MPA sends first FPDU (as   |           +---------------------------+
  |  DDP ULPDUs become        | ========> |MPA receives first FPDU.   |
  |  available).              |           |MPA sends first FPDU (as   |
  +---------------------------+           |  DDP ULPDUs become        |
                                  <====== |  available).              |
                                          +---------------------------+
```

Figure 9: Example Delayed Startup Negotiation

An example Delayed Startup sequence is described below:

    * Active and passive sides start up a TCP connection in the usual fashion, probably using sockets APIs. They exchange some amount of streaming mode data. At some point, one side (the MPA Initiator) sends streaming mode data that effectively says "Hello, let’s go into MPA/DDP mode".

* When the remote side (the MPA Responder) gets this streaming mode message, the Consumer would send a last streaming mode message that effectively says "I acknowledge your Hello, and am now in MPA Responder mode". The exchange of these messages establishes the exact point in the TCP stream where MPA is enabled. The Responding Consumer enables MPA in the Responder mode and waits for the initial MPA startup message.

    * The Initiating Consumer would enable MPA startup in the Initiator mode which then sends the MPA Request Frame. It is assumed that no Private Data messages are needed for this example, although it is possible to do so. The Initiating MPA (and Consumer) would also wait for the MPA connection to be accepted.

* The Responding MPA would receive the initial MPA Request Frame and would inform the Consumer that this message arrived. The Consumer can then accept the MPA/DDP connection or close the TCP connection.

* To accept the connection request, the Responding Consumer would use an appropriate API to bind the TCP/MPA connections to a DDP endpoint, thus enabling MPA/DDP into Full Operation. In the process of going to Full Operation, MPA sends the MPA Reply Frame. MPA/DDP waits for the first incoming FPDU before sending any FPDUs.

* If the initial TCP data was not a properly formatted MPA Request Frame, MPA will close or reset the TCP connection immediately.

    * The Initiating MPA would receive the MPA Reply Frame and would report this message to the Consumer. The Consumer can then accept the MPA/DDP connection, or close or reset the TCP connection to abort the process.

    * On determining that the connection is acceptable, the Initiating Consumer would use an appropriate API to bind the TCP/MPA connections to a DDP endpoint thus enabling MPA/DDP into Full Operation. MPA/DDP would begin sending DDP messages as MPA FPDUs.

7.1.4. Use of Private Data

This section is advisory in nature, in that it suggests a method by which a ULP can deal with pre-DDP connection information exchange.

7.1.4.1. Motivation

Prior RDMA protocols have been developed that provide Private Data via out-of-band mechanisms. As a result, many applications now expect some form of Private Data to be available for application use prior to setting up the DDP/RDMA connection. Following are some examples of the use of Private Data.

An RDMA endpoint (referred to as a Queue Pair, or QP, in InfiniBand and the [VERBS-RDMA]) must be associated with a Protection Domain. No receive operations may be posted to the endpoint before it is associated with a Protection Domain. Indeed under both the InfiniBand and proposed RDMA/DDP verbs [VERBS-RDMA] an endpoint/QP is created within a Protection Domain.

There are some applications where the choice of Protection Domain is dependent upon the identity of the remote ULP client. For example, if a user session requires multiple connections, it is highly desirable for all of those connections to use a single Protection Domain. Note: Use of Protection Domains is further discussed in [RDMASEC].

InfiniBand, the DAT APIs [DAT-API], and the IT-API [IT-API] all provide for the active-side ULP to provide Private Data when requesting a connection. This data is passed to the ULP to allow it to determine whether to accept the connection, and if so with which endpoint (and implicitly which Protection Domain).

The Private Data can also be used to ensure that both ends of the connection have configured their RDMA endpoints compatibly on such matters as the RDMA Read capacity (see [RDMAP]). Further ULP-specific uses are also presumed, such as establishing the identity of the client.

Private Data is also allowed for when accepting the connection, to allow completion of any negotiation on RDMA resources and for other ULP reasons.

There are several potential ways to exchange this Private Data. For example, the InfiniBand specification includes a connection management protocol that allows a small amount of Private Data to be exchanged using datagrams before actually starting the RDMA connection.

This document allows for small amounts of Private Data to be exchanged as part of the MPA startup sequence. The actual Private Data fields are carried in the MPA Request Frame and the MPA Reply Frame.

If larger amounts of Private Data or more negotiation is necessary, TCP streaming mode messages may be exchanged prior to enabling MPA.

7.1.4.2. Example Immediate Startup Using Private Data

```
          Initiator                                 Responder

   +---------------------------+
   |TCP SYN sent.              |           +--------------------------+
   +---------------------------+ --------> |TCP gets SYN packet;      |
   +---------------------------+           |  sends SYN-Ack.          |
   |TCP gets SYN-Ack           | <-------- +--------------------------+
   |  sends Ack.               |
   +---------------------------+ --------> +--------------------------+
   +---------------------------+           |Consumer enables MPA      |
   |Consumer enables MPA       |           |Responder mode, waits for |
   |Initiator mode with        |           |  <MPA Request frame>.    |
   |Private Data; MPA sends    |           +--------------------------+
   |  <MPA Request Frame>;     |
   |MPA waits for incoming     |           +--------------------------+
   |  <MPA Reply Frame>.       | - - - - > |MPA receives              |
   +---------------------------+           |  <MPA Request Frame>.    |
                                           |Consumer examines Private |
                                           |Data, provides MPA with   |
                                           |return Private Data,      |
                                           |binds DDP to MPA, and     |
                                           |enables MPA to send an    |
                                           |  <MPA Reply Frame>.      |
                                           |DDP/MPA enables FPDU      |
   +---------------------------+           |decoding, but does not    |
   |MPA receives the           | < - - - - |send any FPDUs.           |
   |  <MPA Reply Frame>.       |           +--------------------------+
   |Consumer examines Private  |
   |Data, binds DDP to MPA,    |
   |and enables DDP/MPA to     |
   |begin Full Operation.      |
   |MPA sends first FPDU (as   |           +--------------------------+
   |DDP ULPDUs become          | ========> |MPA receives first FPDU.  |
   |available).                |           |MPA sends first FPDU (as  |
   +---------------------------+           |DDP ULPDUs become         |
                                   <====== |available).               |
                                           +--------------------------+
```

Figure 10: Example Immediate Startup Negotiation

Note: The exact order of when MPA is started in the TCP connection sequence is implementation dependent; the above diagram shows one possible sequence. Also, the Initiator "Ack" to the Responder's "SYN-Ack" may be combined into the same TCP segment containing the MPA Request Frame (as is allowed by TCP RFCs).

The example immediate startup sequence is described below:

* The passive side (Responding Consumer) would listen on the TCP destination port, to indicate its readiness to accept a connection.

    * The active side (Initiating Consumer) would request a connection from a TCP endpoint (that expected to upgrade to MPA/DDP/RDMA and expected the Private Data) to a destination address and port.

    * The Initiating Consumer would initiate a TCP connection to the destination port. Acceptance/rejection of the connection would proceed as per normal TCP connection establishment.

* The passive side (Responding Consumer) would receive the TCP connection request as usual allowing normal TCP gatekeepers, such as INETD and TCPserver, to exercise their normal safeguard/logging functions. On acceptance of the TCP connection, the Responding Consumer would enable MPA in the Responder mode and wait for the initial MPA startup message.

    * The Initiating Consumer would enable MPA startup in the Initiator mode to send an initial MPA Request Frame with its included Private Data message to send. The Initiating MPA (and Consumer) would also wait for the MPA connection to be accepted, and any returned Private Data.

* The Responding MPA would receive the initial MPA Request Frame with the Private Data message and would pass the Private Data through to the Consumer. The Consumer can then accept the MPA/DDP connection, close the TCP connection, or reject the MPA connection with a return message.

* To accept the connection request, the Responding Consumer would use an appropriate API to bind the TCP/MPA connections to a DDP endpoint, thus enabling MPA/DDP into Full Operation. In the process of going to Full Operation, MPA sends the MPA Reply Frame, which includes the Consumer-supplied Private Data containing any appropriate Consumer response. MPA/DDP waits for the first incoming FPDU before sending any FPDUs.

* If the initial TCP data was not a properly formatted MPA Request Frame, MPA will close or reset the TCP connection immediately.

- To reject the MPA connection request, the Responding Consumer would send an MPA Reply Frame with any ULP-supplied Private Data (with reason for rejection), with the "Rejected Connection" bit set to ’1’, and may close the TCP connection.

  - The Initiating MPA would receive the MPA Reply Frame with the Private Data message and would report this message to the Consumer, including the supplied Private Data.

    If the "Rejected Connection" bit is set to a ’1’, MPA will close the TCP connection and exit.

    If the "Rejected Connection" bit is set to a ’0’, and on determining from the MPA Reply Frame Private Data that the connection is acceptable, the Initiating Consumer would use an appropriate API to bind the TCP/MPA connections to a DDP endpoint thus enabling MPA/DDP into Full Operation. MPA/DDP would begin sending DDP messages as MPA FPDUs.

7.1.5. "Dual Stack" Implementations

MPA/DDP implementations are commonly expected to be implemented as part of a "dual stack" architecture. One stack is the traditional TCP stack, usually with a sockets interface API (Application Programming Interface). The second stack is the MPA/DDP stack with its own API, and potentially separate code or hardware to deal with the MPA/DDP data. Of course, implementations may vary, so the following comments are of an advisory nature only.

The use of the two stacks offers advantages:

> TCP connection setup is usually done with the TCP stack. This allows use of the usual naming and addressing mechanisms. It also means that any mechanisms used to "harden" the connection setup against security threats are also used when starting MPA/DDP.
>
> Some applications may have been originally designed for TCP, but are "enhanced" to utilize MPA/DDP after a negotiation reveals the capability to do so. The negotiation process takes place in TCP’s streaming mode, using the usual TCP APIs.
>
> Some new applications, designed for RDMA or DDP, still need to exchange some data prior to starting MPA/DDP. This exchange can be of arbitrary length or complexity, but often consists of only a small amount of Private Data, perhaps only a single message. Using the TCP streaming mode for this exchange allows this to be done using well-understood methods.

The main disadvantage of using two stacks is the conversion of an active TCP connection between them. This process must be done with care to prevent loss of data.

To avoid some of the problems when using a "dual stack" architecture, the following additional restrictions may be required by the implementation:

1. Enabling the DDP/MPA stack SHOULD be done only when no incoming stream data is expected. This is typically managed by the ULP protocol. When following the recommended startup sequence, the Responder side enters DDP/MPA mode, sends the last streaming mode data, and then waits for the MPA Request Frame. No additional streaming mode data is expected. The Initiator side ULP receives the last streaming mode data, and then enters DDP/MPA mode. Again, no additional streaming mode data is expected.

2. The DDP/MPA MAY provide the ability to send a "last streaming message" as part of its Responder DDP/MPA enable function. This allows the DDP/MPA stack to more easily manage the conversion to DDP/MPA mode (and avoid problems with a very fast return of the MPA Request Frame from the Initiator side).

Note: Regardless of the "stack" architecture used, TCP’s rules MUST be followed. For example, if network data is lost, re-segmented, or re-ordered, TCP MUST recover appropriately even when this occurs while switching stacks.

7.2. Normal Connection Teardown

Each half connection of MPA terminates when DDP closes the corresponding TCP half connection.

A mechanism SHOULD be provided by MPA to DDP for DDP to be made aware that a graceful close of the TCP connection has been received by the TCP (e.g., FIN is received).

8. Error Semantics

The following errors MUST be detected by MPA and the codes SHOULD be provided to DDP or other Consumer:

Code Error

1 TCP connection closed, terminated, or lost. This includes lost by timeout, too many retries, RST received, or FIN received.

2 Received MPA CRC does not match the calculated value for the FPDU.

3 In the event that the CRC is valid, received MPA Marker (if enabled) and ULPDU Length fields do not agree on the start of an FPDU. If the FPDU start determined from previous ULPDU Length fields does not match with the MPA Marker position, MPA SHOULD deliver an error to DDP. It may not be possible to make this check as a segment arrives, but the check SHOULD be made when a gap creating an out-of-order sequence is closed and any time a Marker points to an already identified FPDU. It is OPTIONAL for a receiver to check each Marker, if multiple Markers are present in an FPDU, or if the segment is received in order.

4 Invalid MPA Request Frame or MPA Response Frame received. In this case, the TCP connection MUST be immediately closed. DDP and other ULPs should treat this similar to code 1, above.

When conditions 2 or 3 above are detected, an optimized MPA/TCP implementation MAY choose to silently drop the TCP segment rather than reporting the error to DDP. In this case, the sending TCP will retry the segment, usually correcting the error, unless the problem was at the source. In that case, the source will usually exceed the number of retries and terminate the connection.

Once MPA delivers an error of any type, it MUST NOT pass or deliver any additional FPDUs on that half connection.

For Error codes 2 and 3, MPA MUST NOT close the TCP connection following a reported error. Closing the connection is the responsibility of DDP’s ULP.

> Note that since MPA will not Deliver any FPDUs on a half connection following an error detected on the receive side of that connection, DDP’s ULP is expected to tear down the connection. This may not occur until after one or more last messages are transmitted on the opposite half connection. This allows a diagnostic error message to be sent.

9. Security Considerations

This section discusses the security considerations for MPA.

9.1. Protocol-Specific Security Considerations

The vulnerabilities of MPA to third-party attacks are no greater than any other protocol running over TCP. A third party, by sending packets into the network that are delivered to an MPA receiver, could launch a variety of attacks that take advantage of how MPA operates. For example, a third party could send random packets that are valid for TCP, but contain no FPDU headers. An MPA receiver reports an error to DDP when any packet arrives that cannot be validated as an FPDU when properly located on an FPDU boundary. A third party could also send packets that are valid for TCP, MPA, and DDP, but do not target valid buffers. These types of attacks ultimately result in loss of connection and thus become a type of DOS (Denial Of Service) attack. Communication security mechanisms such as IPsec [RFC2401, RFC4301] may be used to prevent such attacks.

Independent of how MPA operates, a third party could use ICMP messages to reduce the path MTU to such a small size that performance would likewise be severely impacted. Range checking on path MTU sizes in ICMP packets may be used to prevent such attacks.

[RDMAP] and [DDP] are used to control, read, and write data buffers over IP networks. Therefore, the control and the data packets of these protocols are vulnerable to the spoofing, tampering, and information disclosure attacks listed below. In addition, connection to/from an unauthorized or unauthenticated endpoint is a potential problem with most applications using RDMA, DDP, and MPA.

9.1.1. Spoofing

Spoofing attacks can be launched by the Remote Peer or by a network based attacker. A network-based spoofing attack applies to all Remote Peers. Because the MPA Stream requires a TCP Stream in the ESTABLISHED state, certain types of traditional forms of wire attacks do not apply -- an end-to-end handshake must have occurred to establish the MPA Stream. So, the only form of spoofing that applies is one when a remote node can both send and receive packets. Yet even with this limitation the Stream is still exposed to the following spoofing attacks.

9.1.1.1. Impersonation

A network-based attacker can impersonate a legal MPA/DDP/RDMAP peer (by spoofing a legal IP address) and establish an MPA/DDP/RDMAP Stream with the victim. End-to-end authentication (i.e., IPsec or ULP authentication) provides protection against this attack.

9.1.1.2. Stream Hijacking

Stream hijacking happens when a network-based attacker follows the Stream establishment phase, and waits until the authentication phase (if such a phase exists) is completed successfully. He can then spoof the IP address and redirect the Stream from the victim to its own machine. For example, an attacker can wait until an iSCSI authentication is completed successfully, and hijack the iSCSI Stream.

The best protection against this form of attack is end-to-end integrity protection and authentication, such as IPsec, to prevent spoofing. Another option is to provide physical security. Discussion of physical security is out of scope for this document.

9.1.1.3. Man-in-the-Middle Attack

If a network-based attacker has the ability to delete, inject, replay, or modify packets that will still be accepted by MPA (e.g., TCP sequence number is correct, FPDU is valid, etc.), then the Stream can be exposed to a man-in-the-middle attack. The attacker could potentially use the services of [DDP] and [RDMAP] to read the contents of the associated Data Buffer, to modify the contents of the associated Data Buffer, or to disable further access to the buffer. Other attacks on the connection setup sequence and even on TCP can be used to cause denial of service. The only countermeasure for this form of attack is to either secure the MPA/DDP/RDMAP Stream (i.e., integrity protect) or attempt to provide physical security to prevent man-in-the-middle type attacks.

The best protection against this form of attack is end-to-end integrity protection and authentication, such as IPsec, to prevent spoofing or tampering. If Stream or session level authentication and integrity protection are not used, then a man-in-the-middle attack can occur, enabling spoofing and tampering.

Another approach is to restrict access to only the local subnet/link and provide some mechanism to limit access, such as physical security or 802.1.x. This model is an extremely limited deployment scenario and will not be further examined here.

9.1.2. Eavesdropping

Generally speaking, Stream confidentiality protects against eavesdropping. Stream and/or session authentication and integrity protection are a counter measurement against various spoofing and tampering attacks. The effectiveness of authentication and integrity against a specific attack depend on whether the authentication is machine-level authentication (as the one provided by IPsec) or ULP authentication.

9.2. Introduction to Security Options

The following security services can be applied to an MPA/DDP/RDMAP Stream:

1. Session confidentiality - protects against eavesdropping.

2. Per-packet data source authentication - protects against the following spoofing attacks: network-based impersonation, Stream hijacking, and man in the middle.

3. Per-packet integrity - protects against tampering done by network-based modification of FPDUs (indirectly affecting buffer content through DDP services).

4. Packet sequencing - protects against replay attacks, which is a special case of the above tampering attack.

If an MPA/DDP/RDMAP Stream may be subject to impersonation attacks, or Stream hijacking attacks, it is recommended that the Stream be authenticated, integrity protected, and protected from replay attacks. It may use confidentiality protection to protect from eavesdropping (in case the MPA/DDP/RDMAP Stream traverses a public network).

IPsec is capable of providing the above security services for IP and TCP traffic.

ULP protocols may be able to provide part of the above security services. See [NFSv4CHAN] for additional information on a promising approach called "channel binding". From [NFSv4CHAN]:

> "The concept of channel bindings allows applications to prove that the end-points of two secure channels at different network layers are the same by binding authentication at one channel to the session protection at the other channel. The use of channel

bindings allows applications to delegate session protection to lower layers, which may significantly improve performance for some applications."

9.3. Using IPsec with MPA

IPsec can be used to protect against the packet injection attacks outlined above. Because IPsec is designed to secure individual IP packets, MPA can run above IPsec without change. IPsec packets are processed (e.g., integrity checked and decrypted) in the order they are received, and an MPA receiver will process the decrypted FPDUs contained in these packets in the same manner as FPDUs contained in unsecured IP packets.

MPA implementations MUST implement IPsec as described in Section 9.4 below. The use of IPsec is up to ULPs and administrators.

9.4. Requirements for IPsec Encapsulation of MPA/DDP

The IP Storage working group has spent significant time and effort to define the normative IPsec requirements for IP storage [RFC3723]. Portions of that specification are applicable to a wide variety of protocols, including the RDDP protocol suite. In order not to replicate this effort, an MPA on TCP implementation MUST follow the requirements defined in RFC 3723, Sections 2.3 and 5, including the associated normative references for those sections.

Additionally, since IPsec acceleration hardware may only be able to handle a limited number of active Internet Key Exchange Protocol (IKE) Phase 2 security associations (SAs), Phase 2 delete messages MAY be sent for idle SAs, as a means of keeping the number of active Phase 2 SAs to a minimum. The receipt of an IKE Phase 2 delete message MUST NOT be interpreted as a reason for tearing down a DDP/RDMA Stream. Rather, it is preferable to leave the Stream up, and if additional traffic is sent on it, to bring up another IKE Phase 2 SA to protect it. This avoids the potential for continually bringing Streams up and down.

The IPsec requirements for RDDP are based on the version of IPsec specified in RFC 2401 [RFC2401] and related RFCs, as profiled by RFC 3723 [RFC3723], despite the existence of a newer version of IPsec specified in RFC 4301 [RFC4301] and related RFCs. One of the important early applications of the RDDP protocols is their use with iSCSI [iSER]; RDDP’s IPsec requirements follow those of IPsec in order to facilitate that usage by allowing a common profile of IPsec to be used with iSCSI and the RDDP protocols. In the future, RFC

3723 may be updated to the newer version of IPsec; the IPsec security requirements of any such update should apply uniformly to iSCSI and the RDDP protocols.

Note that there are serious security issues if IPsec is not implemented end-to-end. For example, if IPsec is implemented as a tunnel in the middle of the network, any hosts between the peer and the IPsec tunneling device can freely attack the unprotected Stream.

10. IANA Considerations

No IANA actions are required by this document.

If a well-known port is chosen as the mechanism to identify a DDP on MPA on TCP, the well-known port must be registered with IANA. Because the use of the port is DDP specific, registration of the port with IANA is left to DDP.

Appendix A. Optimized MPA-Aware TCP Implementations

This appendix is for information only and is NOT part of the standard.

This appendix covers some Optimized MPA-aware TCP implementation guidance to implementers. It is intended for those implementations that want to send/receive as much traffic as possible in an aligned and zero-copy fashion.

```
                +-----------------------------------+
                | +-----------+ +-----------------+ |
                | | Optimized | | Other Protocols | |
                | |  MPA/TCP  | +-----------------+ |
                | +-----------+        ||           |
                |         \\     --- socket API --- |
                |          \\          ||           |
                |           \\      +-----+         |
                |            \\     | TCP |         |
                |             \\    +-----+         |
                |              \\    //             |
                |             +-------+             |
                |             |  IP   |             |
                |             +-------+             |
                +-----------------------------------+
```

Figure 11: Optimized MPA/TCP Implementation

The diagram above shows a block diagram of a potential implementation. The network sub-system in the diagram can support traditional sockets-based connections using the normal API as shown on the right side of the diagram. Connections for DDP/MPA/TCP are run using the facilities shown on the left side of the diagram.

The DDP/MPA/TCP connections can be started using the facilities shown on the left side using some suitable API, or they can be initiated using the facilities shown on the right side and transitioned to the left side at the point in the connection setup where MPA goes to "Full MPA/DDP Operation Phase" as described in Section 7.1.2.

The optimized MPA/TCP implementations (left side of diagram and described below) are only applicable to MPA. All other TCP applications continue to use the standard TCP stacks and interfaces shown in the right side of the diagram.

A.1. Optimized MPA/TCP Transmitters

The various TCP RFCs allow considerable choice in segmenting a TCP stream. In order to optimize FPDU recovery at the MPA receiver, an optimized MPA/TCP implementation uses additional segmentation rules.

To provide optimum performance, an optimized MPA/TCP transmit side implementation should be enabled to:

* With an EMSS large enough to contain the FPDU(s), segment the outgoing TCP stream such that the first octet of every TCP segment begins with an FPDU. Multiple FPDUs may be packed into a single TCP segment as long as they are entirely contained in the TCP segment.

* Report the current EMSS from the TCP to the MPA transmit layer.

There are exceptions to the above rule. Once an ULPDU is provided to MPA, the MPA/TCP sender transmits it or fails the connection; it cannot be repudiated. As a result, during changes in MTU and EMSS, or when TCP’s Receive Window size (RWIN) becomes too small, it may be necessary to send FPDUs that do not conform to the segmentation rule above.

A possible, but less desirable, alternative is to use IP fragmentation on accepted FPDUs to deal with MTU reductions or extremely small EMSS.

Even when alignment with TCP segments is lost, the sender still formats the FPDU according to FPDU format as shown in Figure 2.

On a retransmission, TCP does not necessarily preserve original TCP segmentation boundaries. This can lead to the loss of FPDU Alignment and containment within a TCP segment during TCP retransmissions. An optimized MPA/TCP sender should try to preserve original TCP segmentation boundaries on a retransmission.

A.2. Effects of Optimized MPA/TCP Segmentation

Optimized MPA/TCP senders will fill TCP segments to the EMSS with a single FPDU when a DDP message is large enough. Since the DDP message may not exactly fit into TCP segments, a "message tail" often occurs that results in an FPDU that is smaller than a single TCP segment. Additionally, some DDP messages may be considerably shorter than the EMSS. If a small FPDU is sent in a single TCP segment, the result is a "short" TCP segment.

Applications expected to see strong advantages from Direct Data Placement include transaction-based applications and throughput applications. Request/response protocols typically send one FPDU per TCP segment and then wait for a response. Under these conditions, these "short" TCP segments are an appropriate and expected effect of the segmentation.

Another possibility is that the application might be sending multiple messages (FPDUs) to the same endpoint before waiting for a response. In this case, the segmentation policy would tend to reduce the available connection bandwidth by under-filling the TCP segments.

Standard TCP implementations often utilize the Nagle [RFC896] algorithm to ensure that segments are filled to the EMSS whenever the round-trip latency is large enough that the source stream can fully fill segments before ACKs arrive. The algorithm does this by delaying the transmission of TCP segments until a ULP can fill a segment, or until an ACK arrives from the far side. The algorithm thus allows for smaller segments when latencies are shorter to keep the ULP’s end-to-end latency to reasonable levels.

The Nagle algorithm is not mandatory to use [RFC1122].

When used with optimized MPA/TCP stacks, Nagle and similar algorithms can result in the "packing" of multiple FPDUs into TCP segments.

If a "message tail", small DDP messages, or the start of a larger DDP message are available, MPA may pack multiple FPDUs into TCP segments. When this is done, the TCP segments can be more fully utilized, but, due to the size constraints of FPDUs, segments may not be filled to the EMSS. A dynamic MULPDU that informs DDP of the size of the remaining TCP segment space makes filling the TCP segment more effective.

> Note that MPA receivers do more processing of a TCP segment that contains multiple FPDUs; this may affect the performance of some receiver implementations.

It is up to the ULP to decide if Nagle is useful with DDP/MPA. Note that many of the applications expected to take advantage of MPA/DDP prefer to avoid the extra delays caused by Nagle. In such scenarios, it is anticipated there will be minimal opportunity for packing at the transmitter and receivers may choose to optimize their performance for this anticipated behavior.

Therefore, the application is expected to set TCP parameters such that it can trade off latency and wire efficiency. Implementations should provide a connection option that disables Nagle for MPA/TCP similar to the way the TCP_NODELAY socket option is provided for a traditional sockets interface.

When latency is not critical, application is expected to leave Nagle enabled. In this case, the TCP implementation may pack any available FPDUs into TCP segments so that the segments are filled to the EMSS. If the amount of data available is not enough to fill the TCP segment when it is prepared for transmission, TCP can send the segment partly filled, or use the Nagle algorithm to wait for the ULP to post more data.

A.3. Optimized MPA/TCP Receivers

When an MPA receive implementation and the MPA-aware receive side TCP implementation support handling out-of-order ULPDUs, the TCP receive implementation performs the following functions:

1) The implementation passes incoming TCP segments to MPA as soon as they have been received and validated, even if not received in order. The TCP layer commits to keeping each segment before it can be passed to the MPA. This means that the segment must have passed the TCP, IP, and lower layer data integrity validation (i.e., checksum), must be in the receive window, must be part of the same epoch (if timestamps are used to verify this), and must have passed any other checks required by TCP RFCs.

   This is not to imply that the data must be completely ordered before use. An implementation can accept out-of-order segments, SACK them [RFC2018], and pass them to MPA immediately, before the reception of the segments needed to fill in the gaps. MPA expects to utilize these segments when they are complete FPDUs or can be combined into complete FPDUs to allow the passing of ULPDUs to DDP when they arrive, independent of ordering. DDP uses the passed ULPDU to "place" the DDP segments (see [DDP] for more details).

   Since MPA performs a CRC calculation and other checks on received FPDUs, the MPA/TCP implementation ensures that any TCP segments that duplicate data already received and processed (as can happen during TCP retries) do not overwrite already received and processed FPDUs. This avoids the possibility that duplicate data may corrupt already validated FPDUs.

2) The implementation provides a mechanism to indicate the ordering of TCP segments as the sender transmitted them. One possible mechanism might be attaching the TCP sequence number to each segment.

3) The implementation also provides a mechanism to indicate when a given TCP segment (and the prior TCP stream) is complete. One possible mechanism might be to utilize the leading (left) edge of the TCP Receive Window.

   MPA uses the ordering and completion indications to inform DDP when a ULPDU is complete; MPA Delivers the FPDU to DDP. DDP uses the indications to "deliver" its messages to the DDP consumer (see [DDP] for more details).

   DDP on MPA utilizes the above two mechanisms to establish the Delivery semantics that DDP’s consumers agree to. These semantics are described fully in [DDP]. These include requirements on DDP’s consumer to respect ownership of buffers prior to the time that DDP delivers them to the Consumer.

The use of SACK [RFC2018] significantly improves network utilization and performance and is therefore recommended. When combined with the out-of-order passing of segments to MPA and DDP, significant buffering and copying of received data can be avoided.

A.4. Re-Segmenting Middleboxes and Non-Optimized MPA/TCP Senders

Since MPA senders often start FPDUs on TCP segment boundaries, a receiving optimized MPA/TCP implementation may be able to optimize the reception of data in various ways.

However, MPA receivers MUST NOT depend on FPDU Alignment on TCP segment boundaries.

Some MPA senders may be unable to conform to the sender requirements because their implementation of TCP is not designed with MPA in mind. Even for optimized MPA/TCP senders, the network may contain "middleboxes" which modify the TCP stream by changing the segmentation. This is generally interoperable with TCP and its users and MPA must be no exception.

The presence of Markers in MPA (when enabled) allows an optimized MPA/TCP receiver to recover the FPDUs despite these obstacles, although it may be necessary to utilize additional buffering at the receiver to do so.

Some of the cases that a receiver may have to contend with are listed below as a reminder to the implementer:

- A single aligned and complete FPDU, either in order or out of order: This can be passed to DDP as soon as validated, and Delivered when ordering is established.

- Multiple FPDUs in a TCP segment, aligned and fully contained, either in order or out of order: These can be passed to DDP as soon as validated, and Delivered when ordering is established.

- Incomplete FPDU: The receiver should buffer until the remainder of the FPDU arrives. If the remainder of the FPDU is already available, this can be passed to DDP as soon as validated, and Delivered when ordering is established.

- Unaligned FPDU start: The partial FPDU must be combined with its preceding portion(s). If the preceding parts are already available, and the whole FPDU is present, this can be passed to DDP as soon as validated, and Delivered when ordering is established. If the whole FPDU is not available, the receiver should buffer until the remainder of the FPDU arrives.

- Combinations of unaligned or incomplete FPDUs (and potentially other complete FPDUs) in the same TCP segment: If any FPDU is present in its entirety, or can be completed with portions already available, it can be passed to DDP as soon as validated, and Delivered when ordering is established.

A.5. Receiver Implementation

Transport & Network Layer Reassembly Buffers:

The use of reassembly buffers (either TCP reassembly buffers or IP fragmentation reassembly buffers) is implementation dependent. When MPA is enabled, reassembly buffers are needed if out-of-order packets arrive and Markers are not enabled. Buffers are also needed if FPDU alignment is lost or if IP fragmentation occurs. This is because the incoming out-of-order segment may not contain enough information for MPA to process all of the FPDU. For cases where a re-segmenting middlebox is present, or where the TCP sender is not optimized, the presence of Markers significantly reduces the amount of buffering needed.

Recovery from IP fragmentation is transparent to the MPA Consumers.

A.5.1 Network Layer Reassembly Buffers

The MPA/TCP implementation should set the IP Don’t Fragment bit at the IP layer. Thus, upon a path MTU change, intermediate devices drop the IP datagram if it is too large and reply with an ICMP message that tells the source TCP that the path MTU has changed. This causes TCP to emit segments conformant with the new path MTU size. Thus, IP fragments under most conditions should never occur at the receiver. But it is possible.

There are several options for implementation of network layer reassembly buffers:

1. drop any IP fragments, and reply with an ICMP message according to [RFC792] (fragmentation needed and DF set) to tell the Remote Peer to resize its TCP segment.

2. support an IP reassembly buffer, but have it of limited size (possibly the same size as the local link’s MTU). The end node would normally never Advertise a path MTU larger than the local link MTU. It is recommended that a dropped IP fragment cause an ICMP message to be generated according to RFC 792.

3. multiple IP reassembly buffers, of effectively unlimited size.

4. support an IP reassembly buffer for the largest IP datagram (64 KB).

5. support for a large IP reassembly buffer that could span multiple IP datagrams.

An implementation should support at least 2 or 3 above, to avoid dropping packets that have traversed the entire fabric.

There is no end-to-end ACK for IP reassembly buffers, so there is no flow control on the buffer. The only end-to-end ACK is a TCP ACK, which can only occur when a complete IP datagram is delivered to TCP. Because of this, under worst case, pathological scenarios, the largest IP reassembly buffer is the TCP receive window (to buffer multiple IP datagrams that have all been fragmented).

Note that if the Remote Peer does not implement re-segmentation of the data stream upon receiving the ICMP reply updating the path MTU, it is possible to halt forward progress because the opposite peer would continue to retransmit using a transport segment size that is too large. This deadlock scenario is no different than if the fabric MTU (not last-hop MTU) was reduced after connection setup, and the remote node’s behavior is not compliant with [RFC1122].

A.5.2 TCP Reassembly Buffers

A TCP reassembly buffer is also needed. TCP reassembly buffers are needed if FPDU Alignment is lost when using TCP with MPA or when the MPA FPDU spans multiple TCP segments. Buffers are also needed if Markers are disabled and out-of-order packets arrive.

Since lost FPDU Alignment often means that FPDUs are incomplete, an MPA on TCP implementation must have a reassembly buffer large enough to recover an FPDU that is less than or equal to the MTU of the locally attached link (this should be the largest possible Advertised TCP path MTU). If the MTU is smaller than 140 octets, a buffer of at least 140 octets long is needed to support the minimum FPDU size. The 140 octets allow for the minimum MULPDU of 128, 2 octets of pad, 2 of ULPDU_Length, 4 of CRC, and space for a possible Marker. As usual, additional buffering is likely to provide better performance.

Note that if the TCP segments were not stored, it would be possible to deadlock the MPA algorithm. If the path MTU is reduced, FPDU Alignment requires the source TCP to re-segment the data stream to the new path MTU. The source MPA will detect this condition and reduce the MPA segment size, but any FPDUs already posted to the source TCP will be re-segmented and lose FPDU Alignment. If the destination does not support a TCP reassembly buffer, these segments can never be successfully transmitted and the protocol deadlocks.

When a complete FPDU is received, processing continues normally.

Appendix B. Analysis of MPA over TCP Operations

This appendix is for information only and is NOT part of the standard.

This appendix is an analysis of MPA on TCP and why it is useful to integrate MPA with TCP (with modifications to typical TCP implementations) to reduce overall system buffering and overhead.

One of MPA's high-level goals is to provide enough information, when combined with the Direct Data Placement Protocol [DDP], to enable out-of-order placement of DDP payload into the final Upper Layer Protocol (ULP) Buffer. Note that DDP separates the act of placing data into a ULP Buffer from that of notifying the ULP that the ULP Buffer is available for use. In DDP terminology, the former is defined as "Placement", and the later is defined as "Delivery". MPA supports in-order Delivery of the data to the ULP, including support for Direct Data Placement in the final ULP Buffer location when TCP segments arrive out of order. Effectively, the goal is to use the pre-posted ULP Buffers as the TCP receive buffer, where the reassembly of the ULP Protocol Data Unit (PDU) by TCP (with MPA and DDP) is done in place, in the ULP Buffer, with no data copies.

This appendix walks through the advantages and disadvantages of the TCP sender modifications proposed by MPA:

1) that MPA prefers that the TCP sender to do Header Alignment, where a TCP segment should begin with an MPA Framing Protocol Data Unit (FPDU) (if there is payload present).

2) that there be an integral number of FPDUs in a TCP segment (under conditions where the path MTU is not changing).

This appendix concludes that the scaling advantages of FPDU Alignment are strong, based primarily on fairly drastic TCP receive buffer reduction requirements and simplified receive handling. The analysis also shows that there is little effect to TCP wire behavior.

B.1. Assumptions

B.1.1 MPA Is Layered beneath DDP

MPA is an adaptation layer between DDP and TCP. DDP requires preservation of DDP segment boundaries and a CRC32c digest covering the DDP header and data. MPA adds these features to the TCP stream so that DDP over TCP has the same basic properties as DDP over SCTP.

B.1.2. MPA Preserves DDP Message Framing

MPA was designed as a framing layer specifically for DDP and was not intended as a general-purpose framing layer for any other ULP using TCP.

A framing layer allows ULPs using it to receive indications from the transport layer only when complete ULPDUs are present. As a framing layer, MPA is not aware of the content of the DDP PDU, only that it has received and, if necessary, reassembled a complete PDU for Delivery to the DDP.

B.1.3. The Size of the ULPDU Passed to MPA Is Less Than EMSS under Normal Conditions

To make reception of a complete DDP PDU on every received segment possible, DDP passes to MPA a PDU that is no larger than the EMSS of the underlying fabric. Each FPDU that MPA creates contains sufficient information for the receiver to directly place the ULP payload in the correct location in the correct receive buffer.

Edge cases when this condition does not occur are dealt with, but do not need to be on the fast path.

B.1.4. Out-of-Order Placement but NO Out-of-Order Delivery

DDP receives complete DDP PDUs from MPA. Each DDP PDU contains the information necessary to place its ULP payload directly in the correct location in host memory.

Because each DDP segment is self-describing, it is possible for DDP segments received out of order to have their ULP payload placed immediately in the ULP receive buffer.

Data delivery to the ULP is guaranteed to be in the order the data was sent. DDP only indicates data delivery to the ULP after TCP has acknowledged the complete byte stream.

B.2. The Value of FPDU Alignment

Significant receiver optimizations can be achieved when Header Alignment and complete FPDUs are the common case. The optimizations allow utilizing significantly fewer buffers on the receiver and less computation per FPDU. The net effect is the ability to build a "flow-through" receiver that enables TCP-based solutions to scale to 10G and beyond in an economical way. The optimizations are especially relevant to hardware implementations of receivers that process multiple protocol layers -- Data Link Layer (e.g., Ethernet), Network and Transport Layer (e.g., TCP/IP), and even some ULP on top of TCP (e.g., MPA/DDP). As network speed increases, there is an increasing desire to use a hardware-based receiver in order to achieve an efficient high performance solution.

A TCP receiver, under worst-case conditions, has to allocate buffers (BufferSizeTCP) whose capacities are a function of the bandwidth-delay product. Thus:

```
    BufferSizeTCP = K * bandwidth [octets/second] * Delay [seconds].
```

Where bandwidth is the end-to-end bandwidth of the connection, delay is the round-trip delay of the connection, and K is an implementation-dependent constant.

Thus, BufferSizeTCP scales with the end-to-end bandwidth (10x more buffers for a 10x increase in end-to-end bandwidth). As this buffering approach may scale poorly for hardware or software implementations alike, several approaches allow reduction in the amount of buffering required for high-speed TCP communication.

The MPA/DDP approach is to enable the ULP’s Buffer to be used as the TCP receive buffer. If the application pre-posts a sufficient amount of buffering, and each TCP segment has sufficient information to place the payload into the right application buffer, when an out-of-order TCP segment arrives it could potentially be placed directly in the ULP Buffer. However, placement can only be done when a complete FPDU with the placement information is available to the receiver, and the FPDU contents contain enough information to place the data into the correct ULP Buffer (e.g., there is a DDP header available).

For the case when the FPDU is not aligned with the TCP segment, it may take, on average, 2 TCP segments to assemble one FPDU. Therefore, the receiver has to allocate BufferSizeNAF (Buffer Size, Non-Aligned FPDU) octets:

```
    BufferSizeNAF = K1* EMSS * number_of_connections + K2 * EMSS
```

Where K1 and K2 are implementation-dependent constants and EMSS is the effective maximum segment size.

For example, a 1 GB/sec link with 10,000 connections and an EMSS of 1500 B would require 15 MB of memory. Often the number of connections used scales with the network speed, aggravating the situation for higher speeds.

FPDU Alignment would allow the receiver to allocate BufferSizeAF (Buffer Size, Aligned FPDU) octets:

```
    BufferSizeAF = K2 * EMSS
```

for the same conditions. An FPDU Aligned receiver may require memory in the range of ~100s of KB -- which is feasible for an on-chip memory and enables a "flow-through" design, in which the data flows through the network interface card (NIC) and is placed directly in the destination buffer. Assuming most of the connections support FPDU Alignment, the receiver buffers no longer scale with number of connections.

Additional optimizations can be achieved in a balanced I/O sub-system -- where the system interface of the network controller provides ample bandwidth as compared with the network bandwidth. For almost twenty years this has been the case and the trend is expected to continue. While Ethernet speeds have scaled by 1000 (from 10 megabit/sec to 10 gigabit/sec), I/O bus bandwidth of volume CPU architectures has scaled from ~2 MB/sec to ~2 GB/sec (PC-XT bus to PCI-X DDR). Under these conditions, the FPDU Alignment approach allows BufferSizeAF to be indifferent to network speed. It is primarily a function of the local processing time for a given frame.

Thus, when the FPDU Alignment approach is used, receive buffering is expected to scale gracefully (i.e., less than linear scaling) as network speed is increased.

B.2.1. Impact of Lack of FPDU Alignment on the Receiver Computational Load and Complexity

The receiver must perform IP and TCP processing, and then perform FPDU CRC checks, before it can trust the FPDU header placement information. For simplicity of the description, the assumption is that an FPDU is carried in no more than 2 TCP segments. In reality, with no FPDU Alignment, an FPDU can be carried by more than 2 TCP segments (e.g., if the path MTU was reduced).

```
   ----++-----------------------------++-----------------------++-----
   +---||---------------+    +--------||--------+   +----------||----+
   |   TCP Seg X-1      |    |     TCP Seg X    |   |  TCP Seg X+1   |
   +---||---------------+    +--------||--------+   +----------||----+
   ----++-----------------------------++-----------------------++-----
                   FPDU #N-1                  FPDU #N
```

Figure 12: Non-Aligned FPDU Freely Placed in TCP Octet Stream

The receiver algorithm for processing TCP segments (e.g., TCP segment #X in Figure 12) carrying non-aligned FPDUs (in order or out of order) includes:

Data Link Layer processing (whole frame) -- typically including a CRC calculation.

1. Network Layer processing (assuming not an IP fragment, the whole Data Link Layer frame contains one IP datagram. IP fragments should be reassembled in a local buffer. This is not a performance optimization goal.)

2. Transport Layer processing -- TCP protocol processing, header and checksum checks.

    a. Classify incoming TCP segment using the 5 tuple (IP SRC, IP DST, TCP SRC Port, TCP DST Port, protocol).

3. Find FPDU message boundaries.

    a. Get MPA state information for the connection.

        If the TCP segment is in order, use the receiver-managed MPA state information to calculate where the previous FPDU message (#N-1) ends in the current TCP segment X. (previously, when the MPA receiver processed the first part of FPDU #N-1, it calculated the number of bytes remaining to complete FPDU #N-1 by using the MPA Length field).

            Get the stored partial CRC for FPDU #N-1.

            Complete CRC calculation for FPDU #N-1 data (first portion of TCP segment #X).

            Check CRC calculation for FPDU #N-1.

            If no FPDU CRC errors, placement is allowed.

            Locate the local buffer for the first portion of FPDU#N-1, CopyData(local buffer of first portion of FPDU #N-1, host buffer address, length).

            Compute host buffer address for second portion of FPDU #N-1.

            CopyData (local buffer of second portion of FPDU #N-1, host buffer address for second portion, length).

            Calculate the octet offset into the TCP segment for the next FPDU #N.

            Start calculation of CRC for available data for FPDU. #N

            Store partial CRC results for FPDU #N.

            Store local buffer address of first portion of FPDU #N.

            No further action is possible on FPDU #N, before it is completely received.

If the TCP segment is out of order, the receiver must buffer the data until at least one complete FPDU is received. Typically, buffering for more than one TCP segment per connection is required. Use the MPA-based Markers to calculate where FPDU boundaries are.

When a complete FPDU is available, a similar procedure to the in-order algorithm above is used. There is additional complexity, though, because when the missing segment arrives, this TCP segment must be run through the CRC engine after the CRC is calculated for the missing segment.

If we assume FPDU Alignment, the following diagram and the algorithm below apply. Note that when using MPA, the receiver is assumed to actively detect presence or loss of FPDU Alignment for every TCP segment received.

```
   +--------------------------+      +--------------------------+
+--|--------------------------+   +--|--------------------------+
|  |       TCP Seg X          |   |  |         TCP Seg X+1      |
+--|--------------------------+   +--|--------------------------+
   +--------------------------+      +--------------------------+
             FPDU #N                          FPDU #N+1
```

Figure 13: Aligned FPDU Placed Immediately after TCP Header

The receiver algorithm for FPDU Aligned frames (in order or out of order) includes:

1) Data Link Layer processing (whole frame) -- typically including a CRC calculation.

2) Network Layer processing (assuming not an IP fragment, the whole Data Link Layer frame contains one IP datagram. IP fragments should be reassembled in a local buffer. This is not a performance optimization goal.)

3) Transport Layer processing -- TCP protocol processing, header and checksum checks.

   a. Classify incoming TCP segment using the 5 tuple (IP SRC, IP DST, TCP SRC Port, TCP DST Port, protocol).

4) Check for Header Alignment (described in detail in Section 6). Assuming Header Alignment for the rest of the algorithm below.

   a. If the header is not aligned, see the algorithm defined in the prior section.

5) If TCP segment is in order or out of order, the MPA header is at the beginning of the current TCP payload. Get the FPDU length from the FPDU header.

6) Calculate CRC over FPDU.

7) Check CRC calculation for FPDU #N.

8) If no FPDU CRC errors, placement is allowed.

9) CopyData(TCP segment #X, host buffer address, length).

10) Loop to #5 until all the FPDUs in the TCP segment are consumed in order to handle FPDU packing.

Implementation note: In both cases, the receiver has to classify the incoming TCP segment and associate it with one of the flows it maintains. In the case of no FPDU Alignment, the receiver is forced to classify incoming traffic before it can calculate the FPDU CRC. In the case of FPDU Alignment, the operations order is left to the implementer.

The FPDU Aligned receiver algorithm is significantly simpler. There is no need to locally buffer portions of FPDUs. Accessing state information is also substantially simplified -- the normal case does not require retrieving information to find out where an FPDU starts and ends or retrieval of a partial CRC before the CRC calculation can commence. This avoids adding internal latencies, having multiple data passes through the CRC machine, or scheduling multiple commands for moving the data to the host buffer.

The aligned FPDU approach is useful for in-order and out-of-order reception. The receiver can use the same mechanisms for data storage in both cases, and only needs to account for when all the TCP segments have arrived to enable Delivery. The Header Alignment, along with the high probability that at least one complete FPDU is found with every TCP segment, allows the receiver to perform data placement for out-of-order TCP segments with no need for intermediate buffering. Essentially, the TCP receive buffer has been eliminated and TCP reassembly is done in place within the ULP Buffer.

In case FPDU Alignment is not found, the receiver should follow the algorithm for non-aligned FPDU reception, which may be slower and less efficient.

B.2.2. FPDU Alignment Effects on TCP Wire Protocol

In an optimized MPA/TCP implementation, TCP exposes its EMSS to MPA. MPA uses the EMSS to calculate its MULPDU, which it then exposes to DDP, its ULP. DDP uses the MULPDU to segment its payload so that each FPDU sent by MPA fits completely into one TCP segment. This has no impact on wire protocol, and exposing this information is already supported on many TCP implementations, including all modern flavors of BSD networking, through the TCP_MAXSEG socket option.

In the common case, the ULP (i.e., DDP over MPA) messages provided to the TCP layer are segmented to MULPDU size. It is assumed that the ULP message size is bounded by MULPDU, such that a single ULP message can be encapsulated in a single TCP segment. Therefore, in the common case, there is no increase in the number of TCP segments emitted. For smaller ULP messages, the sender can also apply packing, i.e., the sender packs as many complete FPDUs as possible into one TCP segment. The requirement to always have a complete FPDU may increase the number of TCP segments emitted. Typically, a ULP message size varies from a few bytes to multiple EMSSs (e.g., 64 Kbytes). In some cases, the ULP may post more than one message at a time for transmission, giving the sender an opportunity for packing. In the case where more than one FPDU is available for transmission and the FPDUs are encapsulated into a TCP segment and there is no room in the TCP segment to include the next complete FPDU, another

TCP segment is sent. In this corner case, some of the TCP segments are not full size. In the worst-case scenario, the ULP may choose an FPDU size that is EMSS/2 +1 and has multiple messages available for transmission. For this poor choice of FPDU size, the average TCP segment size is therefore about 1/2 of the EMSS and the number of TCP segments emitted is approaching 2x of what is possible without the requirement to encapsulate an integer number of complete FPDUs in every TCP segment. This is a dynamic situation that only lasts for the duration where the sender ULP has multiple non-optimal messages for transmission and this causes a minor impact on the wire utilization.

However, it is not expected that requiring FPDU Alignment will have a measurable impact on wire behavior of most applications. Throughput applications with large I/Os are expected to take full advantage of the EMSS. Another class of applications with many small outstanding buffers (as compared to EMSS) is expected to use packing when applicable. Transaction-oriented applications are also optimal.

TCP retransmission is another area that can affect sender behavior. TCP supports retransmission of the exact, originally transmitted segment (see [RFC793], Sections 2.6 and 3.7 (under "Managing the Window") and [RFC1122], Section 4.2.2.15). In the unlikely event that part of the original segment has been received and acknowledged by the Remote Peer (e.g., a re-segmenting middlebox, as documented in Appendix A.4, Re-Segmenting Middleboxes and Non-Optimized MPA/TCP Senders), a better available bandwidth utilization may be possible by retransmitting only the missing octets. If an optimized MPA/TCP retransmits complete FPDUs, there may be some marginal bandwidth loss.

Another area where a change in the TCP segment number may have impact is that of slow start and congestion avoidance. Slow-start exponential increase is measured in segments per second, as the algorithm focuses on the overhead per segment at the source for congestion that eventually results in dropped segments. Slow-start exponential bandwidth growth for optimized MPA/TCP is similar to any TCP implementation. Congestion avoidance allows for a linear growth in available bandwidth when recovering after a packet drop. Similar to the analysis for slow start, optimized MPA/TCP doesn’t change the behavior of the algorithm. Therefore, the average size of the segment versus EMSS is not a major factor in the assessment of the bandwidth growth for a sender. Both slow start and congestion avoidance for an optimized MPA/TCP will behave similarly to any TCP sender and allow an optimized MPA/TCP to enjoy the theoretical performance limits of the algorithms.

In summary, the ULP messages generated at the sender (e.g., the amount of messages grouped for every transmission request) and message size distribution has the most significant impact over the number of TCP segments emitted. The worst-case effect for certain ULPs (with average message size of EMSS/2+1 to EMSS) is bounded by an increase of up to 2x in the number of TCP segments and acknowledges. In reality, the effect is expected to be marginal.

Appendix C. IETF Implementation Interoperability with RDMA Consortium Protocols

This appendix is for information only and is NOT part of the standard.

This appendix covers methods of making MPA implementations interoperate with both IETF and RDMA Consortium versions of the protocols.

The RDMA Consortium created early specifications of the MPA/DDP/RDMA protocols, and some manufacturers created implementations of those protocols before the IETF versions were finalized. These protocols are very similar to the IETF versions making it possible for implementations to be created or modified to support either set of specifications.

For those interested, the RDMA Consortium protocol documents (draft-culley-iwarp-mpa-v1.0.pdf [RDMA-MPA], draft-shah-iwarp-ddp-v1.0.pdf [RDMA-DDP], and draft-recio-iwarp-rdmac-v1.0.pdf [RDMA-RDMAC]) can be obtained at http://www.rdmaconsortium.org/home.

In this section, implementations of MPA/DDP/RDMA that conform to the RDMAC specifications are called RDMAC RNICs. Implementations of MPA/DDP/RDMA that conform to the IETF RFCs are called IETF RNICs.

Without the exchange of MPA Request/Reply Frames, there is no standard mechanism for enabling RDMAC RNICs to interoperate with IETF RNICs. Even if a ULP uses a well-known port to start an IETF RNIC immediately in RDMA mode (i.e., without exchanging the MPA Request/Reply messages), there is no reason to believe an IETF RNIC will interoperate with an RDMAC RNIC because of the differences in the version number in the DDP and RDMAP headers on the wire.

Therefore, the ULP or other supporting entity at the RDMAC RNIC must implement MPA Request/Reply Frames on behalf of the RNIC in order to negotiate the connection parameters. The following section describes the results following the exchange of the MPA Request/Reply Frames before the conversion from streaming to RDMA mode.

C.1. Negotiated Parameters

Three types of RNICs are considered:

Upgraded RDMAC RNIC - an RNIC implementing the RDMAC protocols that has a ULP or other supporting entity that exchanges the MPA Request/Reply Frames in streaming mode before the conversion to RDMA mode.

Non-permissive IETF RNIC - an RNIC implementing the IETF protocols that is not capable of implementing the RDMAC protocols. Such an RNIC can only interoperate with other IETF RNICs.

Permissive IETF RNIC - an RNIC implementing the IETF protocols that is capable of implementing the RDMAC protocols on a per-connection basis.

The Permissive IETF RNIC is recommended for those implementers that want maximum interoperability with other RNIC implementations.

The values used by these three RNIC types for the MPA, DDP, and RDMAP versions as well as MPA Markers and CRC are summarized in Figure 14.

RNIC TYPE	DDP/RDMAP Version	MPA Revision	MPA Markers	MPA CRC
RDMAC	0	0	1	1
IETF Non-permissive	1	1	0 or 1	0 or 1
IETF permissive	1 or 0	1 or 0	0 or 1	0 or 1

Figure 14: Connection Parameters for the RNIC Types for MPA Markers and MPA CRC, enabled=1, disabled=0.

It is assumed there is no mixing of versions allowed between MPA, DDP, and RDMAP. The RNIC either generates the RDMAC protocols on the wire (version is zero) or uses the IETF protocols (version is one).

During the exchange of the MPA Request/Reply Frames, each peer provides its MPA Revision, Marker preference (M: 0=disabled, 1=enabled), and CRC preference. The MPA Revision provided in the MPA Request Frame and the MPA Reply Frame may differ.

From the information in the MPA Request/Reply Frames, each side sets the Version field (V: 0=RDMAC, 1=IETF) of the DDP/RDMAP protocols as well as the state of the Markers for each half connection. Between DDP and RDMAP, no mixing of versions is allowed. Moreover, the DDP and RDMAP version MUST be identical in the two directions. The RNIC either generates the RDMAC protocols on the wire (version is zero) or uses the IETF protocols (version is one).

In the following sections, the figures do not discuss CRC negotiation because there is no interoperability issue for CRCs. Since the RDMAC RNIC will always request CRC use, then, according to the IETF MPA specification, both peers MUST generate and check CRCs.

C.2. RDMAC RNIC and Non-Permissive IETF RNIC

Figure 15 shows that a Non-permissive IETF RNIC cannot interoperate with an RDMAC RNIC, despite the fact that both peers exchange MPA Request/Reply Frames. For a Non-permissive IETF RNIC, the MPA negotiation has no effect on the DDP/RDMAP version and it is unable to interoperate with the RDMAC RNIC.

The rows in the figure show the state of the Marker field in the MPA Request Frame sent by the MPA Initiator. The columns show the state of the Marker field in the MPA Reply Frame sent by the MPA Responder. Each type of RNIC is shown as an Initiator and a Responder. The connection results are shown in the lower right corner, at the intersection of the different RNIC types, where V=0 is the RDMAC DDP/RDMAP version, V=1 is the IETF DDP/RDMAC version, M=0 means MPA Markers are disabled, and M=1 means MPA Markers are enabled. The negotiated Marker state is shown as X/Y, for the receive direction of the Initiator/Responder.

MPA CONNECT MODE			MPA Responder		
	RNIC TYPE		RDMAC	IETF Non-permissive	
		MARKER	M=1	M=0	M=1
MPA Initiator	RDMAC	M=1	V=0 M=1/1	close	close
	IETF Non-perms.	M=0	close	V=1 M=0/0	V=1 M=0/1
		M=1	close	V=1 M=1/0	V=1 M=1/1

Figure 15: MPA Negotiation between an RDMAC RNIC and a Non-Permissive IETF RNIC

C.2.1. RDMAC RNIC Initiator

If the RDMAC RNIC is the MPA Initiator, its ULP sends an MPA Request Frame with Rev field set to zero and the M and C bits set to one. Because the Non-permissive IETF RNIC cannot dynamically downgrade the version number it uses for DDP and RDMAP, it would send an MPA Reply Frame with the Rev field equal to one and then gracefully close the connection.

C.2.2. Non-Permissive IETF RNIC Initiator

If the Non-permissive IETF RNIC is the MPA Initiator, it sends an MPA Request Frame with Rev field equal to one. The ULP or supporting entity for the RDMAC RNIC responds with an MPA Reply Frame that has the Rev field equal to zero and the M bit set to one. The Non-permissive IETF RNIC will gracefully close the connection after it reads the incompatible Rev field in the MPA Reply Frame.

C.2.3. RDMAC RNIC and Permissive IETF RNIC

Figure 16 shows that a Permissive IETF RNIC can interoperate with an RDMAC RNIC regardless of its Marker preference. The figure uses the same format as shown with the Non-permissive IETF RNIC.

MPA CONNECT MODE			MPA Responder		
	RNIC TYPE		RDMAC	IETF Permissive	
		MARKER	M=1	M=0	M=1
MPA Initiator	RDMAC	M=1	V=0 M=1/1	N/A	V=0 M=1/1
	IETF Permissive	M=0	V=0 M=1/1	V=1 M=0/0	V=1 M=0/1
		M=1	V=0 M=1/1	V=1 M=1/0	V=1 M=1/1

Figure 16: MPA Negotiation between an RDMAC RNIC and a Permissive IETF RNIC

A truly Permissive IETF RNIC will recognize an RDMAC RNIC from the Rev field of the MPA Req/Rep Frames and then adjust its receive Marker state and DDP/RDMAP version to accommodate the RDMAC RNIC. As a result, as an MPA Responder, the Permissive IETF RNIC will never return an MPA Reply Frame with the M bit set to zero. This case is shown as a not applicable (N/A) in Figure 16.

C.2.4. RDMAC RNIC Initiator

When the RDMAC RNIC is the MPA Initiator, its ULP or other supporting entity prepares an MPA Request message and sets the revision to zero and the M bit and C bit to one.

The Permissive IETF Responder receives the MPA Request message and checks the revision field. Since it is capable of generating RDMAC DDP/RDMAP headers, it sends an MPA Reply message with revision set to zero and the M and C bits set to one. The Responder must inform its ULP that it is generating version zero DDP/RDMAP messages.

C.2.5 Permissive IETF RNIC Initiator

If the Permissive IETF RNIC is the MPA Initiator, it prepares the MPA Request Frame setting the Rev field to one. Regardless of the value of the M bit in the MPA Request Frame, the ULP or other supporting entity for the RDMAC RNIC will create an MPA Reply Frame with Rev equal to zero and the M bit set to one.

When the Initiator reads the Rev field of the MPA Reply Frame and finds that its peer is an RDMAC RNIC, it must inform its ULP that it should generate version zero DDP/RDMAP messages and enable MPA Markers and CRC.

C.3. Non-Permissive IETF RNIC and Permissive IETF RNIC

For completeness, Figure 17 below shows the results of MPA negotiation between a Non-permissive IETF RNIC and a Permissive IETF RNIC. The important point from this figure is that an IETF RNIC cannot detect whether its peer is a Permissive or Non-permissive RNIC.

MPA CONNECT MODE			MPA Responder			
	RNIC TYPE		IETF Non-permissive		IETF Permissive	
		MARKER	M=0	M=1	M=0	M=1
MPA Initiator	IETF Non-perms.	M=0	V=1 M=0/0	V=1 M=0/1	V=1 M=0/0	V=1 M=0/1
		M=1	V=1 M=1/0	V=1 M=1/1	V=1 M=1/0	V=1 M=1/1
	IETF Permissive	M=0	V=1 M=0/0	V=1 M=0/1	V=1 M=0/0	V=1 M=0/1
		M=1	V=1 M=1/0	V=1 M=1/1	V=1 M=1/0	V=1 M=1/1

Figure 17: MPA negotiation between a Non-permissive IETF RNIC and a Permissive IETF RNIC.

Normative References

[iSCSI] Satran, J., Meth, K., Sapuntzakis, C., Chadalapaka, M., and E. Zeidner, "Internet Small Computer Systems Interface (iSCSI)", RFC 3720, April 2004.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2401] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[RFC3723] Aboba, B., Tseng, J., Walker, J., Rangan, V., and F. Travostino, "Securing Block Storage Protocols over IP", RFC 3723, April 2004.

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RDMASEC] Pinkerton, J. and E. Deleganes, "Direct Data Placement Protocol (DDP) / Remote Direct Memory Access Protocol (RDMAP) Security", RFC 5042, October 2007.

Informative References

[APPL] Bestler, C. and L. Coene, "Applicability of Remote Direct Memory Access Protocol (RDMA) and Direct Data Placement (DDP)", RFC 5045, October 2007.

[CRCTCP] Stone J., Partridge, C., "When the CRC and TCP checksum disagree", ACM Sigcomm, Sept. 2000.

[DAT-API] DAT Collaborative, "kDAPL (Kernel Direct Access Programming Library) and uDAPL (User Direct Access Programming Library)", Http://www.datcollaborative.org.

[DDP] Shah, H., Pinkerton, J., Recio, R., and P. Culley, "Direct Data Placement over Reliable Transports", RFC 5041, October 2007.

[iSER] Ko, M., Chadalapaka, M., Hufferd, J., Elzur, U., Shah, H., and P. Thaler, "Internet Small Computer System Interface (iSCSI) Extensions for Remote Direct Memory Access (RDMA)" RFC 5046, October 2007.

[IT-API] The Open Group, "Interconnect Transport API (IT-API)" Version 2.1, http://www.opengroup.org.

[NFSv4CHAN] Williams, N., "On the Use of Channel Bindings to Secure Channels", Work in Progress, June 2006.

[RDMA-DDP] "Direct Data Placement over Reliable Transports (Version 1.0)", RDMA Consortium, October 2002, <http://www.rdmaconsortium.org/home/draft-shah-iwarp-ddp-v1.0.pdf>.

[RDMA-MPA] "Marker PDU Aligned Framing for TCP Specification (Version 1.0)", RDMA Consortium, October 2002, <http://www.rdmaconsortium.org/home/draft-culley-iwarp-mpa-v1.0.pdf>.

[RDMA-RDMAC] "An RDMA Protocol Specification (Version 1.0)", RDMA Consortium, October 2002, <http://www.rdmaconsortium.org/home/draft-recio-iwarp-rdmac-v1.0.pdf>.

[RDMAP] Recio, R., Culley, P., Garcia, D., Hilland, J., and B. Metzler, "A Remote Direct Memory Access Protocol Specification", RFC 5040, October 2007.

[RFC792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

[RFC896] Nagle, J., "Congestion control in IP/TCP internetworks", RFC 896, January 1984.

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, September 2007.

[RFC4296] Bailey, S. and T. Talpey, "The Architecture of Direct Data Placement (DDP) and Remote Direct Memory Access (RDMA) on Internet Protocols", RFC 4296, December 2005.

[RFC4297] Romanow, A., Mogul, J., Talpey, T., and S. Bailey, "Remote Direct Memory Access (RDMA) over IP Problem Statement", RFC 4297, December 2005.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[VERBS-RMDA] "RDMA Protocol Verbs Specification", RDMA Consortium standard, April 2003, <http://www.rdmaconsortium.org/home/draft-hilland-iwarp-verbs-v1.0-RDMAC.pdf>.

Contributors

Dwight Barron
Hewlett-Packard Company
20555 SH 249
Houston, TX 77070-2698 USA
Phone: 281-514-2769
EMail: dwight.barron@hp.com

Jeff Chase
Department of Computer Science
Duke University
Durham, NC 27708-0129 USA
Phone: +1 919 660 6559
EMail: chase@cs.duke.edu

Ted Compton
EMC Corporation
Research Triangle Park, NC 27709 USA
Phone: 919-248-6075
EMail: compton_ted@emc.com

Dave Garcia
24100 Hutchinson Rd.
Los Gatos, CA 95033
Phone: 831 247 4464
EMail: Dave.Garcia@StanfordAlumni.org

Hari Ghadia
Gen10 Technology, Inc.
1501 W Shady Grove Road
Grand Prairie, TX 75050
Phone: (972) 301 3630
EMail: hghadia@gen10technology.com

Howard C. Herbert
Intel Corporation
MS CH7-404
5000 West Chandler Blvd.
Chandler, AZ 85226
Phone: 480-554-3116
EMail: howard.c.herbert@intel.com

Jeff Hilland
Hewlett-Packard Company
20555 SH 249
Houston, TX 77070-2698 USA
Phone: 281-514-9489
EMail: jeff.hilland@hp.com

Mike Ko
IBM
650 Harry Rd.
San Jose, CA 95120
Phone: (408) 927-2085
EMail: mako@us.ibm.com

Mike Krause
Hewlett-Packard Corporation, 43LN
19410 Homestead Road
Cupertino, CA 95014 USA
Phone: +1 (408) 447-3191
EMail: krause@cup.hp.com

Dave Minturn
Intel Corporation
MS JF1-210
5200 North East Elam Young Parkway
Hillsboro, Oregon 97124
Phone: 503-712-4106
EMail: dave.b.minturn@intel.com

Jim Pinkerton
Microsoft, Inc.
One Microsoft Way
Redmond, WA 98052 USA
EMail: jpink@microsoft.com

Hemal Shah
Broadcom Corporation
5300 California Avenue
Irvine, CA 92617 USA
Phone: +1 (949) 926-6941
EMail: hemal@broadcom.com

Allyn Romanow
Cisco Systems
170 W Tasman Drive
San Jose, CA 95134 USA
Phone: +1 408 525 8836
EMail: allyn@cisco.com

Tom Talpey
Network Appliance
1601 Trapelo Road #16
Waltham, MA 02451 USA
Phone: +1 (781) 768-5329
EMail: thomas.talpey@netapp.com

Patricia Thaler
Broadcom
16215 Alton Parkway
Irvine, CA 92618
Phone: 916 570 2707
EMail: pthaler@broadcom.com

Jim Wendt
Hewlett Packard Corporation
8000 Foothills Boulevard MS 5668
Roseville, CA 95747-5668 USA
Phone: +1 916 785 5198
EMail: jim_wendt@hp.com

Jim Williams
Emulex Corporation
580 Main Street
Bolton, MA 01740 USA
Phone: +1 978 779 7224
EMail: jim.williams@emulex.com

Authors' Addresses

Paul R. Culley
Hewlett-Packard Company
20555 SH 249
Houston, TX 77070-2698 USA
Phone: 281-514-5543
EMail: paul.culley@hp.com

Uri Elzur
5300 California Avenue
Irvine, CA 92617, USA
Phone: 949.926.6432
EMail: uri@broadcom.com

Renato J Recio
IBM
Internal Zip 9043
11400 Burnett Road
Austin, Texas 78759
Phone: 512-838-3685
EMail: recio@us.ibm.com

Stephen Bailey
Sandburst Corporation
600 Federal Street
Andover, MA 01810 USA
Phone: +1 978 689 1614
EMail: steph@sandburst.com

John Carrier
Cray Inc.
411 First Avenue S, Suite 600
Seattle, WA 98104-2860
Phone: 206-701-2090
EMail: carrier@cray.com

Full Copyright Statement

Intellectual Property